Gómez Jiménez Enrique
Moreno Nuñez Jonathan

Fundamentos de programación Java con NetBeans 8.2

Alfaomega

Director Editorial
Marcelo Grillo Giannetto
mgrillo@alfaomega.com.mx

Jefe de Ediciones
Francisco Javier Rodríguez Cruz
jrodriguez@alfaomega.com.mx

Datos catalográficos
Gómez Jiménez, Enrique/Moreno Núñez, Jonathan
Fundamentos de programación Java con NetBeans
Primera Edición
Alfaomega Grupo Editor, S.A. de C.V. México
ISBN: 978-607-538-303-3
Formato: 17 x 23 cm Páginas 524

Fundamentos de programación Java con NetBeans
Gómez Jiménez, Enrique y Moreno Núñez Jonathan
Derechos reservados © Alfaomega Grupo Editor, S.A. de C.V., México

Primera edición: Alfaomega Grupo Editor, México, febrero 2019

© 2019 Alfaomega Grupo Editor, S.A. de C.V. México
Dr. Isidoro Olvera (Eje 2 sur) No. 74, Col. Doctores, C.P. 06720, Cuauhtémoc, Ciudad de México

Miembro de la Cámara Nacional de la Industria Editorial Mexicana
Registro No. 2317

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

ISBN: 978-607-538-303-3

Derechos reservados:
Esta obra es propiedad intelectual de su autor y los derechos de publicación en lengua española han sido legalmente transferidos al editor. Prohibida su reproducción parcial o total por cualquier medio sin permiso por escrito del propietario de los derechos del copyright.

Nota importante:
La información contenida en esta obra tiene un fin exclusivamente didáctico y, por lo tanto, no está previsto su aprovechamiento profesional o industrial. Las indicaciones técnicas y programas incluidos han sido elaborados con gran cuidado por el autor y reproducidos bajo estrictas normas de control. ALFAOMEGA GRUPO EDITOR, S.A. de C.V. no será jurídicamente responsable por: errores u omisiones; daños y perjuicios que se pudieran atribuir al uso de la información comprendida en este libro, ni por la utilización indebida que pudiera dársele. Los nombres comerciales que aparecen en este libro son marcas registradas de sus propietarios y se mencionan únicamente con fines didácticos, por lo que ALFAOMEGA GRUPO EDITOR, S.A. de C.V. no asume ninguna responsabilidad por el uso que se dé a esta información, ya que no infringe ningún derecho de registro de marca. Los datos de los ejemplos y pantallas son ficticios, a no ser que se especifique lo contrario.

Edición autorizada para venta en todo el mundo.
Impreso en México. Printed in Mexico.
Empresas del grupo:

México: Alfaomega Grupo Editor, S.A. de C.V. – Dr. Isidoro Olvera No. 74, Col. Doctores, C.P. 06720, Cuauhtémoc, Cd. de Méx.
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. – Calle 62 No. 20-46, Barrio San Luis, Bogotá, Colombia
Tels.: (57-1) 746 0102 / 210 0415 – E-mail: cliente@alfaomega.com.co

Chile: Alfaomega Grupo Editor, S.A. – Av. Providencia 1443. Oficina 24, Santiago, Chile
Tel.: (56-2) 2235-4248 – Fax: (56-2) 2235-5786 – E-mail: agechile@alfaomega.cl

Argentina: Alfaomega Grupo Editor Argentino S.A. - Av. Córdoba 1215 Piso 10 - C.P. 1055
Ciudad Autónoma de Buenos Aires, Argentina
Tel/Fax: (54-11) 4811-0887 – E-mail: ventas@alfaomegaeditor.com.ar
www.alfaomegaeditor.com.ar

Acerca de los autores

Mag. Enrique Gómez Jiménez

Es ingeniero en sistemas de información y actualmente tiene el grado de maestría en gestión de la innovación tecnológica de 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 C# .NET, Visual Basic .NET, programación en JAVA, entre otros. Se desempeña como encargado de la cátedra de desarrollo de software de la Universidad Estatal a Distancia, UNED (www.uned.ac.cr), y como profesor de Informática de la Universidad Nacional (www.una.ac.cr). Ha trabajado en varias empresas privadas e instituciones públicas en el área de desarrollo de software.

Mag. Jonathan Moreno

Es ingeniero en sistemas de información y tiene un máster en gestión educativa con énfasis en liderazgo, por la Universidad Nacional de Costa Rica. Se desempeña como profesor de tiempo completo en la Universidad Técnica Nacional (www.utn.ac.cr) en la carrera de ingeniería en tecnologías de información y también imparte clases en la Universidad Nacional de Costa Rica (www.una.ac.cr), en la carrera de ingeniería en sistemas de información. Ha dedicado su vida al trabajo en la academia impartiendo cursos de programación, bases de datos, ingeniería de requerimientos, redes Cisco y otros.

Agradecimientos

Deseamos agradecer a todas las personas que nos han apoyado en este proceso editorial. A Rodrigo Muñoz, que ha sido un apoyo incondicional en los procesos académicos, logísticos y administrativos en los cuales lo hemos requerido. A nuestros estudiantes, de los cuales hemos aprendido mucho. 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 y Jonathan Moreno Núñez.

Dedicatoria

Dedico esta obra a Dios por ser el artífice de mi vida. A mi familia, a mis amigos y de mucho corazón a mis estimados estudiantes. A la editorial Alfaomega y a sus distinguidos colaboradores, que nos dan la oportunidad de escribir y llegar a tanta gente que desea salir adelante con esta profesión.

Enrique Gómez Jiménez

Quiero dedicar esta obra primeramente a Dios, quien es mi amigo, mi razón de ser. A mi esposa, mis hijas por apoyarme y creer en mí, por el sacrificio que han hecho a mi lado para alcanzar la meta; a mis padres, que me han ayudado a ser quien soy; a mi amigo Enrique Gómez por brindarme esta oportunidad y enseñarme tanto; a mis estudiantes, que me motivan cada día a ser mejor. A la editorial Alfaomega y a sus excelentes colaboradores, por su excelente trabajo y acompañamiento a lo largo de este camino, por darnos la oportunidad de escribir y aportar un grano de arena a todas las personas que leerán esta obra.

Jonathan Moreno Núñez

Mensaje del editor

Una de las convicciones fundamentales de Alfaomega es que los conocimientos son esenciales en el desempeño profesional, ya que sin ellos es imposible adquirir las habilidades para competir laboralmente. El avance de la ciencia y de la técnica hace necesario actualizar continuamente esos conocimientos, y de acuerdo con esto Alfaomega publica obras actualizadas, con alto rigor científico y técnico, y escritas por los especialistas del área respectiva más destacados.

Consciente del alto nivel competitivo que debe de adquirir el estudiante durante su formación profesional, Alfaomega aporta un fondo editorial que se destaca por sus lineamientos pedagógicos que coadyuvan a desarrollar las competencias requeridas en cada profesión específica.

De acuerdo con esta misión, con el fin de facilitar la comprensión y apropiación del contenido de esta obra, cada capítulo inicia con el planteamiento de los objetivos del mismo y con una introducción en la que se plantean los antecedentes y una descripción de la estructura lógica de los temas expuestos, asimismo, a lo largo de la exposición se presentan ejemplos desarrollados con todo detalle y cada capítulo concluye con un resumen y una serie de ejercicios propuestos.

Los libros de Alfaomega están diseñados para ser utilizados en los procesos de enseñanza aprendizaje, y pueden ser usados como textos en diversos cursos o como apoyo para reforzar el desarrollo profesional, de esta forma Alfaomega espera contribuir a la formación y al desarrollo de profesionales exitosos para beneficio de la sociedad, y espera ser su compañera profesional en este viaje de por vida por el mundo del conocimiento.

Contenido

Introducción IX

Capítulo 1
Introducción a Java y NetBeans 8.2 ..... **3**
1.1 Introducción 5
1.2 Constantes y tipos de datos en NetBeans 16
1.3 Vectores y matrices 20
1.4 Operadores Java 22
1.5 Estructuras de control 24
1.6 Ejercicios 25

Capítulo 2
Fundamentos de estructuras complejas y concurrencia en Java **39**
2.1 Introducción 41
2.2 Arreglos en Java 41
2.3 Estructuras predefinidas en Java 51
2.4 Manejo de concurrencia en Java 64

Capítulo 3
Programación orientada a objetos con Java **85**
3.1 Introducción 87
3.2 Programación orientada a objetos 88
3.3 Interfaces 104
3.4 Polimorfismo 109

Capítulo 4
Aplicaciones de escritorio con Java **115**
4.1 Introducción 117
4.2 Componentes de un aplicativo de escritorio 117
4.3 Paquetes (Packages) en NetBeans 132

Capítulo 5
Manejo de archivos de texto y binarios en Java **149**
5.1 Introducción 151
5.2 Archivos de texto 151
5.3 Aplicación para el manejo de archivos de texto 153
5.4 Archivos binarios en Java 182

Capítulo 6
Gestión de base de datos MySQL con Java mediante NetBeans 201
6.1 Introducción 203
6.2 Instalación de MySQL 203
6.3 Gestión de datos en MySQL con NetBeans 218

Capítulo 7
Introducción al front-end de una aplicación web Java 265
7.1 Introducción 267
7.2 HTML 268
7.3 CSS3 277

Capítulo 8
Servlets Java 293
8.1 Introducción 295
8.2 Conceptos básicos de Servlets 296

Capítulo 9
Java Server Page (JSP) 333
9.1 Introducción 335
9.2 ¿Qué es un JSP? 335
9.3 Java Beans 341
9.4 Java Standard Tag Library (JSTL) 342
9.5 JavaBean 350
9.6 Manejo de EL (Expression Language) con JSP 357

Capítulo 10
Servicios web (web services) 383
10.1 Introducción 385
10.2 ¿Qué es un servicio web (web service 386
10.3 Creación de un servicio web en Java 388
10.4 Protocolo SOAP versus Arquitectura REST 412
10.5 Creación y consumo de un servicio web SOAP 419
10.6 Creación y testeo de un servicio web RESTfull conectado a una base de datos MySql 430

Capítulo 11
Introducción al patrón arquitectónico MVC en Java 445
11.1 Introducción 447
11.2 Patrones de diseño 447

Bibliografía 507
Índice analítico 511

Introducción

Java representa una opción poderosa para el desarrollo de aplicaciones informáticas dirigidas al escritorio, a la web o a dispositivos móviles, tales 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 implementan en estas herramientas de desarrollo, las cuales ofrecen un IDE con muchos controles y características de programación que a su vez facilitan el desarrollo de aplicaciones. Sobre todo se utiliza una filosofía de gran aceptación por parte de instituciones públicas y privadas, grandes corporaciones y hasta esfuerzos individuales: el software libre.

La mayoría de las necesidades de desarrollo de aplicaciones informáticas en la actualidad 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 frameworks que permiten el desarrollo rápido de aplicaciones (RIA) tales como Spring web MVC, ICEFace, Struts, Java ServerFaces, entre otros. Lastimosamente, muchos de los framework gratuitos, para desarrollo web, aún no poseen un editor gráfico que permita el arrastre de componentes y su posterior programación, tal como lo ofrece ASP .NET de Microsoft®.

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, donde además se han ido incluyendo otros por parte de la compañía, así como plugins de terceros. Se permite la inclusión de plugins de empresas constructoras de móviles, y con ello el desarrollo para BlackBerry, Nokia, Sony Ericsson o cualquier otro fabricante.

El lector podrá iniciarse en el fabuloso mundo de la programación en Java, utilizando uno de los IDE más importantes en el mundo: NetBeans. Con ello, 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 al utilizar estas herramientas sin invertir recursos económicos, como lo tendría que hacer cuando se adquieren tecnologías propietarias.

Con este libro podrá desarrollar aplicaciones orientadas al escritorio y para la web. Podrá también practicar la funcionalidad básica de la programación orientada a objetos,

creando clases, interfaces, atributos, métodos, entre otros elementos; además de reforzar esa importante fase del desarrollo de aplicaciones informáticas. En la parte web tendrá la oportunidad de conocer desde la programación de la interfaz de cliente, con HTML5 y CSS3 hasta la de servidor, utilizando Servlets, JSP y clases.

Capítulo 1: Trata las generalidades acerca del lenguaje de programación Java, así como el uso de NetBeans como IDE de desarrollo; se describen componentes esenciales en un lenguaje de programación como lo son: variables, constantes, tipos de datos, operadores y las estructuras de control, los cuales son eminentemente necesarios para iniciar a programar en Java. Incluye cómo descargar e instalar el producto, el entorno del IDE y una descripción de los tipos de proyectos que se pueden crear con NetBeans, versión 8.2.

