Enrique Gómez Jiménez

Desarrollo de Software con NetBeans 7.1

¡Programe para escritorio, Web y dispositivos móviles!

Apoyo en la

Alfaomega

Desarrollo de Software con NetBeans 7.1

¡Programe para escritorio, Web y dispositivos móviles!

Enrique Gómez Jiménez

Alfaomega

Buenos Aires • Bogotá • México DF • Santiago de Chile

Edición
Alejandro Herrera
Corrección
Alejandro Cruz Ulloa
Gerente editorial
Marcelo Grillo

Datos catalográficos
Gómez Jiménez Enrique
Desarrollo de Software con Netbeans 7.1
¡Programe para escritorio, Web y dispositivos móviles!
Primera Edición
Alfaomega Grupo Editor, S.A. de C.V. México.
ISBN: 978-607-707-522-6
Formato: 17 x 23 cm Páginas 472

Desarrollo de Software con Netbeans 7.1
¡Programe para escritorio, Web y dispositivos móviles!

Enrique Gómez Jiménez

Primera Edición por Alfaomega Grupo Editor, S.A. de C.V. México, agosto 2012

Pitágoras 1139, Col. Del Valle, C.P. 03100, México, D.F.
Miembro de la Cámara Nacional de la Industria Editorial Mexicana
Registro No. 2317

Internet: http://www.alfaomega.com.mx
E-mail: atencionalcliente@alfaomega.com.mx

ISBN: **978-607-707-522-6**

Empresas del grupo

México: Alfaomega Grupo Editor, S.A. de C.V. – Pitágoras 1139, Col. Del Valle, México, D.F. – C.P. 03100.
Tel.: (52-55) 5575-5022 – Fax: (52-55) 5575-2420 / 2490. Sin costo: 01-800-020-4396
E-mail: atencionalcliente@alfaomega.com.mx

Colombia: Alfaomega Colombiana S.A. – Carrera 15 No. 64 A 29, Bogotá, Colombia,
Tel.: (57-1) 2100122 – Fax: (57-1) 6068648 – E-mail: cliente@alfaomega.com.co

Chile: Alfaomega Grupo Editor, S.A. – Dr. La Sierra 1437, Providencia, Santiago, Chile
Tel.: (56-2) 235-4248 – Fax: (56-2) 235-5786 – E-mail: agechile@alfaomega.cl

Argentina: Alfaomega Grupo Editor Argentino, S.A. – Paraguay 1307 P.B. Of. 11, C.P. 1057, Buenos Aires, Argentina, – Tel./Fax: (54-11) 4811-0887 y 4811 7183 – E-mail: ventas@alfaomegaeditor.com.ar

A mi esposa que ha soportado la espera en las noches, frente al televisor, mientras yo escribo este libro.

A mi hija e hijos a los cuales no les he dado el tiempo suficiente para verlos crecer.

A mi madre que ahora volvió a su negocio y se siente feliz de atender a esos viejos transportistas.

.

Enrique Gómez Jiménez

Agradecimientos

Deseo agradecer a todas las personas que me han apoyado durante mi carrera docente.

A mis estudiantes, de quienes he aprendido mucho.

A mis compañeros profesores, que me han transmitido parte del gran conocimiento que poseen. Especialmente a Denicie Moreno Azofeifa quien me ha ayudado mucho en varios aspectos relacionados con HTML y CSS.

A mi amigo y editor Alejandro Herrera, por su apoyo y dedicación incondicionales, en tiempos difíciles, para el trabajo editorial de este libro, mi segunda oportunidad.

A la Editorial Alfaomega que sin tanta burocracia hace posible que muchos podamos publicar y divulgar nuestro poco pero valioso conocimiento.

Gracias.

Enrique Gómez Jiménez

Mensaje del editor

Los conocimientos son esenciales en el desempeño profesional, sin ellos es imposible lograr las habilidades para competir laboralmente. La universidad o las instituciones de formación para el trabajo ofrecen la oportunidad de adquirir conocimientos que serán aprovechados más adelante en beneficio propio y de la sociedad; el avance de la ciencia y de la técnica hace necesario actualizar continuamente esos conocimientos. Cuando se toma la decisión de embarcarse en una vida profesional, se adquiere un compromiso de por vida: mantenerse al día en los conocimientos del área u oficio que se ha decidido desempeñar.

Alfaomega tiene por misión ofrecer a sus lectores actualizados dentro de lineamientos pedagógicos que faciliten su utilización y permitan desarrollar las competencias requeridas por una profesión determinada. Alfaomega espera ser su compañera profesional en este viaje de por vida por el mundo del conocimiento.

Alfaomega hace uso de los medios impresos tradicionales en combinación con las tecnologías de la información y las comunicaciones (TIC) para facilitar el aprendizaje. Libros como éste tienen su complemento en una página web, en donde el alumno y su profesor encontrarán materiales adicionales.

Esta obra contiene numerosos gráficos, cuadros y otros recursos para despertar el interés del lector, y facilitarle la comprensión y apropiación del conocimiento. Cada capítulo se desarrolla con argumentos presentados en forma sencilla y estructurada claramente hacia los objetivos y metas propuestas.

Los libros de Alfaomega están diseñados para utilizarse dentro de los procesos de enseñanza aprendizaje, y se pueden usar como textos para diversos cursos o como apoyo para reforzar el desarrollo profesional.

Alfaomega espera contribuir así a la formación y desarrollo de profesionales exitosos para beneficio de la sociedad.

Acerca del autor

MSc. Enrique Gómez Jiménez

Es Ingeniero en sistemas de información con Maestría en Gestión de la Innovación Tecnológica por la Escuela de Informática de la Universidad Nacional de Costa Rica (www.una.ac.cr)

Es autor de numerosos artículos y manuales sobre desarrollo de software con Visual Basic .NET, estructuras de datos en JAVA, diseño Web con Expression, Web y sistemas operativos, entre otros.

Entre sus libros destaca Aplicaciones con Visual Basic .Net ¡Programe para Escritorio, Web y Dispositivos móviles!, publicado por esta editorial.

Se desempeña como encargado de la cátedra de desarrollo de software de la Universidad Estatal a Distancia, UNED (www.uned.ac.cr), así como profesor de Informática en la Universidad Nacional (www.una.ac.cr) entre otras instituciones de educación superior.

Es consultor externo y desarrollador independiente de software. Ha ocupado los puestos de Gerente de DTIC, analista programador de aplicaciones, así como miembro de equipos de proyectos de software a nivel nacional e internacional

Contenido

Capítulo 4

Capítulo 5

Capítulo 6

Introducción

Java representa una opción poderosa para el desarrollo de aplicaciones informáticas dirigidas al escritorio, a la web o a dispositivos móviles como teléfonos celulares, Tablet, reproductores de audio y video, entre otros. Herramientas como NetBeans o Eclipse refuerzan, con su IDE, la gran gama de posibilidades para el desarrollo consistente y de gran escalabilidad de Java.

Muchas de las funcionalidades, ampliamente probadas de Java, se ven implementadas en estas herramientas de desarrollo las cuales ofrecen un IDE con muchos controles y características de programación que facilitan el desarrollo de aplicaciones. Y sobre todo se utiliza una filosofía ampliamente aceptada por instituciones públicas y privadas, grandes corporaciones y hasta esfuerzos individuales: el software libre.

Las necesidades de desarrollo de aplicaciones informáticas se orientan hacia la web y dispositivos móviles, principalmente. La tendencia del desarrollo de aplicaciones de escritorio (desktop) son cada vez menores, quizá influenciada por la globalización de los negocios y el uso intensivo de las redes como internet y corporativas. En la parte de Internet muchas empresas han desarrollado framework que permiten el desarrollo rápido de aplicaciones (RIA) tales como Spring web MVC, ICEFace, Struts, Java ServerFaces, entre otros.

Lastimosamente, muchos de ellos no ofrecen aun un diseñador gráfico para la versión 7.1 (hasta la fecha de impresión de esta obra) como lo fue el Visual Web Aplication Design que se utilizaba en NetBeans 5-6 (vea la facilidad de diseño que se llevaba a cabo con estas versiones en http://netbeans.org/community/magazine/html/03/visualwebdev/).

ICEFaces fue, en las pruebas, el que mejores prestaciones brindaba a los diseños gráficos de la web, pero sin la interface amigable del Visual Web Aplication Design.

En el desarrollo de aplicaciones para dispositivos móviles, Oracle, propietaria de NetBeans, ofrece a través de esta herramienta muchos controles y funcionalidades que permiten desde el diseño del flujo de ejecución de las diferentes pantallas de la aplicación móvil hasta componentes gráficos reutilizables.

La programación básica para dispositivos móviles se realiza con emuladores que SUN ya había trabajado cuando NetBeans era de su propiedad y se han ido incluyendo otros por parte de la compañía así como plugins de terceros.

La posibilidad de incluir plugins de empresas constructoras de móviles es otra de las grandes facilidades, por ende, desarrollar para BlackBerry, Nokia, Sony Ericsson o cualquier otro fabricante se torna fácil dado que ofrecen también el framework necesario para desarrollar aplicaciones para estos productos específicos, utilizando NetBeans.

Podrá iniciarse en este fabuloso mundo de la programación en Java, utilizando uno de los IDES más importantes en el mundo de la programación de software libre: NetBeans; podrá iniciar su carrera como desarrollador independiente de soluciones informáticas, utilizando software libre como NetBeans y MySQL. También podrá colaborar con la economía de la empresa para la que trabaja creando soluciones importantes de software utilizando estas herramientas sin invertir un costo exorbitante solo por la adquisición de tecnologías propietarias.

Descripción del contenido

Con este libro podrá desarrollar aplicaciones orientadas al escritorio, para la web y para dispositivos móviles. También podrá liberar su espíritu creativo e innovador y crear un juego utilizando el diseñador visual para juegos de NetBeans. Claro que ésta última parte tendrá que desarrollarla completa porque solo se hace una introducción a su uso, suficiente para echar a andar esa gran capacidad que tiene dentro y que debe sacar a flote. Podrá también practicar la funcionalidad básica de la programación orientada a objetos, creando clases, interfaces, atributos, métodos, entre otros elementos y reforzar esa importante fase del desarrollo de aplicaciones informáticas. En la parte web tendrá una idea de los diferentes framework que hay en el mercado y como unos y otros se asemejan en la forma de utilizar las diferentes funcionalidades de Java, incluyendo las JSP (JavaServer Page), JSF (JavaServer Face), AJAX, entre otras.

Más allá de un certificador de conocimientos, este libro constituye un fundamento requerido para profundizar posteriormente en las áreas temáticas del desarrollo que se tratan, dada la abundancia de requerimientos para su aprendizaje y que, para desarrollarlo académicamente es tema suficiente para escribir un libro detallado en forma individual.

1. NetBeans 7.1 IDE

Trata las generalidades de la herramienta, basando en preparar la misma para su uso en el curso. Incluye como descargar el producto, el entorno del IDE y los tipos de proyectos que se pueden crear con NetBeans, versión 7.

2. Fundamentos de programación en NetBeans 7.1

Desarrolla los fundamentos de la programación en NetBeans 7, considerando conceptos tales como constantes, variables, tipos de datos, operadores y estrucuras de control, lo cual es eminentemente necesario para iniciar a programar con NetBeans.

3. Programación orientada a objetos con NetBeans 7.1

Introduce al lector en la programación orientada a objetos en NetBeans 7. Se inicia considerando los paradigmas de programación (incluyendo la programación orientada a objetos) Abstraccionismo de clases, interfaces y polimorfismo son tratados, considerando la importancia que revisten dentro de la programación orientada a objetos.

4. Aplicaciones de escritorio con NetBeans 7.1

Trata sobre las aplicaciones de escritorio en NetBeans 7. Se desarrolla una aplicación de escritorio (desktop) para demostrar el uso de los componentes más comunes en este tipo de aplicaciones. Asimismo, se desarrolla un paquete (package) para ser utilizado en dicha aplicación.

5. Gestión de base de datos MySQL con NetBeans 7.1

Aborda la gestión de bases de datos en MySQL NetBeans 7. Se instala el gestor de archivos (mySQL) y el IDE que la soporta (dbforgemysqlfree) Se explica la teoría que soporta Java con respecto a la gestión de bases de datos (arquitectura JDBC y objetos de conectividad) Finalmente, se desarrollan ejemplos que demuestran la gestión de datos con NetBeans y mySQL.

6. Fundamentos de programación Web con NetBeans 7.1

Fundamenta algunos conceptos de la programación web con Java. Trata el tema de Tomcat y Glassfish como servidores utilizados en aplicaciones web y algunas tecnologías que se basan en lenguajes de marcas para el desarrollo de este tipo de aplicaciones.

7. Patrones de diseño en ingeniería Web

Trata el tema de los patrones de diseño en el desarrollo web. Introduce al lector en los métodos utilizados para el desarrollo web, patrones de diseño J2EE y el Modelo Vista CONTROLADOR.

8. JavaServer Pages en NetBeans 7.1

Trata las JSP (Java Server Page) que constituye la tecnología de SUN Microsystem que permite la creación de código Java del lado del servidor, en forma de scripts, y que se embebe dentro de una página HTML. JSP separa el código Java del HTML, lo que equivale a Servlets ejecutados del lado del servidor.

9. Servicios Web en NetBeans 7.1

Introduce al lector en la creación de servicios web (web services) con NetBeans. Se consideran tecnologías como REST, JAX-WS y WSDL.

10. Gestión de bases de datos en aplicaciones Web con NetBeans 7.1

Desarrolla la gestión de bases de datos en aplicaciones web en NetBeans. Describe como crear y usar un servicio web en una aplicación web y cómo dar mantenimiento a una base de datos mediante un sitio web Java.

11. Spring web MVC

Trata el framework Spring web MVC con NetBeans 7.1 para el desarrollo de una aplicación web sencilla. El objetivo de este capítulo es que el lector comprenda la forma básica de desarrollar utilizando este framework de java.

12. JavaServer Faces / ICEFaces en NetBeans 7.1

Fundamenta el uso básico de dos framework muy utilizados en el ambiente de desarrollo java: JavaServer Faces / ICEFaces. Se describe su funcionamiento principal y se desarrollan dos ejemplos (uno para cada framework) demostrando la funcionalidad básica de los mismos.

13. Fundamentos de programación para móviles en NetBeans 7.1

Introduce al lector en el desarrollo de aplicaciones móviles con NetBeans. Se brindan los conceptos básicos y se crean aplicaciones de ejemplo para comprender como desarrollar este tipo de soluciones con NetBeans.

Antes de comenzar a leer

La estructura de cada capítulo incluye:

- **Listado de capacidades y competencias.** Se describen brevemente los aprendizajes esperados del participante al término del estudio del tema y que integra conocimientos, habilidades, valores y actitudes como un conjunto de comportamientos sociales, afectivos y habilidades cognitivas, psicológicas sensoriales y motoras, que permiten llevar a cabo adecuadamente un papel, un desempeño, una actividad o una tarea.
- **Evidencias de las capacidades desarrolladas.** Describen brevemente lo que se espera que el lector demuestre al término del estudio y que consiste en demostrar que puede llevar a cabo adecuadamente un papel, un desempeño, una actividad o una tarea.
- **Exposición del tema.** Corresponde al desarrollo del capítulo. Cuando se ha considerado pertinente, se han incluido recuadros con ejemplos, casos breves y figuras con esquemas explicativos. También es importante señalar que los conceptos clave están destacados en el texto y sus definiciones están en el glosario al final del libro.
- **Actividades**. Para asegurar un aprendizaje efectivo, resulta imprescindible buscar la interacción del lector con los temas. Es por ello que se privilegian las preguntas abiertas (sin respuestas verdaderas o falsas) y las actividades de discusión.
- **Bibliografía.** Al final de cada capítulo se ha colocado una gran cantidad de bibliografía para estimular y facilitar la profundización en el tema.

Acceso al material complementario

Para tener acceso al material complementario de:

Desarrollo de Software con NetBeans 7.1

¡Programe para escritorio, Web y dispositivos móviles!

Es necesario:

1. Ir a la página: http://virtual.alfaomega.com.mx/
2. Registrarse como usuario y guardar en un lugar seguro su nombre de usuario y clave de acceso para emplearlos para futuros ingresos.
3. Ingrese con su usuario a la sección de libros.
4. Busque y seleccione la imagen correspondiente a este libro para descargar su material complementario.

NOTAS

Se recomienda respaldar los archivos descargados de la página web en un soporte físico, del que el lector pueda recuperarlos posteriormente.

Las descargas no generan ninguna responsabilidad para el autor o la editorial.

NetBeans 7.1 IDE

1

Reflexione y responda las siguientes preguntas

¿Por qué es importante el uso de software libre?

¿Vale el esfuerzo de apoyar a los creadores de software libre desarrollando aplicaciones con esas creaciones?

¿Es seguro el uso de software libre en aplicaciones de misión crítica que considere transacciones delicadas y procesamiento concurrente como las que se realizan en una entidad bancaria?

¿Es tan bueno el software libre como el software propietario?

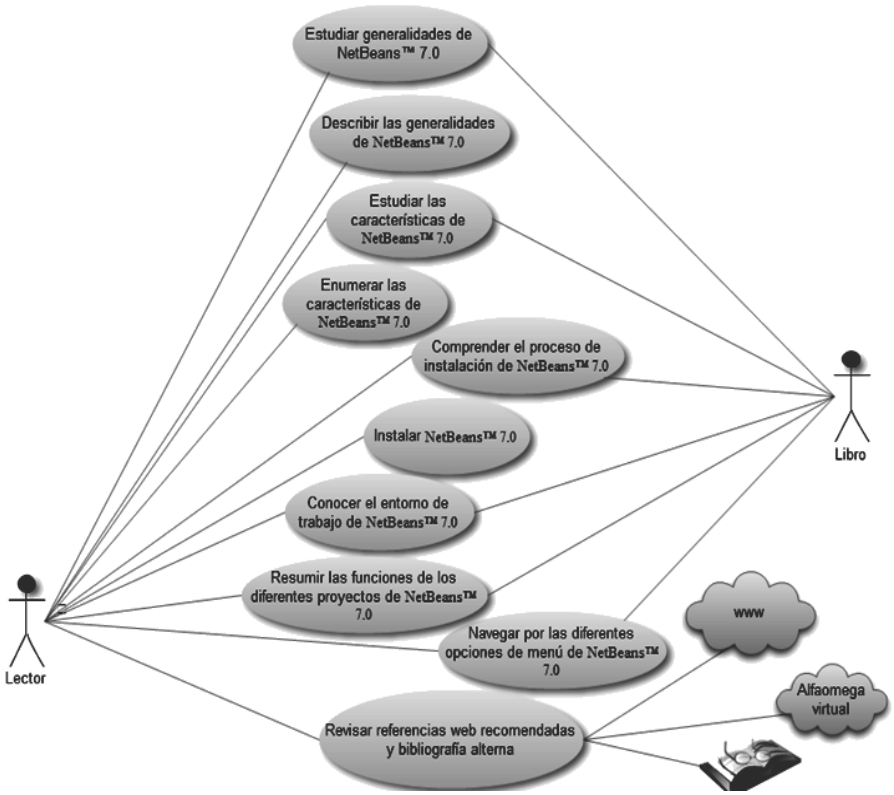
Contenido

EXPECTATIVA

Conocer la importancia de NetBeans 7.1 como lenguaje de programación de uso libre, sus características, la funcionalidad que implementa en su IDE, y los tipos de proyectos que se pueden desarrollar.

Después de estudiar este capítulo, el lector será capaz de

- Describir las generalidades de NetBeans 7.1 como lenguaje de desarrollo de aplicaciones informáticas de uso libre.
- Enumerar las nuevas características de NetBeans 7.1 con respecto a las versiones antecesoras.
- Comprender el proceso de instalación de NetBeans 7.1 para replicarlo en su propio entorno.
- Conocer el entorno de trabajo de NetBeans 7.1 y las funcionalidades que aporta.
- Entender la funcionalidad de los diversos tipos de proyectos NetBeans 7.1, así como su aplicabilidad en el desarrollo de aplicaciones informáticas.



INTRODUCCIÓN

NetBeans es un entorno modular para el desarrollo de aplicaciones informáticas, escrito en lenguaje de programación Java. Este IDE (*Interfaces Development Environment*) está desarrollado para la construcción de sistemas informáticos de diversa índole: aplicaciones de escritorio, para la web o para dispositivos móviles.

NetBeans está en el mercado de los IDE's que soportan la plataforma Java para aligerar el desarrollo de software mediante el suministro de componentes y librerías reutilizables. La lógica funcional es dotar al desarrollador de software de herramientas gráficas que le ayuden a "dibujar" o "pegar" componentes y reutilizar librerías. NetBeans se convierte, entonces, conjuntamente con otro IDE muy conocido, Eclipse, en el facilitador funcional de los desarrolladores de software.

NetBeans es un proyecto de código abierto que cuenta con una gran cantidad de usuarios alrededor del mundo, la cual se encuentra en constante crecimiento. Sun MicroSystem fue la fundadora del proyecto en el año 2000 y lo sigue patrocinando hasta el día de hoy. Los productos generados en el proyecto son NetBeans IDE y NetBeans Platform.

NetBeans IDE constituye un entorno de desarrollo donde los desarrolladores pueden escribir, compilar, depurar y ejecutar programas. Cabe mencionar que NetBeans está desarrollado en Java pero es extensible a cualquier otro lenguaje de programación. Asimismo, que NetBeans IDE es un producto de libre distribución y uso.

NetBeans Platform es una plataforma que permite la integración de módulos en grandes y complejas aplicaciones de escritorio. Empresas desarrolladoras externas a SUN facilitan aplicaciones que se pueden integrar en el IDE de desarrollo de NetBeans generando nuevas funcionalidades. También consta de un código fuente disponible y reutilizable según licenciamiento de la CCDL (Common Development and Distribution License (CDDL) Versión 1.0.

Es importante saber también que existe una organización que orienta el accionar de NetBeans, la cual se denomina NetBeans.org y constituye un portal de código abierto dedicado a construir y mejorar el IDE. Se supone que hay más de 160 países alrededor del mundo que trabajan coordinadamente sobre este objetivo. Cualquier persona puede formar parte de esta comunidad de desarrolladores registrándose en el sitio y participando de las actividades que se programan.

LO NUEVO EN NETBEANS 7.1

La versión 7.1 de NetBeans presenta algunas mejoras importantes con respecto a las versiones anteriores. Esta herramienta permite a los desarrolladores de software crear, de forma rápida, aplicaciones Java, PHP, JavaScript y AJAX, Groovy y Grails. Esta apoyada por una importante comunidad de colaboradores técnicos, documentación y plugins de terceros de amplia gama.

La herramienta introduce soporte para JavaFX 2.0, mejoras en el GUI Builder de Swing, CSS3, utilidades para depuración visual de interfaces de usuario de Swing, nuevo depurador PHP y algunas mjoras en JavaEE y Maven.

A continuación se citan algunas de estas mejoras:

- JDK 7 que representa nuevas interfaces, clases, enumeradores y métodos. Se pueden citar Java.util.Objects, que proporciona métodos estáticos; algunos de estos métodos son:
    - Float.compare
    - Short.compare,
    - Collections.emptyaIterator()
    - Collections.empty.Enumeration()
- incluyen funcionalidad all Calendar, entre otras muchas características.
- Soporte renovado para WebLogic Application Server y GlassFish 3.1. WebLogic Server proporciona una serie de componentes estándares y modulares que se utilizan para el desarrollo de aplicaciones Java EE. Con WebLogic Server se pueden especificar y modelar el comportamiento de una aplicación Java EE en forma automática, sin necesidad de programación.

En febrero 2011 salió a la luz la versión 3.1 de GlassFish, por parte de su creador, Oracle. El propósito de este software es crear clústeres de alto desempeño y administración centralizada para aplicaciones Java EE 6. Permite también reconfiguración dinámica, seguridad sincronizada, monitoreo de servicios, balanceadores de carga, entre otras características.

- Mejoras en la gestión de bases de datos Oracle. Oracle es quizá el DBMS (Data Base Management System) mejor posicionado del mercado. Las poderosas utilidades que posee para el manejo de bases de datos, tanto pequeñas como de grandes cantidades de información, se unen a poderosos asistentes de creación automática de interfaces de mantenimiento de datos.
- Soporte para HTML5. No se considera una versión nueva de HTML, sino más bien una agrupación de varias especificaciones mejoradas de tecnologías orientadas al desarrollo web, tales como HTML 4, XHTML 1 y DOM.
- Soporte para Maven 3. Maven es una herramienta utilizada para la gestión y construcción de proyectos Java. Fue creada por Jason van Zyl, en 2002.

- Soporte mejorado para servicios REST (Representational State Transfer), persistencia en Java y validación Bean. Los servicios REST se emplean a la hora de crear servicios web. Utiliza XML y HTTP para la gestión del envío de información con operaciones tales como GET, POST, PUT y DELETE (propias de HTTP). SOAP es la otra filosofía para la creación de servicios web. Sin embargo, REST es más simple que SOAP, el cual ha sido señalado como obsoleto al igual que WSDL.
- Refactorización para PHP. La refactorización es un proceso de reescritura de código de algún lenguaje a otro, ya sea para dotarlo de una mayor facilidad de comprensión o para optimizarlo. En este caso, NetBeans tiene la fortaleza de refactorizar código PHP y optimizarlo al máximo posible.
- Mejoramiento en la detección de cambios externos.
- NetBeans da y mejora el soporte a muchas de las tecnologías modernas, entre las cuales se encuentran las que se orientan a servidores, a web, dispositivos móviles, servicios web, entre otras.

En la Figura 1.1 se muestra parte de las tecnologías que NetBeans soporta:

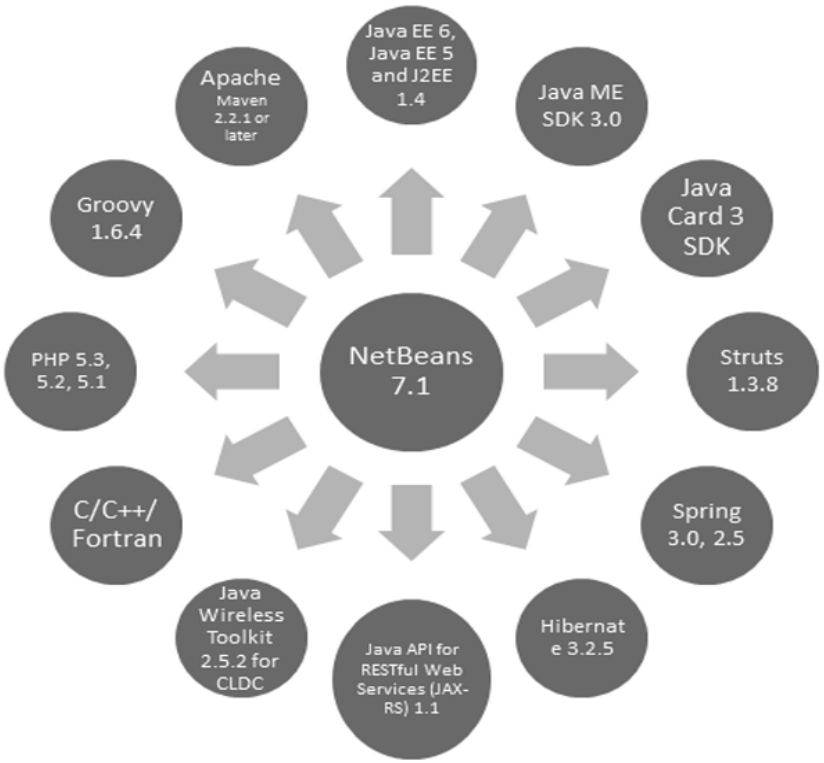


Figura 1.1 Soporte NetBeans 7.1 de otras tecnologías.

Actividades para el lector

Realice una investigación, determine si existen otras herramientas similares a NetBeans.

En un reporte enumérelas, liste e identifique sus principales características.

DESCARGA E INSTALACIÓN DE NETBEANS 7.1

La descarga de NetBeans 7.1 se puede realizar desde el sitio de la organización http://NetBeans.org/ y posteriormente iniciar la instalación del mismo. Este sitio permitirá descargar cualquiera de las versiones de Java que implementa NetBeans 7.1 (Java SE (Estándar Edition), EE (Enterprise Edition), C/C++, PHP o la suite completa)

Una vez en este sitio se presentará una pantalla de descarga similar a la de la Figura 1.2.

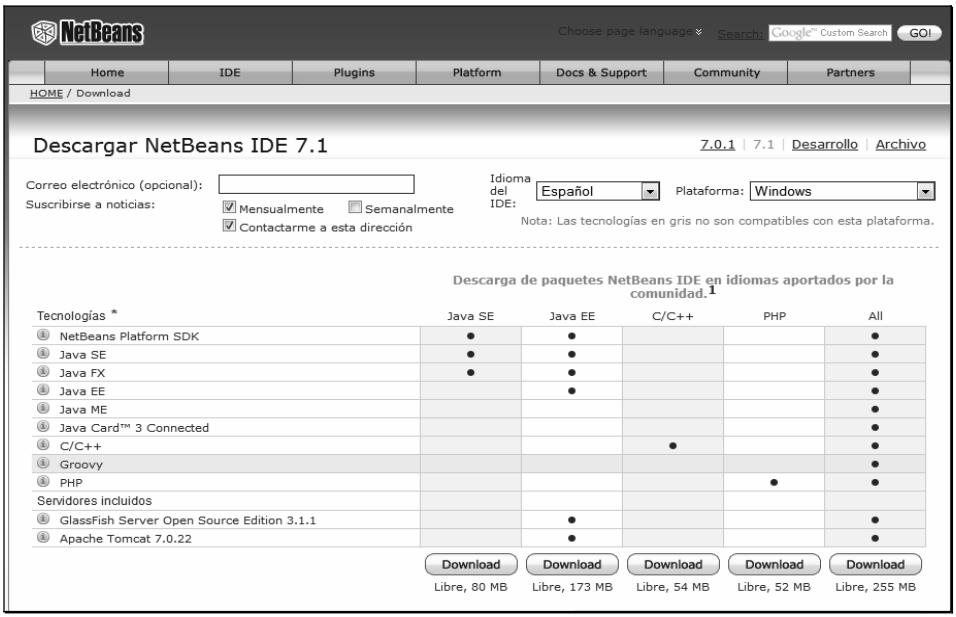


Figura 1.2 Descarga de NetBeans 7.1.

Descargue la versión All y guárdela en alguna carpeta especial donde vaya almacenando todos los programas que necesitará para trabajar con NetBeans 7.1. Una vez que termina la descarga debe proceder a bajar el JDK (Java Development Kit). Puede hacerlo desde la página *http://jdk7.java.net/download.html.* Una vez descargado proceda a instalarlo. Una pantalla similar a la Figura 1.3 se mostrará.

Figura 1.3 Instalación del JDK 7.

Una vez que ha instalado el JDK proceda a instalar el NetBeans. La interface de instalación iniciará con la clásica interface de NetBeans, la cual se muestra en la Figura 1.4.

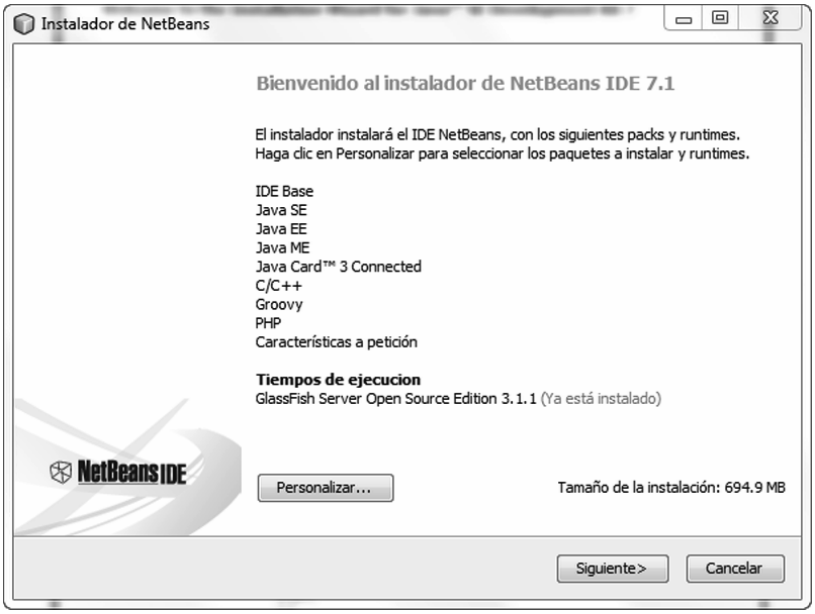


Figura 1.4 Inicio de la instalación de NetBeans 7.1.

Una vez instalado ya tiene toda la funcionalidad de NetBeans 7.1 para que comience a explorar y explotar todo el poder de esta majestuosa herramienta de desarrollo.

EL ENTORNO NETBEANS 7.1

Una vez que ejecuta NetBeans 7.1, se presentará el entorno (el IDE) de la herramienta. Básicamente se podrá notar que mantiene la funcionalidad y los rasgos de las versiones anteriores, con la facilidad de algunas nuevas prestaciones que lo han hecho evolucionar y constituirse en una herramienta de mucha importancia entre los desarrolladores de software.

Si ha utilizado versiones anteriores de este IDE podrá darse cuenta que la lógica funcional de la herramienta no ha cambiado. El entorno sigue siendo similar a las versiones anteriores, tal y como se muestra en la Figura 1.5.

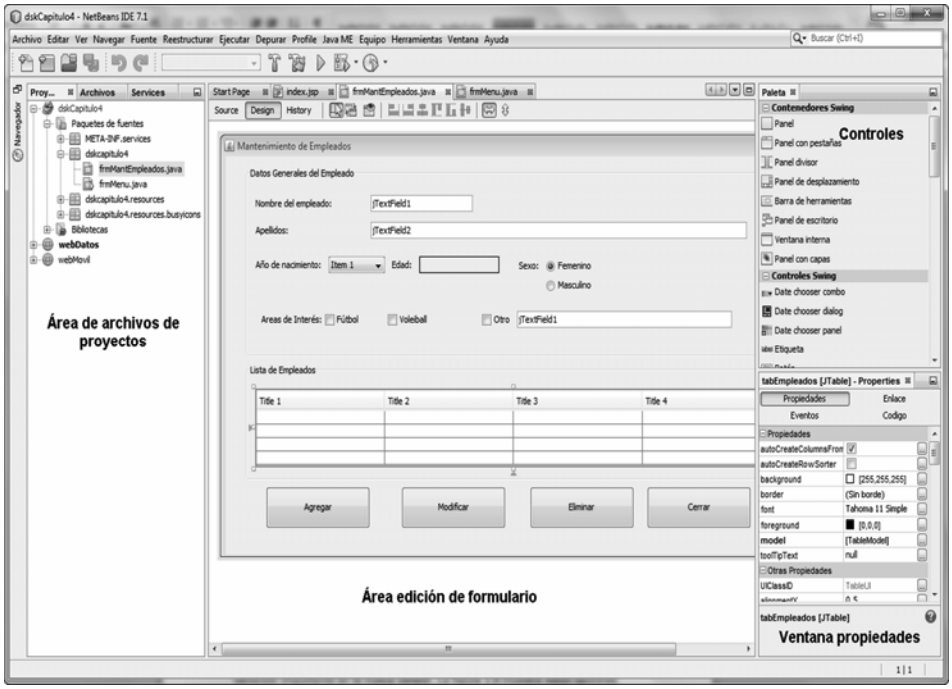


Figura 1.5 Entorno NetBeans 7.1.

Como podrá notar, siempre existe a la izquierda el área de proyectos (aquí se visualizan otras ventanas como las de Archivos, Prestaciones, Paleta, entre otras). Al centro del IDE se muestran los formularios (con su interface de diseño y su editor de código fuente), y a la derecha se muestran los controles que se puede utilizar en nuestras aplicaciones, así como la ventana de propiedades de cada objeto en el formulario actual. Arriba de estas secciones se encuentran el menú principal del IDE y los iconos de ejecución rápida. Si está familiarizado con ellos notará que no hay variación importante en la nueva versión. La Figura 1.6 muestra estas opciones.

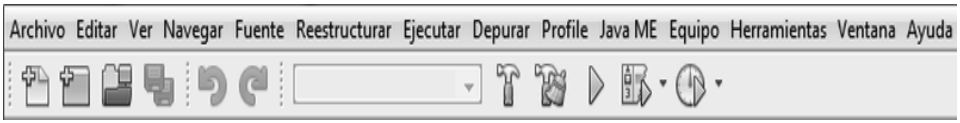


Figura 1.6 Menú principal e íconos rápidos de NetBeans 7.1.

Actividades para el lector

Navegue por el menú de NetBeans y describa las opciones que se presentan en la barra; al terminar el estudio de este capitulo complete su información con aquellas que no fueron descritas en él.

Del menú principal se pueden rescatar algunas funcionalidades importantes y quizás las más utilizadas a nivel de principiante o intermedio. Requerirá un poco de investigación la funcionalidad de las demás opciones del menú. La tabla 1.1 muestra la funcionalidad de algunas opciones de Archivo.

Tabla 1.1 Opciones del menú Archivo.

OPCIÓN	FUNCIÓN
Proyecto Nuevo…	Permite crear un proyecto nuevo NetBeans.
Archivo Nuevo….	Permite crear un archivo nuevo tipo Javacard, Java, JUnit, entre otros.
Abrir Proyecto….	Abre un proyecto existente.
Abrir Proyecto Reciente	Abre el último proyecto abierto.
Cerrar Project (…)	Cierra el proyecto actual.
Abrir archivo…	Abre un archivo existente tipo Javacard, Java, JUnit, entre otros.
Abrir archivo reciente	Abre el último archivo abierto.
Grupo de Proyecto	Permite abrir un conjunto de proyectos o crear uno nuevo.
Proyecto Properties (…)	Habilita las propiedades del proyecto.
Importar Proyecto	Permite importar un proyecto Eclipse.
Guardar	Guarda el objeto actual (formulario, clase, entre otros)
Guardar como…	Permite guardar el objeto actual con otro nombre.
Guardar todo	Permite guardar todos los objetos del proyecto.

Configuración de Página	Configura la página para impresión.
Imprimir	Imprime el contenido del objeto actual
Escribir en HTML…	Permite escribir o imprimir en HTML el código del objeto actual.
Salir	Permite salir del entorno de NetBeans

Para el menú Editar no hay mayor cosa qué explicar, dado que son las operaciones usuales de cualquier aplicativo: copiar, cortar, pegar, entre otros. Quizá destacar la opción Encontrar usos, cuya funcionalidad es la de permitir buscar los usos que se le da al objeto actual entre todos los objetos que componen la aplicacion.

El menú ofrece opciones como ejecutar el navegador, mostrar las barras de herramientas o mostrar el IDE a pantalla completa, entre otras funcionalidades.

El menú Navegar ofrece opciones tales como Ir al archivo, lo que permite ubicar un determinado archivo en el proyecto; seleccionar miembros del archivo o seleccionar proyectos, archivos, entre otras funciones.

El menú Fuentes permite gestionar la edición de código, ofreciendo funcionalidades tales como formatear, eliminar, duplicar o insertar código, entre otras operaciones.

El menú Reestructurar ofrece, entre otras, funcionalidades como las que se muestran en la tabla 1.2.

Tabla 1.2 Opciones del menú Reestructurar.

OPCIÓN	FUNCIÓN
Cambiar de nombre…	Habilita la posibilidad de cambiar el nombre del objeto seleccionado realizando los cambios de nombres en todos los demás objetos que lo utilizan.
Eliminación segura….	Permite eliminar el objeto seleccionado manteniendo la seguridad de eliminar las dependencias que posee el objeto.
Extraer interfaz….	Permite refactorizar. Permite crear una nueva interfaz basada en los métodos públicos no estáticos de una clase seleccionada.
Extraer superclase…	Refactorización. Permite crear una superclase basada en métodos públicos no estáticos en la clase seleccionada.
Encapsular campos	Mediante esta opción se pueden crear métodos de acceso para los campos de una clase. Es posible establecer métodos de acceso de lectura / escritura de una clase definida.

El menú Ejecutar permite las funcionalidades que se muestran en la tabla 1.3.

Tabla 1.3 Opciones del menú Ejecutar.

OPCIÓN	FUNCIÓN
Ejecutar Main Project	Permite ejecutar el código del Main del Proyecto.
Probar Project (…)	Habilita la posibilidad de probar la aplicacion actual (ejecución)
Generar Main Project	Crea el archivo JAR de la aplicacion.
Limpiar y generar Main Project	Elimina cualquier existencia de variables en memoria producto de una creación anterior del JAR y crea éste de nuevo.
Establecer la configuración del proyecto	Posibilita que se establezca la configuración general del proyecto considerando los tipos de fuentes, las bibliotecas a utilizar, formato, entre otras opciones.
Establecer como Proyecto Principal	Habilita la opción de establecer un proyecto existente entre un grupo de proyectos, como el principal de la aplicatción.
Generar Javadoc	Genera el archivo Doc de la aplicacion que conjuntamente con el archivo JAR son importantes en el proyecto. Es el código fuente de las clases que acompañan al archivo JAR.
Ejecutar archivo	Permite ejecutar el archivo seleccionado.

El menú Depurar ofrece una serie de opciones que permiten al desarrollador revisar el código y la funcionalidad del mismo mediante la ejecución pausada. Opciones tales como Pausa, Continuar, Paso a Paso, entre otras, permiten ir ejecutando el código de nuestro aplicativo e ir conociendo su comportamiento en todo momento y darnos cuenta si existe un error sintáctico o funcional del mismo.

El menú Profile ofrece la posibilidad de que se pueda analizar el rendimiento de nuestro aplicativo. Esto permite analizar nuestro código si no estamos satisfechos con su rendimiento. Una interfaz similar a la que se presenta en la Figura 1.7 se nos podría presentar utilizando esta opción.

Figura 1.7 Utilizando Profile de NetBeans 7.1.

En el menú Herramientas se encuentran algunas funciones importantes de detallar. La tabla 1.4 describe algunas de las opciones de este menú.

Tabla 1.4 Opciones del menú Herramientas.

OPCIÓN	FUNCIÓN
Analizar Javadoc	Javadoc es una utilidad Oracle que se utiliza para documentar las clases Java. Esta opción analiza las implementaciones de clases Java en un proyecto NetBeans.
Plataformas Java	Muestra y administra la plataforma Java. Se pueden agregar o eliminar plataformas (por ejemplo un control de fechas).
Variables	Permite administrar variables globales en la plataforma.
Bibliotecas	Posibilita la administración de bibliotecas (JAR) para aportar mayor funcionalidad al IDE.
Servidor	Permite administrar servidores tales como GlassFish, Tomcat u Oracle WebLogic Server. Servidores de Internet.
Plantillas	Permite administrar plantillas Assembler, C, C++, Fortran, etc.
Complementos	Permite la administración de las actualizaciones y los plugins utilizados por el IDE, tales como Maven, Ruby and Rails, GlassFish, entre otros.
Opciones	Permite la configuración general, del editor, tipo de letras y colores, mapa de teclado y varios.

El menú Ventana ofrece la posibilidad de habilitar un escenario de acuerdo con la funcionalidad requerida por el desarrollador. Por ejemplo, la opción Project de este menú facilita visualizar el escenario de los componentes del proyecto; Archivos permite mostrar los archivos del proyecto; Prestaciones las conexiones y bases de datos establecidas en la aplicacion; Paleta la ventana de controles; entre otras ventanas.

Actividades para el lector

Haga una breve descripción de la funcionalidad de todos y cada uno de los íconos rápidos de NetBeans, que generalmente se ubican debajo de la barra de menús.

TIPOS DE PROYECTOS NETBEANS 7.1

Los tipos de proyectos existentes en NetBeans 7.1 son variados y se orientan a varias funcionalidades diferentes. Podemos citar las funcionalidades básicas a los cuales un lenguaje de programación podría orientarse:

- aplicaciones de escritorio (desktop)
- aplicaciones web
- aplicaciones para dispositivos móviles.

A todas ellas NetBeans 7.1 les apoya con las herramientas adecuadas. A continuación se resumen en tablas los principales proyectos NetBeans que se incluyen en la herramienta, agrupados por tecnologías funcionales. En este caso, los proyectos que se citan a continuación se enmarcan dentro de la tecnología de Java SE (Standar Edition), que es una versión liviana de Java para crear aplicaciones no tan complejas.

Iniciaremos con Java SE (Standar Edition), el cual es la versión "liviana" que se implementa en NetBeans para el desarrollo de aplicaciones de escritorio, bibliotecas de clases o la reutilización de código de proyectos existentes. La tabla 1.5 describe los tipos de proyectos que ofrece esta tecnología.

Tabla 1.5 Proyectos de la tecnología Java.

TIPO PROYECTO	ESPECIFICACIÓN
Aplicación Java	Permite crear un proyecto Java vacío. Debemos crear toda las interfaces y clases desde cero.
Aplicación de escritorio Java	Permite crear un proyecto Java tipo escritorio, con el diseñador de interfaces sin necesidad de programar los controles desde cero.
Biblioteca de clases Java	Permite crear una biblioteca de clases Java donde se implemente toda una funcionalidad específica; posteriormente se podrá integrar al proyecto simplemente anexándolo.
Proyectos Java con fuentes existentes	Permite crear un nuevo proyecto basado en archivos de código fuente que ya existen.
Proyectos Java Free-Form	Importa un proyecto Java o modulo Web existente y crea una secuencia de órdenes para ejecutar todas las acciones que se desea realice el proyecto.

Otra tecnología presente es Java web, la cual se orienta a la creación de soluciones orientadas a Internet. En la tabla 1.6 se resumen los proyectos de esta tecnología.

Tabla 1.6 Proyectos de la tecnología Java web.

TIPO PROYECTO	ESPECIFICACIÓN
Web Application	Permite crear un aplicativo web vacío.
Web Application with Existing Sources	Utilizando código fuente existente se puede crear una aplicación web utilizando esta opción.
Web Free-Form Application	Importa un proyecto Java existente para crear un nuevo proyecto web con la estructura que posee.

Java EE (Enterprise Environment) está concebido para el desarrollo de aplicaciones de gran complejidad. Permite el desarrollo de N capas en forma distribuida, apoyándose fuertemente en sistemas modulares. Se le considera un estándar mundial dado que los proveedores del mismo deben cumplir con una serie de especificaciones de conformidad para poder declarar que los productos que distribuyen son conformes a Java EE. En la tabla 1.7 se resumen los tipos de proyectos que podemos crear con esta tecnología.

Tabla 1.7 Proyectos de la tecnología Java EE.

TIPO PROYECTO	ESPECIFICACIÓN
Enterprise Application	Permite crear una aplicación Java de altas prestaciones y complejidad.
Enterprise Application with Existing Sources	Permite reutilizar código de otro proyecto para crear un nuevo aplicativo Java EE.
EJB Module	Se utiliza para ensamblar "enterprise beans" en una unidad. Este ensamblado se almacena en un único archivo de Java (JAR) estándar. Un Enterprise JavaBeans (EJB) constituye una API que forma parte de los estándares para la construcción de aplicaciones JEE.
EJB Module with Existing Sources	Permite construir un módulo EJB reutilizando código existente.
Enterprises Application Client	Mediante esta opción se puede desarrollar una aplicación JEE basado en el cliente. Este programa cliente se ejecuta sobre una máquina virtual diferente al servidor JEE.
Enterprises Application Client with Existing Sources	Permite reutilizar código en la creación de una aplicación JEE cliente.
Packaged Archive	Permite crear un empaquetado de archivos de un proyecto JEE para facilitar el proceso de redistribución. Puede ser un archivo JAR, WAR, EAR o CAR.

Otra de las tecnologías implementadas en NetBeans es Java Card. Ésta permite ejecutar pequeñas aplicaciones escritas en Java (applets) en dispositivos empotrados. Puede ser un pequeño aplicativo como el implementado en una tarjeta SIM de un teléfono móvil que utiliza la tecnología GSM para comunicarse.

En la tabla 1.8 se observan algunos de los proyectos que se pueden crear con esta tecnología

Tabla 1.8 Proyectos de la tecnología Java Card.

TIPO PROYECTO	ESPECIFICACIÓN
Classic Applet Project	Permite desarrollar un applet que básicamente es una máquina de estados que procesa los comandos que se reciben a través del dispositivo, emitiendo respuestas con códigos de estado y datos.
Extended Applet Project	Permite el instanciamiento de una aplicativo applet en el dispositivo o el navegador para que el cliente tenga acceso.
Classic Library Project	Permite crear una librería clásica para un proyecto Java Card.
Extension Library Project	Permite crear librerías Java Card con extensión.
Web Project	Permite crear un proyecto web mediante Java Card.

Finalmente, la tecnología Java ME (Micro Edition) permite la creación de aplicaciones para dispositivos móviles. En la tabla 1.9 se resumen algunos tipos de proyectos que se pueden crear con esta tecnología.

Tabla 1.9 Proyectos de la tecnología Java ME

TIPO PROYECTO	ESPECIFICACIÓN
Mobile Application	Permite crear una aplicación Java ME para dispositivos móviles.
Mobile Class Library	Admite la creación de librerías de clase para utilizar en un aplicativo para móviles mediante Java ME.
Mobile Project with Existing MIDP Sources	Permite la creación de aplicaciones para dispositivos móviles, reutilizando código existente.
Import Wireless Toolkit Project	Importa un proyecto con la funcionalidad de dicho proyecto.
CDC Application	Permite crear una aplicación para un dispositivo CDC (en un capítulo posterior trataremos este tipo de dispositivos).
CDC Class Library	Admite la creación de una librería de clases para CDC.
Import CDC Pack 5.5 Project	Importa un proyecto existente CDC.
Import CDC Toolkit Project	Importa un proyecto CDC con la funcionalidad de dicho proyecto.
Mobile Designer Components	Permite diseñar componentes de aplicaciones para dispositivos móviles.

RESUMEN

En este capítulo se presentó una introducción a NetBeans 7.1, señalando principalmente sus bondades y las funcionalidades que ofrece al desarrollador de software. En especial se trataron los siguientes temas:

1. Lo nuevo de NetBeans 7.1. Nuevas funcionalidades y las mejoras que se le han realizado al IDE, entre otras características del lenguaje.
2. Cómo descargar e instalar el IDE.
3. El entorno del IDE con la funcionalidad de sus menús y opciones.
4. Los tipos de proyectos que se pueden crear con NetBeans 7.1

Autoevaluación

1. ¿Qué representa NetBeans?

2. ¿Cuáles son los productos creados por Sun MicroSystem para NetBeans?

3. ¿Cuál es la funcionalidad de NetBeans Platform?

4. ¿Para qué se utiliza WebLogic Server?

5. La opción de menú que permite crear una interfaz basada en los métodos públicos no estáticos de una clase determinada es.

EVIDENCIA

☐ Realizó una investigación acerca del software y determinó si existen otras herramientas similares a NetBeans. Las enumeró e identificó sus principales características.

☐ Navegó por el menú de NetBeans y describió las opciones que se presentan en la barra de éste y que no fueron descritas en este capítulo.

☐ Citó la funcionalidad de todos y cada uno de los íconos rápidos de NetBeans, que generalmente se ubican debajo de la barra de menús.

REFERENCIAS

Bibliografía

Ceballos, Fco. Javier, (2011). *Java 2: Curso De Programación*, 4a. ed., Alfaomega, México.

Martín, Antonio, (2010). *Programador certificado Java 2. Curso práctico*, 3a. ed., Alfaomega, México.

Rozannski, Uwe, (2010). Enterprise Javabeans 3.0. Con Eclipse Y JBoss. Alfaomega, México.

Sznajdleder, Pablo, (2010) *Java a fondo. Estudio del lenguaje y desarrollo de aplicaciones.* Alfaomega, México.

Páginas Web recomendadas

1. netBeans.org (2011) NetBeans IDE 7.0.1 Releases Notes. Obtenido el 28 de junio de 2012desde http://netbeans.org/community/releases/70/relnotes.html#new
2. programania.net (2006) Servicios web: REST versus SOAP. Obtenido el 28 de junio de 2012desde http://www.programania.net/diseno-de-software/servicios-web-rest-versus-soap/
3. Oracle (2011) *Oracle WebLogic Server*. Obtenido el Obtenido el 30 de junio de 2012desde *http://www.oracle.com/technetwork/middleware/weblogic/overview/index.html*
4. Glashfish.java.net (2011) *GlassFish Server Opensource* Edition. Obtenido el 28 de junio de 2012desde http://glassfish.java.net/
5. Netbeans.dzone.com (2008) *Working with Java Free-Form Projects in NetBeans IDE.* Obtenido el 30 de junio de 2012desde *http://netbeans.dzone.com/tips/working-java-free-form-project*.
6. netBeans.org (2011) *NetBeans IDE Guide for Jbuilder Users.* Obtenido el 28 de junio de 2012desde *http://netbeans.org/kb/articles/import-jbuilder.html#import-web*
7. IBM (2011) *Módulos EJB*. Obtenido el 29 de junio de 2012desde http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.express.doc/info/exp/ae/cejb_emod.html
8. Osmosislatina.com(2011) *Un "Java Bean" NO ES LO MISMO que un "Enterprise Java Bean".* Obtenido el 29 de junio de 2012desde http://www.osmosislatina.com/java/ejb.htm.

9. Java.sun.com (2011) *Creating the J2EE Application Client.* Obtenido el 30 de junio de 2012desde *http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/GettingStarted5.html.*
10. Oracle (2011) *Java Card Technology*. Obtenido el 30 de junio de 2012desde http://www.oracle.com/technetwork/java/javacard/overview/index.html

Respuestas a la autoevaluación

1. Es un proyecto de código abierto que tiene una gran cantidad de usuarios alrededor del mundo que colaboran en el desarrollo de este IDE.
2. SUN MicroSystem ha desarrollado NetBeans IDE y NetBeans Platform.
3. NetBeans Platform permite la integración de módulos en grandes y complejas aplicaciones de escritorio en Java.
4. WebLogic Server proporciona una serie de componentes estándares modulares que se utilizan para el desarrollo de aplicaciones Java EE.

   Extraer interfaz del menú Reestructurar.
5. Extraer interfaz del menú Reestructurar, dado que es una respuesta a la autoevaluacion.

Fundamentos de programación en NetBeans 7.1

2

Reflexione y responda las siguientes preguntas

¿En qué se diferencia una variable de una constante?

¿Hay alguna diferencia entre un procedimiento y una función?

¿Las sentencias de control se manejan de la misma manera en todos los lenguajes de programación?

¿Existen las variables globales en un sistema basado en clases?

CONTENIDO

EXPECTATIVA

Conocer y emplear constantes, variables, tipos de datos, operadores y sentencias de control, las cuales son parte de la fundamentación de cualquier lenguaje de programación que se debe conocer para iniciar la etapa como programadores. Es casi seguro que una vez que se aprende la sintaxis de las instrucciones o comandos de un determinado lenguaje, no se tendrán problemas si se encuentra con otro. Los cambios son poco significativos o muy entendibles, inclusive sin diferenciación alguna.

Después de estudiar este capítulo, el lector será capaz de

- Reconocer y emplear los conceptos de constantes y variables que se utilizan en Netbeans 7.1.
- Conocer los tipos de datos que se emplean en Netbeans 7.1.
- Aplicar adecuadamente los tipos de datos en un programa computacional.
- Utilizar la funcionalidad de los operadores que se usan en Netbeans 7.1 para la solución de un problema computacional.
- Aplicar las estructuras de control en la solución de un problema computacional mediante Netbeans 7.1.

INTRODUCCIÓN

Para iniciar el conocimiento de un lenguaje de programación es importante hacerlo conociendo su estructuración, la forma en que hace las cosas, cuál es su sintaxis, declarativas, comandos, entre otros elementos. Java es un lenguaje muy rico en expresiones algorítmicas, funcionalidades extraordinarias para construir una aplicación de escritorio o un sofisticado juego de video. Sin embargo, se deben ir conociendo poco a poco sus prestaciones para poder ir acumulando conocimientos para luego hacer prácticamente lo que se quiera en cuanto a aplicaciones se refiere.

CONSTANTES EN NETBEANS 7.1

Una constante se puede definir como un valor que permanece inalterable en el tiempo y que se utiliza para incluirlo en uno o varios cálculos, como referencia para un título o para almacenar un valor que se desea sea siempre el mismo. El concepto se maneja igual en todos los lenguajes de programación, por tanto, es una conceptualización muy genérica.

Hay muchas formas de expresar la declarativa de una constante. Básicamente depende del lenguaje de programación, pero en el fondo es la misma funcionalidad. La sintaxis del lenguaje juega un papel importante en este contexto, sin embargo nos orientaremos por la utilizada en Java.

En Java se utiliza la palabra clase *final* y static para declarar una constante. Una vez declarada o inicializada su valor no puede cambiarse durante el tiempo de ejecución.

La sintaxis para la declarativa de una constante es:
static final nombreConstante = valor;

A continuación se presenta el listado de un programa en Java que utiliza constantes.

EJEMPLO: DECLARACIÓN, INICIALIZACIÓN E IMPRESIÓN DE CONSTANTES EN JAVA

```
public static void main(String[] args){
    static final int diasLaborales = 5;
    static final int diasNoLaborales = 2;
     System.out.println("Número días laborables " + diasLaborales);
    System.out.println("Número días No laborables " + diasNoLaborales);
    }
```

Como se puede observar en el código anterior, la segunda y tercera filas del listado declaran dos variables (diasLaborables y diasNoLaborables) y los inicializan en 5 y 2 respectivamente. También se puede observar que se le antepone las instrucciones static final a cada uno. Con ello se declara e inicializa dos constantes. En las dos líneas siguientes (4 y 5) simplemente se imprime por pantalla los valores correspondientes de esas constantes.

Un ejemplo básico, utilizando una clase, es el que se presenta a continuación:

EJEMPLO: DECLARACIÓN E INICIALIZACIÓN DE CONSTANTES EN JAVA, UTILIZANDO UNA CLASE

```
public class factura {
  static final double imp=0.13 ;
  static double impuesto(double monto)
   {
      return monto * imp; }
  }
```

Se puede observar en el código anterior *imp* se declaró como una constante con un valor inicial invariable, con el cual se retorna un cálculo a través de la función impuesto. Por ahora, sólo observe la instrucción *static final imp=0.13* y notará que es muy simple declarar, inicializar y utilizar una constante en Java.

VARIABLES Y TIPOS DE DATOS

Una variable es un valor que cambia durante la ejecución de un programa. Este valor puede ser de cualquier tipo. Por ende, antes de iniciar el tratamiento del tema de las variables se indicará los tipos de datos que se utilizan en Java y cuál es el tamaño que se les asigna.

Tipos de datos enteros

Estos tipos de datos se utilizan como enteros sin signo. La tabla 2.1 muestra este tipo de dato y su tamaño.

Tabla 2.1 Tipo de datos enteros.

TIPO DE DATO	TAMAÑO
byte	1 byte (8 bits)
short	2 bytes (16 bits)
int	4 bytes (32 bits)
long	8 bytes (64 bits)

Literales enteros

Los literales enteros en Java presentan tres formatos:

- Decimal: representan números ordinarios sin ninguna notación especial.
- Hexadecimal: representados en base 16 con una notación 0_x o 0X, similar a la que se utiliza en C/C++
- Octal: se representan mediante un 0 inicial, antecediendo a los dígitos.

A continuación se presenta un programa que demuestra el uso de datos de tipo entero.

EJEMPLO: DECLARACIÓN DE VARIABLES DE TIPO ENTERO

```
public class Enteros {
    public static void main(String[] args){
    {
      byte dato1 = 1;
      short dato2 = 100;
      int dato3 = 1000;
      long dato4 = 10000000;
     system.out.println("Dato tipo byte " + String.valueOf(dato1);
     system.out.println("Dato tipo entero corto " + String.valueOf(dato2);
     system.out.println("Dato tipo entero largo " + String.valueOf(dato3);
    }
```

Tipo de dato coma flotante

Representan números con partes fraccionarias. Se utilizan dos formatos: *float* (almacena un numero de precisión simple de 4 bytes/32 bits) y *double* (almacena un número de precisión doble de 8 bytes/64 bits).

Literales en coma flotante

Representan números con partes fraccionarias que pueden representarse con notación estándar o científica. Por default son de tipo double (8 bytes). También puede ser de tipo float (4 bytes). Por ejemplo, se pueden declarar este tipo de literales de la siguiente manera:

```
double Pi = 314.16e-2 ; // Valor aproximado de π
float tempDia = (float) 29.6; // temperatura del día
```

Tipo de dato boolean

Representan valores que almacenan uno de dos estados: verdadero o falso.

EJEMPLO: DECLARACIÓN DE VARIABLES DE TIPO BOOLEAN

```
public class pruebaBoolean {
  boolean verificar(int numero)
  {
    boolean aux = false;
    if (numero > 0)
      aux = true;
    return aux;
  }
  }
```

Como puede observarse se tiene una clase booleana denominada pruebaBoolean que contiene una función llamada verificar la cual recibe un valor entero (numero).

Al final valida si ese número es mayor a cero devolviendo verdadero si lo es y false si no lo es.

Tipo de dato cadena

Para utilizar datos de tipo cadena se debe emplear la clase String, que posee dos constructores, los cuales se muestran mediante un ejemplo para cada uno:

- String nombre = new String;
- String pais = new String(“Costa Rica”);

o simplemente se declara como String nombre, sin asignarle algún valor inicial.

Vectores y matrices

Los tipos de datos manejados anteriormente se denominan estructuras primitivas. Sólo pueden manejar un valor por cada variable que se declare, así que cada vez que se lee un dato y se almacena en ella se pierde el valor anterior. En ciertas ocasiones es necesario conocer cuáles son los valores de cierto escenario, por ejemplo los nombres de 5 estudiantes o los salarios de 10 trabajadores. Si se utiliza una variable String para los estudiantes o de tipo double para los salarios de los trabajadores, probablemente sólo se conocerá el último valor asignado a cada una de esas variables. Es ahí donde toma importancia el uso de vectores y matrices.

Vectores

¿Qué es un vector? Un vector se define como una estructura con varias posiciones donde se pueden almacenar valores de un mismo tipo de dato. Por ejemplo, se tiene un vector de tipos de datos enteros con 5 posiciones o un vector de cadena con 10 posiciones. A continuación se presenta un programa que muestra la manera de declarar vectores, cómo almacenar valores en ellos y cómo extraerlos para mostrarlos.

EJEMPLO DE USO DE VECTORES
public class pruebaArreglo {
public static void main(String[] args)
{
int vector[] = new int[5];
String vNombres[] = {“Juan”,”Pedro”,”María”}
vector[0]=5; vector[1]=35; vector[2]=16; vector[3]=4; vector[4]=7;
system.out.println("Dato posicion 2 " + String.valueOf(vector[1]);
system.out.println("Dato posicion 2 " + vNombres[1]);
} }

Como se observa en el código anterior, se declara un vector de enteros denominado *vector* y otro de tipo cadena llamado *vNombres*. El primero tiene 5 posiciones y el segundo sólo tres. Posteriormente, tanto al vector de nombres como al de enteros se les asigna valores iniciales. Finalmente, se despliega el contenido de la posición de ambos vectores, produciendo la salida, 35 (posición 1 del vector de enteros) y “Pedro” (posición 1 del vector de cadena).

Matrices

¿Qué es una matriz? Una matriz es una colección de celdas bidimensionales (filas y columnas) donde se pueden almacenar valores de un mismo tipo. A diferencia de un vector que es unidimensional (se considera que tiene una sola fila o una sola columna) se pueden almacenar valores tanto a nivel de fila como de columna.

El siguiente programa demuestra el uso de este tipo de estructura.

EJEMPLO DE USO DE MATRICES
public class pruebaMatriz {
public static void main(String[] args)
{
int matriz[] = new int[3] [4]; //3 columnas, 4 filas
matriz[0][0] = 15; matriz[1][0] = 24; matriz[2][0] = 38;

EJEMPLO DE USO DE MATRICES
matriz[0][1] = 17; matriz[1][1] = 64; matriz[2][1] = 83;
matriz[0][2] = 65; matriz[1][2] = 33; matriz[2][2] = 28;
matriz[0][3] = 28; matriz[1][3] = 17; matriz[2][3] = 99;
system.out.println("Contenido " + matriz[2] [0]);
system.out.println("Contenido " + matriz[0] [3]);
} }

En el código anterior se declara una matriz de 3 columnas y 4 filas. Posteriormente se cargan los datos en la matriz, considerando columna, fila (columna 0, fila 0; columna 1, fila 0; columna 2, fila 0) y así sucesivamente llenando cada columna para cada fila hasta agotar el número de columnas (3) por cada una de las filas (4). Finalmente, se despliegan los valores para la matriz columna 2, fila 0 que es el valor 38 y para columna 0, fila 3 cuyo valor es 28.

Por último, es importante indicar que se pueden utilizar métodos públicos sobre un vector o matriz, como lo es determinar la longitud o tamaño de cualquiera de ellos mediante *length*.

Colecciones

Las colecciones establecen una referencia a un grupo de objetos (de tipo object), a diferencia de los arreglos. En este sentido, se puede almacenar cualquier objeto en una colección y para ser accedidos se requiere hacer casting sobre los objetos de dicha colección. Se puede cambiar el tamaño de la colección en forma dinámica, ordenarlos, insertar o borrar objetos. A continuación, en la tabla 2.2 se presenta un resumen de las principales colecciones que se utilizan en Java.

Actividades para el lector

Desarrolle un programa que implemente al menos dos de las siguientes estructuras complejas

- ArrayList
- Vector
- LinkedList
- HashSet
- LinkedHashSet
- TreeSet
- TreeMap
- PriorityQueue

Tabla 2.2 Colecciones en Java.

COLECCIÓN	DESCRIPCIÓN
ArrayList	Tipo de arreglo que incrementa y/o decrementa sus elementos en forma dinámica. Su iteración es rápida y el método de acceso es aleatorio. No es recomendable su utilización cuando se producen gran cantidad de inserciones y eliminaciones de elementos.
Vector	Es una colección que implementa las mismas características de un ArrayList, con la diferencia que los métodos que implementa se sincronizan mediante el uso de multihilos seguros.
LinkedList	Similar a un ArrayList, con la diferencia que sus elementos se enlazan en forma doble, permitiendo agregar o eliminar elementos al principio o al final de la lista. Puede implementarse fácilmente una pila o una cola, aunque existe otra librería que permite esto.
HashSet	Este tipo de estructuras no permite insertar elementos duplicados en una lista. No mantiene ordenados sus elementos.
LinkedHashSet	Es una versión HashSet que sí mantiene ordenada una lista y no permite duplicados. Puede ser útil para implementar un diccionario o un listado telefónico donde se necesita que los valores no se dupliquen.
TreeSet	Esta estructura permite que los elementos se mantengan ordenados en forma ascendente. Se puede modificar el orden mediante el uso de los métodos Comparable o Comparator.
TreeMap	Mapa que garantiza que los elementos se mantengan ordenados de manera ascendente. Se puede modificar el orden mediante el uso de los métodos Comparable o Comparator.
PriorityQueue	Permite la creación de colas de prioridad. Los elementos se ordenan y mediante un Comparator se puede establecer la prioridad de cada elemento

La implementación de estas estructuras se puede estudiar a fondo en el sitio

http://www.dccia.ua.es/dccia/inf/asignaturas/RG/pdf/colecciones-java.pdf

OPERADORES EN JAVA

Los operadores en Java (y en cualquier otro lenguaje) se dividen en aritméticos, relacionales y lógicos. Los operadores aritméticos nos permiten realizar operaciones matemáticas sobre un conjunto de valores. Los operadores relacionales nos permiten usar comparativas entre elementos o valores; finalmente, los operadores lógicos nos permiten seleccionar acciones de acuerdo con un criterio.

La tabla 2.3 muestra los operadores aritméticos que tiene Java.

Tabla 2.3 Operadores aritméticos.

OPERADOR	REPRESENTACIÓN	EJEMPLO
Asignación	=	int b = 2+3;
Suma	+	a+=5; //a=a+5
Resta	-	a-=5; //a=a-5
Multiplicación	*	A*=5; //a=a*5
División	/	a/=5; //a=a/5
Módulo	%	A = 4 % 2;
Potencia	Math.pow(base,exponente)	Math.pow(4,2);
Raíz cuadrada	Math.sqrt(radicando)	Math.sqrt(16);
Incremento de 1	++	a++; ++a;
Decremento de 1	--	a--; --a;

Los operadores relacionales con que cuenta Java se muestran en la tabla 2.4.

Tabla 2.4 Operadores Relacionales.

OPERADOR	REPRESENTACIÓN	EJEMPLO
Mayor que	>	if (a>b) …//si a es mayor que b
Menor que	<	If (a<b) …//si a es menor que b
Mayor o igual que	>=	if (a>=b) …//si a es mayor o igual que b.
Menor o igual que	<=	If (a<=b) …//si a es menor o igual que b.
Igual que	==	if(a==b) …//si es similar a b
Diferente que	¡=	if(a!=b) …//si a es diferente de b

Finalmente, en la tabla 2.5 se muestran los operadores lógicos.

Tabla 2.5 Operadores lógicos

OPERADOR	REPRESENTACIÓN	EJEMPLO
and	&&	if (a>b) && (a>c) … //si a es mayor que b // y a es mayor que c
not	!	if (¡b) … //no b
or	||	if (a>b) || (a>c) … //si a es mayor que b //o a es mayor que b

ESTRUCTURAS DE CONTROL

Las estructuras de control se utilizan en un programa para regular las acciones que se puedan presentar en la ejecución del código. Mediante estas estructuras se pueden tomar decisiones de acuerdo con criterios lógicos establecidos en el código de un programa. En la tabla 2.6 se muestran las estructuras de control que se implementan en Java.

Tabla 2.6 Estructuras de control en Java.

ESTRUCTURA DE CONTROL	EJEMPLO	DESCRIPCIÓN

if(condición-true)
{
Sentencias;
}
else
{
Sentencias;
}
	if (a>b)
	{
	…
	}
	else
	{
	…
	}
		Mediante if se puede evaluar la condición inicial y si está resulta positiva se ejecutan las instrucciones del primer par de llaves. Si por lo contrario, resulta falsa, se ejecutarían las que se encuentran entre el par de llaves después del else. Cabe destacar que si la instrucción es una sola línea, no hacen falta las llaves.

switch(selector)
{
case 1: sentencias;
break;
case 2: sentencias;
break;
case n: sentencias;
break;
default: sentencias;
}
	switch(dia) {
	case 1 : n = “lunes”; break;
	case 2 : n = “martes”; break;
	case 3 : n = “miercoles”; break;
	case 4 : n = “jueves”; break;
	case 5 : n = “viernes”; break;
	case 6 : n = “sábado”; break;
	case 7 : n = “domingo”; break;
	default: “día no válido”;
	}
		Mediante la estructura de control switch se puede seleccionar una opción de acuerdo con una serie de casos (case). Esta selección se debe al valor que recibe el parámetro selector. Es similar a la estructura if, sin embargo aglutina de mejor manera las posibles sentencias que pueda ejecutar según el valor del parámetro decisor (selector)

ESTRUCTURA DE CONTROL	EJEMPLO	DESCRIPCIÓN

while(condición)
{
Grupo de sentencias
}
	while(a>b)
	{
	System.println(“ a > b”);
	}
		while establece un bucle o ciclo de ejecución que dependerá de la validez de la condición. Cuando la condición no se cumple, while termina la ejecución del grupo de sentencias.

do { Grupo de sentencias; }while(condición);	Do { System.println(“ a > b”); } while(a>b);	Es similar al while anterior, con la diferencia que la comprobación de la condición se realiza al final. En este contexto el grupo de sentencias se ejecuta al menos una vez.

for(exp1;exp2;exp3)
{
Grupo de sentencias
}
	for(i=0;i<10;i++)
	{
	System.println(i);
	}
		Mediante for se puede ejecutar un grupo de sentencias, de acuerdo con un valor inicial y un valor final. Se usa cuando se conoce de dónde y hasta dónde deben ejecutarse las sentencias.

Hasta aquí se han analizado algunos fundamentos de la programación en Java y también se estudió el entorno de NetBeans, por lo tanto se tienen elementos básicos para aventurarnos a realizar nuestros primeros programas.

Actividades para el lector

1.- Desarrolle programas en los que aplique el uso del while y for para:

- Que el usuario vea la tabla de multiplicar de un número cualquiera.
- Determinar el factorial de un número cualquiera.
- Seleccionar la estación del año favorita del usuario y que muestre un mensaje alusivo a la misma.

NUESTRO PRIMER PROGRAMA EN NETBEANS

Vamos a iniciar nuestro primer programa NetBeans, para lo cual se utiliza el siguiente procedimiento:

1. Proceda abriendo Netbeans 7.1.
2. Una vez ahí seleccione Archivo, Proyecto nuevo… En la pantalla que se muestra a continuación seleccione un proyecto de tipo Java, tal como se aprecia en la figura 2.1.

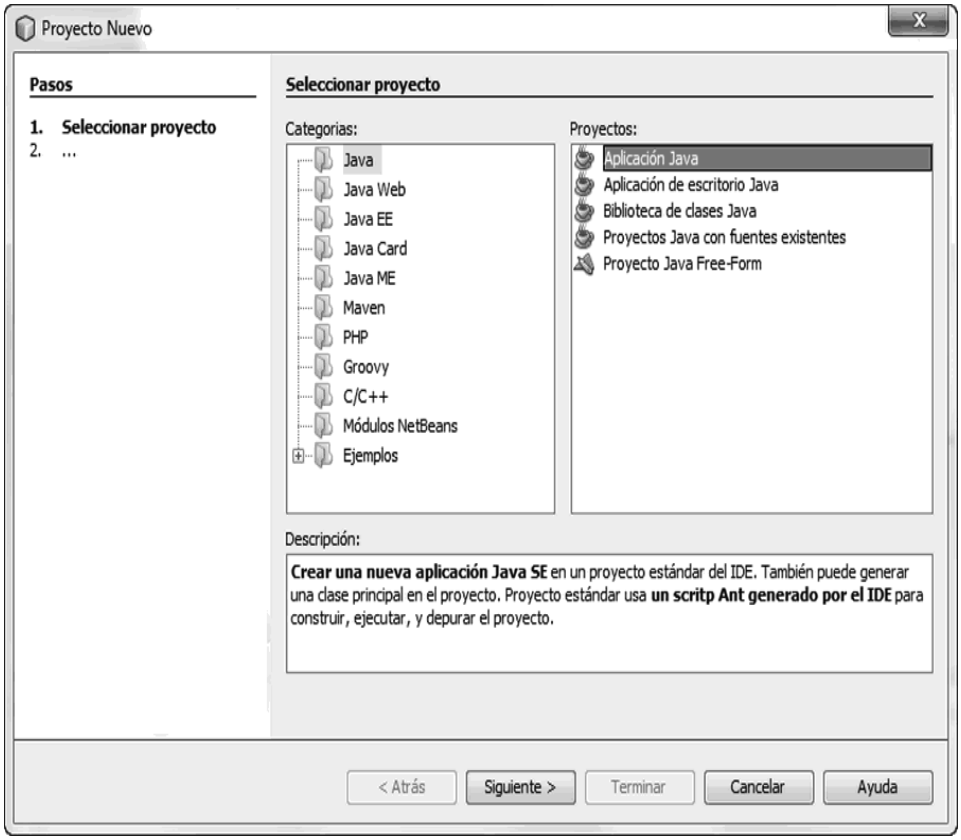


Figura 2.1 Mi primer programa en NetBeans.

3. Ponga un nombre alusivo como *miPrimerProgramaJava.* Guárdelo en un lugar seguro y pulse la tecla *Terminar*.
4. Tendrá un editor de código fuente donde podrá escribir las funciones y llamarlas desde el main del proyecto. Una imagen similar a la figura 2.2 sería del código terminado para el primer ejemplo.

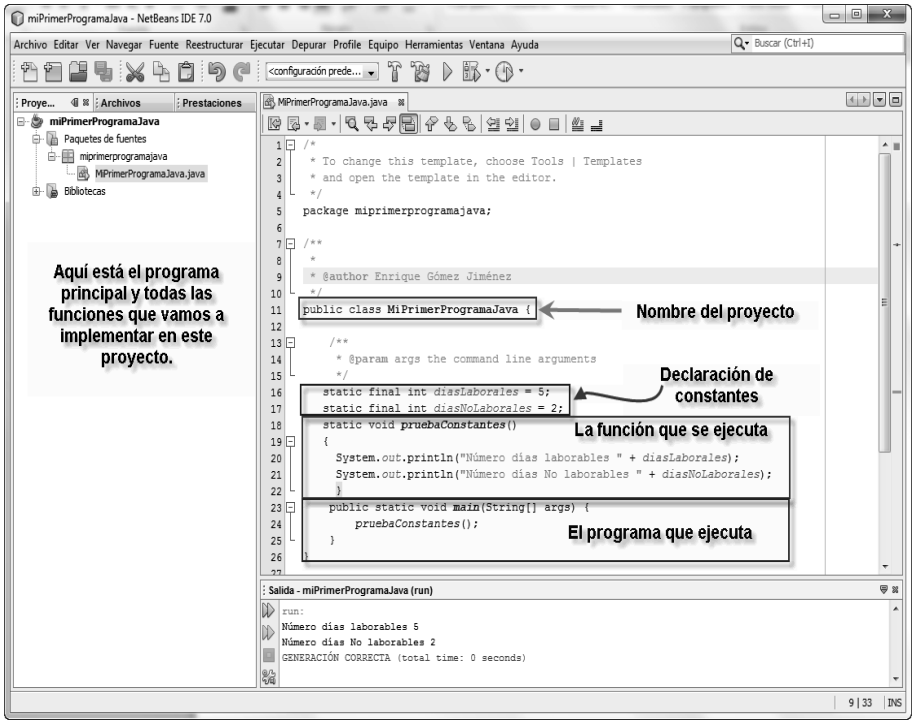


Figura 2.2 Mi primer programa en NetBeans, listo.

5. El código final de este primer ejercicio se muestra a continuación

EJERCICIO: PRIMER EJERCICIO EN NETBEANS 7.1
package miprimerprogramajava;
public class MiPrimerProgramaJava {
static final int diasLaborales = 5;
static final int diasNoLaborales = 2;
static void pruebaConstantes(){
System.out.println("Número días laborables " + diasLaborales);
System.out.println("Número días No laborables " + diasNoLaborales);
}
public static void main(String[] args) {
pruebaConstantes();
}
}

6. Puede ejecutar este programa y podrá observar la salida. Esta salida se muestra en la figura 2.2
7. Para seguir demostrando el uso de aplicaciones Java con NetBeans, escriba otro procedimiento, el cual se lista a continuación.

EJERCICIO: AGREGAR OTRA FUNCIÓN A NUESTRO EJEMPLO
public class MiPrimerProgramaJava {
static final double imp=0.13 ;
static double impuesto(double monto)
{
return monto * imp;
}
public static void main(String[] args) {
System.out.println("Impuesto de 1000: " + String.valueOf(impuesto(1000)));
}}

8. Observe la línea

   System.out.println("Impuesto de 1000: " + String.valueOf(impuesto(1000)));

   básicamente se está invocando al procedimiento impuesto, enviándole el parámetro 1000 y posteriormente se mostrará en la consola el resultado de la ejecución de la función.
9. Ahora se mostrará el uso de otra estructura de control: switch en Java. El código para este programa se lista a continuación:

EJERCICIO: AGREGAR OTRA FUNCIÓN A NUESTRO EJEMPLO
static String dia(int ndia) {
String nomdia="dia!";
switch(ndia) {
case 1 : nomdia = "lunes"; break;
case 2 : nomdia = "martes"; break;
case 3 : nomdia = "miercoles"; break;
case 4 : nomdia = "jueves"; break;
case 5 : nomdia = "viernes"; break;
case 6 : nomdia = "sábado"; break;
case 7 : nomdia = "domingo"; break;
}

EJERCICIO: AGREGAR OTRA FUNCIÓN A NUESTRO EJEMPLO
return (nomdia);
}
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
System.out.print("Digite un numero de dia : ");
int Dia = scanner.nextInt();
System.out.print("Digite un numero de dia : ");
System.out.println("El dia es "+dia(Dia)); }

10. El programa que se realizará a continuación implementa el uso de un vector de 5 elementos. La funcionalidad básica consiste en cargar el vector y desplegarlo. También determina cuál es el valor mayor alojado en el mismo. El código es el siguiente:

EJERCICIO: AGREGAR OTRA FUNCIÓN A NUESTRO EJEMPLO
static void funcionVector() {
int vector[] = new int[5]; //se declara un vector de 5 elementos
int i;
Scanner scanner = new Scanner(System.in);
for(i=0;i<5;i++) {
System.out.println("Digite el valor para la posición "+String.valueOf(i)+" : ");
vector[i] = scanner.nextInt(); }
//desplegando el contenido del vector
for(i=0;i<5;i++)
System.out.println("i : "+String.valueOf(i)+" es "+String.valueOf(vector[i]));
//obteniendo el mayor de los valores del vector.
int mayor = vector[0];
for(i=1;i<5;i++)
if(vector[i]>mayor) mayor=vector[i];
System.out.println("El valor mayor vector es : "+String.valueOf(mayor)); }
public static void main(String[] args) {
funcionVector();
}

La línea que resulta un poco extraña es Scanner scanner = new Scanner(System.in); Sin embargo, no tiene nada de extraño. Simplemente se utiliza para procesar una lectura desde teclado en una aplicación de consola en Java.

11. Ya se ha implementado algunos programas con la teoría desarrollada durante todo el capítulo. Lo que resta es mostrar tres elementos: el icono para ejecutar una aplicacion, el área del código del programa y la ventana de resultados. En la figura 2.3 se muestran estos tres elementos.

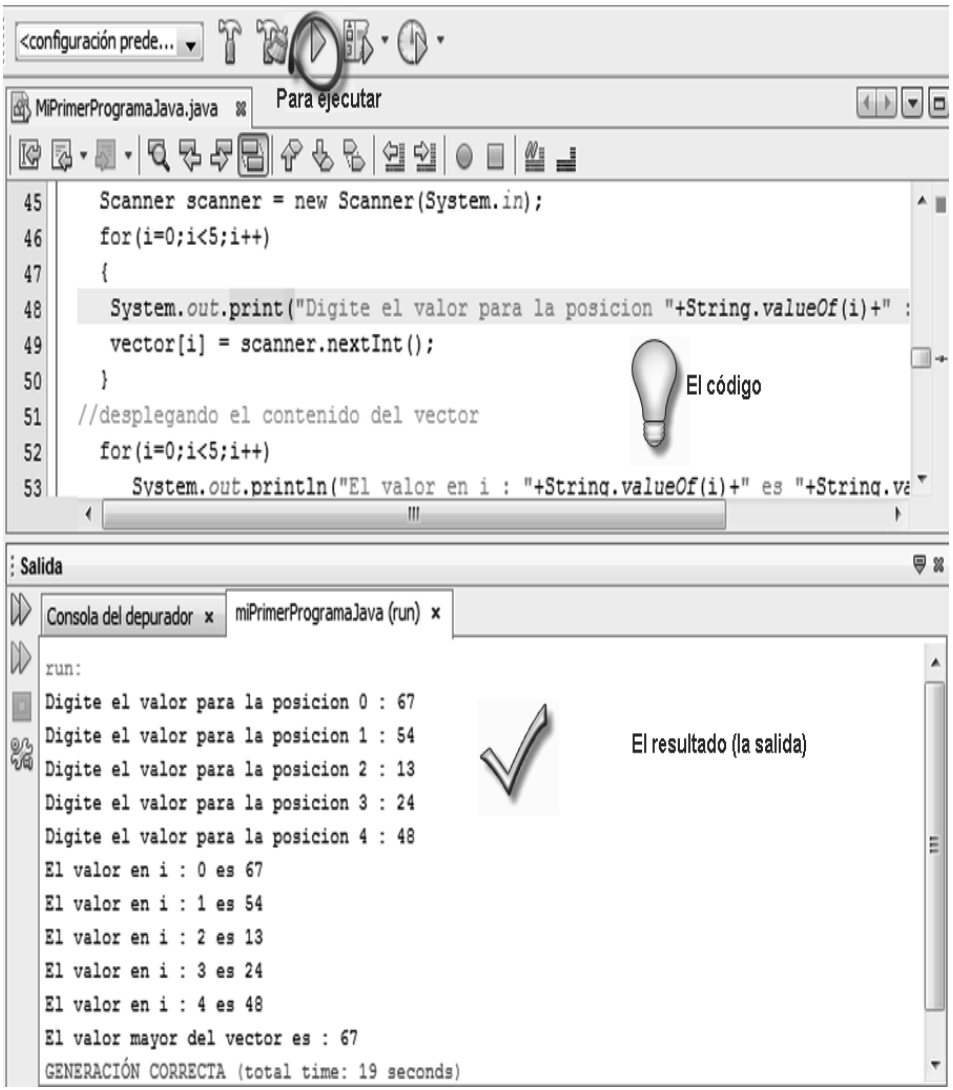


Figura 2.3 Cómo ejecutar y ver los resultados de un programa.

Actividades para el lector

Resuelva los ejercicios propuestos en:
http://sites.google.com/site/tutoriasdeingenieria/lenguajes-de-programacion/operadores-y-operaciones-basicas-de-java-netbeans

Con esto se da por concluido este capítulo. Se debe entender que constituyen sólo los fundamentos de la programación en Java, como una base para conocer el lenguaje de programación NetBeans. Por supuesto, a través de los demás capítulos se irá profundizando en el tema y descubriendo otros.

RESUMEN

En este capitulo hemos desarrollado los fundamentos de la programación en Java. Estos fundamentos constituyen la base para el desarrollo de los temas de programación con NetBeans. Básicamente se trataron los siguientes temas:

1. Cómo declarar y asignar valores a constantes en Java.
2. Las variables implementadas en NetBeans y los tipos de datos sobre los que se declaran. Se tratan, entre ellos, datos primitivos y compuestos.
3. Los operadores aritméticos, lógicos y relacionales existentes en NetBeans.
4. Las estructuras de control que se utilizan en NetBeans.

Autoevaluación

1. Defina qué es una constante.

2. Defina con sus propias palabras qué es una variable.

3. ¿Qué dato entero es de 64 bits?

4. ¿Qué dato de tipo coma flotante es de 32 bits?

5. ¿Qué resultados son posibles con una variable de tipo booleana?

6. ¿Cuál es la forma para declarar la variable cadena ciudad en Java?

7. Defina qué es un vector.

8. ¿Qué es una matriz?

9. ¿Qué tipos de operadores se implementan en Java?

10. ¿Para qué se utiliza la instrucción Scanner?

EVIDENCIA

☐ Desarrolló un programa que implementa al menos dos de las estructuras complejas que se indicaron en este capítulo.

☐ Desarrolló programas en los que aplica el uso del while y for para: Determinar el factorial de un número cualquiera; generarar la tabla de multiplicar de un número cualquiera; seleccionar la estación del año favorita del usuario.

☐ Resolvió los ejercicios propuestos en: *http://sites.google.com/site/tutoriasdeingenieria/lenguajes-de-programacion/operadores-y-operaciones-basicas-de-java-netbeans* .

REFERENCIAS

Bibliografía

Ceballos, Fco. Javier, (2011). *Java 2: Curso de Programación*, 4a. ed., Alfaomega, México.

LÓPEZ, José (2011*), Domine JavaScript* - 3ª ed., Alfaomega, México.

Martín, Antonio, (2010). *Programador certificado Java 2. Curso práctico*, 3ª ed., Alfaomega, México.

ORÓS, Juan (2011), *Diseño de páginas web con XHTML, JavaScript y CSS* 3ª ed., Alfaomega, México.

Rozannski, Uwe, (2010). *Enterprise Javabeans 3.0. Con Eclipse Y JBoss*. Alfaomega, México.

Sznajdleder, Pablo, (2010) *Java a fondo. Estudio del lenguaje y desarrollo de aplicaciones.* Alfaomega, México.

Páginas Web recomendadas

1. Zarsa.usal.es (1999) Guía de Iniciación al Lenguaje JAVA. Obtenido el 15 de octubre del 2011 desde http://zarza.usal.es/~fgarcia/doc/tuto2/II_2.htm
2. usuarios.multimania.es (2002) Tipos de datos en Java. Obtenido el 15 de octubre del 2011 desde http://usuarios.multimania.es/jjmurias/java/Tipos%20de%20datos.htm

3. PC World Digital (2004) Programación en Java (II) Tipos de datos y operadores. Obtenido el 15 de octubre del 2011 desde http://www.idg.es/pcworld/Programacion-en-Java-_II__Tipos-de-datos-y-operado/art161638.htm
4. Dccia.ua.es (2011) Colecciones en Java. Obtenido el 16 de octubre del 2011 desde http://www.dccia.ua.es/dccia/inf/asignaturas/RG/pdf/colecciones-java.pdf
5. Docstoc (2011) Colecciones de Java y excepciones. Obtenido el 17 de octubre del 2011 desde http://www.docstoc.com/docs/22843704/Colecciones-de-Java-y excepciones
6. slideshare.net (2009) Colecciones. Obtenido el 17 de octubre del 2011 desde http://www.slideshare.net/javi2401/colecciones-en-java-presentation sites.google.com (2011)
7. Operadores aritméticos, lectura, conversión y escritura Java - Netbeans. Obtenido el 17 de octubre del 2011 desde http://sites.google.com/site/tutoriasdeingenieria/lenguajes-de-programacion/operadores-y-operaciones-basicas-de-java-netbeans.

Respuestas a la autoevaluación

1. Una constante se puede definir como un valor que permanece inalterable en el tiempo.

2. Una variable es un valor que cambia durante la ejecución de un programa.

3. long

4. float

5. verdadero o falso

6. String ciudad = new String;

7. Un vector es una estructura de datos con varias posiciones de memoria donde se pueden almacenar datos.

8. Es una colección de celdas en forma bidimensional (filas y columnas) donde se pueden almacenar valores de un mismo tipo.

9. Aritméticos, relacionales y lógicos.

10. Para procesar una lectura desde teclado en una aplicación consola en Java.

Programación orientada a objetos con NetBeans 7.1

3

Reflexione y responda las siguientes preguntas

¿A qué se denomina paradigma de programación?

¿Es la programación a objetos el nuevo paradigma del desarrollo de software?

¿Qué es la reutilización de software? ¿Posibilita la programación orientada a objetos la reutilización de código?

¿Es Java un lenguaje de programación orientado a objetos?

Contenido

EXPECTATIVA

La programación orientada a objetos es un paradigma de programación vigente en nuestros días. Mediante este paradigma se pueden diseñar sistemas y aplicaciones informáticas que permiten la especificación funcional y la reutilización de códigos y componentes. Esto significa un alto grado de cohesión y más acoplamiento de las aplicaciones a las reglas de negocios existentes en nuestros días. Por ende, conocer este paradigma se convierte en una necesidad obligada para todos los que se dedican al desarrollo de aplicaciones informáticas.

Después de estudiar este capítulo, el lector será capaz de

- Conocer los componentes de la programación orientada a objetos para conceptualizarla dentro del contexto del desarrollo de aplicaciones informáticas.
- Comprender el paradigma de la programación orientada a objetos y la forma de implementarla con Netbeans 7.1
- Desarrollar aplicaciones en Netbeans 7.1 que utilicen la programación orientada a objetos.
- Aplicar la reutilización de código y componentes en el desarrollo de aplicaciones orientadas a objetos con Netbeans 7.1

INTRODUCCIÓN

En nuestros días el desarrollo de software es una actividad que genera millones de dólares. Esos ingresos se obtienen del uso del software y por la producción del software. En el primer caso, se sabe que la vida comercial y financiera se ve comprometida con el uso intensivo del software para la realización de transacciones y/o generación de información. En el segundo caso, empresas desarrolladoras se enfrascan en una lucha feroz por ser parte del mercado de la venta de aplicaciones comerciales, financieras, juegos, entre otros. Por ello es necesario contar con tecnología de punta y las mejores técnicas para el desarrollo rápido de aplicaciones y de la mejor calidad. El paradigma de orientación a objetos permite muchas de estas necesidades.

Los paradigmas

Un paradigma se conceptualiza como un modelo o patrón de cualquier disciplina científica o de otro contexto. Puede concebirse como un conjunto de ideas, creencias, pensamientos, técnicas, métodos, entre otros elementos. Ese conjunto es adoptado por grupos de personas comprometidos e identificados con la misma para la resolución de un problema relacionada con el tema.

Paradigma de programación

Un paradigma de programación se concibe como una propuesta tecnológica que un grupo de personas adopta para la resolución de un problema claramente delimitado. Un paradigma de programación supone la formulación de técnicas, lenguajes, métodos o cualquier otro elemento que permita el cambio radical o parcial de otro paradigma anterior. Por ejemplo, el paradigma de la orientación a objetos vino a sustituir la programación tradicional basada en una estructura procedimental.

Tipos de paradigmas de programación

A través de la historia de la computación han existido varios paradigmas de programación. Al día de hoy la programación orientada a objetos es la que se encuentra vigente y la que más compañías de software utilizan en el mundo.

Los tipos de paradigmas se han gestado gracias a la investigación tecnológica, el aporte de usuarios y las experiencias vividas tanto en el desarrollo de software como también en la funcionalidad del mismo.

La figura 3.1 muestra los tipos de paradigmas que han existido hasta el momento.

Figura 3.1 Tipos de paradigmas de programación.

PROGRAMACIÓN ORIENTADA A OBJETOS

Como se mencionó, la programación orientada a objetos utiliza objetos y las interacciones que se pueden establecer entre ellos. Incluye varias técnicas entre las que se destacan la herencia, el abstraccionismo, el polimorfismo y el encapsulamiento. Estas técnicas utilizadas en forma adecuada y combinada permiten diseñar y desarrollar sistemas informáticos de calidad en cuanto a su arquitectura y desempeño. En la actualidad existen muchos lenguajes que soportan la programación orientada a objetos y es raro encontrar uno que no lo haga. Muchas suites de desarrollo, además de la herramienta de creación de la aplicación, ofrecen utilitarios que permiten el diseño conceptual del mismo. Entre estos utilitarios se encuentran diseñadores de diagramas UML que implementan diseñadores de código e interfaces que facilitan el desarrollo de aplicaciones.

Introducción a la programación orientada a objetos

Los objetos son entidades que comparten las mismas características. Se agrupan en clases funcionales y pueden ser utilizados mediante instanciación. Conceptualmente, un objeto posee dos contextos: un estado y un comportamiento. El estado se refiere a la configuración inicial de todos y cada uno de los atributos de un objeto. El comportamiento son los cambios que se producen en los estados iniciales de un objeto o la lógica que se ejecuta en un método de la clase instanciada.

La identidad del objeto es otra característica presente en los objetos y se refiere a la identificación unívoca del mismo dentro de la colección de objetos de la aplicación. Finalmente, el tiempo de vida del objeto alude a la duración que tiene de vida dentro de la vida general de la aplicacion.

La figura 3.2 muestra las características básicas de un objeto.

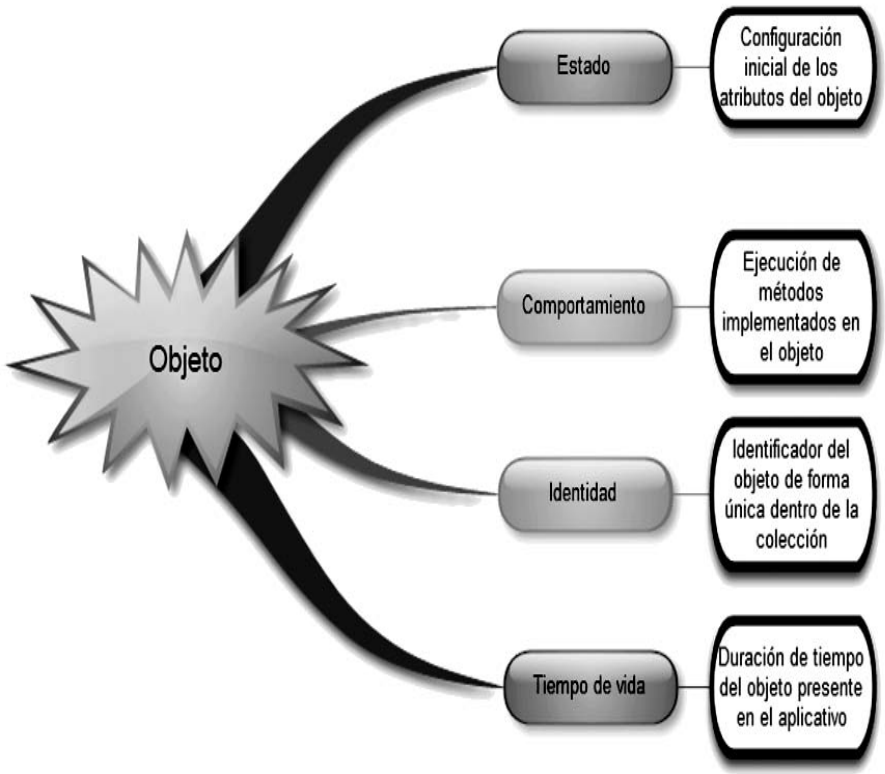


Figura 3.2 Características básicas de un objeto.

Conceptos básicos de la programación orientada a objetos

Para comprender el amplio mundo de la programación orientada a objetos, es necesario definir algunos conceptos importantes. Estos conceptos importantes se detallan en la tabla 3.1.

Tabla 3.1 Conceptos básicos de la programación orientada a objetos.

CONCEPTO	DESCRIPCIÓN
Clase	Una clase conjunta elementos que comparten las mismas propiedades y métodos. La instanciación de una clase crea lo que se denomina un objeto. Existen superclases y subclases. Con ello se crea lo que se llama jerarquía de clases, donde la clase padre hereda a las clases hijas.
Herencia	Es la facilidad mediante la cual una clase hereda las propiedades y métodos públicos de otra clase. La herencia puede ser simple o múltiple. Es simple cuando una clase hereda solamente de otra clase superior (se convierte una relación subclase – superclase) mientras que la herencia múltiple es cuando la clase hereda de 2 o más clases superiores. En Java y .NET sólo existe la herencia simple.
Objeto	Es una entidad instanciada de una clase provista de propiedades o atributos (datos) y de comportamiento o funcionalidad (métodos) que básicamente responden a eventos. El objeto puede instanciarse de una clase directamente o de una clase derivada la cual hereda de otra.
Método	Constituye la lógica de un objeto. Algoritmo que se implementa para realizar una determinada operación sobre los atributos de la clase o de una clase derivada.
Evento	Es el encargado de que un método se ejecute. Entre los eventos programáticos más comunes se tiene el clic que se produce sobre un botón, el keypress sobre un objeto texto o change que se produce sobre dropdown.
Mensaje	Constituye la comunicación que se establece entre objetos para intercambiar parámetros sean de entrada, salida o ambos. Mediante parámetros se establece un esquema de colaboración entre objetos.
Propiedad	Se le denomina también atributo o dato de la clase. Constituye el contenedor de un valor de un tipo de dato determinado. Un método puede variar ese contenido.

También se debe agregar cuáles son las propiedades principales de la programación orientada a objetos. La tabla 3.2 muestra algunas de estas propiedades.

Tabla 3.2 Conceptos básicos de la programación orientada a objetos.

CONCEPTO	DESCRIPCIÓN
Abstracción	Representa las características esenciales de un objeto, sin tener la preocupación de las demás características que no lo son. Se preocupa más de los detalles importantes de la implementación de un objeto, separando el comportamiento esencial. Abstracción es describir una entidad del mundo real, por compleja que sea, y utilizar dicha descripción en un programa. Ejemplo de abstraccionismo es la de un motor. Este motor puede ser muy genérico y comportarse de tal forma cuando es implementado por un motor a explosión, eléctrico o a vapor.
Encapsulamiento	Significa la conjunción de elementos que abstractamente se consideran pertenecientes a una entidad, con mayor cohesión entre los componentes de la misma. No se debe confundir con el principio de ocultamiento.
Modularidad	Tiene como objetivo descomponer una aplicación en otras más pequeñas (módulos) que sea lo más acoplada posible a sí misma (independiente)
Ocultación	Es el sistema de aislamiento generalmente de las propiedades de un objeto para proteger su integridad y el acceso desautorizado que pueda modificar su contenido. Sólo los métodos internos o pertenecientes a la clase pueden modificar dichas propiedades. Se puede afirmar acordes con la Ley de Demeter.
Polimorfismo	Se refiere a la capacidad de los objetos de comportarse de acuerdo a la funcionalidad requerida. Es decir, establecer diferentes comportamientos para los métodos, por ejemplo, del objeto, que implementan diversas funcionalidades de acuerdo con parámetros recibidos a través de los mismos. Por ejemplo, se puede realizar una suma de enteros que devuelva enteros y reciba parámetros enteros, o una suma de valores flotantes donde se reciban parámetros flotantes y devuelva un valor flotante. Ambos métodos pueden llamarse igual pero se diferencian por los parámetros que reciben.

Clases abstractas

Una clase abstracta implementa la estructura genérica de la funcionalidad que deberá ser implementada en una clase derivada. Una clase abstracta en realidad no puede usarse, es decir, no se puede instanciar en un objeto para ser utilizada. Más bien se deriva en otra clase y sobre ésta es que se crea la instanciación.

Una clase abstracta se puede crear o definir cuando se necesita englobar objetos de tipo diferente y se quiere implementar polimorfismo. Supóngase que se requiere crear un sistema que sea lo suficientemente genérico (parametrizable) pero que establezca ALFAOMEGA los métodos que obligatoriamente deben realizarse.

Aquí es donde entran en juego las clases abstractas. La reutilización de código en este contexto es claramente demostrable.

A continuación se muestra un ejemplo de implementación de clases abstractas.

1. Inicie abriendo NetBeans.
2. Proceda a crear un nuevo proyecto Java, tal y como se mostró en las figuras 2.1 y 2.2 del capítulo anterior.
3. El nombre del proyecto será *ejemploTercero*.
4. Una vez que se ha creado el proyecto pulse con el botón derecho sobre el nombre del mismo (ejemploTercero) y seleccione *Nuevo* y *Clase Java*. Nombre esta clase como vehiculo y escriba el código siguiente:

EJERCICIO: CREACIÓN DE UNA CLASE ABSTRACTA LLAMADA VEHICULO
public abstract class vehiculo {
protected int modelo;
protected String color;
protected String marca;
protected double precio;

Scanner Lector = new Scanner(System.in); //para procesar lecturas de
//teclado a usar en clases
//derivadas

public vehiculo (int modelo, String color, String marca, double precio)
{
this.modelo = modelo;
this.color = color;
this.marca = marca;
this.precio = precio;
}
public vehiculo(); //declara un constructor vacío.
{}
public abstract void registrarVehiculo();}

Como se observa se ha declarado la clase pública *vehiculo*, se ha antepuesto la sentencia *abstract* con lo cual se crea como una clase abstracta. Se declaran los atributos modelo, color, marca y precio de tipo protected. ¿Qué es un atributo de

tipo protected? Son atributos que son accesibles desde la clase donde se definen, así como en las clases que se derivan de ella (subclases).

Observe la siguiente línea de código:

```
public vehiculo (int modelo, String color, String marca, double precio)
```

Esta línea es el constructor de la clase (observe que la clase se llama igual: vehiculo). El constructor de la clase recibe los parámetros modelo, color, marca y precio. Pero observe la línea siguiente:

```
this.modelo = modelo;
```

Si es observador podrá notar que en la clase existe un atributo con nombre modelo y que el constructor recibe un parámetro con el mismo nombre. Entonces, ¿qué función cumple el this? Pues this es una referencia al objeto que está ejecutando el método. Es decir, le está diciendo que el atributo modelo de la clase es asignado con el valor del parámetro modelo. Así pasa con el resto de atributos que son cargados con parámetros recibidos en el constructor y que tienen el mismo nombre.

Observe ahora la línea:

```
public abstract void registrarVehiculo();
```

En esta línea se está declarando una función abstracta denominada *registrarVehiculo()* pero observe que no está implementado el código de dicha función. Eso significa que en la derivación de otra clase será donde se escriba el código respectivo.

5. Ahora crearemos una nueva clase, que será donde se implemente el código del método abstracto; se reutilizaran los atributos protegidos de la clase abstracta. Proceda como en el punto 4 creando una nueva clase de nombre *automovil*.

   Cuando cree la nueva clase debe cambiar el encabezado de la clase:

   ```
   public class automovil {
   ```

   por:

   ```
   public class automovil extends vehiculo {
   ```

   La primera línea lo que hace es crear una clase normal, denominada automovil. La segunda línea lo que hace es crear la clase automovil e indicarle que herede (mediante la palabra reservada extends) de la clase abstracta. Una vez que escriba eso NetBeans se dará cuenta que está implementando una clase abstracta, por ende le solicitará que implemente todos los métodos abstractos de dicha clase. La figura 3.3 muestra esta situación.

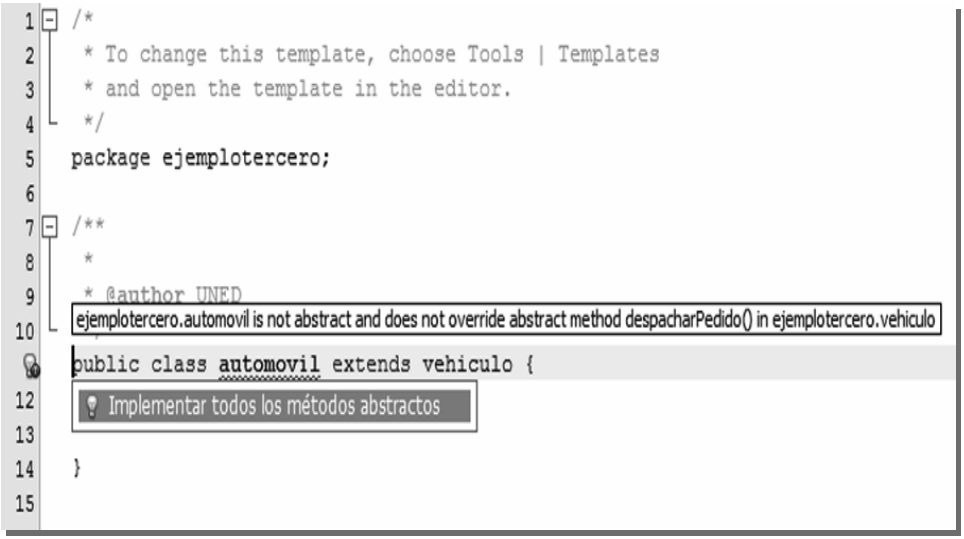


Figura 3.3 Implementación de los métodos abstractos de la clase abstracta.

6. Debe pulsar sobre el ícono de la lamparita para que los métodos abstractos de la clase aparezcan en el código de esta nueva clase.

   El código para esta clase es como el que se indica a continuación.

EJERCICIO: CREACIÓN DE LA CLASE AUTOMÓVIL QUE REESCRIBE EL MÉTODO ABSTRACTO
public class automovil extends vehiculo {
@Override
public void registrarVehiculo() {
String estilo; //coupe, sedan, compacto…
System.out.print("Digite el modelo del automóvil: ");
modelo = Lector.nextInt();
System.out.print("Digite el color del automóvil: ");
color = Lector.next();
System.out.print("Digite el precio del automóvil: ");
precio = Lector.nextInt(); }
}

7. Ahora crearemos otra clase, denominada *bicicleta*, con el siguiente código:

EJERCICIO: CREACIÓN DE LA CLASE BICICLETA QUE IMPLEMENTA LA FUNCIONALIDAD DE LA CLASE ABSTRACTA

```
public class bicicleta extends vehiculo {
    @Override
public void registrarVehiculo() {
    System.out.print("Digite el color de la bicicleta: ");
    color  = Lector.next();
    System.out.print("Digite el precio de la bicicleta: ");
precio = Lector.nextInt();  }
    }
```

8. Como comentarios a este código se puede agregar que existe la palabra @Override, lo cual es una anotación que significa que está sobrescribiendo un método de la clase padre (es decir, está sobrescribiendo el método *registrarVehiculo()* en *bicicleta* que es implementada desde la clase padre *vehiculo*). Considérese otro ejemplo similar.
   Suponga que necesita escribir una clase que contenga una operación llamada cocinar. Sin embargo, también sabe que con dicha operación desea que se procese cualquier tipo de alimento. Atributos genéricos podrían ser tiempoCoccion y temperaturaCoccion. En un determinado caso podría ser que se cree una clase comidaChina que requiere preparar un determinado tipo de alimento, o la clase comidaCasera que es otro tipo. Ambos necesitan algo común que es cocinar y cocinar requiere los atributos tiempoCoccion y temperaturaCoccion. En dicho caso la clase que implementará la operación cocinar podríamos declararla como abstracta con la operación de tipo abstracto (vacio) y las clases comidaChina y comidaCasera implementarán el método según sus necesidades.

Actividades para el lector

Desarrolle un programa Java utilizando NetBeans, que implemente la solución del problema de comida china y comida casera.

9. Ahora vamos a implementar la clase *ejemplotercero* que es la que tiene el main() que permite ejecutar los programas del proyecto.

EJERCICIO: CREACIÓN DE LA CLASE BICICLETA QUE IMPLEMENTA LA FUNCIONALIDAD DE LA CLASE ABSTRACTA
public class EjemploTercero {
public static void main(String[] args) {
bicicleta Bicicleta = new bicicleta();
Bicicleta.registrarVehiculo();
automovil Automovil = new automovil();
Automovil.registrarVehiculo();
}
}

Como podrá observar en el main se instancia las clases *Bicicleta* y *Automovil* de las clases *bicicleta* y *automovil*, respectivamente. Hay que recordar que tanto *bicicleta* como *automovil* implementan el método abstracto *registrarVehiculo()* que se creó en la clase abstracta *vehiculo*.

10. En la figura 3.4 se muestra la ejecución típica de este ejemplo.

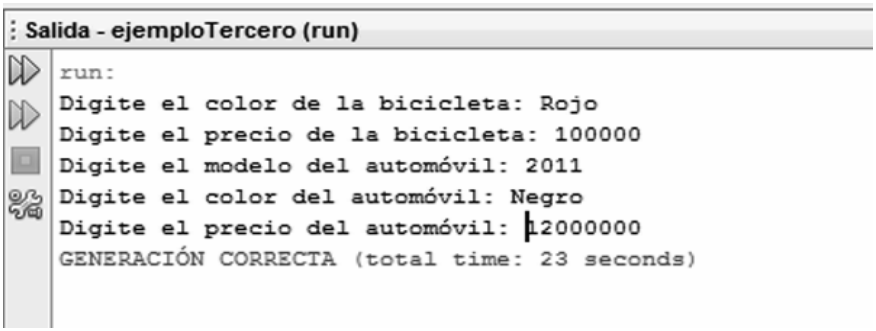


Figura 3.4 Ejecución de la nuestro ejemplo de clase abstracta.

En el ejemplo anterior se ha tratado indirectamente el concepto de herencia. En Java sólo se dispone de la herencia simple y no como en C++ que puede utilizar la herencia múltiple. Sin embargo, debe considerar que la mayoría de lenguajes de programación (si no es que todos) implementan la herencia simple nada más.

La instrucción:

```
public class automovil extends vehiculo {
```

Hace uso de la herencia en Java. En este sentido, extends significa “hereda”. Por tanto, en la línea anterior se establece que la clase automovil hereda de la clase vehiculo. En este sentido se establecen las siguientes reglas (en el caso que vehiculo no fuera una clase abstracta):

- automovil hereda de vehiculo todos sus miembros que sean públicos. Por regla general sólo los métodos se declararían públicos, por tanto, automovil heredaría los métodos públicos de vehiculo.
- Los atributos se declaran de tipo privado “private”, por lo que la clase que hereda no tiene acceso a estos atributos, sino que lo hace a través del método público heredado.
- La instanciación se realiza sobre la clase automovil para que herede de vehiculo. Recuerde que no se puede utilizar directamente una clase en un programa.

Actividades para el lector

Investigue y documente para qué se utiliza la instrucción *super*.

Interfaces

Mediante interfaces puede cubrir la necesidad de que una clase herede de otras clases (al menos dos). Dado que en Java no existe el uso de la herencia múltiple, entonces se tiene que utilizar una interface como enlace entre una subclase y dos superclases. Considere el diagrama de clases de la figura 3.5.

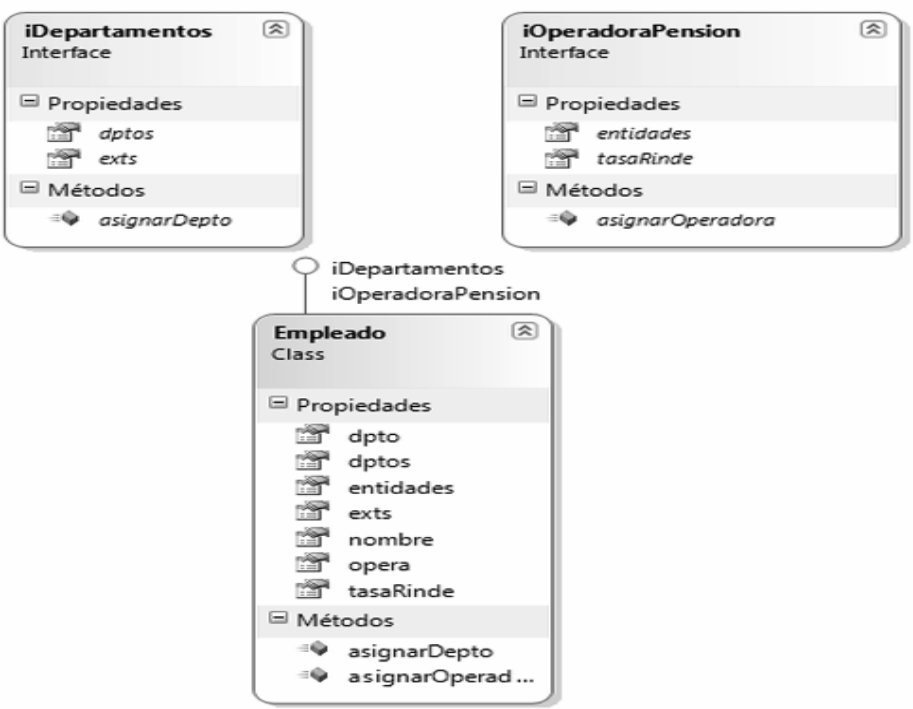


Figura 3.5 Diagrama de clases simulando herencia múltiple mediante interfaces.