Capítulo 2: Desarrolla los fundamentos del uso de arreglos en Java, tanto los que son estáticos como los dinámicos. En el caso de los estáticos se estudian los arreglos unidimensionales, bidimensionales (matrices) y tridimensionales; después, se consideran los arreglos dinámicos basándose en dos de las opciones que Java ofrece: ArrayList y HashMap. Por último, se aborda la concurrencia en la programación de aplicaciones, a través del uso de Hilos (Threads).

Capítulo 3: Introduce al lector en el uso del paradigma de la programación orientada a objetos en Java y su aplicación en el desarrollo de aplicaciones. Se inicia considerando los distintos tipos de paradigmas de programación existentes (incluye la programación orientada a objetos). Se abordan conceptos esenciales en el desarrollo de software orientado a objetos tales como: abstraccionismo y programación de clases, creación de objetos, métodos, atributos, encapsulación, ocultamiento de la información, polimorfismo y herencia. Se aporta una estructura genérica de una clase basada en 5 pasos que facilita la comprensión y aplicación.

Capítulo 4: Trata sobre las aplicaciones de escritorio en Java. Se desarrolla una aplicación de escritorio (desktop) haciendo uso de la librería gráfica Swing para demostrar el uso de los componentes más comunes en este tipo de aplicaciones. Asimismo, se desarrolla un paquete (package) para ser utilizado e incluido en dicha aplicación.

Capítulo 5: Aborda la gestión y manejo de archivos planos desde Java, se analizan sus diferencias y similitudes. Se introduce al lector en gestión de archivos de texto y archivos binarios, analizando las clases que provee Java para el manejo de los mismos. Finalmente, se desarrolla un ejemplo que integra la programación orientada a objetos, el uso de Swing y la programación de aplicaciones de escritorio con el fin de demostrar la utilidad y manejo de estos tipos de archivos.

Capítulo 6: Aborda la gestión de bases de datos en MySQL desde Java. Se descarga e instala el gestor de base de datos (MYSQL), el administrador de bases de datos dbforgemysqlfree para la creación de la base de datos, así como el driver de conexión entre ambos. Se explica la teoría que soporta Java con respecto a la gestión de bases de datos (arquitectura JDBC y objetos de conectividad) y las clases provistas para tal fin. Finalmente, se desarrollan ejemplos que demuestran la gestión de datos con Java y MYSQL.

Capítulo 7: Introduce el tema del desarrollo de aplicaciones web del lado del cliente (front-end). Para el diseño de este tipo de aplicaciones, se explica con brevedad la estructura y uso de HTML5, el cual provee una estructura y un código de marcas para la definición de contenido en una página web. También se aborda CSS3, el cual permite dar formato y estilo al contenido que se mostrará al usuario final con el propósito de brindar al usuario una interfaz limpia, ordenada y fácil de usar.

Capítulo 8: Trata el tema Servlet Java y su funcionamiento. En este capítulo se aborda el desarrollo de aplicaciones back-end, las cuales se ejecutan desde el lado del servidor e interactúan con las aplicaciones front-end desde el lado del cliente. Se explica la forma en que un servlet recibe peticiones desde el cliente y debe brindar una respuesta. Se explica el uso de cookies y de sesiones HTTP cuando se desarrollan soluciones web.

Capítulo 9: Trata el tema de JSP (Java Server Page) que constituye una tecnología que permite la creación de código Java del lado del cliente, 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.

Capítulo 10: Introduce al lector en la creación de servicios web (web services) con Java y su creciente utilización. Se consideran tecnologías como el protocolo SOAP y la arquitectura REST a través de Restfull. Se sientan las base teóricas así como las diferencias y bondades de ambas tecnologías, haciendo hincapié en los escenarios recomendados para el uso de uno u otro. Se proporcionan varios ejemplos programados de creación y consumo de servicios web usando ambas tecnologías.

Capítulo 11: Desarrolla la comprensión del patrón de diseño MVC y su uso en el desarrollo de aplicaciones web. Introduce al lector en los métodos utilizados para el desarrollo web usando el modelo MVC, el cual provee una lógica de reutilización de código instaurando una capa controladora que permite determinar el flujo de funcionalidades de una aplicación y estableciendo capas (vista y modelo) que trabajan de maneras interdependientes.

La estructura de cada capítulo incluye:

- **Preguntas de reflexión.** Son preguntas que se plantean al lector antes de iniciar el estudio de cada capítulo con el fin de que él mismo se autoevalúe o informe respecto de la temática que se trata en cada capítulo.

- **Contenido.** Listado de temas generales que se abordan a lo largo del capítulo.

- **Listado de capacidades y competencias.** Se describen brevemente los aprendizajes esperados del lector al término del estudio del capítulo; por otro lado, se describen las habilidades que se pretende desarrollar en el lector.

- **Evidencias de las capacidades desarrolladas.** Describen con brevedad lo que se espera que el lector pueda desarrollar al término de cada capítulo o tema tratado. Esta sección consiste en demostrar que el lector 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.

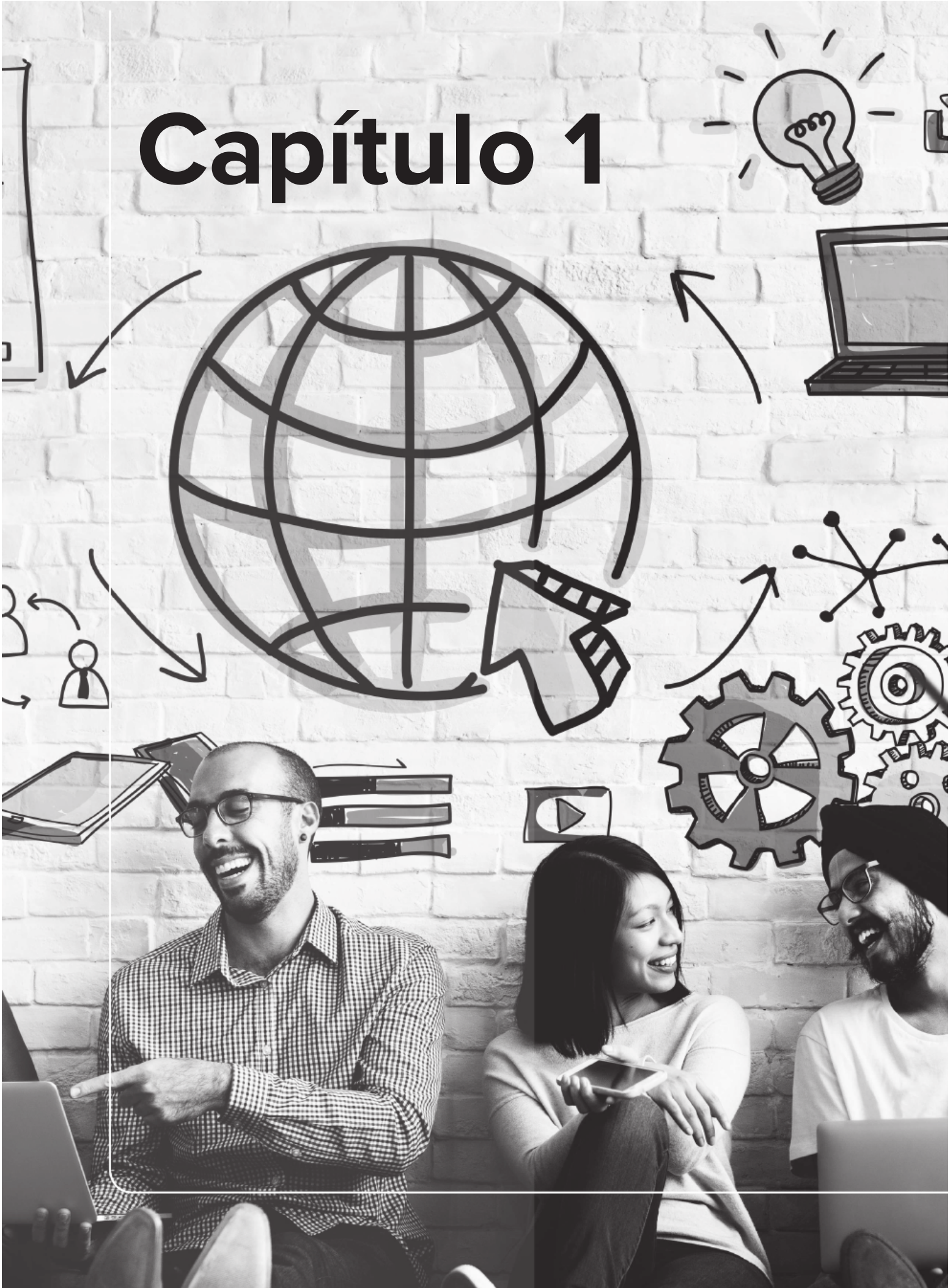
- **Actividades.** Para asegurar un aprendizaje efectivo, resulta imprescindible buscar la interacción del lector con los temas. Es por ello que se privilegian los ejercicios prácticos, las preguntas abiertas (sin respuestas verdaderas o falsas) y las actividades de discusión.

Nota:

En la mayor parte de los capítulos se indican páginas web recomendadas: sitios donde el lector podrá ampliar, profundizar y complementar información, localizar casos de éxito y otros recursos para investigar; también es posible bajar textos y encontrar información actualizada en universidades, institutos de investigación y centros de documentación.

Fundamentos de programación Java con NetBeans 8.2

Capítulo 1



Introducción a Java y NetBeans 8.2

Reflexione y responda
las siguientes preguntas

¿Es Java el lenguaje más utilizado del mundo?

Si Java es un lenguaje altamente utilizado en el mundo de la programación web ¿Por qué no existe un IDE gratuito que permita la creación de interfaces, tal como ASP.NET?

¿Qué tan completo es NetBeans como entorno para el desarrollo de aplicaciones informáticas?

Contenido

1.1 Introducción
1.2 Constantes y tipos de datos en NetBeans
1.3 Vectores y matrices
1.4 Operadores en Java
1.5 Estructuras de control
1.6 Ejercicios

Expectativa

Java constituye un lenguaje de programación extendido por todo el mundo y utilizado por una cantidad importante de desarrolladores de software. Es el lenguaje predilecto en los cursos universitarios y comerciales que se imparten en muchas instituciones públicas y privadas. Por esto, los jóvenes desde temprana edad tienen algún tipo de contacto con este lenguaje para desarrollar lógica computacional. A un nivel mayor, se continúa utilizando en la creación de aplicaciones informáticas de media o alta complejidad.

Después de estudiar este capítulo, el lector será capaz de:

- Describir el lenguaje Java y sus principales componentes.
- Comprender el marco de trabajo de la programación en Java.
- Conocer los tipos de datos que se emplean en Java NetBeans.
- Aplicar adecuadamente los tipos de datos en un programa computacional.
- Utilizar la funcionalidad de los operadores que se usan en Java NetBeans para la solución de un problema computacional.
- Aplicar las estructuras de control en la solución de un problema computacional mediante NetBeans.

1.1 Introducción

Java ha sido uno de los lenguajes de programación preferidos por estudiantes universitarios, no universitarios y profesionales de la informática. Esto quizás se deba al poder de solución que brinda este lenguaje y la accesibilidad que se tiene para obtener marcos de trabajo (framework) gratuitos o de libre distribución. También influye la formación que se brinda desde etapas tempranas y los cursos iniciales de las personas que luego se convierten en desarrolladores privados o funcionarios de alguna institución.

Dada la facilidad de contar con estos marcos de trabajo (framework) gratuitos, son muchos los desarrollos que existen a nivel mundial y que se han convertido en aplicaciones importantes para las empresas. Lo que inició como un compilador para electrodomésticos, poco a poco se convirtió en una herramienta de uso general para el desarrollo de aplicaciones web o para dispositivos móviles.

1.1.1 Introducción a Java

En 1990, Patrick Naughton de Sun Microsystems y sus colegas James Gosling y Mike Sheridan trabajaron en un proyecto conocido como *El proyecto verde*. La idea era desarrollar una tecnología que se utilizara en la programación de dispositivos inteligentes y domésticos. En ese tiempo C++ era el lenguaje más utilizado para dichas labores, aunque ellos no lo consideraban adecuado. Así que Gosling extendió y modificó C++ para sus objetivos, pero no obtuvo los resultados esperados. Luego creó Oak, un lenguaje de programación que contenía funcionalidades de C, C++ y Objective C. Ese fue el nacimiento de Java. Por asuntos de propiedad intelectual, se cambió el nombre de Oak a Java.

Fue en 1995 cuando se fundó la empresa Java Soft, la cual se dedicó al desarrollo de productos basados en Java. Se trabajó con terceros para crear aplicaciones, herramientas, plataformas y servicios para extender las capacidades del lenguaje. Con ello apareció la versión 1.0 del JDK de Java. El desarrollo del lenguaje fue tan rápido y exitoso que empresas como IBM®, Microsoft®, Symantec®, Silicon Valley®, Oracle®, entre otras, contemplaron licencias Java en sus productos. Los navegadores de los años noventa del siglo pasado integraban Java como parte de su software. Por eso era muy común en ese tiempo escuchar sobre la Máquina Virtual de Java (Java Virtual Machine) o JVM. Actualmente, Java representa una herramienta tan importante que está presente en cualquier desarrollo de software.