Este diagrama de clases demuestra la necesidad de asociar una clase empleado con las clases departamentos y operadoras de pensión. Como Java no puede implementar la herencia múltiple, tendrá que acudir a una interface para que las relacione. Es por ello que primero se crean dos interfaces con la declaratoria de los atributos y métodos que requiere y posteriormente las hereda de las interfaces.

11. Desarrolle el proyecto procediendo como se indicó en los puntos 2.1 y 2.2 del capítulo 2. En lugar de crear una clase seleccione una interface que se denomine *iDepartamentos*. El código que escribirá en el editor es el siguiente:

EJERCICIO: CREACIÓN DE LA INTERFACE IDEPARTAMENTOS
public interface iDepartamentos {
//se crea un vector de cadena y se inicializa con sus respectivos valores public static final String dptos[] = {"Recursos Humanos","Informática", "Financiero","Contabilidad","Ventas"};
//se inicializa un vector de enteros con sus respectivos valores public static final int exts[] = {123,345,324,125,423};
//se crear un método vacío (sin implementar) void asignarDepto();
}

12. Ahora proceda a crear la interface iOperadoraPension, procediendo de la misma manera que en el punto anterior.

    El código de esta interface se muestra a continuación:

EJERCICIO: CREACIÓN DE LA INTERFACE IOPERADORAPENSION
public interface iOperadoraPension {
//se crea un vector de cadena con sus respectivos valores public static final String entidades[] = {"Banco de Pensiones", "Pensiones Costa Rica", "Banco Municipal", "Sistema de Pensiones","Pensiones Acme"};
//se crea un vector de tipo double con sus respectivos valores public static final double tasaRinde[] = {0.12,0.13,0.12,0.12,0.12};
void asignarOperadora();
}

13. Ahora se implementan estas interfaces en la clase *Empleado*. Para ello escriba el siguiente código:

EJERCICIO: CREACIÓN DE LA CLASE EMPLEADO

```
import java.util.Scanner;
//implementación de la clase Empleado con las interfaces antes creadas.
public class Empleado implements iDepartamentos,iOperadoraPension{
int i;
//se crea una variable tipo Scanner para realizar lecturas desde teclado
Scanner Lector = new Scanner(System.in);
@Override  //para implementar el método asignarDepto
public void asignarDepto() {
int dpto,ext;
String nombre;
System.out.print("Escriba Nombre del empleado : ");
nombre = Lector.next();  //se lee de teclado el contenido para nombre
for(i=0;i<dptos.length;i++) //se recorre el vector dptos.
  System.out.println(i+1+" "+ dptos[i]);    //se muestran valores del vector
  System.out.print("Seleccione Departamento : ");
  dpto = Lector.nextInt(); //lee por teclado una variable de tipo entera
  System.out.println("Departamento seleccionado : "+dptos[dpto-1]);
for(i=0;i<exts.length;i++) //se recorre el vector de extensiones
  System.out.println(i+1+" "+ exts[i]); //despliega cada valor del vector
  System.out.print("Seleccione la Extensión : ");
  ext = Lector.nextInt();
  System.out.println("Extensión : "+exts[ext-1]);
        }
@Override
public void asignarOperadora() {
int opera,tasa;
for(i=0;i<entidades.length-1;i++)
  System.out.println(i+1+" "+ entidades[i]);
  System.out.print("Seleccione la Operadora : ");
  opera = Lector.nextInt();
  System.out.println("Departamento seleccionado : "+entidades[opera-1]);
for(i=0;i<tasaRinde.length-1;i++)
  System.out.println(i+1+" "+ tasaRinde[i]);
  System.out.print("Seleccione la tasa de rendimiento: ");
  tasa = Lector.nextInt();
    System.out.println("Extensión : "+tasaRinde[tasa-1]);
    }}
```

Observe cómo se implementan las interfaces *iDepartamentos* y *iOperadoraPension* a la clase *Empleado*.

14. Finalmente, implemente el uso de la clase que implementa las dos interfaces creando la instancia de la clase Empleado. El código en el programa principal (que contiene el main) es el siguiente:

EJERCICIO: PROGRAMANDO EL MAIN
public class EjemploTercero {
public static void main(String[] args) {
Empleado aux = new Empleado();
aux.asignarDepto();
aux.asignarOperadora();
}
}

15. Una corrida de la aplicación se muestra en la figura 3.6.

```
run:
Escriba Nombre del empleado : Enrique
1 Recursos Humanos
2 Informática
3 Financiero
4 Contabilidad
5 Ventas
Seleccione Departamento : 2
Departamento seleccionado : Informática
1 123
2 345
3 324
4 125
5 423
Seleccione la Extensión : 3
Extensión : 324
1 Banco de Pensiones
2 Pensiones Costa Rica
3 Banco Municipal
4 Sistema de Pensiones
Seleccione la Operadora : 4
Departamento seleccionado : Sistema de Pensiones
1 0.12
2 0.13
3 0.12
4 0.12
Seleccione la tasa de rendimiento: 2
Extensión : 0.13
GENERACIÓN CORRECTA (total time: 29 seconds)
```

Figura 3.6 Corrida típica de nuestra aplicación de interfaces.

Actividades para el lector

Desarrolle un programa Java utilizando NetBeans que implemente las interfaces necesarias para el caso de Animales, Peces y Aves.

Polimorfismo

El polimorfismo (llamado también upcasting) brinda la posibilidad de que una referencia a un objeto de una clase pueda utilizarse también en las clases derivadas de éstas. En otras palabras el polimorfismo se refiere a la capacitad de clases derivadas de utilizar un método en forma diferente. Por ejemplo, considere el caso de dos animales: *Pez* y *Ave*. Ambos derivarían de una superclase denominada *Animal*. Posiblemente, en la clase *Animal* se implementaría un método abstracto denominado *desplazarse*. Debe ser abstracto dado que las formas de moverse de ambos animales es diferente: el primero nada y el segundo vuela.

Existen dos tipos de polimorfismos: dinámico y estático. El polimorfismo dinámico (llamado también polimórfico) es el que no se incluye especificación alguna sobre el tipo de datos sobre el cual se opera. Por ende, se puede utilizar en todo tipo de datos que sea compatible. Al polimorfismo dinámico se le conoce como programación genérica. El polimorfismo estático (también denominado ad hoc) es donde los tipos de datos deben ser explícitos y declarados en forma individual antes de poder utilizarlos.

Suponga que tiene una clase que implementa tres métodos que tienen el mismo nombre pero que reciben parámetros de distinto tipo y devuelven resultados también diferentes. Proceda a crear el ejemplo:

1. Agregue una nueva clase al proyecto que está trabajando en este capítulo. Denomínelo *clsPolimorfismo*. El código de esta clase es el siguiente:

EJERCICIO: PROGRAMANDO LA CLASE CLSPOLIMORFISMO
public class clsPolimorfismo {
int sumar(int a, int b) {
return a+b; //retorna un valor de tipo entero
}
double sumar(double a, double b) {
return a+b; } //retorna un valor de tipo double
String sumar(String a, String b) {
return a+b; } //retorna un valor de tipo cadena
}

Observe que se tienen tres métodos que se llaman igual, con la diferencia de los tipos de datos que recibe como parámetro y la devolución de la función misma.

La llamada a estas funciones dependerá del tipo de dato que envía dicha llamada, es decir, si se envían datos de tipo cadena se ejecutará la función que recibe ese tipo de parámetros.

2. Y la clase que llama a ejecutar la funcionalidad de la clase (donde se encuentra el main) contiene el siguiente código:

EJERCICIO: PROGRAMANDO EL MAIN
public class EjemploTercero {
public static void main(String[] args) {
clsPolimorfismo aux = new clsPolimorfismo();
System.out.println("Suma de Enteros: "+ aux.sumar(2, 3));
System.out.println("Suma de Cadenas: "+ aux.sumar("Enrique", " Gomez"));
System.out.println("Suma de Doubles: "+ aux.sumar(2.5, 3.3)); }
}

3. La figura 3.7 muestra la salida de la ejecución de este programa.

```
run:
Suma de Enteros: 5
Suma de Cadenas: Enrique Gomez
Suma de Doubles: 5.8
GENERACIÓN CORRECTA (total time: 0 seconds)
```

Figura 3.7 Ejecución del ejemplo de polimorfismo.

Actividades para el lector

Desarrolle un programa Java que utilizando NetBeans abstraiga el pago de impuestos de un país A y de un país B, cuyos procedimientos son diferentes.

RESUMEN

En este capitulo se desarrollaron los fundamentos de la programación orientada a objetos en Java. Estos fundamentos constituyen la base para el desarrollo de los temas de programación con NetBeans. Básicamente se trataron los siguientes temas:

1. Definir los paradigmas existentes en nuestro medio.
2. Resumir los paradigmas existentes en la programación de sistemas computacionales.
3. Sintetizar los tipos de programación existentes.
4. Fundamentar los conceptos básicos de la programación orientada a objetos.
5. Desarrollar ejemplos prácticos y sencillos sobre clases abstractas, interfaces y polimorfismo.

Existe mucha bibliografía sobre el tema; para profundizar se recomienda consultar la que se sugiere al final del capítulo, así como manuales y bibliografía de uso libre.

Autoevaluación

1. Defina paradigma.
2. ¿Qué es un paradigma de programación?
3. Cite los tipos de paradigmas de programación que han existido hasta nuestros días.
4. Cite las características principales de un objeto en cuanto su contexto.
5. Cite las características básicas de un objeto.
6. Defina lo qué es un método.

EVIDENCIA

- ☐ Desarrolló un programa Java en el que utilizando NetBeans implemente las interfaces necesarias para el caso de Animales, Peces y Aves.
- ☐ Desarrolló un programa Java utilizando NetBeans, que implemente la solución del problema de comida china y comida casera.
- ☐ Investigó y documentó para qué se utiliza la instrucción *super*.
- ☐ Desarrolló un programa Java utilizando NetBeans, que abstraiga el pago de impuestos de un país A y de un país B, cuyos procedimientos son diferentes.

REFERENCIAS

Bibliografía

Arce, Fco. Javier, (2011). ActionScript 3.0 Aprenda a programar. Alfaomega, México.

Ceballos, Fco. Javier, (2004). *Programación orientada a objetos con C++*. 3a. ed., Alfaomega, México.

Ceballos, Fco. Javier, (2008). *Java 2. Interfaces gráficas y aplicaciones para internet*. 3a. ed., Alfaomega, México.

Ceballos, Fco. Javier, (2009). *Enciclopedia del lenguaje C++*. 2a. ed., Alfaomega, México.

Ceballos, Fco. Javier, (2011). *Java 2: Curso de programación*, 4a. ed., Alfaomega, México.

López, Leobardo, (2006) *Metodología de la programación orientada a objetos*. Alfaomega, México.

López, Leobardo; Ramírez, Felipe, (2011) *Lógica para computación*, Alfaomega, México.

Martín, Antonio, (2010). *Programador certificado Java 2. Curso práctico*, 3a. ed., Alfaomega, México.

Peñaloza, Ernesto, (2004) *Fundamentos de programación C/C++*. 4a. ed., Alfaomega, México.

Ramírez, Felipe, (2007) *Introducción a la programación: algoritmos y su implementación en VB.Net, C#, Java y C++*. 2a. ed., Alfaomega, México.

Rozannski, Uwe, (2010). *Enterprise Javabeans 3.0. Con Eclipse Y JBoss*. Alfaomega, México.

Sznajdleder, Pablo, (2010) *Java a fondo. Estudio del lenguaje y desarrollo de aplicaciones.* Alfaomega, México.

Páginas Web recomendadas

1. sc.ehu.es (2000) Conceptos básicos de la programación orientada a objetos. Obtenido el 22 de octubre del 2011 desde http://www.sc.ehu.es/sbweb/fisica/cursoJava/fundamentos/clases1/clases.htm
2. Blog de Informática, Arquitectura, Diseño (2008) @override. Obtenido el 22 de octubre del 2011 desde http://elblogdelfrasco.blogspot.com/2008/07/override.html
3. fdi.ucm.es (2005) Polimorfismo en Java. Obtenido el 22 de octubre del 2011 desde http://www.fdi.ucm.es/profesor/lgarmend/LPS/Tema%207%20Polimorfismo%20en%20Java.pdf
4. elvex.ugr.es (2011) Curso de programación en Java: Fundamentos de programación y principios de diseño. Obtenido el 23 de octubre del 2011 desde http://elvex.ugr.es/decsai/java/#oo
5. osmosislatina (2011) Polimorfismo. Obtenido el 23 de octubre del 2011 desde http://javabasico.osmosislatina.com/curso/polimorfismo.htm

Respuestas a la autoevaluación

1. Un paradigma se conceptualiza como un modelo o patrón de cualquier disciplina científica o de otro contexto.
2. Un paradigma de programación se concibe como una propuesta tecnológica que un grupo de personas adopta para la resolución de un problema claramente delimitado.
3. Imperativo, funcional, lógico, declarativo, orientado a objetos.
4. Un objeto posee dos contextos: un estado y un comportamiento. El estado se refiere a la configuración inicial de todos y cada uno de los atributos de un objeto. El comportamiento son los cambios que se producen en los estados iniciales de un objeto o la lógica que se ejecuta en un método de la clase instanciada.
5. Estado, comportamiento, identidad y tiempo de vida.
6. Un método es la implementación de la lógica de un objeto. Son los algoritmos que se crean en una clase y cuyo objetivo es manipular los atributos de la misma.

Aplicaciones de escritorio con NetBeans 7.1

4

Reflexione y responda las siguientes preguntas

¿A qué se denominan aplicaciones de escritorio?

¿Están siendo superadas las aplicaciones de escritorio por aplicativos Web?

¿Cuáles son las principales desventajas de las aplicaciones de escritorio en un mundo globalizado donde la información debe estar permanentemente disponible?

¿Es posible migrar aplicaciones de escritorio a un ambiente Web?

Contenido

EXPECTATIVA

Hace muchos años la programación de aplicaciones autónomas (stand alone) era algo común. Luego vinieron las redes locales y se diseminaron por todo el mundo, principalmente se escuchaba sobre redes Novell o redes Windows. Entonces se practicaba el desarrollo de aplicaciones cliente servidor, donde los archivos de datos radicaban en un servidor y las aplicaciones también radicaban en él o se les instalaba en cada cliente. Cuando se instalaban los clientes de la aplicación se lidiaba con la configuración del acceso a datos (los famosos odbc). Las aplicaciones para Windows eran las más comunes y estaban por todos lados: desde stand alone instaladas en una sola máquina, hasta complejas aplicaciones multiusuario que se implementaban en pequeñas redes LAN. Aún sobreviven este tipo de aplicaciones orientándose básicamente a soluciones empresariales internas (en una red LAN). Poco a poco han sido superadas por aplicativos Web, los cuales pueden ejecutarse en un entorno local (LAN) mediante Intranet o global, por medio de una Extranet. Muchos lenguajes de programación ofrecen la posibilidad de convertir aplicaciones que se ejecutan en un entorno de red LAN (aplicaciones de escritorio o desktop) para que lo hagan en un ambiente de internet (aplicaciones Web).

Después de estudiar este capítulo, el lector será capaz de

- Conocer los conceptos básicos de la programación de escritorio (desktop) que se puede desarrollar en NetBeans 7.1.
- Comprender la funcionalidad de los componentes principales que ofrece el IDE NetBeans 7.1
- Desarrollar aplicaciones de escritorio (desktop) utilizando el lenguaje de programación NetBeans.
- Conocer los fundamentos de la programación concurrente, en NetBeans 7.1.
- Crear aplicaciones básicas con programación concurrente con NetBeans 7.1

INTRODUCCIÓN

Desarrollar una aplicación de escritorio (desktop) en NetBeans es una tarea que puede pasar de sencilla a compleja, de acuerdo con el problema que se enfrente. Sin embargo, NetBeans 7.1 ofrece una serie de componentes, librerías y utilitarios que hace fácil la tarea para rápidamente construir aplicaciones poderosas y que cumplan con los requerimientos más sofisticados de los usuarios.

Una aplicación de escritorio (desktop) está pensada para ser utilizada por un usuario que está interconectado a un sistema común en una red LAN. La base de datos se encuentra generalmente creada en un servidor común y los aplicativos en su propia máquina a través de un archivo ejecutable (EXE). Todos los usuarios comparten los mismos datos, por lo que la aplicación debe manejar eficientemente la concurrencia que se presenta en este tipo de aplicaciones.

A continuación se trata una serie de conceptos que son básicos para el desarrollador de este tipo de aplicaciones.

Componentes de una aplicación de escritorio

Los componentes de una aplicaciónde escritorio (desktop) van desde los controles que NetBeans 7.1 proporciona (tales como etiquetas, cajas de texto, botones de comando, listas, tablas, entre otros) como las bibliotecas de clase que implementan la lógica de aplicación o de datos en el aplicativo.

La tabla 4.1 muestra las principales clases en las que se categorizan estos controles.

Tabla 4.1 Principales controles que proporciona NetBeans 7.1.

BIBLIOTECA	COMPONENTES	DESCRIPCIÓN
Contenedores Swing	• Panel (*JPanel*) • Panel divisor (*JSplitPane*) • Barra de herramientas (*JToolBar*) • Ventana interna (*JInternalPane*) • Panel con pestañas (*JTabbedPane*) • Panel de desplazamiento (*JScrollPane)* • Panel de escritorio (*JDesktopPane*) • Panel con capas (*JLayeredPane*)	El paquete swing define dos tipos de contenedores: a) de alto nivel: JFrame, JApplet, JWindows, JDialog y b) de peso ligero: JButton, JPanel, JMenu, JLabel, entre otros. En los contenedores de alto nivel se pueden establecer los contenedores de peso ligero.

BIBLIOTECA	COMPONENTES	DESCRIPCIÓN
Controles Swing	• Botón: JButton • Casillas de activación: JCheckBox • Grupo de botones: ButtonGroup • Lista: JList • Área de texto: JTextArea • Deslizador: JSlider • Cuadro formateado: JFormattedTextField • Spinner: JSpinner • Etiqueta: JLabel • Botón de 2 posiciones: JToogleButton • Botón de opción: JRadioButton • Lista desplegable: JComboBox • Campo de texto: JTextField • Tabla: JTable • Cuadro de contraseña: JPasswordField • Separador: JSeparator • Barra de desplazamiento: JScrollBar • Panel editor: JEditorPane • Barra de progreso: JProgressBar	Los controles Swing brindan la funcionalidad requerida para una aplicación de tipo escritorio (desktop). Entre estos componentes se tiene de lectura, escritura o de procesamiento de algún tipo de procedimiento (por ejemplo, el avance de algún proceso mediante una barra de desplazamiento). Ofrecen una gran flexibilidad de programación.
Menús Swing	• Barra de menú: JMenuBar • Menú: JMenu • Elemento de menú: JMenuItem • Elemento de menú/casilla de activación (JCheckBoxMenuItem) • Elemento de menú/botón de opción (JRadioButtonMenuItem) • Menú emergente (JPopupMenu) • Separador (Separator)	Esta biblioteca implementa todas las funciones posibles de los menús de una aplicación. Desde la barra de menús, los elementos que la componen, hasta separadores o menús emergentes o con casillas de verificación o de opción.

Existen otras librerías tales como *Ventanas Swing* (cuadros de diálogo, Selector de archivos, Selector de color, entre otros), componentes AWT y otras bibliotecas. Nos interesan, básicamente, las bibliotecas citadas en la tabla 4.1.

La figura 4.1 muestra los controles de las bibliotecas indicadas en la tabla 4.1.

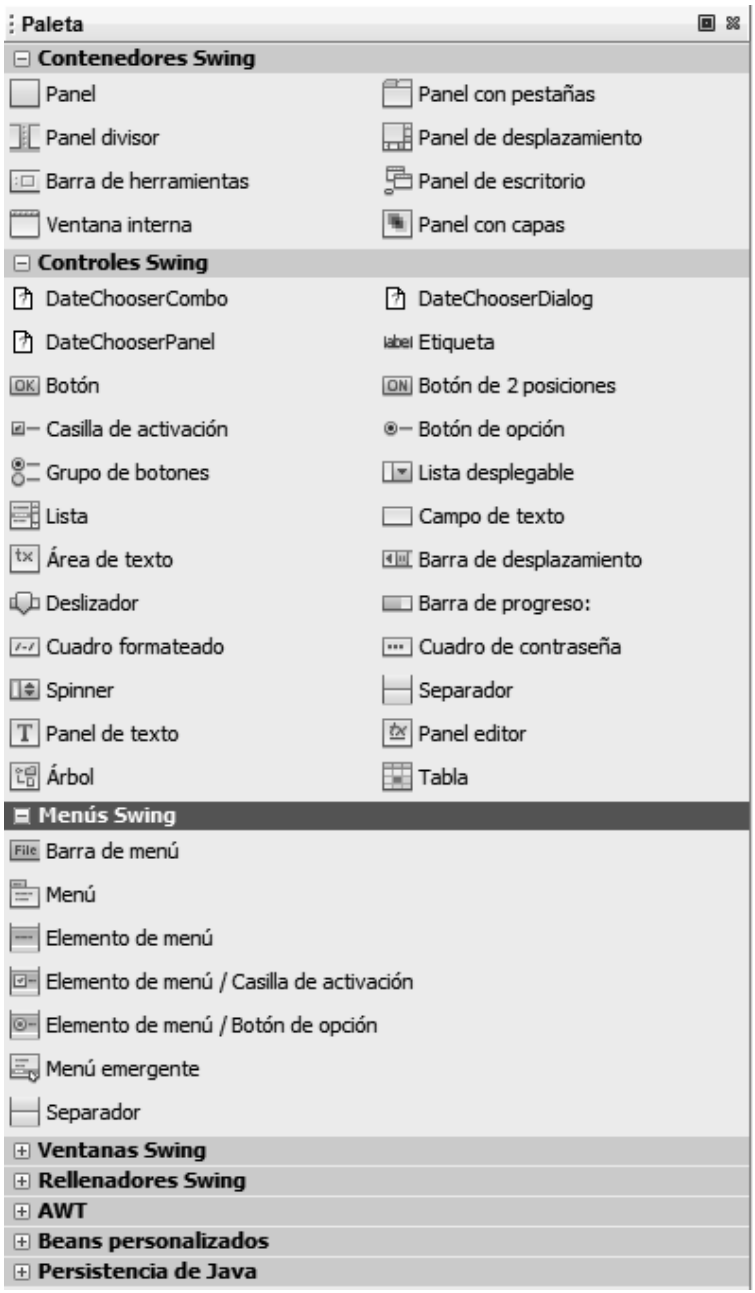


Figura 4.1 Controles que ofrece NetBeans 7.1.

EJEMPLO 1: APLICACIÓN AL ESTILO MDI

A continuación se desarrollará una aplicación que utiliza algunos de estos controles.

1. Ingrese a NetBeans.
2. Una vez en el IDE seleccione Archivo en el menú principal y ahí Proyecto Nuevo…o presione conjuntamente las teclas [Ctrl] + [Mayúsculas] + [N] para crear un proyecto NetBeans.
3. Cree un proyecto nuevo de tipo Aplicación Java, tal como se muestra en la figura 4. 2:

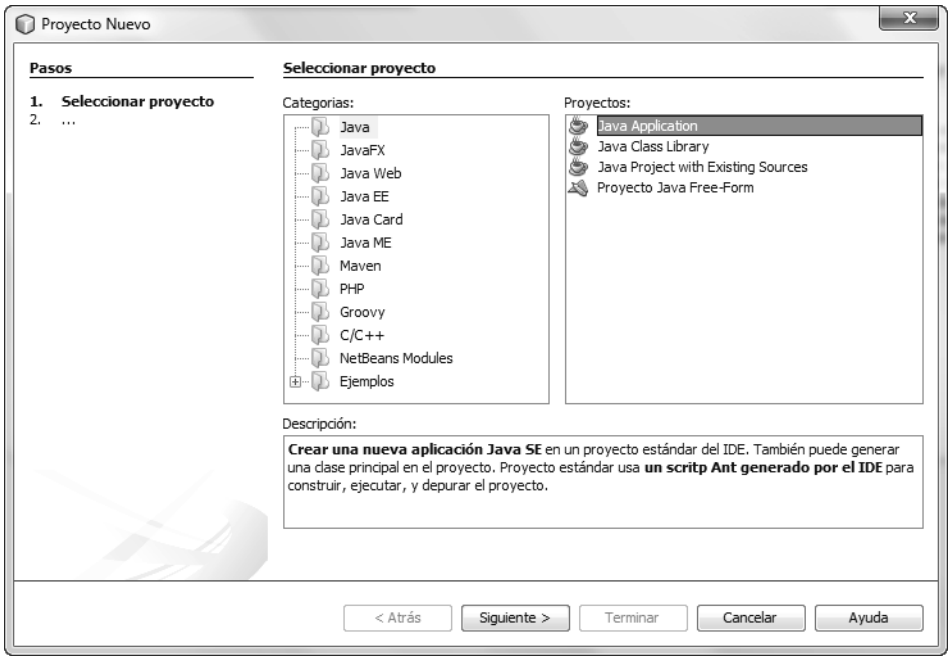


Figura 4.2 Aplicación de escritorio Java (desktop).

4. Pulse el botón *Siguiente* y en la pantalla post siguiente nombre el proyecto como *dskCapitulo4*. Por último, pulse el botón *Terminar*. Con ello se crea una especie de menú inicial.
5. Posteriormente, pulse con el botón derecho del mouse sobre el nombre del proyecto y agregue un nuevo formulario de tipo *JFrame* y nómbrelo *frmMenu*. Observe la figura 4.4.

Figura 4.3 Redefinición de los objetos de la aplicación de escritorio.

6. Una vez creado el formulario debe seleccionar Archivo, Proyecto Properties (propiedades del proyecto) y en las opciones que aparecen seleccionar *Ejecutar* y S*eleccionar* frmMenu como clases principal (Main Class). Puede pulsar el botón *Examinar* para buscar el formulario que servirá como formulario principal de la aplicación.

7. Establezca el tipo de distribución que tendrán los objetos en el formulario *frmMenu*. Pulse sobre el cuerpo del formulario y pulse el botón derecho del mouse. En el menú de contexto que aparece seleccione la opción *Activar gestor de distribución* y ahí elija *Diseño de Borde*. Esto permitirá dividir la ventana del *JFrame* en cinco zonas: norte, sur, este, oeste y centro.

8. Agregue una barra de menú, pulse sobre el cuerpo del formulario con el botón derecho del mouse. Bastará con arrastrarlo hasta el cuerpo del JFrame (formulario).
9. Al principio el formulario presentará sólo dos opciones: File y Edit. Podrá fácilmente editar el título de estas opciones pulsando sobre el mismo y modificar el texto que presenta. Otra forma de editar el texto es pulsando con el botón derecho del mouse sobre la opción y seleccionar *Editar Texto* en el menú de contexto que aparece. Cambie File por el texto *Formularios* y *Edit* por *Salir*.
10. Pulse sobre el cuerpo del formulario y agregue un objeto Panel de escritorio (JDesktopPane) al mismo. Se mostrará una región oscura debajo del menú creado.

JDesktopPane se utiliza en NetBeans para contener otros objetos de tipo ventana como puede se un JInternalFrame, es una aplicación MDI (Multiple Document Interface).

La figura 4.4 muestra como queda este formulario.

Figura 4.4 Un JFrame al estilo MDI.

11. Cámbie el nombre a este panel pulsando sobre el mismo con el botón derecho del mouse y seleccione la opción Cambiar nombre de variable... Ponga el nombre panelInterno.
12. Habilite el menú creado en el formulario frmMenu y pulse con el botón derecho del mouse sobre él. En el menú de contexto que aparece seleccione *Añadir* de la *Paleta* la opción *Elemento de menú* para agregar un nuevo ítem en el menú. En el texto de este ítem escriba *Mantenimiento de Empleados*. Observe la figura 4.5 acerca de este procedimiento.

Figura 4.5 Agregar elementos de menú a la barra de menús.

Observe algunas funcionalidades que se muestran en la figura 4.5, por ejemplo las opciones *Desplazar hacia abajo* y *Desplazar hacia arriba*. Estas opciones lo que hacen es desplazar un elemento de menú hacia abajo o hacia arriba.

13. Agregue ahora un nuevo formulario (*JInternalFrame*) de nombre *frmMantEmpleados*. Si no aparece en los objetos actuales, pulse el clic derecho, *Nuevo*, y vaya al final de las opciones y pulse *Otros* y búsquelo entre los objetos que aparecen ahí. Para agregarlo observe la figura 4.3.

14. A continuación pulse con el botón derecho del mouse sobre el formulario y en el menú de contexto que aparece seleccione *Añadir de paleta*, *Contenedores Swing* y luego *Panel*. Coloque este objeto en la parte superior del formulario. Coloque otro similar en la segunda mitad de la parte inferior.

15. Seleccione el formulario de la primera mitad superior y pulse el botón derecho del mouse escoja *Propiedades* y luego pulse sobre la propiedad *border*. Elija *Borde con título*. En el título escriba *Datos Generales del Empleado* y pulse la tecla ENTER.

    Haga lo mismo con el panel de la segunda mitad inferior del formulario. En el título de este otro panel escriba *Lista de Empleados*.

    La figura 4.6 muestra la forma en que está diseñado hasta el momento nuestro formulario.

Figura 4.6 Agregar dos objetos panel a nuestro formulario.

16. El producto terminado se muestra en la figura 4.7 con los objetos creados para este formulario. Observe las etiquetas numeradas del 1 al 17 con los objetos que componen el formulario y que se detallan en la tabla 4.1

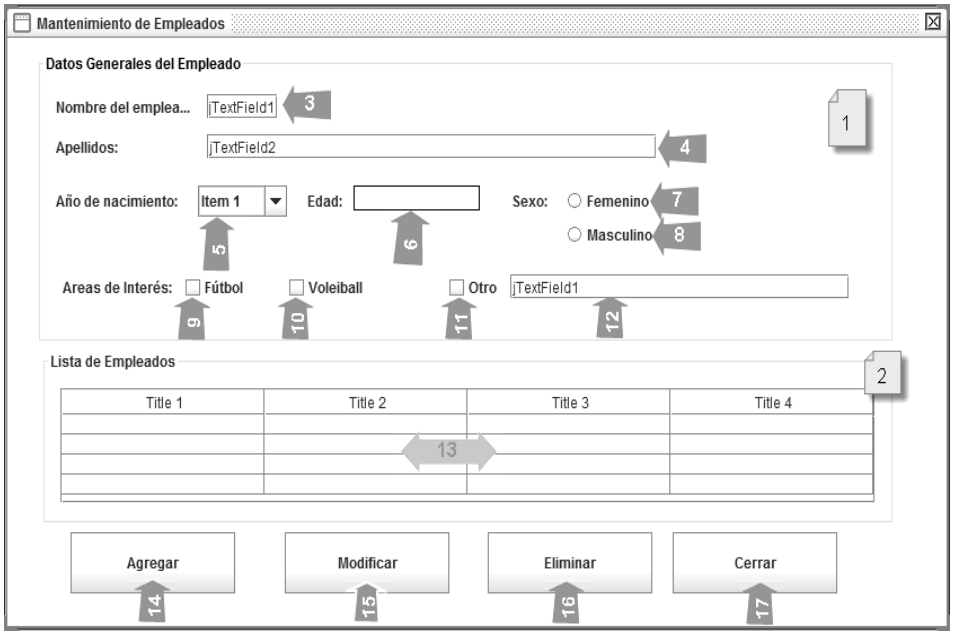


Figura 4.7 El formulario frmMantEmpleados terminado.

Tabla 4.2 Objetos del formulario *frmMantEmpleados.*

ETIQUETA NO.	TIPO DE COMPONENTE	ATRIBUTOS PRINCIPALES
1	**JPanel**	Ninguno
2	**JPanel**	Ninguno
3	**JTextField**	Name: txtNombre
4	**JTextField**	Name: txtApellidos
5	**JComboBox**	Name: cmbAnyos
6	**JLabel**	Name: lblEdad
7	**JRadioButton**	Name: rdFemenino
8	**JRadioButton**	Name: rdMasculino
9	**JCheckBox**	Name: chkFutbol
10	**JCheckBox**	Name: chkValleyball
11	**JCheckBox**	Name: chkOtro
12	**JTextField**	Name: txtOtro
13	**JTable**	Name: lblEmpleados
14	**JButton**	Name: btnAgregar
15	**JButton**	Name: btnModificar
16	**JButton**	Name: btnEliminar
17	**JButton**	Name: btnCerrar

17. Ahora corresponde programar los objetos del formulario, inclusive éste. Inicialmente se escribe el código que se encuentra en el formulario principal (declaración de variables)

EJERCICIO: VARIABLES PRINCIPALES EN EL FORMULARIO
private int anyoActual, fila;
Object[] filas = new Object[6];
javax.swing.table.DefaultTableModel modeloTabla = new javax.swing.table.DefaultTableModel();

La instrucción *Object[] filas = new Object[6]* declara un vector de objetos de 6 posiciones (6 celdas). Una variable de tipo Object define el estado básico y el comportamiento de todos los objetos que deben tener para compararse entre sí,

para notificarse entre ellos, entre otros factores. Todos los objetos del entorno Java heredan sus comportamientos desde la clase Object. En esta línea se declara una variable Object que puede contener dentro de cada fila (en este caso son 6 filas) datos de tipo primitivos tales como int, String o double. Se declara filas como Object, siendo un vector, para que acepte en cada fila un dato de diferente tipo, lo cual en un vector de tipo primitivo no sería posible.

Se crea una variable denominada modelo con la instrucción:

```
javax.swing.table.DefaultTableModel modelo = new javax.swing.table.DefaultTableModel()
```

para efectivamente crear un modelo de tipo tabla donde se cargarán los datos provenientes de filas (Object) y finalmente el modelo será quien cargue el objeto JTable (tabEmpleados). La forma de trabajar en Java para cargar un objeto JTable, por ejemplo, es declar una variable de tipo Object en el cual se cargan las entradas del usuario, posteriormente una variable de tipo JTable, JComboBox o JList y trasladar ahí los datos de la variable Object. De la variable JTable, JComboBox o JList se cargan los datos al objeto JTable, JComboBox o JList, propiamente (el control). Vea la figura 4.8 donde se muestra el procedimiento para la carga de datos en uno de estos componentes.

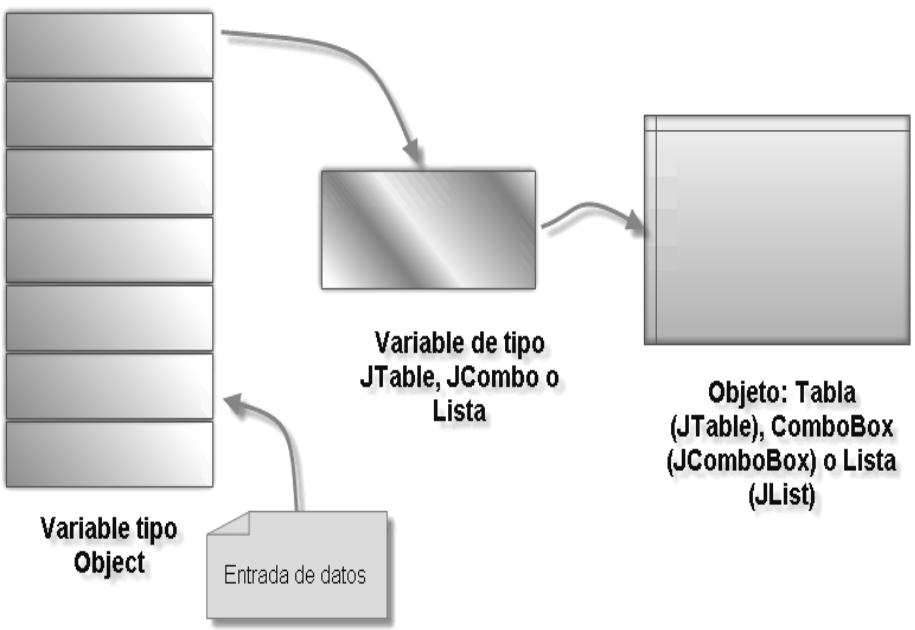


Figura 4.8 Manejo del procesamiento de carga de un objeto JTable, JComboBox o JList.

18. Ahora se procede a programar el evento inicializador de nuestro formulario con los procedimientos que se ejecutarán cuando éste se cargue:

EJERCICIO: INICIALIZAR EL FORMULARIO
public frmMantEmpleados() {
initComponents(); //inicialización de componentes en el formulario
CargarAnyos(); //método para cargar años en el ComboBox
configurarModelo(); //método para configurar el encabezado de la tabla (JTable)
detEdad(); //método para calcular la edad de un empleado
}

19. Ahora se programa el método CargarAnyos() con el siguiente código:

EJERCICIO: PROGRAMAR EL MÉTODO CARGARANYOS()
void CargarAnyos() {
int i;
javax.swing.DefaultComboBoxModel modeloCombo = new javax.swing.DefaultComboBoxModel();
java.util.Calendar fecha = java.util.Calendar.getInstance();
anyoActual = fecha.get(java.util.Calendar.YEAR); //obtener año actual
for (i=1950;i<anyoActual;i++) {
modeloCombo.addElement(i); //se carga un elemento al modelo (un año)
}
cmbAnyos.se odel(modeloCombo); //se carga el combobox desde el modelo
}

20. El método CargarAnyos() con el siguiente código:

EJERCICIO: PROGRAMAR EL MÉTODO CONFIGURARMODELO()
void configurarModelo() { //carga los títulos de la cabecera de la tabla
modeloTabla.addColumn("Nombre");
modeloTabla.addColumn("Apellidos");
modeloTabla.addColumn("Año");
modeloTabla.addColumn("Edad");
modeloTabla.addColumn("Sexo");
modeloTabla.addColumn("Áreas");
}

21. El método detEdad() permite determinar la edad del empleado. El código es el siguiente:

EJERCICIO: PROGRAMAR EL MÉTODO DETEEDAD()
void detEdad() {
int edad = anyoActual - Integer.parseInt(cmbAnyos.getSelectedItem().toString());
lblEdad.setText(String.valueOf(edad)); }

22. Una vez escritos los métodos anteriores, pulse con el botón derecho del mouse sobre el objeto cmbAnyos y seleccione las opciones Eventos, Item y finalmente ItemStateChanged. Ahí haga un llamado al método detEdad().

23. En el botón Agregar (btnAgregar) escriba el siguientes código:

EJERCICIO: PROGRAMAR EL BOTÓN AGREGAR (BTNAGREGAR)
detDatos(); //llama al método detDatos()
modeloTabla.addRow(filas); //carga el modelo con los datos obtenidos //por el método *detDatos()*
tabEmpleados.se odel(modeloTabla); //carga el modelo en JTable //tabEmpleados

24. Observe que el procedimiento detDatos() aún no ha sido definido. Por tanto, escriba el código para ese procedimiento como sigue:

EJERCICIO: PROGRAMAR EL MÉTODO DETDATOS()
void detDatos(){
String AInteres="";
filas[0] = txtNombre.getText(); //carga el nombre
filas[1] = txtApellidos.getText(); //carga el apellido
filas[2] = cmbAnyos.getSelectedItem().toString(); //carga el año de nacimiento
filas[3] = lblEdad.getText(); //carga la edad
if (rdMasculino.isSelected()) {filas[4] = "Masculino";} //si es masculino
else {filas[4] = "Femenino";} //o si es femenino
if (chkFutbol.isSelected()) AInteres=chkFutbol.getText() + ", ";
if (chkVoleiball.isSelected()) AInteres = AInteres + chkVoleiball.getText()+ ", ";

EJERCICIO: PROGRAMAR EL MÉTODO DETDATOS()
if (chkOtro.isSelected()) AInteres = AInteres + txtOtro.getText();
filas[5] = AInteres;
}

25. En el botón Modificar (btnModificar) escriba el siguiente código:

EJERCICIO: PROGRAMAR EL BOTÓN MODIFICAR (BTNMODIFICAR)
detDatos();
for (int i=0;i<6;i++) //par alas 6 columnas de la table
modeloTabla.setValueAt (filas[i], fila, i); //cambie en el modelo por lo que tiene //almacenado el vector filas
tabEmpleados.se odel(modeloTabla); //actualice la tabla (JTable) //con el modelo

26. En el botón Eliminar (btnEliminar) escriba el siguiente código:

EJERCICIO: PROGRAMAR EL BOTÓN ELIMINAR (BTNELIMINAR)
modeloTabla.removeRow(fila);
tabEmpleados.se odel(modeloTabla);

27. Pulse con el botón derecho del mouse sobre la tabla del panel de abajo y asegúrese de seleccionar Eventos, *Mouse* y *MouseClicked* y escriba ahí el siguiente código:

```
fila = tabEmpleados.rowAtPoint(evt.getPoint());
```

Con ello se selecciona el número de fila donde el usuario dio clic.

28. Finalmente se enlazan los objetos *JRadioButton* para que, si seleccionó masculino, no pueda seleccionarse femenino y visceversa. Para estos efectos pulse sobre el formulario con el botón derecho del mouse y seleccione *Anadir de Paleta*, *Controles Swing* y ahí elija un objeto *Grupo de botones*. Se agregará automáticamente un grupo de botones que no tendrán visibilidad (es como crear una variable).

Ahora pulse con el botón derecho del mouse sobre el JRadioButton *rdMasculino* y en las propiedades busque el atributo *buttonGroup* y seleccione el que aparece ahí (probablemente será *buttonGroup1*). Con ello se enlaza el radioButton a este grupo de botones. Seleccione también la propiedad *Selected* habilitando la caja de check que presenta. Con ello se mostrará que este radioButton inicialmente está seleccionado. Haga la primera parte de enlazar el siguiente radioButton (*rdFemenino*) al

buttonGroup. Ahora ambos radioButton están sincronizados, es decir sólo uno de ellos se podrá seleccionar a un mismo tiempo.

Ejecute el ejemplo y podrá agregar, modificar y eliminar registros desde la tabla. Una ejecución común de esta aplicación se muestra en la figura 4.9.

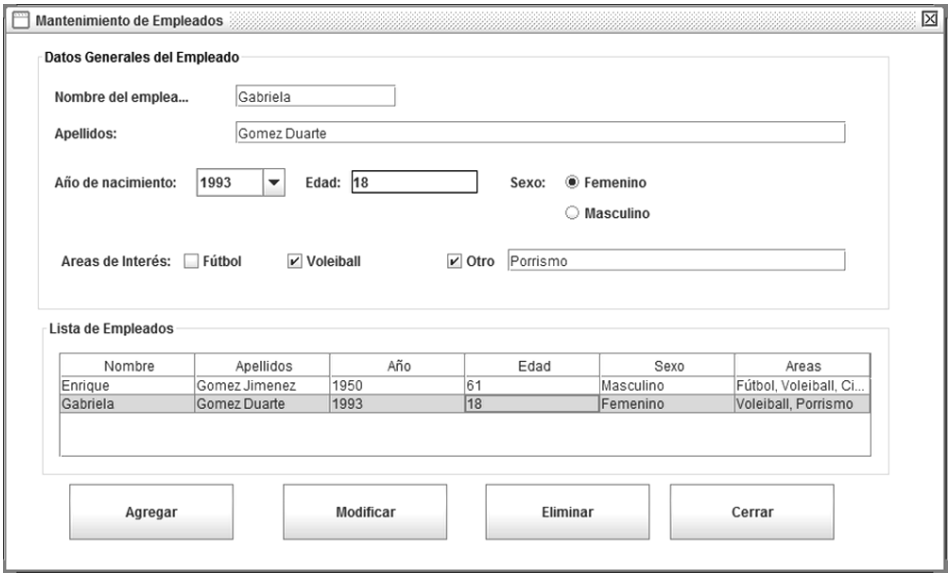


Figura 4.9 Ejecución de nuestro formulario *frmEmpleados.*

Observe también que el formulario de la figura 4.9 queda dentro del entorno del formulario de menú. Observe la figura 4.10 donde se muestra este concepto. Todos los formularios toman como formulario padre a *frmMenu* y nuestra aplicación tendrá un aspecto similar siempre.

Actividades para el lector

Desarrolle programas Java utilizando NetBeans para implementar:

- El mantenimiento de Departamentos del empleado. Los campos serán Nombre del Departamento, Número de Teléfono y Nombre Encargado. La interface debe ser similar al desarrollado en el ejemplo anterior.
- El mantenimiento de nómina de una empresa. Los campos serían nombre del empleado, horas trabajadas, precio por hora, salario bruto (horas trabajas * precio por hora), deducciones y salario neto (salario bruto – deducciones). La interface debe ser similar a la desarrollada en el ejemplo anterior.

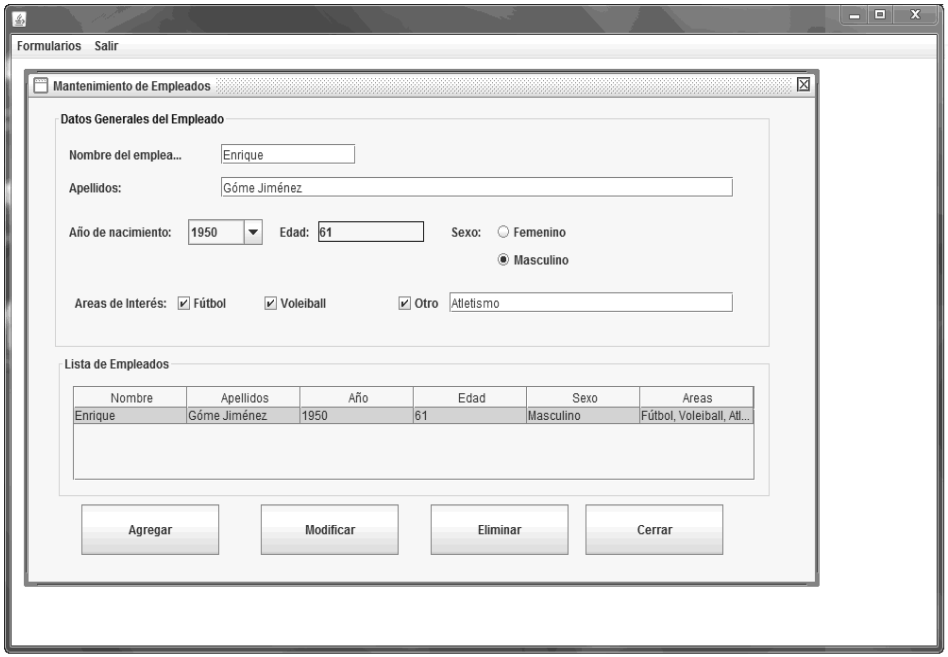


Figura 4.10 Aplicación estilo MDI.

PAQUETES (PACKAGES) EN NETBEANS

Las clases en Java pueden agruparse en familias lógicas. A esto se le denominan paquetes. Al igual que en .NET de Microsoft se utilizan los espacios de nombres (namespaces), en Java se puede agrupar clases con funcionalidades lógicas determinadas. Por default, muchas clases que vienen incluidas en Java pertenecen a un paquete determinado. Por ejemplo, la clase *String* pertenece al paquete *java.lang*. En este caso, para invocar el uso de *String* debería utilizar *java.lang.String*, sin embargo esto es muy tedioso, por lo que al importar el paquete *java.lang* sólo es necesario utilizar el nombre de la clase de la forma *String nombre.*

¿Cuál es la utilidad de los paquetes? Pues básicamente es la ordenación de las clases en grupos funcionales. Asimismo, evita el infierno de las dll's que se producía hace unos años, las colisiones por nombres similares en las dll's (clases) con funcionalidades diferentes, entre otros factores.

La visibilidad de los paquetes dependerá de la declarativa del ámbito de las clases. Una clase sin declaración pública dentro del paquete será reconocida por las demás clases del paquete pero no por otras clases de otros paquetes.

Una vez que ha acumulado muchas clases dentro de un paquete, es importante generar un archivo que pueda importar en sus aplicaciones, cuyo uso

sea fácil y práctico. Para ello se puede generar los denominados archivos JAR (*Java Archives*).

La variable CLASSPATH permite determinar qué librerías se encuentran disponibles para generar nuestro ejecutable. Con CLASSPATH se determina la ubicación de las librerías en una aplicaciónque se desea compilar o generar. Es decir, permite definir la ubicación (mediante directorios) donde están esos paquetes encapsulados en archivos JAR.

EJEMPLO 2: CREACIÓN DE UN PAQUETE (PACKAGE) EN NETBEANS

1. Creación de un proyecto Nuevo en NetBeans 7.1, de tipo Biblioteca de clases Java, tal y como se muestra en la figura 4.11:

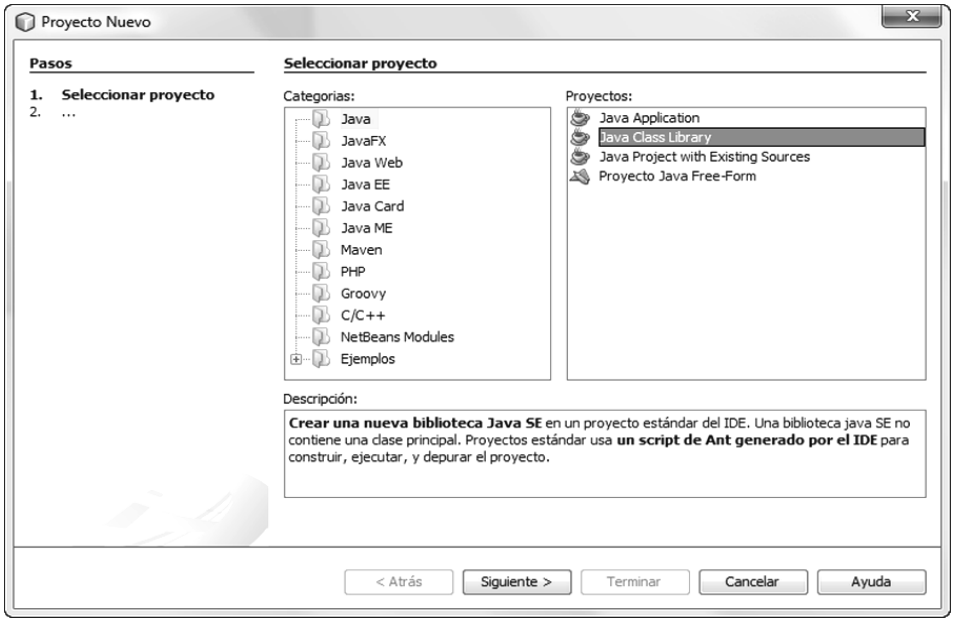


Figura 4.11 Creación de una biblioteca de clases.

2. Pulsar el botón Siguiente.
3. El nombre que se pondrá a esta librería de clases será *LibreriaMatematica*. Pulse el botón *Siguiente* y finalmente *Terminar*. Recuerde guardar este trabajo en una carpeta especial para que siempre tenga presente dónde se encuentra.
4. Pulse sobre el nombre de la biblioteca y con el botón derecho del mouse seleccione un N*uevo* y *Paquete Java,* como se muestra en la figura 4.12.

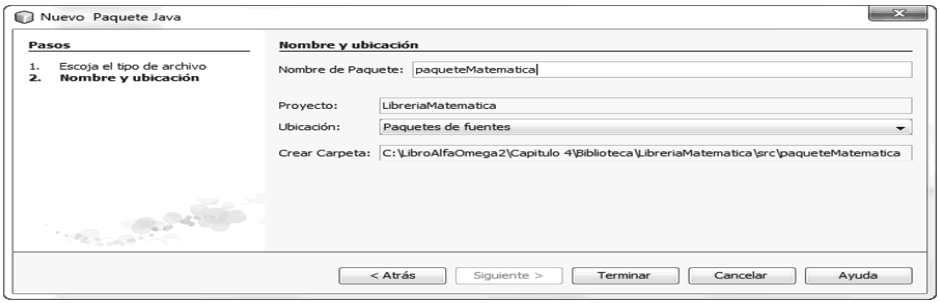


Figura 4.12 Creación de nuestro paquete (package).

5. Observe que el nombre del paquete es *PaqueteMatematica*. Pulse el botón Terminar.
6. Ahora pulse sobre el nombre del paquete seleccione del botón derecho del mouse la creación de una nueva clase denominada *Aritmetica*, tal y como se muestra en la figura 4.13:

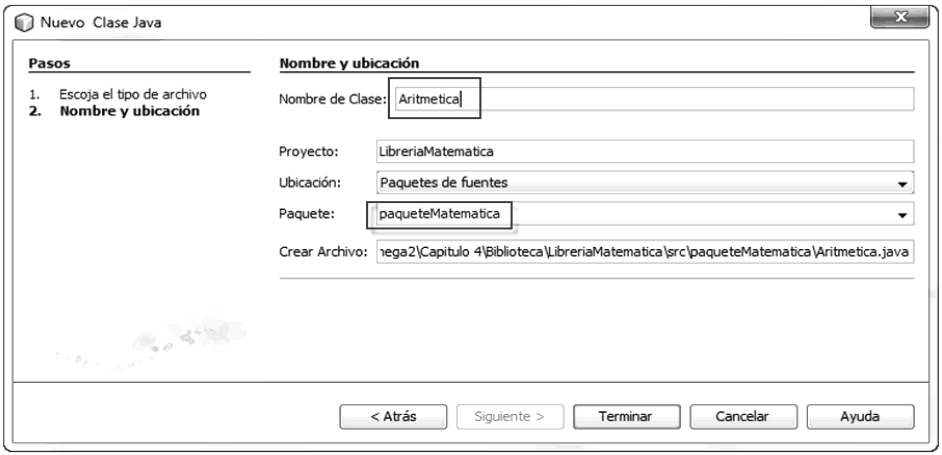


Figura 4.13 Agregr una clase al paquete (package).

7. Pulse el botón *Terminar*. A continuación programe la clase *Aritmetica*, tal y como se muestra a continuación.

EJERCICIO: CREAR LA CLASE ARITMÉTICA
public class Aritmetica {
public double sumar (double a, double b){return a+b;}
public double restar (double a, double b){return a-b;}
public double multiplicar (double a, double b){return a*b;}
public double dividir (double a, double b){return a/b;}
}

8. Luego de guardar la clase Aritmetica, proceda a crear otra clase con el mismo procedimiento del paso 7. Esta vez nombre una clase denominada Calculo. El código para esta clase es el siguiente:

EJERCICIO: CREAR LA CLASE CÁLCULO
public class Matematica {
public double raiz(double x){return Math.sqrt(x);}
public double potencia (double x, double y) {return Math.pow(x, y);}
}

9. Ahora proceda a generar el archivo JAR que se utilizará en otra aplicación. Para ello pulse con el botón derecho del mouse sobre el nombre del paquete y seleccione la opción *Generar*. Con ello ha generado el archivo JAR que utilizará en la próxima sección del ejemplo.
10. Cierre el proyecto luego de que lo haya generado.
11. Después cree un nuevo proyecto, de tipo escritorio, para anexar y utilizar la biblioteca de clase. Denomine este proyecto como *Calculadora*.
12. Observe que el proyecto se llamará Calculadora y se ubicará en C:\Aplicaciones Java. Pulse el botón Terminar.
13. Ahora agregue la librería de clase (*Agregar proyecto...*) pulse sobre la carpeta Librería de su proyecto, tal y como se muestra en la figura 4.14.

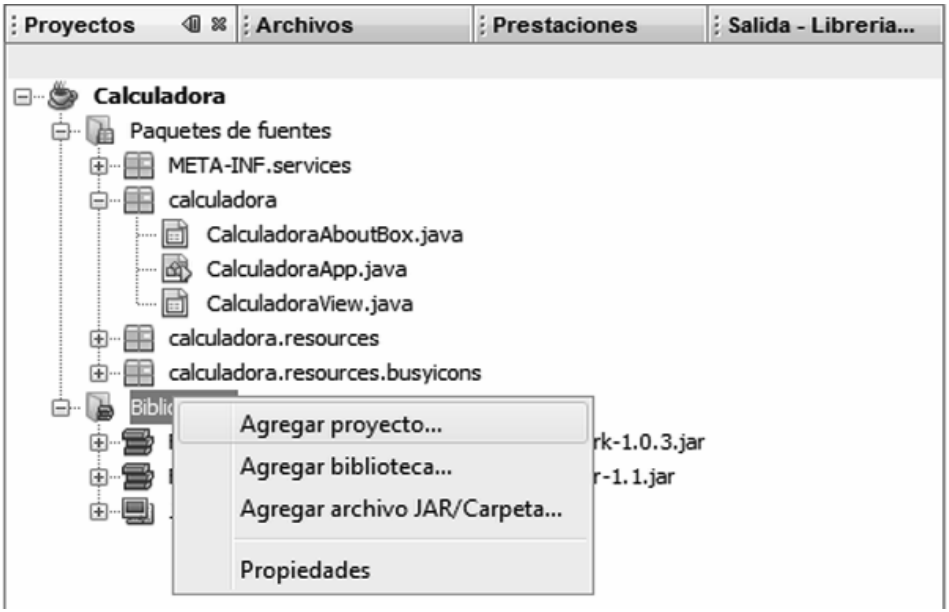


Figura 4.14 Agregar nuestro paquete a una aplicación de escritorio.

14. Una vez que pulse Agregar Proyecto... se mostrará la ventana de selección del JAR que creó. El nuestro se encuentra en la carpeta que se muestra en la figura 4.15:

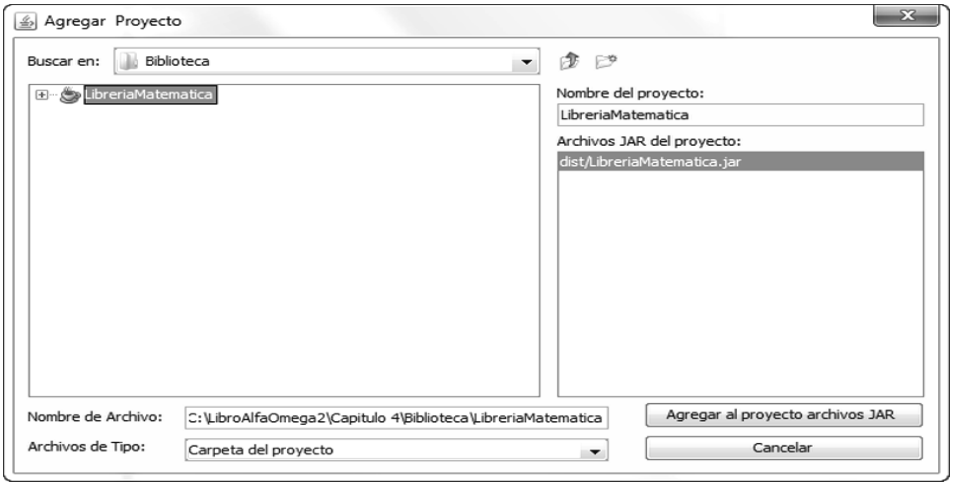


Figura 4.15 Agregar nuestra librera JAR.

15. Pulse el botón Agregar al proyecto archivos JAR. Ya puede ubicar el archivo JAR (nuestro paquete) en la aplicación. Observe la figura 4.16.

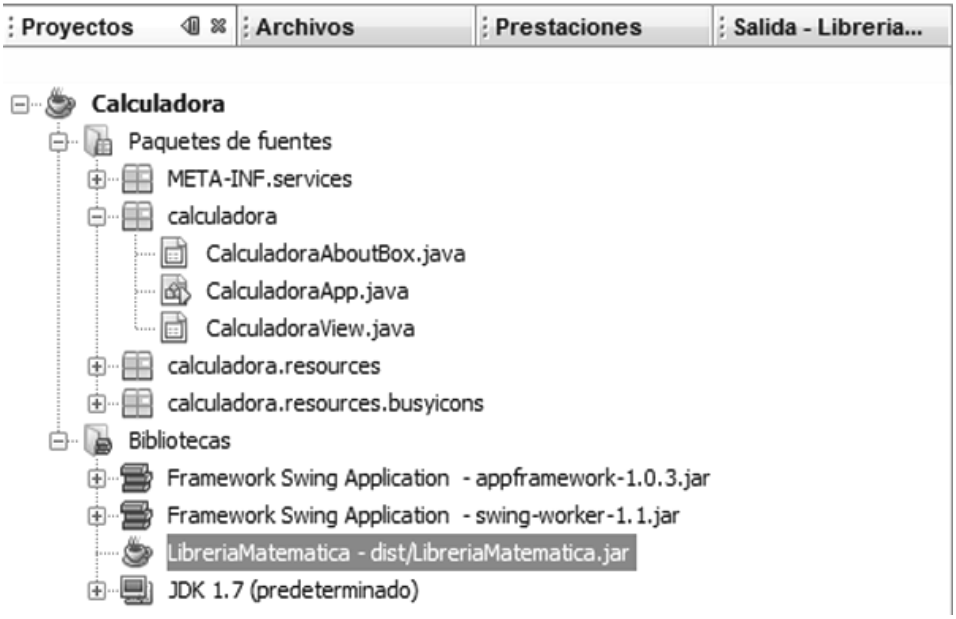


Figura 4.16 Nuestra librera JAR habilitada para ser utilizada.

16. En el formulario principal de nuestra aplicación de escritorio, coloque dos objetos *JTextField*, con sus respectivas etiquetas, tal como se muestra en la figura 4.17.

Figura 4.17 Formulario para probar nuestro paquete.

17. En la clase principal del formulario de la figura 4.17 debe instanciar las clases que derivará de *Aritmetica* y *Calculo*. Observe la figura 4.18 donde se muestra la implementación de estas instanciaciones.

18. El código para utilizar estas instanciaciones es el siguiente;

EJERCICIO: CREAR LAS INSTANCIAS DE CLASE DEL PAQUETE UTILIZADO
PaqueteMatematica.Aritmetica aux = new PaqueteMatematica.Ari etica();
PaqueteMatematica.Calculo aux2 = new PaqueteMatematica.Calculo();

19. Una vez creadas las instancias de ambas clases, seleccione el evento actionPerfomed del botón *Sumar* y escriba el siguiente código:

EJERCICIO: PROGRAMAR EL BOTÓN SUMAR
double a = Double.parseDouble(txtValor1.getText());
double b = Double.parseDouble(txtValor2.getText());
double r = aux.sumar(a,b);
lblResultado.setText(String.valueOf(r));

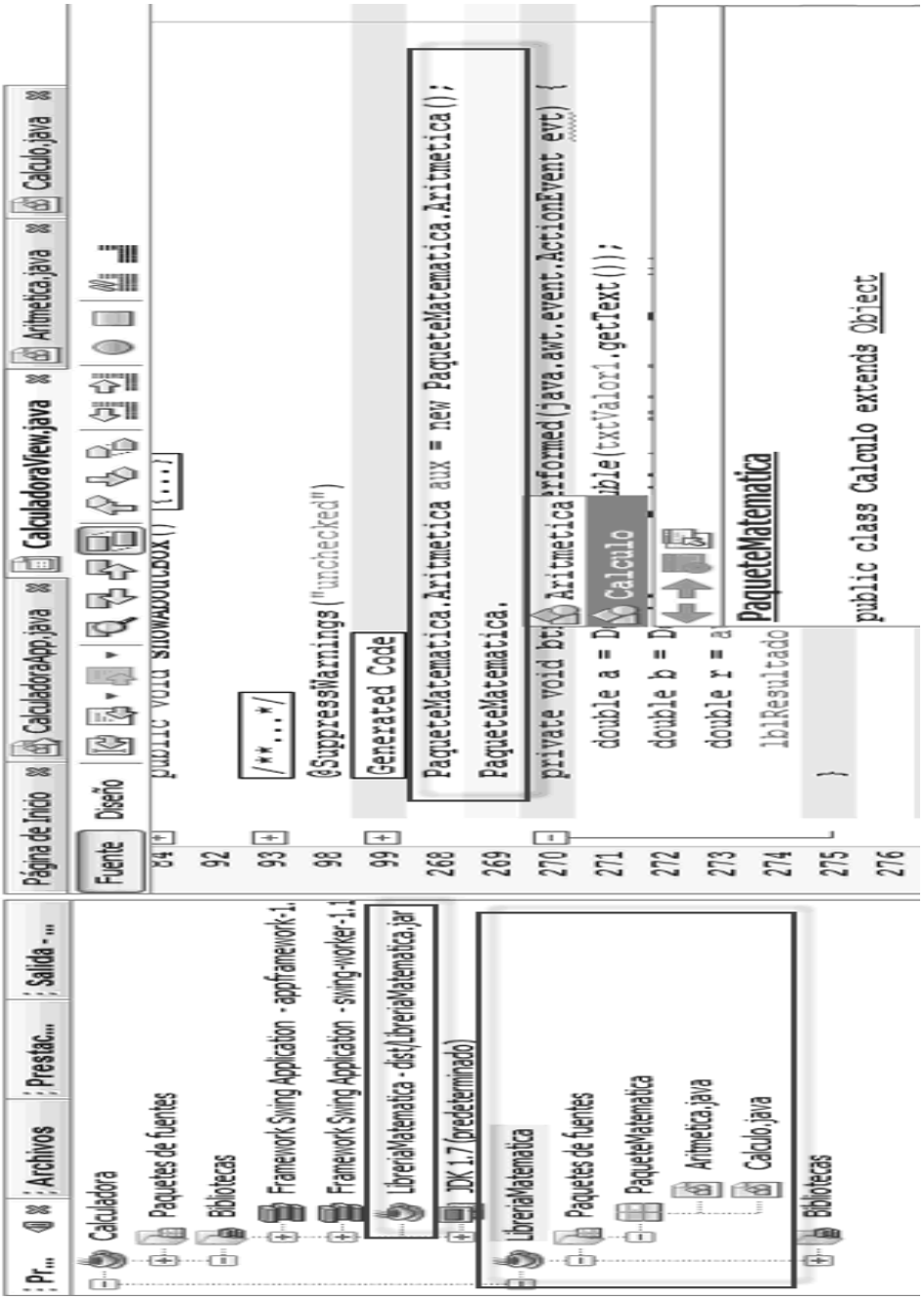


Figura 4.18 Instanciaciones de las clases contenidas en PaqueteMatematica.

EJERCICIO: PROGRAMAR EL BOTÓN POTENCIA
double a = Double.parseDouble(txtValor1.getText());
double b = Double.parseDouble(txtValor2.getText());
double r = aux2.potencia(a, b);
lblResultado.setText(String.valueOf(r));

20. Ahora pruebe su aplicativo. La figura 4.19 muestra una ejecución típica.

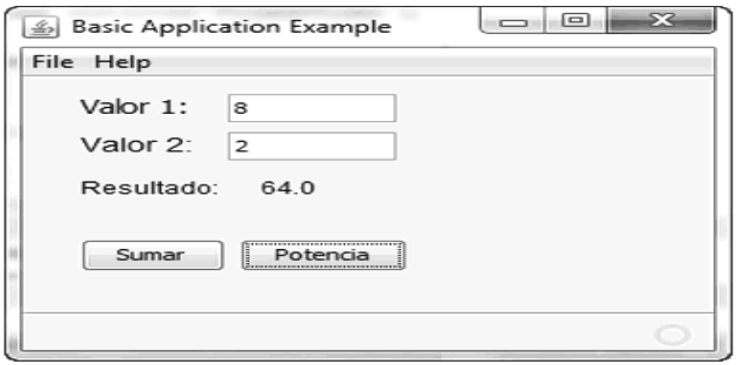


Figura 4.19 Ejecutar la aplicación que usa el paquete Matemático.

Actividades para el lector

- Utilice NetBeans para desarrollar un programa Java que cree un paquete que permita procesar los cálculos empleados en el ejemplo 2.
- Redefina su aplicativo para el uso de este paquete

FUNDAMENTOS DE PROGRAMACIÓN CONCURRENTE

Según Osorio Rivera (2007) "Un programa multihilo contiene dos o más partes que pueden ejecutarse de forma concurrente. Cada parte de ese programa se llama hilo (thread) y cada hilo establece un camino de ejecución independiente. Es un tipo de multitarea distinta a la multitarea por procesos, la cual ejecuta varios programas a la vez. En Java, es un solo programa con varios hilos a la vez"

Lo expresado por Osorio Rivera significa que un programa computacional puede estar conformado por varios subprogramas que se ejecutan en forma individual. Cada ejecución individual se realiza mediante hilos. Mientras que, sistemas como Windows tienen la capacidad de ejecutar varios programas a la vez (multitarea) que constituyen grandes procesos.

Podría afirmase que un hilo es una secuencia de instrucciones que es controlado por un planificador del sistema operativo. Este planificador se encarga de controlar el tiempo de ejecución del hilo en el procesador y de asignar el tiempo a los diferentes hilos que se ejecutan en el sistema, en forma concurrente. A diferencia de un proceso, diversos hilos pueden compartir los mismos datos. El caso de un navegador web puede que una aplicación se encargue de permitir la descarga de un archivo mientras realiza la navegación entre sus páginas o se apresta a descargar otro archivo.

Los hilos son similares a los procesos, con la diferencia que un hilo se ejecuta dentro del contexto de un programa y los procesos lo hacen en su propio espacio de direcciones y de datos. Es decir, los hilos dependen de los recursos de un programa superior (o padre) mientras que un proceso no (tiene sus propios recursos) En la figura 4.20 se muestra la relación existente entre procesos e hilos (como los hilos existen dentro de los procesos).

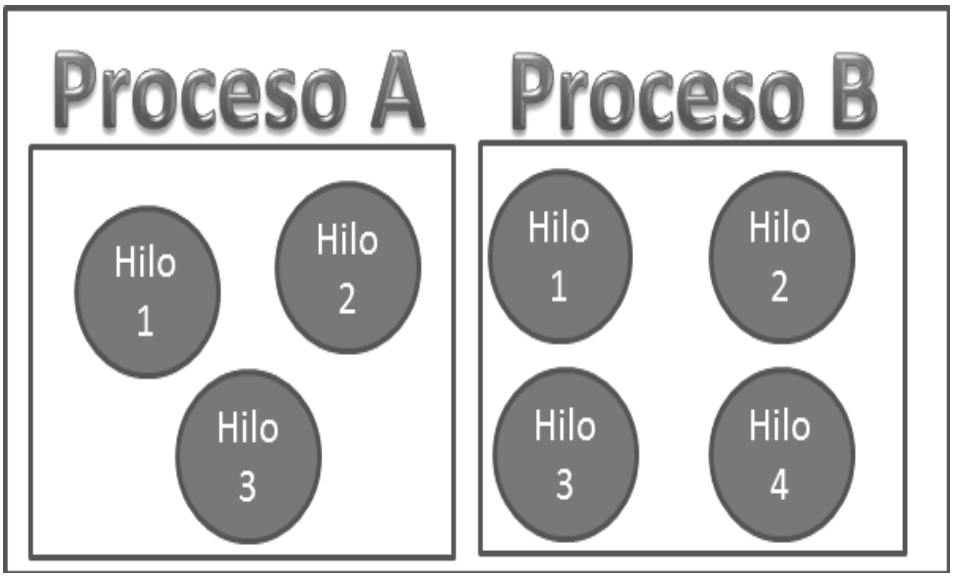


Figura 4.20 Procesos e Hilos.

Programación concurrente en Java

NetBeans permite el manejo de hilos de forma fácil. Basta con heredar de la clase *Thread* y definir la operación del hilo en el método run de la clase donde se implementa el mismo. Posteriormente, se utiliza el método *start()* para poner a funcionar el método del hilo establecido. En Java los hilos se incluyen en el paquete java.lang.thread. Java proporciona soporte a la gestión de hilos mediante una serie de clases y una simple interfaz. Esta interfaz y las clases se resumen en la tabla 4.3.

Tabla 4.3 Algunas clases del paquete java.lang.thread y su interfaz Runnable.

COMPONENTE THREAD	DESCRIPCIÓN
Thread	Constituye una clase que tiene la responsabilidad de implementar la funcionalidad de hilos. Una clase que deriva de Thread implementa la funcionalidad mediante el método run y hereda otros métodos como start, stop (no recomendado su uso), entre otros.
Runnable	Java al no brindar soporte a la herencia múltiple establece el uso de interfaces. Runnable es precisamente eso, una interfaz que proporciona la capacidad de agregar herencia de este tipo, creando la característica multihilo dentro de una clase.
ThreadGroup	Esta clase se usa para el manejo de grupos de hilos, de manera conjunta. Con ello se pretende la gestión eficiente en la ejecución de varios hilos. En esta clase se proporcionan los métodos stop, suspend y resume para controlar esta ejecución.
Object	La clase object proporciona algunos métodos importantes para la ejecución de hilos. Estos métodos son wait, notify y notifyAll. wait hace que un hilo de ejecución se detenga y quede esperando que otro hilo se ejecute (se queda dormido) El método notify le notifica al hilo que esta suspendido mediante wait que continue su ejecución. Finalmente, notifyAll es similar a notify con la diferencia que la notificación es a todos los hilos que están suspendidos.

Como se ha señalado, los hilos comparten el mismo espacio de memoria y los mismos recursos de un proceso determinado. Asimismo, se utilizan diversas vías o caminos para que cada hilo se ejecute en forma concurrente. Por ende, dicho proceso puede ejecutarse de manera más rápida que si lo hiciera en una sola via (solo procesos) Para la creación de hilos existen dos mecanismos: mediante la interfaz Runnable o extendiendo la clase Thread. A pesar que Runnable es recomendable para algunos procesos, se utilizará la derivación de Thread para los ejemplos de hilos.

La sintaxis para crear un hilo en Java, utilizando la clase Thread es la siguiente:

```
class MiHilo extends Thread {
public void run() {
}
}
```

Control de hilos y manejo de estados

La forma en que se comporta un hilo depende del estado que posea en un determinado momento. Es decir, si se encuentra detenido, en suspenso o en ejecución. Los estados que puede tener un hilo en un determinado momento son: New, Runnable, Not running, o Dead.

- New: es el estado inicial. Cuando se crea inicialmente un hilo y posteriormente es llamado a ejecución mediante el método start().
- Runnable: Una vez que se invoca al método start() el método run() es llamado y el hilo entra en un estado de ejecución (runnable)
- Not running: se aplica a todos los hilos que están detenidos por alguna circunstancia. En este caso, está a la espera que sea llamado a volver a ejecutarse. Los posibles estados para que no se ejecute es por que ha sido suspendido (mediante suspend), que esta detenido (mediante sleep), esperando (mediante wait) o ha sido bloqueado por una interrupción de I/O.
- Dead: el hilo se ha destruido. Esto se produce si el método run terminó o se hace un llamado al método stop (no recomendado)

En Java main es la primera función que se invoca tras ejecutarse un programa. Existe solo un main en un programa. Aquí es donde eventualmente se puede implementar el arranque de la ejecución de un hilo. Por ejemplo, la línea:

```
hilo = new HiloUno( "Hilo 1",(int)(Math.random()*123) );
```

Permite la creacion de un Nuevo hilo denominado hilo basado en la implementacion de la clase HiloUno. Los parámetros pasados a la clase es Hilo 1 y (int)(Math.random()*123. Luego para hacer que este hilo inicie su ejecución basta con llamar al método start() de la forma: hilo.start();

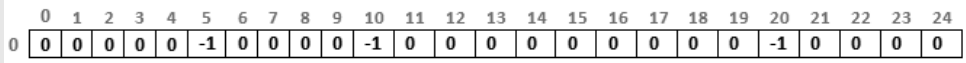
EJEMPLO 3: PROGRAMACIÓN CONCURRENTE

A continuación se iniciará el desarrollo de un ejemplo sencillo de hilos en java. Basado en las especificaciones de un proyecto programado que utilizó una colega profesora en sus clases de programación. El enunciado es el siguiente

Se requiere programar el comportamiento de un juego que siga las siguientes reglas:

a. El juego contiene un vector de 25 casillas las cuales están llenas de 0.
b. Al iniciar el juego tres campos de una JTable deben ser llenados con el valor -1. Los campos son elegidos de forma aleatoria entre todas las posiciones del jTable. Si algún número aleatorio para seleccionar el campo se repite no cuenta siempre deben quedar tres campos llenos con -1

Nota:<u>la posición 0 y la posición 24 no deben quedar con 1</u>



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
0	0	0	0	0	0	-1	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	-1	0	0	0	0

c. Al iniciar el juego, se instancian los hilos jugadores y se empiezan a ejecutar.

d. En el juego participarán 3 jugadores, cada uno de los cuales es un hilo. Cada jugador inicia en la posición cero de la JTable. El jugador lleva el control de su posición en la JTable, cada jugador tiene un nombre y un número de jugador. Se realizarán las siguientes actividades:

   a. Al jugador se le asigna un valor para que avance en la jTable, este valor es generado aleatoriamente en un rango de 0 a 3.

      avance = (int) (Math.random() * 4);

   b. El programa deberá verificar el valor de variable avance con respecto a la posición actual del jugador, ya que esta puede dar un valor mayor al tamaño del jTable.

      i. Si al sumarle a la **posición** actual el valor de *avance* da un número mayor a 24, al jugador se le asigna la **posición** 24

      ii. Si la **posición** a la que el jugador **avanzará** contiene un -1 este debe devolverse a la **posición 0.**

      iii. Una vez que finaliza el turno del jugador, este descansa por 15 segundos.

e. El juego finaliza en el momento en que uno de los jugadores se encuentre en la posición 24, si eso sucede deben detenerse los demás hilos.

f. Durante el juego se deben imprimir las posiciones donde avanzan los jugadores.

1.1 Programe la clase padre correspondiente al Juego. Se debe programar la clase completa y todos los métodos necesarios para ejecutar la funcionalidad mencionada.

1.2 Programe la clase Jugador. Se debe programar la clase completa y todos los métodos necesarios para ejecutar la funcionalidad mencionada.

Programa el formulario. Se debe declarar e inicializar los hilos, y mostrar su funcionamiento.

Una solución propuesta se muestra en el siguiente programa.

1. Ingresa a NetBeans y crea un proyecto tipo Java Application de la categoría Java. Nombre este proyecto como *dskHilos*.
2. Pulse sobre el nombre del proyecto *dskHilos* y en el menú de contexto que se muestra seleccione *Java Package…* de la opción *Nuevo*. Nombre esta carpeta *Logica*.
3. Repita el paso anterior y cree otro paquete denominado *Fisica.*

4. Una vez creados los paquetes pulse con el botón derecho del mouse sobre el paquete *Fisica* y cree un objeto *JFrame* de nombre *FormaHilos*. La interface que debe tener este formulario es el siguiente:

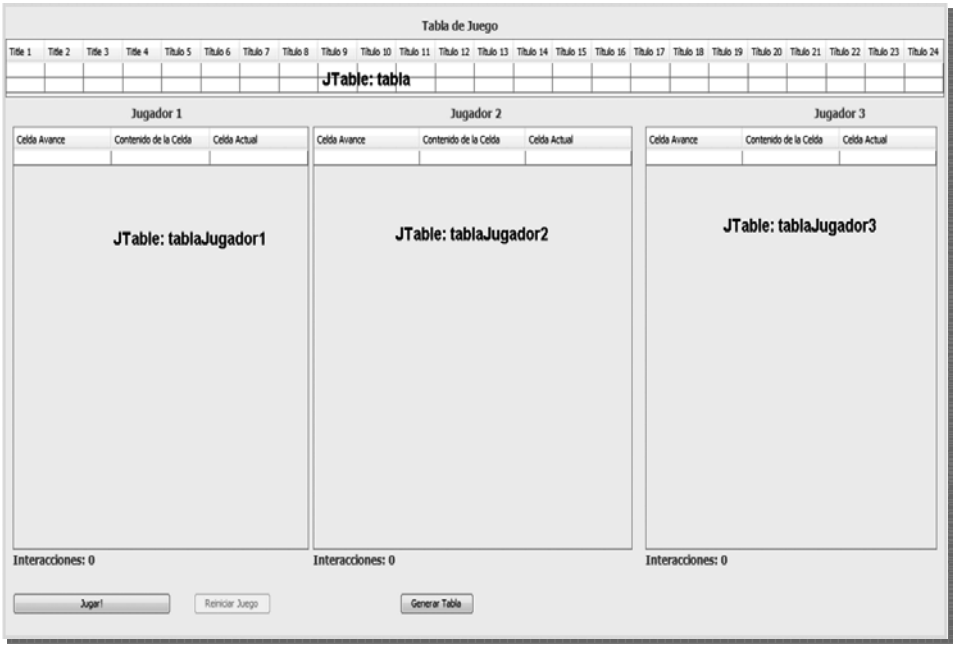


Figura 4.21 Formulario FormaHilos.

Observe que los objetos aparecen con su nombre y tipo en la figura 4.21. Ubica una serie de etiquetas y botones. Las tres etiquetas que aparecen debajo de los JTable se denominan *lblIntJugador1*, *lblIntJugador2* y *lblIntJugador3*, respectivamente. Más debajo de éstas aparecen 3 botones de nombre: *btnJugar*, *btnReinicioJuego* y *btnGenerarTabla*.

5. Ahora proceda a crear las clases que permitirán el funcionamiento de los tres jugadores. Procede a crear la primera clase y nómbrala clsJugador1. El código es el siguiente:

CLASE: CLSJUGADOR1
public class clsJugador1 extends Thread {
private int numero;
private String nombre;
private boolean continuar = true;
public clsJugador1() {

CLASE: CLSJUGADOR1
numero = 1; //le asigna el numero 1 al jugador 1.
nombre = JOptionPane.showInputDialog(null,"Digite el nombre del Jugador 1.");
}
public int getNumero(){return numero;} //para devolver el numero del jugador
public String getNombre(){return nombre;}
public Fisico.FormaHilos formHilo; //declara una variable de tipo FormaHilos
public clsJugador1(Fisico.FormaHilos aThis) { //constructor que recibe por
formHilo = aThis; //defecto el formulario FormaHilos y lo asigna a formHilo.
}
public void detenerHilo(){ //método para detener la ejecución del hilo.
continuar = false;
}
@Override //se sobrescribe el método run
public void run()
{
while (continuar) { //hágase mientras continuar sea verdadero
try {
int avance = (int) (Math.random() * 4); //establece el avance de jugador 1
formHilo.moverJugador1(avance); //invoca al método moverJugador1 que se //encuentra en el formulario FormaHilos
continuar = formHilo.seguir; //para validar si se debe continuar.
Thread.sleep(150); //descansa por 15 segundos...
} catch (InterruptedException ex) {
}
}
}}

6. El código fuente para la clase *clsJugador2*. es:

CLASE: CLSJUGADOR2
public class clsJugador2 extends Thread {
private int numero;
private String nombre;
private boolean continuar = true;
public clsJugador2() {
numero = 2; //le asigna el numero 2 al jugador 2.
nombre = JOptionPane.showInputDialog(null,"Digite el nombre del Jugador 2.");
}
public int getNumero(){return numero;} //para devolver el numero del jugador
public String getNombre(){return nombre;}
public Fisico.FormaHilos formHilo; //declara una variable de tipo FormaHilos
public clsJugador2(Fisico.FormaHilos aThis) { //constructor que recibe por
formHilo = aThis; //defecto el formulario FormaHilos y lo asigna a formHilo.
}
public void detenerHilo(){ //método para detener la ejecución del hilo.
continuar = false;
}
@Override //se sobrescribe el método run
public void run()
{
while (continuar) { //hágase mientras continuar sea verdadero
try {
int avance = (int) (Math.random() * 4); //establece el avance de jugador 2

formHilo.moverJugador2(avance); //invoca al método moverJugador2 que se
//encuentra en el formulario FormaHilos

continuar = formHilo.seguir; //para validar si se debe continuar.

CLASE: CLSJUGADOR2
Thread.sleep(150); //descansa por 15 segundos...
} catch (InterruptedException ex) {
}
}
}}

7. La clase para el jugador 3 posee el mismo código para las dos clases anteriores. Dicho código es el siguiente:

CLASE: CLSJUGADOR3
public class clsJugador3 extends Thread {
private int numero;
private String nombre;
private boolean continuar = true;
public clsJugador3() {
numero = 3; //le asigna el numero 3 al jugador 3.
nombre = JOptionPane.showInputDialog(null,"Digite el nombre del Jugador 3.");
}
public int getNumero(){return numero;} //para devolver el numero del jugador
public String getNombre(){return nombre;}
public Fisico.FormaHilos formHilo; //declara una variable de tipo FormaHilos
public clsJugador3(Fisico.FormaHilos aThis) { //constructor que recibe por
formHilo = aThis; //defecto el formulario FormaHilos y lo asigna a formHilo.
}
public void detenerHilo(){ //método para detener la ejecución del hilo.
continuar = false;
}
@Override //se sobrescribe el método run
public void run()

CLASE: CLSJUGADOR3
{
while (continuar) { //hágase mientras continuar sea verdadero
try {
int avance = (int) (Math.random() * 4); //establece el avance de jugador 1
formHilo.moverJugador3(avance); //invoca al método moverJugador3 que se //encuentra en el formulario FormaHilos
continuar = formHilo.seguir; //para validar si se debe continuar.
Thread.sleep(150); //descansa por 15 segundos...
} catch (InterruptedException ex) {
}
}
}}

8. Una vez que se ha escrito el código para las tres clases que procesarán a cada jugador, procedá a escribir el código que radicará en el formulario principal (FormaHilos.Java) Dicho código es el siguiente:

FORMULARIO: FORMAHILOS
package Fisico;
import Logica.clsJugador1;
import Logica.clsJugador2;
import Logica.clsJugador3;
import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;
public class FormaHilos extends javax.swing.JFrame {
javax.swing.table.DefaultTableModel modeloTabla = new javax.swing.table.DefaultTableModel();
javax.swing.table.DefaultTableModel modTablaJugador1 = new javax.swing.table.DefaultTableModel();
javax.swing.table.DefaultTableModel modTablaJugador2 = new javax.swing.table.DefaultTableModel();

FORMULARIO: FORMAHILOS
javax.swing.table.DefaultTableModel modTablaJugador3 = new javax.swing.table.DefaultTableModel();
Object[] filas = new Object[24];
Object[] datosJugador1 = new Object[3]; //columnas celdaAvance, Contenido,nuevaCelda
Object[] datosJugador2 = new Object[3];
Object[] datosJugador3 = new Object[3];
public Boolean seguir = true;
int celdaJugador1=0;
int celdaJugador2=0;
int celdaJugador3=0;
int p=0;int q=0; int m=0;
clsJugador1 jug1 = new clsJugador1();
clsJugador2 jug2 = new clsJugador2();
clsJugador3 jug3 = new clsJugador3();
public FormaHilos() {
initComponents();
cargarTabla();
jLabel1.setText("Jugador : "+jug1.getNombre().toUpperCase());
jLabel2.setText("Jugador : "+jug2.getNombre().toUpperCase());
jLabel3.setText("Jugador : "+jug3.getNombre().toUpperCase());
}
void cargarDetalle() {
for (int i=0;i<24;i++){
filas[i]=0; }
for(int x=1;x<=3;x++){
int numero = (int) (Math.random()*22+1); //para generar los numeros.
filas[numero]= -1; }

FORMULARIO: FORMAHILOS
modeloTabla.addRow(filas); //carga el modelo con los datos obtenidos
tabla.setModel(modeloTabla); //carga el modelo en JTab //tabEmpleados
}
void cargarTabla(){
for (int i=1;i<=24;i++){
modeloTabla.addColumn(String.valueOf(i)); }
cargarDetalle();}
private void btnGenerarTablaActionPerformed(java.awt.event.ActionEvent evt) {
modeloTabla.removeRow(0);
tabla.setModel(modeloTabla);
cargarDetalle(); }
public void limpiarTabla(javax.swing.JTable miTabla) {
DefaultTableModel modelo = (DefaultTableModel)
miTabla.getModel();
while(modelo.getRowCount()>0)modelo.removeRow(0);
celdaJugador1=0;
celdaJugador2=0;
celdaJugador3=0;
seguir = true;}
void confTablaJugadores(javax.swing.table.DefaultTableModel Modelo){
Modelo.addColumn("Celda Avance");
Modelo.addColumn("Contenido de la Celda");
Modelo.addColumn("Celda Actual");}
private void btnJugarActionPerformed(java.awt.event.ActionEvent evt) {
clsJugador1 j1;
j1 = new clsJugador1(this);

FORMULARIO: FORMAHILOS
clsJugador2 j2;
j2 = new Logica.clsJugador2(this);
clsJugador3 j3;
j3 = new Logica.clsJugador3(this);

j1.start();
j2.start();
j3.start();
if (seguir == false)
{
j1 = null;
j2 = null;
j3 = null;
}
btnJugar.setEnabled(false);
btnReinicioJuego.setEnabled(true);
}

private void btnReinicioJuegoActionPerformed(java.awt.event.ActionEvent evt) {
limpiarTabla(this.tablaJugador1);
limpiarTabla(this.tablaJugador2);
limpiarTabla(this.tablaJugador3);
btnJugar.setEnabled(true);
btnReinicioJuego.setEnabled(false);
lblIntJugador1.setText("Interacciones: 0");
lblIntJugador2.setText("Interacciones: 0");
lblIntJugador3.setText("Interacciones: 0");
}
public void moverJugador1(int celda) {
try{
celdaJugador1+=celda;
if (celdaJugador1 >= 24 && seguir == true)

FORMULARIO: FORMAHILOS
{
celdaJugador1=24;
seguir = false;
JOptionPane.showMessageDialog(null, "Juego Terminado...Ganó el Jugad No."+String.valueOf(jug1.getNumero()) +", " + jug1.getNombre().toUpperCase());
} //aqui gana el jugador...
if (celdaJugador1 <= 24) {
Object valorCelda = tabla.getValueAt(0,celdaJugador1);
if ("-1".equals(valorCelda.toString())) celdaJugador1 = 0;
if (p == 0) {confTablaJugadores(this.modTablaJugador1);p=1;}
datosJugador1[0]=String.valueOf(celda);
datosJugador1[1]=valorCelda;
datosJugador1[2]=celdaJugador1;
modTablaJugador1.addRow(datosJugador1);
tablaJugador1.setModel(modTablaJugador1);
lblIntJugador1.setText("Interacciones: " String.valueOf(modTablaJugador1.getRowCount()));
}
} catch (Exception ex)
{
seguir = false;
}
}
public void moverJugador2(int celda) {
try{
celdaJugador2+=celda;
if (celdaJugador2 >= 24 && seguir == true) {
celdaJugador2=24;

FORMULARIO: FORMAHILOS
seguir = false;
JOptionPane.showMessageDialog(null, "Juego Terminado...Ganó el Jugad No."+String.valueOf(jug2.getNumero()) + ", " + jug2.getNombre().toUpperCase());
} //aqui gana el jugador...
if (celdaJugador2 <= 24) {
Object valorCelda = tabla.getValueAt(0,celdaJugador2);
if ("-1".equals(valorCelda.toString())) celdaJugador2 = 0;
if (q == 0) {confTablaJugadores(this.modTablaJugador2);q=1;}
datosJugador2[0]=String.valueOf(celda);
datosJugador2[1]=valorCelda; datosJugador2[2]=celdaJugador2;
modTablaJugador2.addRow(datosJugador2);
tablaJugador2.setModel(modTablaJugador2);
lblIntJugador2.setText("Interacciones: " String.valueOf(modTablaJugador2.getRowCount()));
}
} catch (Exception ex)
{
seguir = false; } }
public void moverJugador3(int celda) {
try{
celdaJugador3+=celda;
if (celdaJugador3 >= 24 && seguir == true) {
celdaJugador3=24;
seguir = false;
JOptionPane.showMessageDialog(null, "Juego Terminado...Ganó el Jugad No."+String.valueOf(jug3.getNumero()) + ", " + jug3.getNombre().toUpperCase());

FORMULARIO: FORMAHILOS
} //aqui gana el jugador
if (celdaJugador3 <= 24) {
Object valorCelda = tabla.getValueAt(0,celdaJugador3);
if ("-1".equals(valorCelda.toString())) celdaJugador3 = 0;
if (m == 0) {confTablaJugadores(this.modTablaJugador3);m=1;}
datosJugador3[0]=String.valueOf(celda);
datosJugador3[1]=valorCelda;
datosJugador3[2]=celdaJugador3;
modTablaJugador3.addRow(datosJugador3);
tablaJugador3.setModel(modTablaJugador3);
lblIntJugador3.setText("Interacciones: " String.valueOf(modTablaJugador3.getRowCount())); }
} catch (Exception ex)
{
seguir = false;
}
}

9. Una vez que se ha escrito todo el código de la aplicación puede ejecutarla. Para ello, pulsa con el botón derecho del mouse sobre el nombre del formulario FormaHilos y selecciona Ejecutar Archivo. Se mostrará una pantalla similar a la figura 4.22.

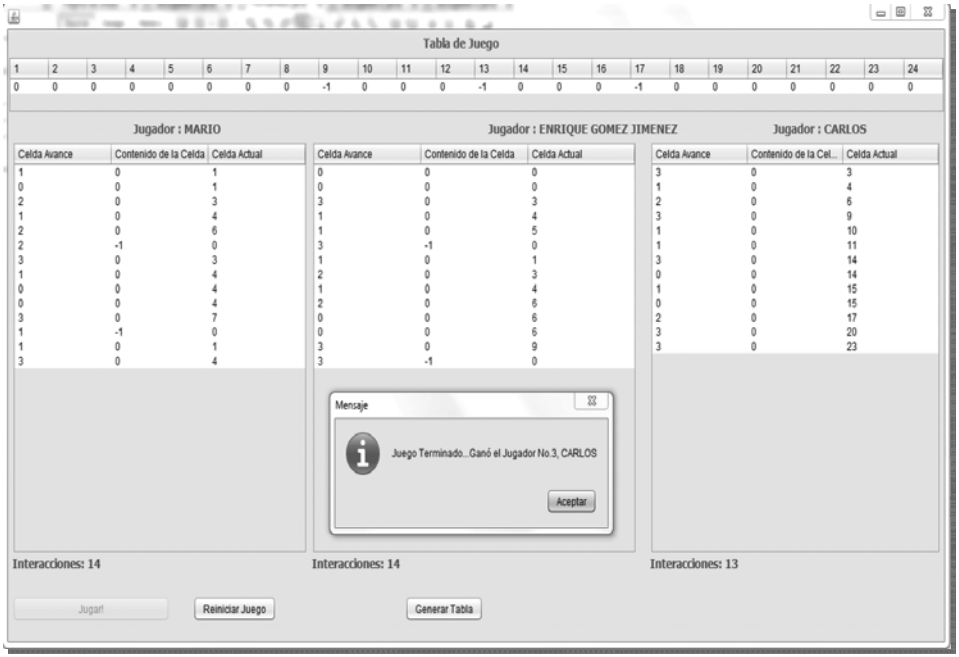


Figura 4.22 Ejecución del formulario FormaHilos.

Actividades para el lector

Desarrolle un programa Java con NetBeans que mediante hilos procese los juegos de 4 equipos de futbol y genere la tabla de posiciones respectivas. Las reglas, de manera general, son las siguientes:

- Jugaran los 4 equipos al mismo tiempo.
- No se podrán enfrentar consigo mismo.
- El equipo que gané obtendrá 3 puntos, un empate 1 y una derrota 0.
- Los resultados serán de forma aleatoria.
- Si persisten los empates para determinar el lugar del equipo se utilizará el gol diferencia (goles a favor menos goles en contra)

Con esto se llega al final del capítulo. Espero que la conceptualización de los temas tratados haya sido clara. Sin embargo, el lector puede revisar las referencias Web para ahondar más en los conceptos, así como descargar los ejemplos y materiales adicionales desarrollados para esta obra, de la página Web de Alfaomega, en la sección correspondiente al libro.

RESUMEN

En este capitulo se desarrollaron los fundamentos de la programación orientada al escritorio (desktop). A pesar de no detallar gran parte de la teoría, se desarrollan tres ejemplos claros y sencillos de cómo utilizar algunos componentes usuales de este tipo de aplicaciones, y cómo crear y utilizar un paquete (packages) en una aplicación desktop. Además se introdujo al uso de hilos en NetBeans. Se trataron los siguientes temas:

1. Explicar los principales componentes que conforman una aplicacióndesktop o de escritorio.
2. Desarrollar una aplicación de escritorio (desktop).
3. Explicar el funcionamiento de los paquetes (packages) en NetBeans.
4. Desarrollar un paquete (package) en NetBeans y la aplicaciónde escritorio (desktop) que lo utiliza.
5. Explicar el funcionamiento de hilos en Java.
6. Crea un ejemplo que demuestre el uso de hilos en Java, utilizando NetBeans.

Autoevaluación

1. ¿Qué es una aplicación de escritorio (desktop)?
2. ¿Qué son los componentes de una aplicación de escritorio (desktop)?
3. ¿Para qué se utilizan los contenedores swing?
4. ¿Para qué se emplea un componente JInternalFrame?
5. ¿Cuál es el procedimiento para cargar datos en objetos JTable, JComboBox o JList?
6. ¿Qué son los paquetes (packages) en NetBeans?
7. ¿Qué es un programa multihilo?
8. ¿Cuáles son los estados que puede tener un hilo en un determinado momento?

EVIDENCIA

- [ ] Desarrolló un programa Java utilizando NetBeans que implementa el mantenimiento de Departamentos del empleado. Los campos son: Nombre del Departamento, Número de Teléfono y Nombre Encargado. La interface debe ser similar a la desarrollada en el primer ejemplo de este capítulo.

- [ ] Desarrolló un programa Java utilizando NetBeans que implementa el mantenimiento de nómina de una empresa. Los campos son: nombre del empleado, horas trabajadas, precio por hora, salario bruto (horas trabajadas*precio por hora), deducciones y salario neto (salario bruto – deducciones). La interface debe ser similar a la desarrollada en el primer ejemplo de este capítulo.

- [ ] Desarrolló un programa Java utilizando NetBeans, para procesar los cálculos empleados en el ejemplo 2 (Crear un paquete en NetBeans) y redefinió la aplicación en para el uso de este paquete.

- [ ] Desarrolló un programa Java utilizando NetBeans, para procesar los cálculos empleados en el ejercicio anterior y redefinió la aplicación para el uso de este paquete.

- [ ] Desarrolló un programa Java que utiliza NetBeans, para ejecutar procedimientos concurrentes, mediante hilos.

- [ ] Desarrolló un programa Java que utiliza NetBeans, para ejecutar procedimientos concurrentes que mediante hilos procese los juegos de 4 equipos de futbol y genere la tabla de posiciones respectivas.

REFERENCIAS

Bibliografía

Arce, Fco. Javier, (2011). *ActionScript 3.0 Aprenda a programar*. Alfaomega, México.

Ceballos, Fco. Javier, (2004). *Programación orientada a objetos con C++*. 3a. ed., Alfaomega, México.

Ceballos, Fco. Javier, (2008). *Java 2. Interfaces gráficas y aplicaciones para internet*. 3a. ed., Alfaomega, México.

Ceballos, Fco. Javier, (2009). *Enciclopedia del lenguaje C++*. 2a. ed., Alfaomega, México.

Ceballos, Fco. Javier, (2011). *Java 2: Curso de programación*, 4a. ed., Alfaomega, México.

López, Leobardo, (2006) *Metodología de la programación orientada a objetos*. Alfaomega, México.

López, Leobardo; Ramírez, Felipe, (2011) *Lógica para computación*, Alfaomega, México.

Martín, Antonio, (2010). *Programador certificado Java 2. Curso práctico*, 3a. ed., Alfaomega, México.

Peñaloza, Ernesto, (2004) *Fundamentos de programación C/C++*. 4a. ed., Alfaomega, México.

Ramírez, Felipe, (2007) *Introducción a la programación: algoritmos y su implementación en VB.Net, C#, Java y C++*. 2a. ed., Alfaomega, México.

Rozannski, Uwe, (2010). *Enterprise Javabeans 3.0. Con Eclipse Y JBoss*. Alfaomega, México.

Sznajdleder, Pablo, (2010) *Java a fondo. Estudio del lenguaje y desarrollo de aplicaciones.* Alfaomega, México.

Páginas Web recomendadas

1. wiki.netbeans.org (2011) *Tutorial Básico de NetBeans para Novatos*. Obtenido el 26 de octubre del 2011 desde http://wiki.netbeans.org/Avbravotutorialbasiconetbeans
2. Jesús Omar Álvarez Márquez (2010) *Manual de creación de Interfaces de Usuario en Netbeans.* Obtenido el 26 de octubre del 2011 desde *three-headed-monkey.googlecode.com.*
3. blog.dairdev.com (2011) *Tutorial : Crear el proyecto en Netbeans 7 IDE - Paso 1.* Obtenido el 26 de octubre del 2011 desde http://blog.dairdev.com/2012/03/tutorial-crear-el-proyecto-en-netbeans.html
4. programatium.com (2011) *NetBeans IDE 7.0.1- Aplicación para desarrollo en Java*. Obtenido el 26 de octubre del 2011 desde http://programatium.com/videos-software/video-tutorial.php?id=NetBeans%20IDE%207.0.1Aplicaci%C3%B3n%20para%20desarrolo%20en%20Java

Respuestas a la autoevaluación

1. Son aquellas que se desarrollan para utilizarse en un ambiente de red local (LAN) de una empresa o de manera stand alone.
2. Son los controles que aporta NetBeans para la creación de aplicaciones de escritorio (desktop).
3. Para incluir en ellos los componentes de peso ligero que se incluyen en NetBeans tales como cajas de texto, etiquetas, entre otros.
4. Para acoplarse en una aplicación de escritorio (desktop) que maneja una interface al estilo MDI.
5. Primero se carga un vector de tipo Object, los datos se pasan a una variable del tipo requerido (JTable, JComboBox o JList) y finalmente de éste se pasa al objeto JTable, JComboBox o JList.
6. Son librerías que agrupan clases funcionales y que pueden ser utilizadas en cualquier programa que se desarrolle con NetBeans.
7. Un programa multihilo contiene dos o más partes que pueden ejecutarse de forma concurrente.
8. Los estados que puede tener un hilo en un determinado momento son: New, Runnable, Not running, o Dead.

Gestión de base de datos MySQL con NetBeans 7.1

5

Reflexione y responda las siguientes preguntas

Mencione al menos dos sistemas gestores de bases de datos propietarias y dos de software libre

¿Es segura una base de datos que se gestiona con software libre como una que se hace con software propietario?

¿En qué situaciones podría utilizar, de manera segura, una base de datos de software libre?

¿En qué puede sustentar la decisión de implementar un sistema gestor de bases de datos que sea de software libre en una empresa?

Contenido

EXPECTATIVA

Que el lector reconozca la importancia de la gestión de bases de datos como tema trascendental en el desarrollo de software actual.

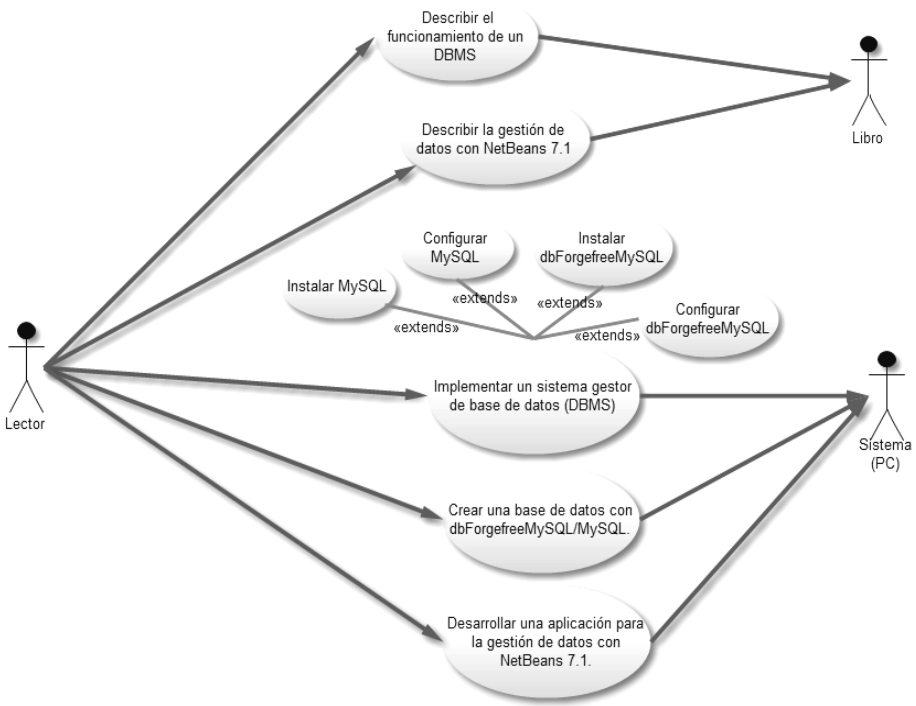
Que el desarrollador utilice la interfaz de NetBeans para la gestión efectiva y dinámica de datos, que conozca sus utilerías y comandos para la creación de objetos y consulta de contenidos.

Que se conozca un lenguaje que pueda facilitar la creación de aplicaciones que gestionen datos sin tener que utilizar el DBMS.

Que el lector desarrolle aplicaciones robustas para todo el proceso de inserción, actualización, eliminación y extracción de información.

Después de estudiar este capítulo, el lector será capaz de

- Instalar y configurar un servidor de base de datos y su respectivo gestor de administración.
- Comprender el funcionamiento de la gestión de datos en NetBeans mediante la arquitectura JDBC.
- Desarrollar aplicaciones que involucren la gestión de bases de datos con NetBeans y MySQL, utilizando mantenimientos de datos y consultas por selección y por acción (procedimientos almacenados).



INTRODUCCIÓN

La gestión de datos representa para la empresa moderna una de las actividades más importantes para su supervivencia en los negocios. Muchas transacciones comerciales, financieras o de cualquier tipo que represente intercambio de productos o servicios, hoy día se hace por medios electrónicos. No se puede imaginar un mundo sin los cajeros automáticos, sin las consultas electrónicas de fondos de cheques o sin las transferencias electrónicas de fondos. Un mundo globalizado requiere que nuestros datos estén disponibles en todo momento y en cualquier lugar: la informática ha llegado a todas partes y con ello las bases de datos. Eventualmente, la decisión de utilizar un sistema gestor de base de datos de características de software libre dependerá de su grado de confiabilidad, de quien tome las decisiones tecnológicas de la empresa, de la fortaleza de la base de datos (experiencias comprobables sobre su utilización en sectores similares al donde se requiere implementar), del grupo que detrás de su desarrollo (MySQL tiene detrás el equipo de Oracle), la funcionalidad crítica de la base de datos (no es lo mismo implementar una base de datos para uso de correos que para realizar operaciones de internet banking), entre otros factores. Esta por demás argumentar que la seguridad y fiabilidad de sistemas gestores de bases de datos de características de software libre tienen buen grado de madurez (como el caso de MySQL) y por tanto, una gran capacidad de administración, procesamiento y seguridad.

Existen muchos sistemas administradores de bases de datos propietarios u Open Source. Todos han mejorado con el tiempo ofrecen seguridad, rapidez, alto desempeño, entre otros factores técnicos. Los lenguajes de programación también deben poner su aporte mediante la gestión que deben realizar sobre dichos datos. La gestión de datos, por ende, es un asunto crucial para los diseñadores de bases de datos, quienes se enfrascan en una constante lucha para ser el mejor.

Instalación de MySQL

Este capítulo se inicia instalando MySQL, versión 5.1, el cual se podrá descargar desde *http://dev.mysql.com/downloads/mysql/5.0.html.* MySQL es un software tipificado como Open Source y que se orienta para su uso general en una variedad importante de aplicaciones de gestión de datos. No se le considera destinado para su uso en aplicaciones de riesgo, entre las que se incluyen las que representan un riesgo de lesiones personales. En caso que se utilice en estos escenarios el desarrollador se responsabiliza de tomar las precauciones del caso, como las que se encuentran las pruebas de fallos, copias de seguridad, redundancia, entre otras que se incluyen en el sitio de Oracle® quien es la responsable del diseño y desarrollo de la herramienta.

1. Inicie la instalación de MySQL pulsando dos veces con el puntero del mouse sobre el archivo mysql-5.5.12-win32.exe, el cual se encuentra en los archivos adicionales de este libro. Se mostrará una pantalla similar a la figura 5.1.

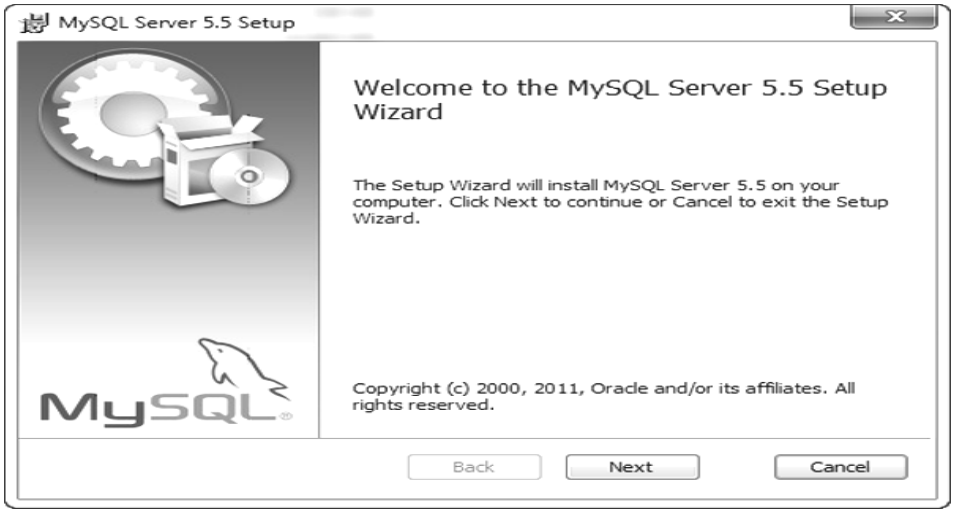


Figura 5.1 Instalación de MySQL.

2. En la tercera pantalla se le preguntará qué tipo de instalación requiere para su instalación de MySQL. Seleccione la instalación típica. Pulse la tecla Next (siguiente).

3. En la pantalla *MySQL Server Instance Configuration Wizard* seleccione la opción *Detailed Configuration*, tal como se muestra en la figura 5.2.

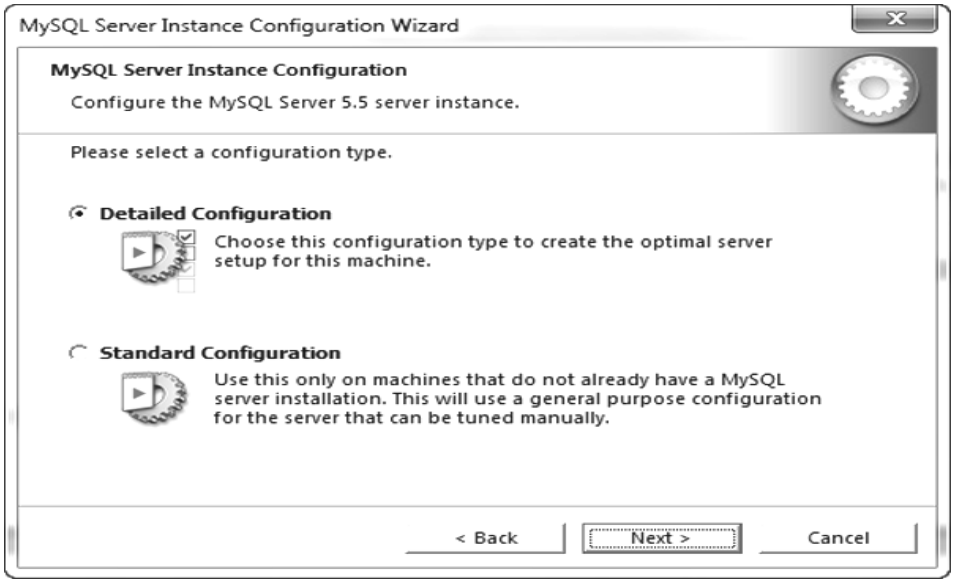


Figura 5.2 Configuración de la instancia de MySQL.

4. A continuación se presentará una pantalla para seleccionar el tipo de servidor que utilizará para MySQL. Elija *Server Machine* y pulse el botón *Next*.
5. En la pantalla siguiente seleccione la opción *Multifunctional database* y pulse sobre el botón *Next*.
6. Dos pantallas posteriores, seleccione *Decision Support (DSS)/OLAP* y pulse sobre el botón *Next*.
7. En la pantalla de seguridad le pide cambiar el password. Escriba Alfaomega y pulse el botón *Next*.
8. Una vez terminada la instalación se podrá ingresar al editor de comandos de MySQL, tal como se ve en la figura 5.3. Recuerde que la contraseña es Alfaomega.

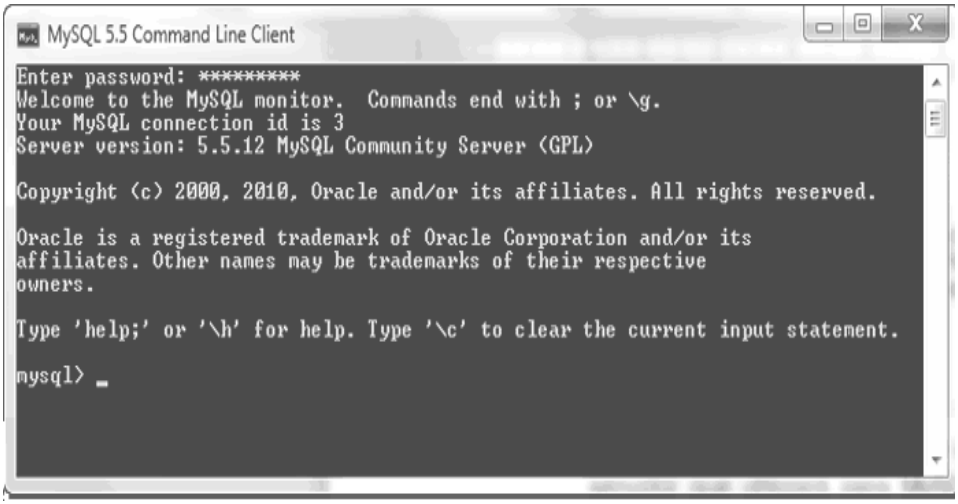


Figura 5.3 El entorno de comandos de MySQL.