Las aplicaciones que se pueden desarrollar con Java son portátiles, desde consolas para juegos, centros de datos, teléfonos móviles, aplicaciones para web, hasta software empotrado. El sitio web de Java (www.java.com) indica que son más de nueve millones de desarrolladores en todo el mundo que utilizan este lenguaje. Las estadísticas señalan que 97% de

los escritorios empresariales ejecutan Java (89% se encuentra en Estados Unidos). Tres mil millones de dispositivos móviles utilizan Java en su plataforma.

1.1.2 Introducción a NetBeans

NetBeans es un entorno de desarrollo de uso libre que fue creado para utilizar lenguaje Java. Existe una variedad de módulos diseñados especialmente para hacerlo extensible. Esta cualidad hace que NetBeans sea lo suficientemente poderoso y escriba aplicaciones Java para escritorio, así como para dispositivos móviles. Cuenta con una interfaz gráfica para aplicaciones de escritorio rica en componentes y librerías de reutilización. En aplicaciones web se crean sitios mediante el uso JSP (Java Server Page), aunque soporta frameworks como Swing MVC, PHP, Groovy, C/C++, HTML5, entre otros.

NetBeans permite la creación de distintos proyectos, según la elección del framework de desarrollo. El editor de código, multilenguaje, por cierto, permite el coloreado habitual de los lenguajes de programación modernos, accesibilidad a las clases con sólo un clic del mouse, control de versiones, comprobaciones sintácticas y semánticas, plantillas reutilizables, entre otras características muy útiles para el desarrollo de software. La **figura 1.1** muestra la pantalla principal de descarga del IDE (www.netbeans.org).

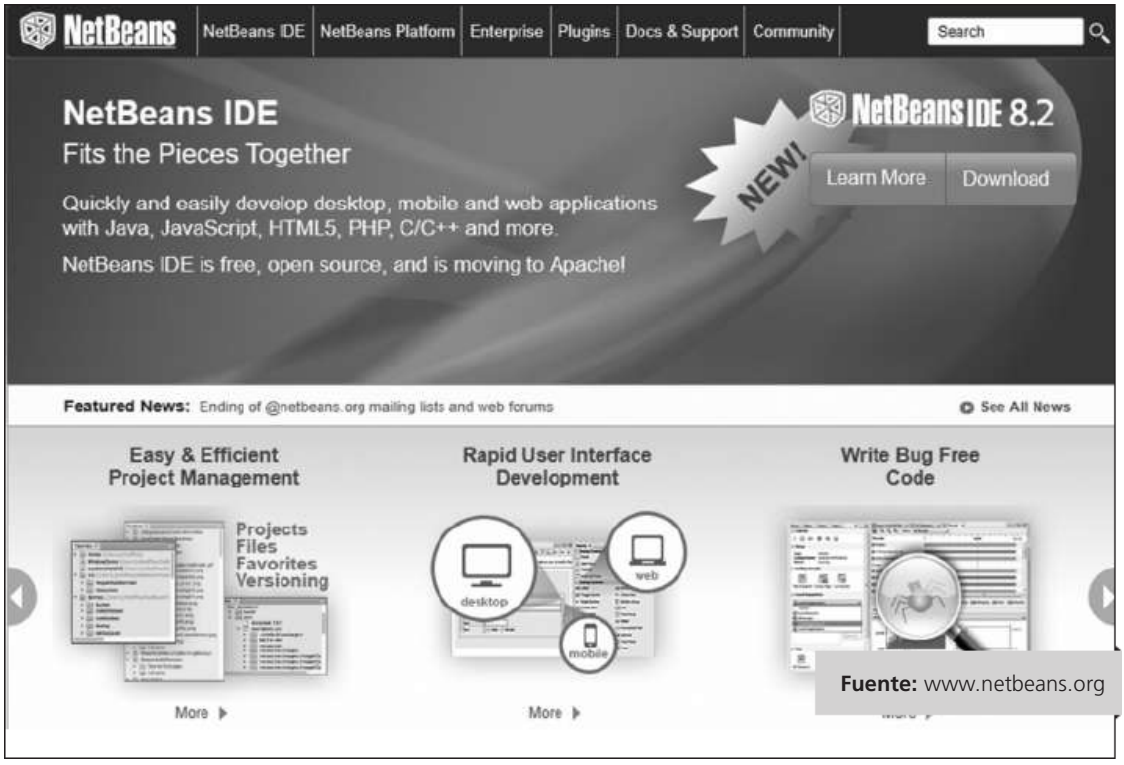


Figura 1.1 Sitio de descarga de NetBeans www.netbeans.org

Una vez instalado el software se tendrá acceso a las funcionalidades que ofrece, de donde destaca su interfaz amigable y la dotación de ayuda en línea. La **figura 1.2** muestra la interfaz que se presenta cuando se ejecuta NetBeans.

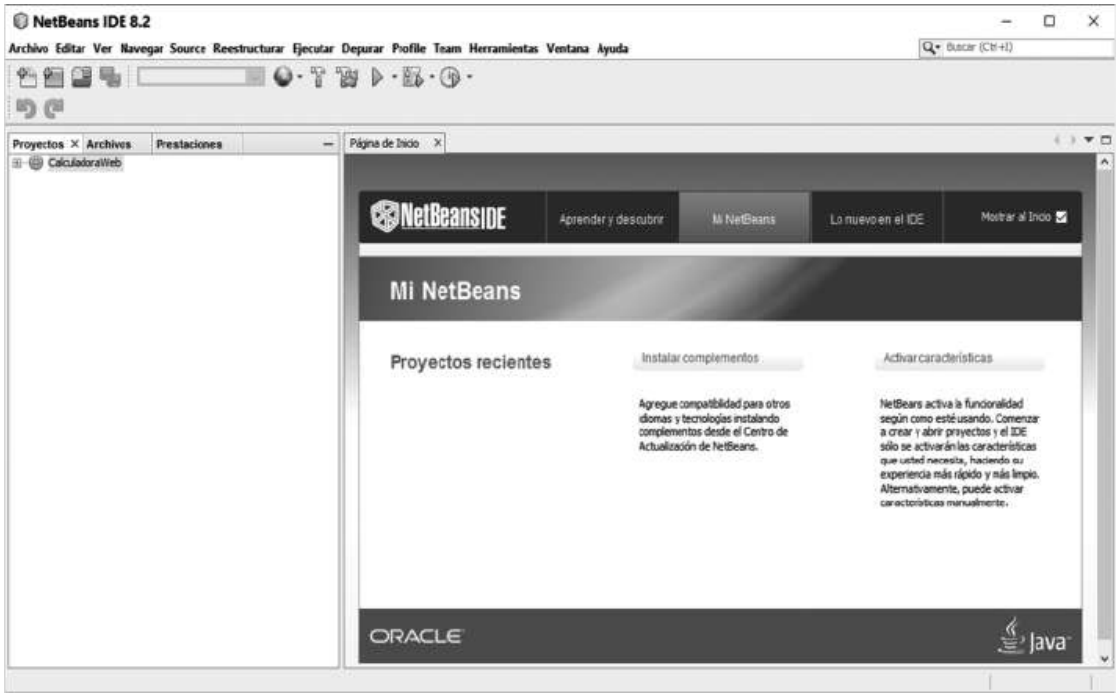


Figura 1.2 Interfaz principal de NetBeans

El código fuente de NetBeans es abierto y gratuito. Puede ser utilizado para escribir aplicaciones comerciales y no comerciales; está disponible para su reutilización de acuerdo con la Common Development and Distribution License (CDDL) v1.0 y por la GNU General Public License (GPL) v2.

Un dato importante es que la organización de NetBeans también ofrece la NetBeans Platform, que constituye una base modular y extensible utilizada como una estructura de integración para el desarrollo de grandes aplicaciones de escritorio. Compañías especializadas en desarrollo de software actúan como proveedoras de extensiones que se integran fácilmente al entorno de desarrollo de NetBeans. Instituciones como la NASA y empresas como Gephi, Dzone, entre otras, han confiado en NetBeans para el desarrollo de soluciones informáticas propias.

Para mayor mayor información puedes visitar los sitios web:

- https://gephi.org/
- https://blogs.oracle.com/geertjan/nasa-mission-operations-on-the-netbeans-platform-part-3-of-4
- https://dzone.com/articles/nb-updated-nato-air-defence-solution

La **tabla 1.1** muestra algunas de las funcionalidades que proporciona NetBeans y que son altamente apreciadas por los desarrolladores de software.

Tabla 1.1 Funcionalidades disponibles en NetBeans

FUNCIONALIDAD	DESCRIPCIÓN
Mayor soporte para las últimas tecnologías Java	Brinda soporte a la tecnología Java 8. Posee editores, analizadores de código y convertidores que facilitan el desarrollo rápido y seguro de aplicaciones Java.
Edición rápida y código inteligente	Un IDE es mucho más que un editor de texto, puesto que ofrece más funcionalidades. NetBeans permite la marca de líneas, destaca el código fuente de forma sintáctica y semántica, refactoriza código, proporciona plantillas de éste, lo genera, entre otras muchas funcionalidades. El editor soporta lenguajes tales como Java, C/C++, XML y HTML, PHP, Groovy, JavaScript y JSP. Pero se pueden agregar otros lenguajes que tienen compatibilidad con NetBeans.
Gestión de proyectos en forma fácil y eficiente	Mantener proyectos grandes con miles de carpetas de archivos, con millones de líneas de código, librerías y otros componentes resulta una tarea difícil, por lo que NetBeans ayuda en esta situación al ofrecer diferentes vistas de los datos desde ventanas de proyectos múltiples y herramientas para configuración y control de versiones.
Desarrollo rápido de la interfaz de usuario	El editor de NetBeans permite el diseño de Gui's para aplicaciones Java Standard Edition (SE), HTML5, Java Enterprise (EE), PHP, C/C++, de una forma rápida y sin problemas mediante el uso de editores y herramientas de arrastrar y soltar.
Escritura segura de código	NetBeans ofrece herramientas de análisis estático de código, lo cual permite la identificación y solución de problemas comunes en código Java. El depurador de NetBeans permite establecer puntos de interrupción en el código fuente, ejecutar métodos, tomar instantáneas y supervisar la ejecución a medida que se produce.

1.1.3 El entorno NetBeans™

Una vez que se ejecuta NetBeans™ se presentará el entorno (IDE) de la herramienta. Básicamente, se podrá notar que mantiene la funcionalidad y los rasgos de las versiones anteriores, con

la ligereza de algunas nuevas prestaciones que lo han hecho evolucionar y constituirse como una herramienta de alta importancia entre los desarrolladores de software.

Si se han utilizado versiones anteriores de este IDE se apreciará que la lógica funcional de la herramienta no ha cambiado: el entorno sigue siendo similar a versiones anteriores, como se muestra en la **figura 1.3**.

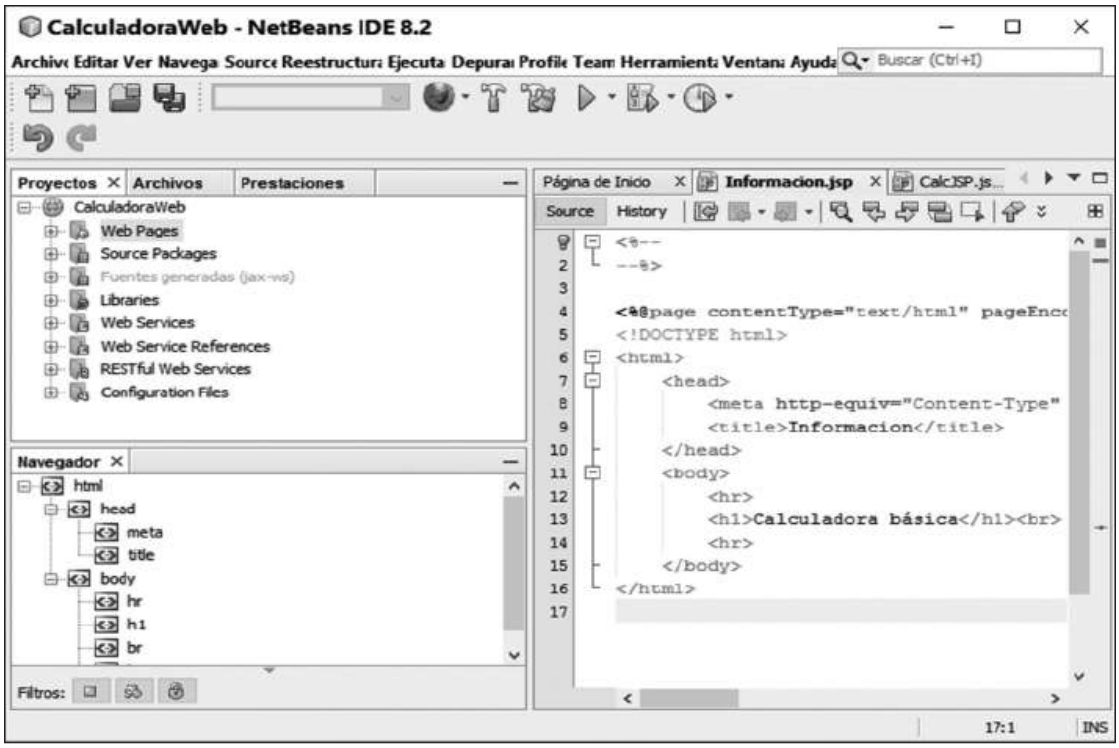


Figura 1.3 Entorno de NetBeans

Como se puede observar, siempre existe a la izquierda el área de proyectos (aquí se visualizan ventanas como las de Archivos, Prestaciones, Paleta, entre otras). Al centro del IDE se muestran los formularios (con su interfaz de diseño y su editor de código fuente), y a la derecha los controles que se pueden utilizar en las aplicaciones.