9. En esta ventana de comandos podría crear nuestras bases de datos, con sus respectivas tablas y demás operaciones de SQL. Sin embargo, aproveche la tecnología de los Front-end e instale uno que sirva para los mismos propósitos pero desde un ambiente gráfico. De esta instalación trata la siguiente sección.

Actividades para el lector

Descargue el IDE (front end) *dbforgemysqlfree* que se utilizará para administrar MySQL, desde el sitio web :
http://www.devart.com/dbforge/mysql/studio/
proceda a configurarlo, tal como se indicó en esta sección: Instalación de MySQL.

Instalación de Front-end dbforgemysqlfree

Una vez instalado MySQL se dará cuenta que MySQL es un back-end, es decir, la parte del software que procesa la entrada a través de comandos o de una interfaz, en este caso, desde una consola o de un front-end. El front-end, por el contrario, es el software que interactúa con el usuario y procesa todas las instrucciones que éste le dé con un back-end. En realidad, entre estos dos conceptos se presenta una abstracción que ayuda a mantener la separación entre la complejidad de lo que se hace y como se hace. La interfaz que se utilizará como front-end, es decir, cómo se comunicará con MySQL será a través del software dbforgemysqlfree en su versión express. La puede descargar desde:

http://www.devart.com/dbforge/mysql/studio/

Inicie entonces con el proceso de instalación de este software.

1. Inicie la instalación de un Front-end para MySQL. Esta herramienta se denomina *dbforgemysqlfree* y es uno de los muchos que se encuentran en el mercado para soportar las operaciones de MySQL.
2. Dé doble clic sobre este archivo, el cual se incluye en este libro. La primera pantalla que se muestra es como la que se muestra en la figura 5.4.

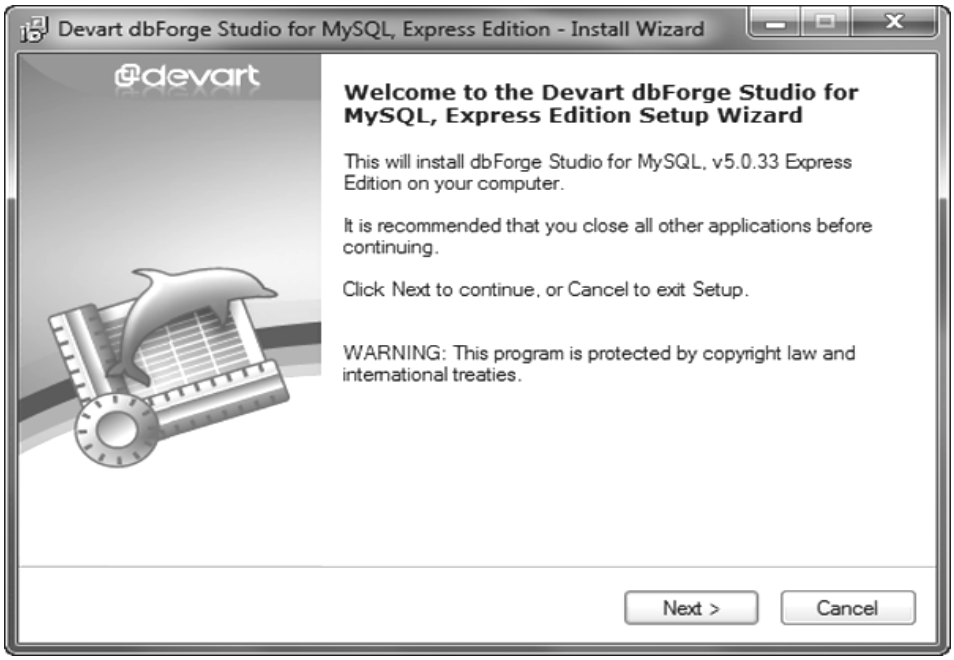


Figura 5.4 Instalación de Front-end para MySQL.

3. El asistente le llevará paso a paso por el instalador. Al final le solicitará si desea ejecutarla. Responda que sí y se mostrará una pantalla como la que se aprecia en la figura 5.5.

Nota

Observe la figura 5.5; verá que tiene varios campos que rellenar para poder acceder al servidor. Deberá completar esos campos tal y como se muestra en la figura. Podría eventualmente tener otros valores de configuración.

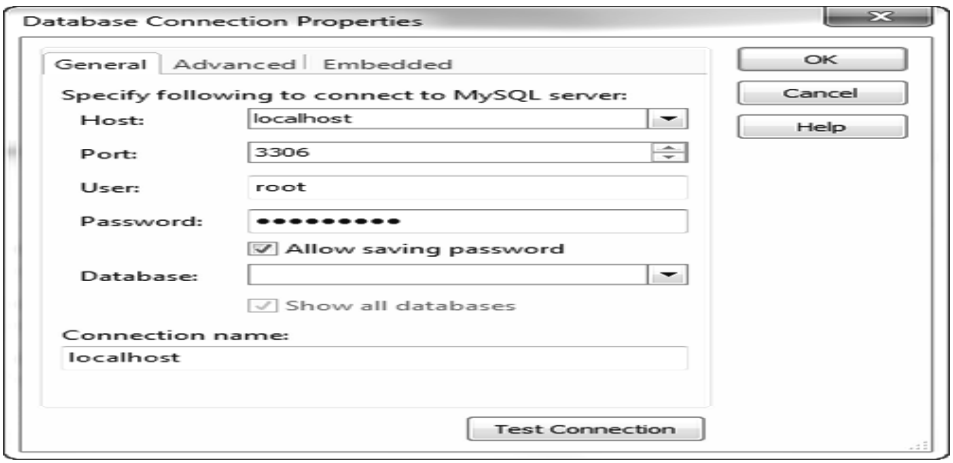


Figura 5.5 El entorno de dbforgemysqlfree.

4. Cuando haya escrito los valores que se indican en la figura 5.5 puede pulsar sobre el botón *Ok* o hacer una prueba de conexión pulsando *Test Connection*. Upsss! ¿Qué pasó? Tal parece que hay un problema… Bueno, pues en cierto modo es correcto… no se puede conectar al servidor. El error se muestra en la figura 5.6.

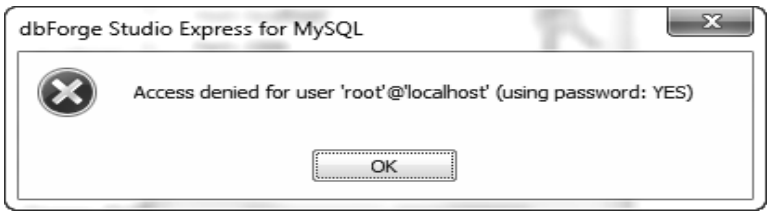


Figura 5.6 Error de conexión a MySQL.

5. Para solucionar el problema anterior debe ejecutar MySQL en su entorno texto y darle derechos al usuario que manipulará el servidor de base de datos. La figura 5.7 muestra este proceso.

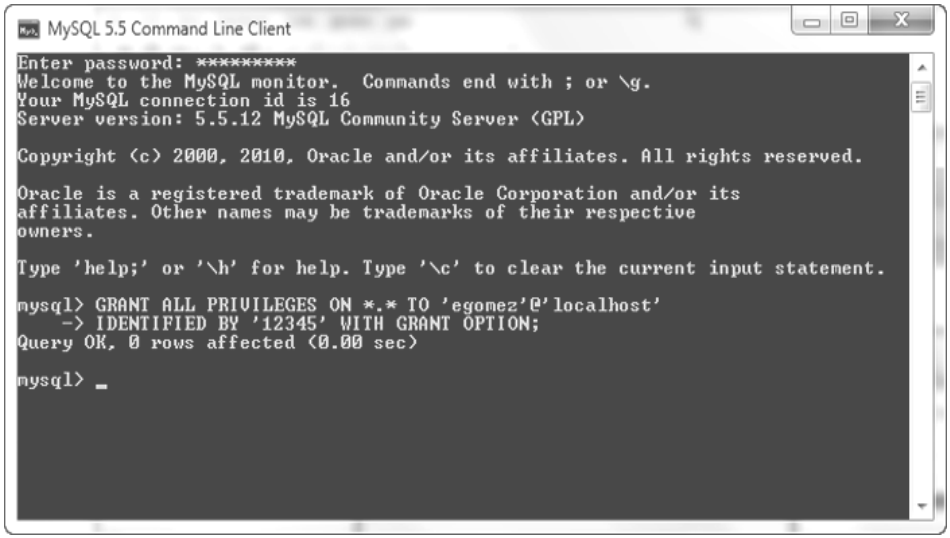


Figura 5.7 Configuración del acceso del administrador a MySQL.

6. Observe la siguiente leyenda de comando que se incluyó en el editor de MySQL. Se crea el usuario *egomez* en el servidor *localhost* con todos los privilegios. La contraseña asignada para este usuario es "12345".

Crear una base de datos con dbforgemysqlfree

1. Ingrese nuevamente a *dbforgemysqlfree.* Conéctese al servidor tal como se muestra en la figura 5.8.

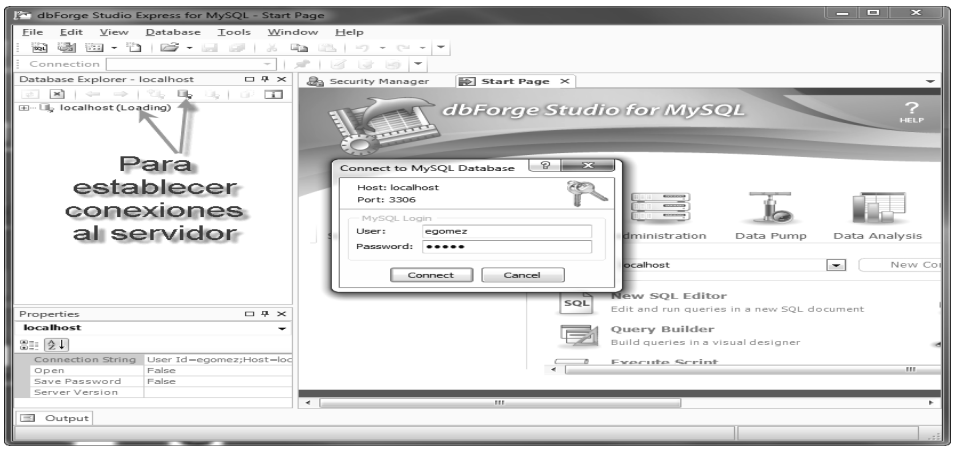


Figura 5.8 Conectando con MySQL.

2. Una vez conectado crearemos nuestra primera base de datos. Se denominará *dbUniversidad*. Para crear una base de datos debe pulsar sobre el nombre de la conexión y en el menú de contexto que aparece seleccionar la opción *New Database…* como se muestra en la figura 5.9.

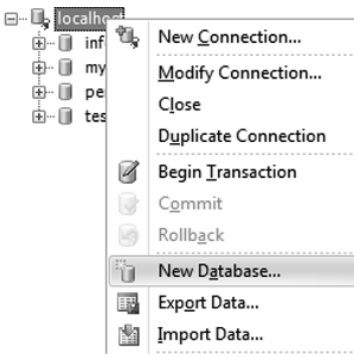


Figura 5.9 Creación una base de datos MySQL.

3. Una vez creada la base de datos *dbUniversidad* se crean las tablas que contendrá. Por el momento se creará una tabla denominada *tbEstudiantes*. Para ello pulse dos veces sobre el nombre de la base de datos para que expanda. Una vez expandida pulse con el botón derecho del mouse sobre *Tables* y seleccione la opción *New Table…* Observe la figura 5.10 donde se muestra la forma de crear una tabla y agregar un campo a la misma.

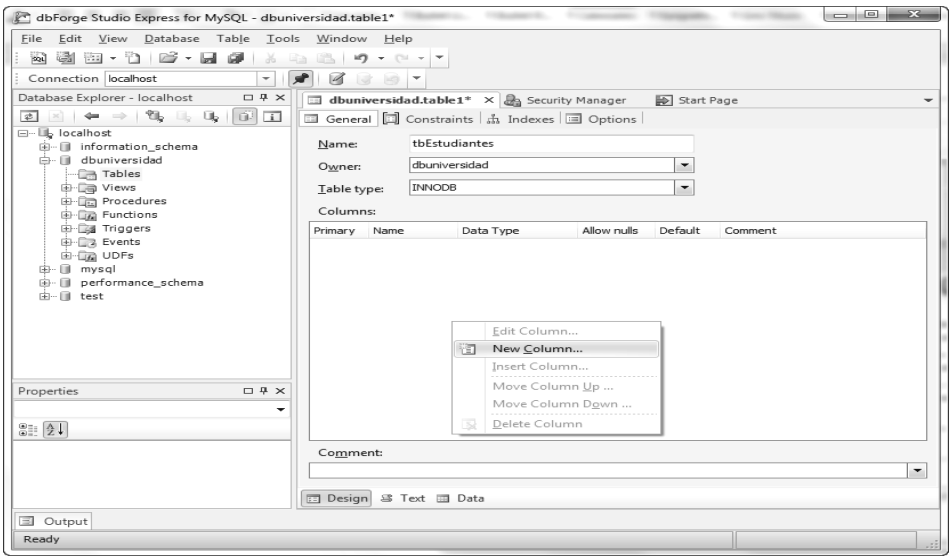


Figura 5.10 Agregando una tabla a una base de datos en MySQL.

Observe que se ha nombrado una tabla *tbEstudiantes* y posicionándonos sobre el campo debajo de *Columns* se puede crear los campos de la tabla. La tabla 5.11 muestra la forma de crear estos campos.

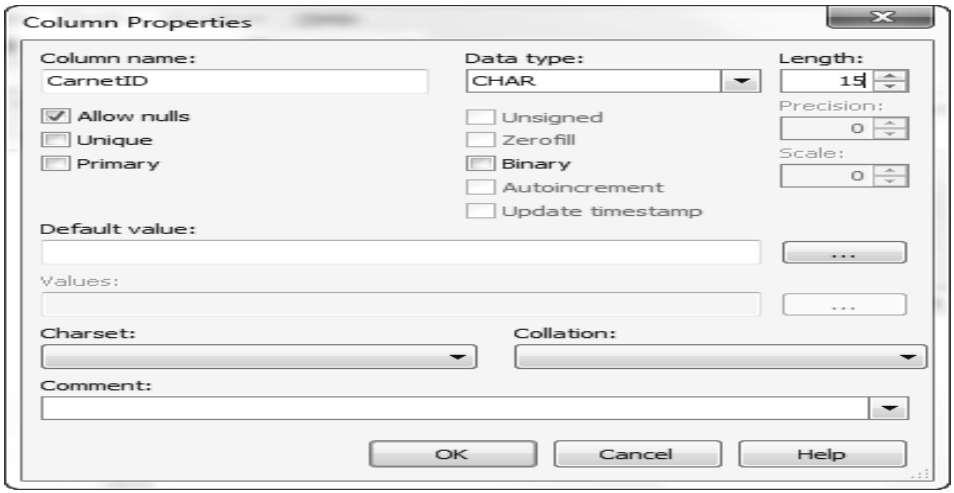


Figura 5.11 Creación de los campos de la tabla.

4. Debe terminar de incluir todos los elementos de la tabla, quedando el esquema final como el que muestra la figura 5.12.

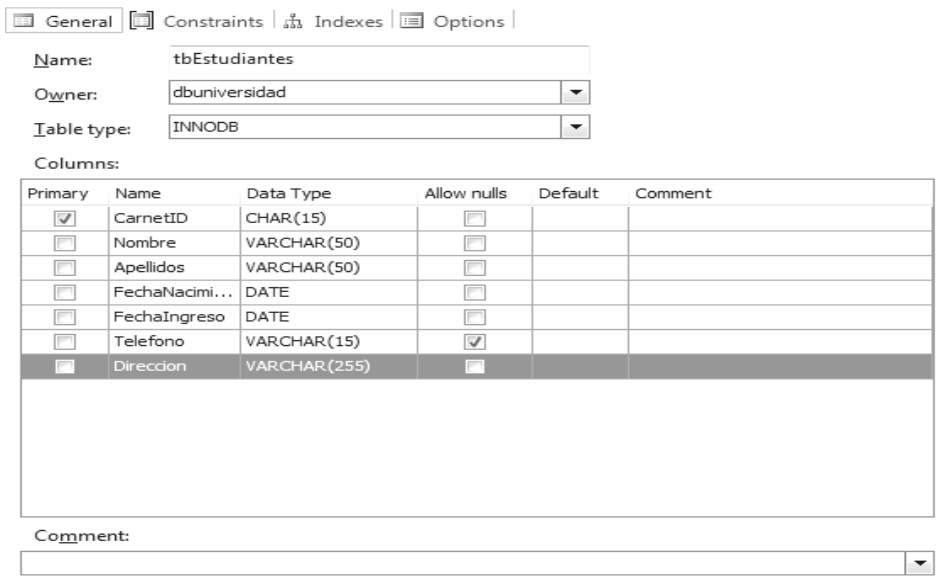


Figura 5.12 Esquema final de la tabla.

5. Cierre el administrador de bases de datos y proceda a abrir NetBeans para trabajar una aplicación que demuestre cómo gestionar datos desde este IDE.

Actividades para el lector

- Descargue el IDE (front end) *dbforgemysqlfree* que se utilizará para administrar MySQL, desde el sitio web http://www.devart.com/dbforge/mysql/studio/
- Configure *dbforgemysqlfree* tal y como se indica en la sección *Instalación de Front-end dbforgemysqlfree.*

Gestión de datos con NetBeans en MySQL

La persistencia de datos se refiere a la capacidad de los mismos a permanecer en la memoria de un computador. Es decir, los datos tienen una vida efímera en una aplicación y por tanto su persistencia se ve amenazada una vez que se cambian sus valores en la memoria de la máquina. A partir de ese momento su persistencia se pierde.

En Java la persistencia también ha sido un tema de estudio. A finales del 2001 existían cuatro estándares: *serialización*, *JDBC (Java DataBase Connectivity)*, *SQLJ* y *ODMG* adaptado a Java.

Serialización. Su objetivo es preservar la integridad referencial de los objetos, con la desventaja que no soportan la concurrencia de muchos usuarios accesando los datos.

JDBC. En este contexto JDBC le da el trabajo al programador para que maneje el estado de los objetos según la manera en que se proyecten en las relaciones que establezcan con el sistema administrador de la base de datos.

SQLJ Su propósito es que Java embeba el código SQL estático en la programación, facilitando el desarrollo de aplicaciones orientadas a datos.

Ninguno de los tres estándares anteriores trata satisfactoriamente la persistencia. Se afirma que la implementación ODMG constituye la mejor tecnología disponible para resolver el problema de la persistencia en Java, inclusive sobre un RDBMS (Relational DataBase Management System) y OODBMS (Object Oriented DataBase Management System).

Centraremos nuestro interés en JDBC para el manejo de la persistencia de datos en un RDBMS. Mediante JDBC se puede interactuar con una base de datos relacional como MySQL, Oracle, SQL Server, entre otras. Esta API está integrada a la plataforma por lo que JVM (Java Virtual Machine) utiliza cualquier driver de RDBMS para invocar la base de datos correspondiente.

Arquitectura JDBC

Se ha comentado que JDBC es una API que permite la gestión de bases de datos desde aplicaciones Java. Es independiente del sistema operativo sobre el que se ejecute o la base de datos que emplee, siempre y cuando se use el driver adecuado. El SQL es el dialecto que se utilizará formalmente en la gestión de la base de datos.

A continuación se ve la arquitectura Java en la figura 5.13

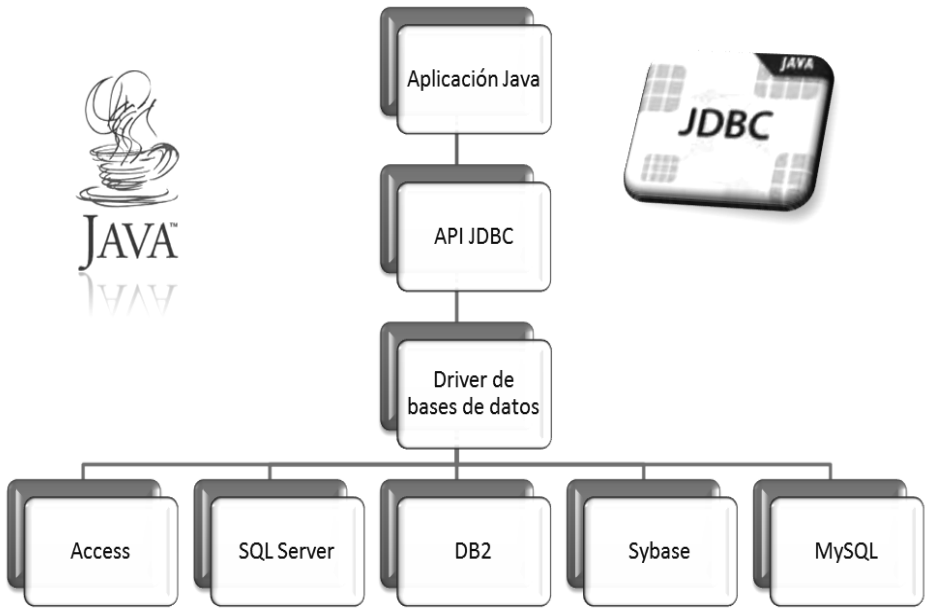


Figura 5.13 Arquitectura JDBC.

Arriba de la estructura se puede observar esta la aplicación JAVA, y bajo ella la API JDBC que permite la interacción entre la aplicación y el driver de las bases de datos. Debajo de la API JDBC encontrará los diferentes driver de base de datos que sean compatibles con Java, tales como MySQL, Sybase, DB2, SQL Server, Access, entre otros.

Para hacer funcionar una aplicación Java el programador implementa el driver de la base de datos requerida en dicha aplicación y con ello es posible realizar cualquier operación permitida sobre la base de datos, tal como actualización, creación, borrado, consulta, entre otras.

Cada diseñador de base de datos aporta el driver para integrarse al JDBC. Existen cuatro tipos de estos driver, a saber:

JDBC – ODBC Bridge. Este driver funciona traduciendo invocaciones JDBC a invocaciones ODBC mediante librerías ODBC del sistema operativo. No se considera una buena solución pero funciona. Un ejemplo es Microsoft® Access.

El puente JDBC – ODBC permite acceder cualquier base de datos, dado que los controladores ODBC de la base de datos se encuentran disponibles.

Entre las desventajas de este controlador se tiene que no está totalmente escrito en Java, por ende no es portable. También el rendimiento, el cual se produce cuando una llamada JDBC debe pasar por el puente hacia el controlador ODBC y posteriormente a la base de datos, produciéndose el efecto inverso en la devolución de la llamada.

Se requiere instalación del cliente ODBC para utilizar el controlador y no se tiene como una buena solución para aplicaciones web.

API parcialmente nativo de Java. Es similar al anterior, con la diferencia que no realiza el proceso de traducción mediante el puente – controlador. Ofrecen un mejor rendimiento que el controlador anterior y puede afirmarse que utiliza API nativa específica para cada base de datos.

Como desventaja puede citarse que el API nativo debe instalarse en el cliente, es inútil su uso para aplicaciones web. La portabilidad no existe para aplicaciones Java con este controlador, dado que no está escrito en Java. Es un controlador de por sí ya obsoleto.

JDBC Network Driver. Constituye un Back-end que accede al servidor de la base de datos. Este controlador se basa en servidor, por lo que no es necesario el controlador en el cliente. Además está escrito totalmente en Java, lo que lo hace portable y adecuado para aplicaciones web. Su única desventaja quizá es que se requiere un servidor de aplicaciones para la instalación y mantenimiento del controlador.

JDBC Driver. Son completamente escritos en Java, con lo que se logra la independencia de la plataforma y la eliminación de problemas de implementación y administración. Son adecuados para aplicaciones web. Se necesita un controlador para cada base de datos.

A continuación la figura 5.14 muestra la API de JDBC. Observe que existen muy pocas capas y componentes, lo que la hace muy versátil, sencilla pero poderosa a la vez.

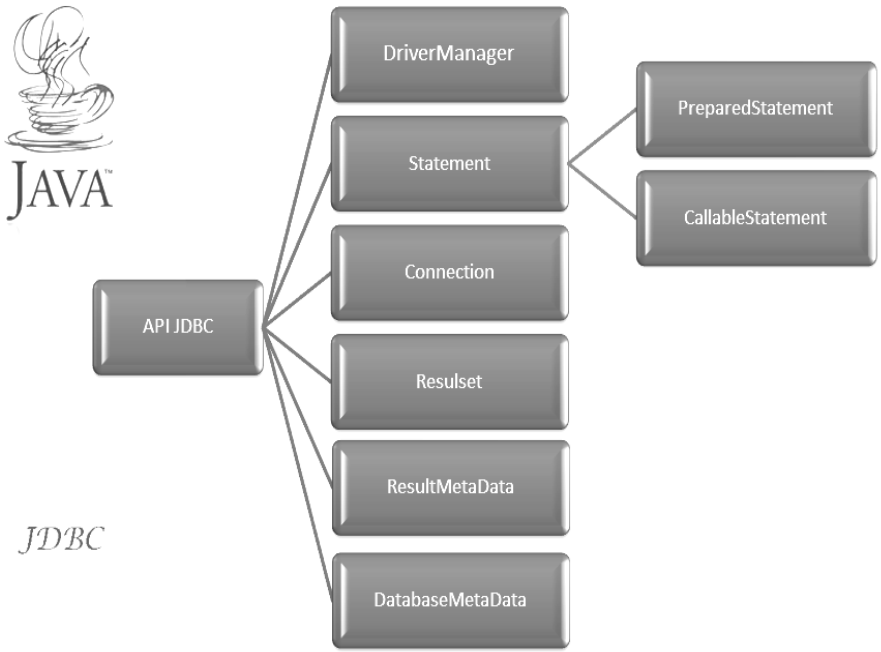


Figura 5.14 Arquitectura JDBC.

Los componentes de la API JDBC se describen en la tabla 5.1.

Tabla 5.1 Componentes de la API JDBC

COMPONENTE	FUNCIÓN
DriverManager	Se utiliza para cargar un driver
Statement	Se usa para crear consultas y enviarlas a la base de datos
Connection	Permite conectarse a una fuente de datos
Resulset	Se emplea para almacenar el resultado de la consulta
ResultMetaData	Contiene información acerca de las propiedades de cada una de las columnas que conforman la tabla de la base de datos o el cursor obtenido a través de una consulta
DatabaseMetaData	Contiene información acerca de la base de datos. Algunos métodos devuelven resultados en forma de tabla o Resulset y pueden ser usados como tal.

Conectividad a una base de datos

Para conectarse a una base de datos se necesitan los componentes que muestra la figura 5.14 y que se explican brevemente en la tabla 5.1. A continuación se detalla estos componentes para ir construyendo la lógica funcional de la API JDBC.

DriverManager Es una clase que se encarga de controlar la conexión y toda la comunicación con la base de datos. El DriverManager permite registrar el driver de la base de datos a utilizar en la máquina virtual de Java (JVM). JDBC supone que todos los drivers de la base de datos se registran automáticamente con el DriverManager cargado en el JVM. Los drivers vienen en clases empaquetadas en archivos JAR o ZIP, dependiendo del fabricante.

Los siguientes métodos se utilizan para cargar o registrar los drivers:

1. Agregando a la variable CLASSPATH el archivo que contiene las clases del driver de la base de datos a utilizar. Puede que también resulte agregarlo al directorio Jre\lib\ext ubicado en el directorio raíz del JDK Standar Edition.
2. Utilizando el método estático *forName()* contenido en la clase *Java.lang*. Algunos ejemplos pueden ser:

   a) Class.forName("com.sybase.jdbc2.jdbc.SybDriver"); para cargar el driver de la base de datos Sybase.
   b) Class.forName("oracle.jdbc.driver.OracleDriver"); para cargar el driver de la base de datos Oracle.
   c) Class.forName("com.mysql.jdbc.Driver"); para cargar el driver de MySQL

Ejemplo de aplicación. En el siguiente extracto de código se expone la forma en que se conjugan la carga del driver de una base de datos MySQL y su conectividad a la misma mediante el objeto *Connection*.

CARGA DE UN DRIVER MYSQL Y ESTABLECIMIENTO DE LA CONEXIÓN A LA BASE DE DATOS
static Connection conn=null; //declaro un objeto de tipo conexión.
static Statement stm = null; declaro un objeto de tipo sentencia SQL
static ResultSet rs = null; //declaro un cursor para almacenar los datos
static String SQL; //declaro una cadena para almacenar sentencias SQL
static String login = "egomez"; //establezco un nombre de usuario
static String pwd = "12345";
static String bd="dbuniversidad";
static String url ="jdbc:mysql://localhost:3306/dbUniversidad"; //establezco

CARGA DE UN DRIVER MYSQL Y ESTABLECIMIENTO DE LA CONEXIÓN A LA BASE DE DATOS
//el string que establece la conexión de la base de datos.
/**Creación de una función para establecer la conexión a la base de datos**/
public static Connection Enlace(Connection conn) throws SQLException
{
try {
Class.forName("com.mysql.jdbc.Driver"); //carga del driver...
conn = (Connection) DriverManager.getConnection(url,login,pwd); //Se //establece la conexión a través del DriverManager
} catch (ClassNotFoundException ex) {
javax.swing.JOptionPane.showMessageDialog(null,ex);
}
return conn;
}

Statement. Se utiliza para la creación de sentencias SQL estáticas que devuelven un resultado. En el caso de los resultados que devuelven un conjunto de registros, Statement se usa asociado al método *executeQuery()*. En este caso, los registros devueltos a través de la consulta deben ser cargados en un objeto *ResulSet*.

La sintaxis para crear un Statement es la siguiente:

```
Statement <nombre del Statement> = nombreconexion.createStatement();
```

Y para utilizar el Statement conjuntamente con un ResultSet:

```
ResulSet <nombre del ResulSet> = <nombre Statement>.executeQuery(SQL)
```

Ejemplo

```
Statement stm = conn.createStatement();
ResulSet rst = stm.executeQuery("Select *From tbEmpleados");
```

El Statement también puede procesar sentencias que signifiquen sólo una operación de ida hacia la base de datos. Es decir, que aplique una operación sobre la base de datos y no devuelva nada a la llamada de la misma. Es el caso de las operaciones *Insert, Update* y *Delete*. En este contexto se utiliza el método *executeUpdate()* con la sentencia SQL como argumento. La siguiente línea muestra esta situación:

```
Statement stm = conn.createStatement();
ResulSet rst = stm.executeUpdate("Insert Into tbEmpleados(Nombre,Apellidos)
Values ('Enrique','Gómez'))";
```

preparedStatement. Es un objeto que permite ejecutar sentencias precompiladas, es decir se ha construido su árbol de ejecución desde la primera vez que se ejecutó y se dinamizan en cuanto los valores de los parámetros que utiliza en la sentencia SQL que usa. Generalmente se emplea para ejecutar sentencias que utilizan Insert, Update o Delete, dado que son las que requieren condicionantes que se implementan desde la propia creación del objeto *PreparedStatement*. Para establecer esta condicionante se usa *setX()* donde X representa el tipo de datos del parámetro y luego se ejecuta la sentencia SQL. Las dos siguientes líneas muestran un ejemplo de aplicación:

```
preparedStatement pstmt = conn.preparedStatement("Update tbEmpleados set
salario = salario * 1.05 set codigo = ?")
pstmt.setInt(1,12345);
int resultado = pstmt.executeUpdate();
```

JDBC en la sentencia SQL anterior sustituye el signo de interrogación (?) por el tipo y contenido de datos en el argumento pstmt.setInt(1,12345)

Cabe mencionar que *PreparedStatement* también permite la ejecución de sentencias SQL que devuelven un conjunto de registros a través de un Select, considerando también el uso de parámetros. En este caso, con el *executeQuery* explicado líneas arriba. Un ejemplo práctico de esta condición es la siguiente:

```
preparedStatement pstmt = conn.prepared.Statement("Select *From
tbEmpleados Where Departamento = ?");
pstmt.setString(1,"01");
ResultSet rst = pstmt.executeQuery();
```

En relación con los parámetros, se pueden utilizar tantos como se requieran en la sentencia SQL. Por ejemplo:

```
SQL = "SELECT *FROM tbEmpleados WHERE Ciudad = ? And Salario > ?";
smt.setString(1,"La Cruz");
smt.setDouble(1,50000);
```

Los tipos de datos para los parámetros son setBoolean,setDate,setDouble, setFloat, setString y setInt.

CallableStatement. Con él se permite la ejecución de proce-dimientos almacenados (stored procedures) que se encuentran creados en una base de datos. JDBC establece una sintaxis única para que todos los procedimientos almacenados de cualquier base de datos sean tratados de la misma manera. CallableStatement puede retornar también objetos ResultSet.

Por ejemplo, se puede tener un procedimiento almacenado en la base de datos dbUniversidad, de tal forma como:

EJERCICIO: CREAR UN PROCEDIMIENTO ALMACENADO EN MYSQL
CREATE DEFINER = 'egomez'@'localhost'
PROCEDURE dbuniversidad.spConsultarEstudiante(IN _CarnetID CHAR(10))
BEGIN
SELECT *FROM tbestudiantes WHERE CarnetID = _CarnetID;
END

La ejecución desde Java, utilizando *CallableStatement* se hará de la siguiente manera:

EJERCICIO: EJECUCIÓN DE CALLABLESTATEMENT INVOCANDO EL PROCEDIMIENTO ALMACENADO
CallableStatement cStmt = (CallableStatement) conn.prepareCall("{call spConsultarEstudiante(?)}");
cStmt.setString(1, txtCedula.getText());
cStmt.execute();
rs = cStmt.getResultSet();
while(rs.next()){
...
}

ResultSet. Mediante ResultSet se obtienen los resultados de la ejecución de una consulta usando Statement. Se asocia con la sentencia Select de SQL dado que es la única que produce un resultado (registros que resultan de un Select). Además se consideran los procedimientos almacenados que también devuelven datos como resultados. Este objeto se vale de un cursor que contiene los datos obtenidos en una determinada consulta. Sólo permite moverse de registro en registro hacia adelante y no es actualizable en forma predeterminada.

Mediante ResultSet se obtienen los resultados de la ejecución de una consulta usando Statement. Se asocia con la sentencia Select de SQL dado que es la única que produce un resultado (registros que resultan de un Select). Además se consideran los procedimientos almacenados que también devuelven datos como resultados. Este objeto se vale de un cursor que contiene los datos obtenidos en una determinada consulta. Sólo permite moverse de registro en registro hacia adelante y no es actualizable en forma predeterminada.

Al igual que PreparedStatement el ResultSet usa *setX()* donde X representa el tipo de datos del parámetro y luego se ejecuta la sentencia SQL. Algunas características de un ResultSet son las siguientes:

- Es el mismo comportamiento de un cursor de datos.

- Contiene los métodos que permiten el acceso a los datos resultantes de la ejecución de un SELECT.
- El puntero del cursor se posiciona antes de la primera fila y para moverse entre los registros se utiliza *ResultSet.next()*
- Si se requiere obtener una columna específica de una fila se utiliza el método *ResultSet.getX* donde x indica el tipo de datos a extraer.

Un ejemplo de aplicación de *ResultSet* se muestra a continuación:

EJEMPLO DE IMPLEMENTACIÓN DE UN RESULTSET EN NETBEANS 7.1
String SQL="SELECT * FROM tbEstudiantes"
Statement stmt = conn.createStatemet();
ResultSet resultado = stmt_.executeQuery(SQL);
while(resultado_.next()){
System.out.println(" Carnet : "+resultado.getString("IDCarnet"));
System.out.println(" Nombre : "+resultado.getString("Nombre"));
}

Observe que en la primera línea se crea un SQL que permitiría obtener todo el listado de estudiantes que se encuentren en la tabla tbEstudiantes. La siguiente línea crea un objeto Statement en conexión actual establecida en el objeto conn. En la tercera línea se declara el ResultSet resultado, el cual obtiene los datos ejecutando el Statement stmt (stmt ejecuta la consulta según el contenido en la variable SQL). Finalmente, mediante un ciclo while se obtienen todos los datos.

Hasta aquí se ha explicado someramente la tecnología JDBC. Ahora finaliza el capítulo con su práctica mediante la creación de un ejemplo de aplicación. Para ello se utilizará la base de datos dbUniversidad que se creó al iniciar este capítulo, se da por hecho que esa parte ya está creada. Ahora proceda a crear una aplicación demostrativa.

EJEMPLO 1: CREAR UN FORMULARIO PARA MANTENIMIENTO DE DATOS

1. En el explorador de Windows (en este caso Windows 7) se crea una carpeta denominada *dskUniversidad.*
2. Lo primero que se debe hacer es descargar un archivo JAR que permita manejar las fechas que utilizaremos en nuestros registros. Para ello descargue el siguiente archivo jdatechooser.JAR de la dirección:

   http://jdatechooser.sourceforge.net/

Guárdelo en una carpeta tal como dskUniversidad o en aquella que crea le será fácil ubicar para utilizar el componente descargado.

3. Una vez descargado se debe crear un proyecto NetBeans, pero esta vez será de tipo escritorio (*desktop*), tal y como se muestra en la figura 5.15. La carpeta que utilizaremos para guardar este proyecto será la que recién creamos, *dskUniversidad.*

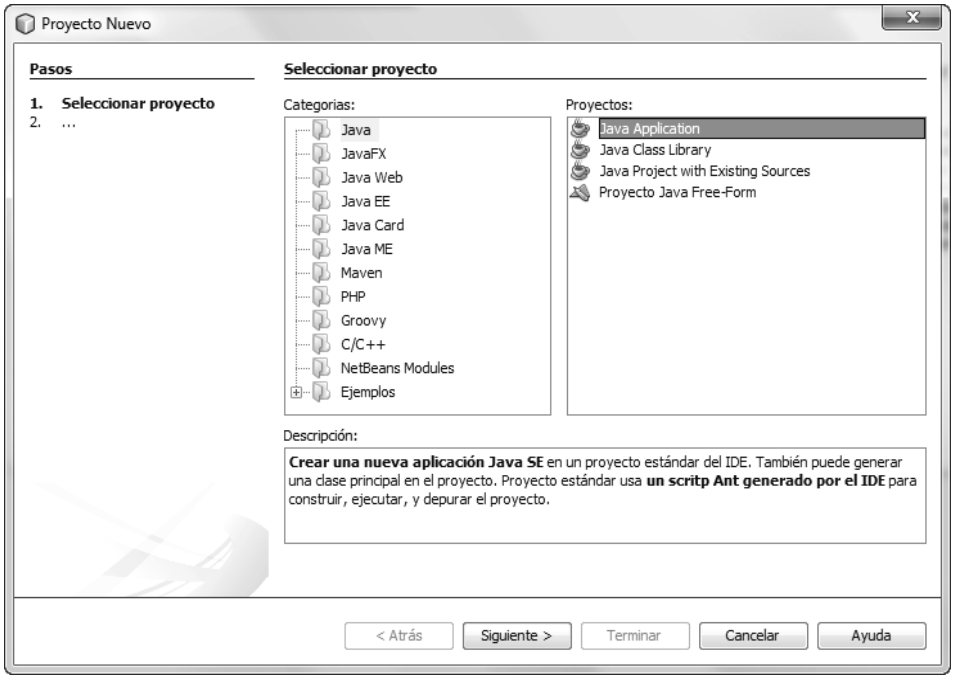


Figura 5.15 Proyecto tipo escritorio (desktop) NetBeans.

4. Pulse el botón *Siguiente*.
5. En la pantalla *Renuncia* que sigue pulse de nuevo el botón *Siguiente*.
6. En la pantalla *Nombre* y *Ubicación* nombre *dskUniversidad* al proyecto y ubíquelo en la carpeta que se creó en el punto 1 de este ejercicio. Pulse el botón *Terminar*.
7. Se abrirá el entorno para desarrollo NetBeans con un formulario mostrando las opciones *File* y *Help*.
8. Ubíquese sobre la opción que dice *File* en el menú del formulario recién creado y pulsando con el botón derecho del mouse seleccione la opción *Editar Texto*. Cambie el texto *File* por *Mantenimiento.*

9. Posiciónese sobre el nombre del proyecto y cree un nuevo formulario *JFrame* de nombre *frmEstudiantes.* Más adelante estableceremos cuáles son los objetos que contendrá este formulario y su respectiva configuración.
10. Asimismo, deberá crear debajo de del menú *Mantenimiento* el submenú *Estudiantes*. En esta opción se programará la siguiente instrucción en el evento *jMenuItem1ActionPerformed*:

```
frmEstudiantes frm = new frmEstudiantes();
frm.setVisible(true);
```

11. Antes de comenzar a colocar los objetos de este formulario se debe cargar el componente fecha que se descarga según las instrucciones del punto 2. Para ello, inicie pulsando sobre el menú Herramientas, tal y como se muestra en la figura 5.16.

Figura 5.16 Agregar un componente JAR a la paleta de componentes de NetBeans.

12. Al pulsar sobre la opción seleccionada (figura 5.16) aparecerá una pantalla como la de la figura 5.17. Ahí deberá pulsar sobre el botón *Añadir de archivo JAR...*

Figura 5.17 Seleccionando el archivo JAR que contiene el componente.

13. En la siguiente pantalla que se muestra busque el archivo JAR que descargó según el punto 2. Se denomina *DateChooser.JAR*. Una vez ubicado pulse *Siguiente*.
14. Aparece ahora una pantalla con las tres opciones de interface que ofrece el componente. Seleccione las tres opciones y pulse *Siguiente*. Debe marcar el primer elemento y manteniéndolo presionado con la tecla SHIFT pulse con el puntero del mouse sobre el tercero. De esta forma se marcarán los tres elementos.
15. En la pantalla que aparece a continuación seleccione que va a colocar los componentes en la opción de *Controles Swing*. Pulse los botones *Terminar* y *Cerrar*. Observe en la figura 5.18 cómo quedarían habilitados estos controles en la paleta de NetBeans.

Figura 5.18 Los controles DateChooser implementados en la paleta.

16. Ahora regresemos al formulario *frmEstudiantes* y procedamos a configurarlo según lo especificado en la tabla 5.2.

Tabla 5.2 Configuración de los objetos de nuestro ejercicio de aplicación.

OBJETO	TEXT	NOMBRE (NAME)
jLabel	Carnet:	jLabel1
jLabel	Nombre:	jLabel2
jLabel	Apellidos:	jLabel3
jLabel	Fecha de Nacimiento:	jLabel4
jLabel	Fecha de Ingreso:	jLabel5
jLabel	Teléfono:	jLabel6
jLabel	Dirección:	jLabel7
JTextField	Nada	txtCedula
JTextField	Nada	txtNombre
JTextField	Nada	txtApellidos
JTextField	Nada	txtFechaNacimiento
JTextField	Nada	txtFechaIngreso
JTextField	Nada	Telefono
dateChooserCombo	(fecha actual)	dtchFecha1
dateChooserCombo	(fecha actual)	dtchFecha1
JTextArea	Nada	txtDireccion
JButton	Agregar	btnAgregar
JButton	Modificar	btnModificar
JButton	Eliminar	btnEliminar
JButton	Ejecutar SP	btnSP

17. Una vez que se ha configurado los objetos el formulario quedará similar al que muestra la figura 5.19.

Figura 5.19 Formulario para el mantenimiento de datos de estudiantes.

Observe que los botones Agregar, Modificar y Eliminar aparecen deshabilitados. Para ello pulse con el botón derecho del mouse sobre cada uno de estos botones y en Propiedades busque la propiedad Enabled y deshabilite el check que lo configura como Enabled = true.

18. Ahora se muestra todo el código que deberá contener este formulario:

CÓDIGO DEL FORMULARIO MANTENIMIENTO DE ESTUDIANTES
public class frmEstudiantes extends javax.swing.JFrame {
static Connection conn=null;
static Statement stm = null;
static ResultSet rs = null;
static String SQL;
static String login = "egomez";
static String bd="dbuniversidad";
static String pwd = "12345";

CÓDIGO DEL FORMULARIO MANTENIMIENTO DE ESTUDIANTES
static String url ="jdbc:mysql://localhost:3306/dbUniversidad";
String Fecha;
public static Connection Enlace(Connection conn) throws SQLException {
try {
Class.forName("com.mysql.jdbc.Driver");
conn = (Connection) DriverManager.getConnection(url,login,pwd);
} catch (ClassNotFoundException ex) {
javax.swing.JOptionPane.showMessageDialog(null,ex); }
return conn; }
//el siguiente evento es por si el usuario pulsa ENTER en el objeto txtCedula
private void txtCedulaKeyPressed(java.awt.event.KeyEvent evt) {
if (evt.getKeyCode() == KeyEvent.VK_ENTER) {
if (buscarRegistro()==true) actualizar();
else nuevo(); } }
void actualizar(){
btnAgregar.setEnabled(false);
btnModificar.setEnabled(true);
btnEliminar.setEnabled(true); }
void nuevo(){
btnAgregar.setEnabled(true);
btnModificar.setEnabled(false);
btnEliminar.setEnabled(false); }
String formatearFecha(String fecha) {
String fecha1;
String cad =fecha;
int longi = cad.length();
//"23-01-1965 debe grabarse 1965-01-23
fecha1 = cad.substring(8,longi)+"-"+cad.substring(5,longi-3)+ "-"+cad.substring(0,longi-6);
return(fecha1);}
String desformatearFecha(String fecha) {
String fecha1;

CÓDIGO DEL FORMULARIO MANTENIMIENTO DE ESTUDIANTES
String cad = fecha;
int longi = cad.length();
//"23-01-1965 debe almacenarse 1965-01-23
fecha1 = cad.substring(6,longi)+"-"+cad.substring(3,longi-5)+ "-"+cad.substring(0,longi-8);
return(fecha1); }
void deshabilitarBotones() {
btnAgregar.setEnabled(false);
btnModificar.setEnabled(false);
btnEliminar.setEnabled(false); }
void limpiarCajasTexto(){
txtCedula.setText("");
txtNombre.setText("");
txtApellidos.setText("");
txtTelefono.setText("");
txtDireccion.setText("");}
void procesarDatos(String strSQL, int operacion) {
String mensaje="";
Try {
conn = Enlace(conn);
stm = conn.createStatement();
stm.execute(strSQL);
conn.close();
deshabilitarBotones();
limpiarCajasTexto();
switch(operacion) {
case 1: mensaje = "El Proceso de Inserción terminó exitosamente!!"; break;
case 2: mensaje = "El Proceso de Actualización terminó exitosamente!!"; break;
case 3: mensaje = "El Proceso de Eliminación terminó exitosamente!!"; break; }
javax.swing.JOptionPane.showMessageDialog(this,mensaje); }
catch(SQLException ex) {
javax.swing.JOptionPane.showMessageDialog(this,"Ocurrió el error: " + ex); }}
private void btnAgregarActionPerformed(java.awt.event.ActionEvent evt) {

CÓDIGO DEL FORMULARIO MANTENIMIENTO DE ESTUDIANTES
String Fecha1, Fecha2;
Fecha1 = desformatearFecha(txtFechaNacimiento.getText());
Fecha2 = desformatearFecha(txtFechaIngreso.getText());
SQL = "INSERT INTO + tbestudiantes(CarnetID,Nombre,Apellidos,FechaNacimiento," + "FechaIngreso,Telefono,Direccion) + VALUES('"+txtCedula.getText()+"','" + txtNombre.getText()+"','"+txtApellidos.getText()+"','"+Fecha1 + "','"+Fecha2+"','"+txtTelefono.getText()+"','" + txtDireccion.getText()+"')";
procesarDatos(SQL,1); }
private void btnModificarActionPerformed(java.awt.event.ActionEvent evt) {
String Fecha1, Fecha2;
Fecha1 = desformatearFecha(txtFechaNacimiento.getText());
Fecha2 = desformatearFecha(txtFechaIngreso.getText());
SQL = "UPDATE tbestudiantes SET Nombre ='"+txtNombre.getText() + "',Apellidos='"+txtApellidos.getText() + "',Direccion='"+txtDireccion.getText() + "',FechaNacimiento='"+Fecha1 + "',FechaIngreso='"+Fecha2 + "',Telefono='"+txtTelefono.getText() + "' WHERE CarnetID='"+txtCedula.getText()+"'";
procesarDatos(SQL,2);}
private void btnEliminarActionPerformed(java.awt.event.ActionEvent evt) {
SQL = "DELETE FROM tbestudiantes" + " WHERE CarnetID='"+txtCedula.getText()+"'";
procesarDatos(SQL,3); }
private void dateChooserCombo1OnCommit(datechooser.events.CommitEvent evt) {
txtFechaNacimiento.setText(dateChooserCombo1.getText());}
private void dateChooserCombo2OnCommit(datechooser.events.CommitEvent evt) {
txtFechaIngreso.setText(dtChFecha1.getText()); }
//Procedimiento para implementar CallableStatement (sobre SP)

CÓDIGO DEL FORMULARIO MANTENIMIENTO DE ESTUDIANTES
private void btnSPActionPerformed(java.awt.event.ActionEvent evt) {
try {
conn = Enlace(conn);
CallableStatement cStmt = (CallableStatement) conn.prepareCall("{call spConsultarEstudiante(?)}");
cStmt.setString(1, txtCedula.getText());
cStmt.execute();
rs = cStmt.getResultSet();
while(rs.next()) {
txtNombre.setText(rs.getString("Nombre"));
txtApellidos.setText(rs.getString("Apellidos"));
Fecha = formatearFecha(rs.getString("FechaNacimiento"));
txtFechaNacimiento.setText(Fecha);
Fecha = formatearFecha(rs.getString("FechaIngreso"));
txtFechaIngreso.setText(Fecha);
txtTelefono.setText(rs.getString("Telefono"));
txtDireccion.setText(rs.getString("Direccion"));}
conn.close();
} catch (SQLException ex) {
Logger.getLogger(frmEstudiantes.class.getName()).log(Level.SEVERE, null, ex);
}}
boolean buscarRegistro(){
boolean enc = false; //al inicio el registro no ha sido encontrado!
Try {
conn=Enlace(conn);
stm = conn.createStatement();
rs = stm.executeQuery("SELECT *FROM tbestudiantes" + " WHERE CarnetID='"+txtCedula.getText()+"'");
while(rs.next()) {
txtNombre.setText(rs.getString(2));
txtApellidos.setText(rs.getString(3));
Fecha = formatearFecha(rs.getString(4));

CÓDIGO DEL FORMULARIO MANTENIMIENTO DE ESTUDIANTES
txtFechaNacimiento.setText(Fecha);
Fecha = formatearFecha(rs.getString(5));
txtFechaIngreso.setText(Fecha);
txtTelefono.setText(rs.getString(6));
txtDireccion.setText(rs.getString(7));
enc = true; } //como se encontró se asigna true a la variable enc.
conn.close();}
catch(SQLException ex){
javax.swing.JOptionPane.showMessageDialog(this,"Ocurrió el siguiente error: " + ex); }
return enc; }
//En el main principal de la aplicación... es decir de frmEstudiantes
public static void main(String args[]) {
java.awt.EventQueue.invokeLater(new Runnable() {
public void run() {
new frmEstudiantes().setVisible(true);
}
});
}

Como se puede ver, existen muchas líneas en este código donde se aplican los temas que se ha desarrollando en la parte teórica de este capítulo. Vale aquí la pena resaltar aquellas líneas que no comprenda del código, busque el tema respectivo en la parte teórica de este capítulo, en las referencias web que citamos al final del mismo o realice su propia investigación, incentivando el autoaprendizaje.

Hay que enfatizar que no es tan fácil explicar cada detalle de la codificación que se desarrolla en un ejemplo como este. Sin embargo, la intención es ofrecer una codificación simple y comprensible para que el lector pueda entender el mismo y desarrollar su propio código posteriormente, incluyendo aspectos más complejos.

19. Una vez que ha completado el código podrá probar la funcionalidad del formulario. Una corrida típica de este ejemplo se muestra en la figura 5.20. Observe que la funcionalidad es básicamente que si el usuario pulsa la tecla ENTER sobre el campo de cédula, posterior a digitar un valor, se buscará en la tabla tbEstudiantes. Si lo encuentra podrá el usuario modificarlo o eliminarlo y si no entonces podrá eliminarlo. La misma funcionalidad aplica para el botón que ejecuta el llamado al procedimiento almacenado, dado que busca usando como parámetro *txtCedula.getText().*

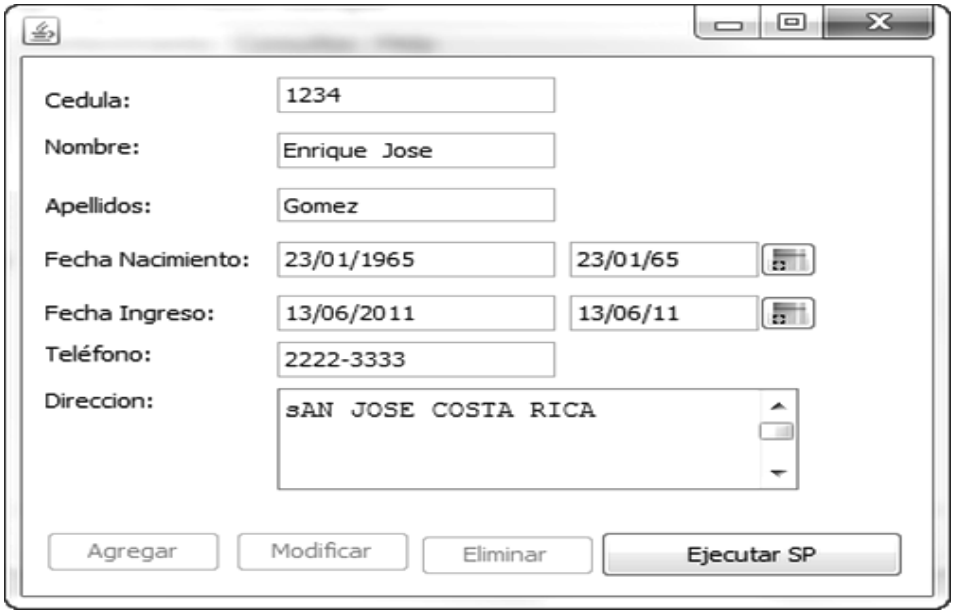


Figura 5.20 Mantenimiento de estudiantes en funcionamiento.

Ahora se creará un segundo y último ejemplo para este capítulo. Consiste en un formulario de consulta de los todos los estudiantes registrados o filtrados según su fecha de nacimiento.

Actividades para el lector

- Cree una nueva tabla en la base de datos dbUniversidad, denominada tbProfesores con los campos ProfesorID char (15), Nombre varchar(50), Apellidos varchar(100), Direccion varchar(200), Telefono varchar(20) y FechaIngreso DATE.
- Cree un nuevo formulario que permita el mantenimiento de la tabla tbProfesores. Básese en la práctica de creación de una base de datos en MySQL y el ejemplo, 1 que acaba de realizar.

EJEMPLO 2: CREAR UNA CONSULTA DE DATOS

1. Posiciónese sobre el nombre del proyecto y cree un nuevo formulario *JFrame* de nombre *frmQEstudiantes.* La figura 5.21 muestra gráficamente todos los objetos que contendrá este formulario, el nombre y la configuración inicial que tendrán

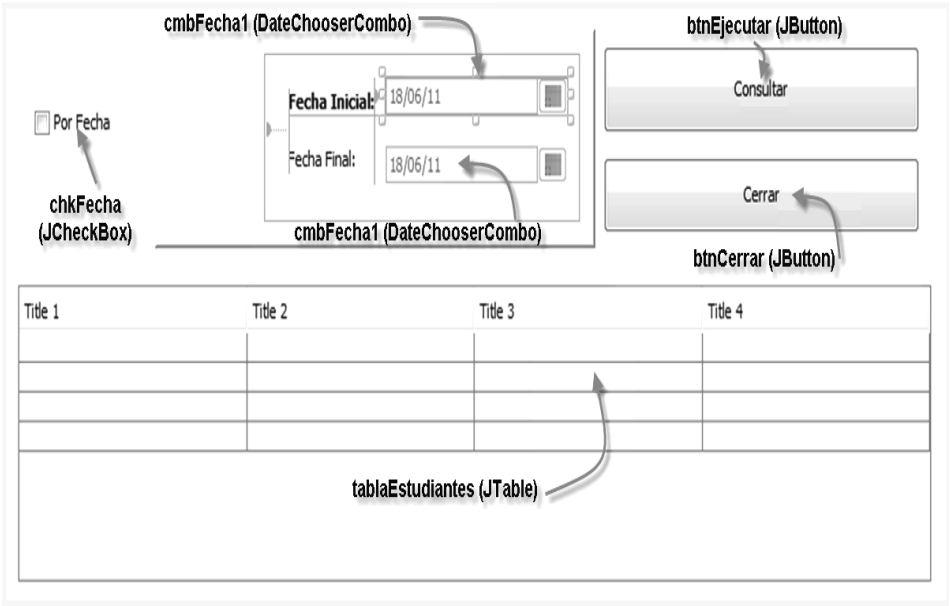


Figura 5.21 Configuración de los objetos de frmQEstudiantes.

Observe bien la figura, coloque los objetos tal y como se muestran en la misma y ponga los nombres que se indican con las flechas. El primer valor es el nombre del objeto y el segundo el tipo de objeto. Por ejemplo cmbFecha1 es el nombre de un objeto tipo DateChooserCombo, que fue el componente JAR que se colocó en este capítulo en la barra. Ambos controles deberán tener el check en su propiedad Enabled para que aparezcan inicialmente deshabilitados.

2. Ahora se creará una clase donde se demostrará que se puede implementar la lógica de gestión de datos (incluida la conexión a la base de datos de la aplicación). La clase se denominará *ConexionMySQL*. Para crear esta clase pulse con el botón derecho del mouse sobre el nombre del proyecto y pulse *Nuevo* y luego *Clase Java…* Ponga el nombre que indicamos.
3. Una vez creada la clase *ConexionMySQL* procede a escribir el siguiente código:

EJERCICIO: CREANDO NUESTRA CLASE CONEXIONMYSQL
package dskuniversidad;
import java.sql.*;
import javax.swing.JOptionPane;
public class ConexionMySQL {
public String baseDato = "dbUniversidad";
public String url = "jdbc:mysql://localhost/"+baseDato;
public String usuario = "egomez"; public String clave = "12345";
public Statement stm = null;
public ResultSet rs = null;
public ConexionMySQL(){ }
public Connection Conectar() {
Connection enlace = null;
try {
Class.forName("com.mysql.jdbc.Driver");//Cargar el Driver MySQL

//Enlazamos la base de datos
enlace = DriverManager.getConnection(this.url, this.usuario, this.clave); }

catch (Exception e)
{ JOptionPane.showMessageDialog(null, e); }
return enlace; }}

4. Sólo se ha implementado la clase con una función de conexión a la base de datos, pero bien se pueden agregar las funciones de mantenimiento de datos. Queda para que lo realice como práctica.
5. Ahora se escribirá el código de la aplicación. Se escribirá con su documentación para que se comprenda la funcionalidad de cada línea escrita en el programa.

EJERCICIO: CÓDIGO DE FRMQESTUDIANTES
package dskuniversidad;
import com.mysql.jdbc.Connection;//librería para crear la conexión.
import com.mysql.jdbc.Statement; //librería para crear la sentencia SQL
import java.sql.ResultSet; //librería para cargar los resultados de la consulta
import java.sql.SQLException; //librería para manejar las excepciones MySQL
import java.util.logging.Level;//librería para configurar nivel de trazas del debug

EJERCICIO: CÓDIGO DE FRMQESTUDIANTES

import java.util.logging.Logger;//mecanismo genérico para registrar mensajes
//de log.

import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;
//la siguiente línea es automática. Es la creación del formulario frmQEstudiantes
public class frmQEstudiantes extends javax.swing.JFrame {
//servirá para modelar la tabla JTable. DefaultTableModel modelo = new DefaultTableModel();
//variable general para crear una cadena con instrucciones SQL String SQL="";
//objeto de tipo ResultSet donde se cargarán los datos de la consulta ResultSet rs;

//se utiliza para generar una sentencia SQL que se cargará en un ResulSet.
//También se puede utilizar para una actualización de la base de datos
Statement st;

//se crea un vector de 7 posiciones de tipo cadena, llamado registro. String[] registro = new String[7];
//se crea una instanciación de la clase ConexionMySQL ConexionMySQL miSQL = new ConexionMySQL();
//a continuación un método para limpiar el modelo y así también la tabla
public void Limpiar() {
modelo = new DefaultTableModel();
int filas = modelo.getRowCount(); //obtiene el número de filas del modelo
if (filas > 0) {
//remueve las filas del modelo para así actualizar el JTable for (int i = 0; i < filas; i++) { modelo.removeRow(0); }}
tablaEstudiantes.setModel(modelo);} //se pone el 0 la tabla
//el siguiente procedimiento configura los títulos de la table y llama a limpiar()
void Titulos(){
Limpiar();
//vector denominado cabecera con los títulos a colocar en JTable String[] cabecera = {"Carnet", "Nombre", "Apellidos", "Fecha Nacimiento", "Fecha Ingreso", "Telefono", "Direccion"};
modelo = new DefaultTableModel(null, cabecera); } //coloca los títulos en JTable

EJERCICIO: CÓDIGO DE FRMQESTUDIANTES
//a continuación método que se utilizará para cargar los datos en el JTable
//a continuación una modificación esencial en el método *desformatearFecha*
String desformatearFecha(String fecha) { //parámetro fecha
int longi = fecha.length(); //obtiene la longitud de la fecha recibida
String dia= fecha.substring(0,longi-8); //obtiene el número de día en String
String mes = fecha.substring(3,longi-5); //obtiene el número de mes en String
String anyo = fecha.substring(6,longi); //obtiene el número de año en String
if(dia.length()==1) dia="0"+dia;//si es sólo 1 carácter le antepone un cero (0)
if(mes.length()==1) mes="0"+mes;//si es sólo 1 carácter le antepone un cero (0)
fecha=anyo+"-"+mes+"-"+dia; //concatena año+mes+dia para usar en MySQL
return(fecha); } //retorna la fecha tal como se debe procesar en MySQL
void cargarDatos() {
Titulos();
Connection conn = (Connection) miSQL.Conectar();
try{
st = (Statement) conn.createStatement(); //se crea la sentencia en la conexión
rs = st.executeQuery(SQL); //se ejecuta la sentencia según la cadena SQL
while(rs.next()){ //se inicia el recorrido del ResultSet
//se inicia la carga de cada campo del ResultSet en el vector registro
registro[0] = rs.getString("CarnetID");
registro[1] = rs.getString("Nombre");
registro[2] = rs.getString("Apellidos");
registro[3] = rs.getString("FechaNacimiento");
registro[4] = rs.getString("FechaIngreso");
registro[5] = rs.getString("Telefono");
registro[6] = rs.getString("Direccion");
registro[0] = rs.getString("CarnetID");
registro[1] = rs.getString("Nombre");
modelo.addRow(registro); } //se agrega la fila al modelo de datos
tablaEstudiantes.setModel(modelo); //se agrega el vector al JTable
conn.close();} //se cierra la conexión

EJERCICIO: CÓDIGO DE FRMQESTUDIANTES
catch (Exception ex) { //captura de alguna excepción
JOptionPane.showMessageDialog(null, ex); } }
//a continuación un procedimiento para cargar todos los registros de la tabla
void cargarTablaGeneral(){

//se crea la cadena SQL. Nota la instrucción Date_format que se utiliza para
//configurar la salida de la fecha en formato día/mes/año
SQL = "SELECT CarnetID, Nombre, Apellidos,"
+ "Date_format(FechaNacimiento,'%d/%m/%Y') As FechaNacimiento,"
+ "Date_format(FechaIngreso,'%d/%m/%Y') As FechaIngreso, "
+ "Telefono, Direccion FROM tbestudiantes";

cargarDatos();se invoca el método cargarDatos, según la sentencia SQL
}
//la siguiente function carga los datos, de acuerdo a las fechas seleccionadas
void cargarPorFechas(){
String Fecha1 = desformatearFecha(cmbFecha1.getText());
String Fecha2 = desformatearFecha(cmbFecha2.getText());
SQL = "SELECT CarnetID, Nombre, Apellidos,FechaNacimiento," + "FechaIngreso, Telefono, Direccion " + "FROM tbestudiantes WHERE FechaNacimiento BETWEEN '" + Fecha1 + "' AND '" + Fecha2 + "'";
cargarDatos();}
//en el botón *btnConsultar* escribe el siguiente código
try {
if (chkFecha.isSelected()) cargarPorFechas(); //si está seleccionado *chkFecha*
else cargarTablaGeneral(); //sino que se carguen todos los registros
} catch (Exception ex) {
javax.swing.JOptionPane.showMessageDialog(this,"Ocurrió el error: " + ex); }
//el siguiente código va en el procedimiento Change del objeto chkFecha1
if (chkFecha.isSelected()) {
cmbFecha1.setEnabled(true); //habilita el combo cmbFecha1
cmbFecha2.setEnabled(true); } //habilita el combo cmbFecha2
else {
cmbFecha1.setEnabled(false); //deshabilita el combo cmbFecha1

EJERCICIO: CÓDIGO DE FRMQESTUDIANTES
cmbFecha2.setEnabled(false); } //deshabilita el combo cmbFecha2
//finalmente en el void main va el código para que se visualice el formulario
new frmQEstudiantes().setVisible(true);

6. El código anterior permite ejecutar el formulario creando dos escenarios de datos que se cargarán en el JTable: el primero es que si el objeto *chkFecha* tiene su propiedad *selected* habilitada, generará los registros que cumplan con la condición dada por un rango de fechas. De lo contrario se mostrarán todos los registros que contiene la tabla *tbestudiantes.* La figura 5.22 muestra el primer escenario y la 5.23 el segundo.

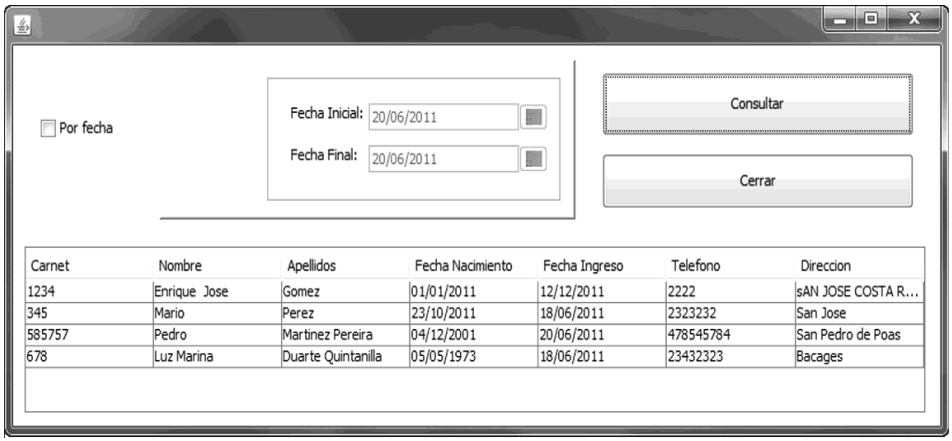


Figura 5.22 Ejecución del formulario con todo el listado de estudiantes.

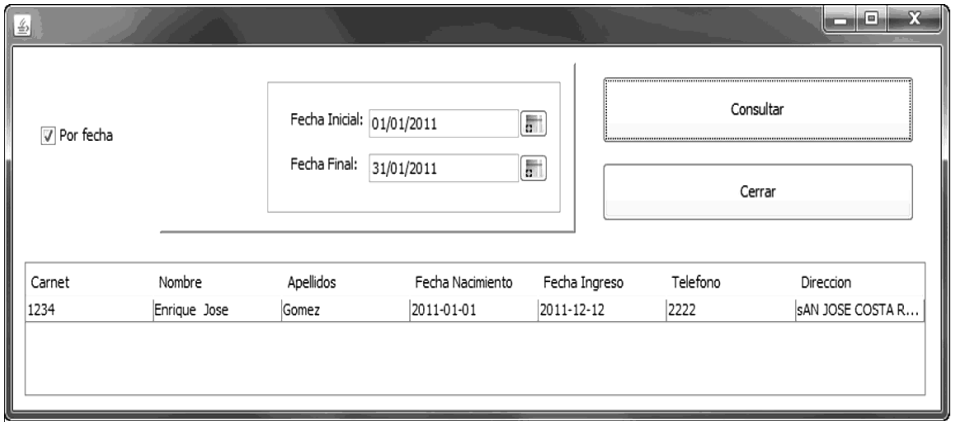


Figura 5.23 Ejecución del formulario filtrado por rango de fechas (fecha nacimiento).

Como podrá observar en la tabla aparecen las fechas en formato MySQL, mientras que nuestro control de fechas aparece en formato conocido. Internamente se programó para que ejecutara la consulta utilizando rangos de fecha que MySQL comprendiera y así pudiera resolver dicha consulta.

Con ello llegamos al final de este capítulo deseando que los conceptos hayan sido claros y concisos. Se aportan referencias web para que pueda reforzar los temas vistos en este capítulo y podrá descargar del sitio web de Alfaomega los ejemplos que desarrollamos aquí. También es importante comentar que puede establecer comunicación con el autor, con el fin de resolver los problemas o dudas que se presenten y que no pueda resolver con el libro, las referencias web o el acceso al sitio de Alfaomega. También puede usar este medio para comunicar las posibles mejoras que se puedan presentar a los programas o temas, los cuales podrán ser desarrollados en otra posible edición.

Actividades para el lector

Cree un formulario que permita generar una consulta que cumpla con los siguientes requerimientos:

- Permita consultar todos los profesores registrados en la tabla tbProfesores (listado general)
- Permita consultar todos los profesores que ingresaron a laborar en la universidad entre dos fechas (de acuerdo con FechaIngreso) mediante el uso de un procedimiento almacenado.

RESUMEN

En este capítulo se desarrolló el tema de la gestión de datos de NetBeans y MySQL. Describimos la parte conceptual de la arquitectura Java para la gestión de datos y cómo se implementa en MySQL. Básicamente se trataron los siguientes temas:

1. La instalación de MySQL como back-end y dbforgemysqlfree como front-end.
2. Cómo crear una base de datos con el front-end y el back-end.
3. La gestión de datos con NetBeans en MySQL.
4. Arquitectura JDBC.
5. Conectividad a una base de datos con NetBeans.
6. Ejemplos de aplicación.

Autoevaluación

1. ¿A qué se denomina back-end? Cite un ejemplo.
2. ¿Qué es un front-end? Cite un ejemplo.
3. ¿A qué se denomina persistencia de datos?
4. ¿Qué es serialización?
5. ¿Cómo funciona JDBC-ODBC-Bridge?
6. ¿Para qué se usa un objeto ResultSet?
7. ¿Para qué se usa un objeto Statement?

EVIDENCIA

☐ Implementó los ejercicios de ResulsetMetaData y DataBaseMetaData explicados en

http://www.chuidiang.com/java/mysql/ResultSet-DataBase-MetaData.php

☐ Desarrolló una aplicación que brinda mantenimiento a una base de datos denominada *dbPresupuesto*; usó las tablas *tbCuentas* y *tbMovimientos*.

☐ Elaboró un procedimiento almacenado en la base de datos *dbPresupuesto* y realizó un formulario que permita su utilización.

REFERENCIAS

Bibliografía

Ceballos, Fco. Javier, (2011). *Java 2: Curso De Programación*, 4a. ed., Alfaomega, México.

Martín, Antonio, (2010). *Programador certificado Java 2. Curso práctico*, 3a. ed., Alfaomega, México.

Rozannski, Uwe, (2010). Enterprise Javabeans 3.0. Con Eclipse Y JBoss. Alfaomega, México.

Sznajdleder, Pablo, (2010) *Java a fondo. Estudio del lenguaje y desarrollo de aplicaciones.* Alfaomega, México.

Páginas Web recomendadas

1. www.sc.ehu.es (2000) *Conceptos básicos de la Programación Orientada a Objetos*. Obtenido el 31 de octubre del 2011 desde http://www.sc.ehu.es/sbweb/fisica/cursoJava/fundamentos/clases1/clases.htm
2. slideshare.net (2011) *Java y bases de datos.* Obtenido el 31 de octubre del 2011 desde http://www.slideshare.net/javi2401/java-y-bases-de-datos-presentation.

3. webTaller (2011) *Conexiones Base de Datos en Java (3)* Obtenido el 31 de octubre del 2011 desde
   http://www.webtaller.com/construccion/lenguajes/java/lecciones/conexiones-db-java-3.php
4. herongyang.com (2011) *JDBC Tutorials - Herong's Tutorial Notes.* Obtenido el 31 de octubre del 2011 desde http://www.herongyang.com/JDBC/JDBC-ODBC-MS-Access.html
5. jdbc-tutorial.com (2008) *JDBC Driver Types.* Obtenido el 1 de noviembre del 2011 desde http://www.jdbc-tutorial.com/jdbc-driver-types.htm
6. jdbc (2011) *Qué es jdbc en Java.* Obtenido el 4 de noviembre del 2011 desde http://molten.latinclicks.info/JDBC.htm.
7. dev.mysql.com (2011) *Basic JDBC Concepts.* Obtenido el 4 de noviembre del 2011 desde http://dev.mysql.com/doc/refman/5.0/es/connector-j-usagenotes-basic.html
8. dev.mysql.com (2011) *Download MySQL Community Server.* Obtenido el 4 de noviembre del 2011 desde http://dev.mysql.com/downloads/mysql/5.0.html

Respuestas a la autoevaluación

1. Es la parte del software que procesa las entradas de datos a través de comandos o de una interfaz. Dbforgemysqlfree es un ejemplo de un software de tipo front-end.
2. Es la parte de software que interactúa con el usuario y procesa las instrucciones que éste le gire al back-end mediante una interfaz (software).
3. Se refiere a la capacidad de los mismos a permanecer en la memoria del computador.
4. Es el mecanismo de preservación de la integridad referencial de los objetos.
5. Funciona traduciendo invocaciones JDBC a invocaciones ODBC mediante librerías ODBC del sistema operativo.
6. Para almacenar el resultado de una consulta.
7. Para crear consultas y enviarlas a una base de datos.

Fundamentos de programación Web con NetBeans 7.1

6

Reflexione y responda las siguientes preguntas

¿Considera que las aplicaciones de escritorio son sustituidas rápidamente por las aplicaciones web?
¿El software libre constituye un aliado para el desarrollo de aplicaciones web de alta calidad?
¿Está nuestra vida y nuestro quehacer diario invadido por la necesidad de utilizar aplicaciones web?
¿En qué medida nos ha beneficiado o afectado el uso diario de aplicaciones web?

Contenido

EXPECTATIVA

En la actualidad las aplicaciones web representan una herramienta eficaz para realizar transacciones, obtener información o realizar algún tipo de negocio. La penetración de Internet en cada rincón de nuestro planeta y cada vez más aplicaciones web orientados a los negocios, a la información o al mundo del entretenimiento, hacen que se convierta en nuestro aliado. Conscientes de ello son cada vez más las empresas que se esfuerzan por crear soluciones para satisfacer las necesidades de los usuarios.

Como desarrollador web es importante conocer algunos fundamentos de la programación web y así, poco a poco, profundizar conocimientos y experiencias con el objetivo de convertirse en un creador exitoso de aplicaciones web. De manera que los fundamentos de la programación web constituyen la puerta de entrada para todos aquellos informáticos que desean penetrar en esta compleja pero excitante aventura de la creación web. Sin embargo, no sólo los informáticos están involucrados en el desarrollo web, sino que cada día más personas, de distintas disciplinas, participan de esta creatividad.

Después de estudiar este capítulo, el lector será capaz de

- Instalar y configurar un servidor web para implementar aplicaciones web utilizando NetBeans.
- Comprender los fundamentos de los objetos y métodos que permiten la comunicación a través de aplicaciones web.
- Conocer la fundamentación de las tecnologías basadas en lenguaje de marcas y su utilización en creación de sitios web.
- Comprender la funcionalidad de AJAX en el desarrollo de aplicaciones web.

INTRODUCCIÓN

El mundo fascinante de internet y todos los beneficios que depara su uso, oculta a veces la complejidad que se esconde detrás de una interface bonita y segura. Sitios web de altas prestaciones gráficas o poderosas aplicaciones de transacciones en línea, nos facilitan muchas veces la vida al hacer solamente un clic sobre botones y selectores, pero internamente puede ocultar una impresionante complejidad de programación. Existen varios factores que colaboran para que esta complejidad sea absoluta: seguridad, disponibilidad, accesibilidad, fácil ubicación y muchos otros factores. Por ejemplo, la W3C o World Wide Web Consortium (*www.w3c.es*) contiene estándares que el programador web debe conocer y aplicar en sus desarrollos web.

Este capítulo se dedicará a desarrollar los fundamentos de programación web. Iniciará con la instalación del servidor, para después continuar con la creación de algunas páginas web con las tecnologías de lenguajes de marcas (HTML, XML, XAML, entre otros).

Instalación de Glassfish y Tomcat

Antes de iniciar la aventura por las aplicaciones web, se debe preparar el camino. Dentro de esos preparativos está la instalación y configuración del servidor que se utilizará para alojar, administrar y cargar sus páginas web. Entre esos servidores están GlassFish y Tomcat.

GlassFish. Es un servidor de aplicaciones de software libre desarrollado por Oracle Corporation. Implementa las tecnologías de Java EE (Enterprise Environment) soporta las aplicaciones que utilizan esta especificación de software. Existe su contraparte comercial denominada Oracle GlassFish Enterprise Server. GlassFish se basa en el servidor Sun Java System Application Server de la corporación Oracle, el cual se derivó de Apache Tomcat. Soporta las últimas versiones de JSP (JavaServer Page), JSF (JavaServer Faces), Servlets, EJBs, Java API para servicios web (JAX-WS), Arquitectura Java para enlaces XML (JAXB), metadatos de servicios web, entre otros.

GlassFish soporta las especificaciones API (JDBC, RMI, e-mail, Web Services, XML, por ejemplo), componentes Java EE (Enterprise JavaBeans, conectores, servlets, portlets, JSP, entre otros). ¿Qué permite esto? Esto permite la creación de aplicaciones empresariales portátiles y de gran escalabilidad. La versión 3.1 se liberó el 28 de febrero del 2011. Muchos consideran que JBoss es un competidor de respeto para GlassFish, sin embargo es cuestión de familiaridad a la hora de desarrollar aplicaciones J2EE.

Se podrá descargar, obtener documentación, participar en actividades diversas, y otros recursos de GlassFish en el sitio:

http://glassfish.java.net/es/

Tomcat. Se le denomina también Jakarta Tomcat o Apache Tomcat. En realidad funciona como un repositorio de servlets y fue desarrollado por Apache Software Foundation. Implementa las especificaciones de JSP y Servlets. No constituye en servidor de aplicaciones dado que es un aliado técnico del servidor web Apache. Sin embargo, puede funcionar como servidor web por sí mismo trabaja íntimamente con la máquina virtual de Java (VMJ) Muchos desarrolladores sostienen que Tomcat es el servidor web que más se utiliza actualmente cuando se trabaja con Java en entornos web. Lo que sí es cierto es que Tomcat es una implementación funcional de JSP y Servlets. No es un servidor de aplicaciones como JBoss, JOnAS o GlassFish pero, como ya se indicó, puede trabajar como servidor web.

Puede descargar la última versión de Tomcat desde

http://jakarta.apache.org/site/binindex.cgi.

A continuación se aborda la habilitación de estos servidores en Netbeans 7.1.

1. Se inicia ingresando en NetBeans, en la pantalla de inicio pulse con el puntero del mouse sobre el botón *Instalar complementos*, como se muestra en la figura 6.1.

Figura 6.1 Instalación de GlassFish y Tomcat.

2. En la segunda pantalla que se muestra debe seleccionar la cejilla *Plugins disponibles* y luego buscar en la lista los servidores Tomcat y GlassFish, tal como se muestra en la figura 6.2. Una vez seleccionados se pulsa el botón *Instalar.*

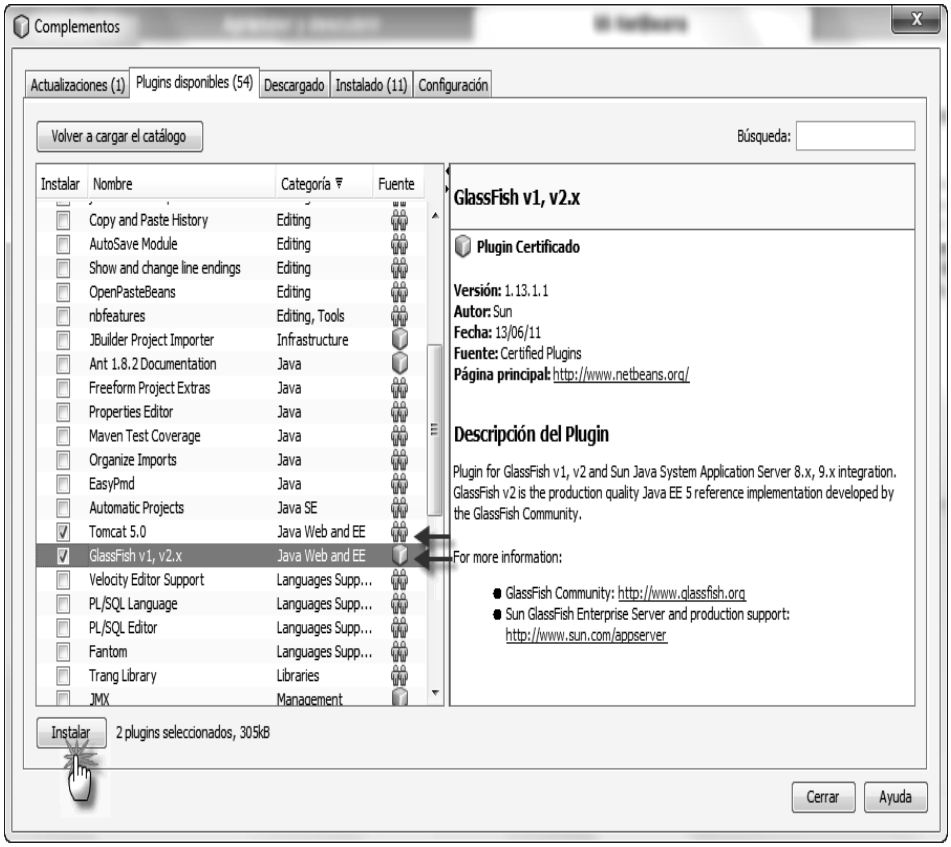


Figura 6.2 Selección de GlassFish y Tomcat para instalar.

3. En las pantallas sucesivas pulse el botón *Siguiente* y seleccione *Acepto los contratos de licencia, respectivamente*. Una vez marcada la aceptación de los acuerdos pulse *Instalar* e iniciará el proceso de instalación. Si pide reiniciar el IDE permita reiniciarlo.
4. Al reiniciarse el IDE probablemente le pedirá actualizar el *Java web applications.* Proceda a instalarlo y cuando finalice permita que el IDE se reinicie de nuevo.

   Con ello ya ha inicializado el IDE para trabajar sus aplicaciones web.

Actividades para el lector

Habilite los servidores Tomcat y GlassFish en su instalación de NetBeans 7.1, tal como se explica en la sección *Instalando GlassFish y Tomcat.*

COMUNICACIÓN POR INTERNET

Java fue diseñado inicialmente para la interconexión de dispositivos interactivos en una red. Esto incluía nodos computacionales distantes entre sí y conectados mediante cualquier medio de comunicación. Con Java.net es posible establecer la comunicación a través de la red con un alto grado de abstracción multiplataforma e independencia del sistema operativo sobre el que se establezca dicha comunicación.

Las funcionalidades básicas de Java.net pueden resumirse en la creación de conexiones, la trasferencia de archivos, la lectura y escritura de archivos mediante el paquete java.nio y la creación de sockets.

El trabajo en la red (networking) es la capacidad de nodos de una red de comunicarse entre sí para el intercambio de información y el ambiente colaborativo. Este trabajo de interconectividad lo lleva a cabo Java utilizando el paquete java.net, que trabaja a través de http o ftp para el manejo de archivos o mediante sockets a un nivel más bajo. Este mecanismo de trabajo se puede visualizar en la figura 6.3.

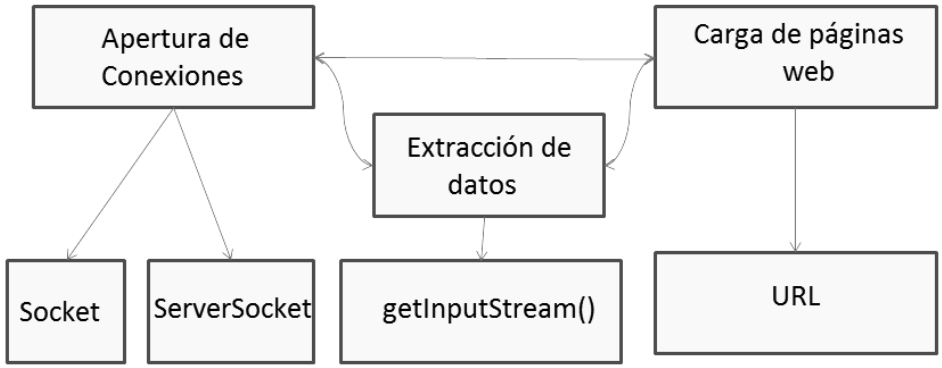


Figura 6.3 Modos de comunicación en la red.

Los pasos intencionales para establecer la carga de un documento en la web son los siguientes:

a) **Objeto URL:** crear un objeto URL que especifica la dirección de la página web que debe cargarse.
b) **Objeto HttpURLConnection**: objeto que realiza la conexión cargando la página indicada por el URL.
c) **Método getContent:** método que realiza la lectura de datos (streams de datos).
d) **Objeto BufferedReader**: objeto que se encarga de procesar la lectura de datos.

Objeto URL

El objeto URL permite especificar la dirección web que debe cargarse en el explorador. Este objeto tiene varios tipos de constructores:

a) URL (String): en el parámetro String se pasa la dirección de la página web que debe cargarse.

b) URL (URL, String): en el parámetro URL se establece la dirección base de la página web que debe cargarse; en String la ruta relativa de la misma.

c) URL (String, String, int, String): en el primer String se coloca el protocolo a utilizar en la conexión (http o ftp), en el segundo String el nombre del host (enriquegomez.com, por ejemplo), en el parámetro int el número de puerto (por ejemplo el puerto 80 en un protocolo http) y finalmente en el último parámetro String un nombre de archivo si es necesario cargar en la página.

d) URL (String, String, String): similar en funcionalidad al señalado en el inciso c), pero sin indicar el número de puerto.

Un ejemplo de aplicación de este objeto es:

```
URL paginaWeb = new URL("http://virtual.alfaomega.com.mx/");
```

Objeto HttpURLConnection

Permite realizar la conexión a una página web según el objeto URL. Por ejemplo:

```
HttpURLConnection conn = (HttpURLConnection) paginaWeb.OpenConnection();
```

Como se observa en la línea anterior, se establece la conexión a través de la variable conn, la cual es de tipo HttpURLConnection.

Método getContent()

Este método permite obtener los datos del Stream que se está procesando en la página web conectada. Por ejemplo:

```
conn.getContent();
```

En la línea anterior se obtiene el archivo xml o el contenido texto de la página a la cual la variable conn (de tipo HttpURLConnection) está conectada.

Objeto BufferedReader

Este objeto permite leer el contenido de un objeto de tipo *InputStreamReader* el cual, a su vez, obtuvo los datos de la conexión establecida por el objeto conn.

Tecnologías basadas en lenguaje de marcas

Un lenguaje de marcas como HTML, XML, XAML, entre otros, constituye una forma de codificación de un documento que incorpora texto y etiquetas (marcas) que permiten definir la estructura de dicho documento y la forma en que éste debe presentarse. El lenguaje que más se reconoce es el HTML (HyperText Markup Language) o lenguaje de marcado de hipertexto. En realidad se le ha considerado como la fundamentación de la World Wide Web (www), aunque no constituya un lenguaje de programación web.

Existen varios tipos de marcados en los lenguajes de marcas:

- **Marcado de presentación:** este tipo de marcado permite formatear la presentación de un documento y pueda leerse. Sin embargo, este formateo no es suficiente para que esta información pueda ser procesada automáticamente. Es muy útil cuando se aplica sobre pequeñas cantidades de información, con las complejidades de su mantenimiento.
- **Marcado de procedimientos:** se orienta hacia la presentación de la información con la posibilidad de que el usuario pueda editarla. Se debe interpretar la información a medida que ésta se procesa, de tal forma que se ejecutan instrucciones como centrar, gestionar el tamaño de la fuente entre textos, aplicar negrita o cursiva, entre otras operaciones. Todas estas acciones deben ser encerradas como operaciones particulares.
- **Marcado descriptivo:** este tipo de marcado semántico utiliza etiquetas para representar fragmentos de texto sin detallar la forma en que deben ser representados o el orden de presentación. Entre estos lenguajes se tiene *SGML* y *XML*. Este tipo de marcado goza de mucha flexibilidad fundamentado en que los fragmentos de texto se etiquetan no como deben aparecer visualmente sino como son en realidad.

Los lenguajes de marcado buscan explícitamente cómo estructurar un documento, el contenido semántico asociado o cualquier otra información que sea necesario implementar.

SGML

SGML (Standard Generalized Markup Language) nació en 1986 como un metalenguaje orientado a definir lenguajes de marcado. Permitía especificar la sintaxis que significaba marcas en los textos, así como las etiquetas permitidas en el mismo. Esto se denominaba *Document Type Definition* (DTD). El DTD permite definir la estructura y el contenido de un tipo de documento.

Ventajas de SGML:

- Los datos pueden ser reutilizados
- Mayor control sobre los datos, lo que permite mantener su integridad
- Portabilidad de datos

- Flexibilidad tanto a nivel de datos como de formato
- Persistencia de la información

La única inconveniencia de SGML es la alta complejidad que se puede presentar para grandes cantidades de información. Ejemplo de SGML:

EJERCICIO: PRUEBA.HTML
<EMail>
<sender>
<persona>
<nombre> Enrique</nombre >
<apellidos> Gómez Jiménez </apellidos>
</persona>
</sender>
<receiver>
<persona>
<listadeCorreo> enrique.gomezcr@gmail.com </listadeCorreo>
</persona>
</receiver>
</EMail>

Este texto lo puede escribir en cualquier editor HTML o Notepad. Luego al ejecutarlo se mostrará el mensaje:

Enrique Gómez Jiménez *enrique.gomezcr@gmail.com*

HTML

HTML (HiperText Markup Language) es quizá el lenguaje de marcas más conocido y difundido en el mundo. Fue creado en el Laboratorio Europeo de Física de Partículas (CERN) en 1989. Fue definido en SGML.

El objetivo principal de HTML es la presentación estática de información y se afirma que fue fundamental en el crecimiento de Internet. Los documentos hipertexto contienen información textual donde cualquier palabra en un documento puede representar un enlace a otro documento. Por tanto, no representa un concepto lineal ni secuencial en cuanto su visualización. Un documento hipermedia

incluye además de texto, imágenes, audio y video (es como incluir la multimedia en los documentos de hipertexto). Los documentos hipertexto contienen información textual donde cualquier palabra en un documento puede representar un enlace a otro documento. Por tanto, no representa un concepto lineal ni secuencial en cuanto su visualización. Un documento hipermedia incluye además de texto, imágenes, audio y video (es como incluir la multimedia en los documentos de hipertexto). Las páginas web constituyen este tipo de documentos en nuestros días, lo cual en su conjunto crea lo que denomina un sitio o un aplicativo web. Para crear estas páginas de hipertexto o hipermedia se utilizará HTML como estándar. Existen muchos generadores de HTML en el mercado, aunque en sus inicios se acostumbraba editar el código en un simple procesador de textos. HTML representa sencillez en su codificación aunque también algunas limitaciones con respecto al procesamiento de información dinámica. Por ejemplo, se puede desarrollar un sitio web estático que contenga texto, audio y video, pero difícilmente un aplicativo que procese bases de datos y gestione archivos de datos.

En resumen, una página creada con HTML es un archivo de texto que pretende representar información en un navegador web (Internet Explorer, Mozilla, Google Chrome, entre otros) mediante el uso de marcas o tags. Estos tags son normalizados internacionalmente, aunque algunos fabricantes de navegadores han incluido algunas nuevas capacidades que incompatibilizan con el resto.

Estructura de un archivo HTML

Un archivo HTML se divide en cabecera (HEAD) y cuerpo (BODY) Mientras HEAD contiene información sobre el archivo como el título, BODY es donde se almacena el contenido del archivo. Los tags o marcas se encierran entre <>, por ejemplo, <head>.

EJERCICIO: MIPRIMER.HTML
<html>
<head>
<TITLE>Mi primera página HTML</TITLE>
</head>
<body>
<H1>Esta es mi primera página HTML</H1>
<P>Este es el primer párrafo</P>
<P>Este es el segundo párrafo</P>
</body>
</html>

Debe escribir el código anterior en un editor de textos como Wordpad y salvarlo con extensión HTML. Posteriormente, al ejecutarlo en un navegador como Internet Explorer o Mozilla, el resultado en la visualización es la siguiente:

Esta es mi primera página HTML

Este es el primer párrafo

Este es el segundo párrafo

Agregue un par de tags más: , que luego de escribir esta instrucción se centrará toda la información que se visualice después hasta encontrar y <hr> que dibujará una línea horizontal de un tamaño dado por el número (de pixeles) que se le asigne, por ejemplo <hr width=600> dibuja una línea horizontal de 600 pixeles. El código para este segundo ejemplo es:

EJERCICIO: MISEGUNDO.HTML
<html>
<head>
<TITLE>Mi primera página HTML</TITLE>
</head>
<body>
<H1>Esta es mi primera página HTML</H1>
<hr width=600>
<P>Este es el primer párrafo</P>
<P>Este es el segundo párrafo</P>
</body>
</html>

La visualización de este segundo HTML es el siguiente:

Esta es mi primera página HTML

Este es el primer párrafo

Este es el segundo párrafo

Ahora agregue más tags:

- **aquí lo que va en negrita** observe se abre con y se cierra con
- *aquí lo que va en cursiva* observe que se abre con y se cierra con
- <p> aquí el párrafo que se quiere. Automáticamente agrega una línea en blanco que significa el salto de línea y retorno de carro. Acepta algunos parámetros como:
  - align ="" para alinear el texto al centro (center), derecha (right), izquierda (left) y justificado (justify). Se ha desaprobado el uso de este *tag* al igual que *basefont*, *font, strike y u*. Se usa CSS en su lugar.
- que nos permite utilizar un tipo de letra.
- <blockquote> para indentar un párrafo. Se cierra con </blockquote>

Ejemplo

EJERCICIO: MITERCER.HTML
<html>
<head>
<TITLE>Formato de texto</TITLE>
</head>
<body>
Formato de texto
<hr width=600>
<P align="center">Primer párrafo centrado y fuente arial 12. Esto está en <B>negrita</B>, y esto en <I>cursiva.</I></P>
<P>El segundo párrafo en letra es Arial y tamaño 10. Alineado a la izquierda y retorno de carro aqui.
Se Utiliza fuente color rojo.</P> <blockquote>Tercer párrafo. Finalizando, se indenta un párrafo, con letra Arial y tamaño 5..</blockquote>
</body>
</html>

La ejecución de este código HTML permitirá visualizar en el explorador algo similar a lo siguiente:

Formato de texto

Primer párrafo centrado y fuente arial 12. Esto está en **negrita**, y esto en *cursiva.*

El segundo párrafo en letra es Arial y tamaño 10. Alineado a la izquierda y retorno de carro aquí. Se utiliza fuente color rojo.

Tercer párrafo. Finalizando, se indenta un párrafo, con letra Arial y tamaño 5.

Para habilitar un fondo de pantalla se puede utilizar el tag body background. El siguiente código crea una página web y le configura el fondo de la misma.

EJERCICIO: MITCUARTO.HTML
<html>
<head>
<TITLE>Fondo (background) de una página web</TITLE>
</head>
<body background="fondo.png">
Fondo página web
<hr width=600>
<P></P>
<P></P>
<P align="center">Primer párrafo después de agregar 2 en blanco. Está centrado y letra por defecto del navegador. Palabra <B>negrita</B>, y esta otra en <I>cursiva.</I></P>
</body>
</html>

En la figura 6.4 se muestra la ejecución del código anterior.

Figura 6.4 Página HTML con fondo.

En HTML existen otros tags importantes en cuanto a su funcionalidad. Uno de ellos es el *input*, el cual le permite agregar datos a un usuario. Este control depende directamente del atributo *type*, que es el que define el tipo de control que se mostrará. A continuación se da un listado de algunos de ellos.

- *Text*: define un control de tipo entrada de texto. Permite el ingreso de datos por parte del usuario.
- *password*: similar al funcionamiento de *text*, con la diferencia de que oculta los caracteres ingresados utilizando puntos o asteriscos en lugar de letras.
- *checkbox:* funciona al estilo Sí/No. Es decir, tiene valores de los cuales se pueden seleccionar todos o ninguno. Varios checkbox pueden tener el mismo nombre.
- *radio:* funcionan como los *checkbox*, con la diferencia que sólo un valor de los contenidos puede ser seleccionado de la lista.
- *submit*: actúa como un botón de comando. Cuando el usuario pulsa un botón *submit* el formulario actual es enviado automáticamente.
- *image*: el tag *submit* puede convertirse a un aspecto gráfico mediante este tag. Pueden utilizarse los tag *image* como mapas de imagen.
- *reset*: crea un botón de comando. Cuando se pulsa un tag de éstos (el botón) todos los campos del formulario web son restablecidos a sus valores iniciales correspondientes.

- *button*: crea un botón común el cual no se le establece una función preestablecida. Este tipo de tag se utiliza básicamente para programarlos con scripts del lado del cliente, principalmente con JavaScript.
- *hidden*: este tipo de control permite capturar valores en un formulario web que no se muestran y que son enviados posteriormente. Por ejemplo, cuando se van a enviar datos de un formulario a otro. Los valores actuales se guardan en controles hidden, los cuales son posteriormente recuperados en el formulario que es llamado.
- *file*: mediante este control el usuario puede seleccionar un archivo para enviar. El archivo seleccionado es enviado conjuntamente con el formulario.

Para terminar el tema de HTML se creará un aplicativo básico que resume los tags (convertidos en controles) que se acaban de citar. El código es el siguiente:

EJERCICIO: EJEMPLOCONTROLES.HTML
<form action="ejemplo HTML" enctype="multipart/form-data">
<fieldset>
<legend>Información Personal del Empleado</legend>
Nombre: <input name="nombreEmpleado" type="text" maxlength="25" size="25" value="*" />
Apellido: <input name="apellidos" type="text" maxlength="50" size="50" value="*" />
Contraseña: <input name="clave" type="password" maxlength="15" size="15" value="*" />
Sexo: <input name="sexo" type="radio" checked="checked" value="masculino" />Masculino
<input name="sexo" type="radio" value="femenino" />Femenino
Fotografía del empleado: <input name="imagen" type="file" size="10" accept="image/gif" value="-" />
</fieldset>
<fieldset>
<legend>Deportes</legend>
<input name="deportes" type="checkbox" checked="checked" />Fútbol
<input name="deportes" type="checkbox" checked="checked" />Voleiball
<input name="deportes" type="checkbox" />Beisbol

EJERCICIO: EJEMPLOCONTROLES.HTML
<input name="deportes" type="checkbox" />Baloncesto
<input name="deportes" type="checkbox" />Tenis
</fieldset>
<input name="borrarform" type="reset" value="Limpiar formulario" />
<input name="enviarform" type="button" value="Enviar formulario" />
<input name="email" type="hidden" value=*enrique.gomezcr@gmail.com* />
<input TYPE="button" VALUE="Cerrar ventana" onClick="window.close();">
<INPUT TYPE="button" VALUE="Cargar otra ventana" onClick="window.location.replace ('segundaPagina.html');">
</form>

Algunos comentarios con respecto a este código HTML:

- La página se encierra entre <form action="ejemplo HTML" enctype="multipart/form-data"> y </form> Es la estructura básica de un formulario HTML.
- <fieldset> ...</fieldset> encierra un grupo de elementos. Se asocia con <legend> ... </legend> para el encabezado del grupo y posteriormente la lista mediante varios input de la forma <input name="nombre del input" type="tipo de componente" />Texto que aparece del elemento
 (<input name="deportes" type="checkbox" />Beisbol
, define un elemento de nombre *deportes* que es de tipo *checkbox* y cuyo título de elemento es *Beisbol*.
- <input name="borrarform" type="reset" value="Limpiar formulario" /> crea un componente (un botón) de tipo *reset* que funciona limpiando todos los elementos *input* del formulario.
- <input TYPE="button" VALUE="Cerrar ventana" onClick="window.close();"> crea un input de tipo button con un título Cerrar ventana que ejecuta un evento onClick. El onClick significa que si el usuario pulsa ese botón se cerrará la ventana. En el caso de <INPUT TYPE="button" VALUE="Cargar otra ventana" onClick="window.location.replace ('segundaPagina.html');"> lo que realiza es que si se pulsa ese botón se ejecuta *segundaPagina.html* que es otro archivo HTML.

Si ejecuta este archivo HTML tendría una página web similar a la figura 6.5.

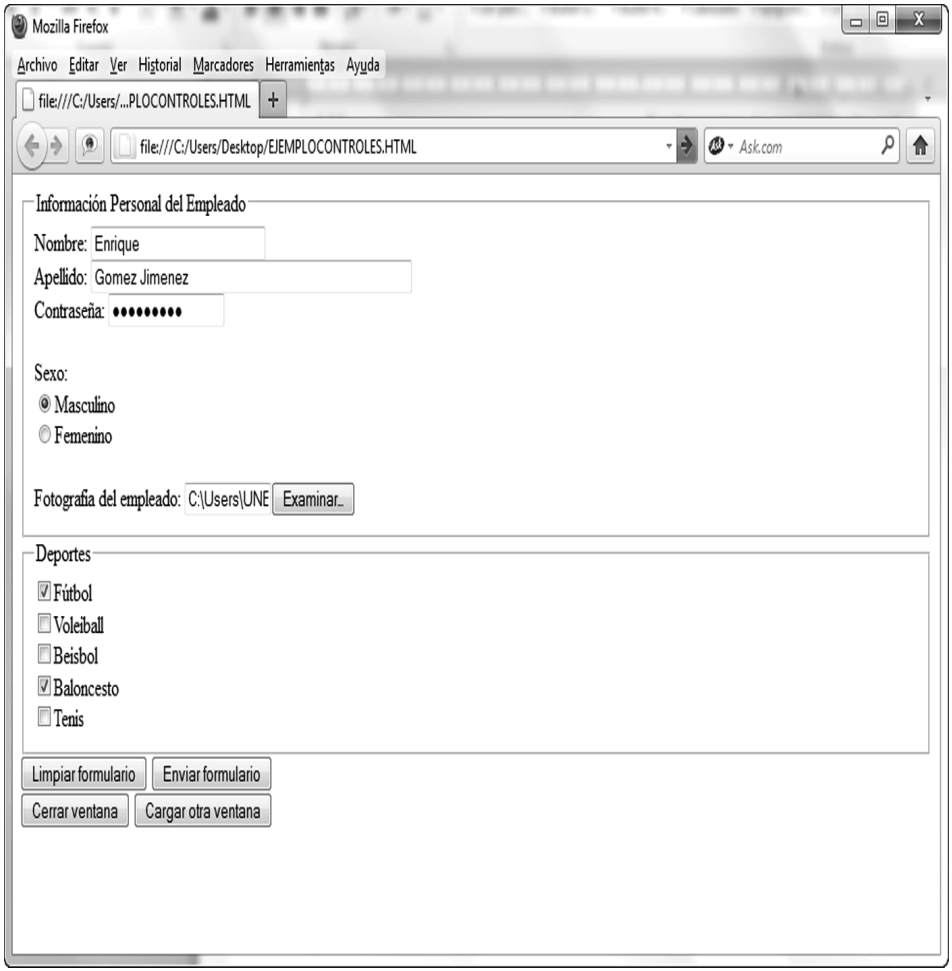


Figura 6.5 Página HTML utilizando controles.

HTML 5

HTML 5 ha sido remozado con una enriquecida semántica y una accesibilidad implícita. Se han eliminado ambigüedades y es asiduo contrincante del Flex de Adobe o el Silverlight de Microsoft. Con HTML 5 se elimina el uso abusivo de los tags *DIV* delimitando cada sección de una página web. Con ello los buscadores web pueden interesarse en una determinada sección de la página y no necesariamente de todo el sitio. HTML 5 pretende reemplazar el actual XHTML.

Theproc de España, hace una clara diferenciación entre el HTML 4 y HTML 5 en su sitio web *http://theproc.es*, representándola como se muestra en la figura 6.6.

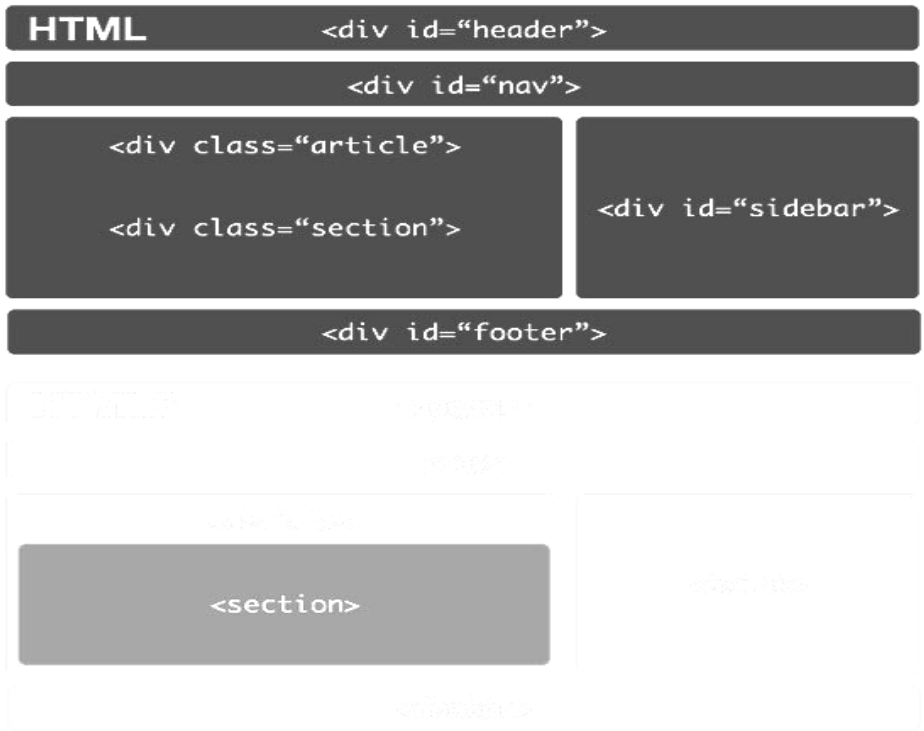


Figura 6.6 Diferencias entre HTML y HTML. Extraído el 6 de julio del 2012 desde http://theproc.es/2010/1/29/12236/manual-de-html5-en-espanol---1-de-3.

Observe en la figura 6.6 que los *divs* han sido sustituidos por secciones. Asimismo, la estructuración de la página resulta mucho más clara de manejar para el programador y para representar en un explorador web. Puede observar sitios diseñados con HTML 5 en

http://html5gallery.com/.

IBM destaca los nuevos elementos semánticos que se incluyen en HTML 5, en su página de internet

http://www.ibm.com/developerworks/ssa/web/tutorials/wa-html5/section2.html.

La tabla 6.1 resume los nuevos elementos semánticos en HTML 5.

Tabla 6.1 Nuevos elementos semánticos en HTML 5, según IBM

ELEMENTO SEMÁNTICO	DESCRIPCIÓN
<header>	Elemento utilizado para la definición del encabezado de la página web. Puede corresponder a la página completa o una parte de ella (por ejemplo un elemento <article> o <section>)
<footer>	Elemento empleado para especificar el pie de página para la página en sí o para una sección de la misma.
<nav>	Representa un contenedor de enlaces que permiten la navegación de la página. Sólo se utiliza para bloques de navegación principales, dado que podría eventualmente crear enlaces desde cualquier sección, siendo ésta la que establezca el enlace y no necesariamente una sección <nav>.
<article>	Se usa para la definición de elementos que son independientes de la página. Puede trabajar por sí mismo, como por ejemplo las noticias que se pasan en una página, un blog, comentarios, entre otros. Se emplea feed RSS para su funcionamiento.
<section>	Representa una sección de un documento o aplicación HTML. Por ejemplo, una sección podría ser un espacio de noticias que dinámicamente se están actualizando cada momento, sin necesidad que el resto de secciones lo haga.
<aside>	Se utiliza, por ejemplo, para marcar un contenido o una barra lateral que contiene información separada del resto de la página. Por ejemplo, los bloques publicitarios que podrían colocarse a un lado de la página HTML podrían diseñarse mediante este elemento.
<hgroup>	Este elemento se usa para crear un grupo de encabezados: por ejemplo, un título y un subtítulo de una sección o una página HTML.

Información sobre HTML5 puede descargarse, ver y analizar el código desde:

http://public.dhe.ibm.com/software/dw/web/wa-html5/html5.examples.zip

En este sitio se publica un claro ejemplo de aplicabilidad de HTML5. Por no ser parte de la temática de este libro, no se desarrollará un caso aplicativo. Sin embargo, es importante citar que existe esta tecnología y que posiblemente pronto estará ampliamente difundida.

Canvas

Canvas es un elemento en HTML 5 orientado a la creación de dibujos en 2D para sitos web. Mediante este elemento se pueden crear objetos que posean transiciones, animaciones y diferentes formas para enriquecer el aspecto gráfico de las páginas HTML. Inclusive, mediante Canvas es posible crear juegos interactivos en la web, aplicaciones de pinturas y cualquier aplicativo gráfico imaginable cuando se utiliza combinado con el poder de JavaScript, entre otras APIs. Para verificar el poder de Canvas en HTML 5, pueden verse los juegos desarrollados en

http://www.baluart.net/articulo/5-geniales-juegos-en-html5

o en

http://www.dacostabalboa.com/es/10-juegos-en-html5-y-canvas/8941

Video y audio en HTML5

Los sitios web que no presenten interacción con el usuario están destinados a no ser vistos por nadie. Del hipertexto se pasó al hipermedia y esto ha significado que las páginas web sean capaces de reproducir audio y video en la forma más sencilla posible. El formato flv (Flash Video) ha sido el propulsor de esta facilidad, bajo el auspicio de Adobe Flash. Son muchos los formatos que han aparecido en el mercado de la reproducción de audio y video en páginas web. HTML 5 posee la capacidad de implementar audio y video en las páginas web, sin necesidad de instalar plug-ins adicionales en el navegador. Existen ciertas limitaciones dado el soporte de los navegadores a los formatos de video y la existencia de patentes y derechos de propiedad intelectual en cuanto codecs de audio.

Almacenamiento local y aplicaciones fuera de línea

El desarrollador web generalmente recurre a las cookies para almacenar información del usuario que posteriormente podrá utilizar. Sin embargo, existen muchas limitaciones de estas cookies, como por ejemplo la capacidad de almacenamiento de una cookie, el número máximo de cookies que un navegador puede mantener o el desperdicio de recursos que significa estarlas enviando con cada request (solicitud) que se haga al servidor. HTML 5 ofrece una serie de APIs que permite resolver este problema. Básicamente, consiste en un almacenamiento local el cual separa del documento HTML 5. La información se almacena en la máquina del cliente y se espera tenerla disponible cuando se vuelva a utilizar. Es decir, no se necesita cargar la información almacenada del usuario en cada solicitud HTTP que se realice al servidor.

En cuanto aplicaciones fuera de línea (off line) HTML 5 descarga todos los archivos necesarios de un aplicativo web para que continúe funcionando a pesar de que el sitio este caído o se requiera trabajar en forma asincrónica. Lógicamente, esto es muy funcional cuando las páginas son estáticas y la información no tiene un comportamiento dinámico. La idea es que el aplicativo web se cargue localmente y

cualquier cambio que realice el usuario en el mismo se ejecutará en el servidor una vez que nuevamente se reconecte el aplicativo al servidor.

Mejoras en formularios web

El lector aprendió cómo crear controles de formulario (vea la figura 6.5) con HTML 4. Con HTML 4 se pueden crear controles de formulario con el tag <*input*>, destacando *button, checkbox, file, image, password, submit*, entre otros. Para implementar la conexión de estos controles se deben utilizar bibliotecas JavaScript facilitadas por los componentes de IU o utilizar algunos de los productos comerciales conocidos actualmente que ofrecen esta funcionalidad, tal como Flex de Adobe, Silverlight de Microsoft o JavaFX presente en NetBens/Java.

HTML 5 ofrece un nuevo tipo de entrada de formulario que pretende solucionar esta dependencia. De estos <*input*> destacan *color, date, datetime, email, month, range, search, url, week*, entre otros. Sin embargo, el soporte para estos campos aún es limitado y se espera mejoren apenas madure más HTML 5. Además de los tipos de entradas citados anteriormente, HTML 5 ofrece dos recursos muy importantes: *autofocus* que permite enfocar un elemento específico de la página HTML (sin necesidad de hacerlo mediante JavaScript), y *placeholder* que permite definir el texto que se mostrará en un determinado control que se basa en casilla de texto en el momento que el mismo esté vacío o se presente un error de formato (por ejemplo, error al escribir una dirección de correo electrónico sin que se digite el @). Para tener una noción de la potencia que ofrece HTML 5 puede descargarse el aplicativo que demuestra la interface y código creado.

Actividades para el lector

- Cree un aplicativo HTML 4 que permita crear un formulario para el registro de estudiantes. En lugar de Deportes describa qué materias académicas son de su agrado.
- Analice e implemente el código del aplicativo ejemplo de HTML 5 publicado en

http://www.ibm.com/developerworks/ssa/web/tutorials/wa-html5/downloads.html

Figura 6.7 Ejemplo de aplicación utilizando HTML 5. Extraído el 16/02/2012 desde http://www.ibm.com/developerworks/ssa/web/tutorials/wa-html5/downloads.html.

XHTML

XHTML (Extend Hyper Text Markup Language) constituye una reformulación de las versiones 1 y 4 de HTML. Con XHTML se pretende corregir el desorden en la normalización del lenguaje, así como también la falta de compatibilidad con muchos navegadores web actuales, ya que XHTML posee la capacidad del formato HTML y la formalidad de XML, lo cual permite que cualquier herramienta estándar pueda validarlo y hacerlo funcionar. En síntesis, XHTML es HTML extendido. Su extensión se basa en utilizar el formalismo de XML en su implementación. Con ello se busca una web semántica donde la información y la estructura de la misma estén correctamente separadas.

XHTML es más estricto que HTML y es una solución de reemplazo de ésta. Esto por cuanto el uso, cada vez más generalizado de herramientas basadas en XML. ¿Para qué sirve XHTML? En parte por que al mercado han llegado gran cantidad de dispositivos y es necesario que éstos tengan facilidad de interpretar la información que se genera en aplicaciones web, independiente del dispositivo. XHTML considera la inclusión de otros lenguajes como MathML, SMIL o SVG, al contrario de HTML.

Algunas ventajas de XHTML con respecto a HTML son las siguientes:

- Con XHTML se pueden incorporar espacios de nombres existentes en XML tales como *Scalable Vector Graphics* y *MathML*.
- No es necesaria la implementación de heurísticas para determinar el contenido de la página que quiso establecer el autor de la misma, dado que mediante un parser se puede formatear la información y así determinar el contenido exacto de lo que el autor quiso representar.
- Dado que se basa en XML se pueden utilizar herramientas creadas para el procesamiento de este tipo de formato.

A continuación se presenta un archivo XHTML para demostrar su funcionalidad.

EJERCICIO: EJEMPLOXHTML.XML
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!--Comentario aqui que no será considerado por el navegador -->
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es" lang="es">
<head>
<title>Primer ejemplo XHTML</title>
</head>
<body>
Hola mundo con XHTML
</body>
</html>

Algunas consideraciones del código anterior:

- <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" que es el encabezado que debe llevar todo documento XHTML según las especificaciones de la w3c.
- Debe iniciar con la etiqueta <html ...> y terminar con </html>.
- El encabezado está entre <head> y </head>. En medio de ellos se encuentra el título o títulos.
- El cuerpo de la página se encuentra entre <body> y <body/>

De ejecutarse el código XHTML anterior se mostraría el siguiente mensaje:

Hola mundo con XHTML

XML

XML (eXtensible Markup Language) constituye un metalenguaje extendido que se basa en etiquetas. Significa la simplificación y adaptación de SGML. Es muy similar a HTML, con la diferencia que se fundamenta en la descripción de los datos y no en cómo muestra éstos, que es la función de HTML. En resumen, XML se utiliza para estructurar, almacenar e intercambiar información entre aplicaciones y diferentes plataformas. Su utilización se extiende a bases de datos, hojas de cálculo, archivos, entre otras formas de descripción de datos.

Se podría definir de la siguiente manera: XML es un formato universal que se utiliza para describir e intercambiar documentos y datos estructurados en Internet.

Algunas características son:

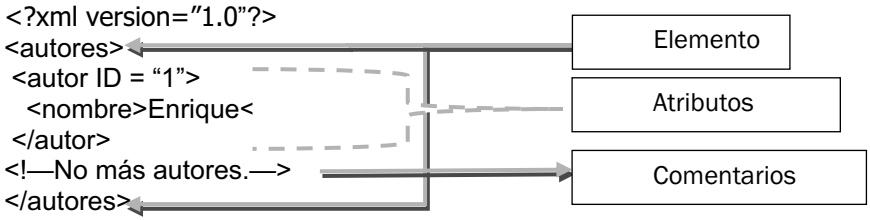
- Suministra un método que permite describir e intercambiar datos estructurados.
- Permite que se definan elementos y atributos personalizados.
- Permite el anidamiento de elementos.

Componentes de un documento XML

Un documento XML está compuesto por:

- Elementos
- Atributos
- Comentarios

A continuación se puede observar la conformación de estos tres componentes;



Los elementos tienen las siguientes características:

- Están formados por un par de etiquetas: una de inicio y una de cierre.
- Entre las etiquetas de inicio y fin está el contenido del elemento (datos) u otros elementos.

- El primer elemento contiene el resto de elementos de un documento XML. A este elemento se le denomina elemento raíz, los demás son secundarios.

Un elemento tiene el siguiente aspecto en XML:

<LIBRO>...<TITLE>XML programando XML</TITLE>...</LIBRO>
Los atributos tienen las siguientes características:

- Cualquier elemento puede contener atributos.
- Los atributos pueden ser una alternativa al uso de elementos para el almacenamiento de información
- Los atributos se definen para un único elemento.

Recuerde que los atributos son partes o propiedades de los elementos.

Por ejemplo, para el caso de <LIBRO>...<TITLE>XML programando XML</TITLE>...</LIBRO>, este elemento tiene un atributo denominado Editorial, tal y como se muestra en la siguiente línea de código XML.

<LIBRO> <EDITORIAL>Alfaomega</EDITORIAL></LIBRO></p>< /p>< /p>

Los atributos no pueden ser jerárquicos, lo que significa que no pueden contener subelementos. Sin embargo, podría cambiarse el atributo por su propio elemento; el código sería:

```
<PUBLICACION>
   <LIBRO>XML programando XML </LIBRO>
   <EDITOR>EDITORIAL ALFAOMEGA</EDITOR>
</PUBLICACION>
```

Tecnologías XML

XML ha venido innovando a través de los últimos años y las tecnologías que se han gestado son las siguientes:

- XSD
- XSLT
- XPath
- DOM
- XQuery

XSD es un estándar que se utiliza para la definición de esquemas, definiendo la estructura requerida por un documento XML válido. Puede ser un documento aislado que actúa como un documento maestro y el cual es referenciado por otros documentos, denominados instancia. El documento aislado tendrá una extensión .xsd.

XSLT permite la transformación del contenido de un documento XML fuente en otro documento diferente en formato o estructura. XSLT es parte de XSL (eXtensible Stylesheet Language); XSLT utiliza un archivo origen, que tiene un formato correcto y sirve como entrada para la transformación a XSLT. Existen dos algoritmos (parseadores) que se utilizan para manipular estructuras XML: SAX y DOM. SAX se basa en eventos que se ejecutan cada vez que encuentra un tag, mientras que DOM se basa en realizar un recorrido secuencial por todo el árbol XML. Se recomienda utilizar SAX cuando los archivos XML son muy grandes, para mejorar el rendimiento. De lo contrario se debe utilizar DOM por el hecho que es muy sencillo de comprender y manejar. Para nuestros efectos, en lugar de DOM se utilizará JDOM que está especialmente desarrollado para Java y en consecuencia para NetBeans.

SAX (Serial Access Xml) es un parser que permite el acceso a la información contenida en un archivo XML. Posee dos inconvenientes principales: difícil de manejar y realiza una lectura sólo hacia adelante (por lo que no se puede devolver a los nodos ya recorridos) SAX posee un objeto denominado *DocumentHandler* que es un manipulador de documentos XML y es quien realiza el trabajo de parser dentro del mismo. Este parseador define una interfaz que responde a eventos, los cuales a su vez llaman a métodos específicos. Entre estos métodos se tienen los que se indican y explican en la figura 6.8.

Figura 6.8 Métodos de manipulación de archivos XML mediante SAX.

JDOM constituye una API que se utiliza para la creación y manipulación de documentos XML. Se utiliza en lugar de DOM y SAX, las cuales son estrategias más genéricas (es decir, se emplean para el mismo objetivo pero sin pensar en un lenguaje especifico). En NetBeans se implementan parser para el manejo de DOM y SAX, tales como Xerces, XML4j, Crimson, Oracle's Parsers, entre otros, a través de complementos. En fin, JDOM, como API, no constituye en sí un parser, pero sí

implementa la abstracción necesaria para trabajar como tal. Esta librería puede descargarse desde *www.jdom.org*.

DOM fue creado por la W3C (World Wide Web) para manipular páginas HTML utilizando JavaScript. Mientras JDOM se diseñó para usar el poder de Java tales como las colecciones, métodos sobrecargados, entre otros. El acrónimo JDOM no necesariamente es Java Document Object Model, pero se conoce que es una implementación para Java que se basa en este modelo. A continuación, en la figura 6.9 se muestran los paquetes que conforman JDOM.

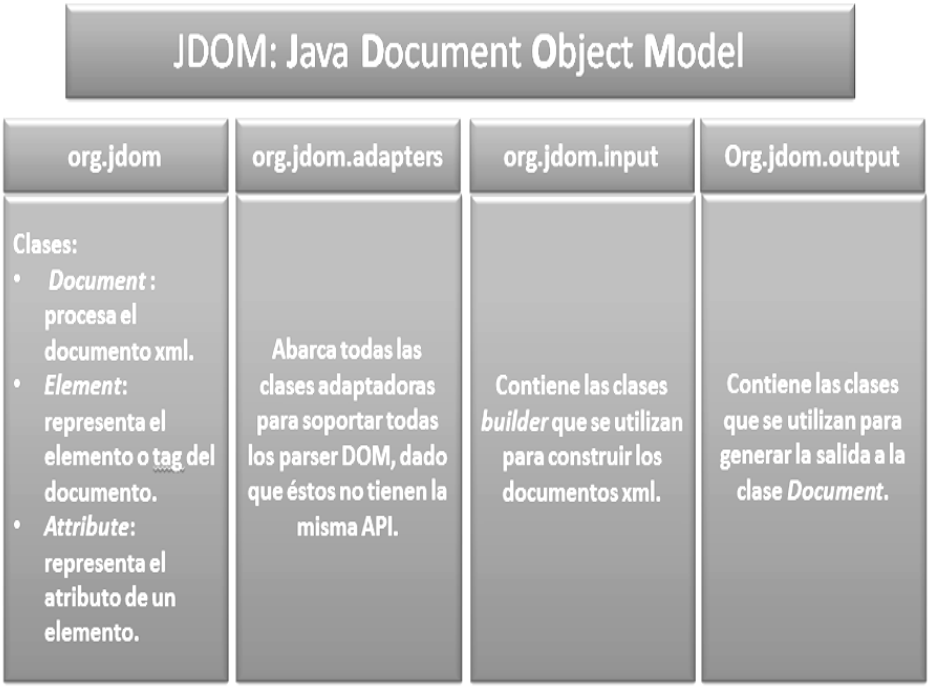


Figura 6.9 Paquetes contenidos en JDOM.

JDOM utiliza parsers para realizar su trabajo y para ello debe indicarle cuál utilizar. Para ello se emplea un método genérico como el siguiente:

```
public SAXBuilder(String parserClass, boolean validation)
```

En este llamado el parser que se va a utilizar sería *Xerces*. Es el primer parámetro de la instrucción anterior. El segundo parámetro se usa para que el parser cumpla con una función de validación.

Un segundo método de uso de un parser lo constituye la siguiente instrucción:

```
public DOMBuilder(String adapterClass, boolean validation)
```

Esta instrucción tiene como primer parámetro el argumento de una clase adaptadora que se utiliza para el parser que se va a emplear. El segundo parámetro es para cumplir una función de validación igual que en SAXBuilder. Por ende, se puede notar que la primera utiliza un parser SAX y el segundo uno DOM.

La siguiente instrucción que le dará al método será la construcción del propio parser (analizador) del documento XML. Esto se consigue mediante la instrucción *build*. La siguiente es una instrucción válida:

```
Document build(File file)
```

En file se recibe el nombre del archivo XML que se analizará mediante el parser, el cual almacenará el documento mediante la clase *Document*. Posterior a este almacenamiento se podrá contar con varios métodos que hacen posible la recuperación de los elementos del archivo. La tabla 6.2 muestra estos métodos.

Tabla 6.2 Métodos para recuperar los elementos XML mediante JDOM.

MÉTODO	DESCRIPCIÓN
Element getRootElement();	Permite obtener el elemento raíz del archivo XMML.
String getText();	Permite obtener el texto del elemento actual o de un tag (etiqueta).
List getChildren();	Obtiene todos los subordinados (hijos) del elemento actual.
List getChildren(String nombre);	Obtiene el elemento cuyo valor concuerde con el parámetro recibido en nombre.
List getMixedContent();	Se utiliza para obtener todo el contenido del elemento mediante una lista.
Element getChild (String nombre);	Obtiene el primer valor que concuerda con el parámetro pasado en nombre.
String getAttributeValue(String nombre);	Obtiene el atributo que concuerda con el valor pasado a través del parámetro.

Cómo maneja esto NetBeans

Para entender el funcionamiento de los archivos xml en NetBeans se desarrollará un ejemplo sencillo. Mediante éste se explicarán algunos de los archivos que se pueden generar a partir de un archivo xml.

1. Ingrese a NetBeans.
2. Seleccione *Proyecto Nuevo…* desde el menú *Archivo*. O presione *CTRL+Mayúscula+N* al mismo tiempo.
3. Seleccione *Java* y *Aplicación Java*, tal como se muestra en la figura 6.10.
4. Nombre este proyecto como *webXML*.
5. Pulse con el botón derecho sobre la carpeta *Source Package* y seleccione *XML* en la opción *Otro…* del menú Nuevo. Estando ahí seleccione *Documento XML*.

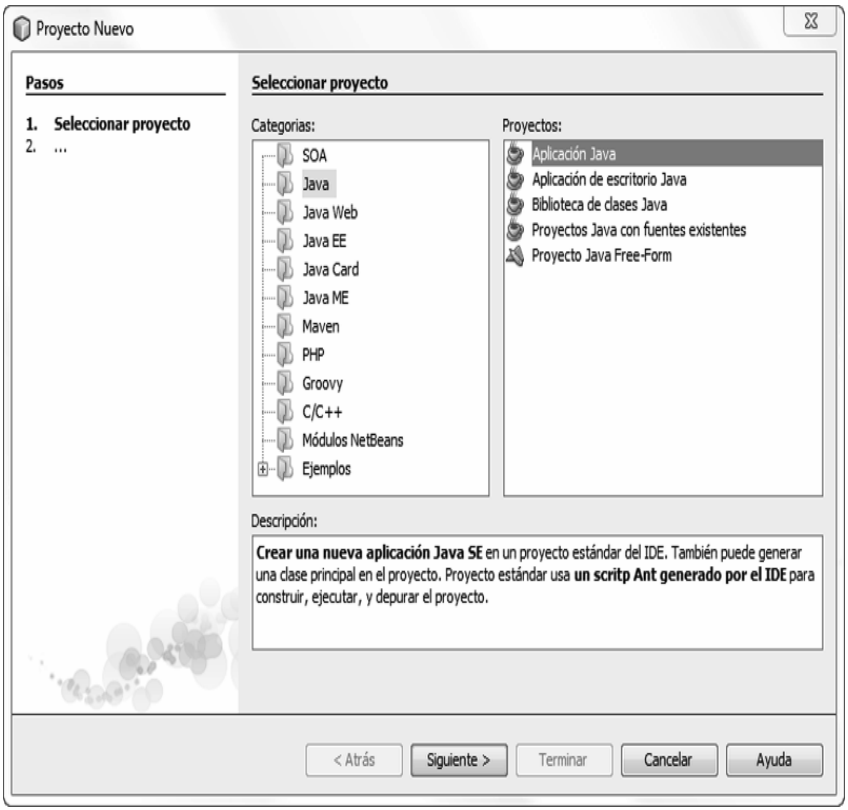


Figura 6.10 Agregando un archivo xml a nuestro aplicativo web.

Eventualmente este tipo de archivo podrá aparecer en el menú de contexto al pulsar sobre la carpeta *Source Package* y seleccionar *Nuevo*.

6. Pulse el botón *Siguiente* y en la pantalla que se presenta posteriormente seleccione una de las opciones para determinar el tipo de documento xml que creará. Las opciones disponibles son las tres que se muestran en la figura 6.11.

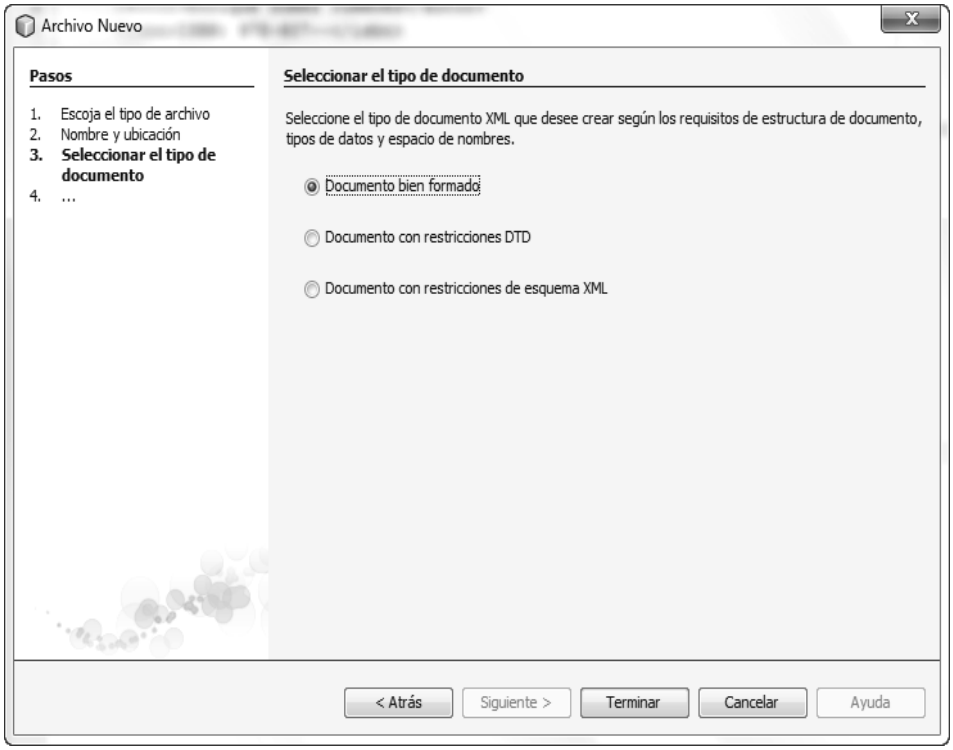


Figura 6.11 Tipos de documentos xml a crear en nuestro aplicativo web.

Los tres tipos de documentos xml son:

Documento bien formado: se refiere al documento xml que debe contar con etiquetas de apertura y cierre. Esto significa que las etiquetas XML representan una información jerárquica que indica cómo se relacionan los elementos entre sí. Si se olvida esta regla se producirán ambigüedades en el procesamiento automático del archivo xml. Existe una jerarquización fuerte en cuanto los elementos del documento.

Los solapamientos tampoco son permitidos, es decir, deben coincidir los elementos que se abren con su respectivo tag de cierre. Por ejemplo:

```
<Libro> Netbeans 7.1<Autor> Enrique Gómez </Libro></Autor>
```

No está bien formado dado que Autor se cierra con Libro cuando debería ser con Autor. Libro es quien debería cerrar Autor. El xml bien formado debería quedar de la siguiente manera (observe la parte más oscura):

```
<Libro>Netbeans 7.1<Autor>Enrique Gómez</Autor></Libro>
```

Documento con restricciones DTD: Pretende mediante este formato que el archivo xml mantenga la consistencia en el documento, por medio de una estructura común. De esta forma los datos pueden ser validados dado que se conoce cómo se estructura el documento y las descripciones que tenga. Por ejemplo, el siguiente es un documento DTD xml válido:

```
<!ELEMENT XXX (AAA* , BBB)>
<!ELEMENT AAA (#PCDATA)>
<!ELEMENT BBB (#PCDATA)>
```

Documento con restricciones de esquema XML: Los esquemas se utilizan para describir estructuras y restricciones de los elementos contenidos en un archivo xml. Las restricciones en estos esquemas permiten ir más allá de la normativa sintáctica impuestas en xml. Existe un alto grado de abstracción en las restricciones, siendo más expresivos que un DTD pero con menos descripción formal. Un esquema válido de este tipo de documentos es el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="Libro">
    <xsd:complexType>
     <xsd:sequence>
       <xsd:element name="Título" type="xsd:string"/>
       <xsd:element name="Autores" type="xsd:string" maxOccurs="10"/>
       <xsd:element name="ISBN" type="xsd:string"/>
     </xsd:sequence>
     <xsd:attribute name="Precio" type="xsd:double"/>
   </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

7. Una vez hecho ese pequeño paréntesis en el desarrollo, pulse sobre la opción de *Documento bien formado* y pulse el botón *Finalizar*. El archivo xml que debe escribir en esta parte es el que se muestra a continuación:

EJERCICIO: XMLLIBROS.XML
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="Presentacion.xsl"?>
<!DOCTYPE libros SYSTEM "xmlLibros.dtd">
<libros>
<libro>
<titulo>PROGRAME CON NETBEANS 7.1: ESCRITORIO, WEB Y DISPOSITIVOS MOVILES</titulo>
<autor>Enrique Gomez Jimenez</autor>
<isbn>ISBN: 978-607--</isbn>
<anyo>2011</anyo>
</libro>
<libro>
<titulo>APLICACIONES CON VISUAL BASIC.NET: PROGRAME PARA ESCRITORIO. WEB Y DISPOSITIVOS MOVILES</titulo>
<autor>Enrique Gomez Jimenez</autor>
<isbn>9788426717054 </isbn>
<anyo>2010</anyo>
</libro>
<libro>
<titulo>VISUAL BASIC.NET (PASO A PASO)</titulo>
<autor>Enrique Gomez Jimenez</autor>
<isbn>9788441514126 </isbn>
<anyo>2002</anyo>
</libro>
</libros>

El archivo xml contiene una serie de registros con título, autor, isbn y año (anyo). Eventualmente, la instrucción que referencia el dtd <!DOCTYPE libros SYSTEM "xmlLibros.dtd"> podría escribirse como:

```
<!DOCTYPE libros[
<!ELEMENT libros (libro)+>
<!ELEMENT libro (titulo, autor,isbn,anyo)>
<!ELEMENT titulo (#PCDATA)>
<!ELEMENT autor (#PCDATA)>
<!ELEMENT isbn (#PCDATA)>
<!ELEMENT anyo (#PCDATA)>
]>
```

Observe que en la segunda línea se encuentra la instrucción *<?xml-stylesheet type="text/xsl" href="Presentacion.xsl"?>*.

Eso significa que el archivo xml hará referencia a otro archivo denominado *Presentacion.xsl*. Un archivo xsl significa eXtensible *Stylesheet Languaje* o su definición en español: lenguaje extensible de hojas de estilo. Con el párrafo anterior entonces lo que se pretende es crear una hoja de estilo sobre la cual el documento xml pueda ejecutarse en un navegador. En otras palabras, un archivo xsl permite dar formato o estilo a un documento xml existente.

8. Habiendo definido someramente lo que es un archivo xsl, escriba código para basar en él su documento xml. Proceda conforme lo hizo en el punto 5 y cree un archivo xsl de nombre *Presentacion.xsl*. Observe la figura 6.10. Este tipo de archivo aparece como *Hoja de estilo XSL*. El código que contendrá este archivo es el siguiente:

EJERCICIO: PRESENTACION.XSL
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<body>
Mis Publicaciones
<table border="1">
<tr bgcolor="skyblue">
<th align="left">Titulo</th>
<th align="left">Autor</th>
<th align="left">ISBN</th>
<th align="left">Ano de publicacion</th>
</tr>
<xsl:for-each select="libros/libro">
<tr>
<td><xsl:value-of select="titulo"/></td>
<td><xsl:value-of select="autor"/></td>
<td><xsl:value-of select="isbn"/></td>
<td><xsl:value-of select="anyo"/></td>
</tr>
</xsl:for-each>
</table>

EJERCICIO: PRESENTACION.XSL
</body>
</html>
</xsl:template>
</xsl:stylesheet>

Observe en el listado anterior que luego de la etiqueta <body> existe el título de la página y luego se dibuja una tabla. Entre los tags principales de este código se tiene:

- table border: borde de la tabla
- th align=”left”: para alinear a la izquierda el encabezado (header) de la tabla. El título del encabezado es *Mis publicaciones.*
- tr: se utiliza para definir elementos de una tabla HTML. Ver http://www.htmlquick.com/es/tutorials/tags-attributes.html para tener más descripciones sobre tags HTML.
- <td><xsl:value-of select="titulo"/></td> es importante aquí dado que representa el procesamiento del valor del elemento titulo que se encuentra en el documento xml. Viene antecedido por <xsl:for-each select="libros/libro"> que el procesador que permite recorrer toda la lista de elementos del documento xml. Este cierra con </xsl:for-each>.

9. Ahora se puede visualizar en un navegador web su archivo xml (*xmlLibros.xml)*, utilizando el archivo de formateador de estilo (*Presentacion.xsl*). Para ello, pulse con el botón derecho del mouse sobre el archivo *xmlLibros.xml* y seleccione *Vista*. Se ejecutará su documento xml en el navegador, siendo similar a la figura 6.12.

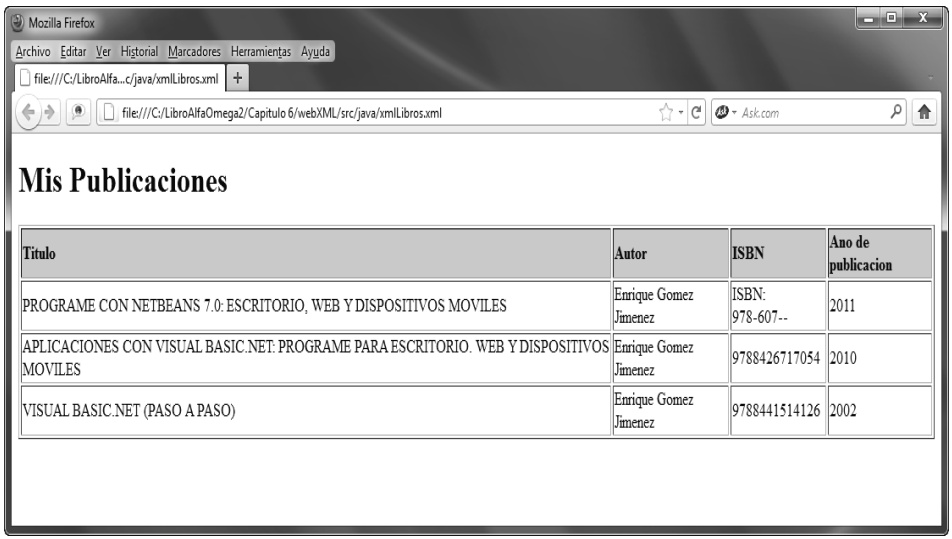


Figura 6.12 Ejecución del archivo xmlLibros.xml.

Crear un archivo XML con DOM

Ahora compete crear un archivo XML con Java. Para ello se creará un proyecto de tipo Aplicación Java denominado *JavaDOMXML*, cuyo código fuente se cita a continuación:

EJERCICIO: JAVADOMXML.JAVA
package javadomxml;
import java.io.*;
import java.util.Vector;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;
import org.jdom.JDOMException;
import org.w3c.dom.*;
import org.xml.sax.SAXException;
public class JavaDOMXML {
static void crearXML() throws IOException, ParserConfigurationException, TransformerConfigurationException, TransformerException{
Vector v = new Vector(); //Vector para almacenar los nombres de los atributos.
String archXML;
BufferedReader bfr = new BufferedReader(new InputStreamReader(System.in));
System.out.print("Ingrese el nombre del elemento raíz (root): ");
String raiz = bfr.readLine();
DocumentBuilderFactory Archivo = DocumentBuilderFactory.newInstance();
DocumentBuilder documentBuilder = Archivo.newDocumentBuilder();
Document documento = documentBuilder.newDocument();
Element elementoRaiz = documento.createElement(raiz);
documento.appendChild(elementoRaiz);
System.out.print("Número de atributos a crear en su archivo XML : ");
String str = bfr.readLine();

EJERCICIO: JAVADOMXML.JAVA
int nElementos = Integer.parseInt(str);
//el siguiente método agrega los atributos, según cuántos decidió el usuario crear
for (int j=1;j<=nElementos;j++){
System.out.print("Nombre del atributo : "+String.valueOf(j)+" ");
str = bfr.readLine();
v.addElement(str); }
//se procede a llenar el contenido de cada atributo creado.
for (int i = 0; i < nElementos; i++){
Element atributo = documento.createElement(v.elementAt(i).toString());
System.out.print("Digite el contenido de: " + v.elementAt(i)+ " ");
String data = bfr.readLine();
atributo.appendChild(documento.createTextNode(data));
elementoRaiz.appendChild(atributo); }
TransformerFactory transformerFactory = TransformerFactory.newInstance();
Transformer transformer = transformerFactory.newTransformer();
DOMSource source = new DOMSource(documento);
System.out.println("Nombre del archivo XML sin la extension XML");
archXML = bfr.readLine();
StreamResult pArchivo = new StreamResult(new File(archXML+".xml"));
StreamResult pPantalla = new StreamResult(System.out);
transformer.transform(source, pArchivo);
transformer.transform(source, pPantalla); }
public static void main(String[] args) throws IOException, ParserConfigurationException, TransformerConfigurationException, TransformerException, SAXException {
crearXML();
}}

La ejecución del código anterior generará una salida similar a la que se muestra en el texto siguiente:

```
run:
Ingrese el nombre del elemento raíz (root): EMPLEADOS
Número de atributos a crear en su archivo XML : 3
Nombre del atributo: 1 Nombre
Nombre del atributo: 2 Apellidos
Nombre del atributo: 3 Edad
Digite el contenido de: Nombre Enrique
Digite el contenido de: Apellidos Gomez Jimenez
Digite el contenido de: Edad 46
Nombre del archivo XML sin la extension XML
xmlEmpleados
<?xml version="1.0" encoding="UTF-8"
standalone="no"?><EMPLEADOS><Nombre>Enrique</Nombre><Apellidos>Gomez
Jimenez</Apellidos><Edad>46</Edad></EMPLEADOS>GENERACIÓN CORRECTA
(total time: 55 seconds)
```

Como podrá observar se crea el atributo raíz (EMPLEADOS) y posteriormente los nombres de atributos. Finalmente, se agrega el valor a cada uno de los atributos que se crearon con antelación. El archivo generado en disco contendría los siguientes datos:

```
<?xml version="1.0" encoding="UTF-8"?>
-<EMPLEADOS>
   <Nombre>Enrique</Nombre>
   <Apellidos>Gomez Jimenez</Apellidos>
   <Edad>46</Edad>
</EMPLEADOS>
```

Actividades para el lector

Modifique el código anterior para que el programa cree tantos elementos (con los atributos establecidos) como el usuario requiera.

XOM

XOM constituye un nuevo modelo de objeto XML. Es una API de código abierto (tipo LGPL) que se usa para el procesamiento de archivos XML utilizando Java. Solamente acepta la creación de espacio de nombres y documentos XML. XOM puede ser descargado y agregado con una API de NetBeans desde

http://www.xom.nu/

Algunas de las características principales de esta API son las siguientes:

- Mientras el árbol del documento XML está siendo construido, se pueden procesar los nodos individuales del mismo.
- XOM optimiza el uso de la memoria, construyendo las partes de un árbol que se requiere y omitir algunas partes innecesarias.
- XOM incluye soporte para tecnologías XML tales como XPath, XSLT, XInclude, entre otras.
- Los documentos XOM se pueden convertir en y desde SAX y DOM.
- XOM depende de un analizador SAX subyacente para leer y completar documentos XML, tal como Xerces.

Ejemplo utilizando XOM

El siguiente programa construye un archivo XML utilizando el API XOM.

EJERCICIO: JAVAXOM.JAVA
package javaxom;
import java.io.IOException;
import nu.xom.Builder;
import nu.xom.Document;
import nu.xom.Element;
import nu.xom.ParsingException;
import nu.xom.Serializer;
public class JavaXOM {
static void leer(){
try {
//se crea el constructor del parser. Builder parser = new Builder();
//construye el documento sobre el cual se analizará el sitio web www.xom.nu Document doc = parser.build("http://www.xom.nu/");

//La serialización de un objeto se produce para obtener la secuencia de bytes que
//representa dicho objeto. Esta secuencia puede almacenarse, enviarse a través
//de la red o para recuperar el objeto inicial, entre otras operaciones.
//para ello se utiliza el objeto Serializer.
Serializer serializer = new Serializer(System.out, "ISO-8859-1");

serializer.setIndent(4);
serializer.setMaxLength(64);

serializer.write(doc); } //escribe la serialización del objeto (la página web
// www.xom.nu) en el Document doc.

EJERCICIO: JAVAXOM.JAVA
catch (ParsingException ex) {
System.err.println("www.xom esta mal formada. que embarazoso!"); }
catch (IOException ex) {
System.err.println("No se pudo conectar con el sitio .." +ex.getMessage());}}
private static void crearXML() throws IOException{
String empleados [] = {"Jose Carlos","Jose Enrique","Gabriela","Luz Marina"};
String apellidos [] = {"Gomez Duarte","Gomez Duarte","Gomez Duarte","Duarte Quintanilla"}; //vector de apellidos. La línea anterior era el vector de nombres
Element root = new Element("Empleados"); //se crea el nodo raíz
for (int i = 0; i <= empleados.length-1; i++) { //se crean los elementos del archivo
Element Nombre = new Element("Nombre"); //se crea el nombre del atributo
Nombre.appendChild(empleados[i]); //se agrega el contenido del atributo creado
root.appendChild(Nombre); //se agrega al nodo raíz del árbol xml.
Element Apellidos = new Element("Apellidos"); //se agrega otro atributo
Apellidos.appendChild(apellidos[i]); //se carga el contenido del atributo creado.
root.appendChild(Apellidos); } //se agrega al nodo raíz
Document doc = new Document(root); //se crea un documento xml con el nodo raíz
System.out.println(doc.toXML()); //salida del documento xml con los nodos
try {
Serializer serializer = new Serializer(System.out, "ISO-8859-1");
serializer.setIndent(4);
serializer.setMaxLength(64);
serializer.write(doc); }
catch (IOException ex) {
System.err.println(ex); }}
public static void main(String[] args) throws IOException {
crearXML();
//leer();
}
}

La salida al ejecutar este programa se observa en la figura 6.13.

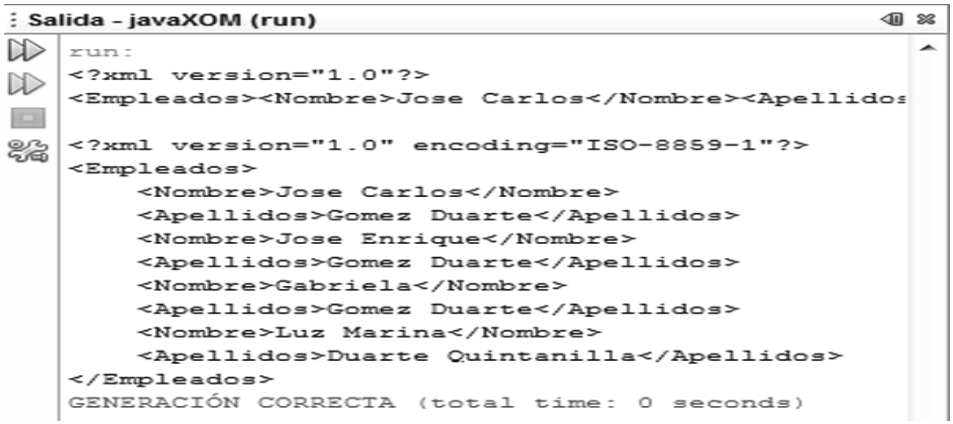


Figura 6.13 Creación de un archivo XML con XOM.

La segunda parte del método main() contiene una llamada a leer. Básicamente lo que hace este procedimiento es serializar a través de un archivo XML la conformación de un sitio web determinado. En este caso, se trata del sitio http://www.xom.nu/, creador del API XOM. Observe que está en comentario, sin embargo, si lo ejecuta podría extraer de ese sitio web la información relativa al XML que conforma esa página.

Actividades para el lector

1. Modifique el código anterior para que el programa cree tantos elementos (con los atributos establecidos) como el usuario requiera.
2. Intente la implementación de Xerces como parser, según código en *http://casidiablo.net/procesamiento-xml-java-xerces/*

Ajax

AJAX es sinónimo de *Asynchronous JavaScript and XML*. Las nuevas experiencias de accesibilidad requeridas en las aplicaciones web modernos hacen de AJAX una opción interesante para el procesamiento local de lógica de negocios en aplicaciones web.

Se sabe que del lado del cliente pueden implementarse componentes JavaScript, básicamente en el navegador, para la interacción cliente–servidor (procesamiento y respuestas a solicitudes). También se dispone de importantes frameworks como Prototype, Dojo, Yahoo, UI (YUI) Toolkit o Google Web Toolkit.

Asincrónico significa que una vez que se han enviado los datos al servidor, el cliente puede continuar su trabajo de procesamiento mientras el servidor lo hace en segundo plano, no requiriéndose esperar por una respuesta. En el caso de

GoogleMaps, por ejemplo, podría moverse por todo el mapa con el puntero del mouse y seleccionar una parte del mismo sin que deba actualizarse todo el mapa (utiliza AJAX), mientras que en un aplicativo web tradicional sincrónico debería esperarse a que toda la página se refrescara.

En realidad AJAX no es una tecnología, sino más bien la conjunción de varias librerías: Java y XML, trabajando asincrónicamente. Con AJAX se han podido desarrollar aplicaciones impresionantes como GoogleMaps, Gmail o el mismo Outlook Web Access. El objetivo principal de AJAX es cargar y renderizar páginas web, trabajando a nivel de cliente (lógica de negocio) y actualizando cuando sea necesario y, en algunas ocasiones, sólo aquellas partes que sean necesarias actualizar. ¿Acaso no está acostumbrado a ver el intellisense adelantándose en su criterio de búsqueda en Google (Google Suggest)? ¡GoogleMaps qué facilidad tiene de arrastrar y hacer zoom en una determinada localidad geográfica! Todo ello es una muestra de las funcionalidades actuales que pueden ser desarrolladas en aplicaciones web con AJAX.

AJAX permite:

- Crear presentaciones basadas en estándares utilizando XHTML y CSS.
- Interactuar y visualizar datos en forma dinámica empleando DOM.
- Intercambiar y manipular datos usando XML y XSLT.
- Recuperar datos en forma asincrónica utilizando XMLHttpRequest.
- Utilizar JavaScript.

La lógica de funcionamiento de un aplicativo web es la siguiente:

- El usuario a través de la interface de una página web lanza un requerimiento HTTP al servidor web.
- El servidor efectúa un proceso de recopilación, procesamiento, entre otras operaciones, y devuelve una página HTML al cliente.
- Se inicia la interacción e intercambio de información entre el servidor y la máquina del cliente.

Es una situación muy evidente pero que requiere de mucho dinamismo y uso de recursos entre ambos extremos. Quizá, en este proceso el más afectado sea el servidor, dado que no solamente trabaja con un usuario sino que lo hace con una gran cantidad de ellos. Lógicamente, como desarrolladores de software es necesario contar con una estrategia para poder disminuir ese procesamiento tan excesivo que pueda producirse en esa interacción cliente-servidor. AJAX representa una de esas estrategias. La figura 6.14 muestra la diferencia entre un modelo clásico de aplicaciones web y el modelo utilizado por AJAX.

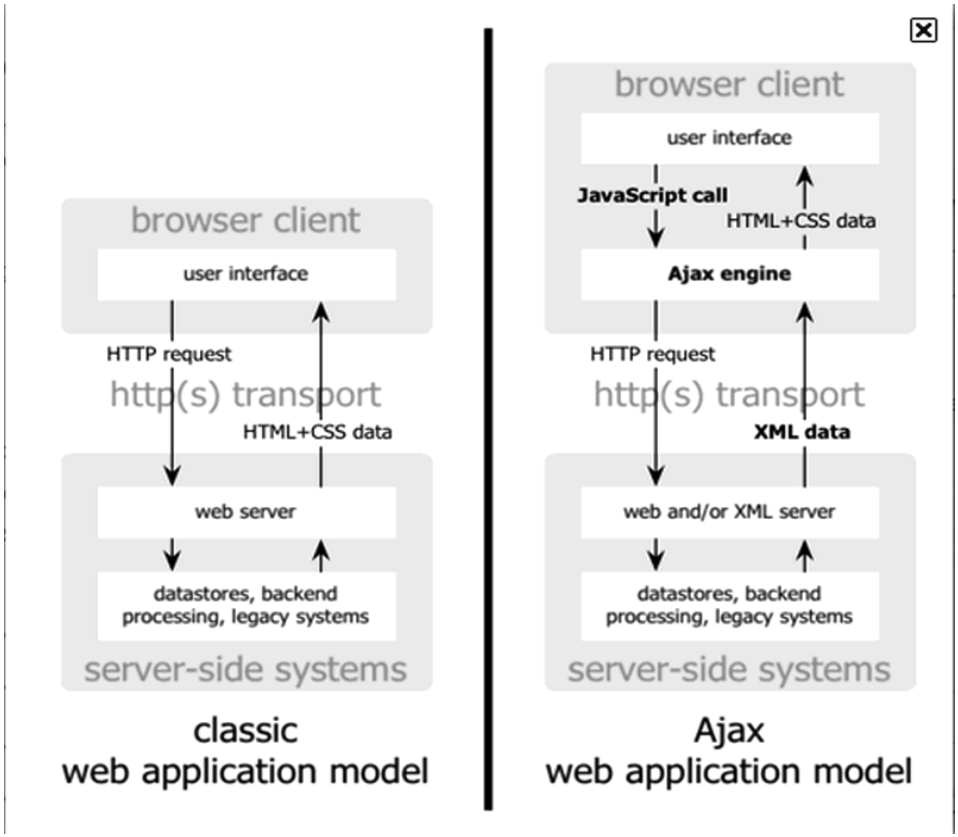


Figura 6.14 Modelo tradicional versus modelo AJAX en aplicaciones web, extraído desde http://www.maestrosdelweb.com/editorial/ajax/ el 7 de diciembre de 2011.

Como se observa en la figura 6.14 se implementa el motor AJAX entre los datos XML que genera un aplicativo web en el servidor y los datos HTML+CSS que se producen en la máquina del cliente.

En el cliente se produce el manejo de lógica de cliente mediante JavaScript, validando, procesando o calculando datos, para luego enviarlos al servidor nuevamente. Por ende, mucho del proceso que se produce en el aplicativo es tratado en esta máquina del cliente, evitando la sobrecarga de solicitudes al servidor.

En el caso de Java, los componentes que soportan AJAX son los JavaServer Faces, los cuales se desarrollarán en un capítulo aparte. Estos componentes encapsulan código JavaScript para ser utilizados en aplicaciones web. En la figura 6.15 se observa el funcionamiento de JavaScript dentro de la filosofía de trabajo de AJAX.

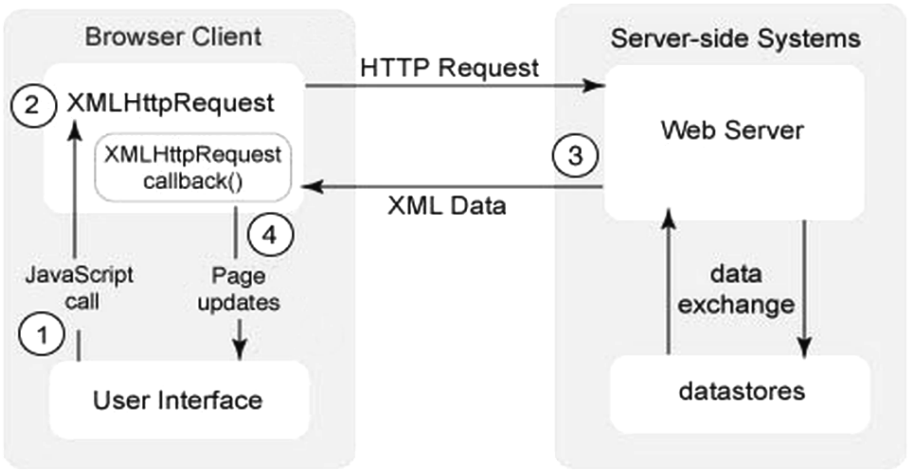


Figura 6.15 JavaScript como parte de AJAX, extraído el 7 de diciembre de 2011 de http://java.sun.com/developer/technicalArticles/J2EE/AJAX/IntroAjaxPageAuthors.html

En la figura 6.15 es posible observar que hay una división de capas (navegador del cliente y el servidor de web). Existe una solicitud del usuario (HTTP Request) a la cual el servidor web responde con información en formato XML. En el lado del cliente existe un procesamiento entre la interface de usuario (la cual sufre la representación parcial de página) y existe un proceso XMLHttpRequest para el manejo de solicitudes al servidor.

En los siguientes capítulos se retomará el concepto de AJAX en el desarrollo de aplicaciones JavaServer Faces.

Actividades para el lector

1. Cree un mapa conceptual de las tecnologías basadas en el lenguaje de marcas explicadas en este capítulo. Incluya la tecnología y su funcionamiento.
2. Investigue qué framework se ofrece para dar soporte a las aplicaciones web que se desarrollan en Java; presente un reporte escrito.

RESUMEN

En este capítulo se desarrollan los temas relacionados con los fundamentos de la programación web; no se tratan profundamente dado que la intención es introducir al lector en ellos. Tal es el caso de la tecnológica XML, que sin detallar mucho en la manera de implementarla mediante un sitio JSP, sí se consideró su manejo básico con Java. Se trataron los siguientes temas:

1. Fundamentación de la programación web.
2. Habilitación de los servidores Tomcat y GlassFish para el desarrollo de aplicaciones web.
3. Aproximación a la comunicación por Internet (objetos y métodos de comunicación web).
4. Tecnologías basadas en lenguajes de marcas (SGML, HTML, XHTML).
5. AJAX

Autoevaluación

1. ¿Para qué se utiliza el objeto URL en la comunicación web?
2. ¿Para qué se utiliza el método getContent en la comunicación web?
3. Cite tres tipos de marcado en las tecnologías de lenguajes de marcas.
4. Defina qué es SGML
5. ¿Cuál es la inconveniencia de utilizar SGML?
6. ¿Para qué se utiliza Canvas en HTML5?
7. ¿Qué es xhtml?
8. ¿Qué es XML?
9. ¿Qué es JDOM?
10. ¿Qué es XOM?
11. Defina qué es AJAX.

EVIDENCIA

- [ ] Habilitó los servidores Tomcat y GlassFish en su instalación de Netbeans 7.1.

- [ ] Realizó un aplicativo HTML 4 para crear un formulario para el registro de estudiantes. Describió qué materias académicas son de su agrado.

- [ ] Analizó e implementó el código del ejemplo de HTML 5 del sitio http://www.ibm.com/developerworks/ssa/web/tutorials/wa-html5/downloads.html

- [ ] Creó un mapa conceptual de las tecnologías basadas en el lenguaje de marcas explicadas en este capítulo. Incluya la tecnología y su funcionamiento.

- [ ] Investigó qué framework se ofrece en el mercado para dar soporte a las aplicaciones web que se desarrollan en Java.

REFERENCIAS

Bibliografía

Arce, Fco. Javier, (2011). *ActionScript 3.0 Aprenda a programar*. Alfaomega, México.

Ceballos, Fco. Javier, (2004). *Programación orientada a objetos con C++*. 3a. ed., Alfaomega, México.

Ceballos, Fco. Javier, (2008). *Java 2. Interfaces gráficas y aplicaciones para internet*. 3a. ed., Alfaomega, México.

Ceballos, Fco. Javier, (2011). *Java 2: Curso de programación*, 4a. ed., Alfaomega, México.

Martín, Antonio, (2010). *Programador certificado Java 2. Curso práctico*, 3a. ed., Alfaomega, México.

Rozannski, Uwe, (2010). *Enterprise Javabeans 3.0. Con Eclipse Y JBoss*. Alfaomega, México.

Sznajdleder, Pablo, (2010) *Java a fondo. Estudio del lenguaje y desarrollo de aplicaciones.* Alfaomega, México.

Gómez Jiménez, Enrique (2010) *Aplicaciones con Visual Basic .NET: Programe para escritorio, web y dispositivos móviles*. Alfaomega, México.

Páginas Web recomendadas

1. htmlgoodies.com (2011) *HTMLGoodies. The ultimate resource*. Obtenido el 10 de noviembre del 2011 desde http://www.htmlgoodies.com/beyond/java
2. *programacion.com (2011) APIs de Java para XML*. Obtenido el 10 de noviembre del 2011 desde http://www.programacion.com/articulo/apis_de_java_para_xml_112
3. javahispano (2011) *XML desde java hecho fácil: JDOM.* Obtenido el 12 de noviembre del 2011 desde http://www.javahispano.org/contenidos/es/xml_desde_java_hecho_fcil_jdom/
4. chuwiki.chuidiang.org (2011) *Ficheros XML*. Obtenido el 12 de noviembre del 2011 desde http://chuwiki.chuidiang.org/index.php?title=Ficheros_XML
5. [J]Zoom (2002) *Tutorial : XML generation with JAVA*. Obtenido el 12 de noviembre del 2011 desde http://www.javazoom.net/services/newsletter/xmlgeneration.html
6. ibiblio.org (2002) *Processing XML with Java.* Obtenido el 14 de noviembre del 2011 desde http://www.ibiblio.org/xml/books/xmljava/
7. maestrosdelweb.com (2005) *Ajax: Un nuevo acercamiento a las aplicaciones web*. Obtenido el 14 de noviembre del 2011 desde http://www.maestrosdelweb.com/editorial/ajax/
8. java.sun.com (2010) *Introduction to Ajax for Page Authors* . Obtenido el 14 de noviembre del 2011 desde http://java.sun.com/developer/technicalArticles/J2EE/AJAX/IntroAjaxPageAuthors.html

Respuestas a la autoevaluación

1. Un objeto URL especifica la dirección de la página web que debe cargarse.
2. El método getContent se utiliza para realizar la lectura de datos de una página web (streams de datos).
3. a) marcado de tipos, b) marcado de procedimientos y c) marcado descriptivo.
4. SGML es un metalenguaje orientado a definir lenguajes de marcado. Son las siglas de *Standard Generalized Markup Language* o "Estándar de Lenguaje de Marcado Generalizado".
5. La inconveniencia de utilizar SGML es la complejidad para la presentación de grandes cantidades de información.
6. Canvas en HTML5 se usa para la creación de dibujos en 2D.
7. XHTML es una reformulación de HTML y XML. Emplea XML para el formalismo de su implementación y HTML para su presentación.
8. XML constituye un metalenguaje extendido que se basa en etiquetas.
9. JDOM es una API que se utiliza para la creación y manipulación de documentos XML.
10. XOM constituye una API de código abierto que se usa para el procesamiento de archivos XML.
11. AJAX es la conjunción de varias tecnologías (JAVA y XML principalmente) para el manejo asincrónico de la comunicación entre un servidor web y sus clientes.

Patrones de diseño en ingeniería Web

7

Reflexione y responda las siguientes preguntas

¿Qué significa el término reusabilidad?
¿Qué son los patrones de diseño?
¿Cómo vincula los patrones de diseño con la reusabilidad?
¿Cómo relaciona los patrones de diseño con las ciencias informáticas?

Contenido

EXPECTATIVA

Es necesario conocer cómo se establecen los patrones de diseño y toda la infraestructura que ampara su filosofía. Como desarrolladores de software es importante no solamente conocer cómo se implementa mediante una herramienta de desarrollo, sino también toda la filosofía que se encuentra detrás de esta implementación.

Los patrones de diseño constituyen abstracciones que permiten la solución de problemas en muchos niveles en el desarrollo de software. Pueden estar implícitos en diferentes etapas del desarrollo de un sistema computacional: desde el análisis, diseño, arquitectura y hasta la propia implementación. Es importante considerar que existen muchas funciones que son muy repetitivas en un contexto de negocios, con muchas características similares y que bien podrían implementarse en un solo proceso: un patrón. Esta práctica permite la reutilización de software y el mejoramiento de la calidad al utilizar componentes de software ampliamente probados.

Después de estudiar este capítulo, el lector será capaz de

- Conocer los diferentes métodos que se utilizan para el desarrollo de aplicaciones Web.
- Comprender el funcionamiento de los patrones de diseño que se utilizan para el desarrollo rápido de software para Web
- Comprender el funcionamiento del modelo MVC como patrón de diseño

INTRODUCCIÓN

Los patrones de diseño están inmersos en la ingeniería web para procurar un desarrollo rápido y de mayor calidad. ¿Por qué rápido? La pregunta parece esconder una respuesta difícil de responder, sin embargo nada más iluso que eso, dado que desde hace muchísimo tiempo se emplea patrones de diseño, quizás sin saber que eso que aplicaba se llama así. La teoría afirma que un patrón de diseño debe haber comprobado su efectividad resolviendo problemas similares y que sea reutilizable. Entonces, ¿cuántas veces ha desarrollado algún tipo de software o componente de software que lo haya usado tantas veces como fuese necesario y siempre con seguridad? Por ejemplo, componentes que a veces constituyen componentes genéricos de software como puede ser una ventana que procesa la autenticación de un usuario en un sistema, que es la misma que se utiliza en otros sistemas.

En este capítulo se incluye el tema de patrones de diseño en la ingeniería web, dada la importancia que reviste para el desarrollador moderno de software, principalmente aquellas que se orientan a la web. En esta época donde el desarrollo de software debe ser lo más rápido posible, pero con calidad, el uso de patrones de diseño, entendido en la reutilización de software, se vuelve cada vez más necesario.

Métodos de desarrollo de sistemas Web

Antes de desarrollar el tema de los patrones de diseño, es importante conocer algunos modelos para la creación de un sistema web. Entre estos sistemas se encuentran WSDM, webML y UWE. Cada uno representa una implementación ligeramente distinta entre sí, por lo que es necesario resumirlos brevemente.

Web Site Design Method (WSDM)

Este modelo fue creado por De Troyer y Leune en 1998. Representa un método que se centra en el usuario más que en los datos. Está orientado hacia el desarrollo de sistemas web que sean personalizables, modelándolo desde la fase esencial donde los actores principales son los usuarios.

Más que un método de diseño, WSDM es una metodología que provee primitivas de modelado que permiten al desarrollador web describir una aplicación desde diferentes perspectivas y niveles de abstracción. Gustavo Rossi escribe al respecto lo siguiente: "…el desarrollo de un sitio web/aplicación desde diferentes perspectivas y en varios niveles de abstracción, […] también proporciona una manera sistemática para desarrollar la aplicación web. Desarrollo de un sitio web/aplicación con WSDM comienza con la formulación de la declaración de la llamada misión y sigue una filosofía de diseño bien definido que ofrece al diseñador la necesario para estructurar el sitio web." (Rossi y otros, 2008)

Para el desarrollo de un sistema web con este método se utilizan cuatro fases, tal y como se muestra en la figura 7.1.

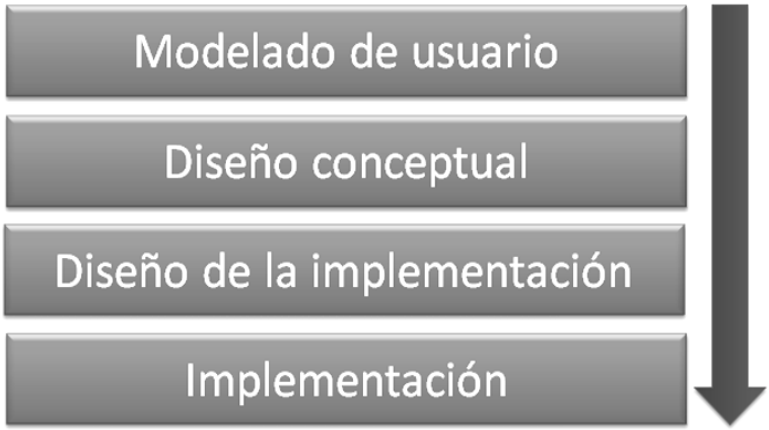


Figura 7.1 Fases para el desarrollo de sistemas web mediante WSDM.

Detallando el modelo, se podría especificar cada fase de la siguiente manera:

- **Fase modelado de usuario:** se crean las clases de acuerdo con la categorización de los actores involucrados en el sistema. Cada usuario o grupo de usuarios posee requisitos muy bien determinados de usabilidad.
- **Fase diseño conceptual:** en esta fase se modelan los requisitos de información de cada usuario o grupo de usuarios. Asimismo, en esta fase se crea un Modelo de Objetos de Usuario y un Modelo de Navegación, considerando cada clase creada en la fase anterior.
- **Fase diseño de implementación:** se modela la interface que tendrá la aplicación (apariencia) según los requisitos de usabilidad. Se utilizan las herramientas predeterminadas para este modelo.

Rossi y otros detallan aún más el modelo al afirmar que: "El desarrollo de un sitio web y aplicaciones con WSDM comienzan con la formulación de la declaración de la llamada misión y sigue una filosofía de diseño bien definido que ofrece al diseñador lo necesario para el soporte a la estructura del sitio web" (Rossi y otros, 2008). Esta afirmación enfatiza que es necesario iniciar el análisis del desarrollo de la aplicación web con el enunciado o la especificación de la visión del sitio. Esta especificación de la visión debe obtenerse de la audiencia (los actores) involucrada en el desarrollo de sitio, para lo cual debe modelarse esta audiencia clasificándola y caracterizándola. La clasificación de la audiencia se basa en los requerimientos de los diferentes usuarios involucrados en el desarrollo de la aplicación web.

Posteriormente, según Rossi, se realiza el diseño conceptual del sistema, creando las tareas necesarias y modelando la información obtenida de la fase de fase de modelado de la audiencia. El diseño conceptual se utiliza para especificar la información, la funcionalidad y la estructura del sistema web a un nivel abstracto

(conceptual). La funcionalidad y la información a manejar en la aplicación web se especifican durante el subproceso de tareas y modelado de la información. En esta fase también se diseña la navegación de las opciones de la aplicación web, considerando las distintas audiencias (usuarios del sistema).

Finalmente, se diseña la implementación de la solución, creando tres sub-productos esenciales de la aplicación: diseño de la estructura del sitio, diseño de la presentación (interfaces) y el diseño de la lógica de datos. Luego resta nada más que implementar el producto. La estructura de un sitio web son los contenidos que tendrá el sitio web y dependerá en gran medida de esos contenidos. Con ello podríamos decidir qué tipo de estructura tendrá el sitio: árbol o jerárquica (página de bienvenida, que abre luego las secciones y éstas las páginas que las soportarán), lineal (página de bienvenida y luego las demás paginas serán en forma secuencial como si fuera un libro) o en red (donde cada página alberga su propio contenido). Las interfaces se crearán posteriormente como elementos esenciales del sitio dado que será la cara que verá el usuario en el sitio web. Esta interface deberá considerar algunos temas de diseño tales como diseño equilibrado, teoría del color o tipografía, además de algunos estándares de la W3C. Por último, la lógica de datos se refiere a la estrategia de accesibilidad y mantenimiento de los datos de la aplicación. Se considera además la seguridad y algunos aspectos propios del manejo de datos. La figura 7.2 muestra las distintas fases en que se divide este modelo, incluyendo los subprocesos que contempla cada fase.

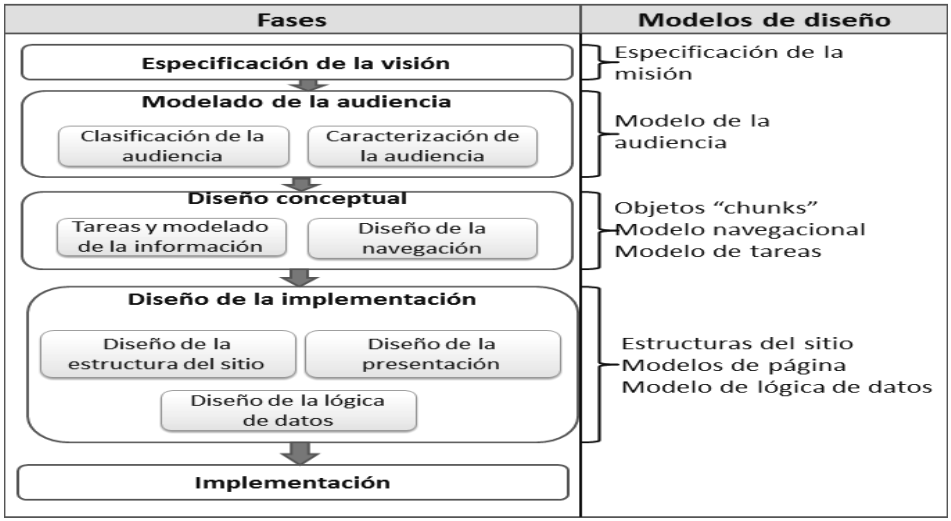


Figura 7.2 Fases para el desarrollo de sistemas web mediante WSDM. Adaptado de Rossi y otros (2008).

Web Modeling Language (webML)

Se orienta a la creación de sitios web no considerando las especificaciones de la arquitectura del sistema, sino que se basa en una perspectiva de alto nivel. Transita desde la recolección de requisitos hasta el diseño mismo de la aplicación, pasando por una serie de actividades interactivas. Cada ciclo de diseño representa una parte del proceso global. La figura 7.3 muestra la manera en que se lleva a cabo este método.

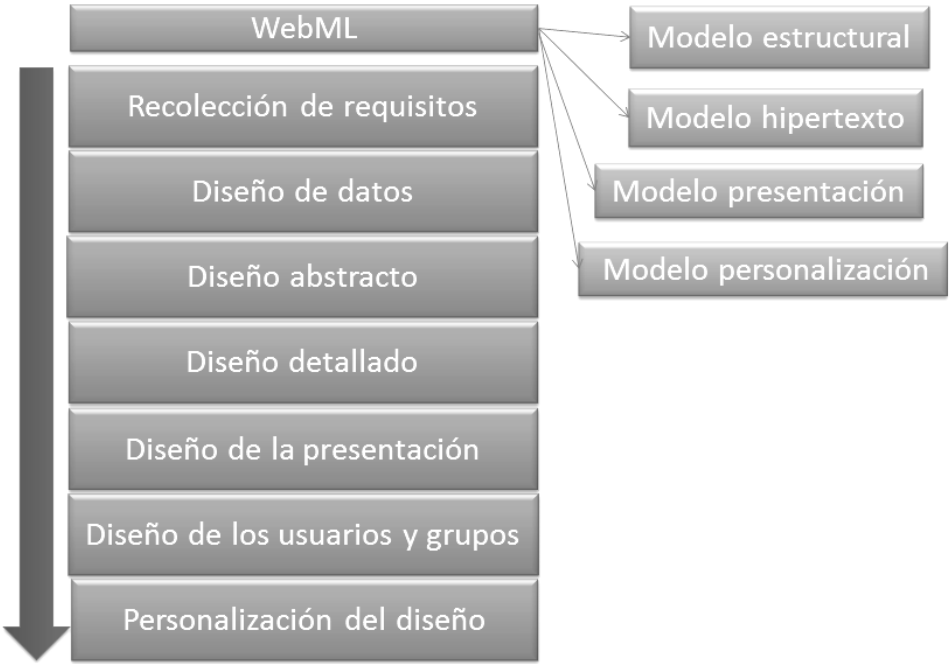


Figura 7.3 Fases para el desarrollo de sistemas web mediante webML.

Como puede observarse en la figura 7.3, el método WebML se vale de varias actividades: recolección de requisitos, diseño de datos, diseño abstracto, diseño detallado, diseño de la presentación, diseño de los usuarios y grupos y personalización del diseño. Los modelos, por su parte, definen su funcionalidad de la siguiente forma:

- El **modelo estructural** representa los datos y las relaciones que se establecen entre ellos. Se puede utilizar un modelo E-R o uno de clases UML para su representación.

- El **modelo de hipertexto** representa las páginas y su contenido y los enlaces que se establecen entre ellas. Incluye los submodelos que precisan esto último: composición y navegación.
- El **modelo de presentación** se encarga de la apariencia que tendrán las páginas independientemente de la herramienta que se utilizará para diseñarlas, basando su labor en sintaxis XML.
- El **modelo de personalización** considera la estructura en la cual se definieron los tipos de usuarios y las reglas asociadas a los contenidos que pueden acceder en el sitio.

Según Stefano Ceri, la implementación de webML en una arquitectura MVC: "Webml consiste en simples conceptos visuales para expresar hipertexto como un conjunto de páginas de operaciones y unidades de contenido vinculadas que se utilizan para enlazar tales operaciones y unidades de contenido a los datos que se refieren" Stefano Ceri y otros (2003). También podemos entender que WebML sigue el estilo de lenguajes de modelado conceptual conocido como relación de entidades y UML: cada concepto tiene una representación gráfica y las especificaciones son diagramas. Por lo tanto, el lector no debe preocuparse acerca de la necesidad de aprender otro idioma.

En su implementación, mediante Java y utilizando la arquitectura MVC, cada página webML es mapeada en cuatro elementos, según Stefano Ceri y otros: a) una página de acción, b) una página de servicio en la capa de negocios, c) una plantilla JSP en la vista y d) una página de mapeo de las acciones en el archivo de configuración del controlador. El accionar de este procedimiento es el siguiente:

1. La página de acción es una instancia de la clase *action*. Ésta extrae la entrada de la solicitud HTTP y llama a la página de servicio en la capa de negocios, pasándole los parámetros necesarios. Cuando la invocación a la página de servicios finaliza, la página de acción notifica al controlador del resultado de la computación de la página.
2. La página de servicios es una función de negocio que soporta el proceso computacional de una página. Ésta expone una función simple denominada *computePage()*, invocada para realizar la propagación del parámetro. La página de servicios actualiza el estado del objeto en el modelo: al final de la ejecución de la página de servicios todos los JavaBeans almacenan los datos obtenidos de las consultas, los cuales son suministrados a las vistas.
3. Las plantillas en la vista procesan la página HTML que será presentada al usuario, basado esto en el contenido del modelo.
4. La asignación de acción es una declaración en el archivo de configuración del controlador, que a la vez une la solicitud del usuario, la página de acción y la vista de página.

Para obtener una representación visual de lo anteriormente citado considérese la figura 7.4, donde se explica en forma gráfica. Puede encontrar documentos, tutoriales, ejemplos, entre otros, en http://www.webml.org/webml/page1.do y en http://dbgroup.como.polimi.it/brambilla/webml.

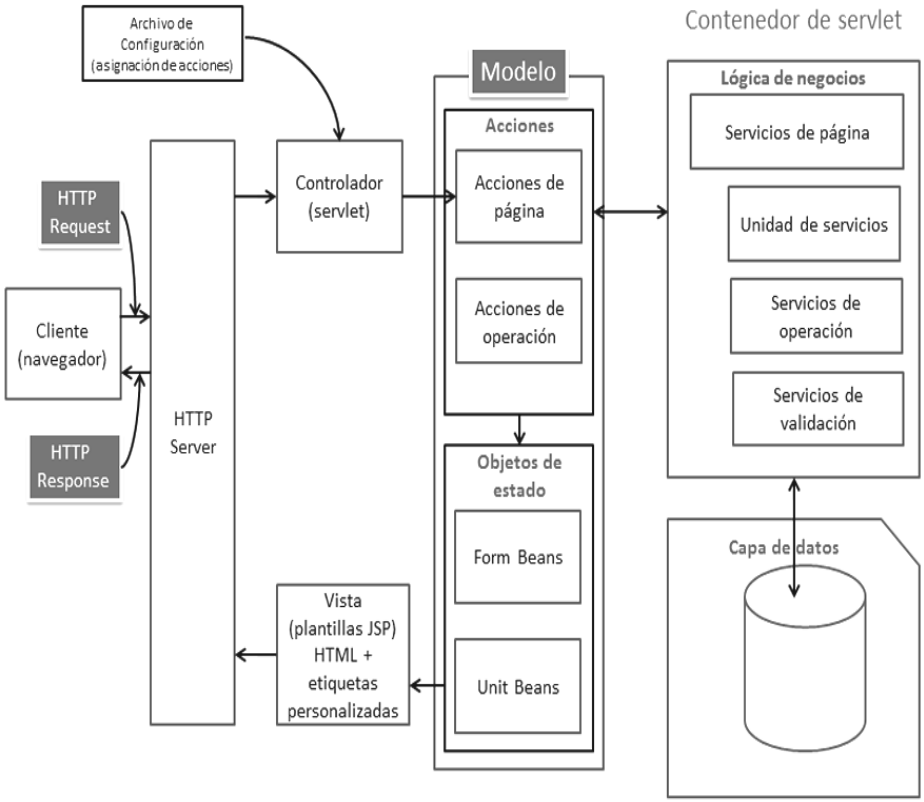


Figura 7.4 Funcionalidad webML en una arquitectura MVC. Adaptado de Stefano (2003).

UML-based Web Engineering methodology (UWE)

La metodología de diseño de UWE se basa en un metamodelo que se define como una ligera extensión del UML. Comprende el modelado separado de los diferentes aspectos de una aplicación web: contenido, estructura, composición y lógica empresarial. Surgió a finales de los años noventa bajo la idea de crear una forma estándar para la construcción de modelos de análisis y diseño de sistemas orientados a la web.

Se orienta a la creación de aplicaciones de multimedia. Utiliza las técnicas de la orientación a objetos para la especificación de requisitos. Fundamentalmente, se

basa en actividades como análisis de requisitos y el diseño conceptual, la navegación y la presentación.

En el contexto de UWE los elementos de hipermedia se presentan mediante diagramas de clases UML (nodos son clases, las relaciones son asociaciones estereotipadas) y el dinamismo se representa mediante diagramas de estados.

UWE adapta, por un lado, las nuevas características del sistema web, tales como aplicaciones basadas en transacciones, aplicaciones personalizadas, aplicaciones dependientes del contexto y aplicaciones asincrónicas. Por otro lado, UWE evoluciona incorporando técnicas de ingeniería de software, tales como orientación a aspectos de modelado, integración del modelo utilizando hugo/RT y los nuevos lenguajes de transformación de modelos para mejorar el diseño y la calidad del mismo. Thomas Baar visualiza la extensión UML implementada en UWE según la figura 7.5.

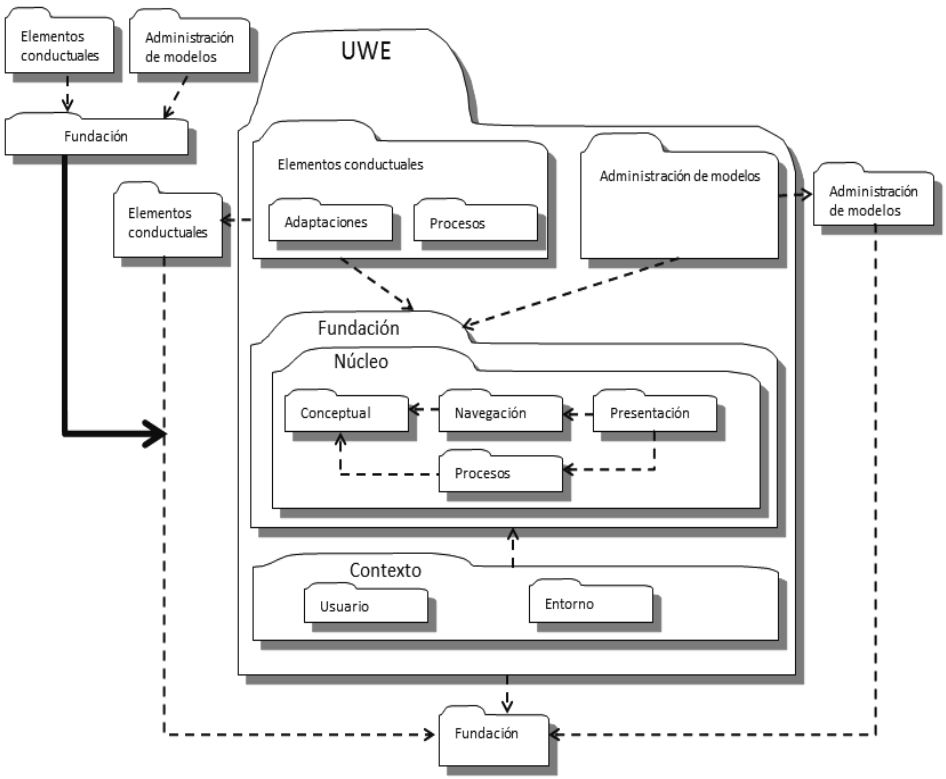


Figura 7.5 Metamodelo UWE incrustado en el metamodelo UML. Adaptado de Thomas Baar (2004).

Según la figura 7.5, se puede conceptualizar que la extensión del metamodelo UML consiste en agregar un paquete UWE de nivel superior a los tres paquetes de nivel superior de UML. Este paquete de UWE contiene los elementos del modelo distribuidos en sub-paquetes. La estructura de los paquetes dentro del paquete de UWE es análoga para el paquete de nivel superior de UML. El paquete *Fundación* contiene todos los elementos básicos del modelo estático, los elementos de comportamiento del paquete dependen de ella y contiene todos los elementos para modelado de comportamiento. Finalmente, el paquete Administración de Modelos, que también depende del paquete de Fundación, contiene todos los elementos para describir los modelos propios específicos de UWE.

UWE UML consta de seis modelos, a saber:

1. Modelo de caso de uso, que se utiliza para capturar los requisitos de usuario.
2. Modelo conceptual, que se emplea para el modelado del dominio (contenido del sitio).
3. Modelo de usuario, que modela la navegación según los requerimientos de usuario. Incluye modelos estáticos y dinámicos.
4. Modelo de estructura de presentación, que establece el flujo de presentaciones.
5. Modelo abstracto de interfaces de usuario y modelo del ciclo de vida de los objetos.
6. Modelo de adaptación, que es cuando se implementa el sistema web y se adecua a los requerimientos del usuario.

Para finalizar, es posible afirmar que UWE es una metodología de desarrollo de aplicaciones web orientada hacia sistemas adaptativos (sistematización y personalización).

Actividades para el lector

Cree una tabla resumen con los principales métodos de desarrollo web. En una columna escriba el nombre del método y en otra su descripción. Trate que dicha descripción sea de acuerdo a su propia comprensión

PATRONES DE DISEÑO WEB

Identificar patrones es un proceso que requiere una buena observación. Con ello es posible encontrar semejanzas y abstracciones en lo que se observa. También permite hallar lo verdaderamente importante, desechando aquellos detalles que resulten irrelevantes.

Para definir un patrón, Paloma Díaz cita a Rhiele y Zullighoven: "un patrón es la abstracción de una forma concreta que se mantiene en contextos específicos y no arbitrarios" (Díaz, 2005).

Muchas implementaciones de patrones de diseño se basan en plantillas que representan la estructura de los mismos. Con ello se facilita la indexación según sus propiedades, la semántica o las relaciones que se establecen. En la ingeniería de software está consensuado que las plantillas de patrones deberían tener al menos las secciones que se presentan en la figura 7.6.

Podría afirmarse que los patrones de diseño constituyen la búsqueda de soluciones a problemas comunes en el desarrollo de software. Generalmente, estos problemas comunes se refieren a interacciones e interfaces que muchas veces tienen un comportamiento único en varios escenarios, por lo que se determina que son genéricos a una funcionalidad. Por tanto, diseñar soluciones que se consideren patrones de diseño debe haber cumplido con una efectividad en implementaciones anteriores. Esto lo convierte en un artefacto de software reutilizable.

¿Por qué son importantes los patrones de diseño? Los desarrolladores diariamente resuelven problemas comunes mediante la creación de algún tipo de aplicación de software. Para cada uno de estos problemas es probable que se piense en forma diferente la manera de resolverlos, inclusive repitiendo una solución con la cual ya se ha tenido éxito en el pasado en situaciones similares, pero que quizá no se recuerde. Si se ordena y documenta esa solución es probable que no se tenga que repetir el proceso de pensar y armar dicha solución, sino simplemente reutilizarla. Los patrones de diseño persiguen ese objetivo: tratar los problemas de diseño que se repiten en situaciones similares, con el fin de reutilizar la solución exitosa que se haya tenido con ellas.

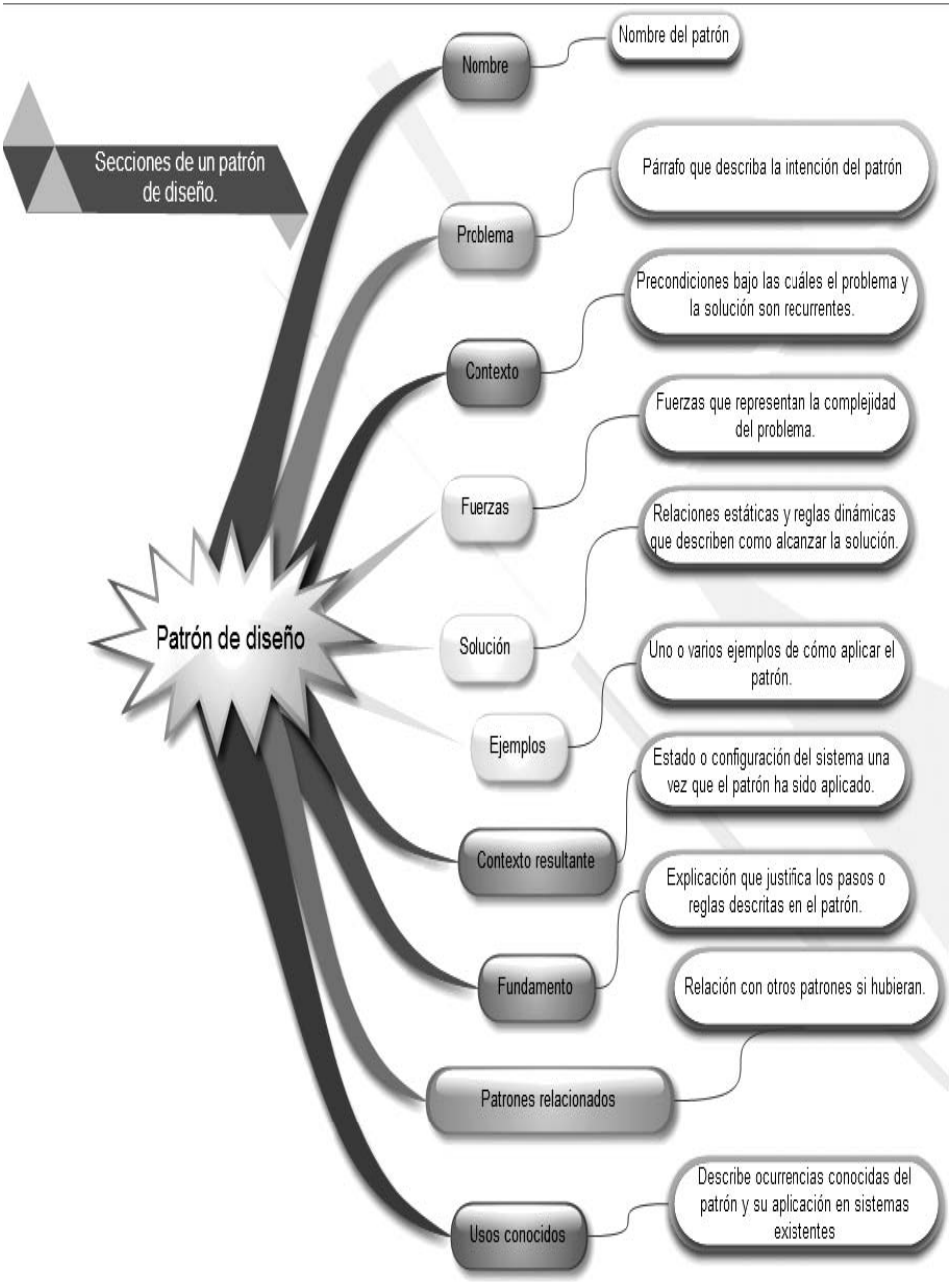


Figura 7.6 Secciones que debería considerar una plantilla de un patrón de diseño. Adaptado de Díaz (2005).

PATRONES DE DISEÑO EN APLICACIONES WEB CON JAVA J2EE

En el año 1994, Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides (el grupo GoF o pandilla de los cuatro) escribieron el libro *Design Patterns: Elements or Reusable Object Oriented Software,* en el cual recopilaron y documentaron 23 patrones de diseño que generalmente los diseñadores de software aplicaban en un entorno orientado a objetos. No es que ellos hayan sido los creadores de la idea, sino más bien que fueron los que ayudaron a difundir la idea entre los constructores de software.

Este grupo (GoF) clasificó los patrones en tres categorías según el propósito para lo cual pudieran utilizarse:

- **Creacionales:** orientados a la creación de instancias de objetos, abstrayendo el proceso de instanciación y ocultando los detalles de cómo se crean e inicializan.
- **Estructurales:** describen la forma en que las clases y objetos se pueden combinar en grandes estructuras proponiendo nuevas funcionalidades.
- **Comportamiento:** permiten definir cómo comunican e interactúan los objetos de un sistema entre sí y evitar o reducir el acoplamiento que se pueda producir entre ellos.

En cuanto un segundo nivel de categorización clasifican los patrones en dos ámbitos: clases y objetos. En esta categoría establecen seis tipos de patrones:

- **Creacional de la clase:** en este patrón se considera la herencia como mecanismo para lograr la instanciación de clases.
- **Creacional del objeto:** la escalabilidad y el dinamismo se considera en este tipo de patrón.
- **Estructural de la clase:** usan la herencia para generar interfaces útiles combinando la funcionalidad de varias clases. Ante la inexistencia de la herencia múltiple en Java, por ejemplo, es un patrón de uso común el empleo de interfaces.
- **Estructural de objeto:** considera la creación de objetos complejos mediante la conjunción de objetos individuales y así crear estructuras complejas. Esta composición puede ser manipulada durante el tiempo de ejecución del software que la utiliza.
- **Comportamiento de clase:** usan la herencia para generalizar el comportamiento entre clases. Cada una de las clases tiene un comportamiento y un patrón definido puede ayudar a distribuir entre un grupo de clases la funcionalidad de cada una de ellas.
- **Comportamiento de objeto:** permiten analizar los patrones de comunicación entre objetos que se relacionan, por ejemplo los objetos que constituyen un objeto complejo.

J2EE (Java 2 Entreprise Environment) posee un catálogo de patrones desde que constituye una arquitectura por sí misma. Incluye Servlets, JavaServer Pages, EnterPrise JavaBeans, entre otros. Estos patrones se reflejan en la figura 7.7.

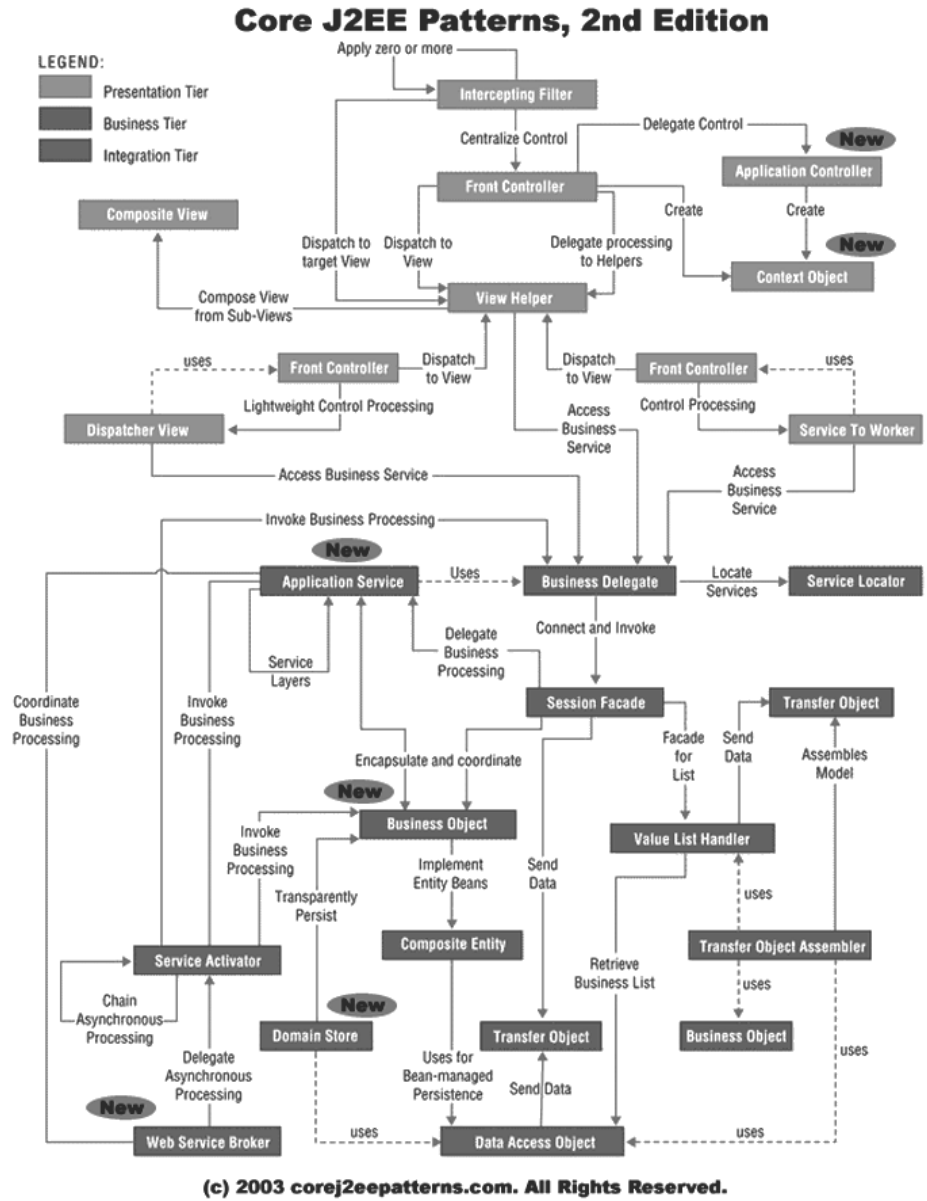


Figura 7.7 Patrones de diseño para J2EE. Extraído el 28 de agosto, 2011 desde http://java.sun.com/blueprints/corej2eepatterns/Patterns/index.html

Si ingresa a http://java.sun.com/blueprints/corej2eepatterns/Patterns/index.html podrá hacer clic en cualquiera de los patrones citados en el diagrama y ver sus detalles. Es posible también relacionar la categorización de GoF (la pandilla de los cuatro): clase Adaptador como un patrón estructural de clase, Facade, Bridge o Composite, como patrón estructural de objetos, Interpretes como patrón de comportamiento de clase o *interator*, *observer* y *visitor* como comportamientos de objeto.

En la figura 7.7 puede observarse que el catálogo de patrones de J2EE se divide en tres capas: presentación, negocios e integración.

En el sitio también se ofrecen ejemplos de implementación de los patrones de J2EE, tal como se muestra en la figura 7.8.

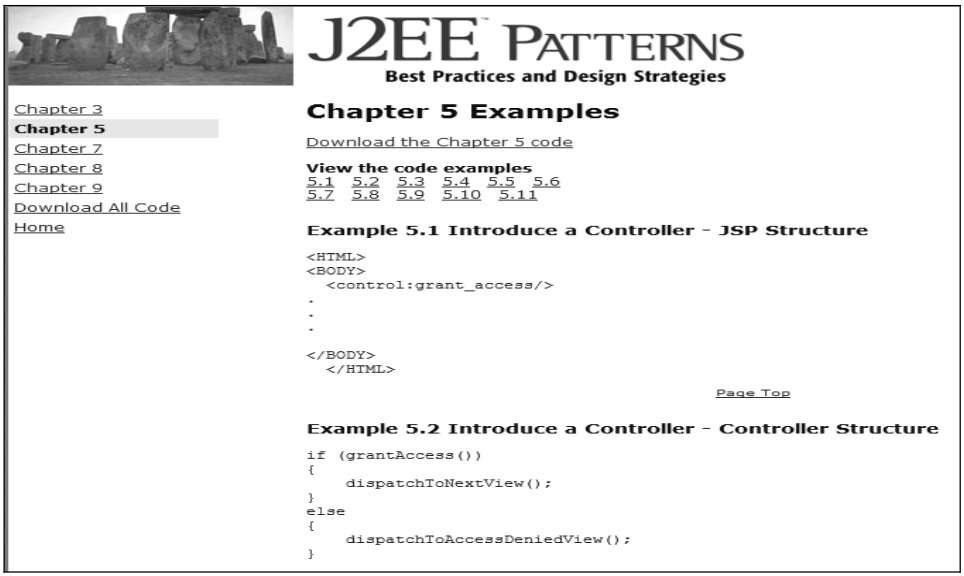


Figura 7.8 Cómo implementar patrones de diseño para J2EE. Obtenido el 28 de septiembre, 2011 de http://authors.phptr.com/corej2eepatterns/codeChap5.html

De acuerdo con el sitio de Sun/J2EE y el libro *J2EE PATTERNS Best Practices and Design Strategies,* existen cinco capas bien definidas en la arquitectura J2EE:

- Cliente
- Presentación
- Negocios
- Integración
- Recurso

La tabla 7.1 muestra los patrones que se relacionan con la capa de presentación.

Tabla 7.1 Patrones relacionados con la capa presentación.

PATRÓN	CONTEXTO	SITIO DE REFERENCIA
Intercepting Filter	Procesa las peticiones y respuestas que se reciben a nivel de presentación. Objeto que media entre el cliente y los componentes web.	http://java.sun.com/blueprints/corej2eepatterns/Patterns/InterceptingFilter.html
Composite view	Patrón utilizado para la presentación de información en páginas JSP. Pueden existir páginas web complejas con muchas subvistas. Ejemplo: página JSP que incluye otras páginas JSP y HTML. Páginas web complejas conteniendo datos de diversas fuentes, que utiliza múltiples vistas en una sola página.	http://java.sun.com/blueprints/corej2eepatterns/Patterns/CompositeView.html
Application Controller	Objeto vista para centralizar y modularizar acciones y administrar vistas.	http://www.corej2eepatterns.com/Patterns2ndEd/ApplicationController.htm
View Helper	Presentación de contenido. Encapsula la lógica de acceso a datos beneficiando los componentes de presentación. Por ejemplo, un componente JavaBeans puede ser utilizado como patrón View Helper en una página JSP.	http://java.sun.com/blueprints/corej2eepatterns/Patterns/ViewHelper.html
Context Object	Objeto que busca evitar el uso de información fuera de un contexto relevante.	http://www.corej2eepatterns.com/Patterns2ndEd/ContextObject.htm
Front Controller	Objeto que acepta requerimientos de un cliente y los direcciona a un manejador adecuado. Divide la funcionalidad en dos objetos: Controller y Dispatcher. El primero acepta y valida los datos y el segundo los direcciona hacia el manejador adecuado.	http://java.sun.com/blueprints/corej2eepatterns/Patterns/FrontController.html

En la tabla 7.2 muestra los patrones relacionados con la capa de negocios.

Tabla 7. 2 Patrones relacionados con la capa de negocios (Business Tier).

PATRÓN	CONTEXTO	SITIO DE REFERENCIA
Application Service	Centraliza el comportamiento de los agregados para proporcionar una capa de servicio uniforme para los servicios del nivel de negocio.	http://www.corej2eepatterns.com/Patterns2ndEd/ApplicationService.htm
Business Delegate	Gestiona la búsqueda de componentes distribuidos y se ocupa de las excepciones.	http://java.sun.com/blueprints/corej2eepatterns/Patterns/BusinessDelegate.html
Service Locator	Encapsula los mecanismos de aplicación para buscar los componentes de servicios de negocios.	http://java.sun.com/blueprints/corej2eepatterns/Patterns/ServiceLocator.html
Session Facade	Proporciona acceso a los servicios a los clientes, ocultando la complejidad de las interacciones de servicios empresariales.	http://java.sun.com/blueprints/corej2eepatterns/Patterns/SessionFacade.html
Transfer Object	Proporciona las mejores técnicas y estrategias para el intercambio de datos en todos los niveles (es decir, a través de los límites del sistema) para reducir la sobrecarga de la red.	http://java.sun.com/blueprints/corej2eepatterns/Patterns/TransferObject.html
Business Object	Implementa el modelo de su dominio conceptual utilizando un modelo de objetos.	http://www.corej2eepatterns.com/Patterns2ndEd/BusinessObject.htm
Value List Handler	Utiliza el patrón iterador GoF para proporcionar la ejecución de consultas y servicios de procesamiento.	http://java.sun.com/blueprints/corej2eepatterns/Patterns/ValueListHandler.html
Compositive Entity	Implementa un objeto de negocio con los beans y la entidad local.	http://java.sun.com/blueprints/corej2eepatterns/Patterns/CompositeEntity.html
Transfer Object Assembler	Construye un compuesto objeto de transferencia a partir de diversas fuentes. Estas fuentes podrían ser componentes EJB, de acceso a datos de objetos u otros objetos arbitrarios.	http://java.sun.com/blueprints/corej2eepatterns/Patterns/TransferObjectAssembler.html
Business Object	Implementa el modelo de su dominio conceptual utilizando un modelo de objetos.	http://www.corej2eepatterns.com/Patterns2ndEd/BusinessObject.htm
Transfer Object	Modelo que proporciona las mejores técnicas y estrategias para el intercambio de datos en todos los niveles (es decir, a través de los límites del sistema) para reducir la sobrecarga de la red.	http://java.sun.com/blueprints/corej2eepatterns/Patterns/TransferObject.html

La tabla 7.3 muestra los patrones relacionados con la capa de integración.

Tabla 7. 3 Patrones relacionados con la capa de integración (Integration Tier).

PATRÓN	CONTEXTO	SITIO DE REFERENCIA
Service Activator	Permite el procesamiento asincrónico en aplicaciones empresariales utilizando JMS. Un Service Activator puede invocar servicios de aplicaciones, Session Facade o Business Objects. También puede utilizar varios Service Activator‘s para proporcionar un procesamiento asíncrono paralelo por mucho tiempo las tareas en ejecución.	http://java.sun.com/blueprints/corej2eepatterns/Patterns/ServiceActivator.html
Domain Store	Proporciona un potente mecanismo para implementar la persistencia transparente para el modelo de objetos	http://www.corej2eepatterns.com/Patterns2ndEd/DomainStore.htm
WebService Broker	Expone uno o más servicios en su aplicación a los clientes externos como un servicio web a través de protocolos Web XML y de estándar	http://www.corej2eepatterns.com/Patterns2ndEd/WebServiceBroker.htm
Data Access Object	Encapsula toda la lógica de acceso a los datos para crear, recuperar, eliminar y actualizar los datos de un almacén persistente	http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html

Suponga la aplicabilidad de uno de estos patrones. Seleccione *Data Access Object* y la propuesta de uso del mismo. Inicie con la definición del Problema, las Fuerzas y la Solución (véase la tabla 7.4).

Tabla 7. 4 Utilizando el patrón Data Acess Object.

UTILIZACIÓN DE OBJETOS DE ACCESO A DATOS.
Problema
Desea encapsular *el acceso y manipulación de datos en una capa separada.*
Fuerzas
• Desea implementar mecanismos de acceso de datos para acceder y manipular datos en un almacenamiento persistente.
• Quiere desvincular la aplicación de almacenamiento persistente del resto de su aplicación.
• Desea proporcionar una API de datos uniforme de acceso con un mecanismo de persistencia de los distintos tipos de fuentes de datos, tales como RDBMS, LDAP, OODP, repositorios XML, archivos planos, y así sucesivamente.
• ¿Desea organizar la lógica de acceso a datos y encapsular características propietarias para facilitar el mantenimiento y la portabilidad?
Solución
Utilizar un objeto de acceso a datos para abstraer y encapsular todos los accesos al almacén persistente. El acceso a objetos de datos servirá para gestionar la conexión con la fuente de datos y así obtener y almacenar datos.

Adaptado de http://www.corej2eepatterns.com/Patterns2ndEd/DataAccessObject.htm

A continuación se crea el diagrama de clases para afrontar el problema. Se muestra en la figura 7.9.

Diagrama de clases

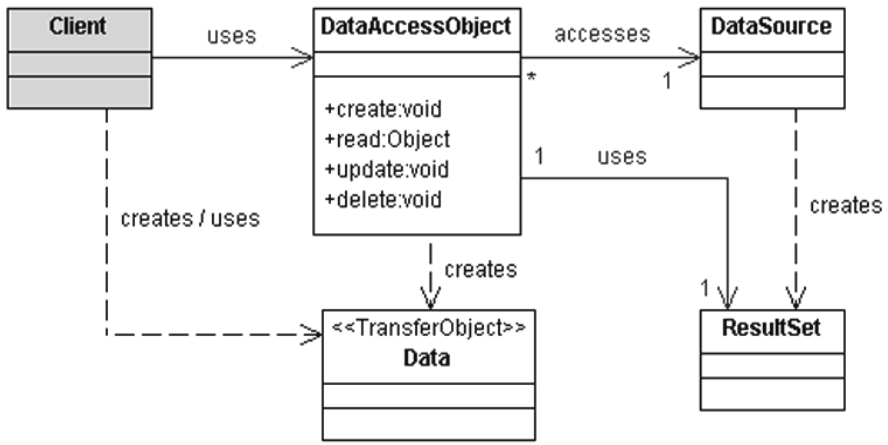


Figura 7.9 Creación del diagrama de clases. Extraído el 8 de septiembre, 2011 de http://www.corej2eepatterns.com/Patterns2ndEd/DataAccessObject.htm.

Una vez definido el diagrama de clases se procede a la creación de un diagrama de secuencia, tal y como se muestra en la figura 7.10.

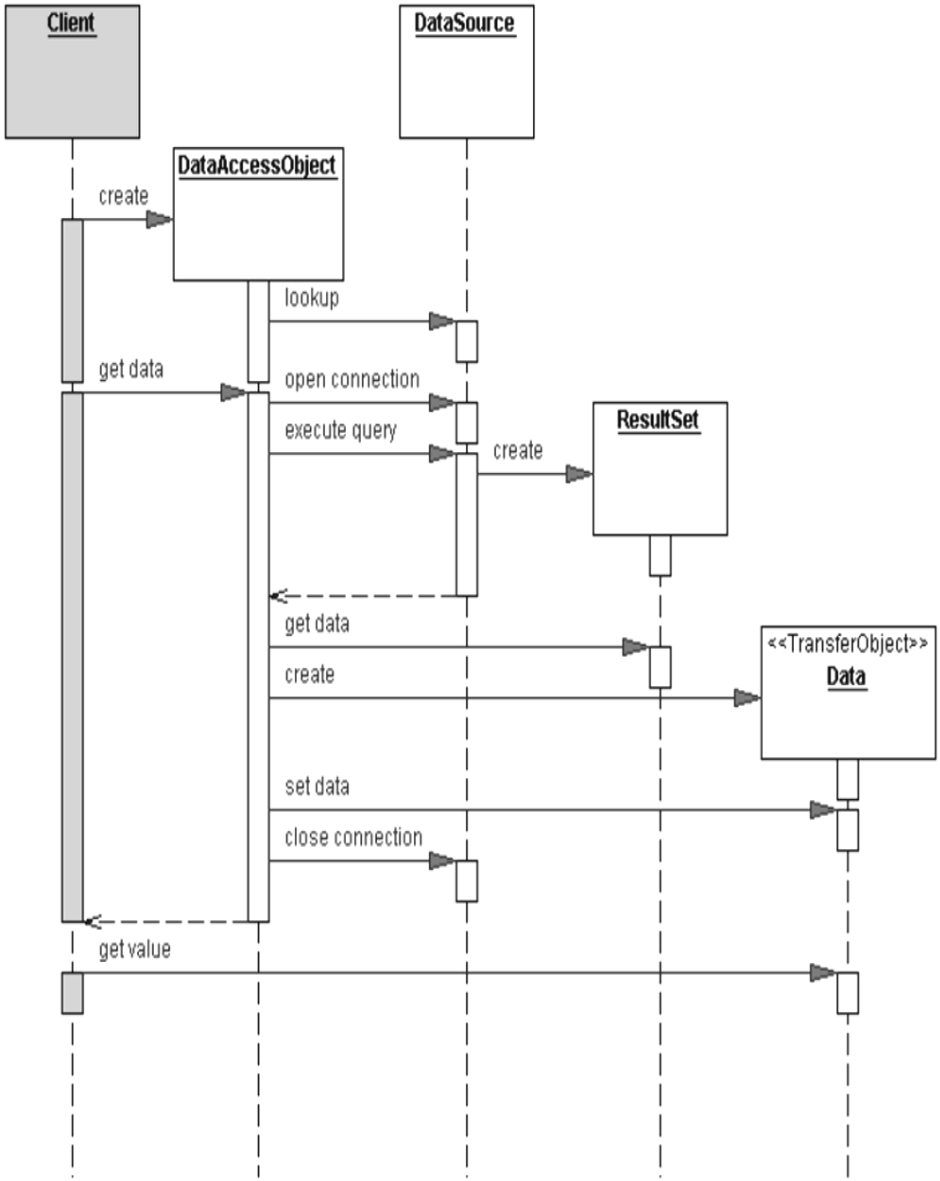


Figura 7.10 Creación del diagrama de secuencia. Obtenído el 8 de septiembre, 2011 de http://www.corej2eepatterns.com/Patterns2ndEd/DataAccessObject.htm.

Se establecen las estrategias y consecuencias para el uso del patrón. La tabla 7.5 muestra la definición de estas actividades.

Tabla 7.5 Estrategias y consecuencias para el uso del patrón.

ESTRATEGIAS
• Objetos personalizados de acceso a datos.
• Objetos de acceso a datos, tipo Factory.
• Colección de objetos de transferencia de datos.
• RowSet en caché.
• Sólo lectura (estrategia RowSet).
• Contenedor de filas (Wrapper RowSet).

Adaptado de http://www.corej2eepatterns.com/Patterns2ndEd/DataAccessObject.htm

CONSECUENCIAS
• Centraliza el control con los controladores debidamente acoplados.
• Permite la transparencia.
• Proporciona vistas con orientación a objetos y encapsula los esquemas de bases de datos.
• Facilita la migración.
• Reduce la complejidad del código en los clientes.
• Organiza todo el código de acceso a los datos en una capa separada.
• Añade capa extra.
• Necesidades de diseño de jerarquía de clases (Estrategia RowSet, Wrapper)
• Introduce una complejidad que permite diseño orientado a objetos (Estrategia RowSet, Wrapper).

Finalmente, se exponen los patrones que se relacionan con el recién desarrollado. La tabla 7.6 muestra esta actividad.

Tabla 7.6 Patrones que se relacionan. }

PATRÓN	DESCRIPCIÓN
Objetos de transferencia	Los objetos de acceso a datos, en su forma más básica, usan los objetos de transferencia para transportar datos entre los clientes. Los objetos de transferencia de datos son empleados en otras estrategias de objetos de acceso a datos.
Factory Method (GoF) y Abstract Factory (GoF)	Objetos de acceso a datos de estos métodos utilizan el Factory Method (GoF) para implementar artefactos concretos de accesibilidad. Abstract Factory se utiliza para describir la estrategia de accesibilidad.
Broker	El patrón DAO se relaciona con este patrón Broker porque describe los enfoques de servidores y clientes en un sistema distribuido. El patrón DAO más concretamente aplica este patrón a disociar el nivel de recursos de clientes en otro nivel, tales como el negocio o la capa de presentación.
Ensamblador de objeto de transferencia	Un objeto mediante este patrón utiliza DAO para obtener datos para construir el objeto que será transferido al cliente.
Controlador de lista de valores	Una lista de valores que se necesita para mostrar al cliente, generalmente se obtiene mediante un objeto DAO.

Adaptado de http://www.corej2eepatterns.com/Patterns2ndEd/DataAccessObject.htm

El producto final de aplicar este patrón, según http://www.corej2eepatterns.com, se encuentra en http://www.codefutures.com/

MODELO VISTA CONTROLADOR (MVC)

Es conocido que el patrón MVC (Modelo Vista Controlador) es el más común en el desarrollo web. Con ello se posibilita la separación de la lógica de control (el QUÉ HACER), la lógica de negocios (CÓMO HACERLO) y la lógica de presentación (interacción con el usuario o interfaces)

Sonia Jaramillo afirma acerca del soporte de Java a este modelo: “Java brinda soporte a esta arquitectura a través de Observer y Observable. La primera clase permite que el objeto sea informado cuando cambie el estado de otro objeto. Mientras que el segundo, es objeto fuente de interés” (Jaramillo, 2008). Por otra parte, Fatos Xhafa sostiene que: “El lenguaje Java da soporte a la combinación MVC

+Observador a través de dos clases: Observer y Observable. … Este modelo es muy útil para el desarrollo de aplicaciones distribuidas en java" (Xhafa, 2008).

Benjamín Aumaille señala que: "La arquitectura MVC propuesta por SUN es la única solución de desarrollo Web en el lado servidor existente en la actualidad que permite separar la lógica de la parte de presentación de una aplicación Web. Esto es muy importante, porque permite que el Desarrollador y el Diseñador Web trabajen por separado en un proyecto, cada uno sobre sus archivos o componentes" (Aumaille, 2002). Con NetBeans, por ejemplo, puede crear una aplicación web tipo Spring Web MVC para implementar el modelo.

Modelo

Representa la información sobre la cual el sistema opera. La lógica de datos permite asegurar que los mismos mantengan su integridad a través de reglas definidas previamente: por ejemplo, realizando acciones, controlando sucesos, generando resultados, entre otras. Éstas se denominan las reglas del negocio. En sí, el modelo es responsable de:

a) Proveer acceso a la capa de almacenamiento de datos, preferiblemente siendo independiente del sistema de administración del mismo.

b) Definir las reglas de negocios que actuarán en el sistema. Por ejemplo: "si la existencia de inventario es igual al stock mínimo entonces generar una orden de pedido"

c) Conservar el registro de las vistas y los controladores del sistema.

d) Controlar los cambios de vistas ante cambios que se presenten frente a sucesos externos, por ejemplo una actualización de los datos en un sistema de facturación.

Según Aumaille, "El modelo está representado por los EJBs y/o los JavaBeans" (Aumaille, 2002).

En resumen, es la representación de los datos y las reglas de negocios. Maneja un registro de las vistas y de los controladores que existen en el sistema.

Controlador

Es el responsable de la respuesta ante los eventos que se presenten en el sistema, generalmente provocados por el usuario. Implica que se produzcan cambios en el modelo y en la visa del usuario (interfaz). Es responsable de:

a) Recibir los eventos de entrada (clic en un botón de comando, change en un combobox, keypress en un campo de texto, entre otros) Handler o Callback.

b) Contener y procesar reglas de gestión de eventos de la forma: "si evento=A entonces acción=B" Puede representar solicitudes al modelo (*generar_pedido(articulo))* o a las vistas (*actualizar:JTablePedidos*).

El controlador se encarga de otorgar significado a las órdenes que da el usuario, actuando sobre los datos que están representados en el modelo. Entra en acción cuando se produce algún cambio en el modelo o alteración en las vistas. En otras palabras, responde a los requerimientos del usuario (eventos) que significan, a su vez, cambios en las vistas y en el modelo del sistema, gestionando adecuadamente las entradas de dicho usuario.

Según Aumaille, "El controlador está representado por los Servlets" (Aumaille, 2002).

Vista

Es la representación visual mediante interfaces gráficas generalmente, aunque puede ser en forma consola, una interface con otro sistema, entre otros. Es responsable de:

a) Recibir los datos del modelo y representarlos visualmente al usuario. Por ejemplo, representar los datos en un JTable producto de la ejecución de una regla de negocio (modelo) y ante un clic a un botón de consulta (controlador).

b) Asociarse a un controlador.

c) Ejecutar acciones que el modelo realiza a través de un controlador.

La vista es un objeto que maneja la representación visual de los datos que administra el modelo. Se encarga de mostrar los datos al usuario, interactuando con el modelo a través de la referencia que establece con éste. Según Aumaille, "La vista está representada por los JSPs" (Aumaille, 2002).

Gráficamente el modelo podría representarse según la figura 7.11.

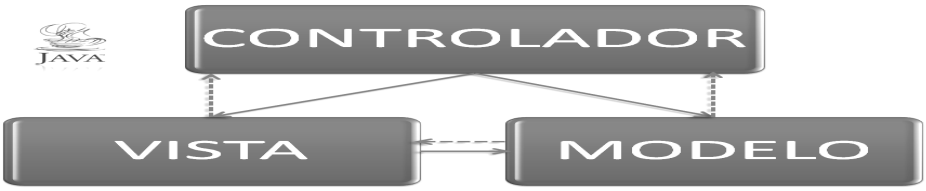


Figura 7.11 Modelo Vista Controlador (MVC).

Las relaciones directas se muestran con una línea firme en las flechas, mientras que las indirectas están representadas con líneas punteadas.

Una implementación de la figura anterior según Aumaille, en un lenguaje como Java, se muestra en la figura 7.12.

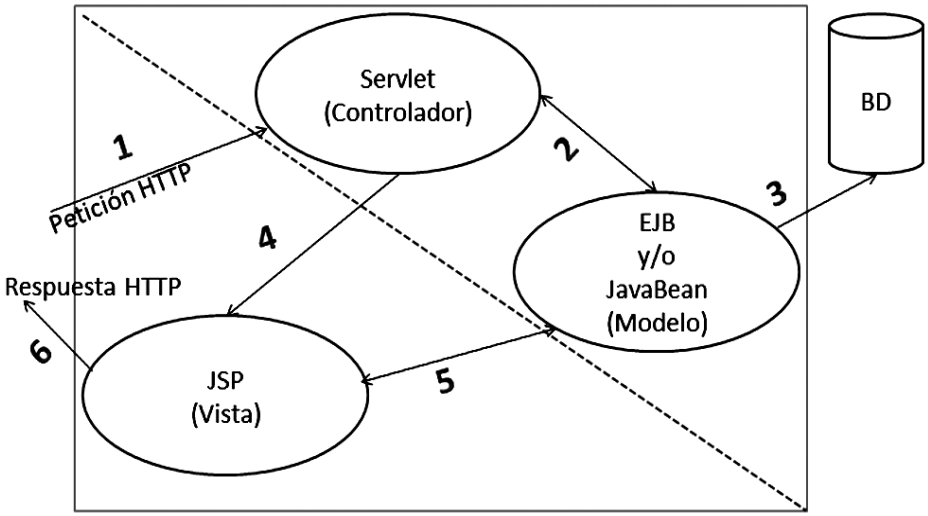


Figura 7.12 Principio de funcionamiento de la arquitectura MVC. Aumaille (2002).

Según Aumaille, el funcionamiento de MVC en Java, de acuerdo con la figura 7.12, se realiza de la siguiente manera:

1. El cliente envía una petición HTTP que tiene como destino un Servlet.
2. El Servlet recupera los datos enviados en la petición HTTP y delega el proceso de los datos a componentes EJB y/o JavaBean.
3. En función del proceso que debe realizarse los componentes EJB y/o JavaBean pueden acceder a fuentes de datos.

   Se vuelve al paso 2, donde una vez concluidos los procesos, los componentes devuelven un resultado al Servlet. Éste almacena el resultado en un entorno concreto (sesión, petición…).
4. El Servlet envía el conjunto del proceso de la petición hacia JSP.
5. El JSP recupera los datos almacenados por el Servlet en uno de los contextos generando la respuesta HTTP.
6. La respuesta HTTP es enviada hacia el cliente.

Para aclarar un poco más el procedimiento anterior véase la figura 7.13 con un diagrama de secuencia.

Cabe aquí brindar la definición de un Servlet. "Un Servlet es una clase que permite realizar la gestión de flujo HTTP en entrada/salida. Un Servlet debe devolver respuesta a una pregunta enviada por un cliente, una vez que haya realizado el proceso que corresponda" (Aumaille, 2002).

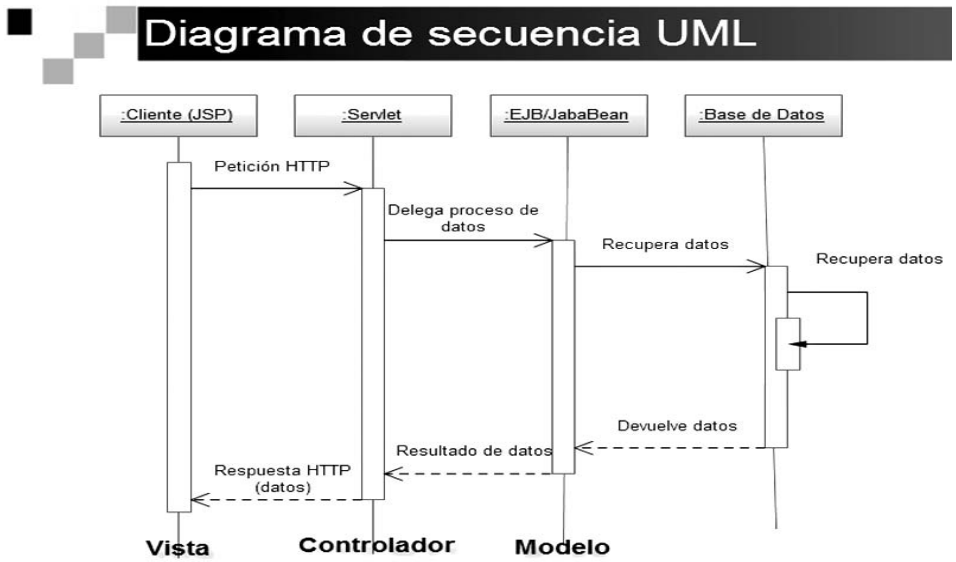


Figura 7.13 Principio de funcionamiento de la arquitectura MVC. Adaptado de Aumaille (2002).