Arriba de las secciones citadas se encuentran el menú principal del IDE y los iconos de ejecución rápida. Si se está familiarizado con ellos se podrá notar que no hay variación importante en la nueva versión. La **figura 1.4** muestra estas opciones.

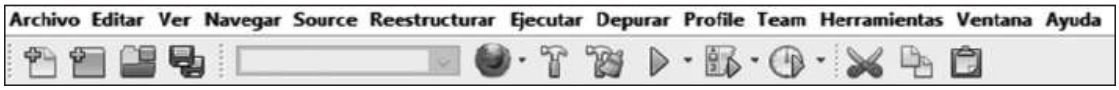


Figura 1.4 Menú principal e iconos rápidos de NetBeans™ 8,2

Del menú principal se pueden rescatar algunas funcionalidades importantes y quizás las más utilizadas a nivel principiante o intermedio. Se requerirá cierto nivel de investigación para el uso de las demás opciones del menú.

La **tabla 1.2** muestra la funcionalidad de algunas opciones de Archivo.

Tabla 1.2 Opciones del menú Archivo

OPCIÓN	FUNCIÓN
Proyecto Nuevo	Permite crear un proyecto nuevo NetBeans™.
Archivo Nuevo	Permite crear un archivo nuevo tipo Java, Java Server Faces (JSP), HTML5, entre otros.
Abrir Proyecto	Abre un proyecto existente.
Abrir Proyecto Reciente	Abre el último proyecto abierto.
Cerrar Proyecto	Cierra el proyecto actual.
Abrir archivo	Abre un archivo existente tipo Java, Java Server Faces (JSP), HTML5, 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 o derivado de un archivo comprimido (Zip).
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 descripción, dado que son las operaciones usuales de cualquier aplicación: copiar, cortar, pegar, entre otros. Quizá haya que destacar la opción Encontrar usos, cuya tarea es permitir la búsqueda de los usos dados al objeto actual entre todos los objetos que componen la aplicación.

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

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

El menú Fuentes permite gestionar la edición de código, brinda opciones como formatear, eliminar, duplicar o insertar código, etc.

Entre otras, el menú Reestructurar brinda funciones como las que se muestran en la **tabla 1.3**.

Tabla 1.3 Opciones del menú Reestructurar

OPCIÓN	FUNCIÓN
Cambiar de nombre	Habilita la posibilidad de cambiar el nombre del objeto seleccionado a través de la modificación de nombres en todos los demás objetos que lo utilizan.
Eliminación segura	Permite eliminar el objeto seleccionado y conserva la seguridad de eliminar las dependencias que posee el objeto.

El menú Ejecutar ofrece, entre otras, opciones como las que se muestran en la **tabla 1.4**.

Tabla 1.4 Opciones del menú Ejecutar

OPCIÓN	FUNCIÓN
Ejecutar Project	Permite ejecutar el código del proyecto.
Probar Project	Habilita la posibilidad de probar la aplicación actual (ejecución).
Generar Project	Crea el archivo JAR de la aplicación.
Limpiar y generar Main Project	Elimina cualquier existencia de variables en memoria producto de una creación anterior del JAR y lo crea nuevamente.
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 aplicación.
Generar Javadoc	Genera el archivo Doc de la aplicación 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; tales como Pausa, Continuar, Paso a Paso, entre otras, que facilitan la ejecución del código de nuestra aplicación y la obser-

vación de su comportamiento en todo momento, lo cual sirve para darnos cuenta si existe un error sintáctico o funcional del mismo.

El menú Profile brinda la posibilidad de revisar el rendimiento de la aplicación. Esto permite analizar el código si no se está satisfecho con el rendimiento.

En el menú Herramientas se encuentran algunas funciones importantes de detallar; la **tabla 1.5** describe algunas de ellas.

Tabla 1.5 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	Permite la administración de bibliotecas (JAR) para aportar mayor funcionalidad al IDE.
Servidor	Permite administrar servidores tales como GlassFish, Tomcat u Oracle WebLogic Server, o servidores de Internet.
Plantillas	Permite administrar plantillas Assembler, C, C++, Fortran, entre otros.
Complementos	Permite la administración de las actualizaciones y los plug-ins 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; la opción Archivos, permite mostrar los archivos del proyecto; Prestaciones, las conexiones y bases de datos establecidas en la aplicación; Paleta, la ventana de controles, entre otras ventanas.

1.1.4 Tipos de proyectos NetBeans™

Los tipos de proyectos existentes en NetBeans™ son diversos y se orientan a funcionalidades distintas. Se pueden citar las tareas básicas a las cuales un lenguaje de programación podría

orientarse: a) aplicaciones de escritorio (desktop), b) aplicaciones web y c) aplicaciones para dispositivos móviles. A todas ellas, NetBeans™ las apoya con las herramientas adecuadas.

A continuación se resumen, mediante tablas, los principales proyectos NetBeans™ que se incluyen en la herramienta, agrupados por tecnologías funcionales. En este caso, los proyectos que se citan se enmarcan dentro de la tecnología de Java SE (Standar Edition) que es una versión liviana de Java para crear aplicaciones no tan complejas.

Java SE (Standar Edition) se implementa en NetBeans™ para el desarrollo de aplicaciones de escritorio, bibliotecas de clases o la reutilización de código de proyectos existentes. La **tabla 1.6** describe los tipos de proyectos que ofrece esta tecnología.

Tabla 1.6 Proyectos de la tecnología Java

TIPO DE PROYECTO	ESPECIFICACIÓN
Aplicación Java	Permite crear un proyecto Java vacío. Se deben generar todas las interfaces y clases desde cero.
Aplicación de escritorio Java	Desarrolla un proyecto Java tipo escritorio con el diseñador de interfaces sin necesidad de programar los controles desde cero.
Biblioteca de clases Java	Genera una biblioteca de clases Java donde se implementa toda una funcionalidad específica que posteriormente se podrá integrar al proyecto simplemente anexándolo.
Proyectos Java con fuentes existentes	Crea un nuevo proyecto basado en archivos de código fuente que ya existen.
Proyectos Java free-form	Importa un proyecto Java o módulo 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 dirige a la creación de soluciones orientadas a Internet. En la **tabla 1.7** se resumen los proyectos de esta tecnología.

Tabla 1.7 Proyectos de la tecnología Java Web

TIPO PROYECTO	ESPECIFICACIÓN
Web application	Permite crear una aplicación web vacía.
Web application with existing sources	Se puede crear una aplicación web utilizando código fuente existente.
Web free-form application	Importa un proyecto Java existente para crear un nuevo proyecto web con la estructura que posee el primero.

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 para poder declarar que los productos que distribuyen corresponden a Java EE. En la **tabla 1.8** se resumen los tipos de proyectos que podemos crear con esta tecnología.

Tabla 1.8 Proyectos de la tecnología Java EE

TIPO PROYECTO	ESPECIFICACIÓN
Enterprise application	Crea una aplicación Java de altas prestaciones y complejidad.
Enterprise application with existing sources	Permite reutilizar código de otro proyecto para crear una nueva aplicación Java EE.
EJB module	Se utiliza para ensamblar "Enterprise Beans" en una sola 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 basada 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	Crea 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 una pequeña aplicación como la implementada en una tarjeta SIM de un teléfono móvil que utiliza la tecnología GSM para comunicarse.

En la **tabla 1.9** se observan algunos proyectos que se pueden crear con esta tecnología.

Tabla 1.9 Proyectos de la tecnología Java Card

TIPO PROYECTO	ESPECIFICACIÓN
Classic applet project	Desarrolla un applet, es decir, una máquina de estados que procesa los comandos recibidos a través del dispositivo, emitiendo respuestas con códigos de estado y datos.

TIPO PROYECTO	ESPECIFICACIÓN
Extended applet project	Permite la instancia de una aplicación applet en el dispositivo o en el navegador para que el cliente tenga acceso.
Classic library project	Crea 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 1.10** se resumen algunos tipos de proyectos que se pueden crear con esta tecnología.

Tabla 1.10 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 una aplicación 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 del mismo.
CDC application	Crea una aplicación para un dispositivo CDC.
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.

1.2 Constantes y tipos de datos en NetBeans

1.2.1 Constantes

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 pretende 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 emplea la palabra clave 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;
```

En el listado No. 1.1 se presenta el ejemplo de un programa en Java que utiliza constantes.

Listado 1.1 Uso de constantes en Java

```
1.  static void pruebaConstantes{
2.      final int diasLaborables = 5;
3.      final int diasNoLaborables = 2;
4.      System.out.println("Número días laborables " + diasLaborables);
5.      System.out.println("Número días No laborables " + diasNoLaborables);
6.  }
```

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

Un ejemplo básico, utilizando una clase, es el que se presenta en el listado de código No. 1.2.

Listado No.1.2 Uso de constantes en Java a través de una clase

```
1.  public class factura {
2.  static final double imp=0.13;
3.      static double impuesto (double monto)
4.      {
5.          return monto * imp;
6.      }
7.  }
```

Se puede observar en el código anterior que imp se declaró como una constante con un valor inicial que no cambia. Se utilizará para realizar un cálculo a través de la función impuesto. Por ahora, sólo hay que observar la instrucción static final imp=0.13 para notar que es muy simple declarar, inicializar y utilizar una constante en Java.

1.2.2 Variables

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 indicarán los tipos de datos que se utilizan en Java y el tamaño que se les asigna a nivel de compilación.

Tipos de datos enteros

Los datos enteros se utilizan como enteros sin signo. La **tabla 1.11** los muestra, así como su tamaño.

Tabla 1.11 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 una 0 inicial, antecediendo a los dígitos.

En el listado No. 1.3 se presenta un programa que demuestra el uso de datos de tipo entero.

Listado No.1.3 Utilización de datos tipo entero en un programa Java

```
1.  public class Enteros {
2.      public static void main (String [ ] args) {
3.      {
4.          byte dato1 = 1;
5.          short dato2 = 100;
6.          int dato3 = 1000;
7.          long dato4 = 10000000;
8.          System.out.println("Dato tipo byte " + String.valueOf(dato1);
9.          System.out.println("Dato tipo entero corto " + String.valueOf(dato2);
10.         System.out.println("Dato tipo entero largo " + String.valueOf(dato3);
11.     }
12. }
```

Tipo de dato coma flotante

Representan números con partes fraccionarias. Se utilizan dos formatos: float (almacena un número 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

Son números con partes fraccionarias que pueden representarse con notación estándar o científica. Por default son de tipo double (8 bytes). También pueden ser de tipo float (4 bytes).

Por ejemplo, es posible 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.

En el listado No. 1.4 se muestra un ejemplo de aplicación de tipo de datos boolean.

Listado No.1.4 Utilización de datos tipo boolean en un programa Java

```
1.  public class pruebaBoolean {
2.  boolean verificar (int numero)
3.  {
4.      boolean aux = false;
5.      if (numero > 0)
6.          aux = true;
7.        return aux;
8.      }
9.  }
```

Como puede observarse, se tiene una clase denominada **pruebaBoolean** que contiene una función llamada verificar la cual recibe un valor entero (número). Su función es validar si ese número es mayor a cero, devolviendo verdadero (true) si lo es y falso (false) si no lo es. En la línea 4 se inicializa la variable booleana **aux** como falsa (false) y la condición, en la línea 5, verifica que si el número recibido como parámetro es mayor a cero entonces la convierte en true, de lo contrario mantiene su valor original falso (false).

Tipo de dato cadena

Para utilizar datos de tipo cadena se debe emplear la clase String, que posee dos constructores: una cadena de 255 caracteres (vacía) o con valores predeterminados.

```
String nombre = new String;

String pais = new String ("Costa Rica");
```

1.3 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 determinadas ocasiones es necesario conocer cuáles son los valores de cierto escenario, por ejemplo, los nombres de cinco estudiantes o los salarios de diez 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 donde toma importancia el uso de vectores y matrices.

1.3.1 Vectores

Un **vector** se define como una estructura con diversas 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.

En el listado No. 1.5 se presenta un programa que muestra la manera de declarar vectores, cómo almacenar valores en ellos y cómo extraerlos para mostrarlos.

Listado No.1.5 Ejemplo de uso de vectores en Java

```
1.  public class pruebaArreglo {
2.    public static void main (String [ ] args)
3.      {
4.         int vector [ ] = new int [5];
5.         String vNombres [ ] = {“Juan”,” Pedro”,” María”}
6.         vector [0] =5; vector [1] =35; vector [2] =16; vector [3] =4; vector [4] =7;
7.         System.out.println(“Dato posición 2 “ + String.valueOf(vector [1]);
8.         System.out. println (“Dato posición 2 “ + vNombres [1]);
9.      }
10. }
```

Como se observa en el código anterior, se declara un vector de enteros denominado **vector** (línea 4) y otro de tipo cadena llamado **vNombres** (línea 5). El primero tiene 5 posiciones y el segundo sólo tres. Posteriormente, al vector de enteros se le asigna valores iniciales (línea 6). 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).

1.3.2 Matrices

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 listado de programa No. 1.6 demuestra el uso de matrices.

Listado No.1.6 Ejemplo de uso de matrices en Java

```
1.  public class pruebaMatriz {
2.      public static void main (String [ ] args)
3.       {
4.       int matriz [ ][ ] = new int [3] [4]; //3 columnas, 4 filas
5.       matriz [0][0] = 15; matriz [1][0] = 24; matriz [2][0] = 38;
6.       matriz [0][1] = 17; matriz [1][1] = 64; matriz [2][1] = 83;
7.       matriz [0][2] = 65; matriz [1][2] = 33; matriz [2][2] = 28;
8.       matriz [0][3] = 28; matriz [1][3] = 17; matriz [2][3] = 99;
9.       system.out.println("Contenido " + matriz [2] [0]);
10.      system.out.println("Contenido " + matriz [0] [3]);
11.  } }
```

En el código anterior se declara una matriz de 3 columnas y 4 filas (línea 4). 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) (líneas 5 a 8). 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 es determinar la longitud o tamaño de cualquiera de ellos mediante **length**.

1.3.3 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 acceder a ellos se requiere hacer casting sobre los mismos. Se puede cambiar el tamaño de la colección en forma dinámica, ordenar, insertar o borrar objetos.

A continuación, en la **tabla 1.12** se presenta un resumen de las principales colecciones utilizadas en Java.

Tabla 1.12 Colecciones de datos en Java

COLECCIÓN	DESCRIPCIÓN
ArrayList	Tipo de arreglo que incrementa o disminuye sus elementos en forma dinámica. Su iteración es rápida y el método de acceso es aleatorio. No es recomendable su uso cuando se produce una gran cantidad de inserciones y eliminaciones de elementos.
Vector	Es una colección que implementa las mismas características de un ArrayList, con la diferencia de que los métodos que aplica se sincronizan mediante el uso de multihilos seguros.
LinkedList	Similar a un ArrayList, con la diferencia de que sus elementos se enlazan en forma doble, por lo que éstos se pueden agregar o eliminar 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 es necesario 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 uno de ellos.

Mayor información sobre colecciones y cómo implementarlas en:

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

1.4 Operadores en Java

Los operadores en Java (y en cualquier otro lenguaje) se dividen en aritméticos, relacionales y lógicos. Los operadores **aritméticos** realizan operaciones matemáticas sobre un conjunto de

valores; los **relacionales** admiten usar comparativas entre elementos o valores; finalmente, los operadores **lógicos** permiten seleccionar acciones de acuerdo con un criterio.

La **tabla 1.13** muestra los operadores aritméticos que tiene Java.

Tabla 1.13 Operadores aritméticos en Java

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 1.14**.

Tabla 1.14 Operadores relacionales en Java

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 a es igual a b
Diferente que	!=	`if(a!=b)` …//si a es diferente de b

Finalmente, en la **tabla 1.15** se muestran los operadores lógicos.

Tabla 1.15 Operadores lógicos en Java

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	ll	`if (a>b) ll (a>c)` … //si a es mayor que b //o a es mayor que b

1.5 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 1.16** se muestran las estructuras de control que se implementan en Java.

Tabla 1.16 Estructuras de control en Java

ESTRUCTURA DE CONTROL	EJEMPLO	BREVE DESCRIPCIÓN

`if(condición-true)`
`{`
`  Sentencias;`
`}`
`else`
`{`
`Sentencias;`
`}`
	`if (a>b)`
	`{`
	`…`
	`}`
	`else`
	`{`
	`…`
	`}`
		Mediante if se puede evaluar la condición inicial y si ésta resulta positiva se ejecutan las instrucciones del primer par de llaves. Si por el contrario, resulta falsa se ejecutarían las que se encuentran entre el par de llaves después de 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 = "miércoles"; 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	BREVE 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 de 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, así como el entorno de NetBeans, por lo tanto, ya se tienen elementos básicos para aventurarse a realizar los primeros programas. En el Capítulo 2 se desarrollarán programas básicos, donde se empleará la teoría de este primer capítulo.

1.6 Ejercicios

Se iniciará esta aventura por el mundo de la programación Java utilizando NetBeans como vehículo personal. Los ejercicios que se abordan serán de una complejidad baja hasta una intermedia, con el fin de familiarizarse con el lenguaje y paulatinamente comprenderlo y dominarlo.

Ejercicio 1.1

1. Se inicia con la creación de nuestro primer proyecto Java; se busca y abre NetBeans desde la computadora. Se selecciona la opción Archivo y luego Proyecto Nuevo, como se muestra en la **figura 1.5**.

2. En la pantalla que se muestra a continuación se selecciona un proyecto de tipo Java, como se observa en la **figura 1.6**. Pulsa el botón Siguiente.

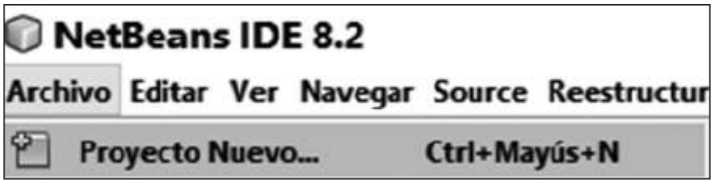


Figura 1.5 Creación de nuestro primer proyecto NetBeans

3. La pantalla que se despliega una vez pulsado el botón Siguiente solicita el nombre del archivo del primer proyecto y la ubicación.

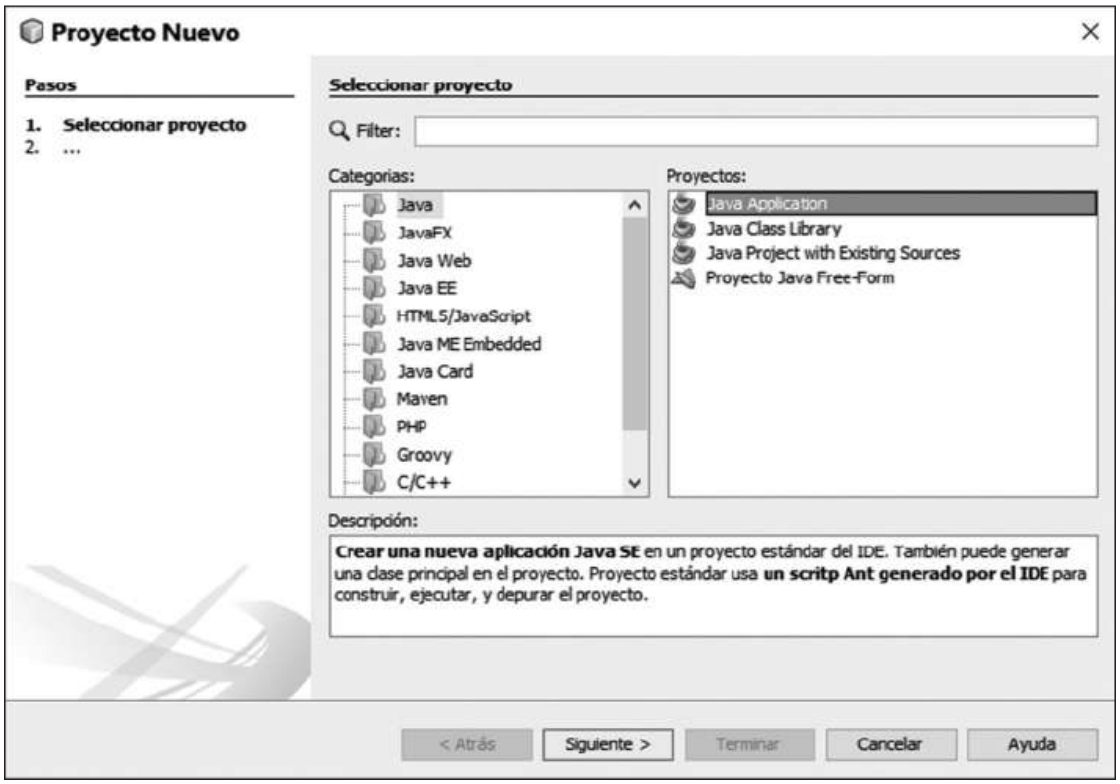


Figura 1.6 Selección del tipo de proyecto NetBeans

Observa en la **figura 1.6** varios proyectos que pueden ser desarrollados con Java utilizando NetBeans. Esta riqueza de posibilidades permite que NetBeans tenga una importante acogida entre los desarrolladores de software.

El ejercicio a desarrollar se nombrará Ejercicio1 y estará ubicado en una carpeta de elección propia. La **figura 1.7** muestra cómo se puede determinar el nombre del archivo y la carpeta de ubicación.

4. Si se pulsa la tecla Terminar se abrirá un entorno que muestra dos ventanas del lado izquierdo. La ventana superior izquierda señala los componentes del proyecto. En este caso, se trata de los paquetes de fuentes (que son los programas, formularios, entre otros que se utilizarán en la solución) y las bibliotecas, que son las librerías requeridas para que la solución funcione.

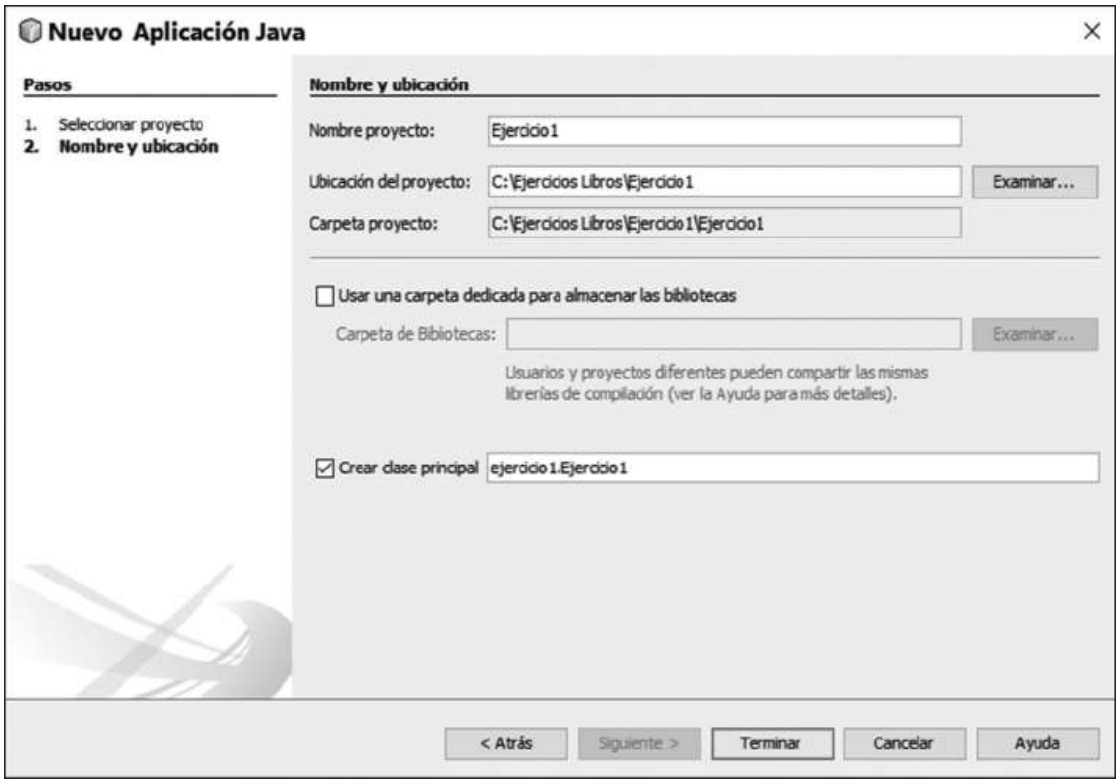


Figura 1.7 Selección del nombre y ubicación del proyecto NetBeans

En la ventana inferior izquierda se observa la funcionalidad de navegación del proyecto para visualizar cada objeto particular. Finalmente, la ventana de la derecha puede ocupar la mitad de la pantalla y es donde se muestra el editor de código o de objetos que forman parte de la solución. En la **figura 1.8** se observan las opciones descritas anteriormente.

A continuación, del editor de código representado en la **figura 1.8**, se describen los elementos en el listado de código 1.7.

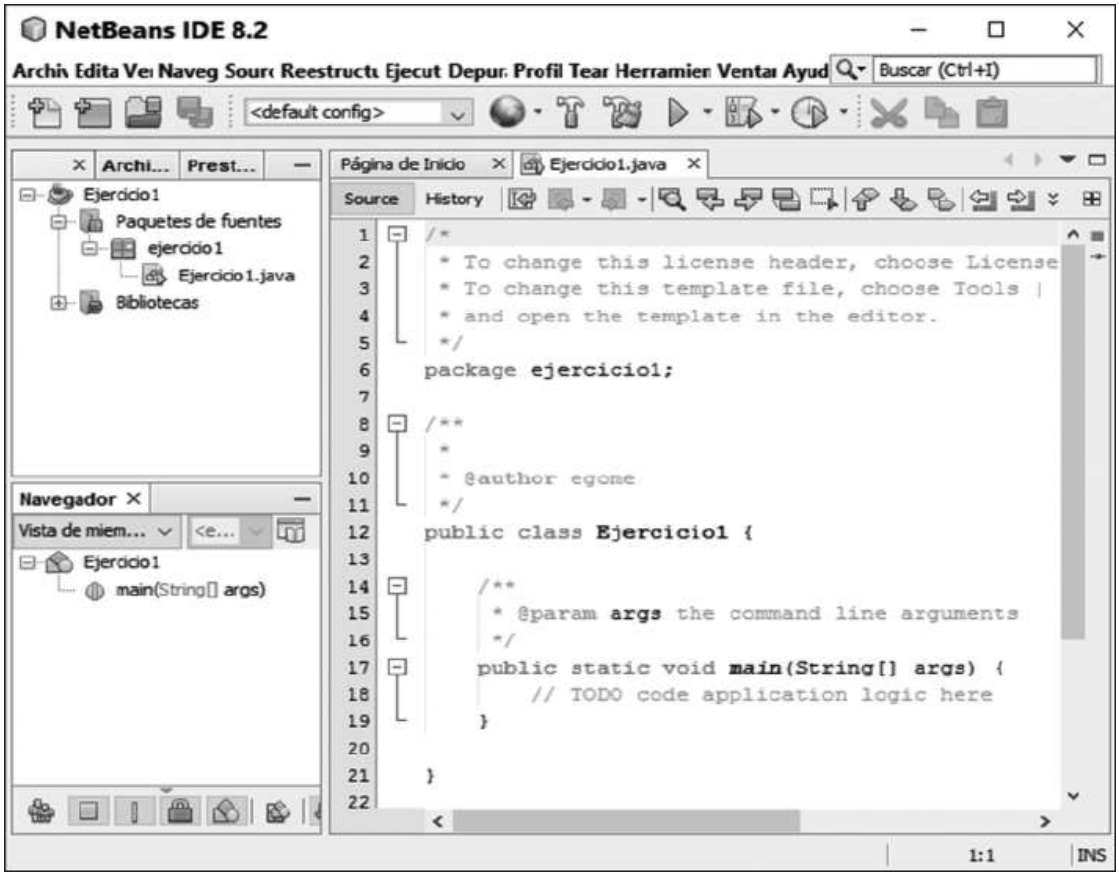


Figura 1.8 Marco de trabajo para nuestro proyecto NetBeans

Listado No.1.7 Código fuente del primer ejercicio en Java-NetBeans