Por tanto, un Servlet funciona como un controlador que recupera información del cliente y la envía posteriormente a un JavaBean o un EJB quien, a su vez, realiza algún tipo de procesamiento sobre los datos. El Servlet recibe de nuevo los datos por parte del JavaBean o del EJB y lo devuelve al cliente.

Otro concepto implícito en este modelo son los JSP. Aumaille los define así: "Un JSP es una página HTML que contiene fragmentos de código Java y/o marcadores JSP específicos, lo que permite realizar procesos e integrar datos con el fin de generar datos de forma dinámica. Una página JSP es, pues, un modelo de generación de código dinámico" (Aumaille, 2002).

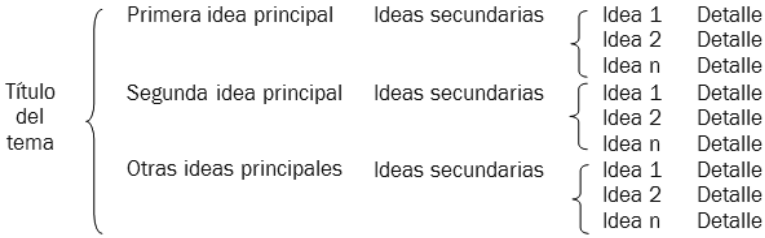
Es importante saber que las páginas JSP permiten separar la capa de presentación de un aplicación web de la capa lógica, integrando los datos mediante marcadores JSP específicos. MVC contribuye así a implementar esta separación.

Finalmente, los EJB se definen de la siguiente manera: "Un EJB es un componente no visual que encapsula una parte lógica y que se utiliza en la creación de aplicaciones distribuidas. Los EJB se encuentran en el interior de contenedores EJB ejecutados en un EJS (Enterprise Java Server) [...] Un EJB es un componente JavaBeans con algunas extensiones" (Aumaille, 2002). Entre las funcionalidades de un EJB se encuentran la gestión de transacciones, seguridad de acceso, servicio de nombres, mensajería asincrónica, entre otras.

Actividades para el lector

a) Elabore un mapa conceptual que describa las partes de MVC: Modelo, Controlador y Vista.

b) Repase nuevamente el tema de patrones de diseño desarrollado en este capítulo, subrayando la definición de cada concepto, y elabore un esquema que sintetice las ideas teóricas de lo que subrayó en el texto. Puede utilizar el formato que más le sea útil. El esquema puede ser de la siguiente manera:



Para mayor detalle revise www.psicopedadogia.com/tecnias-de-estudio/esquema

RESUMEN

Este capítulo presenta una introducción a los patrones de diseño utilizados en la ingeniería de requerimientos, principalmente orientados a la web. Es una preocupación constante el desarrollo de aplicaciones web que se lleven a cabo rápidamente, con calidad y que cumplan con los requerimientos del usuario. Por ende, se trataron los temas siguientes:

1. Métodos de desarrollo web, necesarios para que el estudiante conozca los principales métodos existentes para el desarrollo de aplicaciones web. Entre estas metodologías considere: WSDM, webML y UWE.
2. Conceptualizar y comprender la teoría de patrones de diseño y su aplicabilidad mediante Java J2EE.
3. El modelo MVC como patrón de diseño y su utilización en Java.

EVIDENCIA

- [ ] Creó una tabla resumen con los principales métodos de desarrollo web
- [ ] Elaboró un esquema donde sintetizó las ideas teóricas de lo que subrayó en el texto.
- [ ] Elaboró un mapa conceptual que describe las partes del Modelo, Controlador y Vista.

Autoevaluación

Relación de columnas: en la tercera columna de la derecha (gris) coloque el valor de las descripciones que corresponda a la columna que aparece a la izquierda.

1.	Su principal característica es que se centra en el usuario más que en los datos, haciendo que las aplicaciones web sean más personalizadas.	Categorías de patrones de diseño creados por el grupo GOF.
2.	Se encarga de la apariencia que tendrán las páginas independientemente de la herramienta que se utilizará para diseñarlas.	Fases de desarrollo del modelo WSDM.
3.	Creacionales, estructurales y comportamiento.	Característica principal de WSDM.
4.	Modelado de usuario, diseño conceptual, diseño de la implementación e implementación.	Implementa el modelo de su dominio conceptual utilizando un modelo de objetos.
5.	Abstracción de una forma concreta que se mantiene en contextos específicos y no arbitrarios.	Permite el procesamiento asincrónico en aplicaciones empresariales utilizando JMS.
6	Composite View.	Se utiliza para la presentación de la información en páginas JSP.
7.	Business Object	Servlet.
8.	Service Integrator	Controlador en MVC
9.	Es la representación de los datos y las reglas de negocios.	Función de presentación del modelo webML.
10.	Clase que permite realizar la gestión de flujo HTTP de entrada/salida.	Patrón de diseño.
11.	Encargado de la respuesta ante los eventos que se presenten en el sistema, generalmente provocados por el usuario.	Vistas en MVC.
12.	Representación visual mediante interfaces graficas de los datos del usuario.	Modelo en MVC.

REFERENCIAS

Bibliografía

Díaz, María Paloma (2005) Ingeniería de la web y patrones de diseño. Editorial Pearson, Madrid, España.

Jaramillo Valbuena, Sonia (2008) Programación Avanzada en Java. Editorial Elizcom, Armenia, Quindio, Colombia.

Fatos Xhafa (2008) Aplicaciones distribuidas en Java con tecnología RMI. Publicaciones Delta, Madrid, España.

Aumaille, Benjamín (2002) J2EE Desarrollo de aplicaciones Web. Ediciones ENI, Barcelona, España.

Stefano Ceri y otros (2003) Designing Data-Intensive Web Applications. Morgan Kaufmann Publishers, San Francisco, California.

Gustavo Rossi y otros (2008) Web Engineering. Modelling and Implementing Web Applications" Springer-Verlag London Publication, Inglaterra.

Thomas Baar (2004) <<UML>> 2004 – The Unified Modeling Language. Modeling Languages and Applicattions. Library of Congress, Nueva York.

Páginas Web recomendadas

1. Juan Belón (2009) *Patrones de diseño: Elementos reutilizables para la web*. Obtenido el 15 de noviembre del 2011 desde http://www.programadorphp.org/blog/patrones-de-diseno-elementos-reutilizables-para-la-web/
2. webML.org (2011) *The Web Modeling Language.* Obtenido el 15 de noviembre del 2011 desde http://www.webml.org/webml/page1.do
3. Brambilla, Marco (2011) *WebML: the Web Modeling Language – tutorial with audio and slides.* Obtenido el 15 de noviembre del 2011 desde http://dbgroup.como.polimi.it/brambilla/webml
4. Ciberaula (2011) *Patrones de Diseño en aplicaciones Web con Java J2EE.* Obtenido el 18 de noviembre del 2011 desde http://java.ciberaula.com/articulo/diseno_patrones_j2ee/
5. corej2eepatterns.com (2011) *Core J2EE Patterns. Best Practices and Design Strategies.* Obtenido el 18 de noviembre del 2011 desde http://www.corej2eepatterns.com/Patterns2ndEd/index.htm

Respuestas a la autoevaluación

1.	Su principal característica es que se centra en el usuario más que en los datos, haciendo que las aplicaciones web sean más personalizadas.	3.	Categorías de patrones de diseño creados por el grupo GOF.
2.	Se encarga de la apariencia que tendrán las páginas independientemente de la herramienta que se utilizará para diseñarlas.	4.	Fases de desarrollo del modelo WSDM.
3.	Creacionales, estructurales y comportamiento.	1.	Característica principal de WSDM.
4.	Modelado de usuario, diseño conceptual, diseño de la implementación e implementación.	7.	Implementa el modelo de su dominio conceptual utilizando un modelo de objetos.
5.	Abstracción de una forma concreta que se mantiene en contextos específicos y no arbitrarios.	8.	Permite el procesamiento asincrónico en aplicaciones empresariales utilizando JMS.
6	Composite View.	6.	Se utiliza para la presentación de la información en páginas JSP.
7.	Business Object	10.	Servlet.
8.	Service Integrator	11.	Controlador en MVC
9.	Es la representación de los datos y las reglas de negocios.	3.	Función de presentación del modelo webML.
10.	Clase que permite realizar la gestión de flujo HTTP de entrada/salida.	5.	Patrón de diseño.
11.	Encargado de la respuesta ante los eventos que se presenten en el sistema, generalmente provocados por el usuario.	12.	Vistas en MVC.
12.	Representación visual mediante interfaces graficas de los datos del usuario.	9.	Modelo en MVC.

JavaServer Pages en NetBeans 7.1

8

Reflexione y responda las siguientes preguntas

¿Las aplicaciones modernas para web deberían basar su procesamiento en el servidor o en el cliente?

¿Porqué muchos desarrolladores web utilizan herramientas que no poseen IDE para creación gráfica de sus páginas?

¿Estaremos migrando totalmente del desarrollo desktop al desarrollo web?

Contenido

EXPECTATIVA

Es necesario conocer cómo se establecen los patrones de diseño y toda la infraestructura que ampara su filosofía. Como desarrolladores de software es importante no solamente conocer cómo se implementa mediante una herramienta de desarrollo, sino también toda la filosofía que se encuentra detrás de esta implementación.

Los patrones de diseño constituyen abstracciones que permiten la solución de problemas en muchos niveles en el desarrollo de software. Pueden estar implícitos en diferentes etapas del desarrollo de un sistema computacional: desde el análisis, diseño, arquitectura y hasta la propia implementación. Es importante considerar que existen muchas funciones que son muy repetitivas en un contexto de negocios, con muchas características similares y que bien podrían implementarse en un solo proceso: un patrón. Esta práctica permite la reutilización de software y el mejoramiento de la calidad al utilizar componentes de software ampliamente probados.

Después de estudiar este capítulo, el lector será capaz de

- Establecer comentarios en páginas JSP.
- Crear expresiones en páginas JSP.
- Declarar variables en páginas JSP.
- Crear Scriplet en páginas JSP.
- Crear Directivas @page en páginas JSP.
- Desarrollar Servlets en páginas JSP.
- Utilización de CSS en páginas JSP.

INTRODUCCIÓN

Mediante HTML (HypertText Markup Language o lenguaje de Marcado de HiperTexto) se puede enviar información desde un servidor web hasta un cliente. Sin embargo, esta información es estática y no es posible suministrarla de forma dinámica, es decir, información que es cambiante al pasar el tiempo, como por ejemplo, la que se almacena en una base de datos.

Evidentemente, para construir una página web dinámica con HTML se debe hacer uso de algo más: un lenguaje de script que permita procesar la información y de algún modo mantenerla actualizada en el cliente. Ese script puede estar escrito en ASP, JSP o PHP.

JSP (Java Server Page) posee un funcionamiento básico que trabaja de la siguiente forma:

- Una página JSP es enviada al cliente en lugar de una HTML.
- Si no hay problemas se establece la comunicación con el servidor con información adicional del cliente.
- Mediante JSP se construye la página HTML estática que se envía al cliente y se rompe la comunicación con éste.
- Se visualiza la página HTML en el cliente.

Los Servlets constituyen aplicaciones Java que crean contenido HTML mediante sentencias *out.print (...)*, lo que las convierte en algo tedioso. Por ello, Sun Microsystems desarrolló JSP como alternativa a generar código del lado del servidor. El código generado queda embebido dentro del HTML tal y como se hace también en PHP o ASP. Con JSP se separa el código Java del HTML y equivalen a Servlets que se ejecutan del lado del servidor.

Una página JSP es un archivo que se construye con marcas y etiquetas HTML. Dentro de la sección *body* se pueden intercalar etiquetas especiales o instrucciones Java entre los símbolos <% y %> Un Servlet, por su parte, es un programa desarrollado en Java que posee cierta estructura y un funcionamiento especial. Es una contraposición al termino *Applet* pues éste se ejecuta en el navegador de un cliente, mientras que el Servlet lo hace en el servidor de la aplicación web. En resumen, todo programa JSP en el servidor se convierte en un Servlet para producir una página HTML que envía al cliente.

COMENTARIOS EN JSP

Los comentarios en JSP se pueden definir de acuerdo a los siguientes escenarios:

a) Los comentarios que se construyen como etiquetas HTML en un archivo JSP y que recibe el cliente.

   En este caso, el comentario se construye de la siguiente manera:

   <!- Comentario HTML pasado al cliente ->

b) Los comentarios que se escriben en Java y que se encuentran entre los identificadores <% y %>

```
<%// Comentario en Java de una sola línea %>
```

O

```
<% /* Comentario en Java de varias
      líneas.... */%>
```

Observe la figura 8.1 sobre cómo establecer comentarios en una página JSP:

```
<%--
    Document   : index
    Created on : 19/10/2011, 08:56:51 AM
    Author     : egomez
--%>

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <%// Comentario en Java de una sola línea %>
        <% /* Comentario en Java de varias
            líneas .... */%>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Hello World!</h1>
    </body>
</html>
```

Fig. 8.1 Comentarios en una página JSP.

EXPRESIONES EN JSP

Según Elisa Viso: "*Una expresión Java es cualquier enunciado que regresa un valor. … Se Puede clasificar a las expresiones de acuerdo al tipo del valor que regresen. Si regresan un valor numérico entonces se tiene una expresión aritmética; si regresan falso o verdadero tenemos una expresión booleana; si regresan una cadena de caracteres tenemos una expresión tipo String.*" (Viso, 2007)

Mediante expresiones se pueden insertar valores que se obtienen de Java y enviarlos al cliente. Para sus efectos tiene dos sintaxis:

```
<%= expresión Java %>
```

Observe que existe un signo de igual (=) antes que la expresión Java que se quiere utilizar. Antes de eso se encierra entre par de signos de porcentaje (%) y los terminadores de menor qué y mayor que.

Otra sintaxis es la siguiente:

```
<jsp:expression>expresión></jsp:expression>
```

En ambos casos corresponde a una expresión JSP. Una aplicación simple de esto en el siguiente ejemplo de una página JSP.

1. Ingrese a NetBeans y pulse *Archivo* y seleccione *Nuevo proyecto*.
2. Seleccione un proyecto de tipo *Java Web* y *Web application*. Pulse el botón *Siguiente*.
3. Ponga un nombre alusivo al proyecto. En mi caso le puse *WebApplication15*, por ser el nombre que estaba por defecto en el lenguaje. Pulse el botón *Siguiente* y en la pantalla que sigue el botón *Terminar.*
4. En la edición de la página JSP que se crea por defecto observe el código. Si puede solo sustituya la parte de código que sea diferente al mostrado en la página por defecto, o sino, elimine todo el código de esta página (*index.jsp*) y sustitúyalo por el siguiente código:

EJERCICIO: CREACIÓN DE EXPRESIONES EN JSP
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<%// Comentario en Java de una sola línea %>

EJERCICIO: CREACIÓN DE EXPRESIONES EN JSP
<% /* Comentario en Java de varias líneas.... */%>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JSP Page</title>
</head>
<body>
<h1>Probando expresiones
<h2>
La fecha y hora actual es: <%= new java.util.Date()%>
Fecha y Hora local: <%= new java.util.Date().toLocaleString() %>
La potencia de 16 es <%= Math.pow(16, 2)%>
La raíz cuadrada de 16 es <%= Math.sqrt(16)%>
</h2>
</h1>
</body>
</html>

Como podrá Observar hay algunas líneas de este código que es necesario explicar:

1. *<h1>Probando expresiones*: En esta línea se le indica al compilador que construya el título *Probando expresione*s en un header <h1> Los headers en HTML van desde <h1> hasta <h6> pudiendo jerarquizar el texto como encabezados.
2. *<right>:* que el texto a continuación se alinee a la derecha (todo lo que está entre <right> y </right>
3. *
La potencia de 16 es <%= Math.pow(16, 2)%>
: Se despliega la expresión para la potencia de un valor (<%= Math.pow(16, 2)%>)* como una lista sin ordenar, la cual se delimitada por <li> y

La ejecución del código anterior se muestra en la figura 8.2

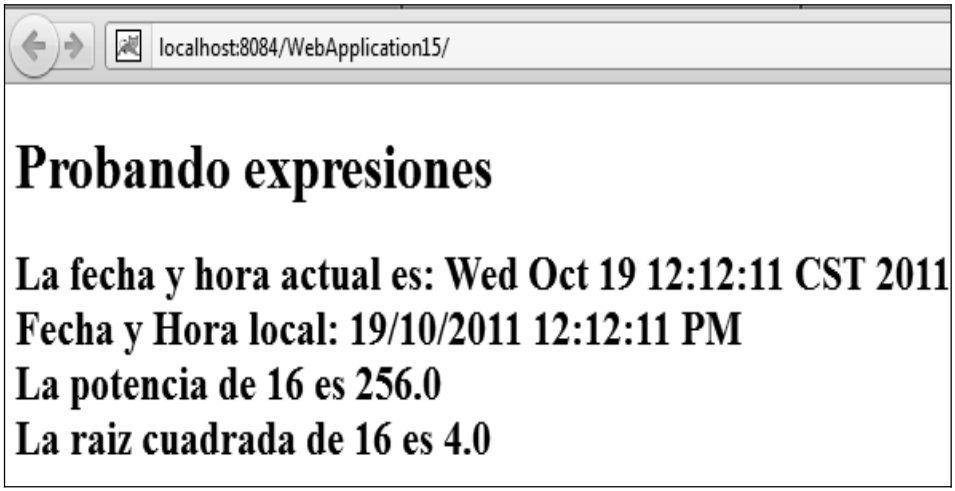


Fig. 8.2 Expresiones en una página JSP.

DECLARACIÓN DE VARIABLES EN JSP

Ya se vio como una expresión en JSP se delimita entre los caracteres <%= y %>. Esta expresión se evalúa cuando un cliente solicita la JSP que contiene una expresión delimitada entre esos caracteres.

Una variable es, según Perez Menor, es "*Una variable es una representación alfanumérica de una posición de memoria. Como tal, se caracteriza por tres propiedades: posición de memoria que almacena el valor, tipo de datos almacenado y nombre que se refiere a esa posición de memoria.*" (Perez, 2003)

Para declarar variables en JSP se realiza de la siguiente manera:

```
<%! declaración de variable o método; %>.
```

En esta declaración el signo de admiración (!) que acompaña al de porcentaje (%) es la que delimita la declaración de una variable o método. Al declararse de esta forma lo que hace es globalizarlas, es decir, tendrán vida dentro de todo el código de la página JSP y por tanto utilizada en cualquier lugar de ella. Se debe recordar que el servidor mediante JSP convierte cualquier página de este tipo en Servlet, el cual es utilizado tantas veces como sea solicitado y las variables deben mantener su valor ante estas llamadas sucesivas.

Otra sintaxis que se puede utilizar es el siguiente:

```
<jsp:declaration> declaración de variable o método; ></jsp:declaration>
```

Vamos a modificar el código que se ha venido trabajando (*WebApplication15)* Marca todo el código de *index.jsp*, bórralo y sustitúyelo por el siguiente:

EJERCICIO: DECLARACIONES EN UNA PÁGINA JSP

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
  <%// Declaraciones en JSP %>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JSP / NetBeans</title>
   </head>
   <body>
   <h1> Probando declaraciones en JSP</h1>
   <h2>
     <%! double x=4;%>
     <%! double y=2;%>
 <%! public double Potencia(double x,double y)
  {return Math.pow(x, y);};%>
<%! public double Raiz(double x)
  { return Math.sqrt(x);};
  %>
<li>Potencia de <%=x%> elevado a <%=y%> es : <%=Potencia(x,y)%></li>
<li>Raíz Cuadrada del número <%=x%> es:  <%=Raiz(x)%></li>
</h2>
</body>
</html>
```

La ejecución de esta aplicación mostraría algo similar a la figura 8.3

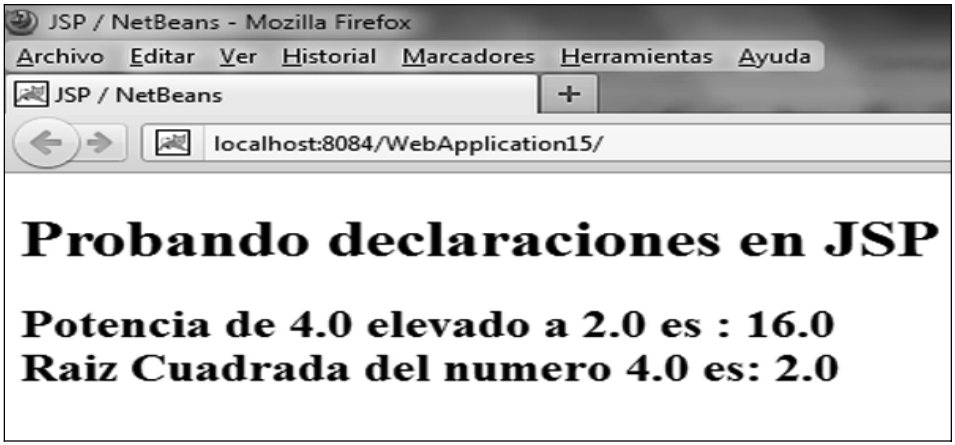


Fig. 8.3 Declaraciones en una página JSP.

Ahora un poco de explicación del código:

1. <%! double x=4;%> Aquí lo que se declara es una variable (x) de tipo double. Observe que es una expresión declarativa.
2. El código que se cita debajo de esta explicación muestra el método o procedimiento para obtener la potencia de un número elevado a otro. Observe que inicia con <%! Y continúa con la declaración normal de un método en Java. Al final cierra el método con %>.

```
<%! public double Potencia(double x,double y)
                    {return Math.pow(x, y);};
 %>
```

3. Otra línea importante es la que se cita debajo de esta explicación. Con ello se indica que se muestre en pantalla el valor de x (Potencia de <%=x%>) y el valor de y (elevado a <%=y%> es :). Asimismo, se invoca al procedimiento indicado en el paso 2 para generar el cálculo (<%=Potencia(x,y)%>) de la potencia, enviándole los dos parámetros que necesita (x, y)

```
<li>Potencia de <%=x%> elevado a <%=y%> es : <%=Potencia(x, y)%></li>
```

La función Raiz trabaja de la misma forma como se explicó en esta parte.

Podría agregarse un par de líneas más al código:

```
<%! private int numVeces=0;%>
He accesado esta página <%=++numVeces%>
```

Estas líneas demuestran como la variable *numVeces* se actualiza a medida que actualizamos la página JSP. ++numVeces es un contador que cada vez que la

página es actualizada aumenta su valor. Así, actualizando la página en el navegador podrá Observarcomo el contador también se actualiza.

Actividades para el lector

Modifique la página JSP anterior incluyendo las operaciones matemáticas de suma, resta, división y multiplicación.

SCRIPTLET EN JSP

Un Scriptlet es un bloque de código que se delimita por los caracteres <% y %>. Está conformado por instrucciones Java que un contenedor en JSP coloca en *_jspService* al momento de compilar. No se pueden colocar comentarios dentro de un scriptlet.

Existen muchos detractores de los Scriptlet, aduciendo que son muy desordenados o muy difíciles de leer. Por ejemplo, en el sitio web http://www.programacion.com/articulo/convenciones_de_programacion_java_para_paginas_jsp_193#scriptles, se afirma lo siguiente:

"*Siempre que sea posible, debemos evitar usar scriptlets JSP mientras que las* librerías *de etiquetas proporcionen una funcionalidad similar. Esto hace que las páginas JSP sean más fáciles de leer y de mantener, ayuda a separar la lógica de negocios de la lógica de presentación, y hará que nuestras páginas evoluciones más fácilmente al estilo JSP 2.0. (JSP 2,0 soporta, pero no favorece el uso de scriptlets)*"

Básicamente, se refiere a utilizar en lugar de Scriptlet alguna librería común, por ejemplos JSTL que es una librería de etiquetas estándar. Si se adopta el modelo MVC se debe reducir el acoplamiento entre la capa de presentación y la de lógica de negocios, en dicho caso, tampoco se deberían usar Scriptlets JSP para escribir esta última.

Por su parte, en el sitio web http://java.ciberaula.com/articulo/introduccion_jstl opinan así de los Scriptlets:

- *El código Java embebido en scriptlets es desordenado.*
- *Un programador que no conoce Java no puede modificar el código Java embebido, anulando uno de los mayores beneficios de los JSP: permitir a los diseñadores y personas que escriben la lógica de presentación que actualicen el contenido de la página.*
- *El código de Java dentro de scriptlets JSP no pueden ser reutilizados por otros JSP, por lo tanto la lógica común termina siendo re-implementado en múltiples páginas.*

- *La recuperación de objetos fuera del HTTP Request y Session es complicada. Es necesario hacer el Casting de objetos y esto ocasiona que se tenga que importar más Clases en los JSP.*

Abogan por el uso de otra librería estándar: JSTL (JSP Standard Tag Library) que en realidad es un conjunto de librerías de Java 2 que permiten el uso de etiquetas simples y estándares para la escritura de páginas JSP. Este conjunto de librerías está conformada por *core* que posee las funciones de script básicas (condicionantes y funciones de entrada/salida), *xml* que contiene funciones para el procesamiento de archivos xml, *fmt* que incluye la internacionalización y el formato de monedas y fechas por ejemplo, *sql* que incluye la funcionalidad para acceso a base de datos.

Un programa sencillo utilizando Scriptlets en una página JSP se muestra a continuación:

EJERCICIO: USAR SCRIPLETS EN UNA PÁGINA JSP
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<%// Declaraciones en JSP %>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JSP / NetBeans</title>
</head>
<body>
Probando Scriptlet en JSP
<h2>
<%
String colores[]={"orange","red","blue","green","black"};
int j = 5;
int vector[]= new int[5];
for (int i=0;i<j;i++){
vector[i]=i;
%>

EJERCICIO: USAR SCRIPLETS EN UNA PÁGINA JSP
<font color=<%=colores[i]%>size="<%=5%>">
<%=vector[i]%>

<%}%>
</h2>
</body>
</html>

Observe el código del Scriptlets. Es lo que se coloca entre <% y %>, es decir la declaración del vector de colores y un vector de enteros y su respectiva procesamiento. La salida típica de la ejecución de este programa es la que se muestra en la figura 8.4:

Fig. 8.4 Prueba de un Scriplet en una página JSP.

Por lo que se puede deducir, existen razones de peso para evitar el uso de Scriptlets en páginas JSP y optar por mecanismos como los librerías JSLT.

DIRECTIVAS @PAGE EN JSP

La directiva page se utiliza para establecer las propiedades generales de una página JSP, por una única vez en toda la página. Es decir, los valores de la directiva se aplicarán a toda la página JSP.

La sintaxis de esta directiva y sus atributos se muestran en la tabla 8.1.

Tabla 8.1 Directiva page y sus atributos.

ATRIBUTO	DESCRIPCIÓN
language=”java”	Define el lenguaje de script usado en los scriplet, declaraciones y extensiones de un archivo JSP.
Extends=”package.class”	Especifica la cualificación de una superclase que se utilizará en el archivo JSP. Sun recomienda usar este atributo con cautela dado que puede limitar la compilación de un archivo JSP de calidad.
import= “{ package.class | package.* }, …”	Mediante este atributo es posible importar una o más paquetes a un archivo JSP para que los use. Las clases de los paquetes pueden ser utilizados por los scriplets, expresiones, declaraciones y etiquetas.
session=“true|false”	Para usar una página JSP el cliente debe establecer una sesión HTTP. Si ésta es false no se pueden usar sesiones en una página JSP. Por defecto es true.
buffer=”none|8kb|sizekb”	Permite determinar el tamaño del buffer (en kb) que se utilizara para el objeto out (que maneja la salida) que se envía a la pagina JSP. Por defecto este atributo es de 8 kb.
autoFlush=“true|false”	Especifica si se envía la salida o no cuando el buffer está lleno. Por defecto está en true por tanto el buffer será descargado. Si es false se notificará una excepción cuando se sobrecargue dicho buffer.
isThreadSafe=“true|false”	Determina si se encuentra implementada la seguridad de los threads (hilos) Por defecto está en true, por lo que el motor JSP puede remitir varias solicitudes concurrentes en la página.
info=“text”	Mediante este atributo se especifica una cadena de texto que se incorpora a la página JSP compilada. Posteriormente, se puede recuperar el String por medio del método *getServletInfo()*
errorPage=“URLrelativa”	Especifica la ruta hacia un archivo JSP donde se envían las excepciones. Si inicia con “/” la ruta es relativa al directorio de documentos JSP y resuelto por el servidor web. Sino dicha ruta es relativa al archivo JSP actual.
isErrorPage=”true|false”	Especifica se el archivo JSP muestra una página de error. Si está en true se puede usar el objeto exception que referencia a la excepción lanzada, en el archivo JSP. Si es false se usa en el archivo JSP.

Ejemplo de aplicación 1

A continuación se creará un ejemplo básico de aplicación de directivas. La aplicación utilizará el atributo *import*. Crearemos para ella un paquete.

1. Proceda a crear un nuevo proyecto *Web Application*. Nombre este proyecto como *webDirectivas.*
2. Modifique el archivo JSP para que sea similar a la que se muestra en la figura 8.5.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page import="prueba.ejemploDirectivas"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Página JSP prueba de directivas </title>
    </head>
    <body>
        <h1>Probando Directivas!</h1>
        <%
          ejemploDirectivas ejemplo = new ejemploDirectivas();
          out.print(ejemplo.mostrar());
        %>
    </body>
</html>
```

Fig. 8.5 Código de implementación de una directiva en una página JSP.

Si Observa el código anterior se puede dar cuenta que existen las instrucciones *prueba.ejemploDirectivas* como llamado a un paquete Java.

Asimismo, *ejemploDirectivas ejemplo = new ejemploDirectivas()* como instanciación de la clase *ejemploDirectivas* que debería esta implementado en el paquete *prueba.ejemploDirectivas.* Claramente se ven las instrucciones subrayadas en el código anterior que indican que hay algo mal en el mismo.

3. Proceda a crear el paquete, que es el motivo por el cual se indica el error en el código anterior. Haga clic derecho con el botón derecho del mouse sobre el nombre del proyecto y en el menú contextual que aparece pulsamos *Nuevo* y luego *Paquete Java*. Nombre el paquete como *Prueba*.

4. Una vez creado el paquete procedamos a crear la clase *ejemploDirectivas* y la función *Saludar*. El código sencillo es el siguiente:

CLASE EJEMPLODIRECTIVAS Y LA FUNCIÓN MOSTRAR
public class ejemploDirectivas {
public String mostrar(){
return "Hola como estas?";
}}

5. Ahora podremos comprobar que los errores en nuestra página JSP desaparecieron. Ejecutar esta página será similar a lo mostrado en la figura 8.6.

Fig. 8.6 Código de implementación de la directiva @page import en una página JSP.

Ejemplo de aplicación 2

A continuación un segundo ejemplo para demostrar el uso de directivas en JSP. Para ello se creará otro proyecto web donde se generá una tabla, la cual se cargará en una hoja Excel. Proceda con este ejercicio.

1. Proceda a crear un nuevo proyecto *Web Application*. Nombre este proyecto como *webDirectivasMultiplicación.*
2. Sustituya el código que se crea por defecto para la página JSP con lo que se muestra en la figura 8.7.

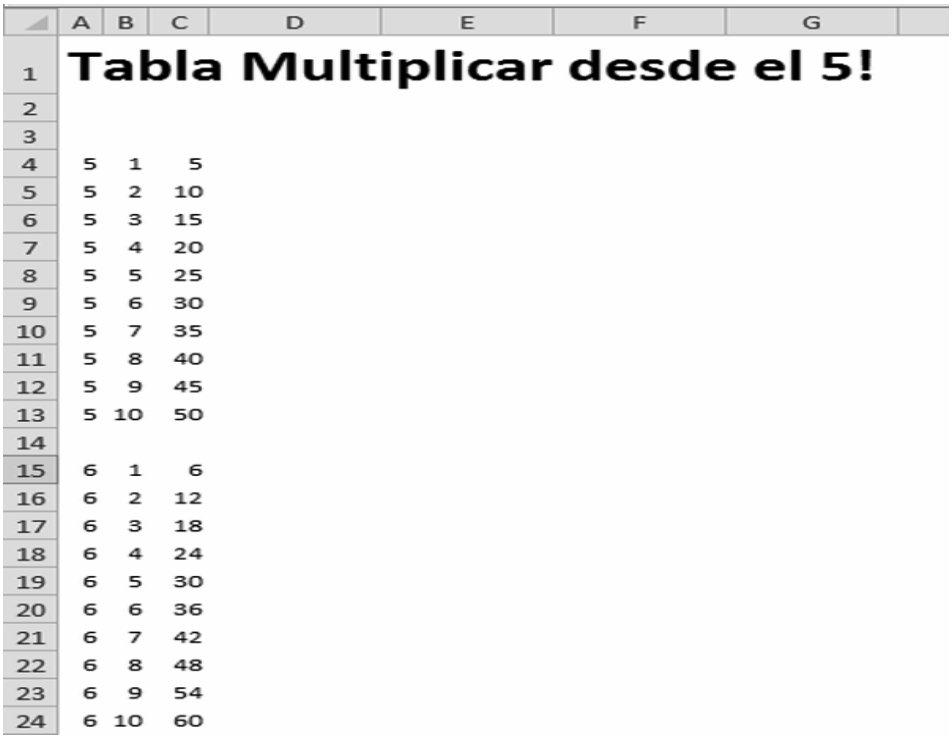
```
<%@page contentType="application/vnd.ms-excel"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Tabla Multiplicar desde el 5!</h1>
        <%--tabla multiplicar--%>
        <table>
        <% for(int i=5;i<=10;i++){%>
        <tr></tr>
        <%for (int j=1;j<=10;j++){%>
            <tr>
              <td><%=i%></td>
              <td><%=j%></td>
              <td><%=i*j%></td>
             <%}%>
            </tr>
         <%}%>

        </table>
    </body>
</html>
```

Fig. 8.7 Código de implementación de una directiva de archivo en una página JSP.

Observe en el código de la figura 8.7 que la directiva a utilizar es @page contentType=”Application/vnd.ms-excel” con lo cual se habilita la generación de un archivo Excel según la tabla creada. En HTML con <tr><tr/> se permite crear una fila en la tabla y con <td><td/> una celda

3. La salida de la ejecución de este código es similar a lo mostrado en la figura 8.8. Es un archivo Excel, pero previamente en la ejecución de la página se solicitó al usuario abrir o guardar dicho archivo. La figura 8.8 es el archivo Excel abierto.



	A	B	C	D	E	F	G
1	Tabla Multiplicar desde el 5!						
2							
3							
4	5	1	5				
5	5	2	10				
6	5	3	15				
7	5	4	20				
8	5	5	25				
9	5	6	30				
10	5	7	35				
11	5	8	40				
12	5	9	45				
13	5	10	50				
14							
15	6	1	6				
16	6	2	12				
17	6	3	18				
18	6	4	24				
19	6	5	30				
20	6	6	36				
21	6	7	42				
22	6	8	48				
23	6	9	54				
24	6	10	60				

Fig. 8.8 Código de implementación de una directiva de generación de un archivo Microsoft® Excel desde una página JSP.

SERVLETS EN JSP

Un Servlets es un componente que se escribe en Java. Como tal, se compone de métodos o funciones. Según José Francisco Ceballos un Servlet es:

"*Un Servlet es un programa que se ejecuta en el contenedor Web de un servidor de aplicaciones. Los clientes pueden* invocarlo *utilizando el protocolo HTTP, comparativamente, lo mismo que un applet es cargado y ejecutado por un explorador, un Servlet es cargado y ejecutado por un contenedor web"* (Ceballos, 2008)

En la figura 8.9 se muestra la funcionalidad de un Servlet. Claramente se puede Observer cómo se generan las peticiones en el explorador del cliente y luego una respuesta en el servidor.

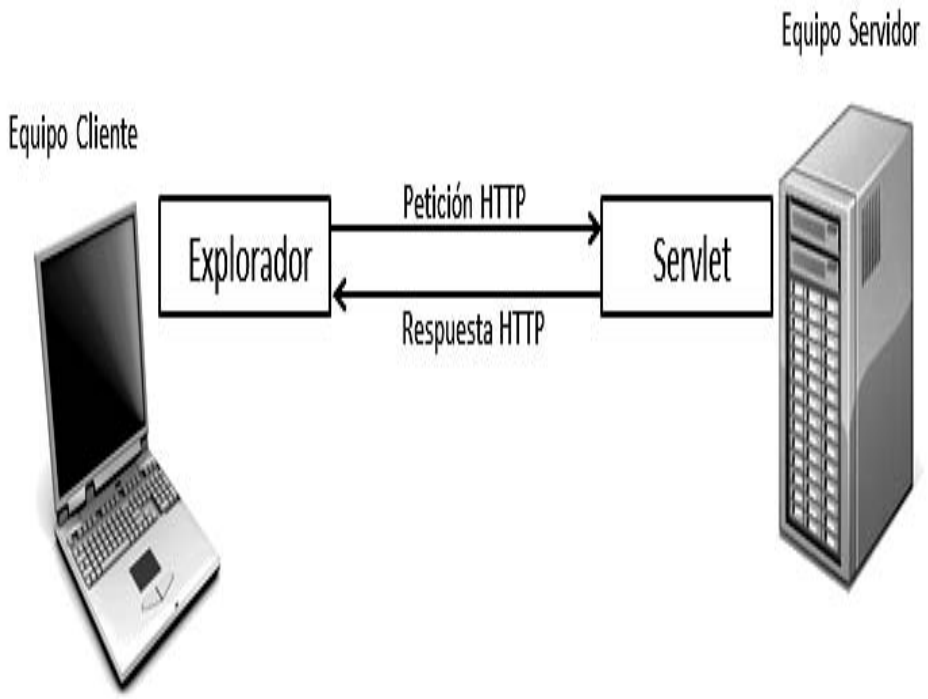


Fig. 8.9 Funcionalidad de un Servlet.

Un Servlet implementa los siguientes métodos.

a) *service:* es un método obligatorio y constituye la medula de un Servlet. Dentro de éste método se incluyen las tareas principales de ejecución del Servlet.
b) *init:* es un método opcional que se ejecuta antes del método *Service*. El objetivo de este método es la inicialización de recursos que serán utilizados por Service, por ejemplo, una conexión a una base de datos.
c) *destroy:* también es un método opcional y se ejecuta una vez que termina *service*. El objetivo de este método es destruir los recursos usados por s*ervice*.

Javier García y otros (1999) definen algunas características de los Servlets, entre las que están:

a) Independencia del servidor que utilizan y el sistema ejecutivo sobre el que se ejecutan.
b) Pueden llamar a otros Servlets e inclusive a sus métodos en concreto. Por ejemplo un Servlet que gestione la conexión a una base de datos y que llame a la función de otro para que valide los datos de un usuario.
c) Pueden obtener fácilmente datos de un cliente, tal como la dirección IP con la que se conecta el cliente, el puerto que usa en sus llamadas, el método utilizado (GET, POST,...)
d) Permite la utilización de cookies y sesiones permitiendo almacenar la información del usuario.
e) Sirven como enlace entre el usuario y una o varias bases de datos en cliente servidor y tres capas.
f) Pueden servir como proxy para un Applet.

Ejemplo de uso de Servlets en NetBeans

El ejercicio tiene como propósito validar la sesión de un usuario mediante un Servlets, con lo que aprovecharemos para conocer cómo se manejan las sesiones en NetBeans. Iniciemos entonces éste ejercicio.

1. Inicia un proyecto web nuevo. Nómbralo *miPrimerServlet*.
2. En la página index.jsp sustituye el código por lo siguiente:

EJEMPLO DEMOSTRATIVO DE UN SERVLET: CÓDIGO DE LA PÁGINA INDEX.JSP
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<%-- inicia la creación de la parte estática de la página (HTML)%--> <html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JSP Page</title>
</head>
<body>
<%--inicio de la creación del formulario web. El action establece el Servlet que será ejecutado (en este caso miServlet) mediante post.--%> <form name="autenticacion" action="miServlet" method="post">

EJEMPLO DEMOSTRATIVO DE UN SERVLET: CÓDIGO DE LA PÁGINA INDEX.JSP
<%-- creación de una tabla dentro del formulario.%--> <table align="center" width="350" style="border-style:blue">
<thead>
<tr>
<th colspan="2">Acceso</th>
</tr>
</thead>
<tbody>
<tr>
<td>
Usuario:
</td>
<%-- crea una caja de texto donde el usuario digitará el nombre--%> <td> <input type="text" name="nombreUsuario" value=""/>
</td></tr>
<tr>
<td>
Clave: </td>
<%-- Crea una caja de texto donde el usuario digitará la contraseña--%> <td><input type="password" name="clave" value=""/></td>
</tr>
<tr>
<%--se crea el botón de comando. <td><input type ="submit" value="Aceptar"/></td>
</tr>
</tbody>
</table>
</form>
</body>
</html>

3. Procede ahora a crear otra página JSP de nombre *paginaPrincipal.jsp*. Puedes copiar la página JSP y pegarla en la misma ubicación de esta y

posteriormente renombrarla. El código que tendrá esta página es como sigue:

EJEMPLO DEMOSTRATIVO DE UN SERVLET. CÓDIGO DE LA PÁGINA PAGINAPRINCIPAL.JSP
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<%
<%--se declara la variable sesionActual de tipo HttpSession--%> HttpSession sesionActual = request.getSession(true);
<%--se carga en sessionActual la variable de sesión *nomUsuario*--%> String usuario = (String) sesionActual.getAttribute("nomUsuario");
if(usuario == null) {
response.sendRedirect("index.jsp");
}
%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<%--Se muestra en el titulo de la pagina el nombre del usuario--%> <title>Bienvenido <%=usuario%> </title>
</head>
<body>
Bienvenido: <%=usuario%>
<%--Se Se crea un menú mendiante una lista de elementos--%>
Mantenimiento   //en el # de *href* ponemos el nombre
Consultas   //de una pagina JSP creada y que
Informes   //contenga la funcionalidad
Acerca de...   //requerida
</body>
</html>

4. Finalmente, vamos a desarrollar el código del Servlet. Para ello pulsa sobre el nombre del proyecto con el botón derecho del mouse y selecciona *Nuevo* y posteriormente selecciona *Servlet*. Nómbralo como *miPrimerServlet.* El código a escribir para este Servlet es el siguiente:

EJEMPLO DEMOSTRATIVO DE UN SERVLET. CÓDIGO DEL SERVLET MIPRIMERSERVLET
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import java.io.IOException;
@WebServlet(name = "miServlet", urlPatterns = {"/miServlet"})
public class miServlet extends HttpServlet {
public void doGet(HttpServletRequest solicitud, HttpServletResponse respuesta) throws ServletException, IOException {

// Obtenemos un objeto Print Writer para enviar respuesta
respuesta.setContentType("text/html");

try {
String nombreUsuario = solicitud.getParameter("nombreUsuario");
String clave = solicitud.getParameter("clave");
if (nombreUsuario.equals("enrique") && clave.equals("gomez"))
{
//se crea una variable de tipo session para guardar el nombre de usuario
HttpSession sesionActual = solicitud.getSession(true);
//se almacena el nombre de usuario en la variable de sesión
sesionActual.setAttribute("nomUsuario", nombreUsuario);
//se direcciona hacia paginaPrincipal.jsp
respuesta.sendRedirect("paginaPrincipal.jsp");
}
else{
//se mantiene en la página actual, refrescándose.
respuesta.sendRedirect("index.jsp");
}

EJEMPLO DEMOSTRATIVO DE UN SERVLET. CÓDIGO DEL SERVLET MIPRIMERSERVLET
}
Finally
{}}

//permite procesar tanto la solicitud del usuario como la respuesta del servidor
public void doPost(HttpServletRequest solicitud, HttpServletResponse respuesta)
throws ServletException, IOException {

doGet(solicitud,respuesta); }}

5. Una vez creados los códigos de las páginas JSP y el Servlet podemos probar nuestro sitio. La primera página en mostrarse es *index.jsp*. La figura 8.10 muestra esta página ejecutándose.

Fig. 8.10 index.jsp en ejecución.

6. Si el usuario y la clave son correctos, se ejecutará la página *paginaPrincipal.jsp*, tal como se muestra en la figura 8.11.

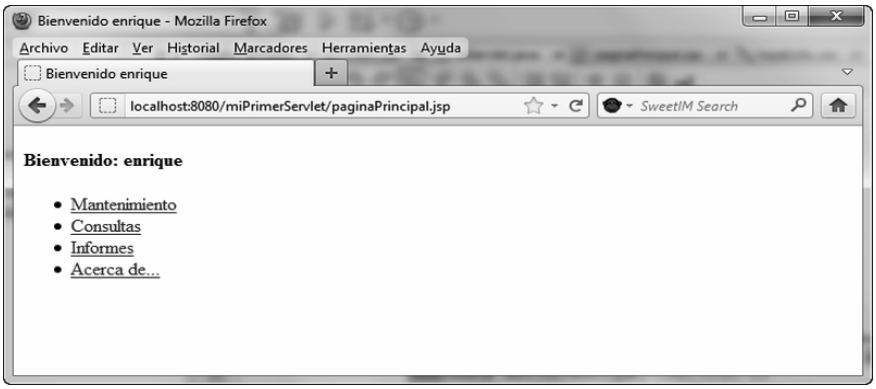


Fig. 8.11 paginaPrincipal.jsp en ejecución.

CREAR UN SITIO WEB SENCILLO CON JSP Y CSS

Vamos a desarrollar un sitio web sencillo con paginas JSP que utilicen un archivo CSS para la ubicación de todas sus secciones. Antes de desarrollar el ejemplo vamos a conocer un poco de CSS.

En la práctica el uso de tablas para la ubicación de objetos y componentes en una pagina web constituye un tema superado, sustituyendo su uso con las denominadas CSS (Cascading Style Sheets) Además de restrictivas las tablas en HTML o en cualquier otro diseñador web no facilita la creatividad del desarrollador. Aporta poca libertad a la hora de colocar los objetos y componentes en una pagina web.

Joseph Aubri señala las ventajas del uso de CSS de la siguiente manera: "…puede separarse el contenido del formato, el cual se centraliza, reduciendo los riesgos de error y, sobre todo, acelerando las actualizaciones: es suficiente con modificar el estilo para que los elementos que utilizan un determinado formato adopten el cambio inmediatamente, haciendo innecesaria la revisión de todas las paginas que constituyen el sitio web" (Aubri, 2009)

Innova (2001) agrega "…las CSS son definiciones del aspecto con que se representa un documento al usuario. Esto permite independizar el contenido de una pagina web de su estructura y de entrada da la posibilidad de alterar su aspecto sin necesidad de modificar la pagina sino tan solo su contenido"

El lenguaje para las CSS se especifican en las normativas CSS1, CSS2 y CSS3 de la W3C o World Wide Web Consortium (http://www.w3.org/Style/CSS/) Este estándar es aceptado por toda la industria que se relaciona con el desarrollo orientado a la Web. Algunas piensan que el único que les da dolor de cabeza es el Internet Explorer. Personalmente no me consta por que habitualmente utilizó otro navegador que por mis pantallazos sabrán determinar cual es.

Hay tres maneras de asociar las reglas de estilos a las marcas, a saber:

1. Directamente en las marcas HTML,
2. En el código de la cabecera (head) de la página o,
3. Agrupando las reglas de estilo en un archivo independiente con extensión *.css.

Definir las reglas de estilo directamente en el HTML

Este método a pesar de ser muy fácil de implementar es el menos recomendado. Se define mediante la propiedad *style* a nivel del código HTML.

Muchos desarrolladores Web definen el estilo directamente en el HTML y luego trasladan dicho estilo a una hoja de estilos. Un ejemplo de cómo definir un estilo en HTML es el siguiente:

REGLAS DE ESTILO (CSS) DIRECTAMENTE EN EL HTML
<html>
<head>
</head>
<body>
<h1 style="color:white;background-color:blue">
Este mensaje tiene letras color blanco sobre fondo color azul.
</h1>
</body>
</html>

Observe que la pareja de valores para el color de texto es color:white. Bien podríamos poner el nombre del color como también en expresión hexadecimal. Por ejemplo color:#ff0000 que es color rojo o color:#ffff00 que es color amarillo. Aplica también para el color de fondo (background-color) Siempre que inicialicemos una propiedad debemos separarla con un punto de coma como se Observe en color y background-color.

Si definimos un estilo en una sección este se aplica solamente a esa sección. Por ejemplo podríamos modificar el código anterior creando dos secciones diferentes. He aquí el código modificado.

REGLAS DE ESTILO (CSS) DIRECTAMENTE EN EL HTML
<html>
<head>
</head>
<body>
<h1 style="color:white;background-color:blue">
Este mensaje tiene letras color blanco sobre fondo color azul.
</h1>
<h1 style="color:blue;background-color:yellow">
<h1 style = "text-align:center">
<h1 style = "font-size:0.14in;">
Este segundo titulo va centrado, con letras azules de 0.14 in y fondo amarillo
</h1>
</body>
</html>

Observe que existen dos secciones h1 que son títulos configurados de forma diferente en cuanto alineación y tamaño de fuente. Observe la figura 18-05 donde se muestra la ejecución del código anterior.

Definir reglas de estilo a nivel de página

A nivel de cabecera de página podríamos hacerlo tal como se muestra en la parte sombreada del siguiente código:

REGLAS DE ESTILO (CSS) A NIVEL DE PÁGINA
<html>
<head>
<title>CSS a nivel de cabecera de página</title>
<style type="text/css">
h1 {color:white;background-color:blue}
h2 {color:blue;background-color:yellow}
h2 {text-align:center}
h2 {font-size:0.28in;}
</style>
</head>
<body>
Este mensaje tiene letras color blanco sobre fondo color azul.
Este segundo titulo va centrado, con letras azules de .14 in y fondo amarillo
</body>
</html>

Observe la parte del código anterior donde se define el estilo que tendrá cada título (h1 y h2) Observe, asimismo que se incluye la instrucción <style type="text/css"> donde se hace alusión al estilo CSS.

Definir reglas de estilo en un archivo CSS aparte

Ahora la versión más utilizada es creado una hoja de estilo en cascada (CSS) por aparte y referenciándolo en la página HTML. Lo primero que debemos hacer es crear nuestro archivo CSS (recuerda que lo puedes hacer en cualquier editor de código HTML o en el propio Notepad de Windows) NetBeans también permite la creación de archivos CSS. Este primer archivo debes guardarlo con extensión *.css. En mi caso lo nombre *hojaEstilo.css*:

REGLAS DE ESTILO (CSS) CREADO EN ARCHIVO APARTE
body {
color:white;background-color:blue;
text-align:center;
font-size:0.28in;
}

Luego de guardarlo se utiliza en un archivo HTML de la siguiente manera:

REGLAS DE ESTILO (CSS) CREADO EN ARCHIVO APARTE
<html>
<head>
<title>Mi primera página con CSS</title>
<link rel="stylesheet" type="text/css" href="hojaEstilo.css" />
</head>
<body>
Utilizando CSS en archivo aparte
</body>
</html>

Observe la línea 4 donde href="hojaEstilo.css" referencia a la hoja de estilo que creamos con antelación. Así, podríamos modificar la hoja de estilo y la página o páginas del sitio Web cambiaría también. Esto permitiría desarrollar sistemas web más dinámicos en cuanto estilo.

A continuación se presenta la tabla 8.2 con algunas propiedades que se pueden utilizar en una CSS.

Tabla 8.2 Propiedades CSS

PROPIEDAD	DESCRIPCIÓN	EJEMPLO
border:	Establece todas las propiedades de borde en una declaración.	*border: 1px solid #d0d0d0;* borde de 1 píxel, sólido (de una línea sólida) y con color grisáceo (#d0d0d0) Más información en: http://www.w3schools.com/cssref/pr_border.asp
border-style:	Especifica qué tipo de borde se utilizará para mostrar.	*border-style:solid;* Más información en: http://www.w3schools.com/css/css_border.asp
top:	Establece el margen superior de un elemento a una unidad por encima / debajo del borde superior de su elemento contenedor.	top:5px; borde de 5 pixeles. Más información en: http://www.w3schools.com/cssref/pr_pos_top.asp
left:	Establece el margen izquierdo de un elemento de una unidad a la izquierda / derecha del borde izquierdo de su elemento contenedor.	left:5px; margen de 5 pixeles Más información en: http://www.w3schools.com/cssref/pr_pos_left.asp
width:	Establece el ancho de un elemento.	width:100px; ancho de 100 pixeles para un objeto. Más información en: http://www.w3schools.com/cssref/pr_dim_width.asp
height:	Establece la altura de un elemento.	height:100px; especifica la altura de un párrafo, por ejemplo. Más información en: http://www.w3schools.com/cssref/pr_dim_height.asp
padding:	Define el espacio entre el borde del elemento y el contenido del elemento. El relleno (padd) borra un área alrededor del contenido (en el interior de las coordenadas) de un elemento. El relleno se ve afectado por el color de fondo del elemento. El relleno superior, derecho, inferior, izquierda y se puede	padding-top:25px; padding-bottom:25px; padding-right:50px; padding-left:50px; rellena los espacios indicados por las coordenadas de top, bottom, right y left. Más información en: http://www.w3schools.com/css/css_padding.asp

PROPIEDAD	DESCRIPCIÓN	EJEMPLO
	cambiar de forma independiente con propiedades distintas.	
margin:	Define el espacio alrededor de los elementos. Se utiliza conjuntamente con las propiedades top, bottom, right o left. También puede utilizarse en forma abreviada, tal como: margin:100px 50px;	margin-top:100px; margin-bottom:100px; margin-right:50px; margin-left:50px; Más información en: http://www.w3schools.com/css/css_margin.asp
background:	Especifica el color de fondo de un elemento.	body {background-color:#b0c4de;} define el color de fondo del cuerpo del documento HTML. Más información en: http://www.w3schools.com/css/css_background.asp
border-rigth: De igual manera trabajan: border-bottom: border-top:	Define el estilo del borde derecho:	border-right:1px solid #ffffff; define el borde derecho de un elemento. Más información en: http://www.w3schools.com/cssref/pr_border-right.asp

NOTA

Podrá encontrar una amplia descripción de propiedades y atributos CSS en la dirección: http://www.w3schools.com/cssref/default.asp

Ahora que hemos definido un poco lo que es CSS y lo que representa vamos a crear una de ellas para utilizar en nuestro sitio JSP.

1. Ingresa a NetBeans y crea un nuevo proyecto web de nombre *webJSPCSS*.
2. Una vez creado el proyecto pulsa con el botón derecho del mouse sobre el nombre del proyecto y selecciona *Nuevo*, *Hoja de Estilos en Cascada (CSS)...* Nombra este archivo como *estilo.css*. El código correspondiente es el siguiente:

HOJA DE ESTILO CSS: ESTILO.CSS
root {
display: block;
}
ul.menu_color2{
list-style:none;
}
ul.menu_color2 li{
display:block;
position:relative;
padding:0px 20px;
background-color: #000000;
font-size: small;
font-family: "Bookman Old Style";
text-align:center;
vertical-align:middle;
color:white;
width:60px; /* Ancho de los li debajo de los ul */
}
ul.menu_color2 ul{
position:absolute;
left:-40px;
top:20px;
display:none;
list-style:none; }
ul.menu_color2 > li > ul{
position:absolute;
left:-40px;
top:20px;/* Ubicación de los ul debajo de los li*/
display:none;
list-style:none;}
ul.menu_color2 li:hover{
background:#CCCD34;
border-right:1px solid #ffffff;
border-bottom:1px solid #ffffff;

HOJA DE ESTILO CSS: ESTILO.CSS
border-top:1px solid #ffffff;
border-left:1px solid #ffffff; }
ul.menu_color2 li:hover > ul{
display:block; }
ul.menu_color2 > li{
display:inline; }
.borde {
border:0; }
.slogan {
border:40px;
position: absolute;
top: 0px;
left: 0px;
width:179px;
height:67px;
padding:0 0 0 0;
margin : 0 0 1em 0;}
.principal {
border: 0px;
position: absolute;
top: 0px;
left: 179px;
width:581px;
height:67px;
padding:0 0 0 0;
margin : 0 0 1em 0; }
.linea {
border:0px;
background-color:#000000;
position: absolute;
top: 67px;
left: 0px;
width:760px;

HOJA DE ESTILO CSS: ESTILO.CSS
height:105px;
padding:0 0 0 0;
margin : 0 0 1em 0;}
/*Área inferior de la página*/
.bottom { border-right:1px solid #000000;
border-bottom:30px solid #000000;
border-top:1px solid #000000;
border-left:1px solid #000000;
position: absolute;
top: 426px;
left: 0px;
width: 760px;
height: 50px;
background-color: #CCCD34;
font-size: small;
font-family: "Bookman Old Style";
text-align:center;
vertical-align:middle;
color:white; }
.sub1 {
border: 0px;
position: absolute;
top: 170px;
left: 0px;
bottom: 319px;
right: 0px;
width:180px;
height:319px;
background-color:#CCCCB3; }
/*Seccion 5 según documento guía, corresponde al área donde podrá escribir y situar información al público*/
.sub2 {

HOJA DE ESTILO CSS: ESTILO.CSS
border: 0px;
position: absolute;
top: 170px;
left: 180px;
width: 300px;
height: 319px;
background-color: #CCCCB3; }
/*Seccion 6 según documento guía, corresponde al área donde se colocan las opciones de menú al estilo botón con imagen*/
.sub3 {
border: 0px;
position: absolute;
top: 170px;
left: 436px;
right: 0px;
width: 324px;
height: 319px;
background-color: #CCCCB3;}

La página JSP donde utilizaremos este archivo CSS se divide en las secciones que se muestran en la tabla siguiente:

<table>
<tr><td>Aquí va una imagen (arribaIzquierda.gif) (179*67 px) (archivo en CSS: slogan)</td><td colspan="2">Aquí va otra imagen (arribaDerecha.gif) (581*67 px) (archivo en CSS: principal)</td></tr>
<tr><td colspan="3">Aquí van las opciones de menú (Actividades, Alimentos, Servicios, Acerca de y Contactos)
(archivos en CSS: linea1 y menú_color2, como div y ul class)</td></tr>
<tr><td>Aquí van dos imagenes (sub1B.gif y dog2.gif) (140*34 y 132*86 px) (en CSS: sub1)</td><td>Aquí van dos imágenes (sub2.gif y dog1.gif) (140*33 y 147*108 px) (en CSS: sub2)</td><td>Aquí van cuatro imágenes (enfermedades.gif, razas.gif, consejos.gif y contactenos.gif) (todas de 135*30 px) (en CSS: sub3)</td></tr>
<tr><td colspan="3">Aquí va el texto: Todos los derechos reservados
(en CSS: bottom)</td></tr>
</table>

Actividades para el lector

Relaciona la tabla anterior con el código CSS (estilo.css) Ubique las secciones que deberán colocarse, asimismo intenta reproducir las imágenes que se consideran en la pagina JSP o en su lugar algunas diferentes. Trate de conservar las medidas que se se señalan en la tabla para cada imagen.

La pagina JSP tendrá una apariencia similar a la que se muestra en la figura 8.12 se relaciona cada sección de la tabla con las de la figura, asi ubicaras cada imagen y su respectiva sección en el archivo CSS.

Fig. 8.12 apariencia que mostrará al ejecutarse index.jsp

Las imágenes que utilice en esta página fueron descargadas de varios sitios en Internet que no interesa ahora (puedes descargar tus propias imágenes) Los botones que utilice en esta página fueron diseñados con el software *Agama Web Buttons* que puedes descargar desde http://www.agamabuttons.com/ Asimismo, las dos imágenes de encabezados los edite con Power Point y SnagIT que puede descargar desde http://www.techsmith.com/ Las formas de como hacer esto te queda de ejercicio dado que no es el objeto de este libro.

Ahora vamos a implementar el código en index.jsp. Sustituye el código que crea esta página por el siguiente:

CÓDIGO FUENTE DE LA PÁGINA INDEX.JSP
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<title>Bienvenido(a) a Servicios y Veterinaria E&D!</title>
<link id='page-skin-1' href='estilo.css' rel='stylesheet' type='text/css'>
<body>
<img src="img/arribaIzquierda.gif" class="dibujoArriba" alt="dibujo">
<img src="img/arribaDerecha.gif" class="borde" alt="dibujoDerecha">
<ul class="menu_color2">
Actividades
Exhibicion
Campeonato
Alimentos
Cachorros
Adultos   //se ejecuta pagCachorros.jsp.
Servicios
Cortes
Pedicure

CÓDIGO FUENTE DE LA PÁGINA INDEX.JSP
Limpieza
Acerca de
Mision
Vision
Objetivos
Contacto
<img src="img/sub1B.gif" alt="imagenB">
<img src="img/dog2.gif" alt="perro2">
Buscar información sobre mascotas
<img src="img/sub2B.gif" alt="dibujo sub2">
<img src="img/dog1.gif" alt="perro 1">
Gracias por visitar nuestro sitio...!
 <img src="img/enfermedades.gif" alt="">
<img src="img/sub3B.gif" alt="">
<img src="img/razas.gif" alt="">
<img src="img/consejos.gif" alt="">
<img src="img/contactos.gif" alt="">

CÓDIGO FUENTE DE LA PÁGINA INDEX.JSP
 Todos los derechos reservados
</body>
</html>

Actividades para el lector

Haga una tabla de 2 columnas. En la primera columna escribe la propiedad o instrucción HTML y en otra la descripción de lo que significa, según lo que hemos visto hasta el momento de HTML y las referencias que se indican al final del capítulo. Recuerda que algunas instrucciones inician con <> y terminan con </>

Comprenda el código fuente que se ha escrito en este ejercicio.

4. La ejecución de esta página web se muestra en la figura 18.13.

Fig. 8.13 index.jsp en ejecución.

Cabe mencionar que no se implementan las páginas internas de este sitio dado que el objetivo es mostrar como manejar JSP con archivos CSS y dar formato a las mismas. Sin embargo, en el código puede verse como ejecutar las páginas internas de este sitio, mediante la instrucción:

`<li><a href=pagCachorros.jsp>Adultos</a></li>` en el menú de arriba (donde aparece el cursor) o

```
<p align="center"><a href='pagEnfermedades.jsp' title=''>
<img src="img/enfermedades.gif" alt=""></a></p>
```

en los botones de comando que yacen debajo de la leyenda Información.

Con esto llegamos al final de este capitulo. Hemos repasado los fundamentos de las paginas JSP sin llegar a profundizar lo que quisiéramos. Las referencias y las investigaciones alternas de este tema te ayudarán a reforzar aquellas partes del libro que más te interesen.

RESUMEN

En este capítulo se dio una introducción a las páginas JavaServer Pages en NetBeans. Se incluyeron algunos temas que son generales en esta estrategia de desarrollo web propuesta por los creadores de Java y que ha sido ampliamente adoptada por la industria y por desarrolladores particulares. En resumen, en este capitulo se trato lo siguiente:

1. La sintaxis propia de las paginas JSP.
2. Scriplet y Servlet en JSP
3. JSP y su vinculación con CSS

EVIDENCIA

- ☐ Amplió la funcionalidad implementada en las operaciones realizadas dentro del código de una página JSP.
- ☐ Relacionó la sintaxis CSS con el código que la referencia en una página JSP.
- ☐ Realizó el análisis del uso de propiedades HTML en una página JSP.

Autoevaluación

1. ¿Cómo se crean los comentarios en JSP?
2. ¿Cuáles son las dos formas de sintaxis que se manejan en Java y que se usan para enviarlas al cliente web?
3. ¿Qué significa la expresión <h1>...</h1>?
4. ¿Qué significa la expresión <li>...?
5. ¿Qué significa la expresión <%! double valor = 0;%>?
6. ¿Qué es un Scriplet?
7. En lugar de Scriplet se recomienda otra estrategia que permite la separación de la lógica de negocios de la presentación. ¿Cuál es esta estrategia?
8. ¿Para que se utiliza la directiva @page en JSP?
9. ¿Qué es un Servlet?
10. ¿Cuáles son los métodos de un Servlet?

REFERENCIAS

Bibliografía

Martín, Antonio, (2010). *Programador certificado Java 2. Curso práctico*, 3a. ed., Alfaomega, México. Aubri, Christoper (2009) CSS 2.1 Adopte las hojas de estilo para dominar los estándares de la web. Ediciones ENI, Barcelona, España.

INNOVA (2001) JavaScript. Editorial Innova, Malaga, España.

Páginas Web recomendadas

1. webtensible (2011) Cómo se hace la estructura de una página web. Obtenido el 22 de noviembre del 2011 desde http://www.wextensible.com/como-se-hace/estructura-pagina-web/
2. desarrolloweb.com (2008) Estilos de borde CSS. Obtenido el 22 de noviembre del 2011 desde http://www.desarrolloweb.com/articulos/estilos-bordes-css.html
3. Eguizabal, Jaime (2008) Simplifica tu CSS: Border. Obtenido el 22 de noviembre del 2011 desde http://www.inkilino.com/simplifica-tu-css-border.html

Respuestas a la autoevaluación

1. <% comentario %>
2. a. <%=expresión Java%> o b. <jsp: expresión> expresión></jps:expresión>
3. Significa construir un encabezado (header) de nivel 1 en una pagina HTML.
4. Significa construir una lista desordenada de elementos en una pagina HTML.
5. Significa declarar una variable *nombre*, tipo double y valor inicial igual cero.
6. Es un bloque de código que se delimita por los caracteres <% y %>
7. Se denomina JSTL que es una librería de etiquetas estándar que apoya el modelo MVC.
8. Para establecer las propiedades generales de una pagina JSP, por una única vez en todas las páginas.
9. Es un programa que se ejecuta en el contenedor web de un servidor de aplicaciones. Se invocan mediante HTTP. Es ejecutado por un contenedor web.
10. service, init y destroy.

Servicios Web en NetBeans 7.1

9

Reflexione y responda las siguientes preguntas

¿Qué tan importante se han vuelto los servicios web para los desarrolladores de aplicaciones web?

Si somos desarrolladores, ¿qué tanto conocemos de las ventajas de utilizar servicios web en nuestras aplicaciones?

¿Es la nube un conjunto de servicios web disponibles para nuestro uso?

Contenido

EXPECTATIVA

Hasta el momento hemos desarrollado aplicaciones web sencillas, estáticas y orientadas a la presentación de componentes. En el presente capítulo utilizaremos una estrategia de implementación de código mediante servicios web. La importancia de los servicios web en una aplicación web radica en la reutilización que podemos establecer de código genérico en muchas aplicaciones. Por ende, es importante conocer su funcionamiento y cómo se implementa en una aplicación web.

Después de estudiar este capítulo, el lector será capaz de:

- Comprender el funcionamiento de los servicios web en el desarrollo de aplicaciones web.
- Crear un servicio web funcional para ser utilizado posteriormente mediante diferentes estrategias.
- Consumir un servicio web para utilizar su funcionalidad.

INTRODUCCIÓN

Los servicios web constituyen componentes de software reutilizable que permiten librar a los desarrolladores de soluciones informáticas de la reescritura y, por tanto, la duplicación de código que puede generalizarse. La funcionalidad de muchos procesos de negocios se pueden implementar mediante codificación genérica que permita usarse una y otra vez en una o varias aplicaciones. Esto es lo que conocemos como reutilización de código. Imaginemos la funcionalidad de un mantenimiento de datos donde los procesos genéricos son insertar, modificar o eliminar registros. Quizás sean pocas las modificaciones que se requieran para usarla en una u otra aplicación: la conexión de la base de datos, los perfiles del usuario, entre otros elementos que pueden ser muy específicos para determinada aplicación.

Al hablar de reutilización no solamente se piensa en aplicaciones web, también existe la forma de implementar esta estrategia en aplicaciones de escritorio, por ejemplo mediante paquetes (packages) en Java o los espacios de nombres (name spaces) en Visual Studio .NET. Los servicios web, sin embargo, trascienden la frontera de la aplicabilidad local (de las soluciones de escritorio) para estar disponibles para su uso a nivel global a través de servicios web publicados en la nube, cuando se trata de una red pública, o en el sitio web corporativo, en caso de una red privada.

Es probable que el análisis de un sistema nos arroje resultados sobre procedimientos que son muy genéricos a muchas de las áreas de las empresas. Con *esta* generalización es posible ponernos a escribir código que implante toda la funcionalidad de esos procesos en un solo componente de software para luego ser

reutilizado en todas esas áreas. Por tal motivo, es interesante conocer y, a la vez necesario, el fundamento de los servicios web y cómo implementarlos en nuestras lógicas de negocio.

Tradicionalmente un servicio web permite que el usuario de un sitio web y el propio sitio web interactúen de una forma transparente: el primero le solicita información al sitio y el servicio web puede proporcionar dicha información. Ramesh Nagappan considera que existen razones poderosas para utilizar servicios web en lugar de meras aplicaciones web. Estas razones son las siguientes:

- Los servicios web pueden ser invocados mediante mecanismos RPC que se basan en XML.
- Los servicios web proveen multiplataforma, soluciones multilenguaje basada en mensajería XML.
- Los servicios web facilitan la integración de aplicaciones al utilizar una infraestructura liviana sin afectar la escalabilidad de la aplicación.
- Los servicios web habilitan la posibilidad de la interoperabilidad de aplicaciones heterogéneas.

Nagappan (2003) considera que los beneficios que nos aportan los servicios web son los siguientes:

- Proporcionan un mecanismo simple para que las aplicaciones se conviertan en servicios que son accesibles por cualquiera, en cualquier lugar y desde cualquier dispositivo.
- Definen servicios basados en conectividad de aplicaciones EAI facilitando las comunicaciones intra e interempresa.
- Definen una solución para las empresas que requieren flexibilidad y agilidad en la aplicabilidad de la aplicación de comunicación a través de Internet.
- Permiten la localización dinámica y la invocación de los servicios a través de corredores de servicio (registros).
- Permiten la colaboración con las aplicaciones existentes que son modeladas como servicios para ofrecer servicios agregados web.

SERVICIOS WEB (WEB SERVICES)

David Chapell se refiere a un servicio web como "*Un servicio web es una pieza de lógica de negocios, situado en algún lugar en internet, que es accesible a través de protocolos de internet basados en estándares como HTTP o SMTP. El uso de un servicio web podría ser tan simple como ingresar a un sitio o tan complejo como*

para facilitar una negociación de múltiples organizaciones empresariales" (Chapell, 2009).

Ramesh Nagappan afirma sobre los servicios web lo siguiente: "*Los servicios web se basan en el concepto de arquitectura orientada al* servicio *(SOA). SOA es la más reciente evolución de la computación distribuida, posibilita el uso de componentes de software, funciones de aplicación, objetos y procesos desde diferentes sistemas, expuestos como servicios"* (Nagappan, 2003).

Un servicio web tiene algunas características de comportamiento especiales. Las mismas son descritas por Chapell como:

- **Basado en XML.** Mediante el uso de la capa de representación de datos para todos los protocolos de servicios web y tecnologías que se crean, estas tecnologías pueden ser compatibles en su nivel básico. Como transporte de datos, XML es muy transparente para cualquier red de datos, sistema operativo o plataforma de enlace que tiene un protocolo de comunicación.
- **Débilmente acoplados.** Un consumidor de un servicio web no está vinculado directamente al mismo. La interfaz del servicio puede cambiar con el tiempo sin comprometer al cliente su interacción con el servicio web. Si el servicio web cambia, probablemente el cambio también se reflejará en el sitio web que lo usa, sin que signifique una dependencia fuerte.

  Un sistema fuertemente acoplado implica que el cliente y el servidor de la lógica están estrechamente vinculados entre sí, lo que significa una dependencia absoluta. La adopción de una arquitectura débilmente acoplada tiende a que los sistemas de software sean más manejables y permiten, en forma sencilla, la integración entre diferentes sistemas.
- **De grano grueso.** Java expone sus servicios a través de métodos individuales. Un método individual es demasiado detallado como para proporcionar cualquier funcionalidad útil a nivel corporativo. La construcción de un programa en Java desde cero requiere la creación de varios métodos de grano fino que luego se conjuntan en un servicio de grano grueso. Este servicio es posteriormente consumido por un cliente o servicio web. Las interfaces y los servicios web que se exponen para una aplicación web deben ser de grano grueso.
- **Capacidad sincrónica o asincrónica.** La sincronización se refiere a la unión de los clientes a la ejecución del servicio. En invocaciones síncronas, el cliente bloquea el servicio y espera a que el servicio termine para continuar. Es decir, espera a que el servicio termine la operación antes de continuar. Operaciones asincrónicas permiten a los clientes invocar un servicio y continuar sus operaciones normales

mientras el servicio se ejecuta. Luego recupera los resultados de la ejecución del servicio. La capacidad asíncrona es un factor clave para los sistemas débilmente acoplados.

- **Admite llamadas a procedimientos remotos (RPCs).** Para invocar procedimientos, funciones y métodos en objetos remotos, los servicios web permiten a los clientes utilizar un protocolo basado en XML. Procedimientos remotos exponen sus parámetros de entrada o salida que el servicio web admite. El desarrollo de componentes a través de Enterprise JavaBeans (EJBs) y .NET se han ido convirtiendo en una parte importante de las arquitecturas y las implementaciones empresariales de software a la fecha. Ambas tecnologías se distribuyen y se hacen accesibles mediante una amplia variedad de mecanismos RPC. Un servicio web es compatible con RPC por la prestación de servicios propios, equivalentes a las de un componente tradicional, o bien mediante la traducción de llamadas entrantes en una invocación de un EJB o un componente en .NET.
- **Apoya el intercambio de documentos.** Una de las principales ventajas de XML es la forma genérica de representar no sólo datos, sino también documentos complejos. Estos documentos pueden ser de simples datos (como una dirección postal, un teléfono o un valor monetario) o uno complejo como lo puede ser un libro o una cotización de múltiples tipos de campos. Los servicios web apoyan el intercambio transparente de documentos para facilitar la integración de negocios.

Nota

Probablemente y de acuerdo a la configuración de NetBeans, en su máquina deberá buscar este tipo de componente. Si no está en la lista de componentes que se presenta en la figura 9.1, deberá ubicarlo abajo del menú, en la opción *Otros*.

TECNOLOGÍAS EMERGENTES DE SERVICIOS WEB

Actualmente han aparecido en el mercado varias tecnologías para la implementación de servicios web, y los diseñadores de compiladores han adoptado los mismos dentro de sus herramientas. Entre estas tecnologías se tiene *Simple Object Access Protocol (SOAP), Web Service Description Language (WSDL), Universal Description, Discovery, and Integration (UDDI)* A continuación se detallan estas tecnologías.

Simple Object Access Protocol (SOAP)

SOAP proporciona una estructura estándar de embalaje para el transporte de documentos XML a través de una variedad de tecnologías estándar de internet, tales como SMTP, HTTP y FTP. Chapell precisa al respecto: *"SOAP proporciona una estructura sencilla para hacer RPC: el intercambio de documentos. Al contar con un mecanismo de transporte estándar, los clientes y servidores heterogéneos pueden llegar a ser interoperables"* (Chapell, 2002). Tanto Microsoft Visual Studio .NET como Java pueden interactuar con servicios SOAP.

En resumen, SOAP es un protocolo estándar *que* posibilita que dos objetos en diferentes procesos puedan comunicarse mediante el intercambio de datos y mensajería XML. Fue creado por Microsoft e IBM y actualmente bajo la tutela de la W3C.

Web Service Description Language (WSDL)

WSDL es una tecnología XML que describe la interfaz de un servicio web en forma estandarizada. Se utiliza muchas veces en combinación con SOAP y XML Schema. Por ejemplo, un programa cliente puede conectarse a un servicio web y determinar a través de WSDL qué funciones están disponibles en un servidor web y realizar una llamada a una de esas funciones a través de SOAP.

Nagappan opina sobre esta tecnología: *"Los servicios web de lenguaje de definición (WSDL) estándar es un formato XML para describir los servicios de red y su información de acceso. Se define un mecanismo de enlace que se utiliza para fijar un protocolo, formato de datos, y el mensaje abstracto, o un conjunto de variables que definen la ubicación de los servicios"* (Nagappan, 2003).

Universal Description, Discovery, and Integration (UDDI)

David Chapell describe esta tecnología de la siguiente manera: *"UDDi proporciona un registro de todos los servicios web para la publicidad, el descubrimiento y los propósitos de integración. Los analistas de negocios y tecnólogos utilizan UDDI para descubrir los servicios web disponibles en la búsqueda de nombres, identificadores, categorías o las implementaciones implementadas por el servicio web"* (Chapell, 2002).

Nagappan, por su parte, afirma que: *"define las interfaces y mecanismos estandarizados para los registros destinados a la publicación y el almacenamiento de las descripciones de los servicios de red en términos de mensajes XML. Es similar a las páginas amarillas o una guía telefónica, donde las empresas ofrecen una lista de sus productos y servicios."* (Nagappan, 2003)

CREAR SU PRIMER SERVICIO WEB

Antes de continuar con la teoría sobre los servicios web, crearemos un ejemplo básico de servicio web en Java utilizando NetBeans. Para ello siga las siguientes instrucciones:

1. Ingresemos a NetBeans y procedamos a crear un nuevo sitio web. Nombre el proyecto *wsBasico*.
2. En la parte de seleccionar el framework no elija alguno. Solamente pulse el botón *Finalizar*.
3. Una vez creado el sitio, agregue un nuevo archivo, el cual será un servicio web. Observe la figura 9.1.

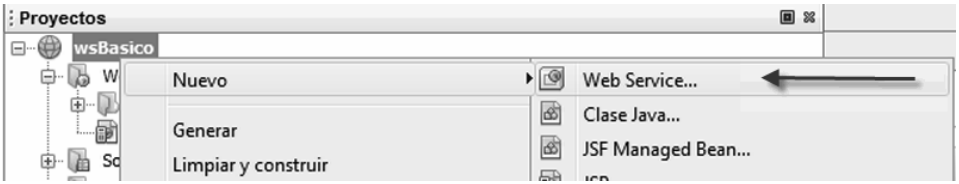


Figura 9.1 Crear el servicio web.

4. En el nombre de la clase denomine este archivo como *wsServicioWeb* y cree un paquete denominado *paqueteServicios*, tal como se muestra en la figura 9.2

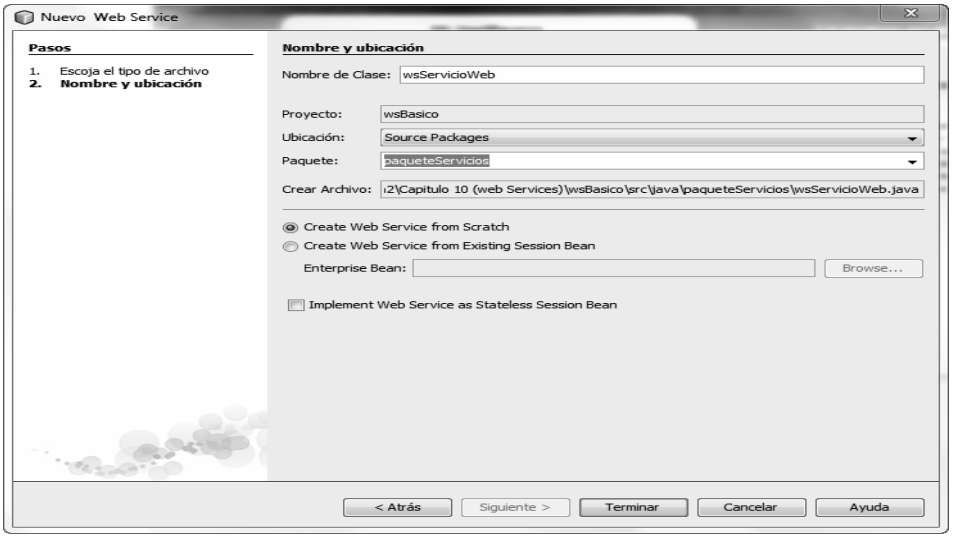


Figura 9.2 Configurar detalles básicos del servicio web.

5. Ya tenemos nuestro servicio web. Ahora debe pulsar sobre el botón *Add Operation...*, tal como se ilustra en la figura 9.3.

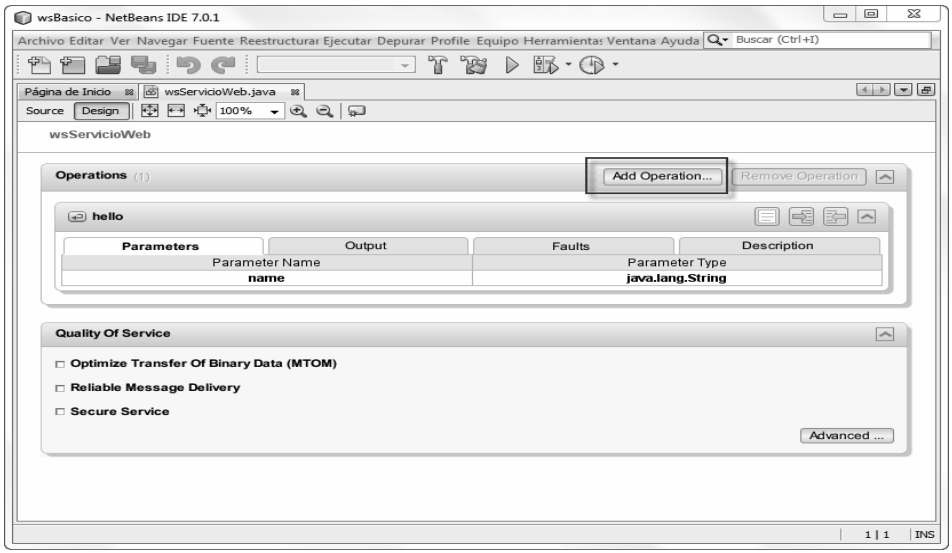


Figura 9.3 Agregar una operación a nuestro servicio web.

6. Una vez pulsado el botón *Add Operation...* se presentará una pantalla como la que muestra la figura 9.4. En esa pantalla deberá nombrar la operación como *devFactorial*, con un parámetro denominado *numFact*, de tipo int.

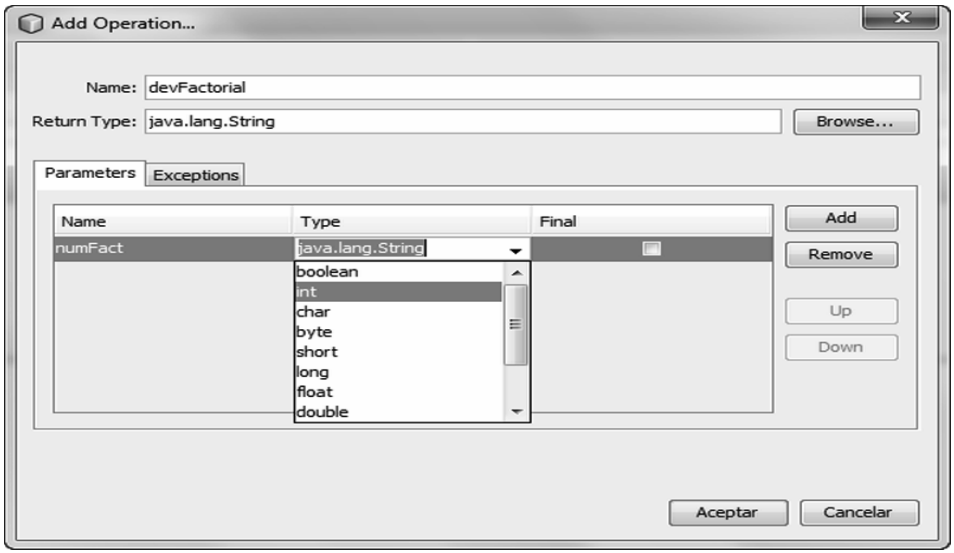


Figura 9.4 Nombrar y configurar la operación del servicio web.

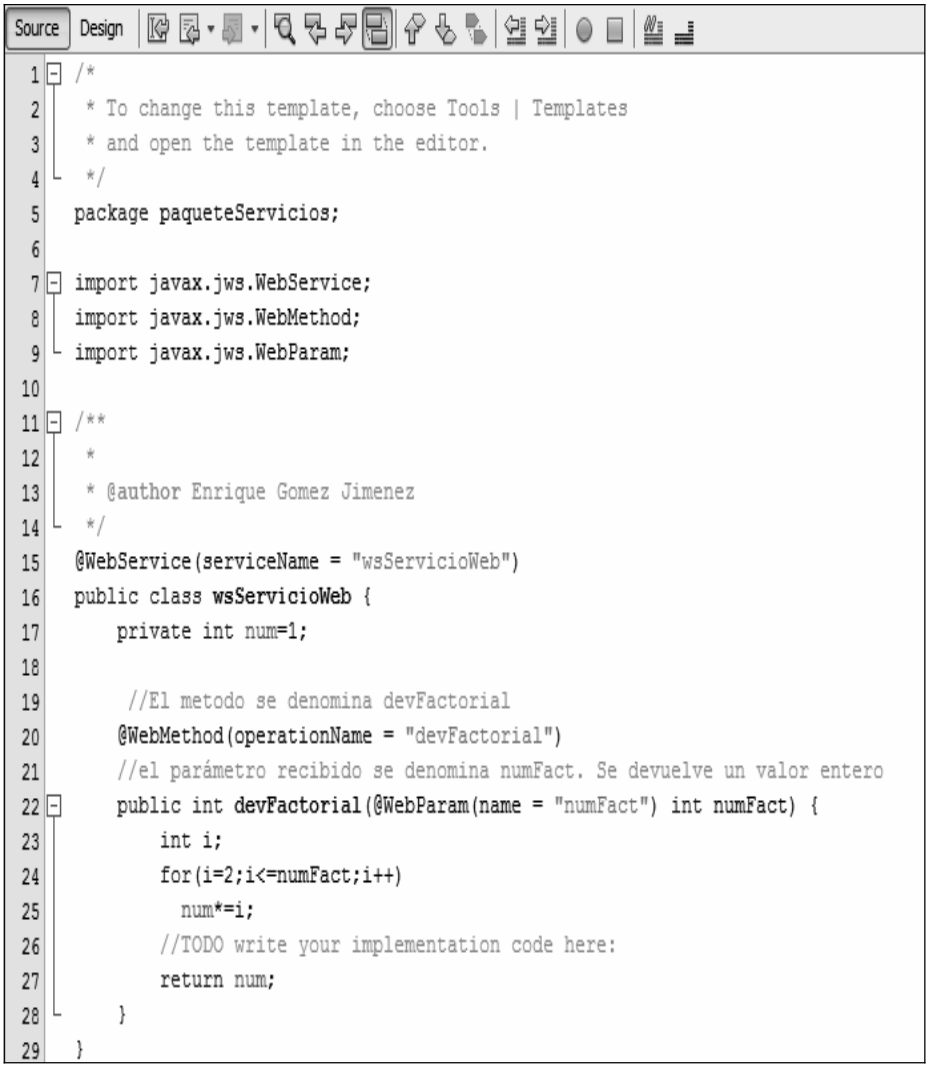
7. Una vez realizada la operación anterior, pulse sobre botón *Aceptar* y se mostrará el editor de código para dicha operación. El código es el siguiente:

EJERCICIO: CREACIÓN DE UN SERVICIO WEB BÁSICO
package paqueteServicios;
import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;
@WebService(serviceName = "wsServicioWeb") public class wsServicioWeb {
private int num=1;
//El metodo se denomina devFactorial
@WebMethod(operationName = "devFactorial")
//el parámetro recibido se denomina numFact. Se devuelve un valor entero
public int devFactorial(@WebParam(name = "numFact") int numFact) {
int i;
for(i=2;i<=numFact;i++)
num*=i;
return num;
}}

Actividades para el lector

Elabore resúmenes que le permitan comprender adecuadamente la funcionalidad de un servicio web.

La figura 9.5 muestra cómo visualiza en el editor de código:



```
Source  Design
1  /*
2   * To change this template, choose Tools | Templates
3   * and open the template in the editor.
4   */
5  package paqueteServicios;
6
7  import javax.jws.WebService;
8  import javax.jws.WebMethod;
9  import javax.jws.WebParam;
10
11 /**
12  *
13  * @author Enrique Gomez Jimenez
14  */
15 @WebService(serviceName = "wsServicioWeb")
16 public class wsServicioWeb {
17     private int num=1;
18
19      //El metodo se denomina devFactorial
20     @WebMethod(operationName = "devFactorial")
21     //el parámetro recibido se denomina numFact. Se devuelve un valor entero
22     public int devFactorial(@WebParam(name = "numFact") int numFact) {
23         int i;
24         for(i=2;i<=numFact;i++)
25           num*=i;
26         //TODO write your implementation code here:
27         return num;
28     }
29 }
```

Figura 9.5 El método de nuestro servicio web.

8. Una vez creada la operación del servicio web se puede probar la funcionalidad del servicio web. Pulse el menú *Ventana* y seleccione la opción *Proyectos.* Aparecerá el administrador de proyectos con todos los objetos que se han creado.

9. Para probar la funcionalidad de nuestro servicio web se tiene que ubicar en la carpeta *Web Services* del proyecto. Una vez ubicado en dicha carpeta, seleccionar el proyecto y pulsar el botón derecho sobre el mismo y ejecutar *Test Web Service,* tal como se observa en la figura 9.6.

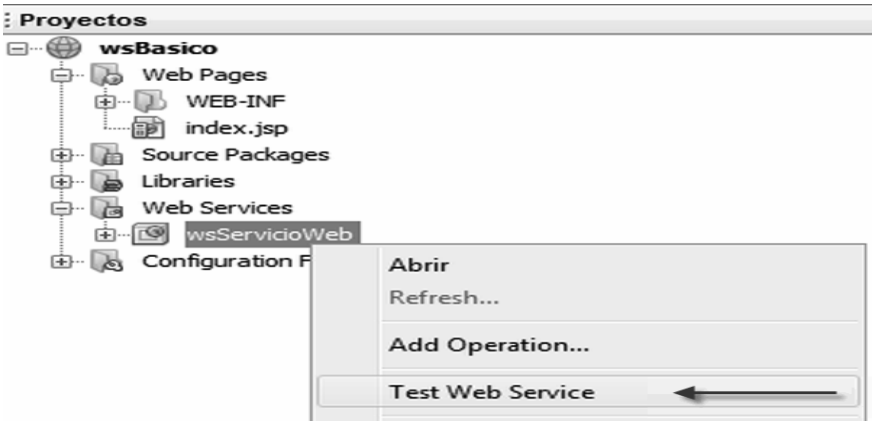


Figura 9.6 El método de nuestro servicio web.

10. Ahora ya podemos ejecutar nuestro servicio web, basta con dar el valor del parámetro y pulsar el botón *devFactorial* para probarlo. Observe la figura 9.7 respecto a esta interface.

Figura 9.7 La interface web para probar nuestro servicio web.

11. Si da un valor (por ejemplo 5) y pulsa el botón *devFactorial* se puede observar el resultado de la ejecución del servicio web, tal como se muestra en la figura 9.8.

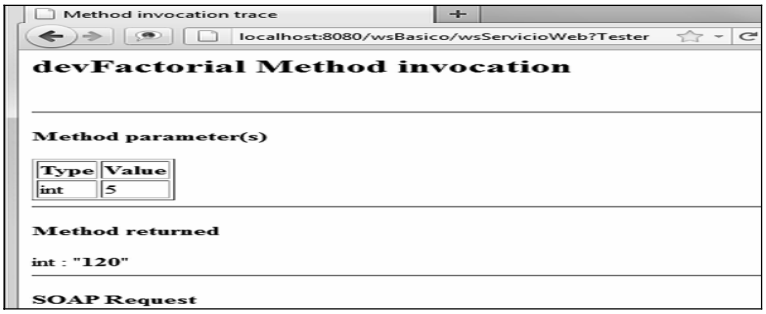


Figura 9.8 Resultados de la prueba de nuestro servicio web.

Observe en la figura 10.8 que el parámetro enviado por el usuario es 5 y el método del servicio web lo convierte a su número factorial (120).

12. Nos damos cuenta que nuestro servicio web funciona, además NetBeans genera automáticamente el código SOAP que requiere el sitio para su ejecución (tanto para la solicitud al servidor como la respuesta de éste al cliente). En la figura 9.9 vemos el código SOAP para la petición al servidor (Request) y en la 9.10 la respuesta (Request).

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Header/>
    <S:Body>
        <ns2:devFactorial xmlns:ns2="http://paqueteServicios/">
            <numFact>5</numFact>
        </ns2:devFactorial>
    </S:Body>
</S:Envelope>
```

Figura 9.9 Código SOAP generado para la petición al servidor (SOAP Request).

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Body>
        <ns2:devFactorialResponse xmlns:ns2="http://paqueteServicios/">
            <return>120</return>
        </ns2:devFactorialResponse>
    </S:Body>
</S:Envelope>
```

Figura 9.10 Código SOAP generado para la respuesta del servidor (SOAP Response).

13. Finalmente, observe la figura 9.11 la cual es la conformación del archivo WDSL, que prácticamente describe la funcionalidad del método web.

Ya tenemos nuestro servicio web creado. Falta ahora crear nuestra interface de consumo, es decir el programa que permitirá consumir el servicio web creado.

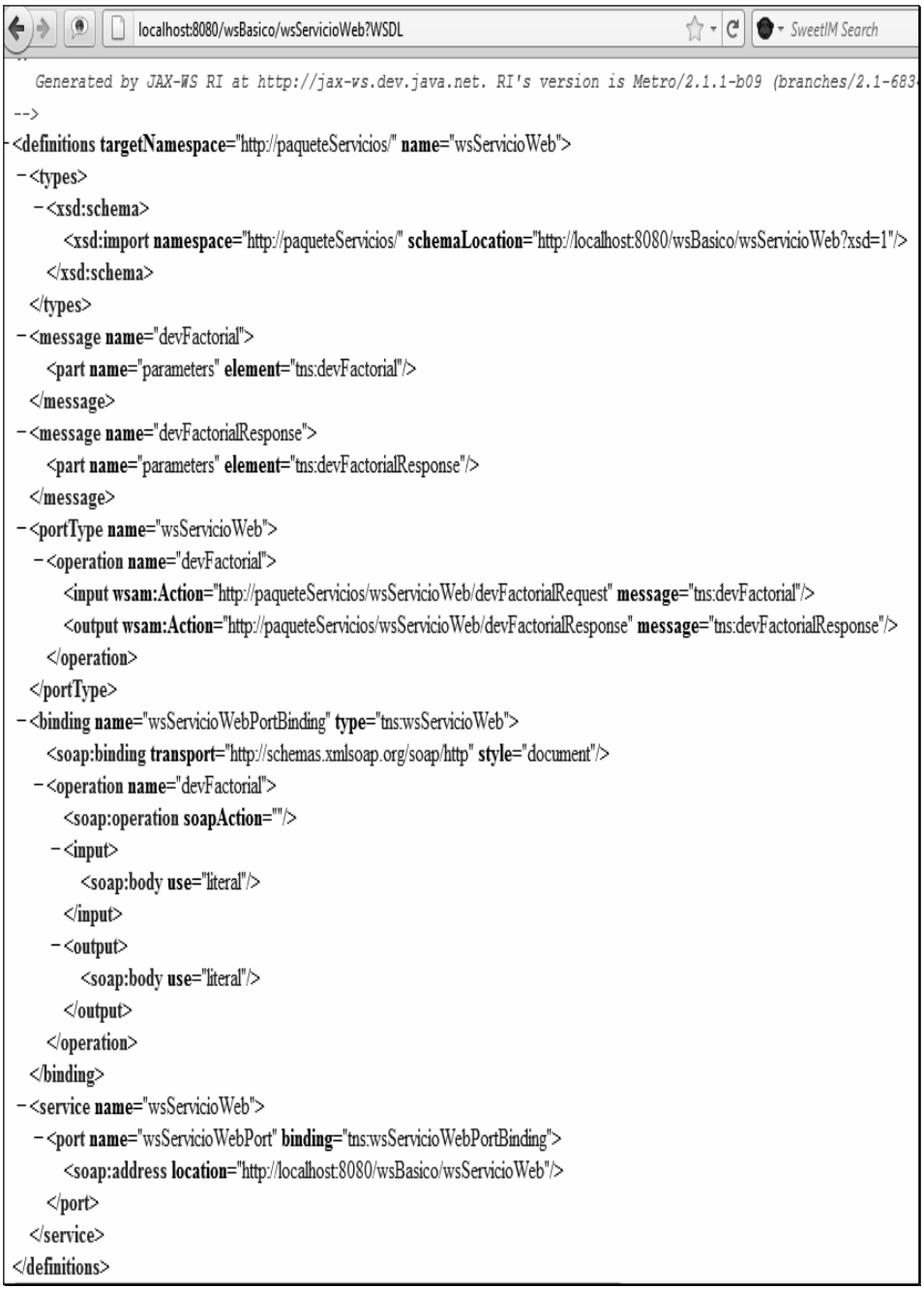


Figura 9.11 Archivo WSDL descriptivo del servicio web.

CONSUMIR SU PRIMER SERVICIO WEB

Para consumir un servicio web se dispone de tres formas distintas de hacerlo. Mediante una clase Java en una aplicación Java SE, un Servlet en una aplicación web y en una página JSP en una aplicación web.

Consumir el servicio Web en una aplicación Java SE

14. Iniciamos con una aplicación Java SE que consuma nuestro sitio web. Vea la figura 2.1 (capítulo 1) para la creación de este tipo de aplicación. Nombre esta aplicación como *javaSEws*.

15. Agregamos un cliente del servicio web (*Web Service Client…*) tal como se observa en la figura 9.12. Si no aparece tal como se muestra en la figura 10.12 puede seleccionar *Otro…* al final de la ventana de objetos y buscar ahí *Web Service*s en las opciones de *Categoría*s y *Web Service Client…* en las de *Archivo*s.

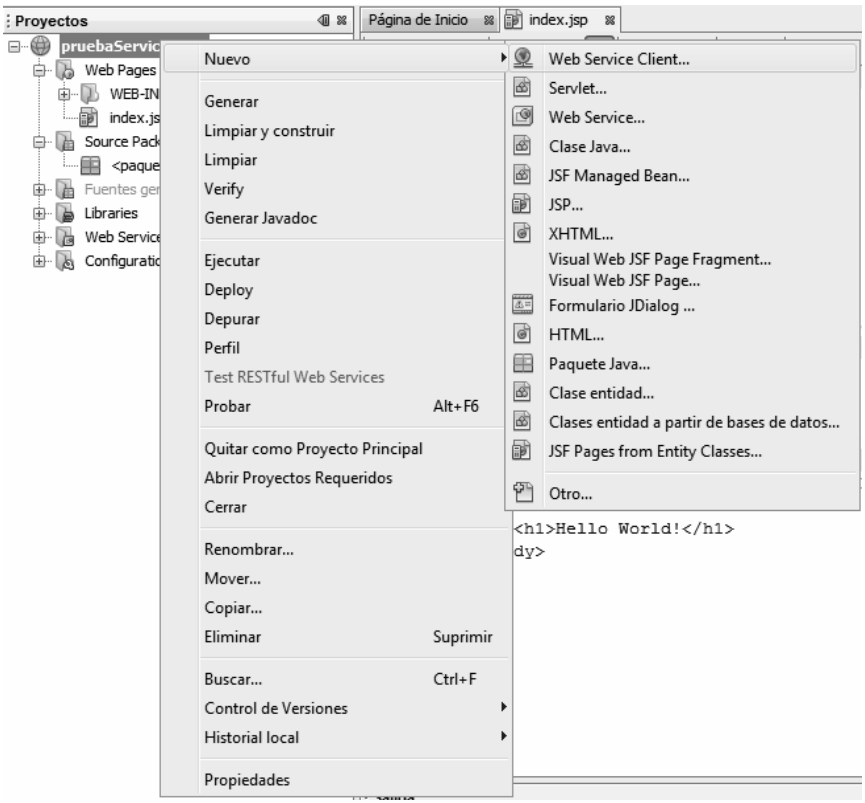


Figura 9.12 Agregar un servicio Web, tipo cliente.

16. Una vez seleccionada la opción de *Web Service Client…* escogeremos el servicio web que utilizaremos, en este caso *wsBasico*, tal como se observa en la figura 9.13. Pulse el botón *Terminar* para iniciar el proceso de implementación del servicio web en esta aplicación Java SE.

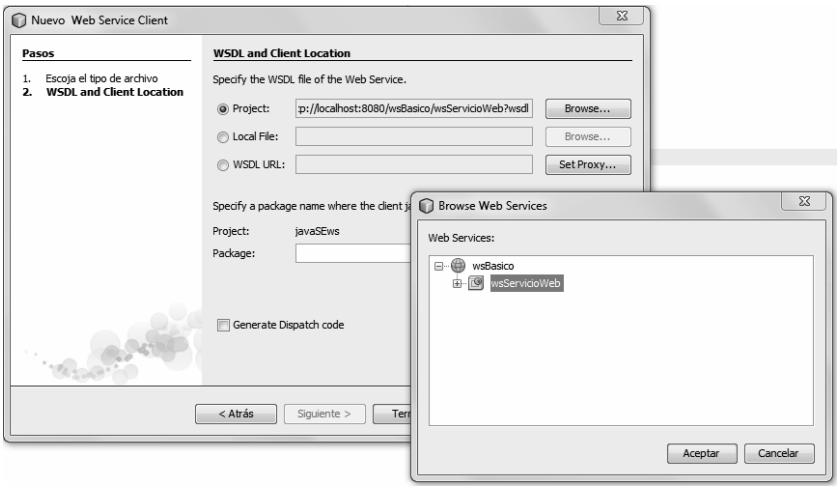


Figura 9.13 Implementar el servicio web en una aplicación Java SE.

17. Una vez implementado el servicio web en nuestra aplicación Java SE veremos cómo está configurada. Observa la figura 9.14 con la implementación del servicio.

Figura 9.14 Configuración de la aplicación Java SE una vez implementado el servicio web.

18. El siguiente paso consiste en arrastrar el servicio web (donde dice *devFactorial*) hacia nuestra aplicación Java SE. Observa la figura 9.15 que muestra este procedimiento.

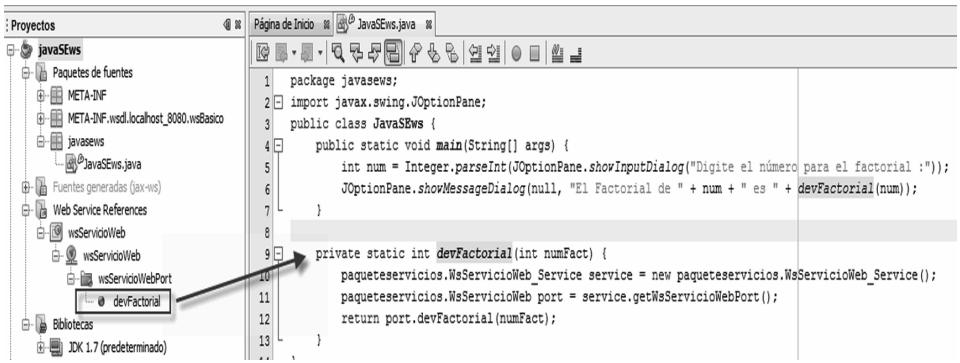


Figura 9.15 Configuración de la aplicación Java SE una vez implementado el servicio web.

19. Observe que implementamos el código para utilizar el servicio web. Dicho código se explica a continuación:

CONSUMIENDO EL SERVICIO WEB EN UNA APLICACIÓN JAVA SE
package javasews; //paquete de nuestra aplicación Java SE
import javax.swing.JOptionPane; //librería necesaria para usar JOptionPane
public class JavaSEws { //inicio de nuestra aplicación Java SE
public static void main(String[] args) { //el main de nuestra aplicación Java SE
//Se carga la variable num mediante un JOptionPane int num = Integer.parseInt(JOptionPane.showInputDialog("Digite el número para el factorial :"));
//Se hace el llamado a *devFactorial* que es la implementación del servicio web JOptionPane.showMessageDialog(null, "El Factorial de " + num + " es " + devFactorial(num));
}
//código heredado del servicio web... private static int devFactorial(int numFact) {
paqueteservicios.WsServicioWeb_Service service = new paqueteservicios.WsServicioWeb_Service();
paqueteservicios.WsServicioWeb port = service.getWsServicioWebPort();
return port.devFactorial(numFact);
}}

20. Una vez implementado el servicio web podemos ejecutar nuestra aplicación Java SE teniendo el resultado que se muestra en la figura 9.16.

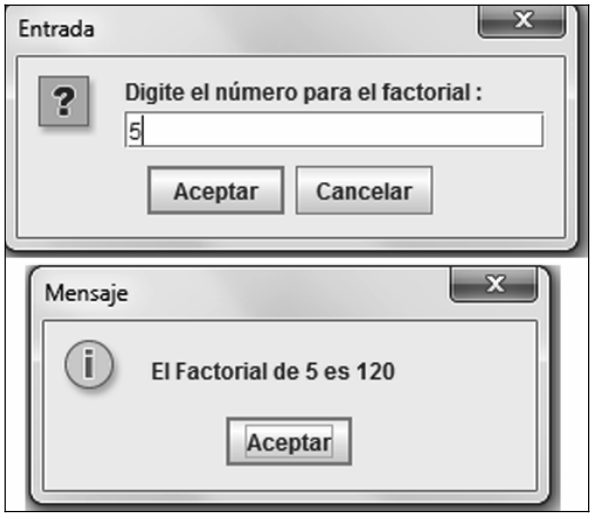


Figura 9.16 Resultado de ejecutar nuestra aplicación Java SE que consume un servicio web.

Con este ejercicio hemos implementado nuestro servicio web en una aplicación Java SE.

Consumir su servicio web en Servlet de una aplicación web

Un Servlet es un objeto que se ejecuta en el contexto de un contenedor de Servlets, como por ejemplo en un servidor Tomcat, permitiendo con ello extender su funcionalidad. Se deriva de *applet* que era un objeto que consistía en pequeños programas que se ejecutaban en el contexto de un navegador web (cliente). En síntesis, el *applet* se ejecutaba en el contexto del navegador (cliente) y el *Servlet* en el contexto de un servidor. La función del *Servlet* es generar páginas web dinámicamente de acuerdo con los parámetros recibidos.

La mecánica para consumir un servicio web en una aplicación web mediante un Servlet inicia creando un proyecto web, a continuación se crea el Servlet y finalmente se utiliza éste para consumir el servicio web.

21. Procederemos, entonces, a crear nuestro proyecto web, el cual llamaremos *wsServletClient*. Repita los pasos 15, 16 y 17 que realizamos para implementar un servicio web en una aplicación Java SE.

22. Una vez implementado el servicio web en nuestra aplicación web proceda a agregar un Servlet, tal como se muestra en la figura 9-17.

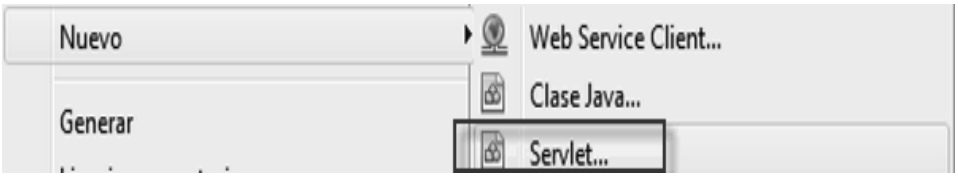


Figura 9.17 Crear un Servlet.

23. El Servlet se llamará *servletFactorial* y el paquete *packServletFactorial*. Luego de nombrarlos pulse el botón *Finalizar*.

24. Vamos ahora al proyecto web (*wsServletClient*) y pulsamos sobre el nombre del mismo y seleccionamos propiedades. Ahí, en *run* vamos a configurar que el punto de entrada de la aplicación sea el Servlet creado (*servletFactorial*), tal como se observa en la figura 9.18.

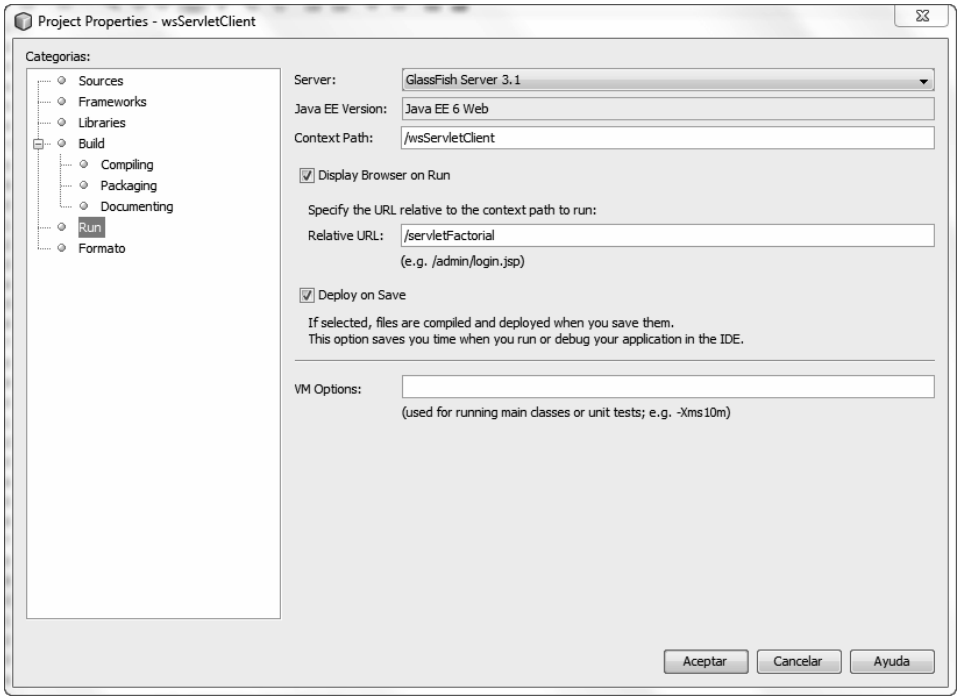


Figura 9.18 Configurar nuestra aplicación web para que el Servlet *wsServletClient* sea el punto de entrada de la misma.

25. Pulsamos el botón *Aceptar* e inmediatamente se generará lo necesario para que el Servlet sea el punto de acceso de nuestra aplicación.

26. Regresamos al Servlet *wsServletClient*. Procedamos a realizar el paso 18 de este ejercicio (arrastrar el servicio web hacia el código del Servlet) y luego

buscamos el método *ProcessRequest* y le quitamos el comentario al código que aparece comentado. El código finalmente queda de la siguiente forma:

CONSUMIENDO EL SERVICIO WEB EN UN SERVLET

```
package packServletFactorial;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.xml.ws.WebServiceRef;
import paqueteservicios.WsServicioWeb_Service;
@WebServlet(name = "servletFactorial", urlPatterns = {"/servletFactorial"})
public class servletFactorial extends HttpServlet {
@WebServiceRef(wsdlLocation =
                "WEB-INF/wsdl/localhost_8080/wsBasico/wsServicioWeb.wsdl")
private WsServicioWeb_Service service;
protected void processRequest(HttpServletRequest
                                request, HttpServletResponse response)
throws ServletException, IOException {
response.setContentType("text/html;charset=UTF-8");
PrintWriter out = response.getWriter();
try {
out.println("<html>");
out.println("<head>");
out.println("<title>Servlet servletFactorial</title>");
out.println("</head>");
out.println("<body>");
//este string es el valor que recibe de un llamado al servlet
//String x = request.getParameter("x");
```

CONSUMIENDO EL SERVICIO WEB EN UN SERVLET
String x="5";
int y = devFactorial(Integer.parseInt(x));
out.println(" Para " + x + " el factorial es " + y + " ");
out.println("</body>");
out.println("</html>");
} finally {
out.close();
}}
// <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click on the + sign on the left to edit the code.">
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
processRequest(request, response);
}
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
processRequest(request, response);
@Override
public String getServletInfo() {
return "Short description";
}// </editor-fold>
private int devFactorial(int numFact) {
paqueteservicios.WsServicioWeb port = service.getWsServicioWebPort();
return port.devFactorial(numFact);
}}

27. La ejecución de este Servlet será similar al de la figura 9-19.

Figura 9.19 Ejecución de nuestro servicio web consumido en un servlet.

Consumir su servicio web en una página JSP de una aplicación web

Ahora toca consumir el servicio web en una página JSP de una aplicación web. Para ello realicemos el siguiente procedimiento.

28. Vamos a crear un nuevo proyecto web denominado pruebaServicioWeb y creamos, igual que en los ejercicios anteriores, un servicio web cliente y lo agregamos a nuestra aplicación web, tal como se muestra en la figura 9.20

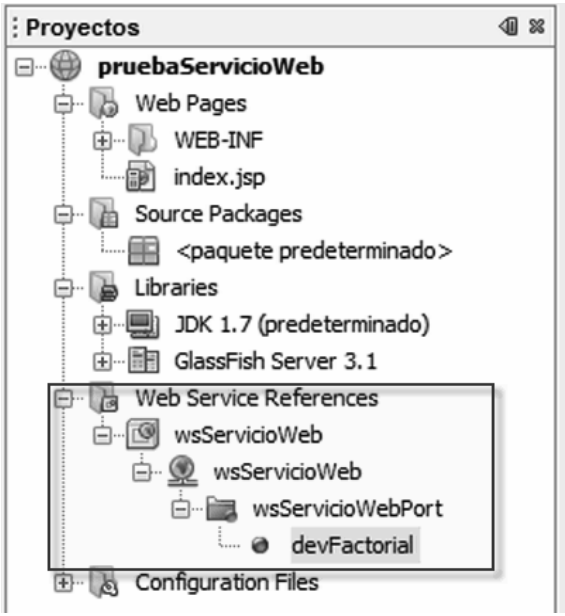


Figura 9.20 Servicio web en nuestra página web JSP.

29. Una vez creada la instancia del servicio web podemos utilizar su funcionalidad arrastrando el nombre de la operación creada en el servicio web (en nuestro caso *devFactorial*) hacia la página JSP, tal y como se muestra en la figura 9.21.