```
1.  public class Ejercicio1 {
2.  static void pruebaConstantes( )
3.  {
4.      final int diasLaborables = 5;
5.      final int diasNoLaborables = 2;
6.      System.out.println("Número días laborables   :" + diasLaborables);
7.      System.out.println("Número días No laborables:" + diasNoLaborables);
8.  }
9.  public static void main(String[ ] args) {
10.     pruebaConstantes( );
11. }
12. }
```

Si se pulsa sobre el botón ejecutar ( ) se puede visualizar el resultado del programa. La **figura 1.9** muestra el resultado de la ejecución.

Se puede observar que el programa principal es **main**. Desde ahí se invoca la función pruebaConstantes(), la cual implementa el código donde se declaran las constantes diasLaborables y diasNoLaborables. Posteriormente, mediante System.out.println se envía a mostrar su contenido en pantalla.

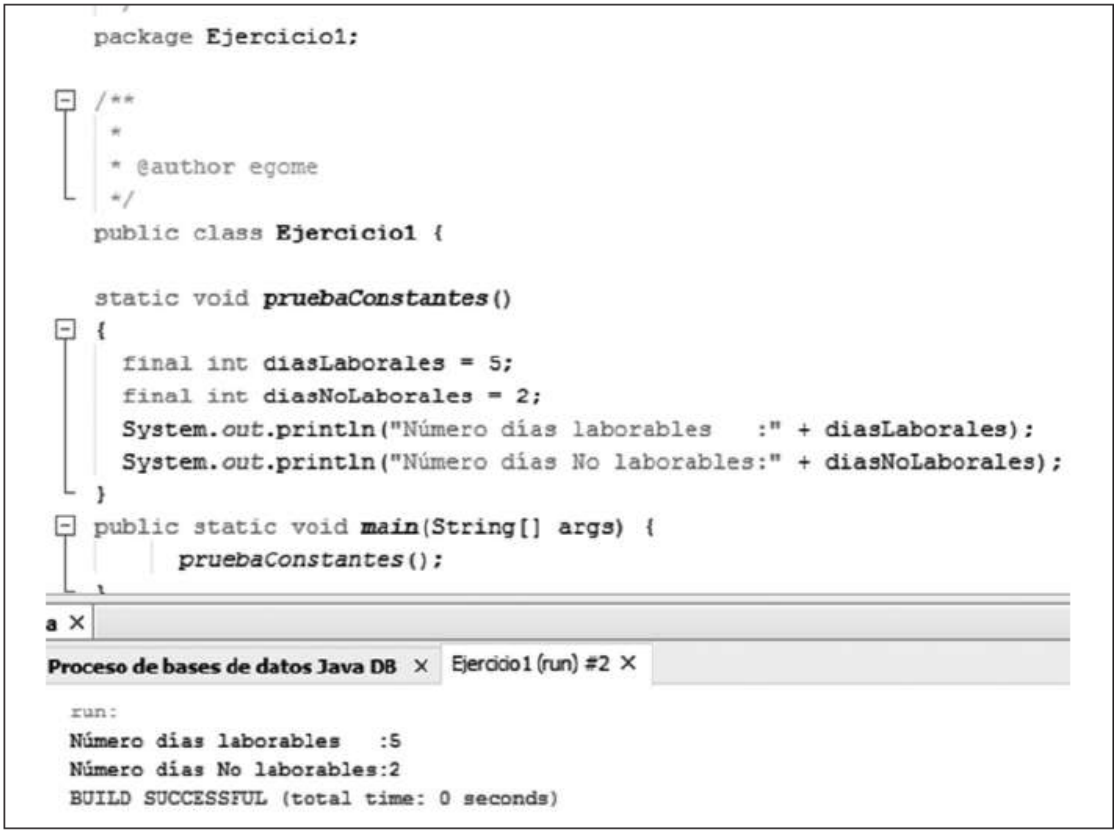


Figura 1.9 Ejecución del proyecto NetBeans

Ejercicio 1.2

Para agregar un ejercicio se creará un nuevo archivo de programa, hay que pulsar con el botón derecho del mouse la opción Paquetes de fuentes del proyecto y seleccionar Nuevo; en el menú que aparece es Java Package, como se observa en la **figura 1.10**. El nombre para este paquete nuevo será **Ejercicio2**.

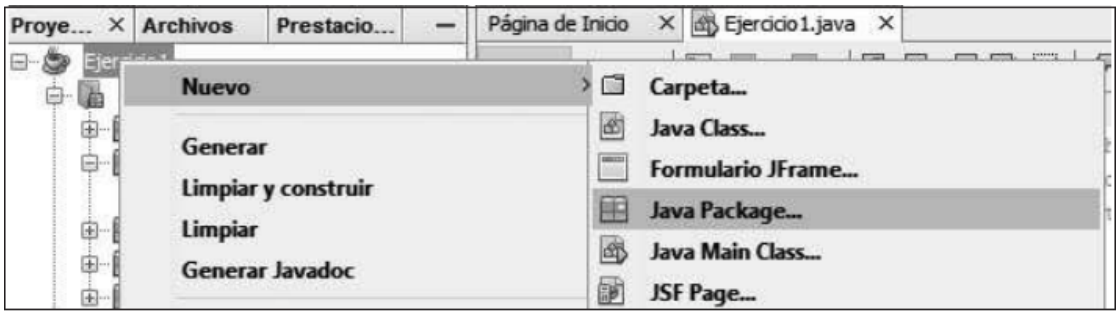


Figura 1.10 Agregar un nuevo paquete al proyecto NetBeans

Una vez creado el paquete, se debe pulsar con el botón derecho del mouse el paquete creado y seleccionar la opción Nuevo, posteriormente Java Main Class, como se muestra en la **figura 1.11**.

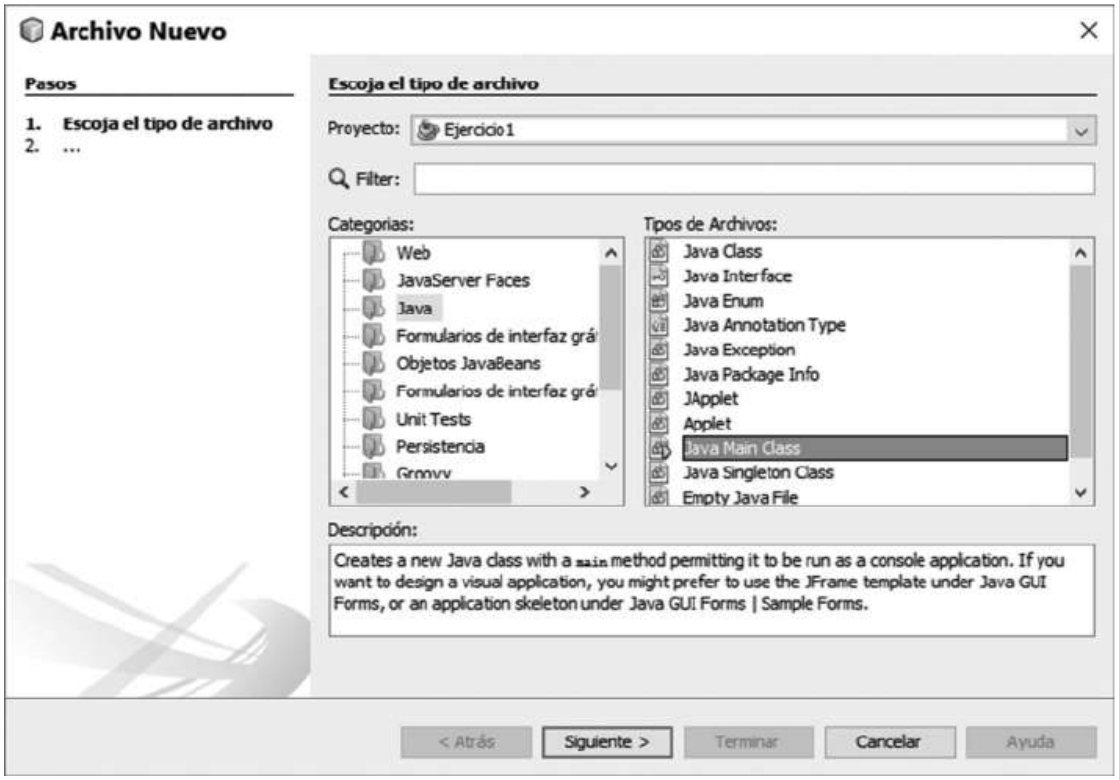


Figura 1.11 Agregar un nuevo archivo a nuestro paquete Ejercicio2

Una vez agregado el archivo hay que proceder a escribir una función que permita calcular el monto de impuesto de una venta. El código respectivo se observa en el listado 1.8.

Listado No. 1.8 Crear la funcionalidad del archivo **Ejercicio2**