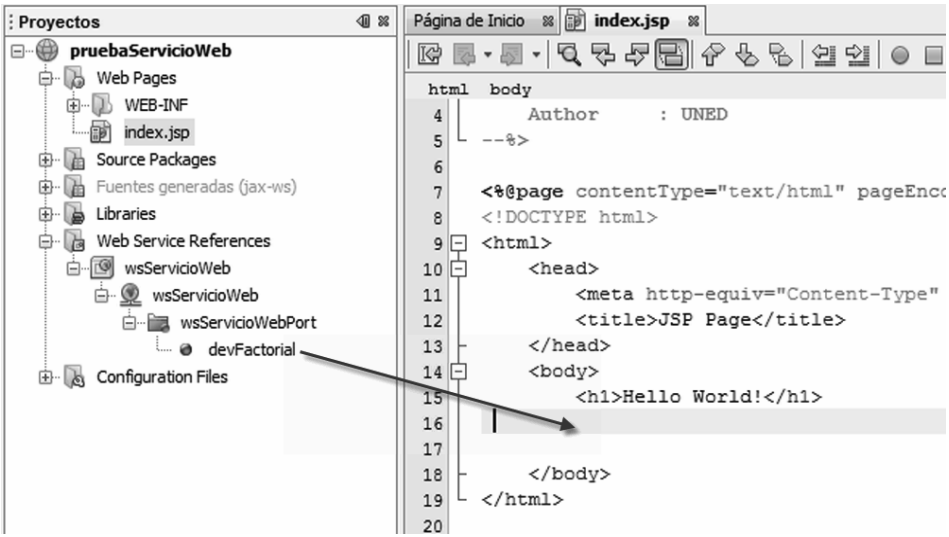


Figura 9.21 Usar el servicio web en nuestra página JSP.

30. Al arrastrar la operación, automáticamente se creará el código necesario para implementar y crear la funcionalidad necesaria para usar el servicio web, tal y como se muestra en la figura 9.22.

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Hello World!</h1>
     <%-- start web service invocation --%><hr/>
    <%
    try {
        wsFactorial.WsServicioWeb_Service service = new wsFactorial.WsServicioWeb_Service();
        wsFactorial.WsServicioWeb port = service.getWsServicioWebPort();
         // TODO initialize WS operation arguments here
        int numFact = 0;
        // TODO process result here
        int result = port.devFactorial(numFact);
        out.println("Result = "+result);
    } catch (Exception ex) {
        // TODO handle custom exceptions here
    }
    %>
    <%-- end web service invocation --%><hr/>
```

Figura 9.22 Código fuente del servicio web en nuestra página JSP.

Observe que automáticamente se implementa el servicio web, creando una instanciación del mismo (*service*). Asimismo, crea de manera automática una variable de prueba (*numFact*). Ahora modificamos ese código para que podamos establecer una prueba más personalizada. Este código es el siguiente:

PROBAR EL SERVICIO WEB EN UNA PÁGINA JSP

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <title>JSP Page</title>
  </head>
<body>
<h1>Prueba del Servicio web</h1>
<%-- start web service invocation --%><hr/>
<%
try {
//se instancia el servicio web
wsFactorial.WsServicioWeb_Service service =
                                    new wsFactorial.WsServicioWeb_Service();
//retorna el servicio web
wsFactorial.WsServicioWeb port = service.getWsServicioWebPort();
int numFact= 5;
//se obtiene el resultado de la operación que radica en el servicio web
int result = port.devFactorial(numFact);
//se muestra el resultado de ejecutar la operación del servicio web
out.println("El Factorial de  " + numFact + " es" + result);
} catch (Exception ex) {
out.println("Error"+ex.getMessage());
}
%>
<%-- end web service invocation --%><hr/>
</body>
</html>
```

31. La ejecución de este nuevo proyecto se muestra en la figura 9.23

Figura 9.23 Página JSP consumiendo el servicio web.

Servicios web RESTful

Los servicios RESTful se construyen con el estilo arquitectónico de REST. RESTful se ha convertido en una alternativa muy común como alternativa al uso de SOAP para el despliegue de servicios web en Internet, debido a su carácter ligero y la capacidad de transmitir datos directamente desde HTTP.

NetBeans permite la construcción de servicios web RESTful. Asimismo, permite crear *las* pruebas, las aplicaciones cliente que acceden a los servicios web REST y generación de código para invocar los servicios web (tanto en REST y SOAP). A continuación presentamos las características de NetBeans con respecto al soporte de estos servicios:

- Rápida creación de servicios web RESTful desde las clases entidad JPA y los patrones.
- Generación rápida de código requerido para invocar los servicios web de repositorios como Google Maps, Yahoo Search News, entre otros.
- Generación de clientes Java REST para los servicios registrados en el administrador de Web Services.
- Pruebas de generación de clientes para los testing de servicios web RESTful.
- Vista lógica para una fácil navegación de clases REST para la implementación de servicios web en un proyecto web.
- Totalmente integrado a Spring Framework.

Rest (Representational State Transfer) ha ganado mucha aceptación como alternativa a SOAP y servicios web basados en WSDL (Web Services Description Language). Varios proveedores de Web 2.0 ocupan a esta tecnología, incluyendo gigantes como Yahoo, Google o Facebook, quienes han marcado como obsoletas otras tecnologías antes mencionadas.

Para manejar un servicio web que utilice RESTful necesitamos una clase donde implementemos algún método. Existen sólo cuatro métodos públicos que podemos crear en esta clase y que constituyen métodos HTTP disponibles para

RESTful: GET, POST, DELETE y PUT. En el caso de GET y POST, básicamente determinan que hará REST sobre nuestra aplicación web. En resumen, estos métodos permiten:

a. GET: permite obtener un valor que bien puede ser un listado de objetos.

b. POST: permite guardar un valor u objeto en la aplicación.

c. DELETE: permite eliminar un objeto.

d. PUT: permite actualizar un objeto.

La clase que implementemos con este recurso actuará como un EJB que manejará la persistencia de una entidad, accedida por una aplicación web. IBM afirma: *"Los servicios web RESTful han surgido como una alternativa prometedora distinta a los servicios basados en SOAP por su simplicidad y naturaleza liviana, además de la capacidad de transmitir datos directamente sobre HTTP"* (IBM, 2011).

Ahora procederemos a crear un pequeño ejemplo de servicios web RESTful. Iniciemos:

1. Crear un nuevo proyecto web en NetBeans. Nombremoslo *webRESTful.*
2. Agregue una nueva clase de nombre Factorial y escriba el siguiente código:

CREAR EL SERVICIO WEB RESTFUL

```
//con la siguiente instruccion convierte automáticamente
//nuestra clase Factorial en un EJB
@Stateless
//con la siguiente instruccion se indica que este recurso
//se accesará vía web desde la ruta "/factorial"
@Path("/factorial")
public class Factorial {
  //se crea el método GET que recibe el parámetro numero
     @GET
     public String factorial(@QueryParam("numero") long numero) {
        //retorna el factorial de un numero
        return Long.toString($factorial(numero));
     }
long $factorial(long numero) {
  if (numero >= 1) {
  return $factorial(numero - 1) * numero;
}
return 1;
}
}
```

3. Una vez que haya escrito este código, proceda a guardar la clase. Inmediatamente NetBeans detectará que se ha creado un recurso REST, solicitándote que active la característica en la aplicación, para accionar los recursos REST correspondientes. Deje la ruta de los recursos REST tal como se indica en la pantalla. Pulse el botón *Aceptar*; véase la figura 9.24.

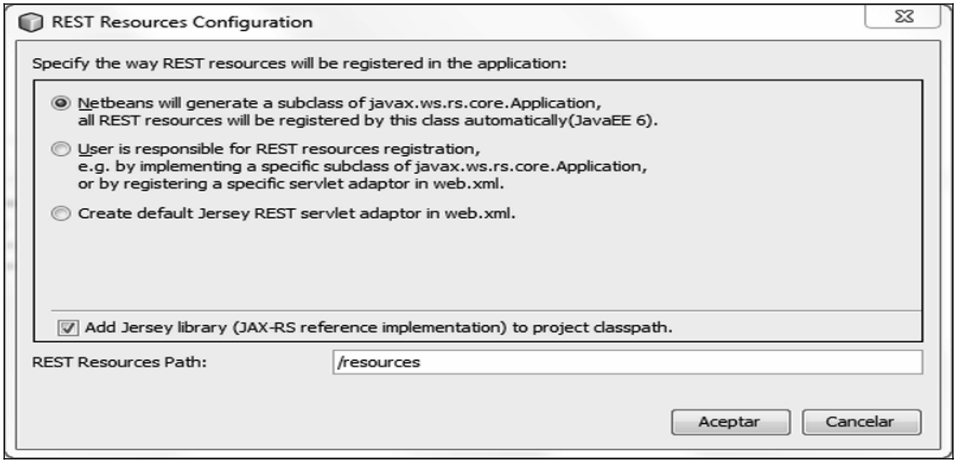


Figura 9.24 Especificar la ruta de los recursos a registrar en la aplicación.

4. Ahora pulse sobre el nombre del proyecto (*webRESTful)* y seleccione la opción *Test RESTful Web Services* e inmediatamente se mostrará una pantalla como la de la figura 9.25.

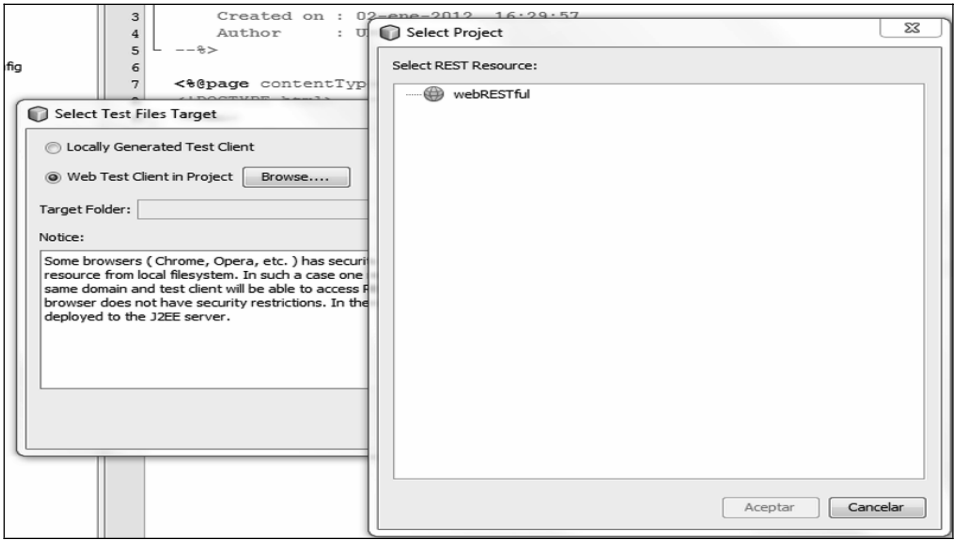


Figura 9.25 Probando nuestro servicio web RESTful.

5. En la figura 9.25 pulse sobre *webRESTful* e inmediatamente la tecla *Aceptar*. Verá en la figura 9.26 la ejecución de este servicio web.

Figura 9.26 Probando nuestro servicio web RESTful.

6. Según la figura 9.26 demuestra cómo probar nuestro servicio web RESTful y podemos comprobar que lo hace bien.

7. Ahora, para consumirlo en una página JSP basta con modificar el código que se crea automáticamente en *index.jsp* por el siguiente:

PROBADO EL SERVICIO WEB RESTFUL EN UNA PÁGINA JSP
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html> <head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Probando nuestro servicio web RESTful</title>
</head>
<body>
Calculando factorial
<form action="resources/factorial">
Número a determinar el factorial: <input name="numero" type="text" />
<button>Calcular</button>
</form>
</body>
</html>

8. Ejecute ahora la página web y se mostrará algo similar a la figura 9.27.

Figura 9.27 Ejecutar el servicio web RESTful en una página JSP.

El resultado de la ejecución de esta página JSP se presentará automáticamente en otra página JSP.

Tambien podemos probar nuestro servicio web RESTful escriendo la siguiente dirección en nuestro navegador:

http://localhost:8080/webRESTful/resources/factorial?numero=5

Donde:

a) **localhost:8080:** es la dirección de nuestro servidor virtual y el puerto donde se ejecutará la aplicación.

b) **webRESTful**: es el nombre de nuestra aplicación web que contiene el servicio web.

c) **resources:** ubicación de los recursos REST de la aplicación. Observe la figura 9.24 donde le indicamos esta ruta.

d) **factorial:** el recurso indicado por la anotación @Path.

e) **numero:** es el párametro para determinar el factorial. Cambiándolo en el propio navegador podrá generar el cálculo requerido.

Actividades para el lector

- Desarrolle un servicio web que realice las operaciones elementales (suma, resta, multiplicación y división) y consúmalo en una página web JSP. Considere su elaboración mediante un Servlet o mediante RESTful.
- Desarrolle un servicio web que permita generar el pago de un empleado, considerando horas trabajadas y precio por hora. Considere que si se labora más de 85 horas se le deberá cancelar tiempo extra (tiempo y medio). Consulte el servicio web en una página JSP, utilizando un Servlet o RESTful.

RESUMEN

En este capítulo se desarrolló el tema de los servicios web, considerando las tecnologías emergentes , así como la creación de ejemplos básicos de su creación.

Básicamente, se desarrollaron los siguientes temas:

1. Los servicios web y los fundamentos que soportan su uso y funcionamiento. Su creación mediante Servlet o RESTful.
2. Las tecnologías emergentes en la actualidad (WSDL, UDDI).
3. Cómo crear y consumir un servicio web.

Autoevaluación

1. ¿Qué es un servicio web?
2. Cite al menos dos razones para utilizar servicios web.
3. Cite al menos dos beneficios que aportan los servicios web.
4. Cite al menos dos características de comportamiento de un servicio web.
5. Cite las tecnologías emergentes de un servicio web.

EVIDENCIA

☐ Elaboró resúmenes que le permitan comprender adecuadamente la funcionalidad de un servicio web.

☐ Desarrolló ejemplos alternativos que incluyen operaciones aritméticas o lógicas utilizando servicios web.

REFERENCIAS

Bibliografía

1. Chapell, David (2002) *Java Web Services: Using Java in Service-Oriented-Architectures*. O'Reilly & Associates, Inc. Publishing, Sebastopol, CA.
2. Nagappan, Ramash (2003) Pro JSF: Developing Java Web Services: Architecting and Developing Secure Web Services using Java. Whiley Publishing, Inc. Indianapolis, Indiana.

Páginas Web recomendadas

1. Belón, Juan (2010) *Creación de Servicios Web con NetBeans y Glassfish – JAVA y PHP*. Obtenido el 28 de noviembre del 2011 desde http://www.programadorphp.org/blog/cursos/creacion-de-servicios-web-con-netbeans-y-glassfish-java-y-php/
2. Exforsys Inc (2011) *Servlets Advanced.* Obtenido el 28 de noviembre del 2011 desde http://www.exforsys.com/tutorials/j2ee/servlets-advanced.html
3. IBM (2011) *wa-aj-multitier.* Obtenido el 28 de noviembre del 2011 desde http://www.ibm.com/developerworks/ssa/library/wa-aj-multitier/index.html

Respuestas a la autoevaluación

1. Es un componente de software que permite la reutilización de código.
2. Invocación mediante RPC basado en XML, integración de aplicaciones, interoperabilidad de aplicaciones, soluciones multilenguaje.
3. Accesibilidad, conectividad de aplicaciones, flexibilidad y agilidad de comunicación en internet, colaboración entre aplicaciones.
4. Comportamiento basado en XML, por acoplamiento, según los llamados de procedimientos remotos (RPC)
5. SOAP, WSDL, UDDI.

Gestión de bases de datos en aplicaciones Web con NetBeans 7.1

10

Reflexione y responda las siguientes preguntas

¿Qué tan importante es el uso de bases de datos en aplicaciones web?

¿Existen mayores facilidades del lado de ASP .NET que de JSP para el diseño de interfaces web que permitan la gestión automática de datos?

¿Cuál será el panorama actual con respecto a MySQL y su adquisición por parte de Oracle?

Contenido

EXPECTATIVA

Ya se trató el tema de la gestión de bases de datos en el capítulo 5. La tecnología de gestión es casi la misma, diferenciándose en la implementación en la propia aplicación web. Al finalizar este capítulo, el estudiante será capaz de:

- Crear consultas de datos MySQL mediante páginas JSP.
- Crear páginas JSP que permitan la inclusión de datos en una tabla MySQL.
- Crear páginas JSP que permitan la modificación de datos en una tabla MySQL.
- Crear páginas JSP que permitan la eliminación de datos en una tabla MySQL.

Después de estudiar este capítulo, el lector será capaz de

- Comprender la gestión de datos mediante páginas JSP y MySQL.
- Crear aplicaciones de gestión de datos utilizando MySQL y JSP.

INTRODUCCIÓN

La gestión de bases de datos es un procedimiento esencial en una aplicación informática. Permite mantener y consultar información relevante para empresarios, clientes y usuarios en general. Resulta entonces de suma importancia conocer cómo realizar una gestión adecuada de datos, principalmente en un ambiente web donde el acceso y la seguridad son aspectos muy importantes a considerar.

Tal como se hizo en el capítulo 5, para realizar una conexión a una base de datos mediante JSP o un Servlet, se utiliza un driver. Este driver es suministrado generalmente por la proveedora del DBMS (Data Base Management System). Este archivo constituye un JAR que contiene una serie de clases que implementan la funcionalidad necesaria para la gestión de datos.

Asimismo, en el capítulo 5 se explicó la arquitectura de gestión de datos con el JDBC, los objetos que la componen considerando MySQL, entre otros factores. Por tanto, le invito a que repase los conceptos, dado que en este capítulo se creará un ejemplo orientado a aplicaciones web.

GENERAR UN LISTADO GENERAL EN UN SOLO ARCHIVO JSP

Se utilizará la tabla tbEstudiantes creada en el capítulo 5 (véase la figura 5.9 y siguientes) para generar un listado general. Para iniciar nuestro primer ejercicio realice los siguientes pasos:

1. Ingrese a NetBeans 7.1 y cree un proyecto web. Nómbrelo *webDatos*.
2. A continuación se establece la referencia al driver MySQL. Para ello pulse sobre la carpeta *Libraries* y seleccione *Agregar biblioteca* del menú de contexto que aparece. Ahí deberá agregar *Driver MySQL JDBC,* tal como se muestra en la figura 10.1.

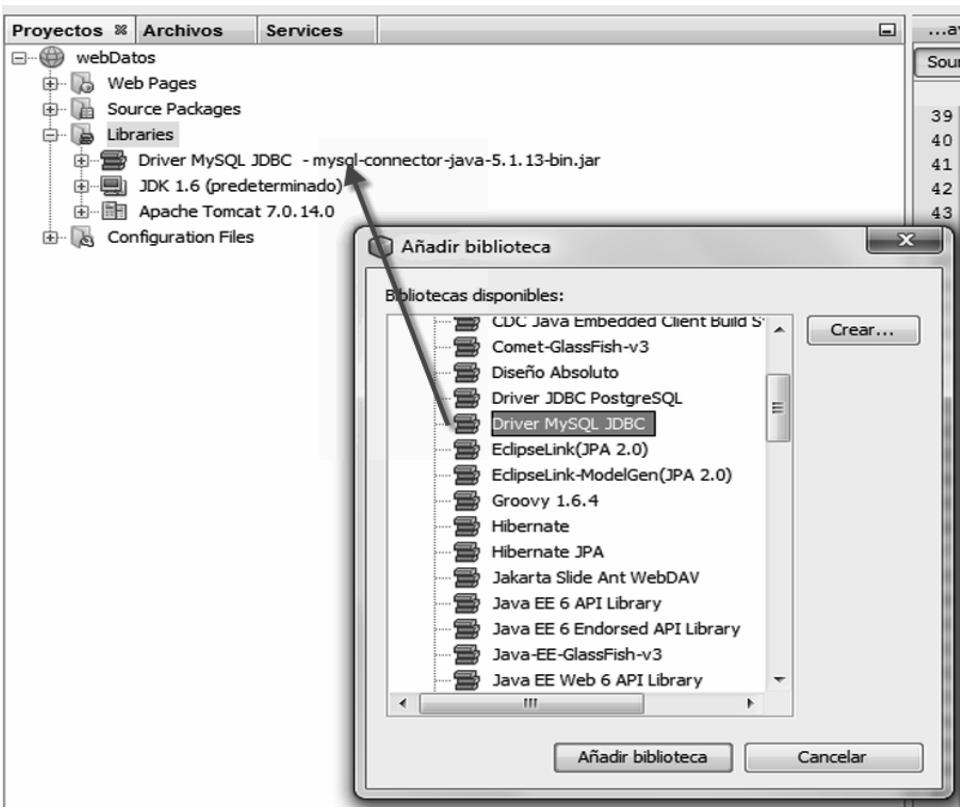


Figura 10.1 Referenciar la conectividad MySQL.

3. Cree una página JSP de nombre *listadoGeneral.jsp* y sustituya el código por el siguiente. Observe las líneas de comentarios que aparecen antes de cada línea.

NUESTRA PÁGINA LISTADOGENERAL.JSP
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
<!--Se realizan las importaciones para utilizar los objetos de gestión de datos -->
<%@ page import="java.sql.ResultSet" %>
<%@ page import="java.sql.Connection" %>
<%@ page import="java.sql.DriverManager" %>
<%@ page import="java.sql.SQLException"%>
<%@ page import="java.sql.PreparedStatement"%>
<!--Se inicia la construcción del archivo HTML-->
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Generacion Listado de Estudiantes</title>
</head>
<body>
<!--Se inicia la codificación en Java, inicia con '< %' y termina con '% >' --> <%

//Inicia con la declaración de los objetos que se utilizarán
//en este caso para la conexión y los cursores de datos
Connection conn = null;

PreparedStatement ps = null;
ResultSet rs = null;
String SQL = null;
try{

//instancia la conexión del driver
Class.forName("com.mysql.jdbc.Driver").newInstance();

conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/dbUniversidad", "egomez","12345");

NUESTRA PÁGINA LISTADOGENERAL.JSP
conn.setAutoCommit(false);

//establece la sentencia SQL que utilizará para obtener los datos.
SQL = "select *from tbestudiantes";

//vincula nuestro comando con la conexión abierta y el SQL de la consulta.
//observe que el ResultSet es de sólo consulta.
ps = conn.prepareStatement(SQL, ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);

//ejecuta nuestro ResultSet, obteniendo el resultado.
rs = ps.executeQuery();

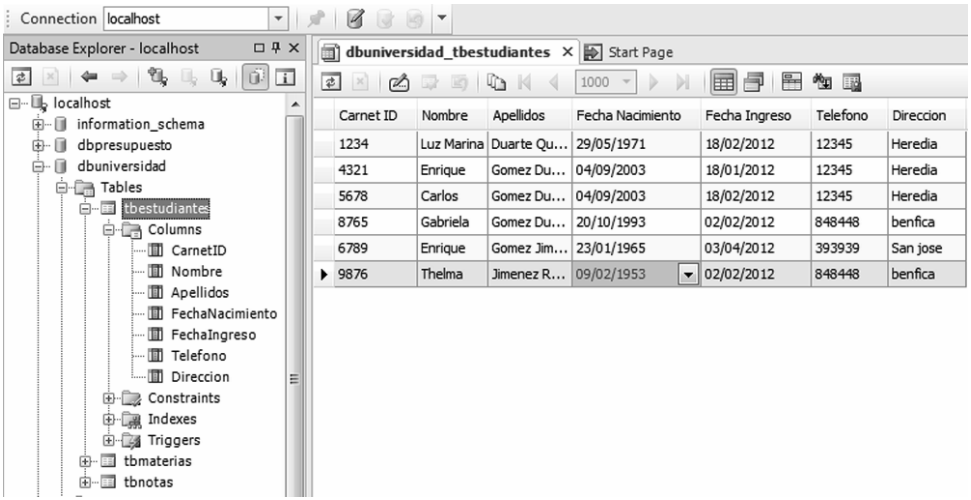
//confirmar la acción de la consulta en la tabla, según la conexión abierta
conn.commit();

//se inicia la presentación de los datos según los resultados obtenidos.
out.println(" Lista de Estudiantes ");

out.println("<table border=1>");
out.println("<tr>");
out.println("<td>Carnet</td>");
out.println("<td>Nombre</td>");
out.println("<td>Apellidos</td>");
out.println("<td>Fecha Nacimiento</td>");
out.println("<td>Fecha Ingreso</td>");
out.println("<td>Telefono</td>");
out.println("<td>Direccion</td>");
out.println("</tr>");
//se recorre el ResultSet y se cargan los datos en cada fila de la tabla while(rs.next()) {
out.println("<tr>");
out.println("<td>" + rs.getString("CarnetID") + "</td>");
out.println("<td>" + rs.getString("Nombre") + "</td>");
out.println("<td>" + rs.getString("Apellidos") + "</td>");
out.println("<td>" + rs.getDate("FechaNacimiento") + "</td>");
out.println("<td>" + rs.getDate("FechaIngreso") + "</td>");

NUESTRA PÁGINA LISTADOGENERAL.JSP
out.println("<td>" + rs.getString("Telefono") + "</td>");
out.println("<td>" + rs.getString("Direccion") + "</td>");
out.println("</tr>");
} // end while
out.println("</table>");
} catch (SQLException exQL) {
out.println(exQL.toString());
} catch (Exception ex) {
out.println(ex.toString());
} finally {
//se cierran los objetos abiertos if (rs != null) rs.close();
if (ps != null) ps.close();
if (conn != null) conn.close();
} // end try
//fin del código Java %>
</body>
</html>

4. Una vez escrito el código anterior se puede ejecutar nuestra primera página web que consulta una tabla MySQL. Si se realiza una consulta mediante nuestro front-end (dbForge) se obtendría una lista de nuestros registros, tal como se muestra en la figura 10.2.



Carnet ID	Nombre	Apellidos	Fecha Nacimiento	Fecha Ingreso	Telefono	Direccion
1234	Luz Marina	Duarte Qu...	29/05/1971	18/02/2012	12345	Heredia
4321	Enrique	Gomez Du...	04/09/2003	18/01/2012	12345	Heredia
5678	Carlos	Gomez Du...	04/09/2003	18/02/2012	12345	Heredia
8765	Gabriela	Gomez Du...	20/10/1993	02/02/2012	848448	benfica
6789	Enrique	Gomez Jim...	23/01/1965	03/04/2012	393939	San jose
9876	Thelma	Jimenez R...	09/02/1953	02/02/2012	848448	benfica

Figura 10.2 Generar un listado mediante dbForge

5. En forma análoga ahora se ejecuta nuestro JSP *listadoGeneral.jsp*, nos mostrará la misma información, tal y como se muestra en la figura 10.3.

Carnet	Nombre	Apellidos	Fecha Nacimiento	Fecha Ingreso	Telefono	Direccion
1234	Luz Marina	Duarte Quintanilla	1971-05-29	2012-02-18	12345	Heredia
4321	Enrique	Gomez Duarte	2003-09-04	2012-01-18	12345	Heredia
5678	Carlos	Gomez Duarte	2003-09-04	2012-02-18	12345	Heredia
6789	Enrique	Gomez Jimenez	1965-01-23	2012-04-03	393939	San jose
8765	Gabriela	Gomez Duarte	1993-10-20	2012-02-02	848448	benfica
9876	Thelma	Jimenez Reyes	1953-02-09	2012-02-02	848448	benfica

Figura 10.3 Generar un listado mediante listadoGeneral.jsp.

Cabe anotar que en el ejemplo anterior todo el código HTML como Java reside en un único archivo JSP, por lo cual mucho código podría ser creado aparte, con lo cual se podría reutilizar.

GENERAR UN LISTADO GENERAL DE DATOS MEDIANTE JSP Y EL USO DE CLASES

El siguiente ejercicio pretende la utilización de un modelo de datos (mediante la clase *Estudiante*) y un modelo de negocios (mediante la clase *ProcesosEstudiante*). En la primera se construye el modelo que establece los atributos:

1. Lo primero que se hace será crear la clase *Estudiante*. Por tanto, pulse con el botón derecho del mouse sobre el nombre del proyecto, pulse la opción *Nuevo* y luego *Java Class…* Póngale el nombre *Estudiante* y escriba el siguiente código en el editor que se muestra.

NUESTRA CLASE ESTUDIANTE
package clases; import java.util.Date;
public class Estudiante {
private String carnet;
private String nombre;
private String apellidos;
private String telefono;
private String direccion;
private Date fechaNac;
private Date fechaIng;
public String getCarnet() {return carnet;}
public String getNombre() {return nombre;}
public String getApellidos() {return apellidos;}
public Date getFechaNac() {return fechaNac;}
public Date getFechaIng() {return fechaIng;}
public String getTelefono() {return telefono;}
public String getDireccion() {return direccion;}
public void setCarnet(String carnet) {this.carnet = carnet;}
public void setNombre(String nombre) {this.nombre = nombre;}
public void setApellidos(String apellidos) {this.apellidos = apellidos;}

NUESTRA CLASE ESTUDIANTE
public void setFechaNac(Date fechaNac) {this.fechaNac = fechaNac;}
public void setFechaIng(Date fechaIng) {this.fechaIng = fechaIng;}
public void setTelefono(String telefono) {this.telefono = telefono;}
public void setDireccion(String direccion) {this.direccion = direccion;}
}

No se explicó nada de la tabla anterior dado que el código está claramente delimitado por la declaración de atributos y las propiedades públicas que permiten aceptar o devolver los valores contenidos en cada atributo.

2. A continuación se procede a crear la clase *ProcesosEstudiante*, con la siguiente codificación:

NUESTRA CLASE PROCESOSESTUDIANTE
package clases;
import com.mysql.jdbc.PreparedStatement;
import com.mysql.jdbc.Statement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.LinkedList;
public class ProcesosEstudiante {
static String SQL;
static Connection conn = null;
static PreparedStatement ps = null;
public static LinkedList<Estudiante> getEstudiante(String sqlCad)
{

//LinkedList es una lista ligada, que en este caso crea los nodos basados //en la estructura dada por la clase estudiante. Es decir, cada nodo
//estará compuesto por los atributos definidos en la clase estudiante.
LinkedList<Estudiante> listaEstudiantes=new LinkedList<Estudiante>();

try {

NUESTRA CLASE PROCESOSESTUDIANTE

//se establece la instancia de la conexión Java.
Class.forName("com.mysql.jdbc.Driver").newInstance();

conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/dbUniversidad", "egomez","12345");

//Para desactivar el modo autocommit (setAutoCommit(false)) en la conexión
//Activa o desactiva las modificaciones de base de datos de auto-commiting
conn.setAutoCommit(false);

//Una Prepared Statement es una sentencia SQL de base de datos //precompilada.
ps = (PreparedStatement) conn.prepareStatement(sqlCad, ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);

//Se ejecuta la consulta, generando el ResultSet. Fíjese en la instrucción //executeQuery
//que significa que la consulta devuelve datos, en lugar de sólo ejecutarse como //en el caso de sólo execute
ResultSet rs = ps.executeQuery();

//se recorre el ResultSet o cursor rs y se cargan los datos en la lista ligada o //enlazada
while (rs.next()){
//se crea una instancia de la clase estudiante. Estudiante estudiante = new Estudiante();

//se carga cada nodo de la lista ligada con los datos del ResultSet
estudiante.setCarnet(rs.getString("CarnetID"));

estudiante.setNombre(rs.getString("Nombre"));
estudiante.setApellidos(rs.getString("Apellidos"));
estudiante.setFechaNac(rs.getDate("FechaNacimiento"));
estudiante.setFechaIng(rs.getDate("FechaIngreso"));
estudiante.setTelefono(rs.getString("Telefono"));
estudiante.setDireccion(rs.getString("Direccion"));

//luego de agregados los datos a cada atributo del nodo, éste se agrega a la lista //enlazada
listaEstudiantes.add(estudiante);

NUESTRA CLASE PROCESOSESTUDIANTE
}

//se cierran los objetos abiertos
ps.close();

rs.close();
conn.close();
}
catch (Exception e)
{ }
//returna la lista enlazada con la información de los estudiantes, almacenada en sus nodos.
return listaEstudiantes; }

//Este procedimiento se utiliza para actualizar la base de datos, es decir,
//para insertar, modificar o eliminar registros de la tabla tbestudiantes.
public static String actualizar(String SQL) throws ClassNotFoundException,
InstantiationException, SQLException, IllegalAccessException

{
String mensaje;
try {
Class.forName("com.mysql.jdbc.Driver").newInstance();
conn= DriverManager.getConnection("jdbc:mysql://localhost:3306/dbUniversidad", "egomez","12345");
Statement st = (Statement) conn.createStatement();

//se ejecuta el cursor (Statement) sin devolución alguna de datos (sólo para //Insert, Update o Delete)
st.execute(SQL);

mensaje = "Proceso exitoso!!!";
}
catch (Exception e) {
mensaje = e.getMessage();}

NUESTRA CLASE PROCESOSESTUDIANTE
//retorna un mensaje de proceso exitoso o de alguna excepción producida. return mensaje;
}
}

Observe que algunas líneas están previamente documentadas, lo que significa que son parte fundamental del código.

3. Ahora cree una nueva página web JSP. Se nombrará *listadoGeneral_Clase*. El código es el siguiente:

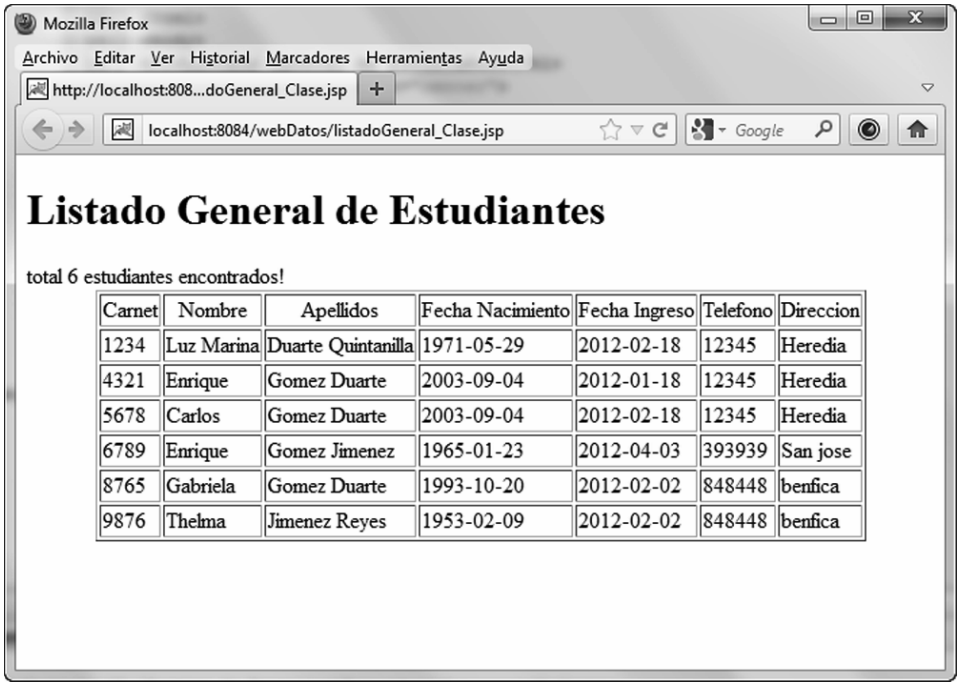
CÓDIGO DE LISTADOGENERAL_CLASE
<!-Se cargan los imports que se van a utilizar en la clase. Debe tomarse en consideración que se importa el proceso *ProcesosEstudiante* donde radica la lógica de procesamiento (consulta y actualización de datos) y también la clase *Estudiante* que se utilizará para formatear los datos que se almacenarán temporalmente en la lista enlazada-> <%@page import="clases.ProcesosEstudiante"%>
<%@page import="clases.Estudiante"%>
<%@page import="java.util.LinkedList"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<body>
Listado General de Estudiantes
<!—No se recomienda el uso de align=”center” por no cumplir con las recomendaciones de HTML 4. Debe ser manejado mediante CSS. En este caso se utiliza por cuanto lo que se requiere es demostrar la funcionalidad del código JSP --> <table border="1" align="center">
<tr> <td align="center">Carnet</td>
<td align="center">Nombre</td>
<td align="center">Apellidos</td>
<td align="center">Fecha Nacimiento</td>

CÓDIGO DE LISTADOGENERAL_CLASE
<td align="center">Fecha Ingreso</td>
<td align="center">Direccion</td>
<td align="center">Telefono</td>
</tr>
<%
String SQL = "select *from tbestudiantes";

//se crea la lista enlazada donde se almacenarán temporalmente los datos
//que obtenemos de la tabla tbestudiantes.
LinkedList<Estudiante> lista = ProcesosEstudiante.getEstudiante(SQL);

out.println("<align=\"center\">");
for (int i=0;i<lista.size();i++){
out.println("<tr>");
out.println("<td>"+lista.get(i).getCarnet()+"</td>");
out.println("<td>"+lista.get(i).getNombre()+"</td>");
out.println("<td>"+lista.get(i).getApellidos()+"</td>");
out.println("<td>"+lista.get(i).getFechaNac()+"</td>");
out.println("<td>"+lista.get(i).getFechaIng()+"</td>");
out.println("<td>"+lista.get(i).getTelefono()+"</td>");
out.println("<td>"+lista.get(i).getDireccion()+"</td>");
out.println("</tr>");}
out.println("total "+lista.size()+" estudiantes encontrados!");
%>
</table>
</body>
</html>

4. Ahora, si se ejecuta esta página se obtendrá una interface similar a la que se muestra en la figura 10.4.



Carnet	Nombre	Apellidos	Fecha Nacimiento	Fecha Ingreso	Telefono	Direccion
1234	Luz Marina	Duarte Quintanilla	1971-05-29	2012-02-18	12345	Heredia
4321	Enrique	Gomez Duarte	2003-09-04	2012-01-18	12345	Heredia
5678	Carlos	Gomez Duarte	2003-09-04	2012-02-18	12345	Heredia
6789	Enrique	Gomez Jimenez	1965-01-23	2012-04-03	393939	San jose
8765	Gabriela	Gomez Duarte	1993-10-20	2012-02-02	848448	benfica
9876	Thelma	Jimenez Reyes	1953-02-09	2012-02-02	848448	benfica

Figura 10.4 Generar un listado mediante listadoGeneral_Clase.jsp.

CREAR UNA PÁGINA JSP QUE INSERTE DATOS

El siguiente ejercicio pretende reutilizar la clase *Estudiante* y el modelo de negocios de la clase *ProcesosEstudiante* para crear unas páginas JSP que permitan el ingreso de datos en la tabla *tbestudiantes*. El proceso que se requerirá para realizar esto se resume a continuación:

- Creará una página denominada *pagInsercion.jsp* que se encargará de la captura de los datos a incluir en la tabla *tbestudiantes*. Esta página configurará el método *GET*. Cabe recordar que el método GET se utiliza para obtener datos desde el servidor y POST para enviar datos. Sin embargo, GET permite agregar datos de un formulario mediante una llamada (un query, por ejemplo) de una URL.
- Creará una página *pagInsertar.jsp* que es la responsable de ejecutar el código en el servidor, invocando las clases necesarias *Estudiante* y *ProcesoEstudiante* y luego devolverá el control a *pagInsercion.jsp* mediante una referencia.

La figura 10.5 muestra, esquemáticamente, cómo sería la ejecución de las páginas que harán posible la inserción de registros en la tabla *tbestudiantes*.

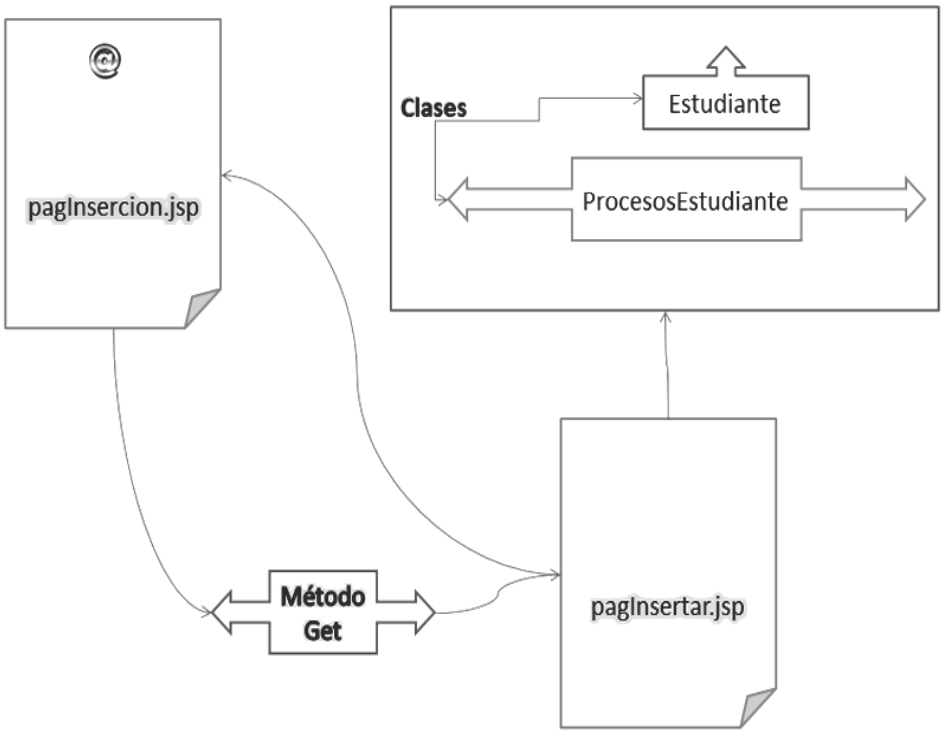


Figura 10.5 Flujo de ejecución para la inserción de registros.

5. Una vez que se sabe cómo se ejecutará ese procedimiento, escriba el código de la página jsp *pagInsercion*. Dicho código es el siguiente:

CÓDIGO DE PAGINSERCIONJSP
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<body>
<html>
Inclusion de Estudiante
<%--Se llama a la página pagInsertar.jsp mediante GET, con el objetivo de que ésta realice el procedimiento de incluir el registro --%> <form method="get" action="pagInsertar.jsp">
<label for="carnet">Carnet del estudiante : </label>

CÓDIGO DE PAGINSERCIONJSP
<input type="text" size="10" width="20" name="carnet">
<label for=“nombre”>Nombre del estudiante :</label>
<input type="text" name="nombre" width="20" value="" />
  <label for=“apellidos”>Apellidos del estudiante : </label>
<input type="text" size="40" width="20" name="apellidos" value="" />
<label for=“fechaNacimiento”>Fecha de Nacimiento: </label>
<input type="text" size="10" width="20" name="fechanacimiento" value="" />
<label for=“fechaIngreso”>Fecha de Ingreso: </label>
<input type="text" size="10" width="20" name="fechaingreso" value="" />
<label for=“telefono”>Telefono del estudiante : </label>
<input type="text" size="10" width="20" name="telefono" value="" />
<label for=“direccion”>Direccion del estudiante : </label>
<textarea name="direccion" rows="4" cols="20">
</textarea>
<hr>
<input type="submit" value="Incluir">
</form>
</body>
</html>

6. Ahora escriba el código para la página *pagInsertar.jsp*. El código es el siguiente:

CÓDIGO DE PAGINSERTAR.JSP
<%@page import="java.util.Locale"%>
<%@page import="java.text.DateFormat"%>
<%@page import="java.util.Date"%>
<%@page import="java.text.SimpleDateFormat"%>
<%@page import="clases.ProcesosEstudiante"%>
<%@page import="clases.Estudiante"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<body>
Datos registrados <table border="1">
<tr>
<td>Carnet</td>
<td>Nombre</td>
<td>Apellidos</td>
<td>Direccion</td>
<td>Telefono</td>
<td>Fecha de Nacimiento</td>
<td>Fecha de Ingreso</td>
</tr>
<%

//se obtienen los parámetros de la página pagInsercion.
//observe que carnet es el nombre del objeto Entrada de Texto
//así como los demas objetos de esa página.

String pCarnet = request.getParameter("carnet");
String pNombre = request.getParameter("nombre");
String pApellidos = request.getParameter("apellidos");
String pTelefono = request.getParameter("telefono");
String pDireccion = request.getParameter("direccion");

CÓDIGO DE PAGINSERTAR.JSP
String fecha1 = request.getParameter("fechanacimiento");
String fecha2 = request.getParameter("fechaingreso");
//se realiza una conversión de fechas, de tipo año-mes-dia para poderla //almacenar en MySQL
String fechaNac = fecha1.substring(6,10)+"-"+fecha1.substring(3,5)+ "-"+fecha1.substring(0,2);
String fechaIng = fecha2.substring(6,10)+"-"+fecha2.substring(3,5)+ "-"+fecha2.substring(0,2);
//se realiza la inserción de datos en la tabla tbestudiantes. String SQL = "insert into tbestudiantes (CarnetID,Nombre,Apellidos,FechaNacimiento,FechaIngreso,Telefono,Direccion) values('"+pCarnet+"','"+pNombre+"','"+pApellidos+"','"+ fechaNac+"','"+fechaIng+"','"+pTelefono+"','"+pDireccion+"')";
//se invoca al método actualizar de la clase ProcesosEstudiante, enviándole la //cadena SQL de la inserción. String proceso = ProcesosEstudiante.actualizar(SQL);
//se muestran los datos ingresados.
out.println("<tr>");
out.println("<td>"+pCarnet+"</td>");
out.println("<td>"+pNombre+"</td>");
out.println("<td>"+pApellidos+"</td>");
out.println("<td>"+pTelefono+"</td>");
out.println("<td>"+pDireccion+"</td>");
out.println("<td>"+fecha1+"</td>");
out.println("<td>"+fecha2+"</td>");
out.println("</tr>");
%>
</table>
<h3>
<%

CÓDIGO DE PAGINSERTAR.JSP
out.println(proceso);
%>
</h3>
<%-- Se muestran las opciones de regresar al proceso de inserción.--%>
<input type="submit" value="Regresar" onclick="location.href = 'pagInsercion.jsp'"/>
</body>
</html>

7. Si ejecuta la página *pagInsercion.jsp* (basta con seleccionarla, pulsar el botón derecho del mouse y seleccionar *Ejecutar archivo*) y se mostrará una pantalla como la que aparece en la figura 10.6.

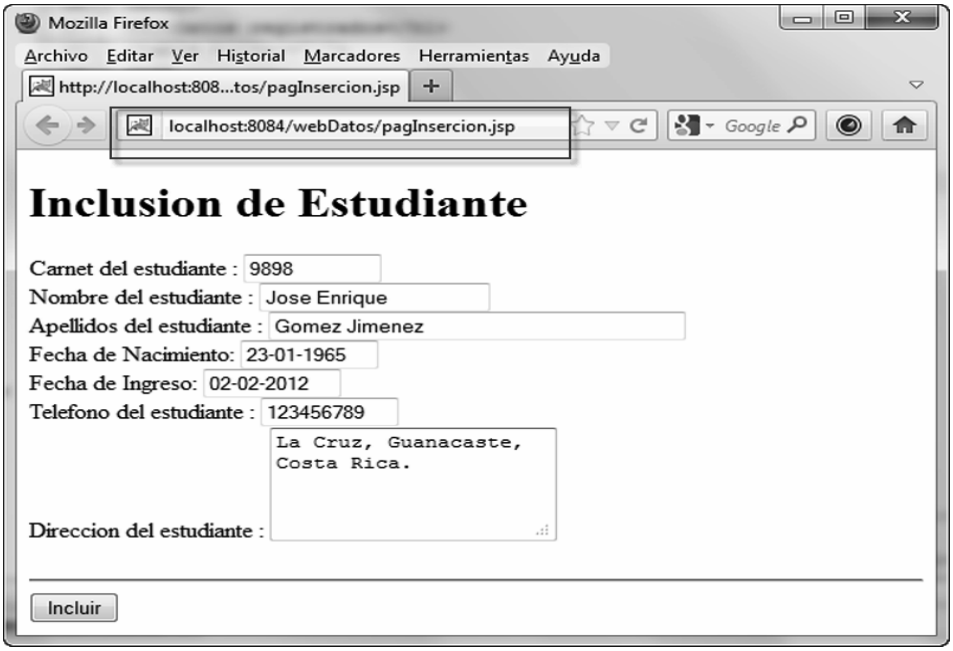


Figura 10.6 Ejecutar la página pagInsercion.jsp.

8. Y si pulsó en esta página el botón *Incluir* nos llevará directamente a la página *pagInsertar.jsp*, que nos mostraría algo similar a la figura 10.7.

Figura 10.7 Ejecutar la página pagInsertar.jsp.

Observe en el recuadro de la figura 10.6 sobre la ejecución de la página *pagInserción.jsp* y 10.7 *pagInsertar.jsp*. En esta última se puede notar los parámetros que se le pasan a la página, derivados de la página *pagInsercion.jsp.*

9. Ahora puede verificar en la tabla *tbestudiantes* que el registro efectivamente se haya ingresado, tal como se muestra en la figura 10.8.

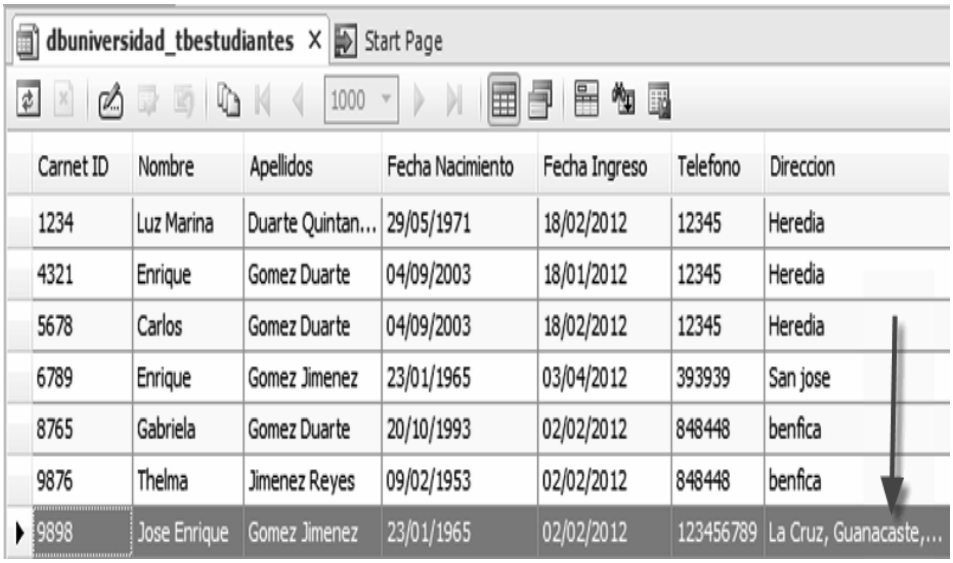


Figura 10.8 Verificar la inclusión del registro.

CREAR UNA PÁGINA JSP QUE MODIFIQUE DATOS

El siguiente ejercicio pretende reutilizar la clase *Estudiante* y el modelo de negocios de la clase *ProcesosEstudiante* para crear unas páginas JSP que permitan la modificación de datos en la tabla *tbestudiantes*. El proceso que se requerirá para realizar esto se resume a continuación:

- Cree una página denominada *pagModificacion.jsp* que se encargará de la captura del número de carnet que servirá para buscar el registro respectivo en la tabla *tbestudiantes*. Esta página configurará el método *GET*.
- Cree la página *pagDatosModificados.jsp* que es la responsable de capturar los nuevos datos que reemplazarán el registro encontrado. Utilizará la funcionalidad de *ProcesosEstudiante* para realizar la búsqueda del registro y remitir a la página *pagModificado.jsp* que será la encargada de gestionar la modificación respectiva. La figura 10.9 muestra el flujo de este proceso.

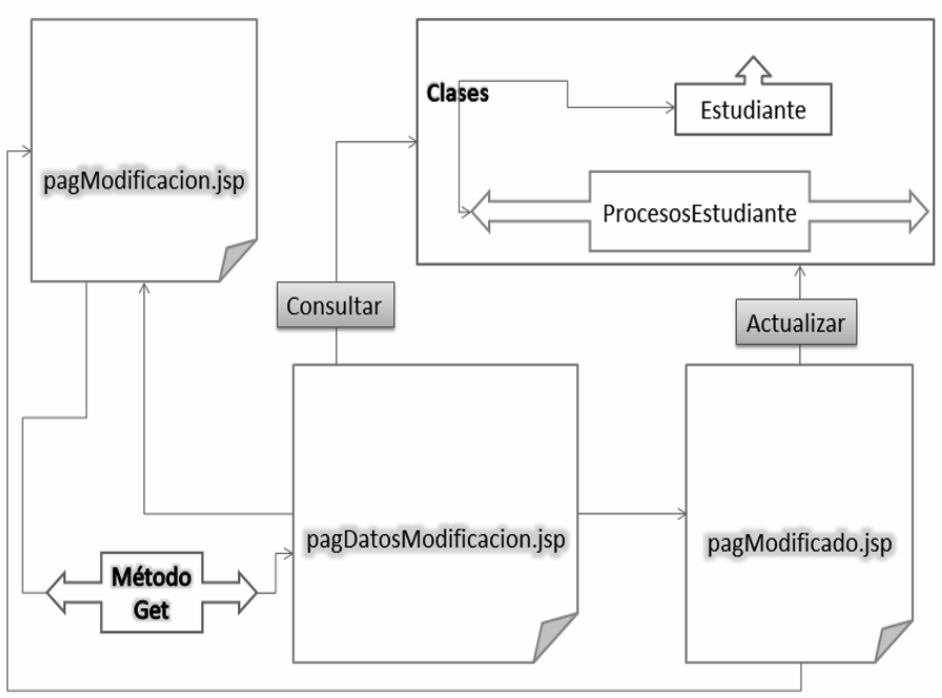


Figura 10.9 Flujo del proceso de modificación de un registro.

10. Ahora proceda a la codificación de estas páginas. Inicie con la página *pagModificacion.jsp* (ya sabe cómo crear una nueva página JSP). El código de esta página es el siguiente:

CÓDIGO DE PAGMODIFICACION.JSP
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<body>
Consulta de Estudiante
<form method="get" action="pagDatosModificacion.jsp">
<label for="campo1">Carnet a Buscar :</label>
<input type="text" size="10" name="carnetBuscar">
<hr>
<input type="submit" value="Consultar">
</form>
</body>
</html>

11. Ahora continúe con el código de la página *pagDatosModificacion.jsp*, tal como se observa en la línea <form method="get" action="pag DatosModificacion.jsp"> del código anterior (*pagModificacion.jsp* hace el llamado de esa página). El código respectivo es el siguiente:

CÓDIGO DE PAGDATOSMODIFICACION.JSP
<%@page import="clases.ProcesosEstudiante"%>
<%@page import="clases.Estudiante"%>
<%@page import="java.util.LinkedList"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<body>
Datos del estudiante a modificar
<body>
Inclusion de Estudiante
<form method="get" action="pagModificado.jsp">
<hr>

CÓDIGO DE PAGDATOSMODIFICACION.JSP
<table border="1">
<tr>
<td>Carnet</td>
<td>Nombre</td>
<td>Apellidos</td> <td>Direccion</td>
<td>Telefono</td>
</tr>
<%

//prepara la consulta que se realizará a la base de datos, según el registro que
//se desea modificar.
String SQL = "select *from tbestudiantes where CarnetID="+
request.getParameter("carnetBuscar");

LinkedList<Estudiante> lista = ProcesosEstudiante.getEstudiante(SQL);
for (int i=0;i<lista.size();i++){
out.println("<tr>");
out.println("<td>"+lista.get(i).getCarnet()+"</td>");
out.println("<td>"+lista.get(i).getNombre()+"</td>");
out.println("<td>"+lista.get(i).getApellidos()+"</td>");
out.println("<td>"+lista.get(i).getDireccion()+"</td>");
out.println("<td>"+lista.get(i).getTelefono()+"</td>");
out.println("<td>"+lista.get(i).getFechaNac()+"</td>");
out.println("<td>"+lista.get(i).getFechaIng()+"</td>");

//las siguientes dos líneas habilitan el uso de variables de sesión y crean una
//variable de nombre carnet cuyo valor es tomado de lista.get(i).getCarnet()
HttpSession sesion=request.getSession();

sesion.setAttribute("carnet",lista.get(i).getCarnet());
out.println("</tr>");
}
%>

CÓDIGO DE PAGDATOSMODIFICACION.JSP
</table>
Datos de modificacion
<hr>
<label for=“nombre”>Nombre del estudiante :</label>
<input type="text" name="txtnombre" width="20" value="" />
<label for=“apellidos”>Apellidos del estudiante : </label>
<input type="text" size="40" width="20" name="txtapellidos" value="" />
<label for=“fechaNacimiento”>Fecha de Nacimiento: </label>
<input type="text" size="10" width="20" name="txtfechanacimiento" value="" />
<label for=“fechaIngreso”>Fecha de Ingreso: </label>
<input type="text" size="10" width="20" name="txtfechaingreso" value="" />
  <label for=“telefono”>Telefono del estudiante : </label>
<input type="text" size="10" width="20" name="txttelefono" value="" />
<label for=“direccion”>Direccion del estudiante : </label>
<textarea name="txtdireccion" rows="4" cols="20">
</textarea>
<input type="submit" value="Regresar" onclick="location.href = 'pagModificacion.jsp'"/>
<input type="submit" value="Modificar" onClick="location.href = 'pagModificado.jsp' ">
</form>
</body>
</html>

12. Finalmente, escriba el código para la página *pagModificado.jsp*, que es donde se realiza realmente la modificación del registro. El código es el siguiente:

CÓDIGO DE PAGMODIFICADO.JSP
<%@page import="java.util.Locale"%>
<%@page import="java.text.DateFormat"%>
<%@page import="java.util.Date"%>
<%@page import="java.text.SimpleDateFormat"%>
<%@page import="clases.ProcesosEstudiante"%>
<%@page import="clases.Estudiante"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<body>
Los nuevos datos
<table border="1">
<tr>
<td>Carnet</td>
<td>Nombre</td>
<td>Apellidos</td>
<td>Direccion</td>
<td>Telefono</td>
<td>Fecha de Nacimiento</td>
<td>Fecha de Ingreso</td>
</tr>
<%
HttpSession sesion=request.getSession();

CÓDIGO DE PAGMODIFICADO.JSP
String Carnet=(String)sesion.getAttribute("carnet");
String Nombre = request.getParameter("txtnombre");
String Apellidos = request.getParameter("txtapellidos");
String Telefono = request.getParameter("txttelefono");
String Direccion = request.getParameter("txtdireccion");
String Fecha1 = request.getParameter("txtfechanacimiento");
String Fecha2 = request.getParameter("txtfechaingreso");
String fechaNac = Fecha1.substring(6,10)+"-"+Fecha1.substring(3,5)+ "-"+Fecha1.substring(0,2);
String fechaIng = Fecha2.substring(6,10)+"-"+Fecha2.substring(3,5)+ "-"+Fecha2.substring(0,2);
String SQL = "Update tbestudiantes set Nombre='"+ Nombre+"',Apellidos='"+Apellidos+"',FechaNacimiento='"+ fechaNac+"',FechaIngreso='"+fechaIng+"',Telefono='"+Telefono+ "',Direccion='"+Direccion+ "' Where CarnetID='"+Carnet+"'";
//ejecuta la operación actualizar que se encuentra programada en //ProcesosEstudiante
String proceso = ProcesosEstudiante.actualizar(SQL);
out.println("<tr>");
out.println("<td>"+Carnet+"</td>");
out.println("<td>"+Nombre+"</td>");
out.println("<td>"+Apellidos+"</td>");
out.println("<td>"+Telefono+"</td>");
out.println("<td>"+Direccion+"</td>");
out.println("<td>"+fechaNac+"</td>");

CÓDIGO DE PAGMODIFICADO.JSP
out.println("<td>"+fechaIng+"</td>");
out.println("</tr>");
%>
</table>
<h3>
<%
out.println(proceso);
%>
</h3>
<input type="submit" value="Regresar" onclick="location.href = 'pagModificacion.jsp'"/>
</body>
</html>

13. Si ejecuta la página *pagModificacion.jsp* se dará cuenta que la misma pide un número de carnet para buscar el registro que desea modificar. Vea la figura 10.10.

Figura 10.10 Búsqueda de un registro para ser modificado.

14. Una vez que pulse el botón *Consultar* se ejecutará la página *pagDatosModificacion.jsp*, tal como se observa en la figura 10.10. Observe que en la línea del URL se muestra cuál es el valor de la variable *carnetBuscar*.

Figura 10.11 Modificar el registro.

15. Si se pulsa el botón *Regresar* en esta página se ejecutará de nuevo la página *pagModificacion.jsp* (figura 10.10) y si por el contrario, se pulsa el botón *Modificar* se ejecutará la página *pagModificado.jsp* (figura 10.12).

Actividades para el lector

- Observe en la tabla *Modificación del Estudiante* que en las columnas de la tabla no aparecen los títulos de Fecha de Nacimiento ni Fecha de Ingreso; proceda a corregir el archivo HTML para que se desplieguen adecuadamente.
- Observe que las fechas de nacimiento y de ingreso no aparecen en el formato tradicional dd-MM-yyyy. Proceda a realizar el formateo de estos datos para que la visualización sea la correcta.

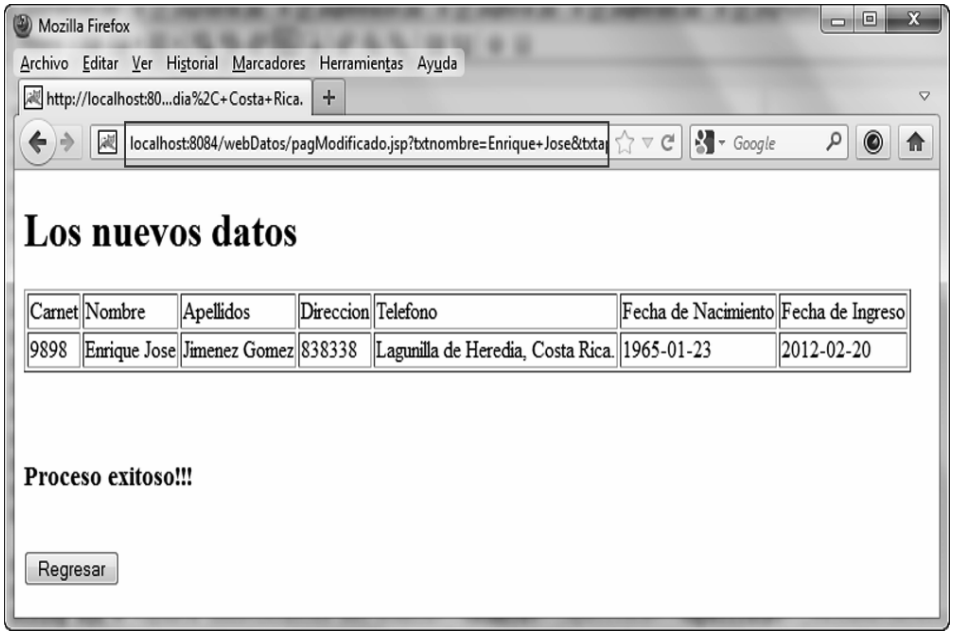


Figura 10.12 El registro ya modificado.

Observe cómo se modificó el registro. Puede también verificar en la base de datos si dicha modificación fue realmente realizada.

Actividades para el lector

- Observe que en la columna Dirección aparece el teléfono del estudiante y en la de Telefono la dirección respectiva. Proceda a corregirlo en el HTML respectivo y realice una nueva modificación de datos donde se muestre la corrección.

- Observe que las fechas de nacimiento y de ingreso no aparecen en el formato tradicional dd-MM-yyyy. Proceda a realizar el formateo de estos datos para que la visualización sea la correcta.

CREAR UNA PÁGINA JSP PARA CONSULTA DE DATOS ESPECÍFICOS

Cree una página JSP que permita consultar un registro específico. Con ello puede buscar y encontrar un registro. Realice el siguiente procedimiento

1. Inicie creando una página JSP de nombre *pagConsultarEstudiante.jsp*. El código es el siguiente:

CÓDIGO DE PAGCONSULTARESTUDIANTE.JSP
<%@page import="clases.ProcesosEstudiante"%>
<%@page import="clases.Estudiante"%>
<%@page import="java.util.LinkedList"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<body>
Consulta de Estudiante
<form method="get" action="pagDespliegue.jsp">
<label for="campo1">Carnet a Buscar :</label> <input type="text" size="10" name="carnetBuscar">
<hr>
<input type="submit" value="Consultar">
</form>
</body>
</html>

2. Como habra notado, la página anterior crea una referencia a otra página web denominada pagDespliegue.jsp que es donde se muestran los datos de la búsqueda. El código para esta página es el siguiente:

CÓDIGO DE PAGDESPLIEGUE.JSP
<%@page import="clases.ProcesosEstudiante"%>
<%@page import="clases.Estudiante"%>
<%@page import="java.util.LinkedList"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<body>
Seleccion de Estudiantes
<input type="text" name="txtCarnet" value="" />
<table border="1">
<tr>
<td>Carnet</td>
<td>Nombre</td>
<td>Apellidos</td>
<td>Direccion</td>
<td>Telefono</td>
</tr>
<%
String SQL = "select *from tbestudiantes where CarnetID=" + request.getParameter("carnetBuscar");
LinkedList<Estudiante> lista = ProcesosEstudiante.getEstudiante(SQL);
for (int i=0;i<lista.size();i++){
out.println("<tr>");
out.println("<td>"+lista.get(i).getCarnet()+"</td>");
out.println("<td>"+lista.get(i).getNombre()+"</td>");

CÓDIGO DE PAGDESPLIEGUE.JSP
out.println("<td>"+lista.get(i).getApellidos()+"</td>");
out.println("<td>"+lista.get(i).getDireccion()+"</td>");
out.println("<td>"+lista.get(i).getTelefono()+"</td>");
out.println("</tr>");
}
out.println("total "+lista.size()+" estudiantes");
%>
</table>
<input type="submit" value="Regresar" onclick="location.href = 'pagInsercion.jsp'"/>
</body>
</html>

3. Si ejecuta la página pagConsultarEstudiante.jsp se mostrará una pantalla similar a la figura 10.10.
4. Si se pulsa el botón Consultar se mostrará el resultado de la consulta. En caso que seleccione el carnet 9898, se visualizará una pantalla similar a la de la figura 10.13

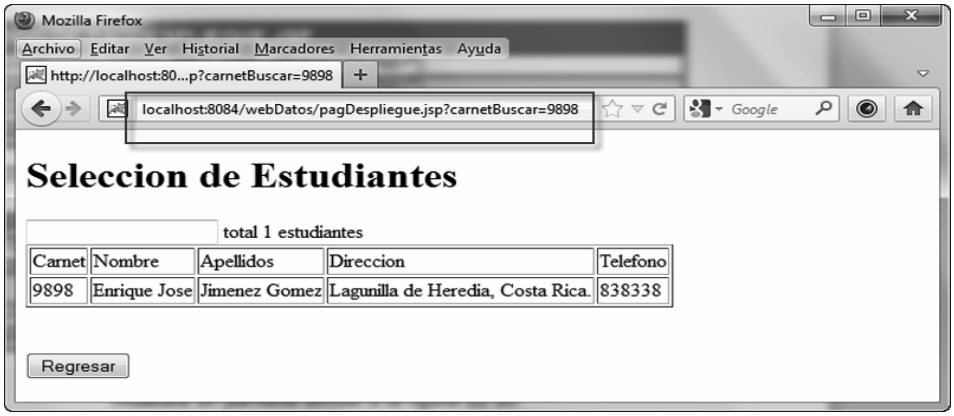


Figura 10.13 Visualizar un registro determinado.

CREAR UNA PÁGINA JSP QUE ELIMINE DATOS

Para finalizar, creará un proceso de eliminación de registros. El proceso que se requerirá para realizar esto se resume a continuación:

1. Creará una página denominada *pagEliminacion.jsp* que se encargará de la captura del número de carnet que servirá para buscar el registro respectivo en la tabla *tbestudiantes*. Esta página configurará el método *GET*. Es similar a lo que se muestra en la figura 10.10. El código de esta página es el siguiente:

CÓDIGO DE PAGELIMINACION.JSP
<%@page import="clases.ProcesosEstudiante"%>
<%@page import="clases.Estudiante"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<body>
Borrado de Estudiante
<form method="get" action="pagBorrar.jsp">
<label for=“carnet”>Digite el carnet del estudiante a eliminar : </label>
<input type="text" size="10" width="20" name="carnet">
<hr>
<input type="submit" value="Buscar dato">
</form>
</body>
</html>

2. Observe cómo esta página hace un llamado a la página *pagBorrar.jsp* mediante el botón *Buscar dato*. Si el usuario pulsa ese botón se consumará la acción de buscar el registro mediante una función que se encuentra en *ProcesosEstudiante*. Una vez pulsado ese botón aparecerá una pantalla similar a la figura 10.14, que es la ejecución de la línea *<form method="get" action="pagBorrar.jsp">* Es decir, se ejecuta la visualización del formulario *pagBorrar.*

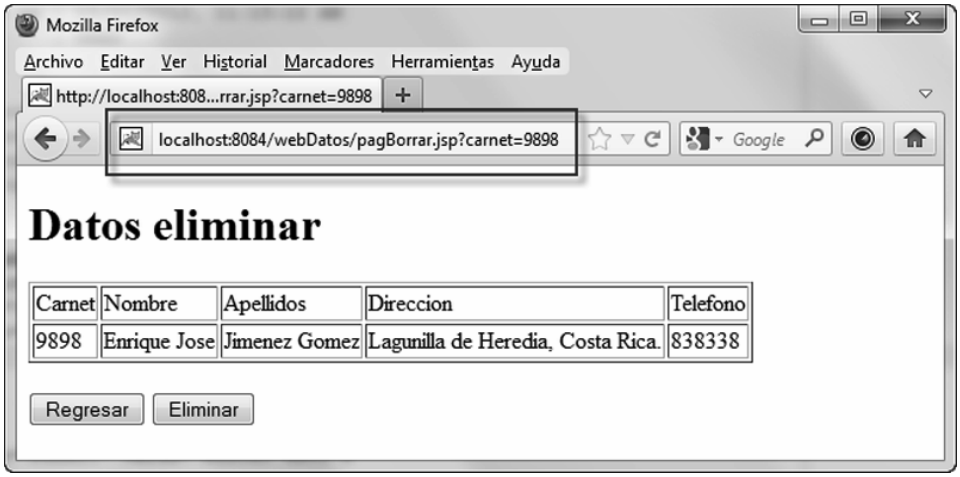


Figura 10.14 Visualizar un registro a eliminar.

3. Observe que existen dos botónes en esta página (*Regresar* y *Eliminar*) cuyo código está implementado en esta página (*pagBorrar.jsp*). Dicho código es el siguiente:

CÓDIGO DE PAGBORRAR.JSP
<%@page import="javax.servlet.http.HttpSession"%>
<%@page import="java.util.LinkedList"%>
<%@page import="java.util.Locale"%>
<%@page import="java.text.DateFormat"%>
<%@page import="java.util.Date"%>
<%@page import="java.text.SimpleDateFormat"%>
<%@page import="clases.ProcesosEstudiante"%>
<%@page import="clases.Estudiante"%>

CÓDIGO DE PAGBORRAR.JSP
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<body>
Datos eliminar
<table border="1">
<tr>
<td>Carnet</td>
<td>Nombre</td>
<td>Apellidos</td>
<td>Direccion</td>
<td>Telefono</td>
</tr>
<%
String SQL = "select *from tbestudiantes where CarnetID="+ request.getParameter("carnet");
LinkedList<Estudiante> lista = ProcesosEstudiante.getEstudiante(SQL);
for (int i=0;i<lista.size();i++){
out.println("<tr>");
out.println("<td>"+lista.get(i).getCarnet()+"</td>");
out.println("<td>"+lista.get(i).getNombre()+"</td>");
out.println("<td>"+lista.get(i).getApellidos()+"</td>");
out.println("<td>"+lista.get(i).getDireccion()+"</td>");
out.println("<td>"+lista.get(i).getTelefono()+"</td>");
out.println("</tr>");

CÓDIGO DE PAGBORRAR.JSP
HttpSession sesion=request.getSession();
//guarda en la variable sesión nombre el nombre y apellidos del usuario sesion.setAttribute("nombre",lista.get(i).getNombre()+" "+lista.get(i).getApellidos());

//guarda en una variable denominada carnet sesión los datos del carnet.
sesion.setAttribute("carnet",lista.get(i).getCarnet());

}
%>
</table>
<input type="submit" value="Regresar" onclick="location.href = 'pagEliminacion.jsp'"/>
<input type="submit" value="Eliminar" onClick="location.href = 'pagBorrado.jsp' "/>
</body>
</html>

4. Observe que al final del código se presentan dos botones *submit*. El primero es por si el usuario se arrepiente del borrado y decide regresar a la página anterior, y el segundo es para confirmar la eliminación. Si el caso es que el usuario pulsa el segundo botón, entonces se ejecutará la eliminación y se visualizará entonces una pantalla como la que muestra la figura 10.15.

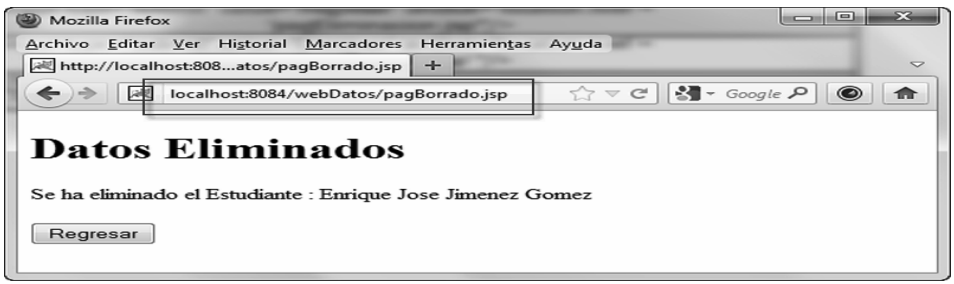


Figura 10.15 Pantalla de confirmación de borrado.

El código de la página *pagBorrado.jsp* es el que se muestra a continuación.

CÓDIGO DE PAGBORRADO.JSP
<%@page import="java.util.Locale"%>
<%@page import="java.text.DateFormat"%>
<%@page import="java.util.Date"%>
<%@page import="java.text.SimpleDateFormat"%>
<%@page import="clases.ProcesosEstudiante"%>
<%@page import="clases.Estudiante"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page import="java.text.SimpleDateFormat"%>
<!DOCTYPE html>
<html>
<body>
<h1>Datos Eliminados</h1
<h2>
<%
//habilita el uso de sesiones HttpSession sesion=request.getSession();
//obtiene las sesiones nombre y carnet almacenadas en pagBorrar.jsp String Nombre=(String)sesion.getAttribute("nombre");
String Carnet=(String)sesion.getAttribute("carnet");
out.println("Se ha eliminado el Estudiante : "+Nombre);
String SQL = "delete from tbestudiantes where CarnetID ='"+ Carnet + "'";
//invoca el proceso actualizar para realizar el borrado del registro. String proceso = ProcesosEstudiante.actualizar(SQL);
%>

CÓDIGO DE PAGBORRADO.JSP
</h2>
<input type="submit" value="Regresar" onclick="location.href = 'pagEliminacion.jsp'"/>
</body>
</html>

5. Si pulsa el botón *Regresar* podrá volver a digitar el carnet, en este caso 9898, y se dará cuenta que ya no existe, puesto que la eliminación se llevó a cabo exitosamente. Es decir, el sistema se volverá a la página *pagEliminacion.jsp*.

Con este ejercicio llega al final de este capítulo. Ha desarrollado, en forma muy básica, el manejo de datos con JSP. No se considera el uso de CSS en este capítulo, dado que el objetivo era demostrar cómo se producia ese manejo y no tanto el formateo de los datos (la presentación).

Actividades para el lector

- Desarrolle una aplicación web que permita el mantenimiento de la tabla *tbnotas*, considerando los registros de los estudiantes.
- Cree un archivo CSS y formatee la presentación de información en las páginas de mantenimiento de datos que acabamos de realizar.

RESUMEN

En este capítulo se desarrolló el tema de la gestión de datos, utilizando JSP y MySQL. Básicamente, se desarrollaron los siguientes temas:

1. Cómo generar listados de datos utilizando sólo un archivo JSP.
2. Cómo generar listados de datos mediante JSP y el uso de clases.
3. Cómo crear programas de mantenimiento y consulta de datos, utilizando JSP y MySQL.

EVIDENCIA

☐ Elaboró resúmenes que le permitan comprender adecuadamente la funcionalidad del uso de una página web para la consulta y mantenimiento de datos.

☐ Desarrolló ejemplos alternativos que incluyen operaciones de gestión de datos utilizando JSP y MySQL.

Autoevaluación

1. ¿Para qué sirve la instrucción Class.forName("com.mysql.jdbc.Driver").newInstance() utilizada en este capítulo?
2. ¿Cuál es la diferencia entre ResultSet rst.executeQuery(); y ResultSet rst.execute(SQL)?
3. En el capítulo, ¿para qué se usa la siguiente instrucción: %@page import="clases.ProcesosEstudiante"%?
4. En el capítulo, ¿para qué se usa la siguiente expresión: String pNombre = request.getParameter("nombre");?
5. ¿Para qué se utilizan las siguientes instrucciones?: HttpSession sesion=request.getSession(); sesion.setAttribute("carnet",lista.get(i).getCarnet());
6. ¿Para qué se utiliza en el capítulo la siguiente instrucción: String Nombre = request.getParameter("txtnombre");?
7. ¿Para qué se utiliza en el capítulo la siguiente instrucción: <form method="get" action="pagDespliegue.jsp">?
8. ¿Para qué utilizó en el capítulo la instrucción: %@page import="java.util.LinkedList"%?
9. ¿Para qué utilizó en el capítulo la instrucción <input type="submit" value="Regresar" onclick="location.href = 'pagInsercion.jsp'"/>?
10. ¿Para qué utilizó en el capítulo la instrucción: LinkedList<Estudiante> lista = ProcesosEstudiante.getEstudiante(SQL);?

REFERENCIAS

Bibliografía

Anil Hemrajani (2006) Agile Java Development with Spring, Hibernate and Eclipse. Sarns Publishing, Library of Congress, New York, USA.

Ceballos, Fco. Javier, (2011). Java 2: Curso De Programación, 4a. ed., Alfaomega, México.

Mak, Gary (2010) Spring Recipes. A Problem-Solution Approach. Apress Edition, New York (USA)

Martín, Antonio, (2010). Programador certificado Java 2. Curso práctico, 3a. ed., Alfaomega, México.

Rod Johnson (2005) Professional Java Development with the Spring Framework. Wiley Publishing inc., Indianapolis, USA.

Rozannski, Uwe, (2010). Enterprise Javabeans 3.0. Con Eclipse Y JBoss. Alfaomega, México.

Seth Ladd (2006) Expert Spring MVC and Web Flows. Library of Congress, New York, USA.

Sznajdleder, Pablo, (2010) Java a fondo

Páginas Web recomendadas

1. chuidiang (2011) *Uso de PreparedStatement con Java y MySQL*. Obtenido el 2 de diciembre del 2011 desde http://www.chuidiang.com/java/mysql/PreparedStatement-java-mysql.php
2. Chacín, Carlos (2009) *Jugando con PreparedStatement*. Obtenido el 2 de diciembre del 2011 desde http://cchacin.blogspot.com/2009/07/jugando-con-preparedstatement.html

Respuestas a la autoevaluación

1. Se utiliza para crear una instancia de conexión a la base de datos MySQL
2. Que la primera ejecuta una consulta y por tanto devuelve un resultado (conjunto de datos vacío o con información) y la segunda solamente ejecuta un comando sql (por ejemplo una inserción, una actualización o una eliminación).
3. Sirve para establecer una referencia a las funciones que se encuentran en la clase ProcesosEstudiante.
4. Para almacenar en una cadena de nombre pNombre el contenido de un parámetro, que en este caso es el valor que se digitó en un objeto denominado nombre (que puede ser una caja de texto de una página JSP).
5. Se utiliza para habilitar el uso de sesiones y almacenar en una sesión de nombre carnet el contenido de la variable lista.get(i).getCarnet.
6. Para obtener el valor de la variable de sesión txtNombre y almacenarla temporalmente en Nombre.
7. Se utiliza para que si se pulsa el botón submit se proceda a la ejecución de la página JSP *pagDespliegue.jsp*.
8. Se usa para habilitar el uso de listas ligadas mediante LinkedList.
9. Permite ejecutar la pagina *pagInsercion.jsp* en caso de que se pulse el botón *Regresar*.
10. Para almacenar en cada nodo de una lista enlazada la información de acuerdo con los atributos creados en la clase Persona, de acuerdo con la ejecución de alguno de los procesos contenidos en la clase *ProcesosEstudiante*.

Spring Web MVC

11

Reflexione y responda las siguientes preguntas

¿Qué tanto conoce de la aplicabilidad del Modelo Vista Controlador *MVC*?

¿Qué herramientas se puede utilizar con el *MVC*?

¿Qué relación se puede establecer entre Spring web *MVC* y Struts?

¿Se puede afirmar que con *MVC* se apuesta a la verdadera reutilización de código y componentes?

Contenido

EXPECTATIVA

En el capítulo anterior se describió el Modelo Vista Controlador (MVC), así como la forma en que mediante toda una lógica de reutilización de código y componentes establece una capa controladora que nos permite determinar hacia dónde dirigir la funcionalidad de nuestro sitio web.

En el presente capítulo se podrá hacer uso de una herramienta para desarrollar una aplicación web que utilice patrones de diseño, tal como lo es MVC. Para ello se utilizará el framework Spring donde se implementa el patrón MVC. Un aporte importante de este framework es la inyección de independencia, lo que permite que automáticamente este motor se encargue de incluir las dependencias necesarias que requiere la aplicación, eliminando de las clases la creación de objetos.

Como desarrollador debe tener presente la necesidad de utilizar patrones de diseño en las creaciones web que realice. A través de controladores puede establecerse algunas reglas entre modelos y vistas para obtener un sistema totalmente modular pero que trabaja perfectamente sincronizado. Por tanto, nos interesará comprender cómo se implementa el modelo MVC en una aplicación Java utilizando Spring web MVC.

Después de estudiar este capítulo, el lector será capaz de

- Comprender el funcionamiento de Spring web MVC como framework que apoya la aplicabilidad de patrones de diseño.
- Aplicar Spring web MVC en la creación de un sitio web sencillo.

INTRODUCCIÓN

Desarrollar aplicaciones web es y será siempre una tarea compleja. Esto se debe a que, en primer lugar, la seguridad se ve comprometida al estar expuesta a un número importante de usuarios en la red. En segundo lugar, la usabilidad del sitio y las prestaciones que brinde al usuario final serán factores críticos de éxito en este contexto. En tercer lugar, el deber irrenunciable de que el sitio esté todo el tiempo disponible, hace que la robustez de la misma y de la base tecnológica que la soporta (redes, dispositivos, entre otros) sea una necesidad imperante.

Otro aspecto importante es crear una aplicación robusta pero llena de buenas prácticas de desarrollo. Ello se logra poniendo a disposición los patrones de diseño, lo cual ha demostrado a través de los años que se trata de una disciplina obligada en el desarrollo web. Con MVC, que es un patrón de diseño muy conocido, podrá desarrollar aplicaciones web de mucha robustez, además de las buenas prácticas que se ven implícitas en su uso. Spring web MVC es un framework que facilita la implementación de este patrón de diseño en las aplicaciones web y, por ende, es necesario conocer un poco la lógica funcional que está detrás.

SPRING WEB MVC

Spring Web MVC es un framework Open Source que está conformado por un contenedor, el mismo framework (para el manejo de componentes) y un snap-in services (módulos para las interfaces web de usuarios y la persistencia).

Algunas funcionalidades de Spring es la existencia de archivos de configuración Spring XML Bean, el completado automático de código, el sistema de navegación con hiperenlazado, y la refactorización que consiste en el renombrado de referencias a clases java de los archivos XML Beans de Spring. Constituye un patrón arquitectónico que separa los datos de una aplicación, la interfaz de usuario y la lógica de control, todo en capas diferentes. Este patrón MVC se utiliza frecuentemente en aplicaciones web donde las páginas HTML representan la vista, mientras que el código que provee el acceso a datos es el modelo y el controlador la lógica de negocios.

Spring Web MVC se creó con la premisa de facilitar el desarrollo de aplicaciones J2EE mediante buenas prácticas de diseño y programación. En ese sentido, el manejo de patrones de diseño como Factory, Abstract Factory, Builder, Decorator, entre otros, era indispensable de implementar en esa buena práctica. También permite la modularidad en la programación utilizando el Core Container, que es el responsable de la configuración de los objetos de la aplicación; Aspect-Oriented Programming Framework, que permite la reutilización de soluciones, y Data Access Framework, que permite el uso de JDBC, Hibernate, entre otros. La figura 11.1 muestra algunas de las características de Spring Web Framework.

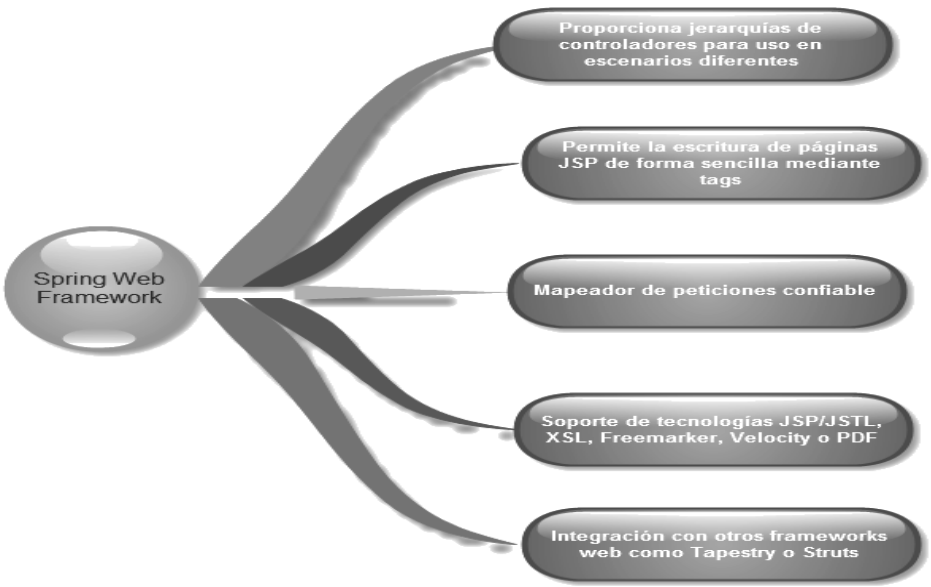


Figura 11.1 Características de Spring Web Framework.

¿Cómo funciona?

El funcionamiento de Spring Web Framework consta de una serie de procedimientos que interactúan desde el momento en que se presenta una solicitud de página (request). En este caso, las peticiones se procesan a través de un Servlet que actúa de *Front Controller* (*DispatchServlet*).

El objeto DispatcherServlet procesa una solicitud de página (request) e invoca al Handler Mapping para saber qué controlador utilizar. Es responsable de resolver la vista adecuada que se mostrará al usuario. Como cualquier otro Bean, puede ser configurado en el *WebApplicationContext*. Un Bean es un artefacto de software reutilizable, lo que representa ahorro de tiempo de desarrollo dado que encapsula varios objetos en uno solo. DispatchServlet utiliza la vista para mostrarle el modelo al usuario.

El *Handler Mapping* establece una lista de pre y post procesos y controladores que deben ser ejecutados para las solicitudes (*request*). Es similar a *ActionMapping* de Struts.

Los controladores permiten procesar las solicitudes (*request*); funciona de manera similar como en Struts con actions.

El *View resolvers*: provocan la asignación de nombres lógicos a las vistas de la aplicación. En Struts se denominan *ActionFormward*. Más adelante se retomará este procedimiento. La figura 11.2 muestra gráficamente este comportamiento y la lógica de realización según la numeración que cada actividad realiza.

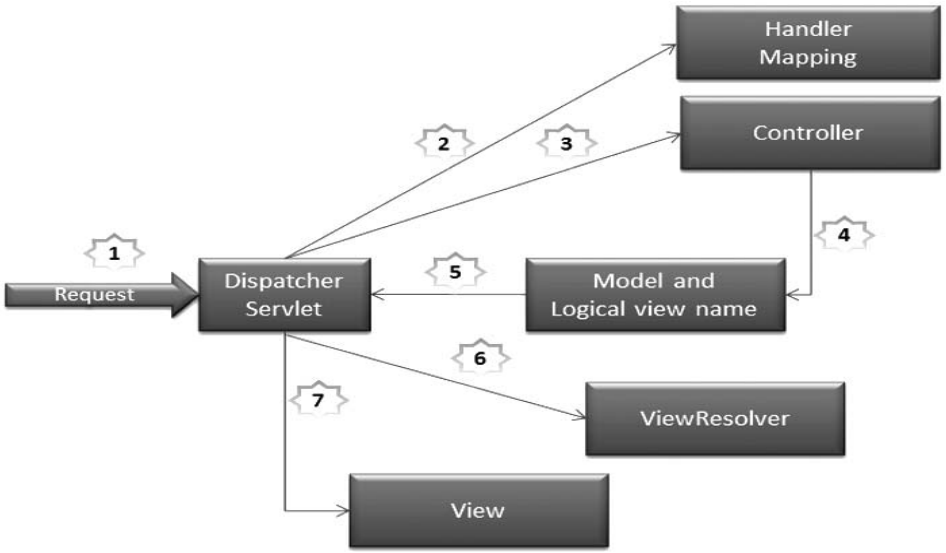


Figura 11.2 Procedimiento para ejecutar solicitudes de página en Spring Web Framework.

CREAR UNA APLICACIÓN SPRING WEB MVC

Ejercicio

Crear una aplicación web que utilice el Spring Web MVC como framework.

1. La lógica de uso de la creación de nuestra aplicación web se orientará por la funcionalidad expuesta en el diagrama de caso de uso que se presenta en la figura 11.3.

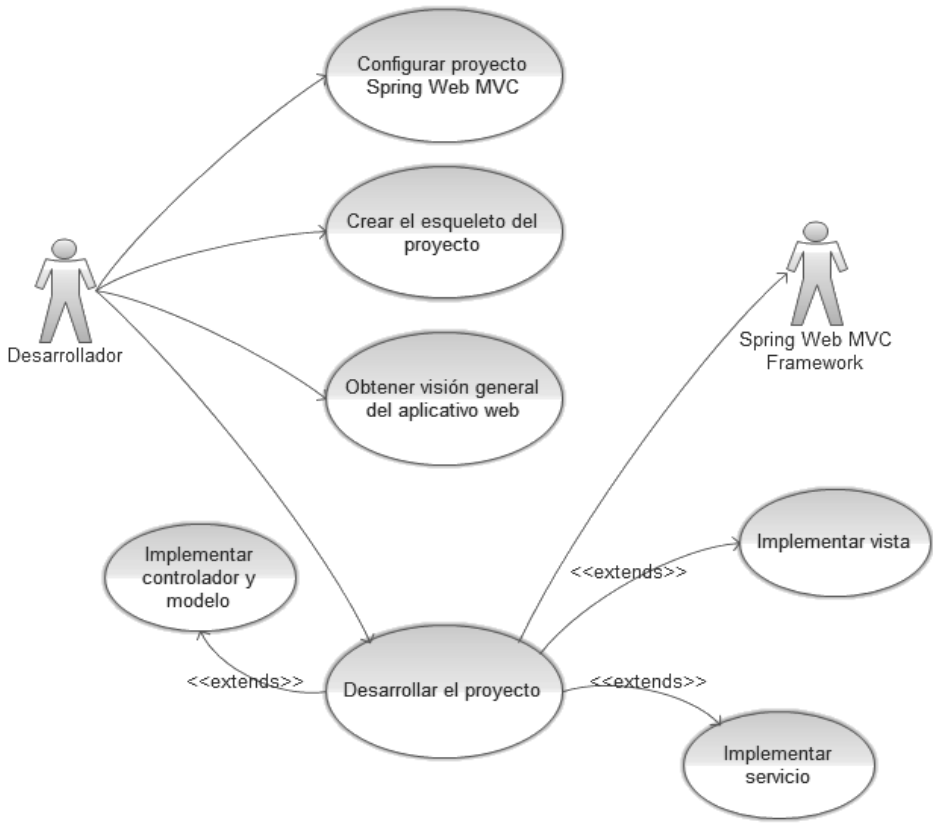


Figura 11.3 Creación de una aplicación web con el framework Spring Web MVC.

Actividades para el lector

Elabore un resumen en forma de esquema que agrupe los principales conceptos del framework Spring web MVC.

Se ha expuesto una síntesis de cómo funciona Spring Web MVC. Para comprender mejor esta funcionalidad se desarrollará una aplicación con este framework.

1. Inicie con un nuevo proyecto NetBeans. El tipo de proyecto a crear es *Java Web* y *Web Application*, tal como se muestra en la figura 11.4.

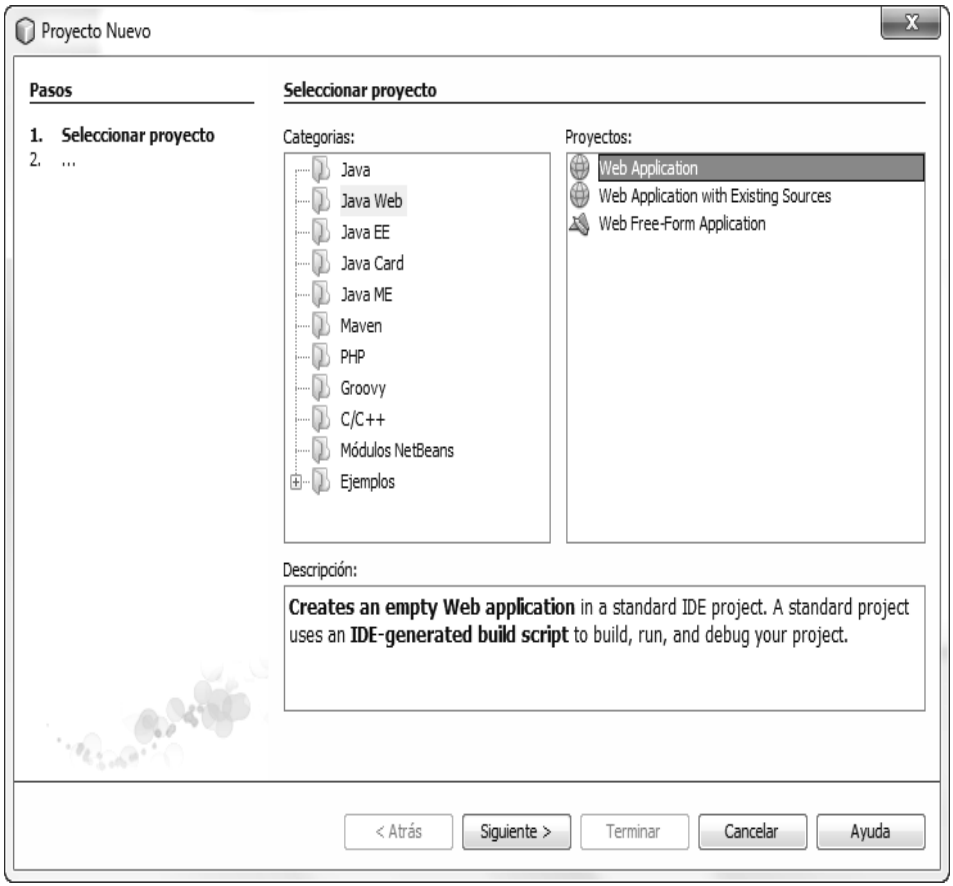


Figura 11.4 Mi primera aplicación web con Spring Web MVC.

2. Pulse en el botón *Siguiente*, según la figura 11.4.
3. En la pantalla que sigue nombre el proyecto como *miPrimeraAplicacionWeb*. Asegúrese que la caja de verificación en *Set as Main Project* esté habilitada. Pulse el botón *Siguiente*.
4. Se muestra a continuación la pantalla que permite seleccionar el servidor que se utilizará; en este caso se tiene habilitado GlassFish, como muestra la figura 11.5.

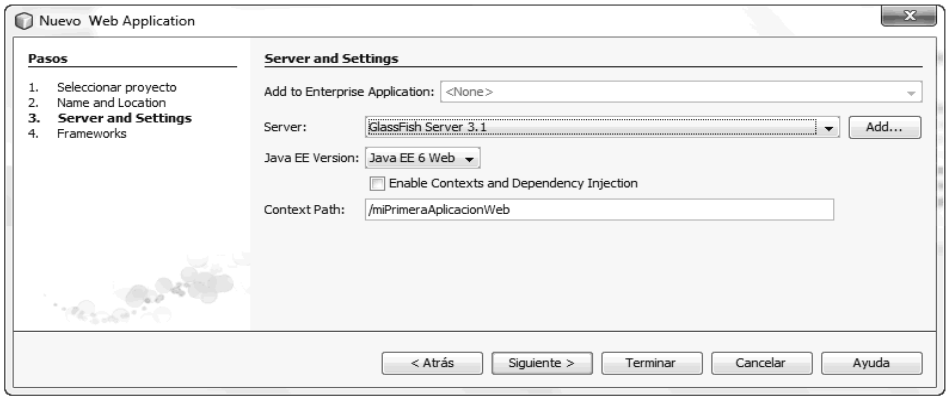


Figura 11.5 Habilitación del servidor de aplicaciones.

Cabe detenerse en este punto, dado que si se pulsa el botón Agregar se puede añadir otro servidor; en este caso se podría establecer Tomcat, versión 7.0. Recuerde que Tomcat funciona como un repositorio de servlets y también puede funcionar como un servidor de internet.

5. En la figura 11.6 se muestran los posibles servidores que se pueden agregar a su aplicación web. Si hubiera necesidad de utilizar Tomcat seleccione el elemento de la lista y luego haga clic en el botón *Siguiente.* Si lo anterior fuera una realidad, en la pantalla siguiente se tendrá que ubicar el directorio donde se instaló Tomcat para habilitarlo, así como un usuario (podría ser root) y una contraseña (la que el administrador del servidor defina). Como no es el caso, este procedimiento se pasa por alto y sólo pulse el botón *Siguiente*, según se aprecia en la figura 11.6.

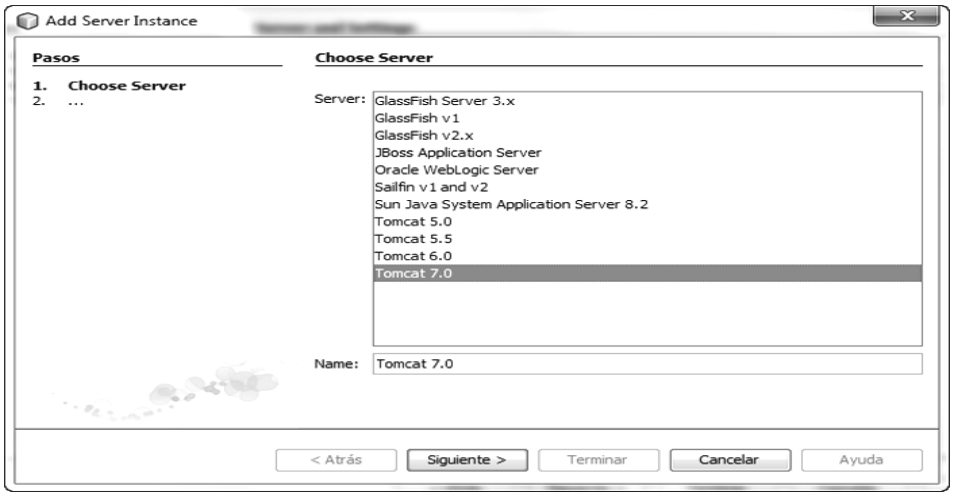


Figura 11.6 Posibles servidores que se pueden agregar a su aplicación web.

6. Una vez que pulse sobre el botón *Siguiente* se mostrará una pantalla como la de la figura 11.7.

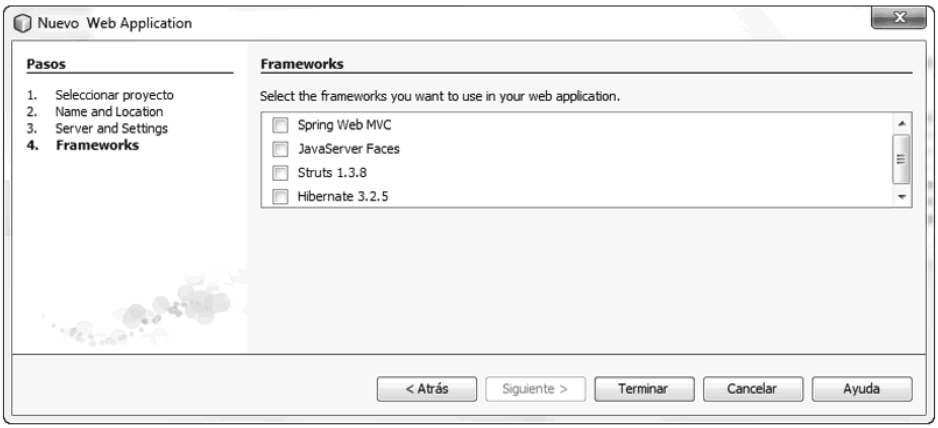


Figura 11.7 Selección del framework que se va a utilizar en nuestra web.

7. Seleccione la opción *Spring Web MVC* y pulse la tecla *Terminar*. Asegúrese de deshabilitar la caja de chequeo *Include JSTL*.

8. Una vez creado el proyecto puede notar que se habilitaron cuatro archivos:

   a) dispatcher-servlet.xml
   b) applicationContext.xml
   c) redirect.jsp
   d) index.jsp

   Respecto de estos archivos ya se ha comentado antes, pero vale la pena hacer otras observaciones. Por ejemplo, si se ejecuta este sitio web, mostrará una interface como la de la figura 11.8.

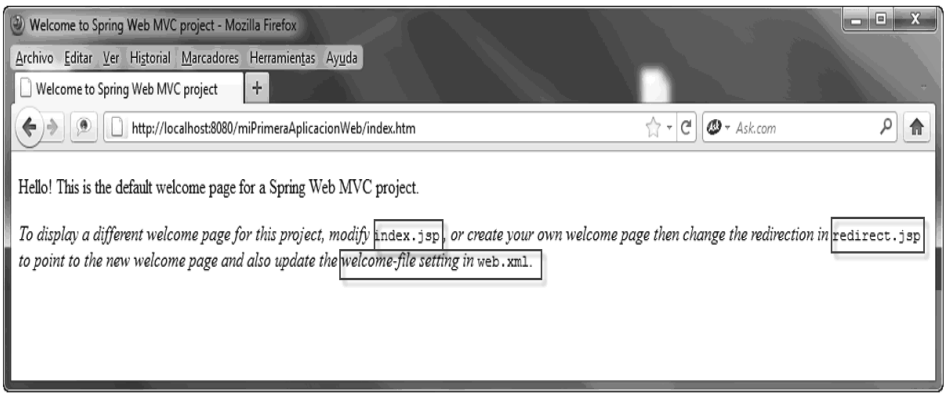


Figura 11.8 Ejecución de nuestro primer sitio Spring Web MVC.

Observe que la página inicial utiliza los archivos anteriormente citados (index.jsp y redirect.jsp) y donde podría cambiar esa configuración. Para un análisis un poco más detallado vea el archivo *web.xml* (posiciónese en el archivo y con un clic derecho sobre él seleccione Editar) y vea el párrafo que se ubica casi al final del archivo XML:

```
<welcome-file-list>
  <welcome-file>redirect.jsp</welcome-file>
```

Observe en la parte sombreada que se indica que el archivo de bienvenida estará configurado en *redirect.jsp*

Nota

Si no puede ver el archivo *web.xml* debe pulsar al mismo tiempo las teclas CTRL+1 o seleccionar *Proyectos* en el menú *Ventana*.

Ahora abra el archivo *redirect.jsp* para determinar a quién mapea este archivo para ser ejecutado por el servlet. Observe al final de este archivo la línea:

```
<% response.sendRedirect("index.htm"); %>
```

Esto indica que el archivo redirecciona (mapea) la ejecución de la página de bienvenida a *index.htm*. Por tanto, esa página es la primera que aparece al ejecutarse la página.

Sigamos con el análisis y volvamos al archivo *web.xml*; observe el mapa del patrón htm que muestra la figura 11.9.

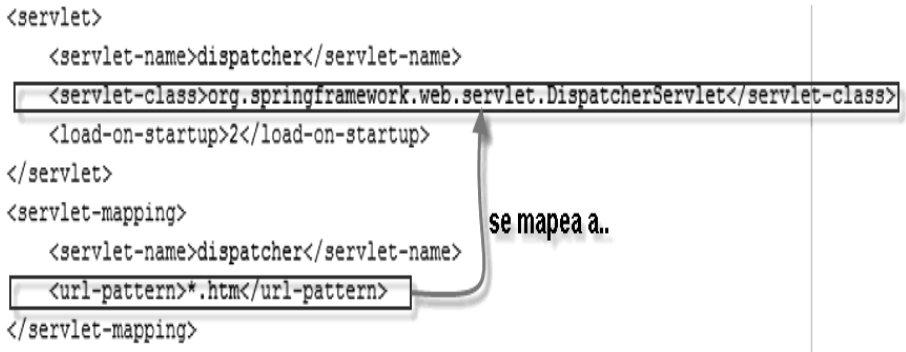


Figura 11.9 Mapa del patrón htm al framework Spring.

Observe que el patrón de las páginas serán *.htm y que se mapean al Spring Framework para su manejo.

Otro archivo importante que se muestra es *dispatcher-servlet.xml*. Este archivo es el encargado de manejar las peticiones de entrada basándose en la configuración que contiene. Este archivo está compuesto por tres beans, los cuales se muestran en la figura 11.10.

```
<bean id="urlMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="mappings">
        <props>
a)          <prop key="index.htm">indexController</prop>
        </props>
    </property>
</bean>

<bean id="viewResolver"
    class="org.springframework.web.servlet.view.InternalResourceViewResolver"
    p:prefix="/WEB-INF/jsp/"
b)  p:suffix=".jsp" />

<!--
The index controller.
-->
<bean name="indexController"
    class="org.springframework.web.servlet.mvc.ParameterizableViewController"
c)  p:viewName="index" />
```

Figura 11.10 Beans del dispatcher-servlet.xml.

El funcionamiento se explica en la figura 11.10. Podría agregar que si el dispatcher-servlet recibe una petición, por ejemplo "index.htm" se busca en este bean el controlador. Observe que la propiedad *key* mapea "index.htm" hacia otro bean denominado *indexController*, que es el nombre del tercer bean (opción c). Este es quien devuelve la vista de la petición mapeada en a) quien la remite a b). El bean viewResolver opción b) resuelve la vista agregando un prefijo "/WEB-INF/jsp/" y un sufijo con extensión jsp. Localiza y ejecuta la página.

Cabe mencionar que dispatcher-servlet.xml es el corazón de Spring MVC y se configura en el archivo web.xml de la aplicación web. Este Framework busca un archivo nombre-servlet (donde nombre es cualquier válido: demo, prueba…) en el directorio WEB-INF.

A este dispatch-servlet se asocia el ContentLoaderListener que es donde se crean las configuraciones adicionales. Todo ello en el web.xml.

Crear la clase Persona

9. Inicie una nueva clase, pulse sobre la opción *Archivo Nuevo* del menú *Archivo*. Seleccione un archivo de tipo *Clase Java* de la opción *Java*. El nombre de la clase será *Persona.* En la opción *paquete* escriba *controlador*. Es decir, el paquete asociado a esta clase será denominada *controlador*.

   Pulse el botón *Finalizar*. Ahí escriba el siguiente código:

EJERCICIO: CREACIÓN DE LA CLASE PERSONA
package controlador; //paquete de la clase
public class Persona {
private String nombre;
private String apellidos;
private String direccion;
private String telefono;
private Integer edad;
public String getNombre() {
return nombre; }
public void setNombre(String nombre) {
this.nombre = nombre; }
public String getApellidos() {
return apellidos;
}
public void setApellidos(String apellidos) {
this.apellidos = apellidos; }
public String getDireccion() {
return direccion; }
public void setDireccion(String direccion) {
this.direccion = direccion; }
public String getTelefono() {
return telefono; }
public void setTelefono(String telefono) {
this.telefono = telefono; }
public Integer getEdad() {
return edad; }
public void setEdad(Integer edad) {
this.edad = edad; }
}

Crear el controlador

10. Una vez creada la clase anterior proceda a crear el controlador. Se utilizará un *Form Controller*. Para ello seleccione *Archivo Nuevo* desde el menú *Archivo* y escoja un formulario *Simple Form* que se encuentra en la categoría *Marco de trabajo de Spring*, como muestra la figura 11.11.

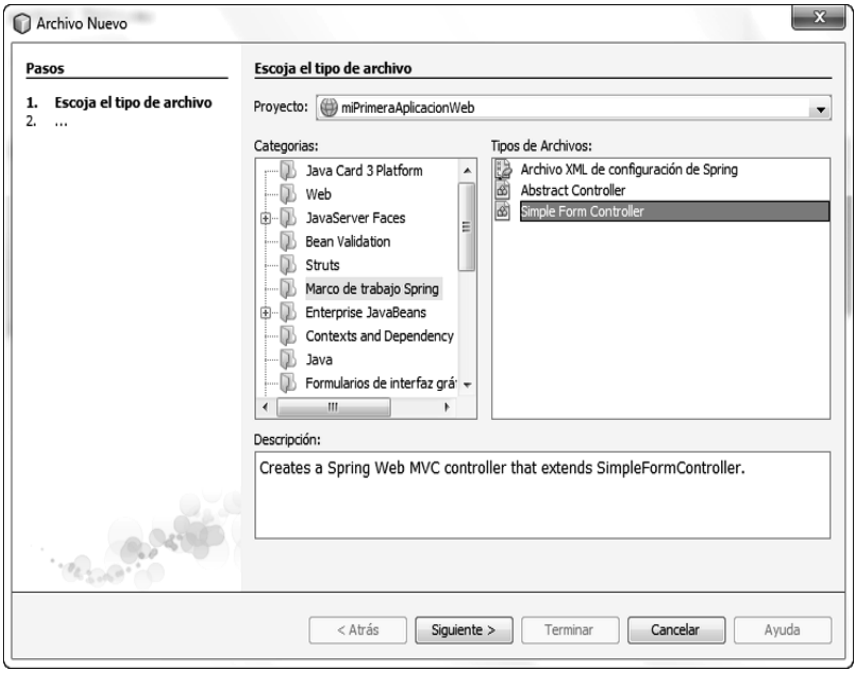


Figura 11.11 Creación de un formulario Simple Form Controller.

11. Nombre este formulario como *controladorPersona.* Asegúrese que el paquete se llame *controlador*, al igual que la clase creada anteriormente. Pulse el botón *Finalizar*.
12. Desmarque algunas instrucciones que inician con *set* para poderlas utilizar en su controlador.
13. El constructor de este formulario quedará como sigue:

EJERCICIO: CONSTRUCTOR DEL FORM CONTROLLER
package controlador; //vinculación al paquete controlador
//se importa el servlet para manejar el MVC import org.springframework.web.servlet.ModelAndView;
//se importa la clase *datosPersonales* del paquete *servicio* que crearemos más //adelante y que servirá para generar los datos a mostrar en la vista.

EJERCICIO: CONSTRUCTOR DEL FORM CONTROLLER
import servicio.datosPersonales;
//se importa la clase SimpleFormController para el manejo del controlador import org.springframework.web.servlet.mvc.SimpleFormController;
public class ControladorPersona extends SimpleFormController {
public ControladorPersona() {//constructor de la clase
setCommandClass(Persona.class); //es el bean vista controlador
setCommandName("egomez"); //nombre del bean. Usamos luego en index.jsp
setSuccessView("respuesta"); //vista para mostrar los datos
setFormView("index"); //configura el nombre de la vista que va a usarse.
}
@Override
protected ModelAndView onSubmit(Object command) throws Exception {
Persona persona= (Persona) command; //casting de command.
ModelAndView mv= new ModelAndView(getSuccessView()); //instancia mvc
mv.addObject("Servidor", datosPersona.imprimeDatos(persona.getNombre(), persona.getApellidos(), persona.getDireccion(),persona.getTelefono(), persona.getEdad())); //agregando datos a la vista a mostrar (servidor)
return mv; //retorna el controlador
}
private servicio.datosPersonales datosPersona;
public void setServiciopersona(datosPersonales datosPersona) {
this.datosPersona = datosPersona;
}
}

14. A continuación proceda a crear la clase *datosPersonales*. Debe hacer clic sobre el nombre del proyecto y seleccionar *Archivo Nuevo* del menú *Archivo* y escoja *Clase Java* de la categoría *Java*. Asegúrese que el controlador que utilizará para esta clase será *servicio*. Observe la figura 11.12.

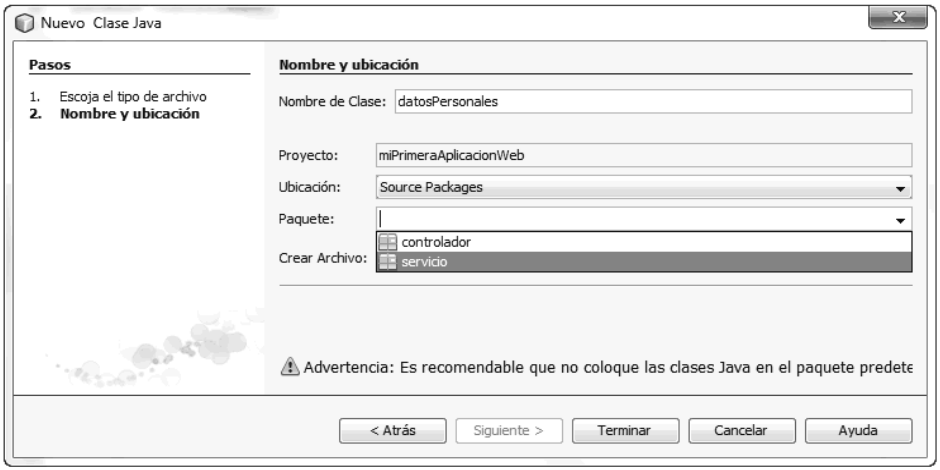


Figura 11.12 Crear la clase datosPersonales, asociándola a su controlador servicio.

15. Pulse el botón *Terminar*, según el formulario de la figura 9.12, y después escriba el código siguiente:

EJERCICIO: CONSTRUCTOR DEL FORM CONTROLLER
package servicio;
public class datosPersonales {
public String imprimeDatos(String nombre,String apellidos, String direccion, String telefono, Integer edad){
String respuesta="";
respuesta="Nombre : " + nombre + " " //br para el salto de línea + "Apellidos : " + apellidos + " " //en HTML + "Dirección : " + direccion + " " + "Teléfono : " + telefono + " " + "Edad : " + edad + " ";
return respuesta;
}
}

16. Con los apartados anteriores ya hemos construido tanto el modelo como el controlador del sistema. Resta solamente crear las vistas. Para ello crearemos dos páginas .jsp que nos servirán para modelar los datos que vamos a presentar y además controlar su presentación.

Crear las vistas

17. Pulse con el botón derecho del mouse sobre el nombre del proyecto y seleccione *JPS* del menú *Nuevo*. Este primer archivo JSP se denominará *respuesta*. Esta página tendrá el siguiente código:

EJERCICIO: CONSTRUCTOR DEL FORM CONTROLLER
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Respuesta desde el SERVIDOR</title> //esta línea debes escribirla.
</head>
<body>
${Servidor} //esta línea debes escribirla. Crea la vista de Servidor //que se encuentra en el controlador
</body>
</html>

Como se puede observar, la instrucción *<h1>${Servidor}</h1>* es la que permite invocar el objeto *mv* que es el que gestiona los datos que se presentarán en la vista, posterior a ser capturados en la página *index.jsp*. Este objeto *mv* recuerda qué está en el controlador *controladorPersona* (puede verse el punto 13 donde lo creamos).

18. El segundo archivo JSP es por default *index.jsp* y será la primera página que ejecutará la aplicación. Recuerde que esta página por default es la que el archivo *dispatcher-servlet* ejecuta por primera vez. En este archivo es donde crearemos la vista de entrada de los datos de usuario. El código en esta página es el siguiente:

EJERCICIO: CONSTRUCTOR DEL FORM CONTROLLER
<%@taglib uri="http://www.springframework.org/tags" prefix="spring" %>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

EJERCICIO: CONSTRUCTOR DEL FORM CONTROLLER
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Datos Personales</title> //esta línea debe quedar así
</head>
<body>
<spring:nestedPath path="egomez"> //nombre del bean que permitirá asociar //los inputText de las entradas, por ejemplo //<spring:bind path="nombre"> lo asocia con //el primer inputText del formulario vista //vía getters y setters en la clase *persona*.
<form action="" method="post">
<table align="center">
<tr>
<td>Nombre</td>
<td>
<spring:bind path="nombre"> //se crea primera variable de entrada
//se configura la variable nombre de tipo input (entrada), texto y se captura valor <input type="text" name="${status.expression}" value="${status.value}" >
</spring:bind>
</td>
</tr>
<tr>
<td>Apellidos</td>
<td>
<spring:bind path="apellidos">
<input type="text" name="${status.expression}" value="${status.value}">

EJERCICIO: CONSTRUCTOR DEL FORM CONTROLLER
</spring:bind>
</td>
</tr>
<tr>
<td>Direccion</td>
<td>
<spring:bind path="direccion">
<input type="text" name="${status.expression}" value="${status.value}">
</spring:bind>
</td>
</tr>
<tr>
<td>Telefono</td>
<td>
<spring:bind path="telefono">
<input type="text" name="${status.expression}" value="${status.value}">
</spring:bind>
</td>
</tr>
<tr>
<td>Edad</td>
<td>
<spring:bind path="edad">
<input type="text" name="${status.expression}" value="${status.value}">
</spring:bind>
</td
</tr>
<tr>
<td></td>

EJERCICIO: CONSTRUCTOR DEL FORM CONTROLLER
<td><input type="submit" value="Ver Registro"></td> //botón de comando
</tr>
</table>
</form>
</spring:nestedPath>
</body>
</html>

19. En el archivo *applicationContext.xml*, en la penúltima fila debemos agregar la siguiente línea de código:

```
<bean name="datosPersonales" class="servicio.datosPersonales"/>
```

datosPersonales es el nombre del bean que permite referenciar la clase datosPersonales que creamos en el paquete servicio (vea el punto 14 de este ejercicio). Cabe aclarar que un bean, en el contexto de Spring, es un objeto creado y gestionado por un contenor Spring. No es una clase como se maneja en el contexto de J2EE, sino que es un objeto. Por tanto, este bean será el que nos permita generar la vista en nuestra página JSP.

Constituye un bean que referencia a la clase *datosPersonales* que creamos en el paquete *servicio*, de donde generamos la vista de datos.

20. En el archivo *dispatcher-servlet.xml* creamos dos beans básicamente:

```
<beanclass="org.springframework.web.servlet.mvc.support.ControllerClassNameHandlerMapping"/>
<bean name="persona"
    class="controlador.ControladorPersona"
    p:serviciopersona-ref="datosPersonales" />
```

Para referenciar a la clase *controladorPersona* que se encuentra en el controlador *controlador* Y.

```
<bean id="urlMapping"
 class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="mappings">
      <props>
        <prop key="index.htm">persona</prop>
      </props>
    </property>
  </bean>
```

El bean que mapea el archivo JPS al bean controlador persona. Observe que <prop key="index.htm">persona</prop> vincula index.htm con persona en el primer bean de la clase Contralador.Persona

21. Con esto finaliza el presente ejercicio de Spring Web MVC. Si ejecuta su aplicación web verá de entrada la primera vista *index.jsp* como muestra la figura 11.13.

Figura 11.13 Vista index.jsp.

22. Y posteriormente la vista *respuesta.jsp* luego de pulsar el botón *Ver Registro*. La figura 11.14 muestra esta página.

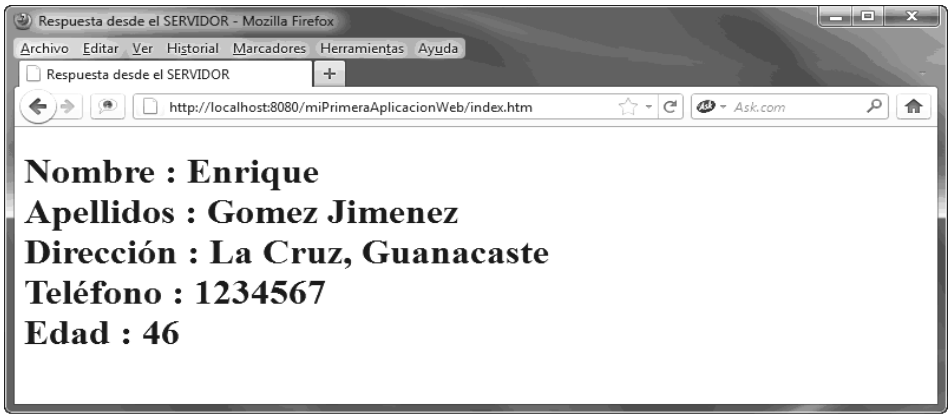


Figura 11.14 Vista respuesta.jsp.

23. Finalmente, se resume el ejercicio Spring Web MVC anterior en la figura 11.15; observe los códigos parciales de cada capa.

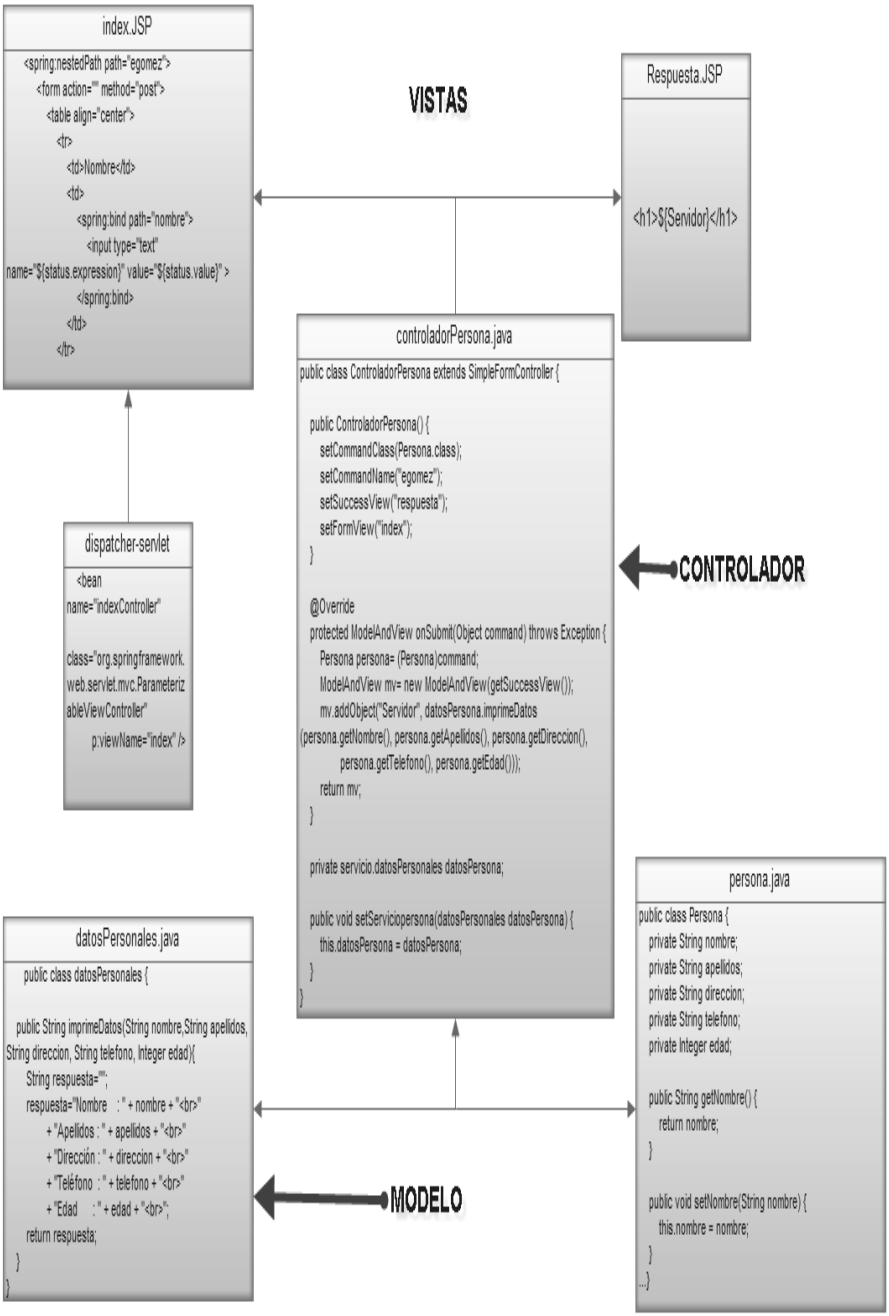


Figura 11.15 Resumen MVC del ejercicio Spring Web MVC.

Actividades para el lector

Cree un sitio web que utilice Spring web MVC, para simular una calculadora que realice las operaciones: suma, resta, multiplicación y división.
Divida el procedimiento en dos vistas:

1. la primera recogerá los datos del usuario.
2. la segunda presentará los datos del resultado de la operación realizada por su calculadora.

RESUMEN

En este capítulo se desarrolló el tema de implementación de un modelo muy conocido y utilizado en patrones de diseño web: MVC. Para ello se empleó el framework Spring web MVC que viene implementado en NetBeans. Fundamentalmente se trató lo siguiente:

1. La arquitectura de Spring MVC que refuerza el tema de patrones de diseño.
2. Cómo crear una aplicación Spring MVC utilizando para ello NetBeans.

Autoevaluación

1. ¿Qué es Spring web MVC?
2. ¿Cuál es el funcionamiento del objeto DispatchServlet?
3. ¿Para qué se utiliza el ViewResolver?

EVIDENCIA

☐ Elaboró en un resumen el esquema que agrupa los principales conceptos del framework Spring web MVC.

☐ Creó un sitio web que utiliza Spring web MVC, el cual simula una calculadora (suma, resta, multiplicación y división) que en la primera vista recoge los datos del usuario y en la segunda presenta los datos del resultado de la operación de la calculadora.

REFERENCIAS

Bibliografía

Anil Hemrajani (2006) Agile Java Development with Spring, Hibernate and Eclipse. Sarns Publishing, Library of Congress, New York, USA.

Ceballos, Fco. Javier, (2011). Java 2: Curso De Programación, 4a. ed., Alfaomega, México.

Mak, Gary (2010) Spring Recipes. A Problem-Solution Approach. Apress Edition, New York (USA)

Martín, Antonio, (2010). Programador certificado Java 2. Curso práctico, 3a. ed., Alfaomega, México.

Rod Johnson (2005) Professional Java Development with the Spring Framework. Wiley Publishing inc., Indianapolis, USA.

Rozannski, Uwe, (2010). Enterprise Javabeans 3.0. Con Eclipse Y JBoss. Alfaomega, México.

Seth Ladd (2006) Expert Spring MVC and Web Flows. Library of Congress, New York, USA.

Sznajdleder, Pablo, (2010) Java a fondo. Estudio del lenguaje y desarrollo de aplicaciones. Alfaomega, México.

Páginas Web recomendadas

1. Spring Framework (2011) *Aplicación Base y Configuración del Entorno.* Obtenido el 8 de diciembre del 2011 desde http://www.davidmarco.es/tutoriales/spring-mvc-sbs/part1.html
2. Spring Framework (2011) *Chapter 13. Web MVC framework.* Obtenido el 8 de diciembre del 2011 desde http://static.springsource.org/spring/docs/2.0.x/reference/mvc.html
3. Risberg, Thomas (2011) *Developing a Spring Framework MVC application step-by-step using NetBeans and GlassFish.* Obtenido el 8 de diciembre del 2011 desde *http://sites.google.com/site/springmvcnetbeans/step-by-step/*

Respuestas a la autoevaluación

1. Es un framework opensource que está conformado por un contenedor, el mismo framework (para el manejo de componentes) y un snap-in services (módulos para las interfaces web de usuario y la persistencia).
2. Se utiliza para procesar una solicitud de página (request) e invoca al HandlerMapping para determinar el controlador que se va a usar para dicho procesamiento.
3. Se emplea para asignar nombres lógicos a las vistas de la aplicación.

JavaServer Faces / ICEFaces en NetBeans 7.1

12

Reflexione y responda las siguientes preguntas

¿Cuánto conoce de desarrollo web con Java?

¿Existe un IDE visual en Java con las prestaciones que ofrecen otras herramientas para el desarrollo web tales como Visual Studio .NET o DreamWeaver?

¿Qué tanta pasión despierta el software libre o el software open Source para el desarrollo de aplicaciones web?

Contenido

EXPECTATIVA

Se ha expuesto algunos temas sobre el desarrollo de aplicaciones web con Java utilizando NetBeans. En el capítulo anterior se usó el modelo MVC a través de Spring web MVC para desarrollar una aplicación que empleara este patrón de diseño. En este capitulo se utilizará los fundamentos de dos framework muy conocidos entre los desarrolladores web en Java: JavaServer Faces e ICEFaces.

Después de estudiar este capítulo, el lector será capaz de:

- Comprender el funcionamiento de ServerFaces como framework en el desarrollo de aplicaciones web.
- Aplicar ServerFaces en la creación de un sitio web sencillo.
- Comprender el funcionamiento de ICEFaces como framework en el desarrollo de aplicaciones web.
- Aplicar ICEFaces en la creación de un sitio web sencillo.

INTRODUCCIÓN

Java quizá sea el lenguaje predilecto de muchos desarrolladores dada la potencia y versatilidad que ofrece. Otros lo preferirán simplemente porque no son adeptos a las herramientas de Microsoft®. Sea cual sea la razón de su uso, es bien sabido que poderosas aplicaciones están realizadas sobre esta plataforma y que hay mucha experiencia sobre su funcionalidad y fortaleza. Desafortunadamente no existen IDES gráficos para las versiones 6.x en adelante (considerando la 7.1x) que aporten una riqueza visual que permitan diseñar rápidamente aplicaciones web. Es cierto que JavaServer Faces tenía un editor Visual de aplicaciones web, sin embargo tal parece que fue descontinuado y no se incluye en esta versión 7.1x. ICEFaces incluye la paleta de controles los cuales deben ser arrastrados a las páginas web que se diseñan en su propio código xhtml, lo cual es un poco confuso para los que trabajan directamente con la edición gráfica del sitio. El interés y la pasión por el software abierto o libre siguen, sin embargo, con mucha fuerza.

JavaServer Faces (JSF) pretende el desarrollo de aplicaciones web de la forma en que lo hacen otras API´S como Java Swing, AWT (Abstract Windows Toolkit), SWT (Standard Widget Toolkit), entre otras. Tradicionalmente, en Java se codifican páginas web a través de JSP (Java Server Pages), con peticiones mediante formularios y respuestas en HTML. Con ello se tiene acceso a datos, así como también el mantenimiento de los mismos

JavaServer Faces proporciona un entorno de trabajo en que las peticiones del usuario se realizan a través de páginas HTML y las procesa mediante eventos que envía al servidor con el fin de obtener una nueva vista de la página (puede ser con menos datos, datos modificados o el acceso a otra página). Esta situación es muy delicada porque podría suponer que existe un procesamiento de servidor importante ante las frecuentes solicitudes que un usuario podría realizar a un sitio web. Esto significaría un ancho de banda importante necesario y quizá una merma en el rendimiento de la aplicación a medida que hayan más usuarios y más solicitudes.

Con el escenario planteado se encuentra que no es como lo pregona AJAX, por ejemplo, de mermar el acceso al servidor para el procesamiento, sino que más bien es darle más tarea al cliente para procesar algún tipo de lógica de negocio y solamente obtener los datos necesarios del servidor. Esto se irá descartando o reafirmando a medida que se desarrolle el tema de JSF.

ICEFaces es el otro framework que se tratará en este capítulo, haciendo hincapié en la funcionalidad AJAX que fundamenta el uso de este framework.

JavaServer Faces (JSF)

JavaServer Faces (JSF) es un framework de interfaces de usuario que trabajan del lado del servidor. Se basa en una API que le permite representar una serie de componentes UI, manejo de estados y eventos, validación desde el lado del servidor, definición de la navegación de páginas web, internacionalización, extensibilidad de características básicas, entre otras. JavaServer Faces es la última tecnología de aplicaciones web Java, que aprovecha la experiencia adquirida en java Servlets, JavaServer Pages, entre otros marcos de trabajo (framework)

Bergsten afirma sobre JSF "*JavaServer Faces simplifica el desarrollo de interfaces de usuario en aplicaciones web complejas mediante la definición de la interfaz del usuario y el modelo de componentes vinculados a la solicitud del mismo en su ciclo de vida*" (Bergsten, 2004). En otras palabras, se creará apenas lo necesario (la interfaz y los componentes necesarios) para el procesamiento de una solicitud de un usuario en el ciclo de vida que esta solicitud tenga.

JavaServer Faces es un framework (igual a Spring Web MVC). Filosóficamente *pretende* la normalización y estandarización del desarrollo web, nutriéndose ampliamente de la experiencia de Struts y corrigiendo los errores que este framework ha presentado. Craig R. McClanahan ha sido el líder y creador tanto de Struts como de JavaServer Faces.

La programación de las interfaces en JSF es similar a como se realiza en Swing, Visual Basic o Delphi. Es decir, la programación se lleva a cabo mediante componentes y eventos. Permite crear componentes personalizados y renderizarlos a criterio del programador.

JSF posee algunas especificaciones que permiten conocer su funcionalidad. Los siguientes sitios web permiten conocer estas especificaciones:

- http://www.jcp.org/en/jsr/detail?id=127
- http://www.jcp.org/en/jsr/detail?id=252
- http://www.jcp.org/en/jsr/detail?id=276

La lógica básica de JSF es ejecutar la aplicación web en el servidor y renderizar las páginas solicitadas (request) del lado del cliente. Esto quiere decir que la aplicación, como la mayoría de aplicaciones Java, se ejecuta en un contenedor Servlet Java, que típicamente cuenta con:

- Componentes JavaBeans (que son los objetos pertenecientes al modelo). En el mismo radican los datos y funcionalidades del aplicación.
- Escuchas de los eventos.
- Páginas JSP como vistas de la aplicación.
- Clases del lado del servidor, las cuales permiten funcionalidad específica del lado del servidor (como Beans que acceden datos, por ejemplo).

Características principales de JSF

Entre las principales características de JSF se encuentran:

- Representación de componentes de interfaz de usuario (UI-User Interface) y manejo de su estado.
- Manejo de eventos, conversión de datos y validación del lado del servidor.
- Definición de la navegación entre páginas.
- Soporte internacional y accesibilidad.
- Extensibilidad y accesibilidad.
- Etiquetas JSP personalizadas para componentes UI internamente en una página JSP.

Es un hecho que con JSF se facilita la construcción de aplicaciones web con UI's del lado del servidor. Con ello se quiere decir que es posible conectar eventos que se generen en el cliente a código que radica en el lado del servidor, mapeo de componentes UI a una página de datos que radica en el servidor o construir la interfaz del usuario mediante un componente extensible y reutilizable.

Beneficios de JSF

JSF proporciona una serie de beneficios a los desarrolladores de aplicaciones web que probablemente le serán de mucha utilidad; entre ellos se encuentran:

- Especificación estándar por lo que existen muchas implementaciones de fabricantes. Con esto se evita depender de un solo proveedor de la tecnología y seleccionar aquel que se adapte más a nuestros requerimientos de desarrollo.
- La vista (view del modelo MVC) es tratada en forma similar a como se hace en Java Swing, Delphi o Visual Basic. Es decir, se basa en componentes de interfaz que contienen eventos.
- La flexibilidad de JSF permite al desarrollador crear sus propios componentes.
- Separa el comportamiento y la presentación, lo cual tradicionalmente se hace del lado del cliente. En JSP no pueden mapearse peticiones de HTTP a manejo de eventos específicos o elementos UI como objetos con estado en el servidor. Al separar la lógica de negocio de la presentación cada desarrollador puede trabajar en la parte de código o de diseño que le fue encomendada.

Funcionamiento de JSF

El funcionamiento de JSF se basa en integrar las tres fases del modelo MVC, cada una de ellas con sus propios componentes. La Figura 12.1 muestra la arquitectura de funcionamiento básico de la tecnología JSF.

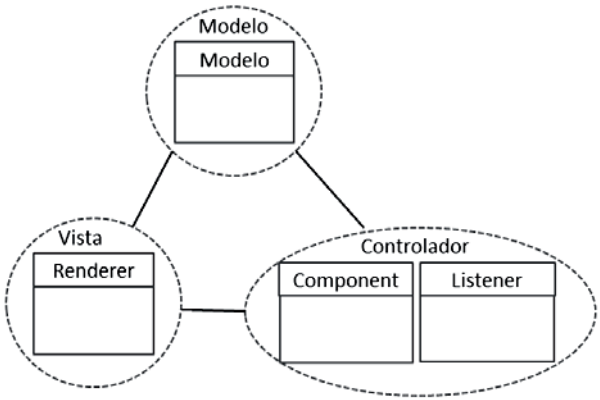


Figura 12.1 Diseño JSF MVC. Adaptado de Hans Bergsten (2004).

En la Figura 12.1 se representa el diseño JSF MVC. En el mismo, el modelo es representado por propiedades de objetos de aplicación, por ejemplo las propiedades de un nombre de usuario en un sistema de información. Los componentes UI de JSF declaran qué eventos pueden desencadenarse, como por ejemplo "*value changed*" o "*button clicked*" y eventos externos tipo listener (escuchas) los cuales representan los controladores. Finalmente, la vista es representada por los componentes UI JSF.

Desarrollo del back-end de la aplicación

Los desarrolladores de aplicaciones son responsables del desarrollo del back-end de la misma (o parte del mismo). Este back-end representa las clases que identifican la lógica de negocio y el acceso a datos. Por ejemplo, la siguiente clase representa un back-end de una aplicación de software.

EJERCICIO: CREACIÓN DE LA CLASE REGISTRO
public class Registro{
private String nombre;
private int edad;
public String getNombre(){
return nombre;}
public int getEdad(){
return edad;}
public void setNombre(String Nombre){
this.nombre = Nombre;}
public void setEdad (int Edad){
this.edad = Edad;}
}

Como podrá observar, en esta clase existe código que establece propiedades (getNombre y getEdad) y métodos (setNombre y setEdad) que permiten el manejo de datos de la clase registro. Estas clases podrían ser más complejas conteniendo código de validación o implementando el mantenimiento a una base de datos.

Primer ejemplo JSF

Como introducción a JSF se creará una sencilla aplicación web que servirá para introducir al lector en la sintaxis y funcionalidad que requiere. Inicie entonces su aplicación.

1. Ingrese a NetBeans y seleccione un proyecto nuevo, de tipo *Java Web*, tal como se muestra en la Figura 12.2.

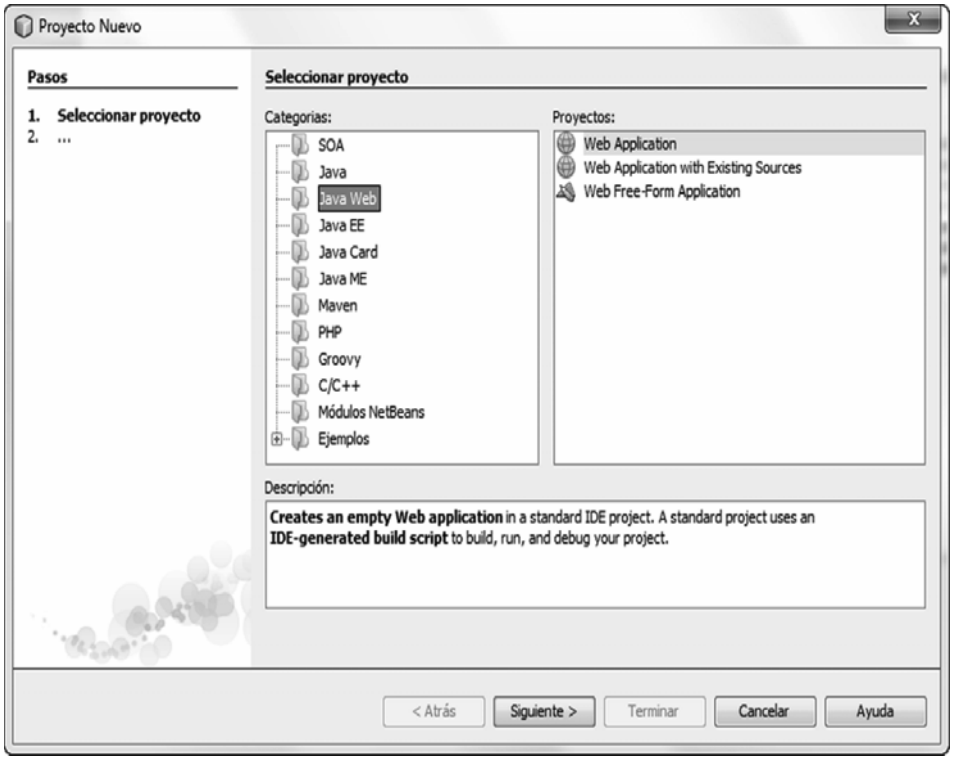


Figura 12.2 Primera aplicación JSF.

2. Una vez seleccionado el tipo de proyecto que se va a trabajar pulse el botón *Siguiente*. En la siguiente pantalla se especifica el nombre y la ubicación que tendrá su aplicación web. El nombre que se asignará a la aplicación será *miPrimerwebJSF*, colóquela en una carpeta donde considere que podrá ubicarla posteriormente. Observe la Figura 12.3 y complete los datos tal y como aparecen ahí.

Pulse el botón *Siguiente* una vez que complete los datos solicitados.

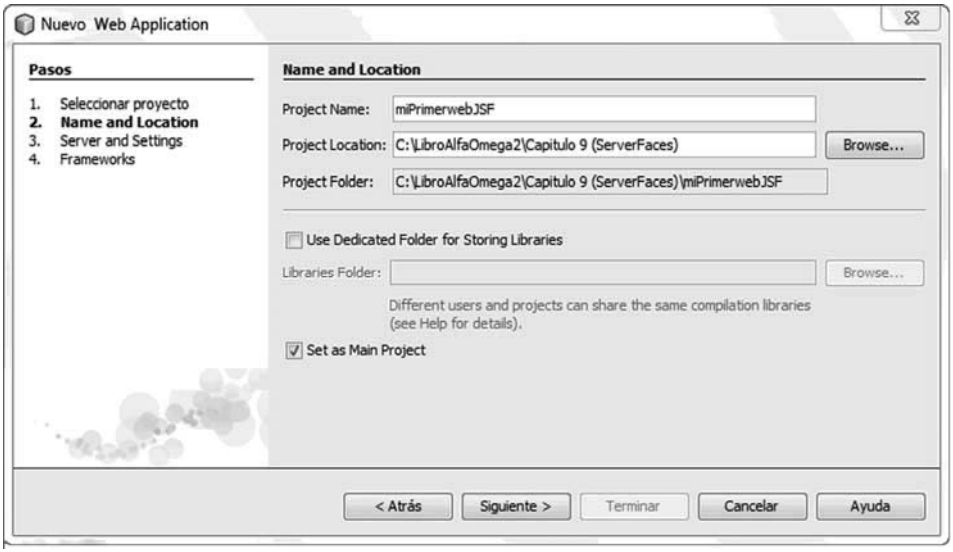


Figura 12.3 Nombre y ubicación de su primera aplicación JSF.

3. Una vez nombrada y ubicada, proceda a seleccionar el servidor donde se ejecutará (en este caso el servidor GlassFish) y la versión de Java EE que se utilizará. Observe la Figura 12.4 respecto de este procedimiento.

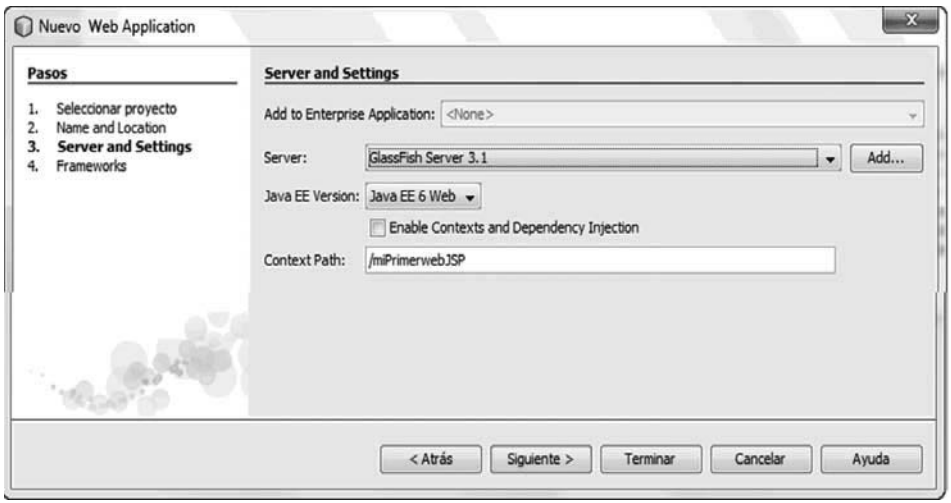


Figura 12.4 Servidor y configuración de su primera aplicación JSF.

4. Pulse el botón *Siguiente* de acuerdo con la Figura 12.4. Se mostrará una pantalla como la que se visualiza en la Figura 12.5.

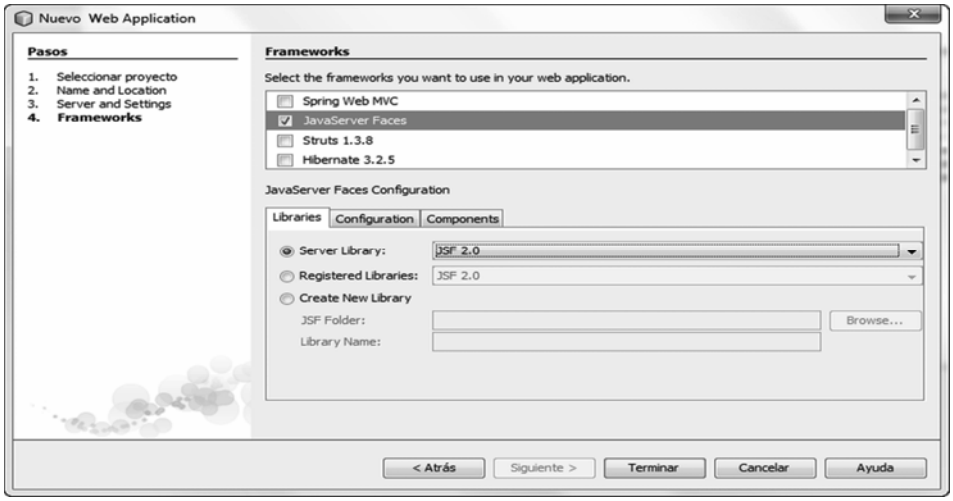


Figura 12.5 Selección del framework JavaServer Faces.

5. Una vez en la Figura 12.5 pulse sobre el botón *Terminar* para que se cree su aplicación web. Aparecerá entonces el código de la página principal (*index.xhtml*). Observe la Figura 12.6.

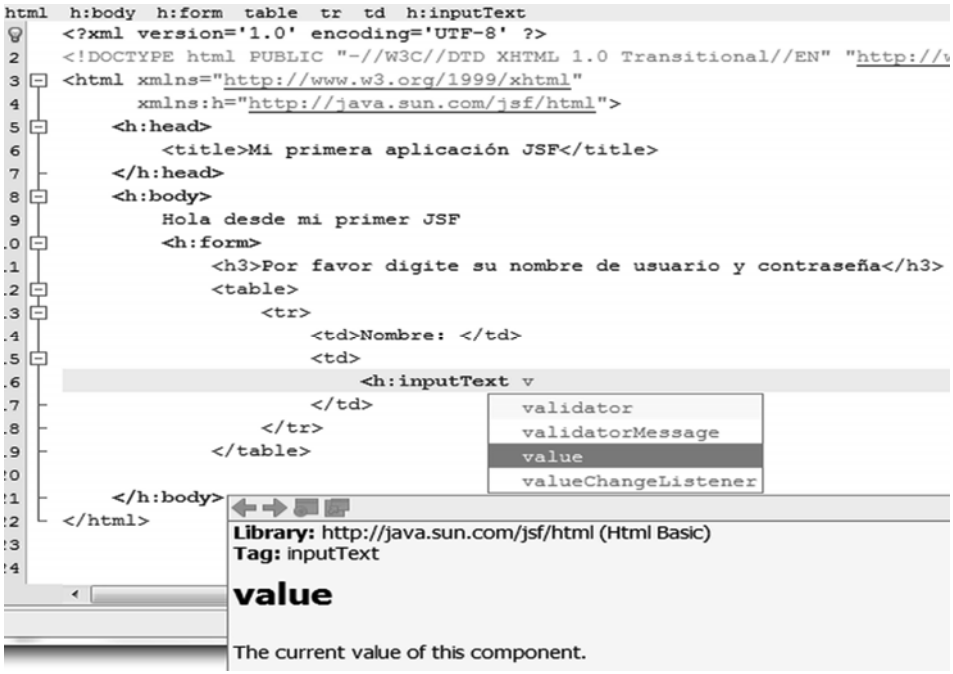


Figura 12.6 Manejo de intellisense en su página index.xhtml.

Como se observa en la Figura 12.6, mediante JavaServer Faces se puede manejar una especie de intellisense que ayuda a conformar el código de la página. Mediante este mecanismo es posible crear componentes en la página, valores iniciales, tipos de componentes, entre otras operaciones. En la Figura 12.7 observe cómo permite construir un componente común como una entrada de texto.

Figura 12.7 Crear un componente en su página index.xhtml.

Una vez que se construye la línea de código que creará el componente se presentará una ventana de contexto que pedirá que se suministre los datos del nombre del componente, valor inicial, tipo, si está o no desactivado, entre otras configuraciones. Esto dependerá del tipo de componente que se construye. En caso

de que se sea un control de tipo *inputSecret* (utilizado para capturar datos de contraseña) mostrará una ventana de contexto como se ve en la Figura 12.8

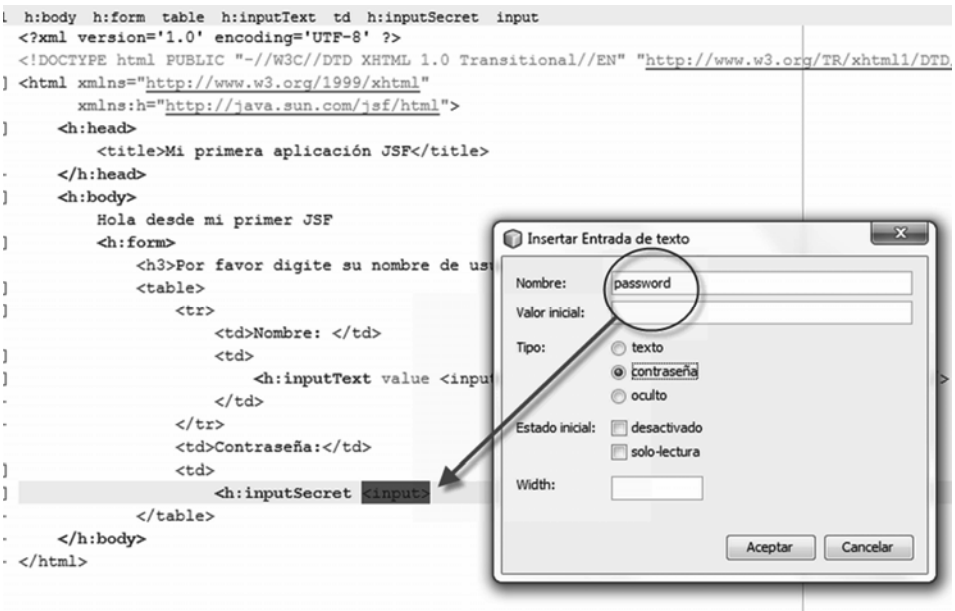


Figura 12.8 Configurar un componente en su página index.xhtml.

Antes de continuar se analizará los tag's que se presentan en esta parte donde se inicia la codificación de su aplicación JavaServer Face, dado que en el componente que permitirá generar la vista (el documento XHTML) se utilizan mucho, tanto para dar formato al documento como para declarar, inicializar y configurar los componentes que estarán presentes en la página. La tabla 12.1 muestra los tag's correspondientes al core del framework.

Tabla 12. 1 Descripción de Tag's en XHTML (Core Tag's).

TAG	DESCRIPCIÓN
f:view	Crea una vista al más alto nivel (en la raíz)
f:subview	Crea una subvista de la vista
f:facet	Agrega una faceta a un componente
f:attribute	Agrega un atributo (clave/valor) a un componente
f:param	Agrega un parámetro a un componente
f:actionListener	Agrega una acción de escucha a un componente
f:valueChangerListener	Agrega una escucha de cambio de valor a un componente
f:converter	Agrega un convertidor arbitrario a un componente
f:convertDateTime	Agrega un convertidor de fecha tiempo a un componente

TAG	DESCRIPCIÓN
f:convertNumber	Agrega un convertidor de número a un componente
f:validator	Agrega un validador a un componente
f:validateLength	Valida la longitud de un componente
f:validateLongRange	Validador un rango long a un componente
f:selectitems	Especifica los ítems que están seleccionados
f:selectitem	Especifica el ítem actualmente seleccionado
f:verbatim	Agrega una marca a una página JSF.

También es importante listar los tag's HTML de JavaServer Faces y la forma en que se utilizan en la configuración de una página XHTML. La tabla 12.2 describe algunos de estos tag's.

Tabla 12. 2 Descripción de Tag's HTML para JSF.

TAG	DESCRIPCIÓN
h:form	Crea un formulario tipo HTML
h:inputText	Crea un control de entrada de texto de una sola línea.
h:inputTextArea	Crea un control de entrada de texto multilínea.
h:inputSecret	Crea un control de entrada para contraseña.
h:inputHidden	Crea un control de entrada oculta.
h:outputLabel	Crea un control de salida (etiqueta)
h:outputLink	Crea un control de salida a través de un enlace
h:outputFormat	Crea un control de salida formateado
h:outputText	Crea un control de salida de texto
h:commandButton	Crea un control de comando
h:commandLink	Crea un control de comando en forma de enlace
h:message	Despliega el más reciente mensaje para el componente
h:messages	Despliega todos los mensajes
h:grapicImage	Despliega una imagen
h:selectOneListBox	Selección en una lista de texto
h: selectOneMenu	Selección en un menú
h:selectOneRadio	Conjunto de botones de radio
h:selectBooleanChecBox	Caja de chequeo
h:selectManyCheckBox	Selecciona varias opciones en la caja de chequeo
h:selectManyListBox	Selecciona varios elementos en la lista de texto
h:selectManyMenu	Selecciona varias opciones del menú

TAG	DESCRIPCIÓN
h:panelGrid	Tabla HTML
h:panelGroup	Dos o más componentes (grupo de paneles)
h:dataTable	Control tipo table
h:column	Columna de una tabla

Ahora continúe con la aplicación JSF, codificando el archivo XHTML. El código a escribir es el siguiente:

PROGRAMA INDEX.XHTML
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:h="http://java.sun.com/jsf/html" xmlns:f="http://java.sun.com/jsf/core">
Comentario: se crea la vista a nivel de la página web <f:view>
Comentario: se crea la cabecera de la página web <h:head>
Comentario: se crea el título de la página web. <title>Mi primera aplicación JSF</title>
Comentario: fin de la cabecera de la página </h:head>
Comentario: inicia el cuerpo de la página web <h:body>
Comentario: inicia el formulario HTML de la página web <h:form>
Comentario: se crea un encabezado (header) HTML de tercer nivel TAG's en JSF
Comentario: se crea una tabla HTML <table>
Comentario: <tr> table row (crea una fila en la tabla) <td> table data (datos que van en esa fila) <tr><td>
Comentario: primera línea (outputLabel) muestra la etiqueta, la segunda (inputText) el texto de entrada (solo lectura y tamaño 30) <h:outputLabel for="cajaTextoRO" value="Caja de Texto Solo Lectura: "/> <h:inputText value="NO PUEDES ESCRIBIR AQUI #{cajaTextoRO}"

PROGRAMA INDEX.XHTML
readonly="true" size="30"/>
</td></tr>
<tr><td>
Comentario: Etiqueta con una entrada de texto tipo Password. <h:outputLabel for="password" value="Caja de texto para contraseña: "/> <h:inputSecret value="#{password}" redisplay="true"/>
</td></tr>
<tr><td>
Comentario: caja de texto letra amarilla, fondo verde azulado <h:outputLabel for="cajaTextoColor" value="Caja con fondo y letra en color "/> <h:inputText value="cajaTextoColor" style="color: Yellow; background: Teal;"/>
</td></tr>
<tr><td>
Comentario: caja de texto con valores predeterminados, tamaño 10 caracteres <h:outputLabel for="cajaTextoColor"value="Caja con valores predeterminados "/> <h:inputText value="1234567890" maxlength="10" size="10"/>
</td></tr>
<tr><td>
Comentario: caja de texto multilínea de 3 filas y 10 columnas. <h:outputLabel for="CajaTextoArea" value="Caja de Texto Area "/> <h:inputTextarea value="123456789012345" rows="3" cols="10"/>
</td></tr>
<tr><td>
Comentario: imagen con borde y de tamaño 200 <h:outputLabel for="enrique.jpg" value="Imagen (graphicImage) "/> <h:graphicImage value="/enrique.jpg" style="border: thin solid black" width="200" />
</td></tr>
<tr><td>
Comentario: botón de comando que permite ir a la página paginaPrin <h:outputLabel for="botonTexto" value="Boton de Comando: "/> <h:commandButton value="Ir a la pagina principal." action="paginaPrin" />
</td></tr>
<tr><td>
Provincias :
Comentario: se crea una lista desplegable de valores

PROGRAMA INDEX.XHTML
<select name="listaDesplegable">
<option>Guanacaste</option>
<option>Puntarenas</option>
<option>Alajuela</option>
<option>Heredia</option>
<option>San Jose</option>
<option>Cartago</option>
<option>Limon</option>
</select>
</td></tr>
<tr><td>
Comentario: se crea una lista de opciones mediante botones de radio <h:selectOneRadio layout="pageDirection" border="">
<f:selectItem id="opt1" itemLabel="Lunes" itemValue="1" />
<f:selectItem id="opt2" itemLabel="Martes" itemValue="2" />
<f:selectItem id="opt3" itemLabel="Miercoles" itemValue="3" />
<f:selectItem id="opt4" itemLabel="Jueves" itemValue="4" />
<f:selectItem id="opt5" itemLabel="Viernes" itemValue="5" />
<f:selectItem id="opt6" itemLabel="Sabado" itemValue="6" />
</h:selectOneRadio>
</td></tr>
Comentario: fin de la table </table>
Comentario: fin del formulario </h:form>
Comentario: fin del cuerpo del formulario </h:body>
Comentario: fin de la vista de página </f:view>
Comentario: fin de la pagina HTML </html>

Al ejecutar este archivo XHTML se obtiene una página HTML que muestra todos los posibles controles que puede utilizar en esta primera aplicación JSF. La Figura 12.9 muestra la ejecución de este archivo.

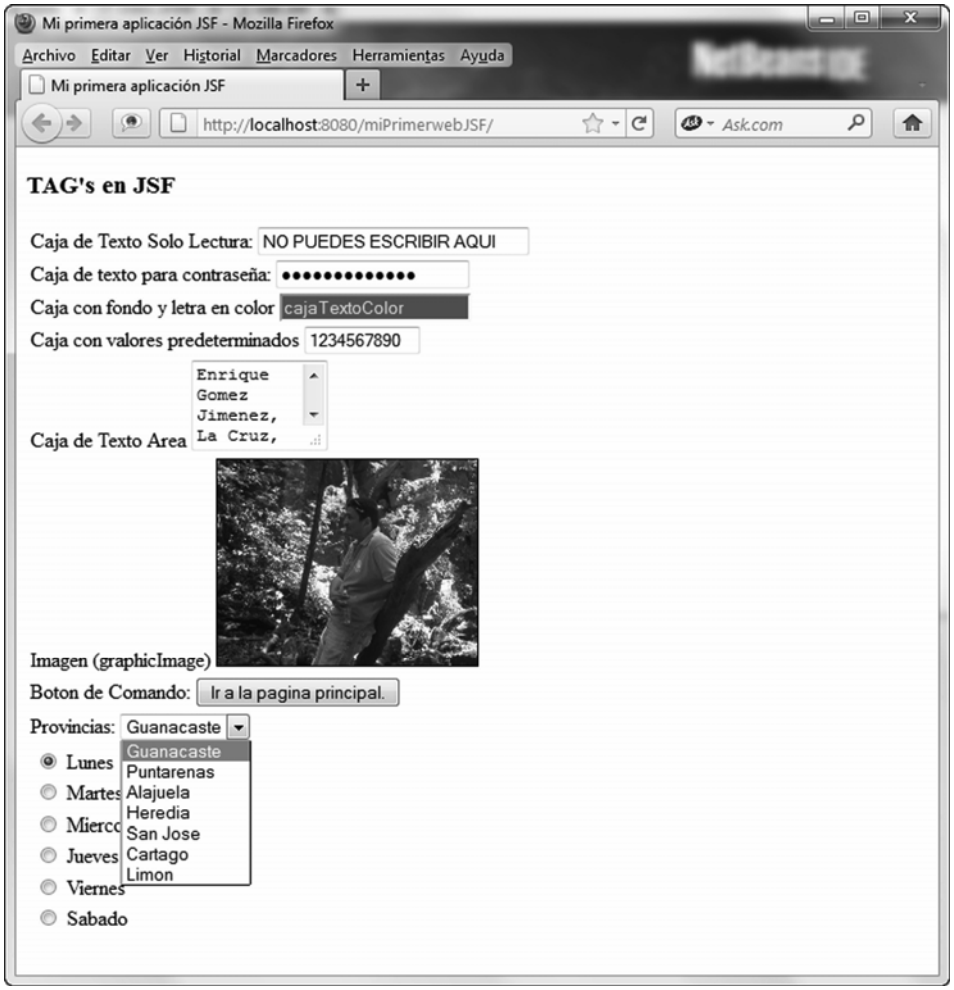


Figura 12.9 Ejecución de la página index.xhtml.

Se ha utilizado una página XHTML para crear el código y en la misma se han usado etiquetas de la forma h:inputText o h:inputSecret (etiquetas propias de JSF) que en realidad corresponden a las etiquetas HTML TextField y PasswordField. También se puede corroborar que los campos de entrada se conectan al atributo del objeto. Por ejemplo, el atributo #{password}, según la implementación JSF conecta al PasswordField HTML con el atributo password.

Actividades para el lector

Elabore un resumen que le permita comprender adecuadamente el Framework: *JavaServer Faces.*

Actividades para el lector

Implemente el siguiente código en el ejemplo creado para los componentes *selectManyMenu* y *selectManyMenu*.

a)
```
<h:selectManyMenu value="#{restaurante.comidas}">
<f:selectItem itemValue="ArrozconPollo" itemLabel="Arroz con Pollo"/>
<f:selectItem itemValue="PescadoFrito" itemLabel="Pescado Frito"/>
<f:selectItem itemValue="SopadeCarne" itemLabel="Sopa de Carne"/>
<f:selectItem itemValue="Hamburguesa" itemLabel="Hamburguesa"/>
<f:selectItem itemValue="PapasFritas" itemLabel="Papas Fritas"/>
</h:selectManyMenu>
```

b)
```
<h:selectManyCheckbox value="#{Persona.actividades}">
<f:selectItem itemValue="Futbol" itemLabel="Fútbol"/>
<f:selectItem itemValue="Baile" itemLabel="Baile"/>
<f:selectItem itemValue="Pesca" itemLabel="Pesca"/>
<f:selectItem itemValue="Lectura" itemLabel="Lectura"/>
<f:selectItem itemValue="Beisbol" itemLabel="Beisbol"/>
</h:selectManyCheckbox>
```

ICEFaces

ICEFaces es un framework de código abierto (open Source) que compatibiliza con JSF. Está orientado a la construcción de aplicaciones web con AJAX al estilo RIA (Rich Internet Application). La estrategia principal es incluir un conjunto de TAG's en las páginas JSP o XHTML las cuales generan en forma automática el código AJAX requerido. En otras palabras, colocar los controles ICEFaces en las páginas JSF o XHTML se implementan los flujos de comunicación entre el cliente y servidor de sólo lo estrictamente necesario. Está basado en los estándares JEE y extensiones JEE.

Beneficios y novedades para el desarrollo de aplicaciones en la empresa

ICEFaces permite adaptarse a las necesidades de una empresa, con los siguientes beneficios y novedades:

- Maximización de la productividad de desarrollo.
- Permite la escalabilidad
- Facilita la portabilidad
- Se basa en un framework conocido y exitoso: JSF
- Automatiza el uso de AJAX (incluido AJAX Push en Java EE)

- Adoptado por gran cantidad de empresas a nivel mundial, lo que demuestra su aceptación.
- Es parte de la tecnología JEE6
- El modelo Facelets es la estrategia para el desarrollo de componentes
- Soporte amplio para eventos del sistema y guardado parcial de estados.

Se considera que ICEFaces es el líder en la integración de AJAX con JEE. Ofrece un amplio soporte para aplicaciones de servidor y constituye un poderoso framework Open Source. Mediante este framework es posible incluir un conjunto de tags AJAX en las páginas JSP o xhtml, de tal manera que el código AJAX se genera automáticamente. Es decir, al colocar los controles ICEFaces en la página JSP o xhtml, el control de enviar la información necesaria entre cliente y servidor se ejecuta automáticamente.

Entre los elementos principales de este framework se encuentran:

- PFS: Persistent Faces Servlet, los cuales son url's con extensión ".iceface" que se encargan de la ejecución del ciclo de vida de una página JSF.
- BS: Blocking Servlet, el cual se encarga de todas las peticiones de bloqueo y el no bloqueo entre las primeras páginas.
- Parseador D2D: se responsabiliza de la gestión de componentes en documentos JSP. Ejecuta la etiqueta de procesamiento del ciclo de vida con el objetivo de construir el árbol parse de la página (sólo una vez para cada página)
- DOM Response Writer. Responsable de la serialización y escritura de documentos DOM.
- DOM Serializer. Se responsabiliza de la serializar el DOM de la página inicial.
- DOM Update. Conjunta las "DOM mutations" en una única actualización DOM.
- Component Suite: conjunto de componentes rich JSF con influencia AJAX.
- Client-Side AJAX Bridge. Se responsabiliza de la actualización DOM.

La tabla 12.3 muestra algunos de los componentes de ICEFaces, para el desarrollo de aplicaciones web.

Tabla 12.3 Componentes de ICEFaces.

COMPONENTE	DESCRIPCIÓN
Checkbox	Casilla de verificación. Se permite seleccionar varias opciones.
Columns	Columna de datos dentro de un componente padre UIData
columnGroup	Se utiliza para crear múltiples datatables con cabeceras o pies de grupos.
Columns	Empleado para procesamiento de tablas de múltiples columnas.
commandButton	Permite crear un botón de envío o reinicio.
commandLink	Permite crear un botón de enlace web.
commandSortHeader	Permite ordenar datos en una tabla relacionada al componente.
dataExporter	Exporta el contenido de datos a una tabla en una variedad de formatos.
dataPaginator	Utilizado para procesar la navegación en un conjunto de páginas.
dataTable	Tabla HTML vinculada al modelo de datos subyacentes
effect	Agrega efectos a un componente padre
form	Formulario HTML con agregación de características especiales.
gMap	Para uso de mapas Google.
gMapControl	Agrega controles a gMap.
gMapDirection	Direcciona de un punto X a un punto Y en un mapa google.
graphicImage	Elemento HTML “img”.
headerRow	Fila de la cabecera de una tabla.
graphicImage	Elemento HTML “img”.
inputHidden	Elemento “input” de tipo “hidden”.
inputRichText	Componente RIA basado en JSP.
inputSecret	Elemento HTML “input” de tipo “password”.
inputText	Elemento HTML “input” de tipo “text”
inputTextarea	Elemento “HTML” de tipo “textarea”
menuBar	Provee un robusto sistema de menú.
menuItem	Ítems contenidos por un elemento menú.
menuItemSeparator	Separa un grupo de ítems de menú
message	Mensaje simple para un componente específico

Nota
En la página http://wiki.icefaces.org/display/ICE/ICEfaces+Components encontrará más información sobre éstos y otros componentes

Una importancia sustancial en ICEFaces es que utiliza AJAX para la actualizacion inteligente de las páginas de un sitio web. Con JavaServer Faces (JSF) 2.0 se usaban controles AJAX de la forma <f:ajax> en cada componente de interfaz de usuario, pero esto degradaba sensiblemente el rendimiento de los desarrolladores porque tenían que colocar este tag en cuanto control requiriera la funcionalidad de AJAX. Por ende, el resultado eran aplicaciones web no escalables.

Con ICEFaces 2.0 combinado con las capacidades de JSF 2.0 este tag ya no es necesario, pues mediante el renderizado Direct 2 de DOM (D2D) la funcionalidad AJAX es automática. Se utiliza entonces un mecanismo de actualizacion incremental de las páginas JSF para producir cambios en el DOM del navegador empleado en el cliente. Por ejemplo, el siguiente fragmento de código muestra entradas y salidas controladas mediante la funcionalidad AJAX. Observe la línea:
<icecore:singleSubmit>. Con ello se implementa esta funcionalidad AJAX.

```
<h:form>
  <icecore:singleSubmit>
    <h:panelGrid columns="1">
      <h:inputText id="miTextoEntrada" value=""/>
      <h:outputText id="miTextoSalida" value=""/>
    </panelGrid>
  </icecore:singleSubmit>
</h:form>
```

La Figura 12.10 muestra la manera en que funciona la secuencia de actualizacion de los componentes de una página en ICEFaces al utilizar AJAX.

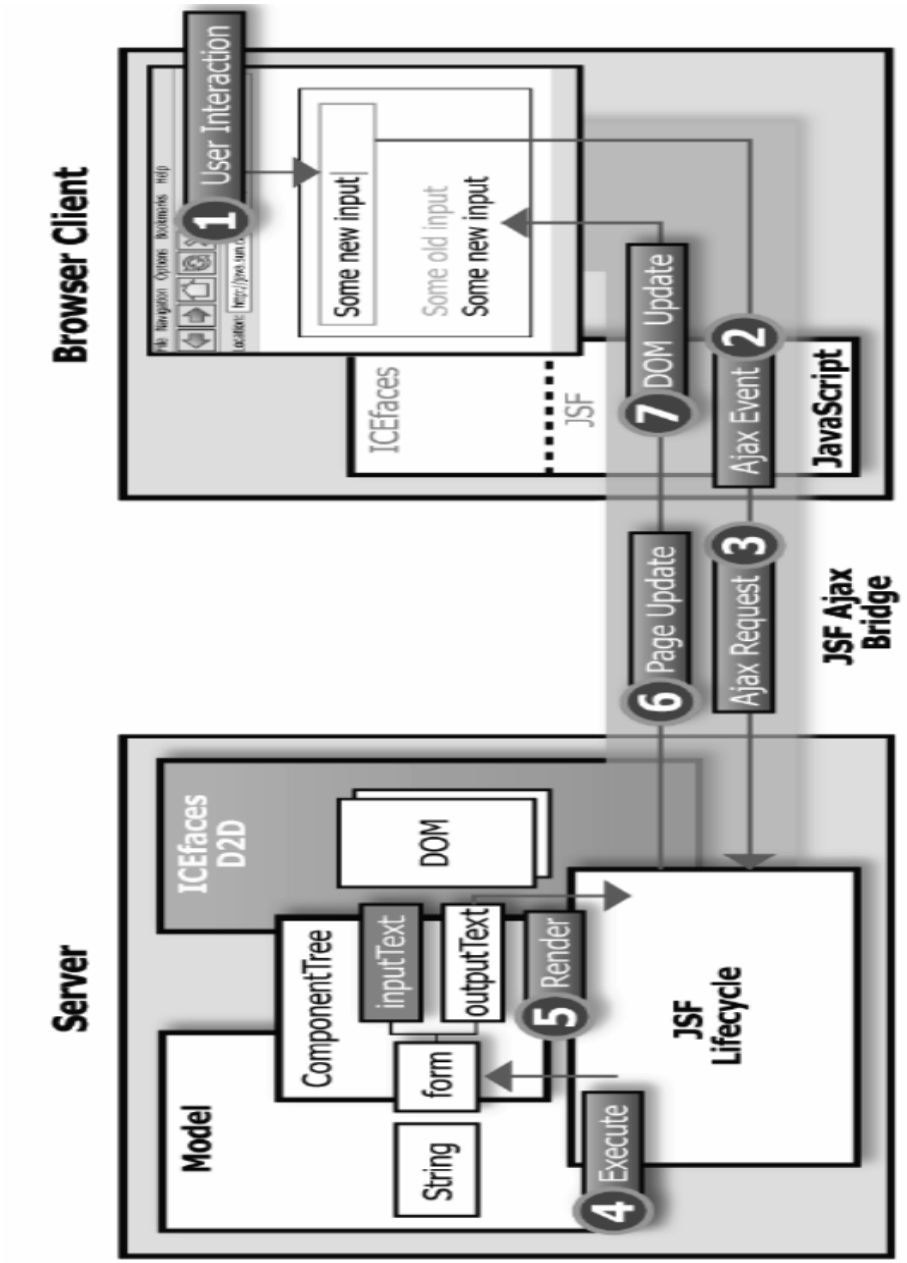


Figura 12.10 Actualización de los componentes de una página con ICEFaces que utiliza AJAX, extraído de http://www.kamlov.site90.net/?p=1040 el 21-09-2011

Como se observa en la Figura 12.10, existe una interacción entre el servidor y el navegador del cliente de tal manera que se establece un puente AJAX entre estas capas para gestionar la actualizacón del cliente. El mecanismo establecido según la Figura 12.10 es la siguiente:

1. El usuario inicia la interaccion con el sistema al ingresar datos en el campo de texto.
2. Una vez que el foco de atención del usuario ya no es el campo de texto (ya digitó el dato, por ejemplo, y pasa a otro componente) se dispara un evento AJAX.
3. Se produce una petición (request) AJAX a través de un puento JSF AJAX Bridge, al servidor.
4. Sólo en el componente de texto (inputText) JSF se ejecuta el ciclo de vida utilizando *Single Submit*, es decir, es como si este único objeto es el que existiera entre la página del cliente en el navegador y el servidor. Los demás objetos no tienen por qué interactuar con el servidor dado que no se ha procesado nada en ellos. Es un renderizado parcial de páginas, como llamarían en Microsoft®. En este contexto el modelo del objeto texto es el que se actualiza.
5. Automáticamente, mediante D2D se actualiza en forma automática el modelo (en este caso el inputText).
6. La actualización de página, en este caso del modelo, se envía al navegador del cliente.
7. La actualización DOM se aplica en el navegador del cliente.

Como se observa, mediante *Single Submit*, que es una característica del AJAX automático, solamente se permitió actualizar el componente que en realidad se requería en la aplicación. Así, cada componente que requiera procesamiento se tratará en forma individual para no realizar funciones sobre otros que no lo requieren.

Actividades para el lector

Elabore un resumen que le permita comprender adecuadamente elFramework: *ICEFaces.*

A continuación se desarrollará un ejemplo muy sencillo de la utilización de esta funcionalidad en una aplicación web ICEFaces:

Ejemplo ICEFaces

1. Ingrese a NetBeans y proceda a crear un nuevo proyecto web de nombre *webICEFace*, tal como se observa en la Figura 12.11.

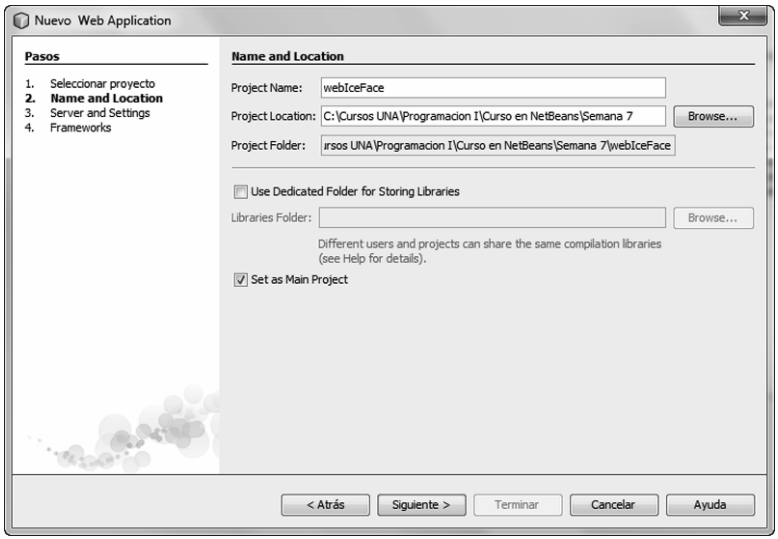


Figura 12.11 Creación del sitio web.

2. Pulse el botón *Siguiente* y en la pantalla que sigue pulse *Siguiente* también.
3. En la pantalla que aparece deberá seleccionar ICEFaces, como muestra la Figura 12.12.

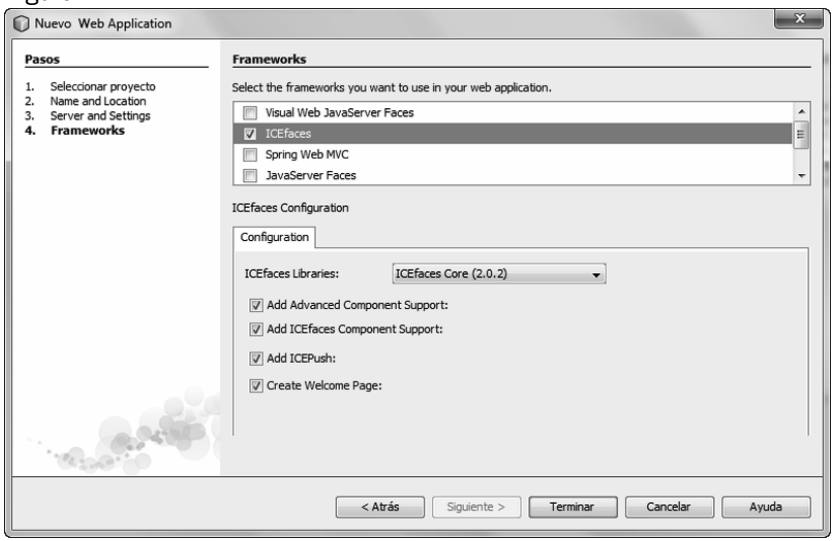


Figura 12.12 Mi primera aplicación ICEFaces.

4. Pulse el botón *Terminar*.
5. Proceda ahora a crear un bean que utilizará la aplicación web (recuerde que un bean es un componente de software que tiene el objetivo de ser

reutilizable y así evitar la reescritura de código). En este caso el bean será una clase que albergará aquellos atributos que puede generalizar en la aplicación. Por ende, proceda a crear una clase denominada *BeanPersona*. El código se muestra a continuación:

CLASE BEANPERSONA
Comentario: Clase basada en ejemplo en org.icefaces.tutorial package org.icefaces.tutorial.singlesubmit.beans;

Comentario:
//la serialización de un objeto, en este caso de input/output consiste en
//generar una secuencia de bytes para almacenamiento o transmisión.
//Posteriormente, se deserializa para reconstruir el objeto.
import java.io.Serializable;

Comentario: //librería que maneja los beans de javax.faces.bean
//un managedBean es una clase Java que se utiliza para procesar información
//Cada página JSP puede tener un managedBean para manejar sus datos
import javax.faces.bean.ManagedBean;

import javax.faces.bean.ViewScoped;
import javax.faces.event.ActionEvent;
@ManagedBean(name="beanPersona")
@ViewScoped
public class BeanPersona implements Serializable {
private String nombre;
private String apellidos;
private int edad;
private String genero;
private String estadoCivil;
public BeanPersona() {} //constructor de la clase
public String getNombre() { return nombre;}
public void setNombre(String nombre) { this.nombre = nombre;}
public String getApellidos() { return apellidos;}
public void setApellidos(String apellidos) { this.apellidos = apellidos;}
public int getEdad() { return edad;}
public void setEdad(int edad) { this.edad = edad;}

CLASE BEANPERSONA
public String getGenero() { return genero;}
public void setGenero(String genero) { this.genero = genero;}
public String getEstadoCivil() { return estadoCivil;}
public void setEstadoCivil(String estadoCivil) { this.estadoCivil = estadoCivil;}
public void submitButton(ActionEvent event) { System.out.println("Botón enviar pulsado: " + nombre + " , " + apellidos + ", " + edad + ", " + genero + ", " + estadoCivil);} }

6. Guarde la clase creada y proceda a escribir código ICEFaces en el archivo *welcomeICEfaces.xhtml* que se creó por default en el proyecto. Sustituya el código que se crea por default por el siguiente código:

WELCOMEICEFACES.XHTML
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:h="http://java.sun.com/jsf/html"
xmlns:f="http://java.sun.com/jsf/core"
Comentario: //llamado a las instancias de icefaces xmlns:icecore="http://www.icefaces.org/icefaces/core">
Comentario: //Tag cabecera de JSP en HTML <h:head>
Comentario: //Título de la página <title>Formulario de envio simple!</title>
Comentario: //fin de cabecera </h:head>
Comentario: //inicio del cuerpo de la página <h:body>

WELCOMEICEFACES.XHTML
Comentario: //formulario JSP en HTML <h:form>
Comentario: //implementación de la funcionalidad de AJAX en la página <icecore:singleSubmit/>
Comentario: //objeto panelGrid de 3 columnas para la ubicación de otros componentes <h:panelGrid columns="3" border="1" rules="all" title="PanelGrid">

Comentario:
//Se crea la etiqueta Nombre: y el objeto nombre que es de tipo inputText, el cual
//referencia al atributo nombre de la clase BeanPersona nombre a través del
//manejador beanPersona. Asimismo, se establece que sea requerido
//(requerid="true") Se establece una longitud máxima de 50 caracteres y que se
//habilite un mensaje de control para él.
<h:outputLabel for="nombre" value="Nombre:"/>
<h:inputText id="nombre" value="#{beanPersona.nombre}" required="true"
maxlength="50"/>
<h:message for="nombre"/>

<h:outputLabel for="apellidos" value="Apellidos:"/> <h:inputText id="apellidos" value="#{beanPersona.apellidos}" required="true" maxlength="100"/> <h:message for="apellidos"/>

Comentario:
//igual que el componente nombre, con la diferencia de que se establece
//que exista un mínimo de caracteres de 2 y un máximo de 2 caracteres.
//asimismo, se establece que se valide un rango entre 1 y 99.
<h:outputLabel for="edad" value="Edad:"/>
<h:inputText id="edad" value="#{beanPersona.edad}" required="true"
size="2" maxlength="2">
<f:validateLongRange minimum="1" maximum="99"/>
</h:inputText>
<h:message for="edad" style="color:red"/>

Comentario:
//se crea un objeto selectOneMenu donde se despliegan una serie de opciones
//para que el usuario escoja sólo una de ellas. En este caso, masculino o
//femenino.
<h:outputLabel for="genero" value="Genero:"/>
<h:selectOneMenu id="genero" value="#{beanPersona.genero}" required="true">
<f:selectItem noSelectionOption="true" itemValue="" itemLabel="--Seleccione--"/>

WELCOMEICEFACES.XHTML
<f:selectItem itemValue="Masculino"/> <f:selectItem itemValue="Femenino"/> </h:selectOneMenu><h:message for="genero"/>
<h:outputLabel for="estadoCivil" value="Estado Civil:"/> <h:selectOneMenu id="estadoCivil" value="#{beanPersona.estadoCivil}" required="true"> <f:selectItem noSelectionOption="true" itemValue="" itemLabel="--Seleccione--"/> <f:selectItem itemValue="Soltero"/> <f:selectItem itemValue="Casado"/> <f:selectItem itemValue="Viudo"/> <f:selectItem itemValue="Unión Libre"/> <f:selectItem itemValue="Divorciado"/> </h:selectOneMenu> <h:message for="estadoCivil"/>
// **Comentario**: final de panelGrid </h:panelGrid>

Comentario::
//se implementa un botón de comando, el cual referencia al manejador del bean
//beanPersona de la clase BeanPersona.
<h:commandButton value="Enviar"
actionListener = "#{beanPersona.submitButton}"/>

// **Comentario**: fin del formulario en la página </h:form>
<h:panelGrid columns="2" border="1">
// **Comentario**:: se inicia el facelet para la presentación de datos. <f:facet name="header"> Información Personal </f:facet>
Comentario: //se crea una etiqueta y un objeto outputText para generar la salida de datos <h:outputLabel for="nombreSalida" value="Nombre:"/> <h:outputText id="nombreSalida" value="#{beanPersona.nombre}"/>
<h:outputLabel for="apellidosSalida" value="Apellidos:"/> <h:outputText id="apellidosSalida" value="#{beanPersona.apellidos}"/>
<h:outputLabel for="edadSalida" value="Edad:"/> <h:outputText id="edadSalida" value="#{beanPersona.edad}"/>

WELCOMEICEFACES.XHTML
<h:outputLabel for="genero" value="Genero:"/> <h:outputText id="genero" value="#{beanPersona.genero}"/>
<h:outputLabel for="estadoCivil" value="Estado Civil:"/> <h:outputText id="estadoCivil" value="#{beanPersona.estadoCivil}"/>
</h:panelGrid>
</h:body>
</html>

Se ha explicado someramente el código escrito en la página XHTML. Al ejecutar esta aplicación se aprecia algo similar a lo que muestra la Figura 12.13.

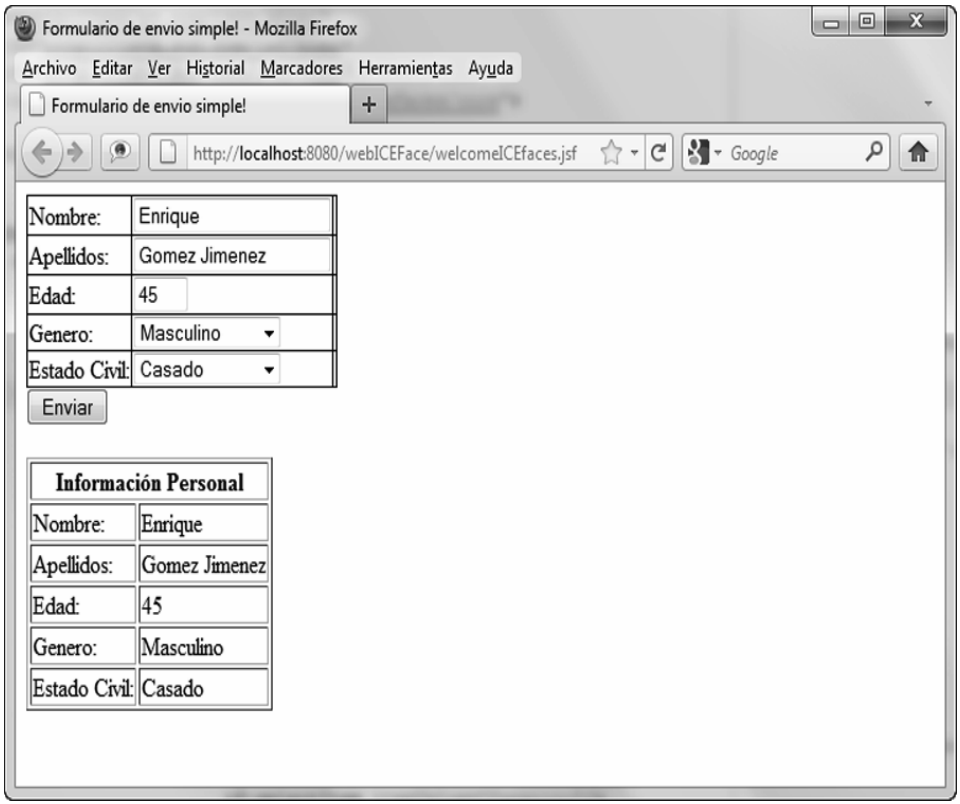


Figura 12.13 Ejecución de mi primera aplicación ICEFaces con AJAX.

Al ejecutar la página anterior notará que una vez que escriba sobre cualquiera de los objetos que la componen (inputText o selectOneMenu), automáticamente se actualiza la salida en la tabla inferior. Esto es trabajo funcional de AJAX en la página.

Hasta aquí se ha trabajado los dos framework que se planteó desarrollar en este capítulo. En realidad, cada framework es un tema para un libro cada uno, por lo que recomiendo consultar las referencias web donde se profundiza el tema.

Actividades para el lector

1. Elabore un resumen que le permita comprender adecuadamente el Framework: *ICEFaces.*

2. Implemente el código ICEFaces que se explica en las páginas:

- http://facestutorials.icefaces.org/tutorial/converter-tutorial.html
- http://www.compujuy.com.ar/postx.php?id=89

Nota

Para profundizar en el uso de JavaServer Faces / ICEFaces, consulte el material complementario en los apoyos web del libro.

RESUMEN

En este capítulo se desarrolló el tema de dos framework importantes en el desarrollo de aplicaciones web con Java y soportados por NetBeans: JavaServer Faces e ICEFaces. Ambos framework están extendiéndose rápidamente entre los desarrolladores de software, por lo que las prestaciones del mismo aumentan cada vez más. Básicamente se trataron los siguientes temas:

1. El fundamento teórico básico sobre los framework JavaServer Faces e ICEFaces para el desarrollo de aplicaciones web con Java.
2. La creación de aplicaciones sencillas con los framework JavaServer Faces e ICEFaces.

EVIDENCIA

☐ Elaboró resúmenes para comprender adecuadamente los framework JavaServer Faces e ICEFaces.

☐ Desarrolló ejemplos alternativos utilizando JavaServer Faces o ICEFaces, que incluyen operaciones aritméticas o lógicas.

☐ Implementó el código ICEFaces que se explica en las páginas:

http://facestutorials.icefaces.org/tutorial/converter-tutorial.html

http://www.compujuy.com.ar/postx.php?id=89

Autoevaluación

1. ¿Qué es un JavaServer Faces?
2. ¿Cómo funciona JSF?
3. Describa al menos dos componentes de un contenedor Servlet Java.
4. Describa al menos dos características de JSF.
5. ¿Qué es ICEFaces?
6. Cite al menos dos beneficios de ICEFaces.

REFERENCIAS

Bibliografía

Ceballos, Fco. Javier (2011). *Java 2: Curso de programación*, 4a. ed., Alfaomega, México.

Hans Bergsten (2004) *JavaServer Faces*. O'Reilly Media Publisher, Sebastopol, CA (USA).

Jonas Jacobi (2006) *Pro JSF: Building Rich Internet Components*. Editorial Board: Steve Anglin, Dan Appleman.

Martín, Antonio, (2010). *Programador certificado Java 2. Curso práctico*, 3a. ed., Alfaomega, México.

Rozannski, Uwe, (2010). *Enterprise Javabeans 3.0. Con Eclipse Y JBoss*. Alfaomega, México.

Ruping, Andreas (2009) *Where Code and Content Meet: Design Patterns for Web Content Management and Delivery, Personalisation and User Participation*. John Wiley & Sons, Limited Publiching.

Sznajdleder, Pablo, (2010) *Java a fondo. Estudio del lenguaje y desarrollo de aplicaciones.* Alfaomega, México.

Páginas Web recomendadas

1. JSF tutorials.net (2011) JSF 2.0 Tutorials and Blogs. Obtenido el 14 de diciembre del 2011 desde http://www.jsftutorials.net/
2. Pérez García, Alejandro (2006) Java Server Faces II parte. Obtenido el 14 de diciembre del 2011 desde http://www.desarrolloweb.com/articulos/2392.php
3. Horstmann.com (2004) JSF Core Tags. Obtenido el 14 de diciembre del 2011 desde http://www.horstmann.com/corejsf/jsf-tags.html
4. Lou Torrijos, Ricard (2011) Introducción a la Tecnología JavaServer Faces. Obtenido el 19 de junio del 2012 desde http://www.programacion.com/articulo/introduccion_a_la_tecnologia_javaserver_faces_233/2
5. icesoft.org (2011) ICEfaces Downloads. Obtenido el 19 de junio del 2012 desde http: http://www.icesoft.org/downloads/icefaces-downloads.jsf

6. infosoftw (2011) Primer programa con JSF y NetBeans. Creando el proyecto. Obtenido el 19 de junio del 2012 desde http://www.infosoftw.com/JSFNetBeansTutorialxHTML/node2.html
7. Pérez García, Alejandro (2006) JSF – Java Server Faces III parte. Obtenido el 19 de junio del 2012 desde http: http://www.desarrolloweb.com/articulos/2404.php
8. Sicuma.uma.es (2011) Tutorial de Java Server Faces. Obtenido el 19 de junio del 2012 desde http://www.sicuma.uma.es/sicuma/Formacion/documentacion/JSF.pdf
9. exadel (2011) JavaServer Faces HTML Tags Reference. Obtenido el 19 de junio del 2012 desde http://exadel.com/web/portal/jsftags-guide
10. ICEfaces wiki (2011) Getting Started with ICEfaces 3. Obtenido el 19 de junio del 2012 desde http://wiki.icesoft.org/display/ICE/Getting+Started+with+ICEfaces+3
11. ICEfaces.org (2011) Ajax automático. Obtenido el 19 de junio del 2012 desde http://www.kamlov.site90.net/?p=1040
12. ICEfaces.org (2011) Tutorials. Obtenido el 19 de junio del 2012 desde http://wiki.icesoft.org/display/ICE/Tutorials

Respuestas a la autoevaluación

1. Es un framework de interfaces de usuario que trabajan del lado del servidor.
2. Ejecutando la aplicación web en el servidor y renderizar las páginas solicitadas (request) del lado del cliente.
3. a. Páginas JSP como vistas de la aplicación; b. Escuchas de los eventos.
4. a. Extensibilidad y accesibilidad; b. Soporte internacional.
5. Es un framework de código abierto que compatibiliza con JSF y que se orienta al desarrollo de aplicaciones web con AJAX.
6. a. Portabilidad; b. Automatización del uso de AJAX.

Fundamentos de programación para dispositivos móviles en NetBeans 7.1

13

Reflexione y responda las siguientes preguntas

¿Reemplazarán los dispositivos móviles a los equipos de sobremesa?
¿Cuál considera usted es la principal ventaja de los equipos móviles?
¿Qué desarrollo considera que es más importante hoy día: para web o para dispositivos móviles?

Contenido

EXPECTATIVA

El diseño de aplicaciones móviles se ha extendido en el mundo, tanto como la tecnología móvil ha avanzado. La actividad comercial ha pasado del escritorio a la web y también a la tecnología móvil, por ello ninguna empresa puede arriesgarse a no tener presencia en ambos medios. Es posible descargar, instalar y utilizar tantas aplicaciones para acceder al correo, así como realizar transacciones bancarias y comerciales entre otras actividades, con sólo tener la aplicación instalada y una conexión a Internet. En este capitulo se trata el tema del desarrollo de aplicaciones con Netbeans 7.1 para dispositivos móviles, así como una introducción al diseño de aplicaciones para juegos.

Después de estudiar este capítulo, el lector será capaz de:

- Comprender el funcionamiento de la tecnología J2ME para el desarrollo de aplicaciones para dispositivos móviles.
- Aplicar el uso de componentes J2ME en el desarrollo de una aplicación para dispositivos móviles.
- Aplicar J2ME en el desarrollo de una aplicación RMS para la persistencia de datos.
- Utilizar J2ME para desarrollar un juego básico para ejecutarse en un dispositivo móvil.

INTRODUCCIÓN

Java ofrece tanta versatilidad en el desarrollo de aplicaciones computacionales que enriquece las probabilidades de implementación en cualquier dispositivo electrónico o forma de despliegue de información.

Sun ha desarrollado distintas ediciones de Java de acuerdo a las necesidades de los usuarios. Así, la versión J2SE (Java Standar Edition) se ha diseñado para el desarrollo de aplicaciones que sean independientes de la plataforma; J2EE (Java Enterprise Edition) se orienta al desarrollo de aplicaciones complejas para el entorno empresarial, y J2ME (Java Micro Edition) se utiliza en el desarrollo de aplicaciones para dispositivos con capacidades limitadas. En esta última es que se ubican las aplicaciones para dispositivos móviles.

- Se utilizá J2ME para desarrollar aplicaciones sencillas que demuestren el uso de los principales objetos que suministra la herramienta, como manejar la persistencia de los datos (que no se pierdan y que se almacenen) mediante Record Management System (RMS) y los fundamentos para la creación de juegos, esto con el objetivo de presentar las principales áreas donde se utiliza el desarrollo de estas aplicaciones.

J2ME (Java2 Micro Edition)

J2ME está pensado para desarrollar aplicaciones para dispositivos móviles con capacidades limitadas tales como PDA's y electrodomésticos inteligentes, entre otros. Posee clases que se implementan en J2SE pero con algunas restricciones y su funcionalidad se ve disminuida debido a los recursos que tienen dichos dispositivos móviles.

A diferencia de una aplicación Java para escritorio con J2SE, que puede utilizar de una JVM (Java Virtual Machine) con toda su potencia, una aplicación J2ME debe emplear KVM (Kilo Virtual Machine) la cual tiene grandes limitaciones de procesamiento.

Según Sergio Galvez: "Una máquina virtual de Java (JVM) es un programa encargado de interpretar código intermedio (bytecode) de los programas Java precompilados a sistema operativo subyacente y observar las reglas de seguridad y corrección de código definidas para el lenguaje Java" (Sergio Galvez, 2003).

Sergio Galvez afirma que la KVM es una implementación de JVM pero reducida y especialmente orientada a dispositivos con bajas capacidades computacionales y de memoria. Está escrita en C y se diseñó para ser pequeña (carga de memoria entre 40 kb y 80 kb), altamente portable, modular y rápida. Las plataformas sobre las que trabaja KVM son Solaris, Windows y Palm OS.

J2ME posee dos configuraciones básicas que aglomeran las características de Java (clases y demás funcionalidades) para trabajar:

a. CLDC (connected limited device configuration), orientada al desarrollo de aplicaciones para dispositivos electrónicos con restricciones de procesamiento y memoria.

b. CDC (connected device configuration), orientada a dispositivos con mayores recursos.

La configuración del entorno de ejecución de J2ME se observa en la figura 13.1.

Figura 13.1 Entorno de ejecución de J2ME.

La tecnología CDC se orienta al desarrollo de aplicaciones con alguna capacidad computacional y de memoria tales como televisores digitales, aparatos médicos inteligentes, sistemas computacionales en automóviles, entre otros, lo cual no compete al interés de este libro.

En el tema de CLDC las limitaciones gráficas, de procesamiento y de memoria de los dispositivos hacen que las tecnologías de desarrollo J2ME también sean limitadas. J2ME incluye las librerías que se presentan en la tabla 13.1.

Tabla 13.1 Librerías incluidas en CLDC.

PAQUETE CLDC	DESCRIPCIÓN
java.io	Permite gestionar las operaciones de E/S en los dispositivos móviles.
java.lang	Se encarga de la gestión que lleva a cabo la KVM.
java.util	Aglomera las clases, interfaces y utilidades para el desarrollo de aplicaciones para dispositivos móviles.
Java.microedition.io	Contiene las clases e interfaces que permiten la conexión genérica de los CDLC.

Perfiles

Los perfiles de una aplicación CLDC controlan el ciclo de vida de la misma. Según Sergio Gálvez: "es un conjunto de APIS, orientado a un ámbito de aplicación determinado. Los perfiles identifican un grupo de dispositivos por la funcionalidad que proporcionn (electrodomésticos, teléfonos, móviles, etc.) y el tipo de aplicaciones que se ejecutan en ellos" (Galvez, 2003).

Los perfiles CLDC suministran una interfaz de usuario, mecanismos de entrada y de persistencia para una aplicación. También representa un entorno de desarrollo que permite la creación de aplicaciones específicas para un grupo determinado de dispositivos. Un dispositivo puede cubrir más de un tipo de perfil. En el caso de CLCD, el perfil es MIDP (Mobile Information Profile).

SISTEMAS OPERATIVOS PARA DISPOSTIVOS MÓVILES

Hoy día se encuentran en el mercado varios sistemas operativos para dispositivos móviles que ofrecen diversas funcionalidades. Con ello se da una batalla fuerte en las prestaciones que se ofrece, alianzas estratégicas entre fabricantes de hardware para dispositivos y desarrolladores de software para los mismos. Son siete los sistemas operativos que están catalogados como los más populares en el mercado:

- **Google Android**: es un sistema operativo para dispositivos móviles como teléfonos celulares o tabletas inteligentes. Es desarrollado por Open Handset Alliance, una empresa de Google, que a su vez también fabrica hardware. Se

considera el sistema operativo para dispositivos móviles más utilizado en el mundo.

- **iOS (iPhone, iPad**): antes se le denominó iPhone OS, siendo el sistema operativo móvil para Apple. Se utiliza en iPod Touch, iPad y Apple TV. Se estima que es el segundo sistema operativo para dispositivos móviles detrás del gigante Google Android.
- **Windows 7 Phone**: es un sistema operativo para dispositivos móviles desarrollado por Microsoft® como suscesor de Windows Mobile. No está pensado para el mundo empresarial sino para el público general. Con Windows Phone 7 Microsoft® integra varios servicios en el sistema operativo y un control estricto sobre el hardware del dispositivo, sin embargo sigue detrás de gigantes como Android e iOS.
- **RIM (BlackBerry)**: constituye una línea de teléfonos celulares inteligentes fabricados por la empresa canadiense RIM (Research In Motion). El teléfono celular es conocido como BlackBerry y se reconoce por su capacidad para enviar y recibir correos electrónicos mediante internet y las compañías que ofrecen el servicio.
- **Symbian**: es un sistema operativo que nació como producto de la alianza de compañías como Nokia, Sony Ericsson, Samsung, LG, Motorola, entre otras. El objetivo del sistema operativo fue competir con Palm o Windows Phone y ahora contra Android de Google, iOS de Apple y BlackBerry de RIM. Actualmente Nokia basa su tecnología en el sistema operativo Symbian.
- **WebOS**: es un sistema operativo multitarea desarrollado para ser utilizado en sistemas embebidos. Se basa en Linux y fue creado por la empresa Palm Inc., propiedad de HP. El Palm Pre y WebOS fueron lanzados al mercado en el 2009, sin embargo HP anunció que descontinuará los dispositivos con WebOS, pero que continuará dándole soporte. Se asume que formará parte del software libre.
- **MeeGo**: constituye la combinación de los sistemas operativos Maemo y Moblin, con los cuales Intel y Nokia pretendían competir contra Android. En el caso de Nokia, su nuevo teléfono celular N9 incluye este sistema operativo. Este sistema operativo no solamente trabaja en dispositivos móviles sino que también lo hace en sistemas vehiculares, televisores, portátiles, entre otros.

Como se puede observar en la lista anterior, existen muchos sistemas operativos que han sido creados por empresas desarrolladoras de software para sus propios dispositivos celulares o para terceros. Por ende, la variedad de opciones y oportunidades de crear software para estos entornos también se amplía debido a que estos fabricantes también desarrollan los plugin que podrán ser utilizados por lenguajes de programación tal como NetBeans. También crean sus propias herramientas de desarrollo, específicas para cada dispositivo.

En la actualidad existen muchos lenguajes para el desarrollo de aplicaciones para dispositivos móviles. No es de extrañar que cada fabricante de dispositivos móviles

diseñe su propio lenguaje o por lo menos un plugin que pueda ser utilizado en cualquier otro lenguaje de programación de terceros.

Un interesante enlace para visualizar las tendencias y el comportamiento de sistemas operativos móviles y otras tecnologías orientadas hacia la web móvil puede verse en http://www.undernews.com/2010/05/03/la-importancia-de-los-desarrolladores-en-la-batalla-por-la-web-movil/

PROGRAMACIÓN PARA MÓVILES CON NETBEANS

Inicie la programación de dispositivos móviles con NetBeans. Para ello siga estas instrucciones:

1. Ingrese a NetBeans.
2. Una vez iniciado el IDE pulse sobre *Proyecto Nuevo* del menú *Archivo* y ahí seleccione *Mobile Application* de la carpeta de proyectos *Java ME*, tal como se muestra en la figura 13.2.

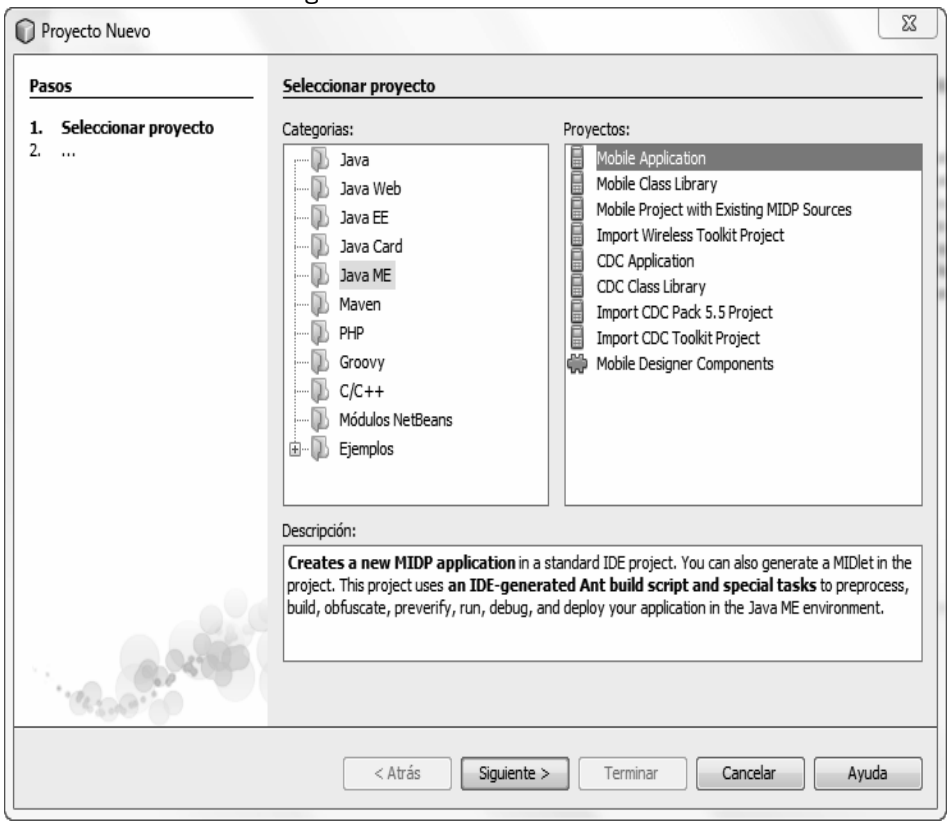


Figura 13.2 Creación de su primera aplicación J2ME.

3. Pulse el botón *Siguiente* (figura 13.2) y en la pantalla que sigue escriba el nombre del proyecto como *miPrimerAplicacionMovil*, tal como se observa en la figura 13.3.

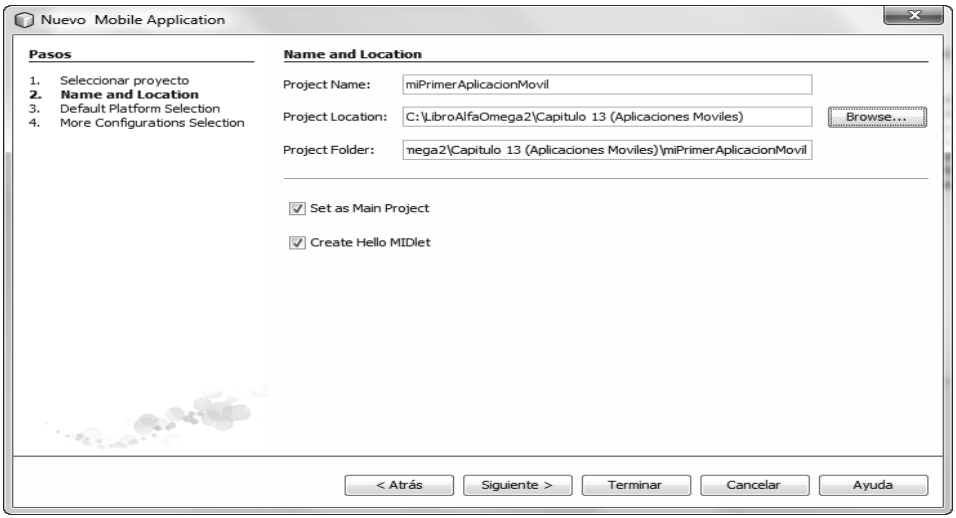


Figura 13.3 Nombrar su primera aplicación J2ME.

Observe que se encuentra habilitada la opción *Create Hello MIDlet*. Ya se ha definido líneas arriba que es un MIDlet. Por esta vez deje marcada esta opción.

4. Pulse el botón *Siguiente* y en la pantalla que sigue asegúrese que se habiliten las opciones de CLDC-1.1 y MIDP-2.1. Por último, pulse el botón *Terminar*. Se visualizará el entorno de desarrollo de la aplicación móvil como lo muestra la figura 13.4.

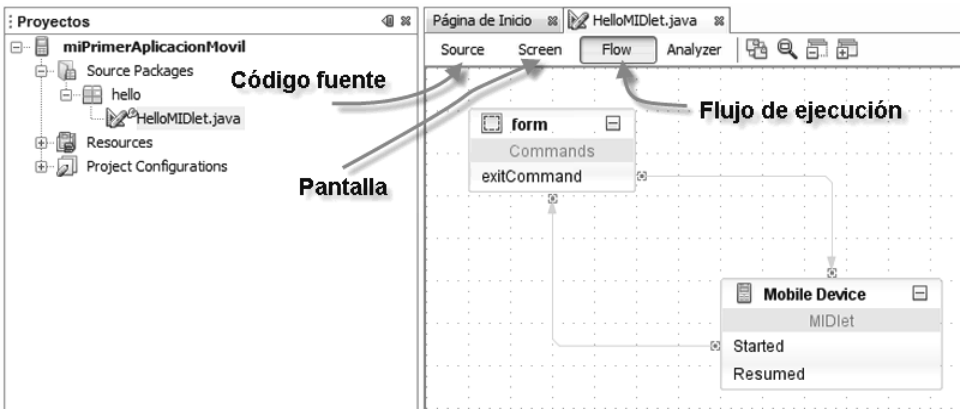


Figura 13.4 Entorno de desarrollo de su aplicación móvil.

Observe que en el entorno que se presenta en la figura 13.04 se encuentran algunas secciones con las cuales hay que familiarizarse. Estas secciones son:

- **Source:** es donde se escribe el código que se utilizará en la aplicación.
- **Screen:** muestra cómo se verá el dispositivo en el emulador. Todos los objetos gráficos que coloque en él se verán en esta sección.
- **Flow:** permite dibujar prácticamente el flujo de ejecución que tendrá su aplicación. Es decir, la secuencia de ejecución de los formularios que formarán parte de la aplicación. Las flechas indican la ruta de ejecución de cada objeto.
- **Analyzer:** permite visualizar el comportamiento de su aplicación móvil. Esto es, cumplimiento con MIDP 1.0 e información sobre qué recursos no están siendo utilizados en la aplicación.

5. Una ejecución típica de esta aplicación se observa en la figura 13.5.

Figura 13.5 Ejecución de su aplicación móvil.

Como podrá observar, es la ejecución de un emulador de un dispositivo celular estándar que acompaña al IDE de NetBeans. Lógicamente existen muchos más emuladores como Sony Ericcson, Nokia, Samsung, entre otros, que presentarán diferentes interfaces de acuerdo con el modelo del teléfono celular que se esté usando. Más adelante descargue un plugin de éstos y se mostrará cómo crear aplicaciones para estos dispositivos sin ningún problema.

6. Ahora se procederá a agregar algunos componentes a su primera aplicación móvil. Para ello pulse sobre la pestaña *Screen* de su aplicación móvil, tal como se muestra en la figura 13.6.

En la figura 13. 6 se ha dado click derecho sobre la pantalla del emulador y aparece inmediatamente un menú de contexto con diferentes opciones.

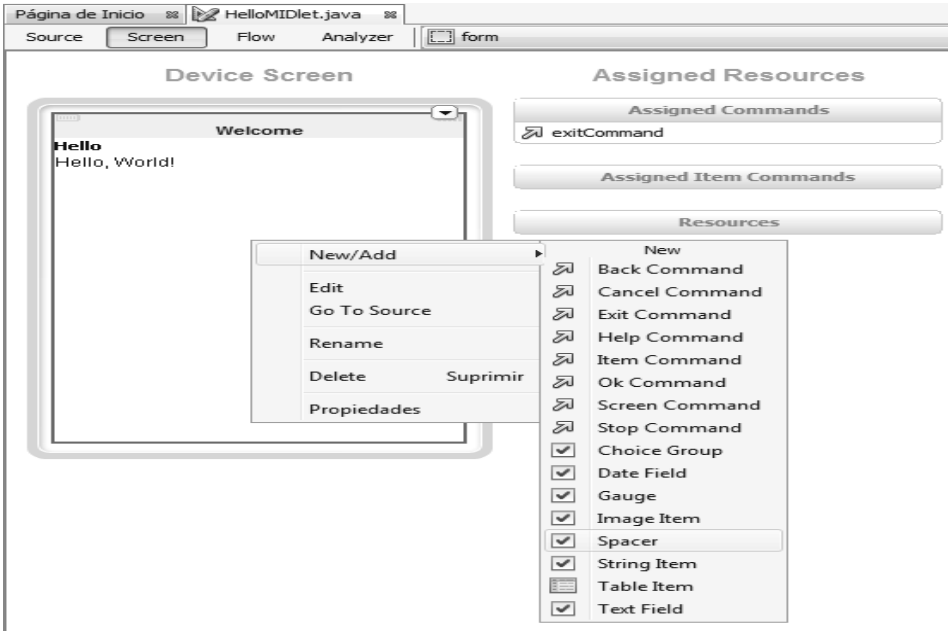


Figura 13.6 Entorno de la sección Screen.

La primera ventana del menú se explica en la tabla 13.2.

Tabla 13.2 Opciones del menú de contexto de la sección *Screen.*

PAQUETE CLDC	DESCRIPCIÓN
New/Add	Permite agregar nuevos objetos y métodos a la aplicación móvil, los cuales se observan en la figura 13.06.
Edit	Permite editar el objeto actualmente seleccionado.
Go To Source	Permite acceder a la parte de código fuente de la aplicación móvil.
Rename	Permite cambiar el nombre del objeto actualmente seleccionado.
Delete	Elimina el objeto actualmente seleccionado.
Propiedades	Permite configurar las propiedades del objeto actualmente seleccionado.

Asi también se tiene las opciones que permiten la opción de menú *New/Add*. La tabla 13.3 explica estos componentes y sus funcionalidades.

Tabla 13.3 Opciones del menú New/Add.

PAQUETE CLDC	DESCRIPCIÓN
Back Command	Permite establecer el retorno al formulario anterior.
Cancel Command	Permite cancelar una operación en el formulario actual.
Exit Command	Permite salir de la aplicación móvil.
Help Command	Permite invocar la ayuda de la aplicación móvil.
Item Command	Permite crear un comando que puede ser programado para que ejecute algo, por ejemplo ir a un formulario. Se usa principalmente para las opciones de menú.
Ok Command	Permite programar un botón de comando.
Choice Group	Permite crear un grupo de opciones.
Date Field	Permite crear un campo tipo fecha.
Gauge	Permite crear un objeto tipo barra de avance.
Image Item	Permite crear un objeto tipo imagen.
Spacer	Permite crear un objeto tipo espacio.
String Item	Permite crear un objeto tipo etiqueta.
Table Item	Permite crear elementos tipo tabla.
Text Field	Permite crear objetos de texto.

7. Proceda a utilizar algunos de los componentes y métodos que se indicaron en la tabla 13.3. Para ello observe los componentes que se colocó en el formulario, tal como se muestra en la figura 13.7

Figura 13.7 Cambiar las propiedades de objetos.

Observe lo siguiente:

a. Cambie el título del formulario de Welcome a Bienvenido. Esto se hace simplemente dando un doble clic sobre el mismo y sobreescribiéndolo, o pulse con el clic derecho del mouse sobre el mismo y seleccione propiedades y ahí modifique la propiedad *Title.*
b. Agregue componentes *Text Field* (Nombre y Apellidos), *Choice Group (Pasatiempos)* y *Date Field* (Fecha de Nacimiento). Observe las relaciones en la figura 13.07. Para los elementos del *Choice Group* se deben agregar elementos (*Fútbol, Cine, Lectura*): pulse sobre Pasatiempos y ahí seleccione *New/Add* y luego *Choice Element*.

8. Al ejecutar la aplicación móvil podemos ver la manera en que se establece la funcionalidad de los objetos, sin mayor codificación. La figura 13.8 muestra la ejecución de la misma.

Probar la gestión de teclado.

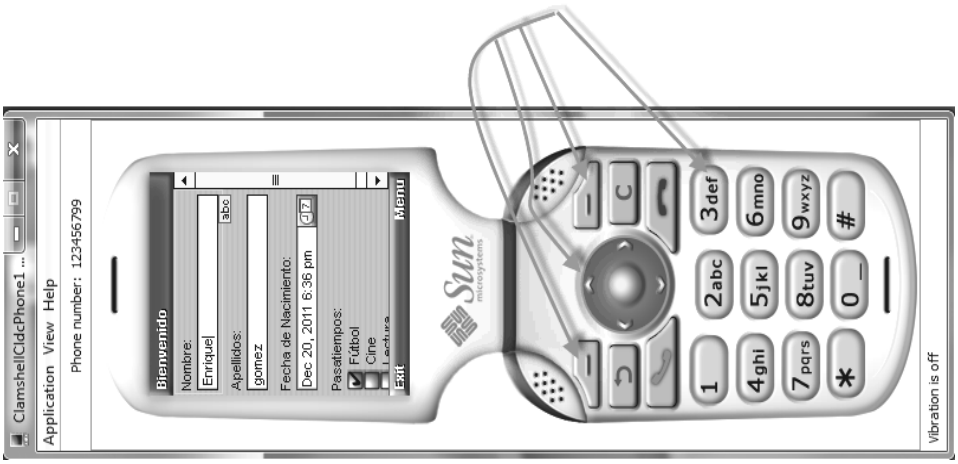


Figura 13.8 Ejecución de su primera aplicación móvil.

En el manejo de la aplicación que muestra la figura 13.08, debe considerarse que el usuario tiene que utilizar los botones del centro del emulador del teléfono para desplazarse entre las opciones del mismo. Por ejemplo, las teclas de la esquina superior izquierda y derecha permiten manejar la opción de Menú y Exit, respectivamente. Asimismo, mediante la opción de Menú se puede seleccionar el uso del teclado en modo numérico o alfabetico, así como las teclas del centro que permiten ir hacia arriba, abajo o hacia la derecha o izquierda.

Plugin (complementos) en NetBeans para móviles

Los plugin o complementos son herramientas o utilitarios de software desarrollados por los fabricantes de dispositivos para el diseño de aplicaciones. Las empresas que desarrollan tecnologías móviles, tales como Nokia, Samsung, Sony – Ericsson, RIM, entre otros, desarrollan sus propios framework para que sean utilizados en el desarrollo de aplicaciones para estos dispositivos. Dichos framework pueden trabajar de manera independiente o empotrada en algún otro software como NetBeans, Eclipse, .Net, entre otros.

Como referente se utilizará la historia de Nokia S60 c. Puede argumentarse que la plataforma S60 es una plataforma que utiliza el sistema operativo Symbian, siendo una de las que lideran el mercado de Smartphone conjuntamente con Android e IPhone. S60 es tecnología Nokia y licenciada por Lenovo, LG, Electronics, Panasonic y Samsung. Para muchos el S60 es memoria de tiempos pasados y Symbian Foundation (creadora ahora del sistema operativo) hace grandes esfuerzos por posicionar la marca.

Sin más preámbulos y con más acción vamos a instalar el complemento del S60 en el IDE NetBeans. Para ello descargue el archivo **nS60_jme_sdk_3rd_e_FP1.zip** desde http://www.developer.nokia.com/info/sw.nokia.com/id/6e772b17-604b-4081-999c-31f1f0dc2dbb/S60_Platform_SDKs_for_Symbian_OS_for_Java.html.

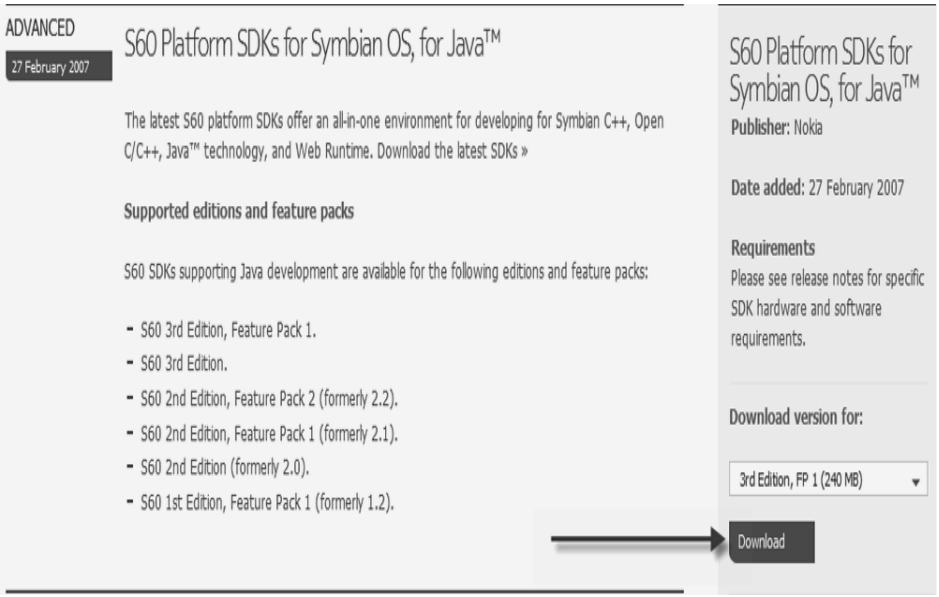


Figura 13.9 Descarga del Plugin.

Una vez descargado proceda a abrir el archivo. En éste se encuentra el archivo *S60_3rd Ed_ MIDP_SDK_FP1_InstallationGuide_1.01.pdf* que le permitirá tener una idea de la instalación del plugin. A pesar de que se creó para una versión anterior de NetBeans, en cierto modo aplica.

Una vez que haya leído el documento que se le indicó (aunque más adelante se verá aquí cómo habilitarlo en NetBeans) busque *setup.exe* en el archivo comprimido y ejecútelo. Inmediatamente aparece el instalador que se muestra en la figura 13.10. Instálelo según le va a indicar el mismo.

Figura 13.10 Instalar el Plugin del S60.

Una vez instalado el SDK S60 de Nokia en su sistema, se requiere su habilitación en NetBeans. Para ello ingrese a NetBeans, pulse sobre el menú Herramientas y posteriormente en Plataformas Java. Una vez ahí pulse sobre el botón *Añadir plataforma…*, seleccione la opción *Java ME MIDP Platform Emulator* y pulse el botón *Siguiente*. Busque el directorio donde se instaló el SDK S60 (generalmente en c:\S60…) y se habilitará la posibilidad de instalarlo. Pulse el botón *Finalizar* y listo, tenemos el emulador para usar.

Emulador Nokia S60

Una vez que hemos instalado el emulador y configurado en NetBeans, es hora de probarlo. Para ello vamos a crear una nueva aplicación móvil mediante las siguientes instrucciones:

1. Ingrese a NetBeans y cree un nuevo proyecto para móviles (*Archivo, Proyecto Nuevo…, Java ME, Mobbile Application…*). Pulse el botón *Siguiente*.
2. En la pantalla siguiente dele el nombre de proyecto *mobAppS360* (tal como se observa en la figura 13.11) y pulse el botón *Terminar*.

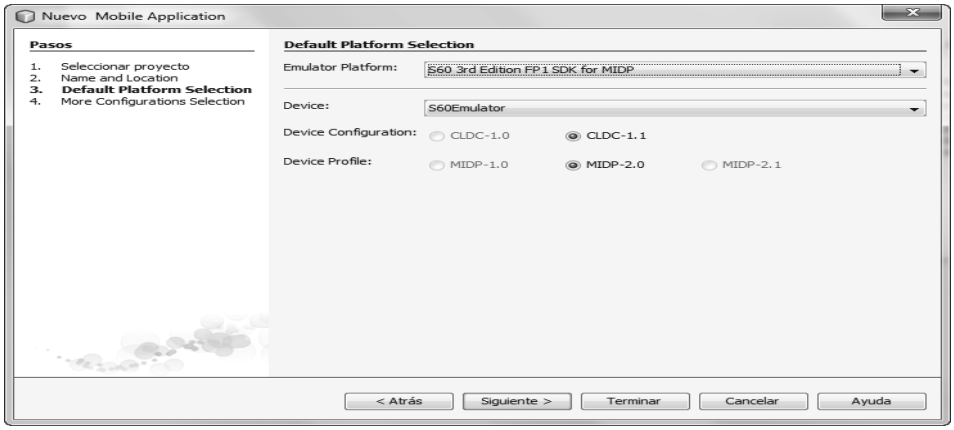


Figura 13.11 Seleccionar la plataforma de desarrollo movil.

3. Si ejecutamos esta aplicación podremos observar que, paulatinamente, se carga el SDK, simula el sistema operativo Symbian, y hace la carga de todos los utilitarios que quizá traerá instalado el dispositivo real. La figura 3.12 muestra el proceso de carga del SDK, para luego habilitar las aplicaciones de fábrica que trae el dispositivo y las que se desarrollen por terceros (la que usted desarrolle).

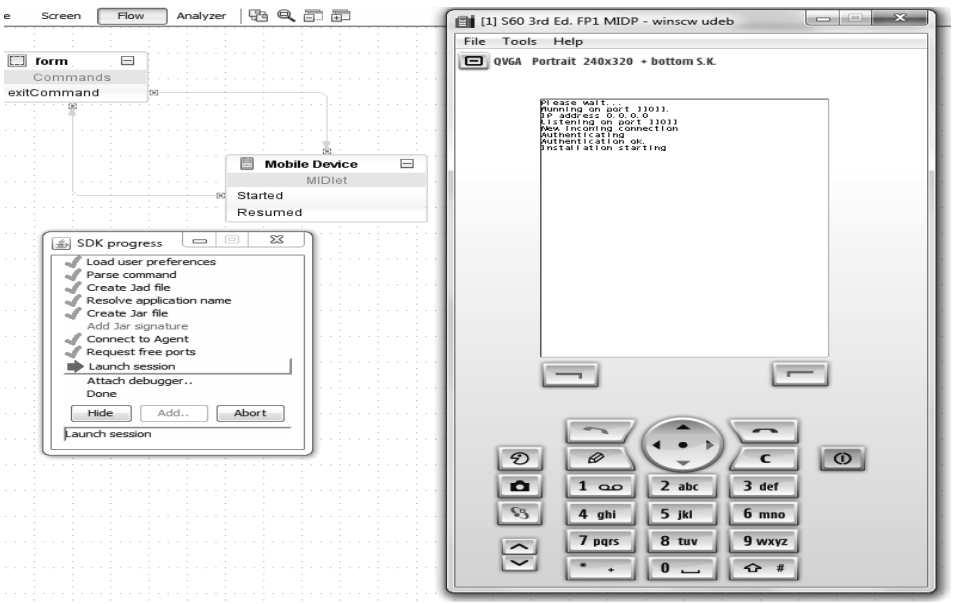


Figura 13.12 Ejecutar su primera aplicación móvil con el S60.