```
1.  public class Ejercicio2 {
2.  static final double imp=0.13;
3.  static double impuesto (double monto) {
4.      return monto * imp;
5.  }
6.  public static void main (String [ ] args) {
7.      System.out.println("Impuesto de 1000: " +
8.      String.valueOf(impuesto (1000)));
9.  }}
```

Al observar la línea 7, System.out.println("Impuesto de 1000: " + String.valueOf(impuesto (1000))); básicamente se está invocando al método **impuesto**, enviándole el parámetro 1000 y posteriormente se mostrará en la consola el resultado de la ejecución de la función.

El código y su respectiva salida se muestran en la **figura 1.12**.

Figura 1.12 Ejecución del archivo *Ejemplo2*

Ejercicio 1.3

Para probar otra funcionalidad de Java–NetBeans, se demostrará el uso de la estructura de control switch en Java. Se debe crear el paquete y su respectivo archivo, tal como se hizo en el ejercicio anterior. El código para este programa se muestra en el listado 1.9 mediante la inclusión de una nueva función llamada **dia**.

Listado No. 1.9 Prueba de estructura switch

```
1.  import java.util.Scanner;
2.  public class Ejercicio3 {
3.      static String dia (int ndia) {
4.          string nomdia=“día!”;
5.          switch(ndia) {
6.              case 1: nomdia = “lunes”; break;
7.              case 2: nomdia = “martes”; break;
8.              case 3: nomdia = “miércoles”; break;
9.              case 4: nomdia = “jueves”; break;
10.             case 5: nomdia = “viernes”; break;
11.             case 6: nomdia = “sábado”; break;
12.             case 7: nomdia = “domingo”; break;
13.         }
14.         return (nomdia);
15.     }
16.     public static void main (String [ ] args) {
17.         Scanner scanner = new Scanner(System.in);
18.         System.out.print(“Digite un numero de día: “);
19.         int dia = scanner. nextInt ( );
20.         System.out.println(“El día es “+dia(dia));
21.     }
22. } /*fin public class
```

Se puede observar en el listado 1.9 que en la línea 17 se utilizó una variable de tipo Scanner. La clase Scanner se encuentra en el paquete java.util.Scanner, por lo que se debe incluir como parte de las librerías del programa, pues facilita la lectura de datos obtenidos por consola.

En el listado 1.9 se declara la variable scanner (se observa que está en minúscula, por lo tanto, es diferente a la palabra reservada Scanner). En la fila 18 simplemente se invoca a la visualización de una etiqueta. Hay que observar la fila 19 donde se le solicita al usuario que

digite un número de día. La variable Dia almacena la captura. Finalmente, en la fila 20 se visualiza un mensaje que muestra el nombre del día, previo a que se invoque el llamado a la función dia, a la cual se le envía el parámetro numérico Dia.

Ejercicio 1.4

El siguiente ejercicio consiste en demostrar el uso de la estructura de control for en Java. Se debe crear un nuevo paquete y un nuevo archivo dentro de éste, como se ha hecho anteriormente.

El código para este programa se muestra en el listado 1.10, mediante la inclusión de una nueva función llamada funcionVector.

Listado No. 1.10 Prueba de estructura for

```
1.  import java.util.Scanner;
2.  /*
3.  * @sunombre
4.  */
5.  public class Ejercicio4 {
6.      static void funcionVector( ) {
7.          int vector[ ] = new int[5]; //se declara un vector de 5 elementos
8.          int i;
9.          Scanner scanner = new Scanner(System.in);
10.         for(i=0;i<5;i++) {
11.           System.out.println("Digite el valor para la posición "+
12.           String.valueOf(i)+" : ");
13.           vector[i] = scanner.nextInt( );
14.         }
15.         //desplegando el contenido del vector
16.         for(i=0; i<5;i++)
17.           System.out.println("i : "+String.valueOf(i)+" es "
18.             +String.valueOf(vector[i]));
19.         //obteniendo el mayor de los valores del vector.
20.         int mayor = vector[0];
21.         for(i=1;i<5;i++) {
22.           if(vector[i]>mayor) mayor=vector[i];
23.                System.out.println("El valor mayor vector es : "
24.             +String.valueOf(mayor));  }
```

```
25.  public static void main(String[ ] args) {
26.          funcionVector( );
27.      }
28.  }
```

La salida para este programa se muestra en la **figura 1.13**.

```
run:
Digite el valor para la posición 0 :
3
Digite el valor para la posición 1 :
2
Digite el valor para la posición 2 :
1
Digite el valor para la posición 3 :
5
Digite el valor para la posición 4 :
4
i : 0 es 3
i : 1 es 2
i : 2 es 1
i : 3 es 5
i : 4 es 4
El valor mayor vector es : 5
BUILD SUCCESSFUL (total time: 11 seconds)
```

Figura 1.13 Ejecución del archivo Ejercicio4

Ejercicio 1.5

El siguiente ejercicio consiste en demostrar el uso de la estructura de control for y la estructura while en Java. Se debe crear un nuevo paquete y un nuevo archivo dentro de éste, como ya se ha hecho. El listado 1.11 muestra el código correspondiente.

Listado No. 1.11 Prueba de estructura for y while

```
1.  public class Ejercicio5 {
2.  static int factorial (int num)
3.  {
4.      int fact = 1;
5.      int i;
6.      for (i=2; i<=num;i++)
7.          fact*=i;
8.      return fact;
9.  }
```

```
10. static int factorial2 (int num)
11. {
12.     int fact = 1;
13.     int i=2;
14.     while (i <= num)
15.     {
16.         fact*=i;
17.         i++;
18.     }
19.     return fact;
20. }
21.     public static void main(String[ ] args) {
22.         System.out.println("Factorial de 5 con For " + factorial(5));
23.         System.out.println("Factorial de 5 con While " + factorial2(5));
24.     }
25. }
```

Como se puede observar entre las líneas 10 a 18 se utiliza un ciclo while para implementar la misma lógica que se encuentra en las filas 2 a 9. El resultado de la ejecución de este ejercicio se presenta en la **figura 1.14**.

```
run:
Factorial de 5 con For 120
Factorial de 5 con While 120
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figura 1.14 Ejecución del archivo Ejercicio5

Resumen

En este capítulo se ha desarrollado someramente el tema de Java y uno de los IDE más versátiles y utilizados para la implementación de programas que utilicen este lenguaje de programación: NetBeans. Se trataron los siguientes temas:

- El lenguaje Java y las fortalezas que brinda para el desarrollo de aplicaciones de varios tipos.
- Cómo descargar e instalar el IDE NetBeans.
- El entorno del IDE con la funcionalidad de sus menús y opciones.

- Los tipos de proyectos que se pueden crear con NetBeans.
- Tipos de datos, operadores y estructuras de control utilizados en NetBeans.
- Ejercicios que ayudarán a reforzar los conceptos vistos.

Evidencia

a) Realizar una investigación acerca del software y determinar si existen otras herramientas similares a NetBeans. Enumerarlas e identificar sus principales características.

b) Navegar por el menú de NetBeans y describir algunas de las opciones que se presentan en la barra de éste y que no fueron descritas en el presente capítulo.

c) Citar la funcionalidad de todos y cada uno de los iconos rápidos de NetBeans, que generalmente se ubican debajo de la barra de menús.

d) Inventar escenarios para el uso de iteraciones, ciclos y utilización de sintaxis Java en NetBeans. Utilizar los siguientes vínculos para implementarlos en Java– NetBeans:

http://www.randyvarela.es/ejercicios-java-principiante/
https://github.com/parzibyte/ejercicios-java-escuela
http://puntocomnoesunlenguaje.blogspot.com/2012/06/java-ejercicios-basicos-2.html
http://puntocomnoesunlenguaje.blogspot.com/p/ejercicios.html

Preguntas de autoevaluación

1. ¿Qué es NetBeans?
2. ¿Qué es una constante?
3. ¿Qué es una variable?
4. ¿Qué tipos de datos se utilizan en Java?
5. ¿Qué son estructuras primitivas y complejas?

Respuestas a las preguntas de autoevaluación

1. Es un proyecto de código abierto que cuenta con una gran cantidad de usuarios alrededor del mundo que colaboran con el desarrollo de este IDE.

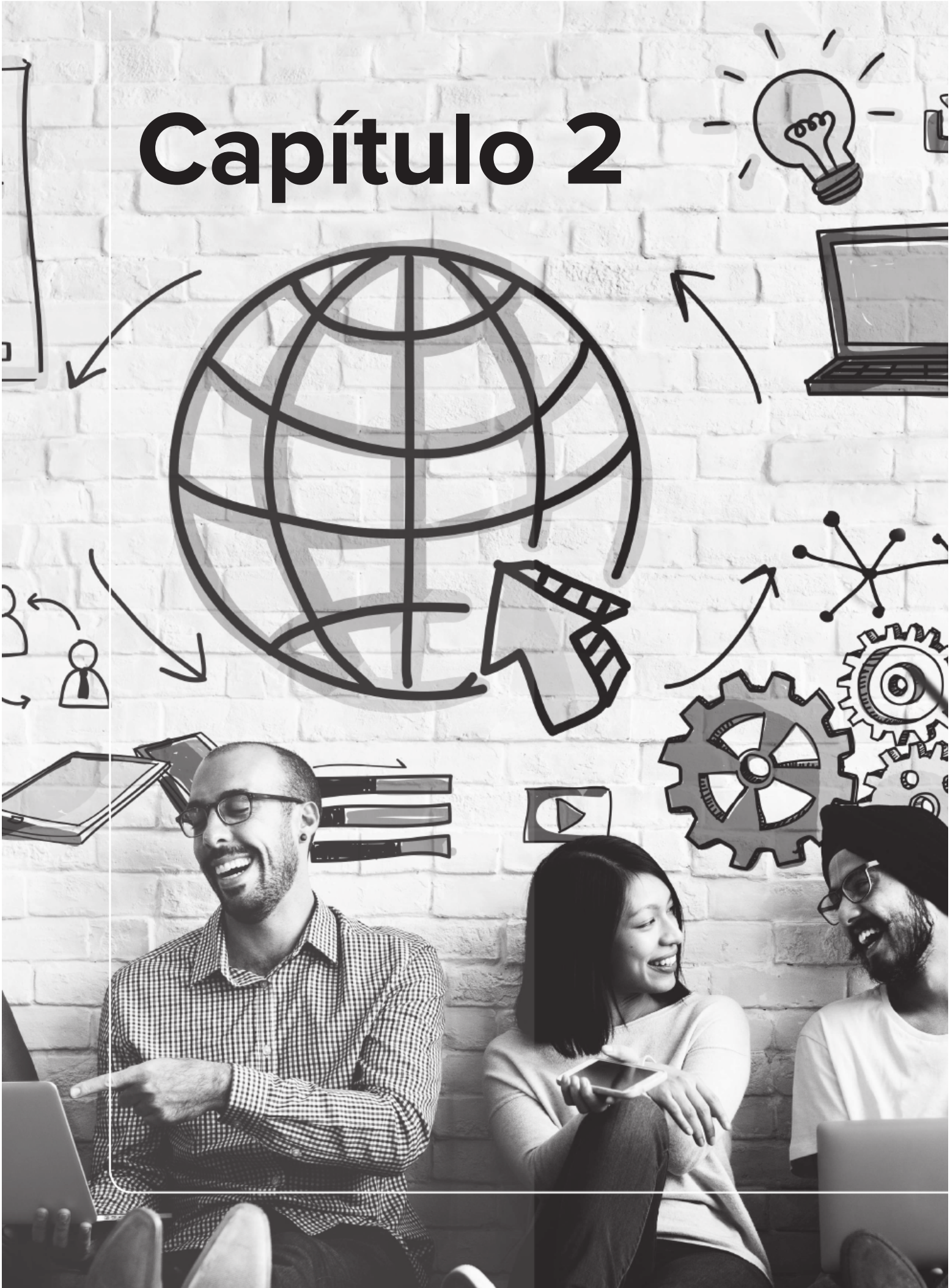
2. Una constante es un objeto cuyo valor no puede ser modificado durante la ejecución de un programa computacional.

3. Una variable es un objeto cuyo valor puede ser modificado durante la ejecución de un programa computacional.

4. Los tipos de datos que se utilizan en Java son enteros, cadenas, lógicos, de fecha, entre otros.

5. Una estructura primitiva es aquella que está conformada por tipos de datos de un solo tipo (int, short, long, double, date, entre otros), mientras que una estructura compleja está formada por datos de diferente tipo (Vector, ArrayList, TreeMap, entre otros).

Capítulo 2



Fundamentos de estructuras complejas y concurrencia en Java

Reflexione y responda
las siguientes preguntas

¿Cuántas veces se utilizan estructuras complejas de datos en la solución de nuestros programas computacionales?

¿Por qué es importante conocer los conceptos de concurrencia en Java?

¿En qué tipo de aplicaciones se utilizan arreglos y concurrencia Java?