4. Una vez cargado el emulador, podrá tener una idea de este emulador de dispositivo. La figura 13.13 lo muestra.

Figura 13.13 Entorno funcional del emulador S60.

5. Ahora se crearán dos formularios donde se muestre, en el primero, su información personal y una pequeña calculadora en el segundo. Proceda a crear su primer formulario, tal y como se muestra en la figura 13.14. Este formulario se denominará *DatosPersonales*. Una vez creado pulse sobre el mismo y en el menú de contexto que aparece seleccione propiedades, y ahí elija *Title* y escriba *Datos Personales*. Ese será el título del formulario que se mostrará cuando se ejecute.

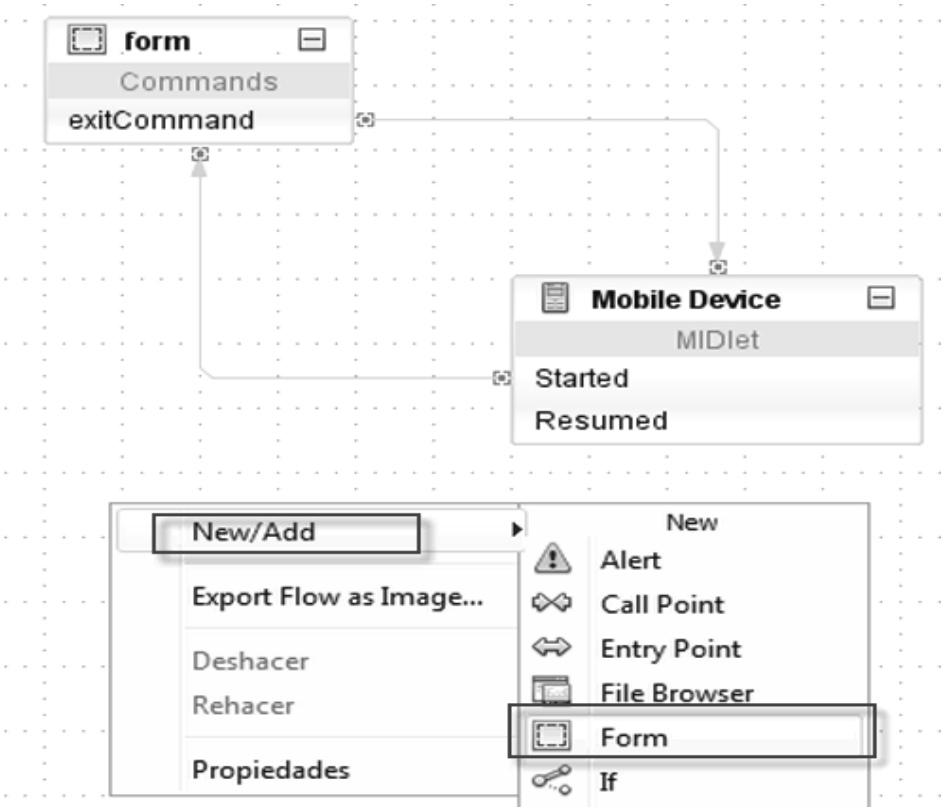


Figura 13.14 Agregando un formulario a su aplicación móvil.

6. Una vez creado el formulario proceda a crear otro denominado *Calculadora*. La figura 13.15 muestra los objetos que se han aplicado a todos los formularios. Recuerde que mediante clic derecho sobre el formulario se pueden agregar componentes al mismo.

7. Observe en la figura 13.15 que en los formularios se colocan principalmente los botones de comando (backCommand e itemCommand). En el caso del formulario *Calculadora* encontramos los itemCommand *sumar* e *igual* y un backCommand que es el que permite retornar al menú principal de la aplicación (en este caso sólo debe arrastrar desde este backCommand hasta el formulario *form* para que automáticamente se produzca el regreso de *Calculadora* a *form*). En el formulario *DatosPersonales* observe que sólo hay un backCommand para regresar a *form*. Todo esto se realiza en la pestaña o cejilla *Flow*.

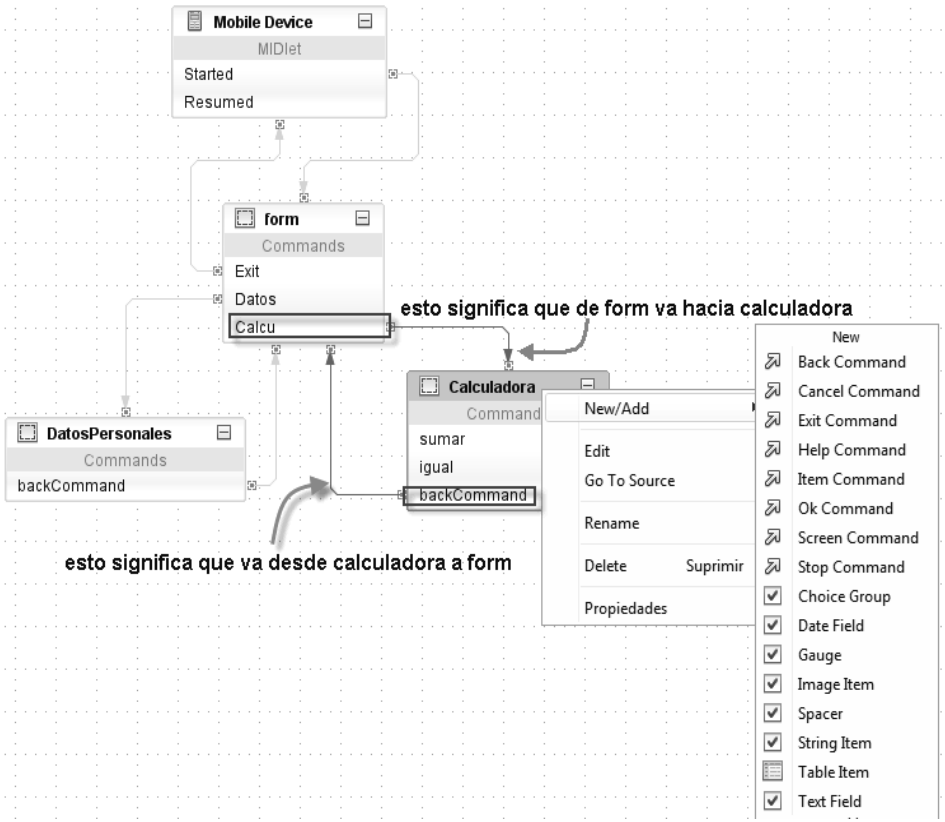


Figura 13.15 Agregando componenentes a su aplicación móvil.

8. Ahora proceda a diseñar la interface que tendrá el formulario *DatosPersonales*. Para ello pulse la cejilla *Screen* y proceda a colocar los objetos según se indica en la figura 13.16. Los detalles son los siguientes:
    - Nombre y apellidos son objetos tipo *Text Field.*
    - Fecha de Nacimiento es un objeto de tipo *Date Field*.
    - Fotografía es un objeto de tipo *Image Item*.
9. Basta con que pulse dos veces sobre el título del objeto para cambiar el texto del mismo, o pulse sobre cualquiera de ellos, seleccione propiedades y ahí busque la propiedad *Label* y escriba lo que requiera. Observe que para el objeto *Image Item* se puede seleccionar el botón superior derecho para asignar una imagen al objeto (en la figura 13.17 se aprecia dónde se muestra el procedimiento para cargar una imagen).

Figura 13.16 Configuración del formulario DatosPersonales.

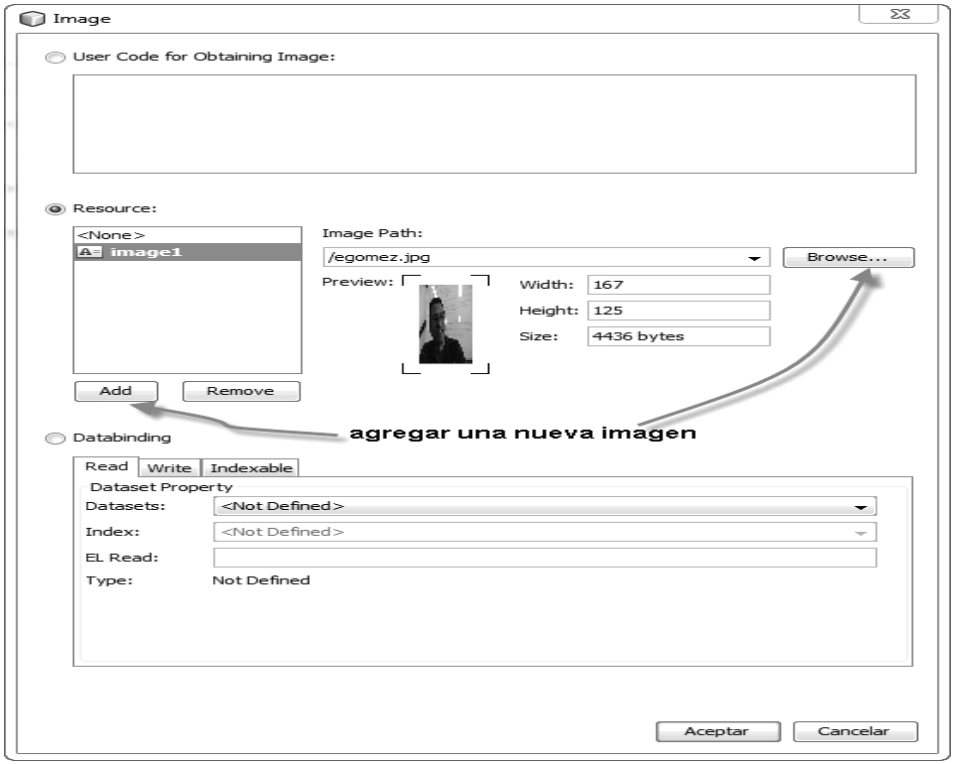


Figura 13.17 Agregando una imagen.

10. Al final su formulario tendrá un aspecto similar al de la figura 13.18.

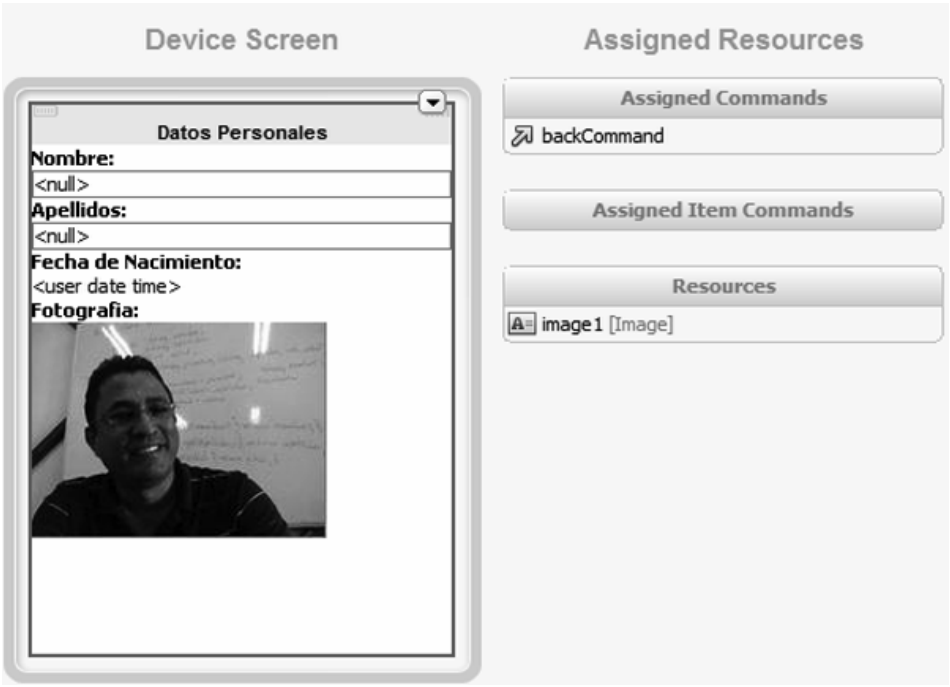


Figura 13.18 Formulario DatosPersonales listo.

11. Ahora procedamos a configurar el formulario *Calcu,* de tal manera que se muestre como en la figura 13.19.

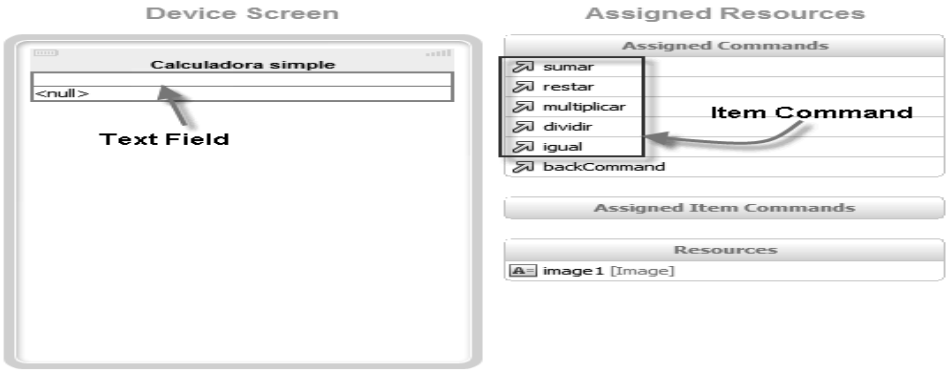


Figura 13.19 Formulario Calculadora listo.

12. Por último, la interface del formulario *form* inicial semejará la figura 13.20.

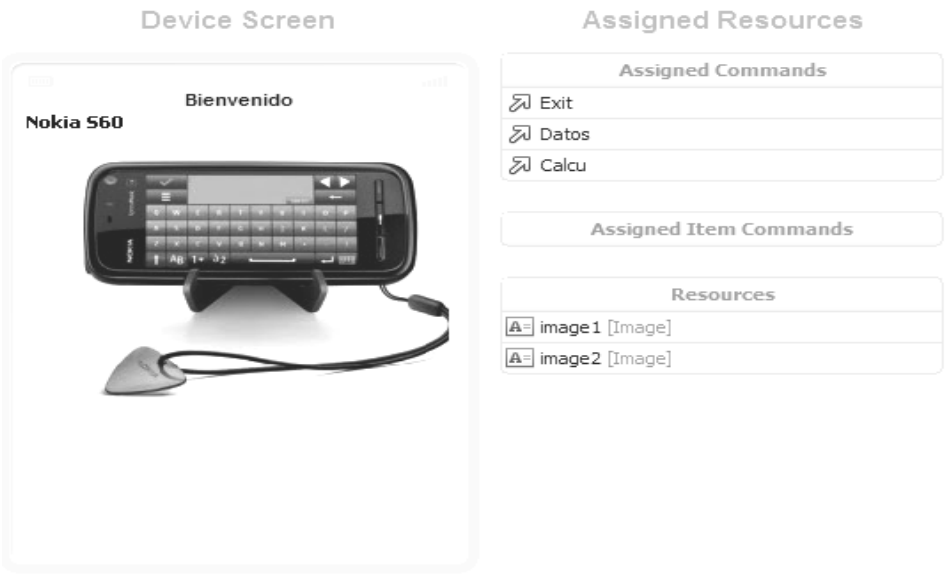


Figura 13.20 Formulario principal listo.

Observe que el título Nokia S60 no es más que un objeto Image Item. Busque una imagen alusiva al formulario principal de esta aplicación móvil.

13. Pulse sobre la pestaña Screen y seleccione el formulario Calculadora. Vamos a desarrollar el código fuente para que pueda funcionar como una calculadora sencilla.
14. Una vez en el formulario calculadora posiciónese sobre el command sumar y pulsando con el botón derecho del mouse sobre el mismo seleccione la opción Go To Source. Esta opción le permitirá escribir código para ese botón. El código que debe escribir es el siguiente:

CÓDIGO PARA EL BOTÓN SUMAR
operador = 1; //para saber que el operador es de suma
operacion(operador); //llama a un procedimiento para efectuar la operación.
txtPantalla.setString(""); //limpia la caja de texto para que no quede ningún valor.

15. Vuelva a la pestaña *Screen*, ubique el botón *restar* y proceda de la misma manera que en el punto 12. El código para este objeto será:

CÓDIGO PARA EL BOTÓN RESTAR
operador = 2; //para saber que el operador es de resta
operacion(operador); //llama a un procedimiento para efectuar la operación.
txtPantalla.setString(""); //limpia la caja de texto para que no quede ningún valor.

16. Para el botón *multiplicar* el código será:

CÓDIGO PARA EL BOTÓN MULTIPLICAR
operador = 3; //para saber que el operador es de multiplicación.
operacion(operador); //llama a un procedimiento para efectuar la operación.

17. Para el botón dividir el código será:

CÓDIGO PARA EL BOTÓN DIVIDIR
operador = 4; //para saber que el operador es de suma
operacion(operador); //llama a un procedimiento para efectuar la operación.

18. Para el botón igual, escribamos el siguiente código:

CÓDIGO PARA EL BOTÓN IGUAL
switch(operador){ //recibe un valor para determinar que tipo de operación es.
case 1: txtPantalla.setString(String.valueOf(suma));
case 2: txtPantalla.setString(String.valueOf(resta));
case 3: multiplica*=Integer.parseInt(txtPantalla.getString()) ; txtPantalla.setString(String.valueOf(multiplica)); break;

case 4: divide = divide/Integer.parseInt(txtPantalla.getString());
//((divide/Integer.parseInt(txtPantalla.getString())) > 0) ?
//(divide/Integer.parseInt(txtPantalla.getString())) : 0;
txtPantalla.setString(String.valueOf(divide));

}

19. Ahora necesita crear el código general de declaraciones de variables y el procedimiento operación que es el que resuelve las operaciones de su calculadora. El siguiente código es el que cumple esta función (lo subrayado no debe escribirlo). Ubíquese al inicio del código en el editor y escriba este código:

CÓDIGO GENERAL DE LA APLICACIÓN
public class HelloMIDlet extends MIDlet implements CommandListener {
static int suma,resta,multiplica,divide,operador;
private boolean midletPaused = false;
void operacion(int tipoOperacion) {
switch(tipoOperacion) {
case 1: suma+= Integer.parseInt(txtPantalla.getString());break;
case 2: resta-= Integer.parseInt(txtPantalla.getString());break;
case 3: multiplica = Integer.parseInt(txtPantalla.getString());break;
case 4: divide = Integer.parseInt(txtPantalla.getString());txtPantalla.setString("");break;
}}

20. Ahora podemos ejecutar su aplicación y en la figura 13.21 es posible ver el formulario principal.

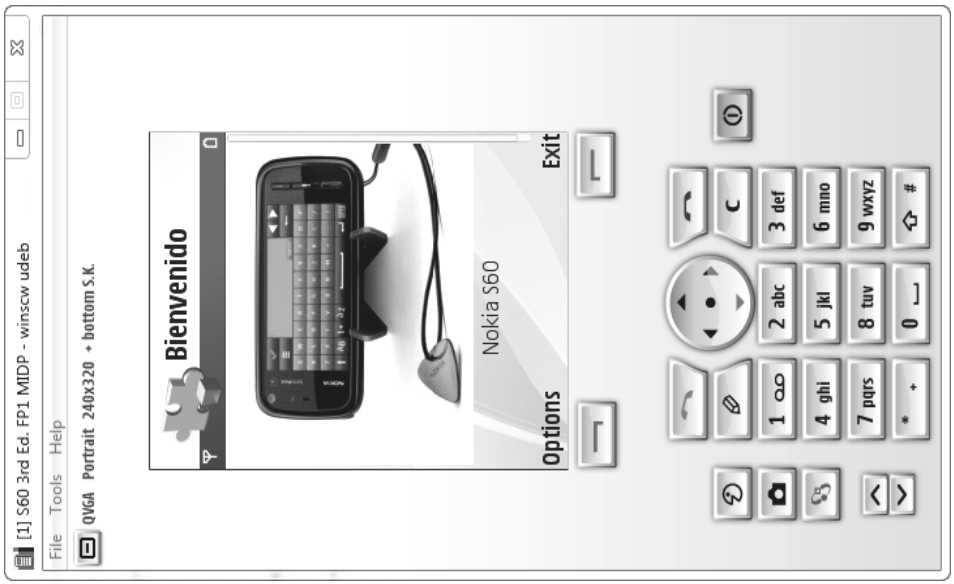


Figura 13.21 Formulario principal ejecutándose.

21. Si pulsamos sobre el botón que está debajo de *Options* podremos ver que se habilita el menú que permite ejecutar los dos formularios que posee la aplicación. La figura 13.22 muestra estas opciones.

Figura 13.22 Opciones de menú del formulario principal.

22. Si seleccionamos la opción *Datos Personales* tendremos acceso al formulario respectivo, que se mostrará como en la figura 13.23.

Figura 13.23 Ejecutándose el formulario Datos Personales.

23. Pulsando el botón que está debajo de la leyenda *Back* podemos retornar al menú principal. Si ahí optamos por la opción de calculadora (vea la figura 13.22) se ejecutaría el formulario respectivo, de manera similar a lo que se aprecia en la figura 13.24.

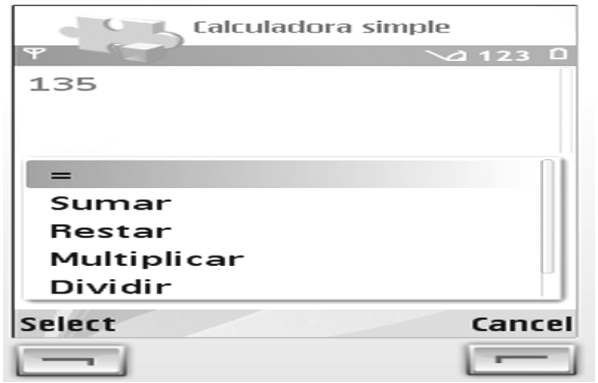


Figura 13.24 Ejecutándose el formulario Calculadora.

Actividades para el lector

Implemente el código en este emulador para las operaciones Potencia, Raíz cuadrada, Factorial, entre otras, que le ayuden a asimilar el tema.

Emulador Sony Ericsson

Ahora vamos a probar otro emulador que trabaja con NetBeans. Se trata del jdk () de Sony-Ericsson que podrá descargar desde

http://developer.sonyericsson.com/wportal/devworld/search-downloads/docstools/sdk?cc=gb&lc=en.

Una vez descargado proceda a instalar el archivo *semc_java_me_cldc_sdk.2-5-0-6.exe* que viene incluido en el comprimido. Ya instalado proceda a crear la plataforma Java, tal como lo hicimos con Nokia S-60. La figura 13.25 muestra parte de este proceso.

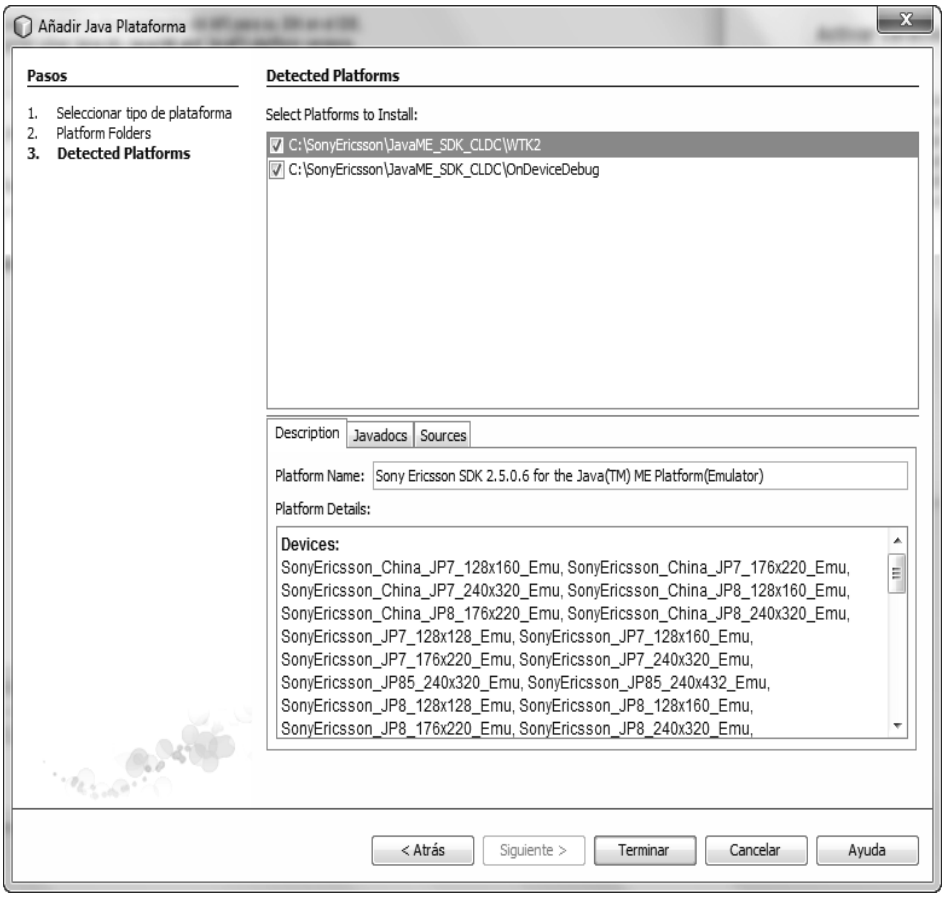


Figura 13.25 Instalación de SDK para Sony-Ericsson.

Ahora podríamos probar su aplicación desarrollada para el S-60 de Nokia con alguno de los modelos de Sony-Ericsson. Para ello ejecute las siguientes instrucciones:

24. Pulse *Establecer configuración del proyecto* del menú *Ejecutar* y ahí la opción *Personalizar*. Se mostrará una figura como la 13.26.
25. En la pantalla de configuración del dispositivo (vea la figura 13.26) seleccione el dispositivo *SonyEricsson_W950* y pulse el botón *Aceptar.* Proceda a ejecutarlo. Algo similar a la figura 13.27 aparecerá.

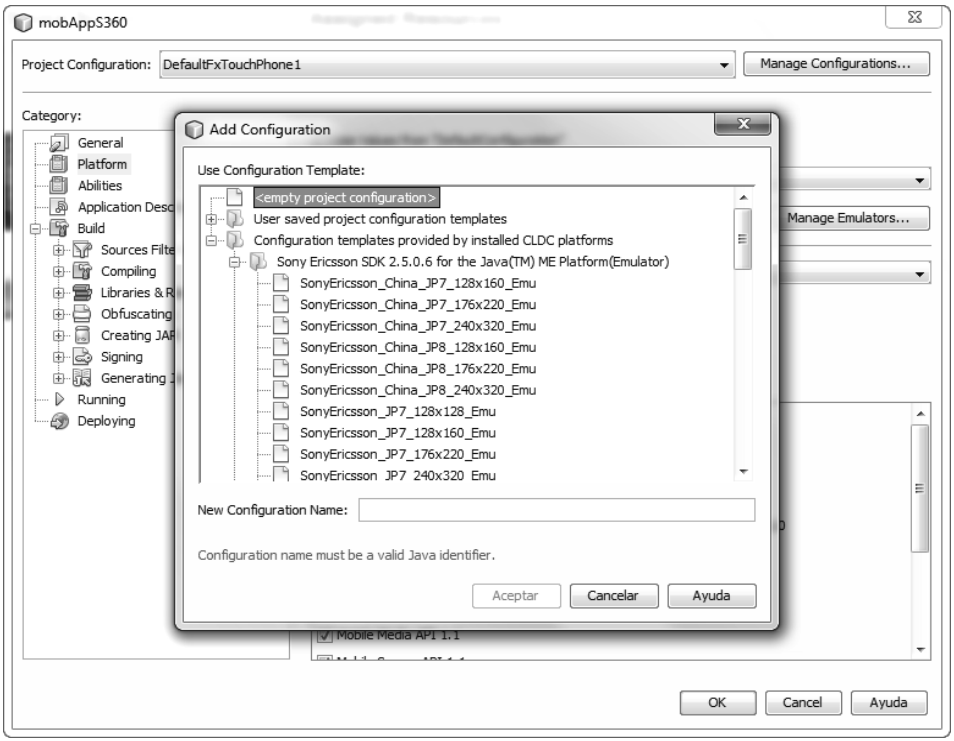


Figura 13.26 Selección de otro dispositivo.

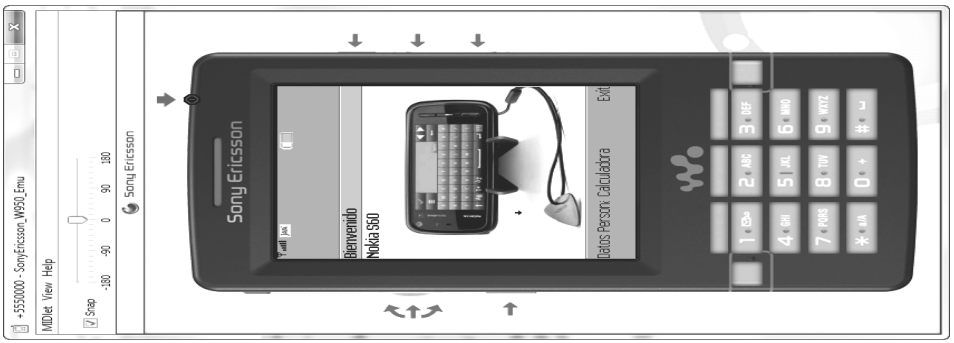


Figura 13.27 Ejecutando su aplicación móvil con otro dispositivo.

26. Ahora proceda a probar su funcionalidad.
27. Por último, probaremos su aplicación móvil con un dispositivo que incluye el SDK 3.0 de NetBeans. Para ello proceda según los puntos 22 y 23 de este ejercicio, pero seleccione ahora la plataforma y dispositivo que se indican en la figura 13.28.

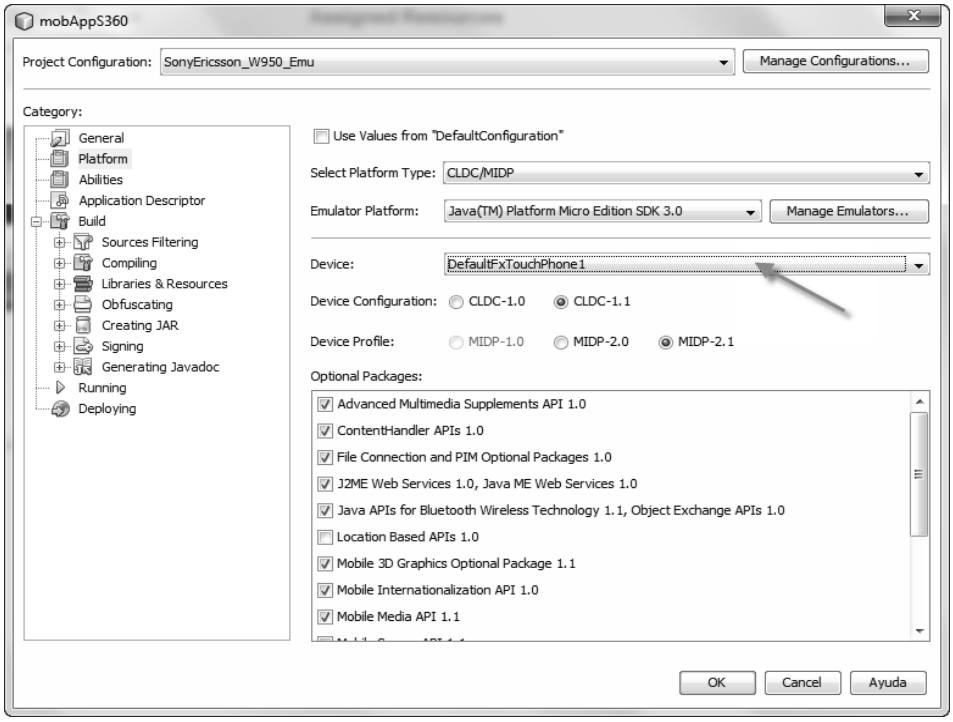


Figura 13.28 Selección de otro dispositivo.

28. Al ejecutarse se mostrará algo similar al que presenta la figura 13.29.

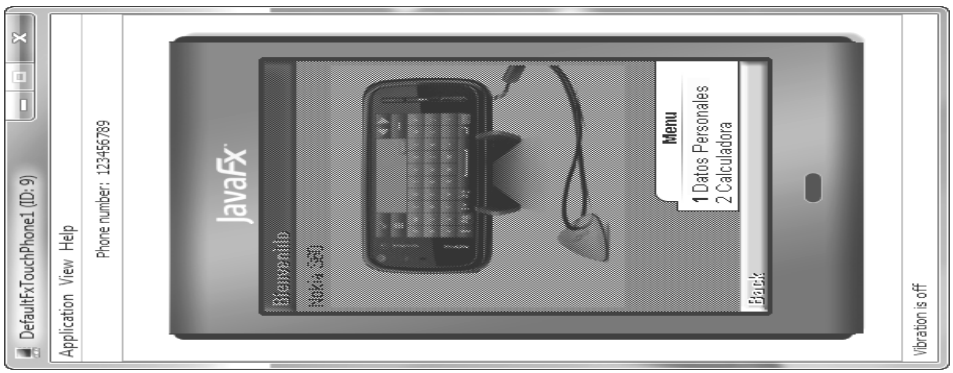


Figura 13.29 Ejecutando su aplicación móvil con dispositivo del SDK 3.0.

RMS (Record Management System)

Cada vez que utilizamos un dispositivo quizás querramos guardar los datos en el mismo. Por ejemplo, los datos de un contacto en un listado telefónico, los puntos obtenidos en un juego o los recordatorios de fechas importantes. Esa información que deseamos se mantenga es lo que conocemos como persistencia de los datos. Por fortuna, en J2ME se tiene un sistema de gestión de registros denominado RMS (Record Management System) que ayuda a que esa persistencia de los datos sea posible.

RMS tiene el objetivo de almacenar la información en registros (denominados *record*) en bases de datos especializadas llamadas RecordStore. Un registro o record se almacena en esta base de datos con un identificador único que generalmente es su posición dentro del conjunto de registros. Mediante ese identificador es posible ubicarlo en la colección y poder realizar cualquier operación válida sobre el mismo, esto es modificarlo o eliminarlo.

RMS está contenido en la librería *javax.microedition.rms* e incluye una serie de clases e interfaces que hace posible la gestion de registros. Esta gestión incluye capacidades como agregar, modificar, eliminar registros, obtener registros, determinar cuántos registros contiene el recordstore, entre otras funcionalidades.

RecordStore

Un RecordStore es una clase que permite la gestión de registros en un dispositivo celular. Consta de un conjunto de registros, siendo un registro un array de datos de longitud variable. La tabla 13.4 muestra los métodos esenciales de la clase *RecordStore*:

Tabla 13.4 Métodos de la clase RecordStore

MÉTODO	DESCRIPCIÓN
openRecordStore	Permite la apertura del almacén de registros.
closeRecordStore	Permite el cierre del almacén de registros.
deletedRecordStore	Elimina el almacén de registros.
getName	Obtiene el nombre del almacén de registros.
getNumRecords	Obtiene el número de registros contenidos en el almacén de registros.
addRecord	Agrega un registro en el almacén de registros.
getRecord	Permite recuperar un registro determinado (según su identificador) desde un almacén de registros.
deleteRecord	Elimina un registro según su identificador.
enumerateRecord	Obtiene un enumerador del almacén de registros, permitiendo su recorrido, sea hacia adelante o hacia atrás. Similar a una lista doblemente enlazada.

También es importante conocer las interfaces que se pueden utilizar en una aplicación desarrollada para dispositivos móviles. Estas interfaces se muestran en la tabla 13.5.

Tabla 13.5 Interfaces de la clase RecordStore

INTERFACE	DESCRIPCIÓN
RecordEnumeration	Permite mantener una secuencia lógica de los identificadores almacenados en un RecordStore.
RecordComparator	Permite la comparación entre registros. Con esta interfaz es posible realizar la ordenación de los record en el almacen de registros (RecordStore).
RecordFilters	Permite establecer filtros para obtener ciertos registros de la colección.
RecordListener	Mediante *RecordListener* se pueden monitorizar cambios en el RecordStore, tal como agregación, actualización o eliminación de registros.

openRecordStore

La apertura de un almacen de registros se lleva a cabo mediante el método openRecordStore. Con ello podemos tener acceso a cada registro que esté almacenado en el RecordStore. La sintaxis para aplicar este método es el siguiente:

```
RecordStore rs = RecordStore.openRecordStore(“Libreta”,true);
```

rs es la variable que referenciará el almacen de datos, permitiendo el flujo hacia y desde el RecordStore cuyo nombre es Libreta. La indicación de true indica que si dicho almacen no existe se permita su creación inicial. Cabe mencionar que el nombre no debe exceder los 32 caracteres, de lo contrario se producirá una excepción al crear el openRecordStore. Para cerrar un RecordStore abierto se procede según la siguiente instrucción:

```
rs.closeRecordStore();
```

addRecord

Para agregar un nuevo registro se utiliza el método *addRecord*. Este método tiene la siguiente sintaxis:

```
rs.addRecord(outputRecord, 0, outputRecord.length);
```

outputRecord es en realidad un objeto de tipo byte (byte[] outputRecord;) que contiene los bytes correspondientes al registro. Este registro lógicamente contendrá datos de tipos primitivos que pueden ser diferentes: String, int, long, boolean, entre otros. El cero (0) representa el inicio del registro y outputRecord.length la longitud total del registro (es decir, los bytes que puedan tener un registro de n cantidad de

bytes String, n cantidad de bytes numérico, n cantidad de bytes tipo fecha, entre otros).

setRecord

Para modificar un registro se utiliza el método *setRecord*. Para llevar a cabo este método debe conocerse el identificador del registro a modificar. Este método tiene la siguiente sintaxis:

```
recordstore.setRecord(RecordId, outputRecord, 0, outputRecord.length);
```

En este caso se utiliza setRecord para la modificación, siendo *RecordId* el identificador del registro a modificar y 0 (cero) y outputRecord.length el inicio y final del tamaño del registro.

deleteRecord

Para eliminar un registro se utiliza el método *deleteRecord*. Para llevar a cabo este método debe conocerse el identificador del registro a eliminar. Este método tiene la siguiente sintaxis:

```
recordstore.deleteRecord(RecordId);
```

Como se muestra en la línea anterior, debe conocerse el identificador del registro para proceder a su eliminación.

getRecord

Para buscar un registro se utiliza el método *getRecord*. Para llevar a cabo este método debe conocerse el identificador del registro a buscar. Este método tiene la siguiente sintaxis:

```
recordstore.getRecord(RecordId,registro,0);
```

Como se muestra en la línea anterior, debe conocerse el identificador del registro para proceder a buscarlo (RecordId) y también la estructura temporal donde se buscará (registro).

RecordEnumeration

Enumerar los registros que están guardados en un almacén significa establecer un flujo de los mismos que es posible recorrer. Mediante la enumeración podemos obtener cada registro y mostrarlos a través de un objeto que lo permita, tal como un itemTable o una lista. La sintaxis de uso de RecordEnumeration es la siguiente:

```
RecordEnumeration registros = rs.enumerateRecords(null, null, false)
```

Como se muestra en la línea anterior, registros es el enumerador de registros que se obtienen del RecordStore *rs*. El primer parámetro le indica al enumerador de registros que no posee un objeto *RecordFilter* con lo cual se obtendrán todos los datos del RecordStore. Si hubiera algún filtrado en este parámetro entonces sólo se obtendrían los registros que cumplan con el patrón del filtro establecido. El segundo parámetro es el objeto *RecordComparator* que permite establecer el orden en que el

enumerador enumerará los registros (pueden ser un ordenamiento por algún campo del RecordStore). El último parámetro, el cual es de tipo lógico, determinará si la enumeración será actualizada en el almacén. Es decir, si existen inserciones y eliminaciones éstas deberán ser reflejadas también en el almacén de datos.

La interfaz *RecordEnumeration* posee métodos que son esenciales para la navegacion entre los registros que se obtienen del RecordStore. Por ejemplo, el método *hasNextElement* permite recorrer la colección de registros; *nextRecord* ir al próximo registro o *nextRecordId* al próximo identificador de registros. Existe una gran versatilidad en el uso del filtrado (RecordFilter) y ordenamiento (RecordComparator) para esta interfaz para obtener registros específicos.

En el caso de *RecordFilter* se define un método único que permite definir un criterio para la selección de registros. Es decir, se enumerarán aquellos registros que cumplan con una determinada condición establecida en este RecordFilter. Devuelve true si se cumple el patrón del filtrado o false en caso contrario. *RecordComparator* proporciona, por su parte, una forma sencilla de clasificar los datos del RecordStore. Aquí se manejan tres constantes básicas para el ordenamiento: *Precedes*, *Follows* y *Equivalent*. En el caso de *Precedes* devuelve true si el registro1 se ha creado antes de registro2. Con *Follows* si registro1 se ha creado después y *Equivalen* si han sido creados en forma simultánea.

El siguiente código muestra un recorrido secuencial de los registros de un RecordStore que ha sido enumerado:

RECORDENUMERATION

```
RecordEnumeration registros = null;
try{
while(registros.hasNextElement()){ //mientras se encuentre un próximo elemento
   int identificador = registros.getRecordId(); //se obtiene el identificador.
   System.out.println(identificador);  //se muestra el identificador de registro.
}catch (Exception ex){
Ex.printStackTrace();
}
```

Reglas para la creación de RecordStore

Es importante tener en cuenta ciertas reglas para la creación de RecordStore en un MIDlet Java de una aplicación para dispositivo móvil. Algunas de estas reglas son:

- El nombre de un RecordStore no debe superar los 32 caracteres. También debe considerarse que el nombre es sensible a minúsculas y mayúsculas.
- Los RecordStore que se hayan creado en una suite de MIDlet se almacenan en un mismo espacio de nombres, por lo que se pueden compartir sus contenidos. Los nombres de estos RecorStore deben ser únicos.

- Los RecordStore creados en una aplicación de un MIDlet no pueden ser compartidos por MIDlet externos.

Luego de esta teoría vamos a proceder a desarrollar una sencilla aplicación que utilice RMS. Para ello imagine que tiene que almacenar datos de clientes en su celular y que sólo necesita guardar su identificador (que será automático), el nombre y la dirección del cliente. Proceda a crear su aplicación RMS.

1. Inicie creando un proyecto para dispositivo móvil (observe la figura 13.2 y 13.03). El nombre de esta aplicación será *Clientes*.
2. El MIDlet creado para esta aplicación se muestra en la figura 13.30.

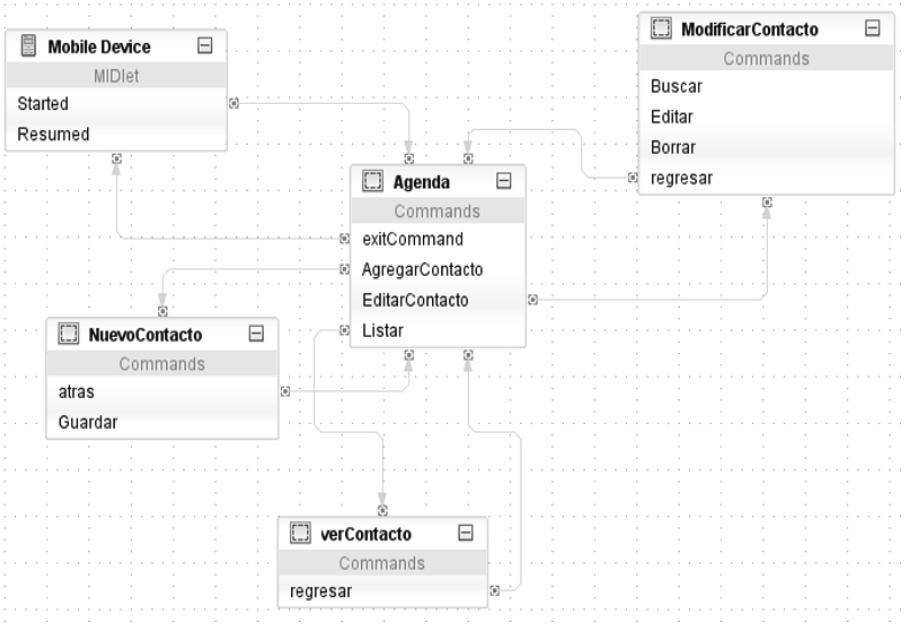


Figura 13.30 MIDlet de su aplicación móvil Clientes.

Como podrá observar en la figura 13.30 existe el MIDlet *Mobile Device* que básicamente inicia la aplicación con el formulario *Agenda*. Este formulario posee varios Item Command (*agregarContacto*, *editarContacto* y *Listar*). Asimismo, tiene un exitCommand. Este formulario será el que permita ejecutar los otros tres formularios.

Otro formulario es *NuevoContacto*. Este formulario es donde agregaremos los registros que estarán almacenados en el RecordStore. Incluye un backCommand (*atrás*) y un Item Command (*Guardar*).

ModificarContacto es un formulario que utilizaremos para modificar los datos de un registro determinado o eliminar el registro. Está conformado por tres Item Command (*Buscar*, *Editar* y *Borrar*). También posee un back command (*regresar*) que le permite volver al menú principal (formulario *Agenda*).

Por último, se tiene el formulario *verContacto* que básicamente presenta un listado de los registros almacenados en el RecordStore. Sólo cuenta con un backCommand (*regresar*) que le permite volver al menú principal (formulario *Agenda*).

Ejemplo de aplicación

El formulario Agenda

3. El contenido de este formulario (pestaña *Screen* del MIDlet) se muestra en la figura 13.31.

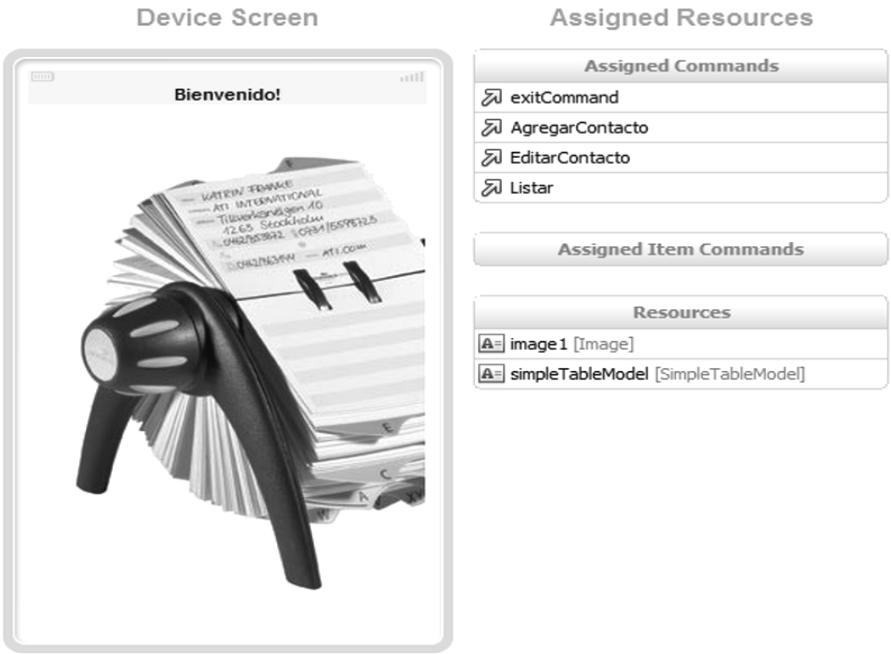


Figura 13.31 Formulario Agenda.

Observe que solamente se ha agregado una imagen al formulario (recuerde cómo agregar imágenes remitiéndose a la figura 13.17). Busque una imagen alusiva a esta aplicación. Observe también los recursos asignados (Assigned Resources), en ellos podrá ver que existen los diferentes Item Command que conforman lo que es el menú principal.

El formulario NuevoContacto

4. El contenido de este formulario (pestaña *Screen* del MIDlet) se muestra en la figura 13.32.

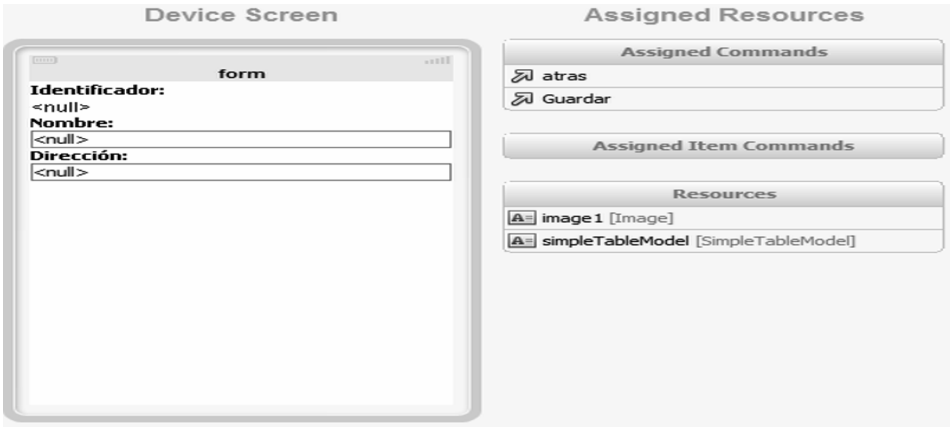


Figura 13.32 Formulario NuevoContacto.

Observe que en este formulario se encuentran tres objetos: String Item (*Identificador)* y dos Text Field (*Nombre* y *Dirección*). Si pulsa con el botón derecho del mouse sobre cada uno de ellos podrá cambiar su nombre seleccionando *Rename* en el menú de contexto que aparece. Los nombres de cada uno serán *lblIdentificador, txtNombre* y *txtDireccion*, respectivamente. Si pulsa de nuevo sobre cada control, podrá seleccionar *Propiedades* y ahí cambiar las propiedades *Label* de cada uno para poner sus títulos.

5. Seleccione la pestaña *Flow* (observe la figura 13.30) y escoja el formulario *NuevoContacto*. Ahí observe que existe la opción *Guardar*. Pulse con el botón derecho del mouse y seleccione *Go To Source*. Se abrirá enseguida el editor de código. Escriba el procedimiento *guardarRegistro()*, tal como se indica en las siguientes líneas (observe que se coloca debajo de la leyenda *//write pre-action user code here*).

```
// write pre-action user code here
   guardarRegistro();
```

6. El compilador le dará un mensaje de error dado que dicho procedimiento no existe. Por tanto, si pulsa con el botón izquierdo sobre el indicador de error le aparecerá la leyenda *Crear método guardarRegistro() en hello.HelloMIDlet*, lo cual le ubicará al final del editor de código para que lo implemente. Una vez ahí deberá escribir el código del procedimiento, tal como se indica a continuación.

PROCEDIMIENTO GUARDARREGISTRO()
void guardarRegistro(){
try {
//el RecordStore Libreta se abre y se referencia mediante recordstore. //si no existe entonces lo crea, según el seguno parámetro (true)

PROCEDIMIENTO GUARDARREGISTRO()
recordstore = RecordStore.openRecordStore("Libreta", true);

//outputRecord es una variable de tipo byte y se utiliza para almacenar los datos
//de salida.
byte[] outputRecord;

//buffer de salida de 32 bytes que actua como una matriz de bytes. Ahí debemos
//escribir los datos.
ByteArrayOutputStream outputStream = new ByteArrayOutputStream();

//se utiliza para leer y almacenar temporalmente los datos almacenados en el
//buffer de salida outputStream.
DataOutputStream outputDataStream = new DataOutputStream(outputStream);

//se almacena en outputStream el contenido del objeto lblIdentificador.
outputDataStream.writeUTF(lblIdentificador.getText());

//se almacena en outputStream el contenido del objeto txtNombre
outputDataStream.writeUTF(txtNombre.getString());

//se almacena en outputStream el contenido del objeto txtDireccion.
outputDataStream.writeUTF(txtDireccion.getString());

//realiza el vaciado de la secuencia de datos capturados de los objetos en
//outputStream
outputDataStream.flush();

//se almacena en el objeto tipo arreglo de bytes la secuencia de datos de o
outputRecord = outputStream.toByteArray();

//almacena en el RecordStore (referenciado por recordstore) los datos
//guardados en outputRecord, desde la posición 0 hasta la longitud de cada reg.
recordstore.addRecord(outputRecord, 0, outputRecord.length);

//reinicia el flujo de lectura en outputStream.
outputStream.reset();

outputStream.close();
outputDataStream.close();
RecordId=recordstore.getNumRecords()+1;
recordstore.closeRecordStore();
lblIdentificador.setText(String.valueOf(RecordId));
txtNombre.setString("");
txtDireccion.setString("");
}
catch (Exception error) {
alerta = new Alert("Error : ",

PROCEDIMIENTO GUARDARREGISTRO()
error.toString(), null, AlertType.WARNING);
alerta.setTimeout(Alert.FOREVER);
}
}

El formulario verContacto

Este formulario tiene el propósito de mostrar los contactos que están almacenados en el RecordStore. El contenido de este formulario (pestaña *Screen* del MIDlet) se muestra en la figura 13.33.

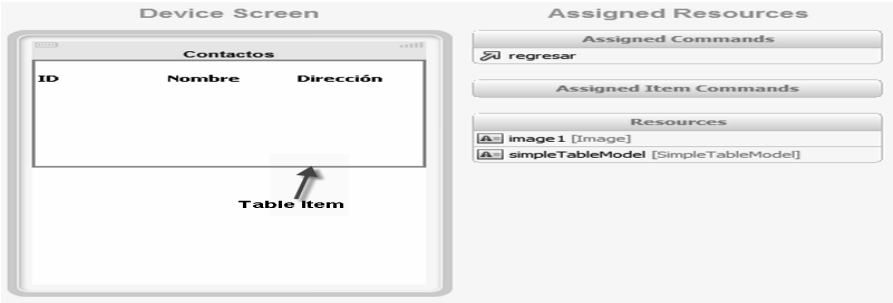


Figura 13.33 Formulario verContacto.

Observe que este formulario sólo consta de un objeto Table Item. El recurso *regresar* no es más que un backCommand que servirá para retornar al menú principal. Las propiedades de la tabla se pueden apreciar en la figura 13.34.

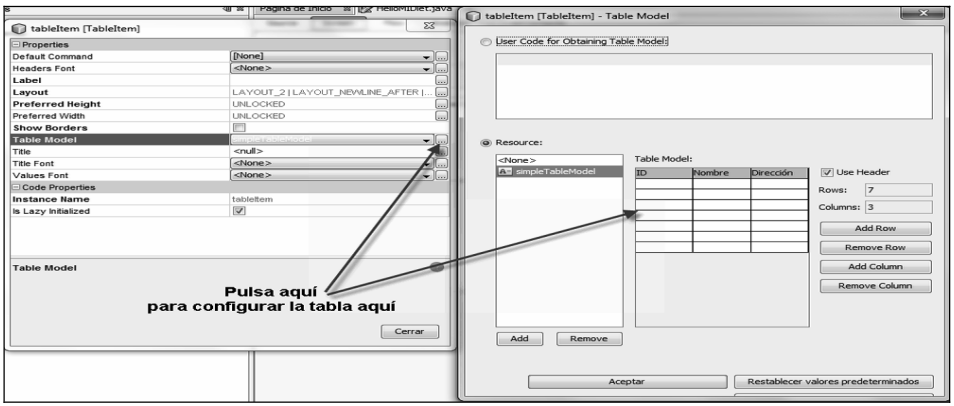


Figura 13.33 Propiedades de la tabla del formulario verContacto.

Observe que si se pulsa sobre la opción *Table Model* podemos configurar el formato de la tabla: agregar filas (Add Row), eliminarlas (Remove Row) o agregar columnas (Add Column), eliminarlas (Remove Column). También se puede poner títulos a las cabeceras de las columnas habilitando *Use Header.*

7. Ahora habilite el formulario *ModificarContacto* y seleccionando la opción *Buscar* con el botón derecho del mouse pulse *Go To Source*. Ahí escriba la siguiente instrucción que permitirá ejecutar el procedimiento

```
buscarRegistro();
```

8. El compilador te dará un mensaje de error dado que dicho procedimiento no existe. Por tanto, si pulsa con el botón izquierdo sobre el indicador de error le aparecerá la leyenda *Crear método buscarRegistro() en hello.HelloMIDlet*, lo cual le ubicará al final del editor de código para que lo implemente. Una vez ahi deberá escribir el código del procedimiento, tal como se indica a continuación.

PROCEDIMIENTO BUSCARREGISTRO()
void buscarRegistro() throws RecordStoreException, IOException {

//crea una referencia al RecordStore Libreta
recordstore = RecordStore.openRecordStore("Libreta", false);

Filtro buscar = new Filtro(txtIdentificador.getString());
RecordEnumeration re = recordstore.enumerateRecords(buscar,null,false);
if (re.numRecords() > 0) {

//stream (flujo) de entrada que internamente se gestiona como una matriz de
//bytes
ByteArrayInputStream datosByteEntrada;

//stream (flujo) desde el cual se leen los datos como tipos primitivos. DataInputStream datosEntrada;
byte[] registro = new byte[75];
try{
datosByteEntrada = new ByteArrayInputStream(registro);
datosEntrada = new DataInputStream(datosByteEntrada);
RecordId = Integer.parseInt(txtIdentificador.getString());
label.setText(String.valueOf(re.numRecords()));
recordstore.getRecord(RecordId,registro,0);
txtIdentificador.setString(datosEntrada.readUTF());
txtNom.setString(datosEntrada.readUTF());

PROCEDIMIENTO BUSCARREGISTRO()
txtDir.setString(datosEntrada.readUTF());
datosByteEntrada.reset();
datosByteEntrada.close();}
datosEntrada.close();
catch (Exception error){
alerta = new Alert("Exception : ",
error.toString(), null, AlertType.WARNING);
alerta.setTimeout(Alert.FOREVER); }
registro = null; }
else{
txtNom.setString("Registro No Existe!!!");
txtDir.setString("");
}}

9. Ahora repita los pasos anteriores para escribir el código para el procedimiento *Editar* del formulario *ModificarContacto*. El procedimiento a ejecutar se denominará *modificarRegistro()*, tal como se lee a continuación:

```
modificarRegistro();
```

10. El código correspondiente a *modificarRegistro()* es el siguiente:

PROCEDIMIENTO MODIFICARREGISTRO()
void modificarRegistro() throws RecordStoreNotOpenException, InvalidRecordIDException, IOException, RecordStoreException {
recordstore = RecordStore.openRecordStore("Libreta", true);
byte[] outputRecord;
ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
DataOutputStream outputDataStream = new DataOutputStream(outputStream);
outputDataStream.writeUTF(txtIdentificador.getString());
outputDataStream.writeUTF(txtNom.getString());
outputDataStream.writeUTF(txtDir.getString());
outputDataStream.flush();
outputRecord = outputStream.toByteArray();
RecordId = Integer.parseInt(txtIdentificador.getString());
recordstore.setRecord(RecordId, outputRecord, 0, outputRecord.length);
outputStream.reset();

PROCEDIMIENTO MODIFICARREGISTRO()
outputStream.close();
outputDataStream.close();
recordstore.closeRecordStore();
txtIdentificador.setString("");
txtNom.setString("");
txtDir.setString("");
}

11. Y para este formulario *ModificarContacto* tenemos el último procedimiento denominado *borrarRegistro*, que se determina así:

```
borrarRegistro();
```

12. Y el código correspondiente sería como se escribe a continuación:

PROCEDIMIENTO BORRARRREGISTRO()
void borrarRegistro() throws RecordStoreException{
recordstore = RecordStore.openRecordStore("Libreta", false);
RecordId = Integer.parseInt(txtIdentificador.getString());
recordstore.deleteRecord(RecordId);
recordstore.closeRecordStore();
txtIdentificador.setString("");
txtNom.setString("");
txtDir.setString("");}

13. Regresemos al formulario *Agenda* y en el mismo ubiquemos la opción *Listar*. Con el botón derecho del mouse seleccionemos *Go To Source* en el menú de contexto que aparece, e inmediatamente escribamos la siguiente instrucción:

```
mostrarListado();
```

14. El código correspondiente se describe a continuación:

PROCEDIMIENTO MOSTRARLISTADO()
private void mostrarListado() {
try {
borrarTabla();
recordstore = RecordStore.openRecordStore("Libreta", false);
byte[] byteInputData = new byte[300];
String Telefono, Nombre, Direccion = null;

PROCEDIMIENTO MOSTRARLISTADO()
ByteArrayInputStream inputStream = new ByteArrayInputStream(byteInputData);
DataInputStream inputDataStream = new DataInputStream(inputStream);
Comparator comparator = new Comparator();
recordEnumeration = recordstore.enumerateRecords(null, comparator, false);
int fila = 0;
while (recordEnumeration.hasNextElement()) {
recordstore.getRecord(recordEnumeration.nextRecordId(),byteInputData, 0);
Telefono = inputDataStream.readUTF();
Nombre = inputDataStream.readUTF();
Direccion = inputDataStream.readUTF();
simpleTableModel.setValue(0, fila, Telefono);
simpleTableModel.setValue(1, fila, Nombre);
simpleTableModel.setValue(2, fila, Direccion); fila=fila+1;
inputStream.reset();}
simpleTableModel.fireTableModelChanged();
inputStream.close();
recordstore.closeRecordStore();
}
catch (Exception error) {
alerta = new Alert("Exception : ",
error.toString(), null, AlertType.WARNING);
alerta.setTimeout(Alert.FOREVER); }
}}

15. Dará un error porque este procedimiento invoca a un método que no existe: *borrarTabla*. Por tanto, procedamos a escribirlo en el MIDlet correspondiente, colocándolo debajo del código que acabamos de escribir. Este código es el siguiente:

PROCEDIMIENTO BORRARTABLA()
void borrarTabla(){
int a = simpleTableModel.getRowCount()-1;
int i;
for(i=1;i<a;i++){

PROCEDIMIENTO BORRARTABLA()
simpleTableModel.setValue(0, i, "");
simpleTableModel.setValue(1, i, "");
simpleTableModel.setValue(2, i, "");
}
simpleTableModel.fireTableModelChanged();
}

16. Ahora procedamos a ejecutar su aplicación. El formulario Agenda es el que se presenta primero, siendo algo similar a la imagen de la figura 13.33. Observe las opciones de menú correspondientes.

Figura 13.33 Formulario Agenda en ejecución.

17. Si pulsamos sobre la opción *Nuevo Contacto* se habilitará el formulario *NuevoContacto,* tal como se muestra en la figura 13.34. Ingrese los siguientes registros indicados en la tabla 13.6.

Tabla 13.6 Registros agregados al RecordStore

IDENTIFICADOR	NOMBRE	DIRECCIÓN
1	Enrique Gómez Jiménez	San José, Costa Rica.
2	Luz Duarte Quintanilla	San José, Costa Rica.
3	Gabriela Gómez Duarte	San José, Costa Rica.

Figura 13.34 Formulario NuevoContacto en ejecución.

18. Luego de ingresados los registros pulse la opción *Back* para retornar al menú principal (*Agenda*). Una vez ahí, ahora seleccione la opción *Ver Contactos* para listar los registros que acaba de ingresar. Se mostrará una pantalla similar a la figura 13.35.

Figura 13.35 Formulario verContactos en ejecución.

19. Ya vimos que los datos se almacenan en el RecordStore. Ahora vamos a proceder a modificar alguno de los datos. Pulsamos la opción *Back* para retornar al menú. Estando ahí seleccione la opción *Editar/Eliminar Contacto*. En esa pantalla va a escribir el número 2, luego seleccione en el menú que posee este formulario y elija *Buscar*. Se mostrarán los datos correspondientes a ese idenficador y procederemos a modificarlo, tal como se indica en la figura 13.36.

Figura 13.36 Formulario ModificarContacto en ejecución.

20. En *Menu* seleccionemos *Editar* para modificar el registro correspondiente. Pulsemos la opción *Back* y observemos en *Ver Contactos* los registros donde se constata el cambio realizado. Puede ver dichos cambios en la figura 13.37.

Figura 13.37 Constatación de los cambios realizados en el RecordStore.

21. Pulsamos *Back* de nuevo y volvamos al punto 19 para proceder a borrar uno de los registros. Ahí digitemos 1 en el campo de identificador y procedamos a buscarlo. Una vez que lo encontremos seleccionamos menú y elegimos la opción *Borrar*. Ahora seleccionemos *Back* y seleccionemos *Ver Contactos* en el menú principal. Ahí podemos ver que el registro ya no existe, tal como se muestra en la figura 13.38.

Figura 13.38 constantando los cambios realizados en el RecordStore.

Como podrá comprobar, según la figura 13.38 el registro borrado ya no existe y no se muestra en la tabla. También podra buscarlo en Editar/Eliminar o en Agregar Contacto. Compruebe que ya no se usa el mismo identificador del registro recién borrado.

Actividades para el lector

Implemente el código para que el RecordStore incluya el número de teléfono del contacto y su dirección de correo electrónico.

Con este ejercicio terminamos el tema de RMS, esperando que haya quedado suficientemente claro. Las referencias que se hacen al final de este capítulo ayudarán a reforzar el tema.

Uso de servicios web en aplicaciones móviles

Quizá una de las ventajas de las aplicaciones web sea la de poderse utilizar ya sea en maquinas estacionarias o móviles. Los dispositivos móviles son un medio eficaz y muy utilizado para el uso de aplicaciones y servicios web. En este apartado vamos a implementar el servicio web que desarrollamos en el capítulo 9 y que básicamente es muy sencillo. Manos a la obra.

1. Ingrese a NetBeans y cree un nuevo proyecto J2ME denominado *mobServicioWeb*.
2. Proceda a configurar un formulario tal y como se observe en la figura 13.39. Observe que sólo contiene un *TextField* de nombre txtFactorial y un *String Item* de nombre *lblFactorial*.

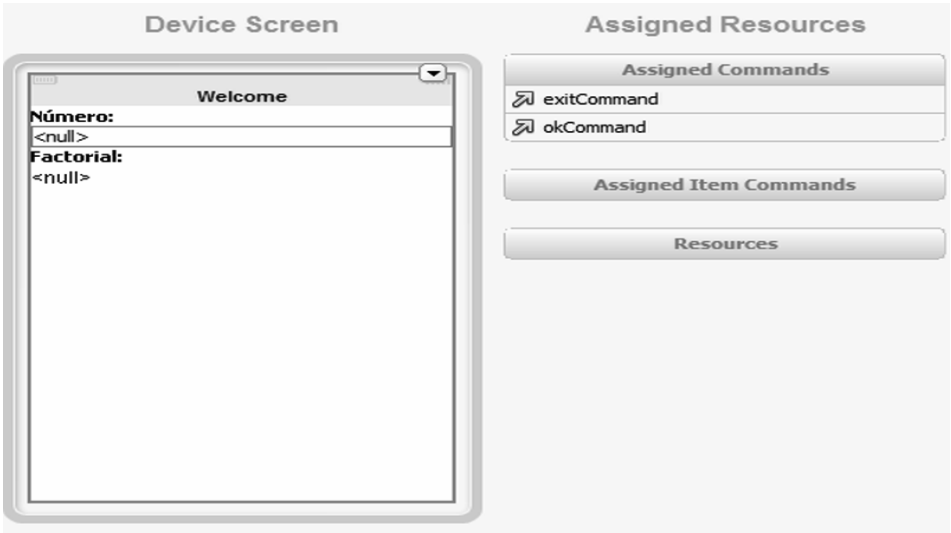


Figura 13.39 Formulario para probar el servicio web.

3. Ahora proceda a crear la referencia a su servicio web. Proceda estableciendo la misma agregando un nuevo archivo, tal y como se muestra en la figura 13.40.

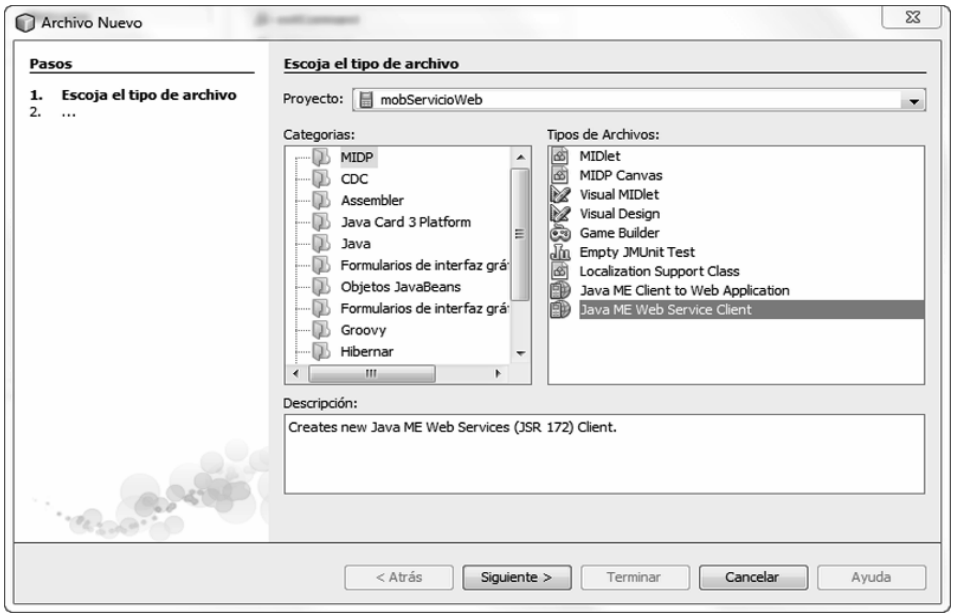


Figura 13.40 Estableciendo la referencia al servicio web.

4. En la pantalla siguiente incluye los datos que se muestran en la figura 13.41.

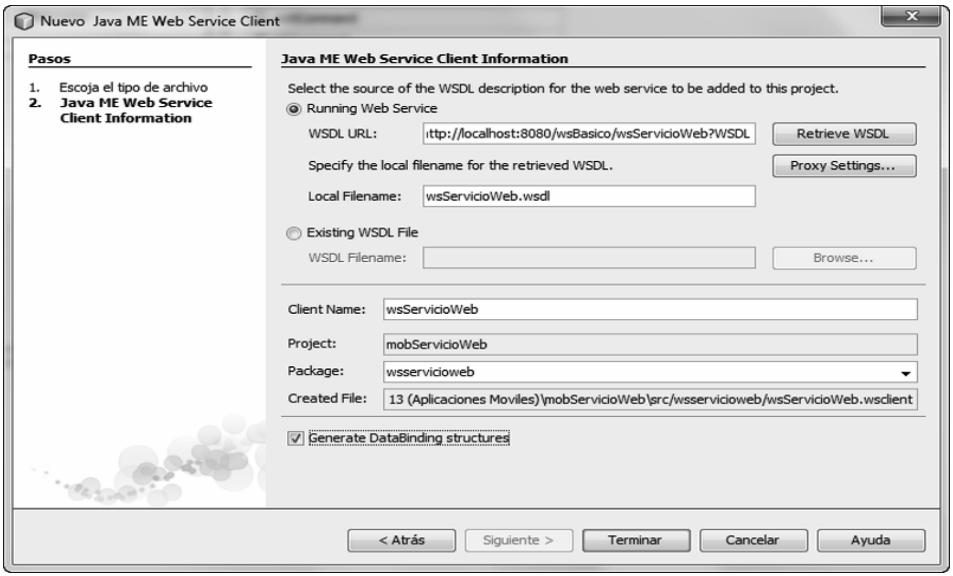


Figura 13.41 Referencia establecida al servicio web.

5. La URL del WSDL es http://localhost:8080/wsBasico/wsServicioWeb?WS en la figura 13.41. Pulse a continuación el botón *Terminar*.
6. Una vez establecida la referencia al servicio web pulse con el botón derecho del mouse sobre la opción okCommand que se muestra en la figura 13.39 y seleccione la opción Go To Source y agregue el siguiente código en la acción de comando (command == okCommand):

PROCEDIMIENTO PARA EJECUTAR EL SERVICIO WEB

//se establece una instancia del servicio.
wsservicioweb.wsServicioWeb servicio = new _
wsservicioweb.wsServicioWeb_Stub();

try {

//se asigna a la etiqueta lblFactorial el cálculo generado en el servicio web.
lblFactorial.setText(String.valueOf(servicio.devFactorial(Integer.parseInt(_
txtFactorial.getString())));

}
catch(Exception ex)
{
lblFactorial.setText("Excepcion "+ex.getMessage());
}

7. La ejecución de este servicio web, invocado desde su dispositivo móvil se vería tal como se muestra en la figura 13.42.

Figura 13.42 Ejecutando su aplicación móvil que utiliza un servicio web.

Si se le presenta algún problema, abra el servicio web que creamos en el capítulo 9 y ejecútelo. Pruebe que funcione bien.

Creación de juegos con J2ME (Java Micro Edition)

Mediante J2ME es posible crear desde aplicaciones de carácter personal o comercial, como también juegos interactivos de gran calidad. Posee librerías especializadas que permiten la creación impresionante de juegos que se adaptan cómodamente a las limitaciones de memoria, almacenamiento y procesamiento que poseen los dispositivos móviles. La interfaz gráfica que proporciona NetBeans para el desarrollo de juegos J2ME, que se muestra a través de *gamedesign*, está dividida en tres partes:

- *Scenes*. Representa el escenario completo del juego. Aquí se colocan los *sprites* y *tiles*. Es decir, cuando se crea un escenario (scene) se colocan los *TiledLayer* y los *sprites* en dicho escenario.
- *TiledLayer*. Se compone de mapas de tiles. Es como una matriz (o mapa) compuesto por una cantidad definida de tiles. En realidad es una clase que hereda de otra clase denominada *Layer*. Es una composición de celdas que se asocian a una imagen que se denomina *tile* (baldosa). Es como tener un rompecabezas con sus respectivas piezas las cuales coloca a su gusto para crear su imagen.
- *Sprites.* Es una subclase de *Layer* que se utiliza para crear una imagen a partir de varias imágenes (frames).

La imagen 13.43 muestra estas tres divisiones:

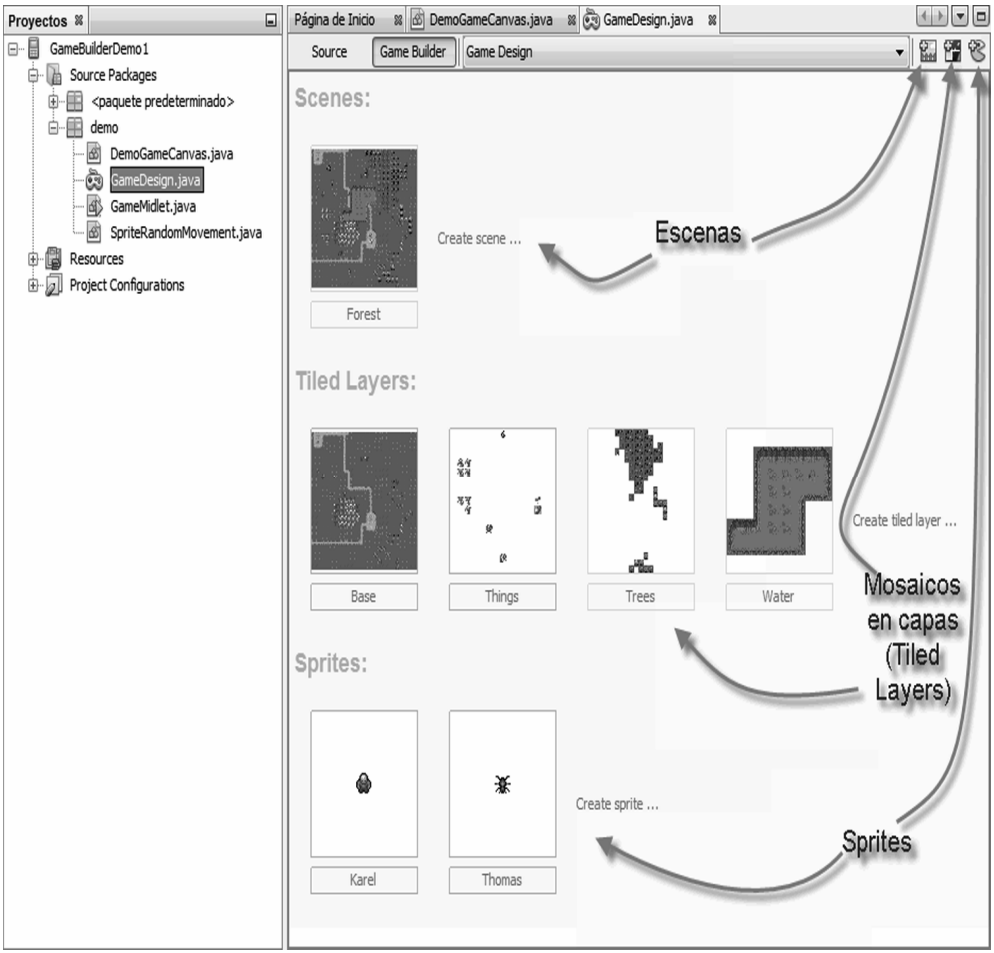


Figura 13.43 Divisiones de diseño de Game Desing en NetBeans 7.1.

Scenes

Las escenas son ubicaciones dentro de un ambiente de juego que poseen características visuales y probablemente audio, que definen la apariencia del mismo. En un juego se pueden crear varias escenas (que pueden ser los niveles del mismo). Pero también un juego sólo puede tener una escena general que se utiliza en todos los niveles. En realidad dependerá del tipo de juego que se requiera desarrollar y de la capacidad del dispositivo donde se ejecutará el juego.

En la figura 13.44 se observa la primera división del Game Design, el nivel scenes.

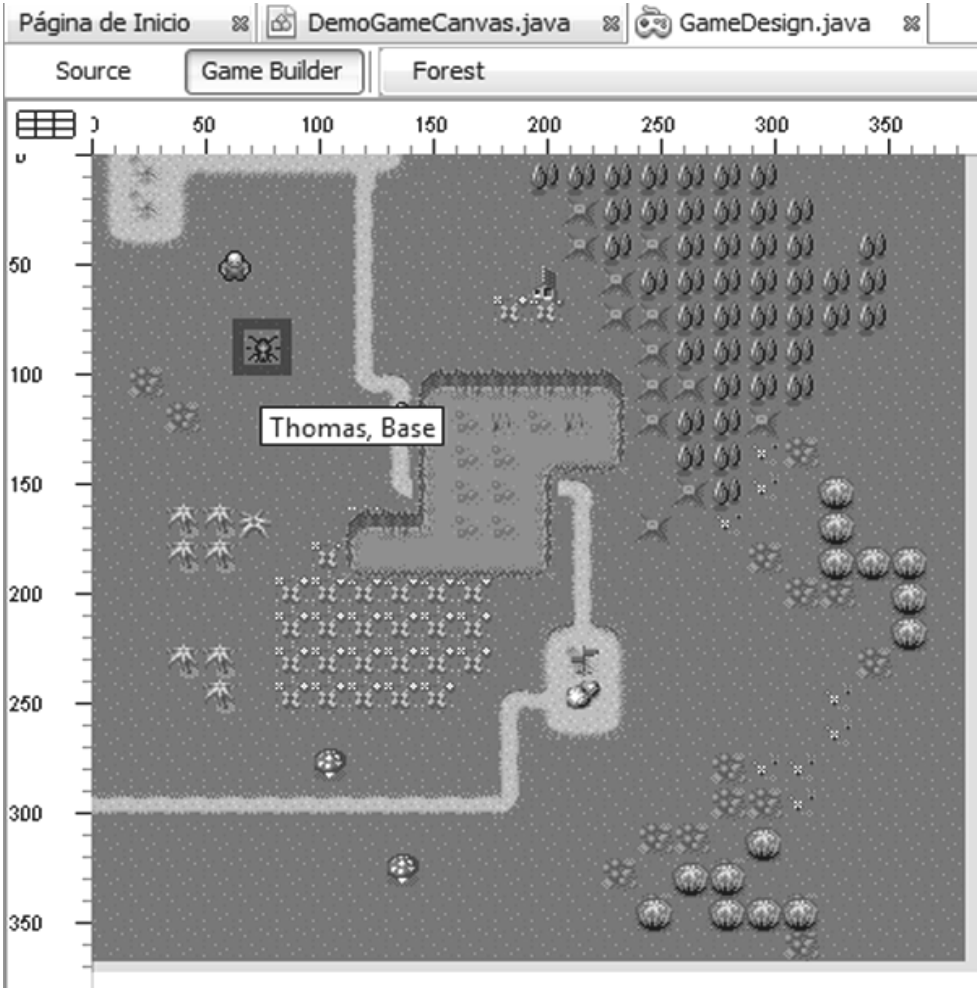


Figura 13.44 Escena del juego GameBuilderDemo1 que acompaña a NetBeans 7.1.

TiledLayer

Los TiledLayer son mosaicos de capas que se crean para hacer escenas en el diseñador de juegos. Si se crea una capa de estos mosaicos debe especificarse el alto y ancho de cada uno y la imagen que debe utilizarse. En la figura 13.45 se observa una capa de tiles denominada *Base* que crea una imagen en mosaicos donde cada tile significa una pequeña imagen dentro de la matríz. Observe que una estrella es el tile índice 8 de la imagen y dónde se coloca dentro del Tiled Layer.

Asimismo, observe que para este Tiled Layer existen 112 tiles de 16x16 pixeles cada uno (es decir, cada celda tiene esa medida, siendo una matriz de 112 celdas).

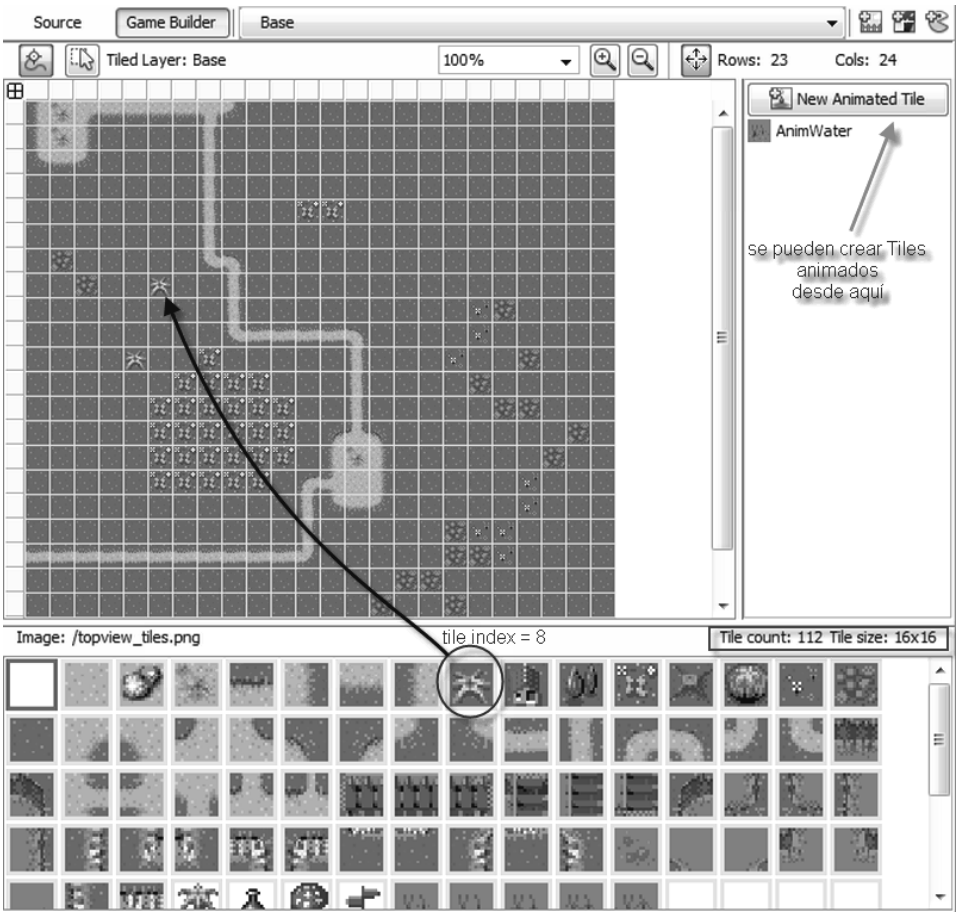


Figura 13.45 Tiled Layer Base del juego GameBuilderDemo1 que acompaña a NetBeans 7.1.

Sprites

Los sprites son capas de personajes u objetos que se utilizan para crear la parte dinámica del juego. Se muestra en la capa superior del juego y se moviliza de acuerdo con las reglas del mismo (la programación). En el caso del juego que venimos estudiando se trata de dos personajes: Karel (que representa un hombrecito) y Thomas (que representa una araña). La imagen 13.46 muestra a Karel y el accionar que representa. Observe que se crea una imagen animada a partir de varias imágenes (al estilo de un archivo gif animado). Se muestra a Karel

yendo hacia arriba, hacia abajo y hacia un lado (a diversas velocidades en este último caso).

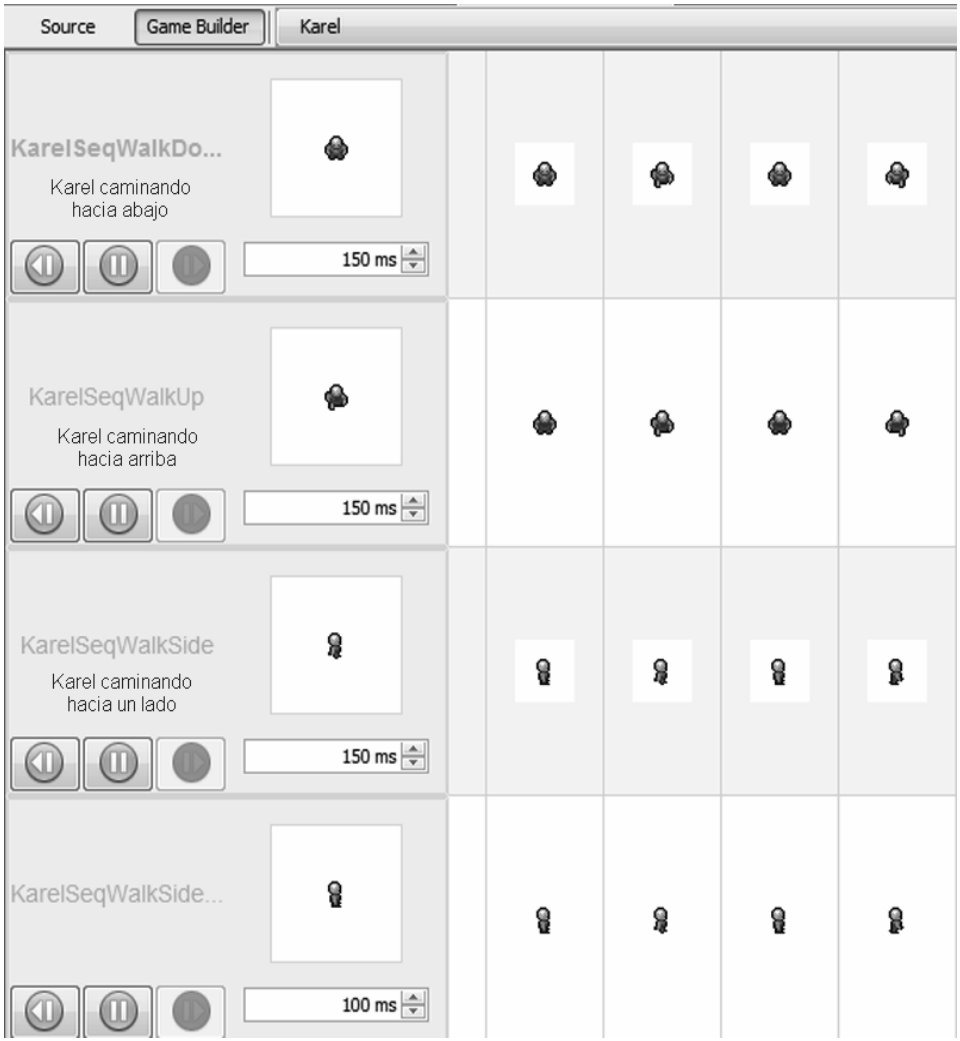


Figura 13.46 Sprite Karel del juego GameBuilderDemo1 que acompaña a NetBeans 7.1.

Para observar este juego debe ingresar a NetBeans y pulsar *Proyecto Nuevo..*, en las categorías de proyectos seleccionar *Ejemplos* y ahí *Java Me* y *Simple Game created with Game Builder (MIDP).* Se visualizará todo el contenido del juego que es 100% funcional.

La lógica del juego se muestra en la figura 13.47. Básicamente se compone de tres programas cuya lanzadera es *GameMidlet.java*.

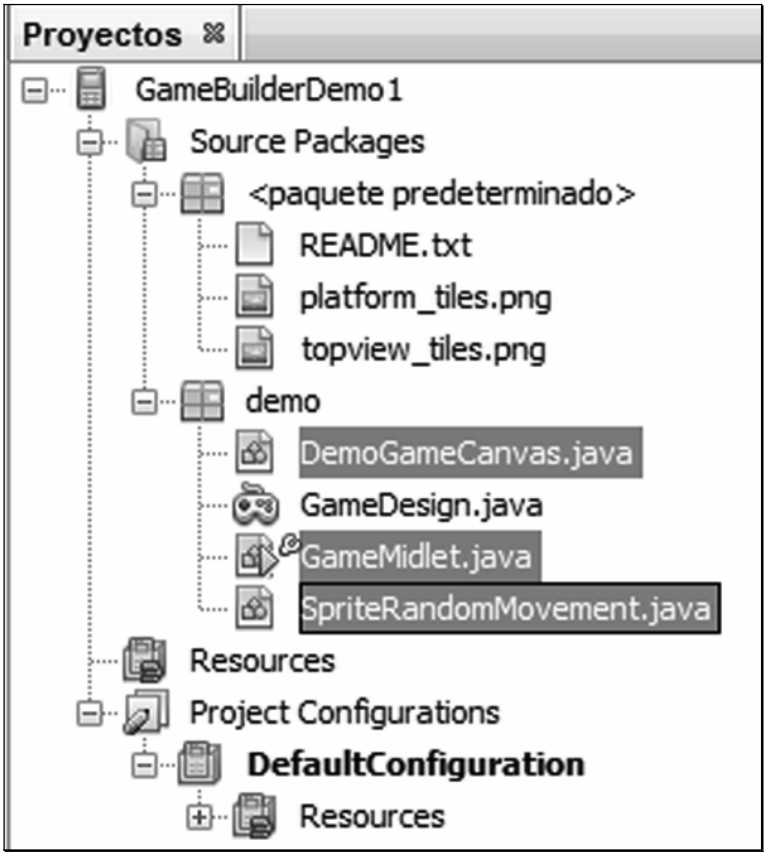


Figura 13.47 Sprite Karel del juego GameBuilderDemo1 que acompaña a NetBeans 7.1.

En la figura 13.47 se muestran los tres programas que acompañan este juego: *DemoGameCanvas.java*, *GameMidlet.java* y *SpriteRandomMovement.java*.

GameMidlet.java es el programa que inicia el juego. Inicia con la creación de un hilo de ejecución (denominado t) y un objeto Display (de nombre d) El hilo t crea una hebra de tipo gameCanvas e inicia su ejecución mediante startApp inicializando el display (variable d).

DemoGameCanvas.java, por su parte, hereda la funcionalidad de la clase gameCanvas que se ofrece con NetBeans. En este programa java se definen

variables de tipo *LayerManager* para manejar las diferentes capas del juego (Base, Things, Trees y Water), gameDesign, sprites, entre otros, que servirán para manejar el accionar y los escenarios del juego. Se invoca desde este programa a *SpriteRamdomMovement* que básicamente administra los movimientos de los diferentes actores del juego.

Existen muchas contribuciones de destacados desarrolladores de juegos a nivel mundial, que han compartido con la comunidad el código fuente de sus creaciones. Uno de estos colaboradores es José Manuel García Maestre, quien se propuso elaborar el remake del juego Manic Miner, originalmente desarrollado por Matthew Smith, un informático londinense. El código fuente del remake de Manic Miner desarrollado por García Maestre se puede descargar desde http://forja.rediris.es/frs/?group_id=823&release_id=1409. Algunas otras referencias a este proyecto son las siguientes:

- http://manicminermovil.forja.rediris.es/index.html
- http://www.linkedin.com/pub/josé-manuel-garcía-maestre/26/172/a69
- https://forja.rediris.es/projects/manicminermovil/

Podemos también conocer algunas historias de Matthew Smith en las siguientes direcciones electrónicas:

- http://blogs.vandal.net/8981/vm/220632982008
- http://en.wikipedia.org/wiki/Matthew_Smith_(games_programmer)
- http://www.pixfans.com/manic-miner-la-obra-maestra-de-matthew-smith/
- http://www.zona48k.com/matthew-smith/

No se darán más detalles de la historia de estos dos programadores de juegos, dado que sólo interesa conocer un poco cómo se fundamenta la creación de éstos para su ejecución en teléfonos móviles. Tampoco se desarrolla uno propio porque no es el objeto de este libro.

Lo que sí queda claro es el interés de realizar remakes de este juego tan popular. Por ejemplo, podemos ver su implementación en C# de Visual Studio 2010, cuyas características fueron remozadas y, siguiendo los patrones y la secuencia de la historia original, han servido de apoyo a otros programadores para imitarlo. Sobre esta implementación y descarga de código fuente e información relacionada puede visitarse los sitios:

- http://code.google.com/p/manic-miner/downloads/detail?name=miner015.zip
- http://www.nachocabanes.com/videojuegos/manicminer/miner01.php

La figura 13.48 muestra la funcionalidad del remake del juego desarrollado con Visual Studio .NET 2010, específicamente en C#.

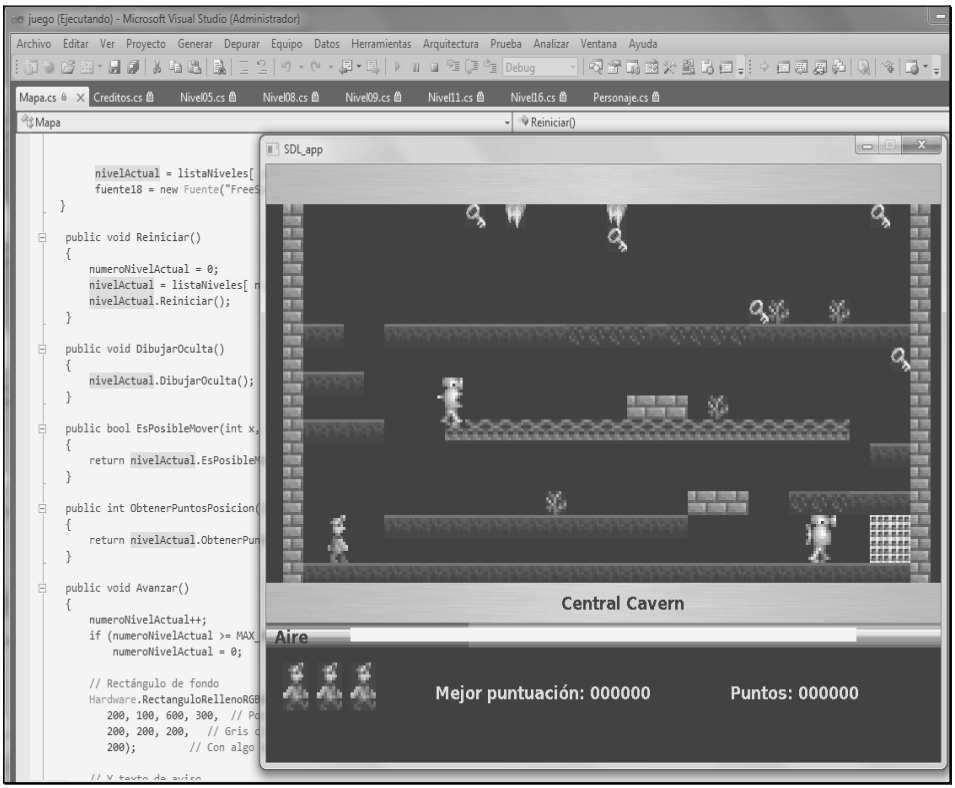


Figura 13.48 Remake del juego Manic Miner, desarrollado en Visual Studio .NET 2010 (C#).

El remake de Manic Miner en NetBeans

Ahora se explicará la funcionalidad y parte de su implementación de la versión descargada desde forja.iris. El procedimiento es el siguiente:

1. Inicie con la descarga del juego desde el sitio http://forja.rediris.es/frs/?group_id=823&release_id=1409 de este juego.
2. Asegúrese de descargar el archivo cuya leyenda es *Proyecto y código netbeans.*

3. Descomprima el proyecto en alguna carpeta identificable (en nuestro caso se respeta el nombre de PFC). El proyecto se verá tal como se muestra en la figura 13.49.

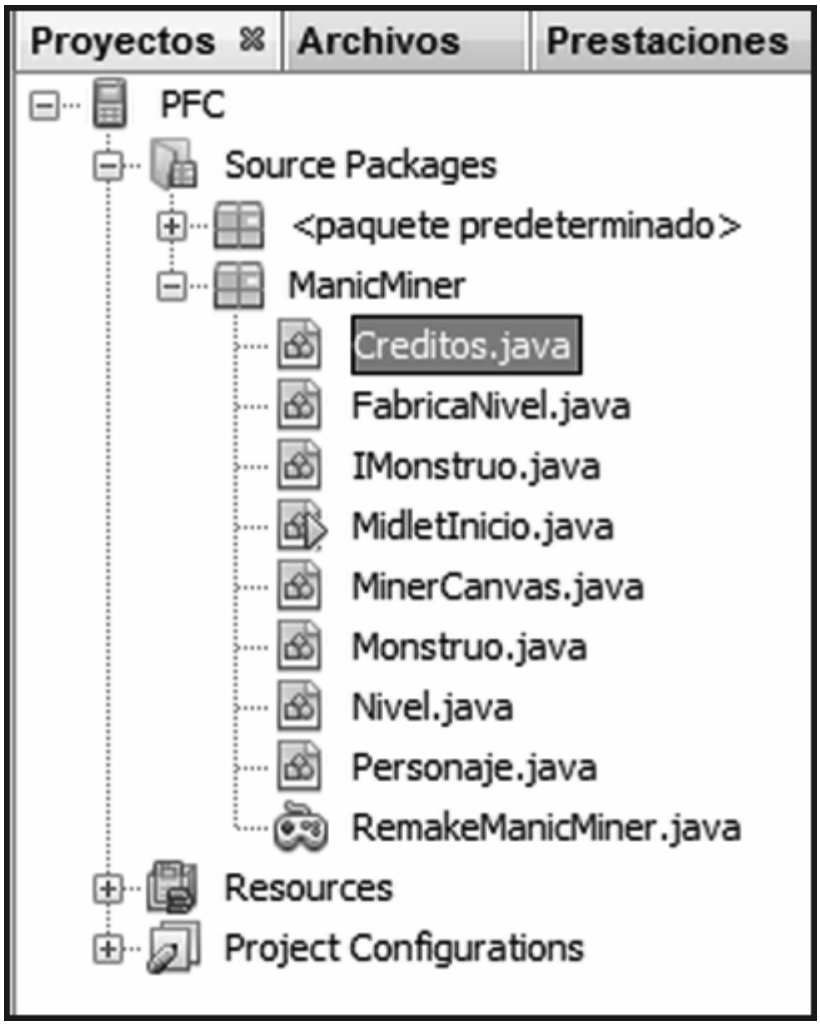


Figura 13.49 El proyecto ReMake de Miner Manic, por José Manuel García.

Observe en el proyecto varios archivos Java (incluida una interfase) que implementan la funcionalidad del juego. También observe el archivo del Game Design denominado *RemakeManicMiner.java*, el cual se muestra en la figura 13.50.

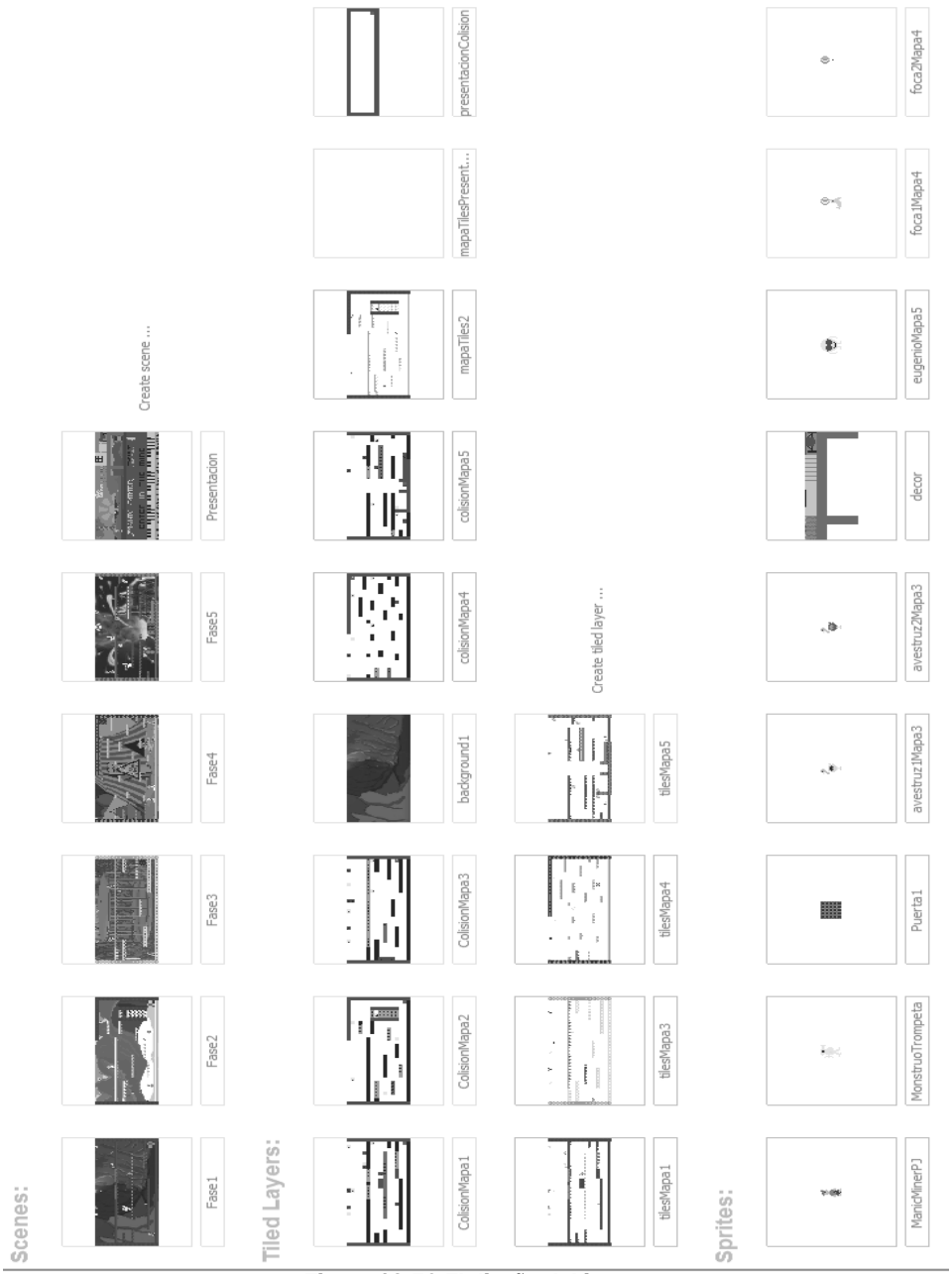
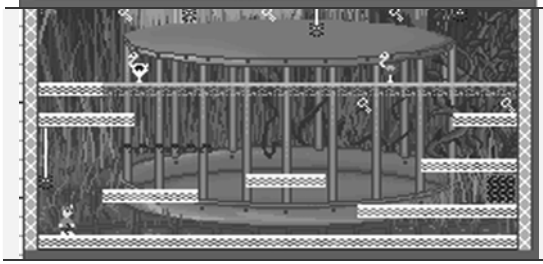


Figura 13.50 El diseño del juego.

4. En la tabla 13.7 se muestran los diseños de pantalla para los cinco niveles (de los 20 que posee el juego original) de este remake.

Tabla 13.7 Niveles del juego del remake de Manic Miner.

DISEÑO	NOMBRE SCENE
	Presentacion
	Fase1
	Fase2
	Fase3

DISEÑO	NOMBRE SCENE
	Fase4
	Fase5

Como podrá observar existen cinco niveles para este remake de los 20 que trae el juego original.

5. En la figura 13.51 podrá observar el diagrama de clases que arma este juego. Del mismo podemos resumir lo siguiente:

    a. MidletInicio es la clase que inicia el juego, creando una instancia de MinerCanvas. Tanto la variable *d* (de tipo Display) como *mc* (derivada de MinerCanvas) son los atributos principales de esta clase. Su funcionalidad básica es iniciar, pausar y terminar el juego.
    b. MinerCanvas es la clase encargada de inicializar las clases Personaje y Nivel. Es la clase principal del juego, dado que con ella se inicia el mismo a través de la creación de los personajes (minero, monstruos y otros).
    c. La clase Nivel administra los personales a través de los sprites, los niveles en que se desenvuelve y sus respectivos estados (con respecto a las colisiones y las vidas que puedan tener).
    d. La clase Personaje administra los actores del juego (movimientos, colisiones, animaciones, estados, entre otros).

Se adjunta documentación de este juego por si le interesa profundizar en el tema del desarrollo de juegos para dispositivos móviles.

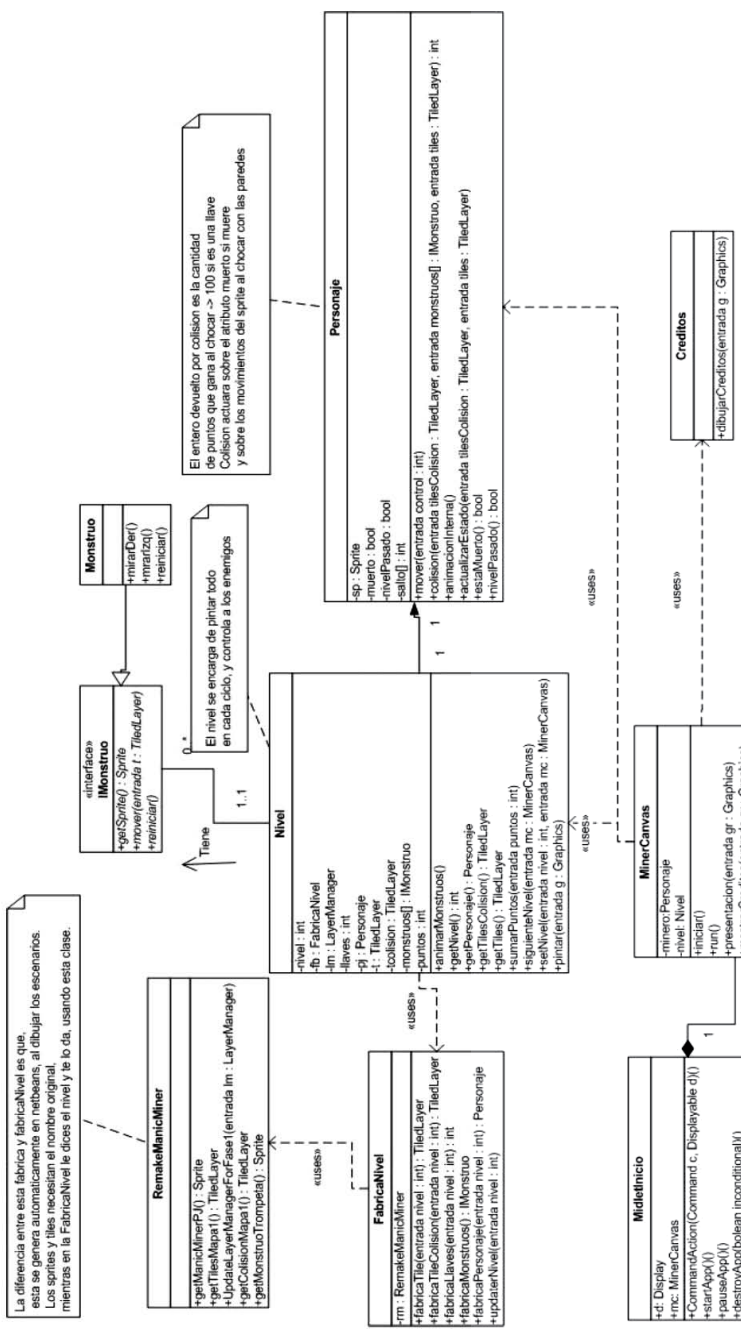


Figura 13.51 Diagrama de clases del juego. Extraído de la documentación del juego.

Sin profundizar en este tema, se ha querido dejar patente que NetBeans facilita enormemente la creación de juegos con J2ME. Falta poner una cuota alta de creatividad en el diseño, mientras que la programación requerirá de mucho planeamiento para controlar toda la fase de animación, incluyendo movimientos, colisiones, creación de escenarios, entre otros.

RESUMEN

En este capítulo se desarrolló el tema del desarrollo de aplicaciones para dispositivos móviles, utilizando NetBeans. Básicamente se trataron los siguientes temas:

1. Introducción a J2ME.
2. Sistemas operativos para dispositivos móviles.
3. Complementos en NetBeas para programar dispositivos móviles.
4. Aplicaciones con persistencia de datos con RMS.

EVIDENCIA

- ☐ Elaboró resúmenes para comprender adecuadamente el funcionamiento de J2ME.
- ☐ Desarrolló ejemplos alternativos de programación para dispositivos móviles, utilizando NetBeans.

Autoevaluación

1. ¿Qué es una máquina virtual de Java (JVM)?
2. ¿Cuáles son las dos configuraciones básicas de J2ME?
3. ¿Qué es un perfil CDLC?
4. Cite al menos dos sistemas operativos para dispositivos móviles.
5. ¿Cuáles son las secciones de una aplicación J2ME con NetBeans?
6. ¿Cuál es el objetivo principal de RMS?
7. ¿Qué es un RecordStore?
8. ¿Para qué se usa setRecord?
9. ¿Para qué se usa RecordEnumeration?

REFERENCIAS

Bibliografía

Arce, Francisco Javier (2012) *Programe Juegos con ActionScript 3.0,* Distrito Federal, México:Alfaomega.

Gálvez Rojas Sergio (2003) *Java a tope: J2ME (Java 2 Micro Edition)* Departamento de Lenguajes y Ciencias de la Computación E.T.S. de Ingenería Informática, Universidad de Málaga, España.

González, Daniel (2011) *Diseño de Videojuegos, da forma a tus Sueños,* Distrito Federal, México:Alfaomega.

Páginas Web recomendadas

1. Uhurulabs.com (2011) *Java en aplicaciones para móviles.* Obtenido el 10 de junio del 2012 desde http://www.uhurulabs.com/category/aplicaciones-moviles/
2. Undernews.com (2011) *Tendencias en plataformas y desarrollo móvil.* Obtenido el 10 de junio del 2012 desde http://www.undernews.com/2011/05/09/tendencias-en-plataformas-y-desarrollo-movil/
3. javamidlet (2008) *How to install the nokia S60 SDK in NetBeans for J2ME.* Obtenido el 10 de junio del 2012 desde http://www.javamidlet.com/2008/07/12/how-to-install-the-nokia-s60-sdk-in-netbeans-for-j2me-development.html
4. scribd.com (2011) *Introducción a J2ME (Java 2 Microedition)* Obtenido el 10 de junio del 2012 desde http://es.scribd.com/doc/7136526/Manual-Programacion-Java-Curso-J2ME
5. mailxmail.com (2011) *Capítulo 25: Almacenamiento. RMS.* Obtenido el 18 de enero del 2012 desde http://www.mailxmail.com/curso-programacion-juegos-moviles-j2me/almacenamiento-rms
6. ibm.com (2011) *J2ME record management store.* Obtenido el 18 de enero del 2012 desde http://www.ibm.com/developerworks/library/wi-rms/
7. web.ing.puc.cl (2010) *Creación de Videojuegos.* Obtenido el 18 de enero del 2012 desde http://web.ing.puc.cl/~iic3686/j2me.html
8. blogs.vandal.net (2008) *Manic Miner: La leyenda de Matthew Smith.* Obtenido el 10 de junio del 2012 desde http://blogs.vandal.net/8981/vm/220632982008
9. it.uc3m.es (2011) *Programación del API de juegos.* Obtenido el 10 de junio del 2012 desde http://www.it.uc3m.es/celeste/docencia/j2me/tutoriales/midp2_0/PracticaGame/
10. programacion-j2me.blogspot (2010) *Programación de Teléfonos Celulares Usando Java Micro Edition.* Obtenido el 10 de junio del 2012 desde http://www.programacion-j2me.blogspot.com/

Respuestas a la autoevaluación

1. Una máquina virtual de Java (JVM) es un programa encargado de interpretar código intermedio (bytecode) de los programas Java precompilados a sistema operativo subyacente y observar las reglas de seguridad y corrección de código definidas para el lenguaje Java.
2. CLDC (connected limited device configuration) orientada al desarrollo de aplicaciones para dispositivos electrónicos con restricciones de procesamiento y memoria. CDC (connected device configuration) orientada a dispositivos con mayores recursos.
3. Es un conjunto de APIS, orientado a un ámbito de aplicación determinado. Los perfiles identifican un grupo de dispositivos por la funcionalidad que proporcionn (electrodomésticos, teléfonos, móviles, etc.) y el tipo de aplicaciones que se ejecutan en ellos.
4. Google Android, IOS, RIM (BlackBerry), Symbian, WebOS, MeeGo.
5. Source, Screen, Flow y Analyzer.
6. Tiene el objetivo de almacenar la información en registros (denominados record) en bases de datos especializadas llamadas RecordStore
7. Es una clase que permite la gestión de registros en un dispositivo celular.
8. Se utiliza para la modificación, siendo RecordId el identificador del registro a modificar y 0 (cero) y outputRecord.length el inicio y final del tamaño del registro.
9. Mediante la enumeración podemos obtener cada registro y mostrarlos a través de un objeto que lo permita, tal como un itemTable o una lista.