Contenido

2.1 Introducción
2.2 Arreglos en Java
2.3 Estructuras predefinidas en Java
- **2.3.1** Clase ArrayList
- **2.3.2** HashMap

2.4 Manejo de concurrencia en Java
- **2.4.1** La programación concurrente en Java

Expectativa

Cuando se inicia una carrera informática, los cursos de programación aumentan el nivel a medida que se avanza. Llegará el momento en que se dejen las estructuras simples como cadenas, enteros, tipos lógicos, entre otros, para adentrarse en estructuras complejas: arreglos, pilas, colas, etc. En esta etapa, es posible gestionar varios espacios de memoria en nuestra computadora, donde se pueden colocar variables posibles de cambiar en el tiempo. Por ello, en el presente capítulo se hará un corto recorrido por la teoría y la implementación de este tipo de estructuras. No pretende ser un riguroso análisis de las mismas, pero sí un introductorio al fascinante mundo de las estructuras de datos complejas. También se expondrá el conocimiento básico sobre el uso de la concurrencia en Java.

Después de estudiar este capítulo, el lector será capaz de:

- Enumerar algunas de las estructuras complejas existentes en Java.
- Comprender la sintaxis de uso de las estructuras complejas existentes en Java.
- Solucionar problemas computacionales básicos mediante el uso de estructuras complejas con el uso de NetBeans–Java.
- Solucionar problemas computacionales básicos mediante el uso de concurrencias en NetBeans–Java.

2.1 Introducción

En este capítulo se pretende que el lector aprenda a desarrollar programas básicos en Java mediante el uso de NetBeans. La idea principal es construir pequeños programas que permitan acumular conocimientos para luego crear soluciones con un poco más de complejidad, la cual ahora se encuentra en nivel medio. El presente capítulo es una introducción en el mundo de las estructuras de datos complejas y la comprensión de su uso. También se fundamentará la concurrencia en Java a través de la utilización de hilos.

Existen varios tipos de estructuras de datos, mismas que se construyen, generalmente, sobre otras más simples. Por ejemplo, un vector es una colección de elementos, casi siempre de un mismo tipo, en donde se puede acceder mediante su posición en la estructura. Estos arreglos pueden ser unidimensionales (representados por una fila de tamaño n), bidimensionales (representados por filas y columnas) o multidimensionales (formados por planos, filas y columnas).

Los lenguajes modernos de programación, como Java, han implementado estructuras propias; por ejemplo, ArrayList, HashMap, entre otros, que pueden contener objetos en cada uno de sus elementos.

Finalmente, la concurrencia es tratada en este apartado para conocer cuál es su filosofía de uso e implementación en la solución de problemas computacionales básicos. Los **arreglos** (unidimensionales y multidimensionales) y las **matrices** (bidimensionales), se denominan estructuras estáticas, mientras que las estructuras implementadas en un lenguaje de programación (ArrayList, HashMap y otros) se llaman estructuras de datos dinámicas.

2.2 Arreglos en Java

Los **arreglos** se definen como estructuras compuestas por un conjunto de celdas donde se pueden almacenar datos de un mismo tipo. Por ejemplo, hay arreglos de datos de **tipo entero** o arreglos de datos de **tipo carácter**, entre otros. En cada celda se puede acceder por su posición, generalmente referenciada mediante un índice el cual es un valor numérico que señala dicha posición dentro de la colección de elementos.

Al existir arreglos unidimensionales (solo una fila), bidimensionales (filas y columnas) y multidimensionales (matriz, filas y columnas), un índice representará la posición de la celda en la fila para el primero, la fila-columna para el segundo y matriz-fila-columna en el último. La figura 2.1 muestra un vector unidimensional que contiene un valor de tipo cadena (nombres). Debajo de la estructura se muestra su respectivo índice (posición del elemento respecto del resto de elementos del vector).

Alfonso	María	Juana	Pedro	Sebastián	Mariana
[0]	[1]	[2]	[3]	[4]	[5]

Figura 2.1 Estructura de un vector unidimensional

Como se observa, cada valor tiene una posición (de 0 al tamaño -1) dentro del conjunto de elementos del vector. Si se quisiera mostrar por pantalla qué valor se tiene en la posición 3 del vector, se escribe *System.out.println("El contenido en la posición 3 del vector es:" + vector[3])*. El listado 1.5 del capítulo 1 muestra un ejemplo de implementación de un arreglo unidimensional. Existen muchas maneras de declarar un vector unidimensional en Java. Los siguientes son ejemplos de sintaxis válidas para su declaración:

a) `int[ ] miArrreglo1 = {1,5,7,9,4,5,100};`
b) `String array[ ] = new String[n];`
c) `int [ ] miArreglo=new int [n];`

En el caso anterior de las declaraciones en el inciso a), se observa que al vector denominado *miArreglo1* se le asignan los valores 1, 5, 7, 9, 4,5 y 100 de forma inicial y que su tamaño es 7. En el inciso b) se declara un arreglo denominado *array* con un número de celdas dadas por la variable o constante *n*. Finalmente, para c) se declara un arreglo de forma diferente pero que en esencia es igual que b). En los casos b) y c), n es un valor entero que representa el número de celdas o elementos que contendrá el vector. El listado 2.1 muestra un ejemplo de uso de vectores:

Listado No. 2.1 Programación de vectores en Java

```
1.  package capitulo2;
2.  import java.util.Scanner;
3.  /**
4.   *
5.   * @author sunombre
6.   */
7.  public class Capitulo2_1 {
8.    int[ ] miArrreglo1 = {1,5,7,9,4,5,100};
9.    String array[ ] = new String[10];
10.   int [ ] miArreglo=new int [6];
11.   public static void main(String[ ] args) {
12.     Scanner entrada = new Scanner(System.in);
13.     float promedio = 0;
14.     float suma=0;
```

```
15.      int n=5;
16.
17.      for (int i = 0; i < n; i++) {
18.        System.out.print("Ingrese el número para la posición (" + i + ") :");
19.        miArreglo[i] = entrada.nextInt( );
20.      }
21.         // calcular la suma, el promedio y el mayor
22.     int mayor = miArreglo[0]; //el mayor inicia siendo el primer elemento del vector
23.     int menor = miArreglo[0]; //el menor inicia siendo el primer elemento del vector
24.     suma += miArreglo[0];
25.     for (int i = 1; i < miArreglo.length; i++) {
26.            suma += miArreglo[i];
27.            if (mayor < miArreglo[i]) {
28.                mayor = miArreglo[i];
29.            }
30.            if (menor > miArreglo[i]) {
31.                menor = miArreglo[i];
32.            }
33.      }
34.      //se muestran los valores de los elementos del arreglo
35.      for (int i = 0; i < miArreglo.length; i++)
36.         System.out.println(String.format("Posición [%d] Elemento: %d", i,
37.                                          miArreglo[i]));
38.      //calculando el promedio
39.      promedio = suma / miArreglo.length;
40.      System.out.println("****Salida de datos****");
41.      System.out.println(String.format("La suma es %.2f, "
42.              + "el promedio es %.2f, el mayor es %d, el menor es %d",
43.                       suma, promedio, mayor, menor));
44.   }
45. }
```

Sobre el código del listado 2.1:

Cabe aclarar algunos conceptos contenidos en dicho listado:

- Las líneas 8, 9 y 10 representan los tipos de declaración que se pueden realizar en Java para un vector unidimensional.

- La línea 12 tiene la sentencia *Scanner entrada = new Scanner(System.in)*; la cual hace declarar una variable (*entrada*) para que acepte una entrada de datos por teclado en una aplicación tipo consola.

- En la línea 35 aparece la sentencia *miArreglo.length*, que permite obtener el tamaño del vector, es decir, cuántos elementos posee.

- En las líneas 41 a 43 se encuentra la sentencia *System.out.println (String.format("La suma es %.2f, "+ "el promedio es %.2f, el mayor es %d, el menor es %d", suma, promedio, mayor, menor));* lo relevante es señalar la sentencia %.2f, que significa que el valor a mostrar tendrá un formato de tipo flotante y de dos decimales. %d, por su parte representa el formato para un valor entero. Así suma, promedio, mayor y menor, que son valores que se desplegarán por pantalla tendrán formato flotante, flotante, entero y entero, respectivamente.

La salida de datos que se produce al ejecutar el listado de programa 2.1 se muestra en la **figura 2.2**.

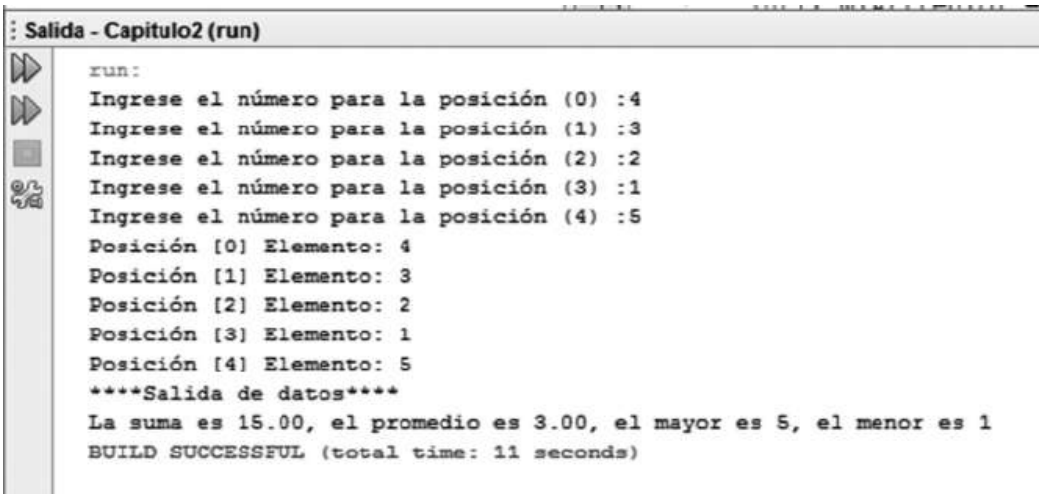


Figura 2.2 Ejecución de listado 2.1, uso de vectores en Java

Para el caso de un arreglo bidimensional los valores se encuentran en lo que se denomina una **matriz**, la cual está representada por filas y columnas. En este caso, en cualquier lenguaje de programación se debe utilizar un índice para las filas (que puede ser representado por i) y otro para las columnas (que puede ser representado por j). La **figura 2.3** muestra la estructura de una matriz (o arreglo bidimensional).

	Valores de j				
Valores de i	**[0]**	**[1]**	**[2]**	**[3]**	**[4]**
[0]	Alfonso	María	Juana	Pedro	Sebastián
[1]	Carlos	Romario	Francini	Marielos	Sofía
[2]	Francisco	Erika	Enrique	Edgar	Maritza

Figura 2.3 Estructura de un vector bidimensional

En la **figura 2.3** podemos observar valores para el índice de filas representado por i y valores para el índice de columnas representada por j; la figura se conforma por 3 filas y 5 columnas. Por ejemplo, para determinar el valor de la posición (i,j) = (0,1) se observa que es María (sombreado en la figura 2.3). Asimismo, para la posición (i,j) = (2,4) el valor es Maritza. Para implementar la funcionalidad explicada en esta sección se debe crear un programa en Java, similar al mostrado en el listado 2.2.

Listado No. 2.2 Programación de matrices en Java

```
1.  package Ejercicio2;
2.  import java.util.Scanner;
3.  /**
4.  * @author sunombre
5.  */
6.  public class Ejercicio2 {
7.    static final int rows = 3;
8.    static final int cols = 5;
9.    static String [ ][ ] nombres = new String[rows][cols];
10.   static void capturarDatos( ){
11.     String nombre;
12.     Scanner entrada = new Scanner(System.in);
13.     for (int i=0;i<rows;i++){
14.       for (int j=0;j<cols;j++){
15.          System.out.print("Ingrese el nombre para la posición "
16.          + "Fila("+i+") Columna("+j+")");
17.           nombres[i][j]=entrada.next( );
18.     }}
19.   }
20.   static void verDatos( ){
21.     for (int i=0;i<rows;i++){
```

```
22.       for (int j=0;j<cols;j++) {
23.         System.out.println("Posicion: Fila: " + i+
24.             " columna: "+j+" Nombre : "+nombres[i][j]);
25.         }
26.       }
27.     }
28.     static void dato( ){
29.          System.out.println("Posicion: Fila 0 Columna 1 " +nombres[0][1]);
30.          System.out.println("Posicion: Fila 2 Columna 4 " +nombres[2][4]);
31.       }
32.       public static void main(String[ ] args) {
33.            capturarDatos( );
34.            verDatos( );
35.            dato( );
36.     }
37. }
```

Sobre el código del listado 2.2:

Cabe aclarar algunos conceptos contenidos en dicho listado:

- Las líneas 7 y 8 declaran dos constantes para delimitar el número de filas y columnas que tendrá la matriz.

- La línea 9 contiene la sentencia *static String [ ][ ] nombres = new String[rows][cols]*; que indica que la matriz *nombres* tendrá el número de filas determinado por *rows* y el número de columnas determinado por *cols*.

La salida que mostraría la ejecución del programa del listado 2.2 se puede apreciar en la **figura 2.4**.

Para concluir el recorrido por las estructuras de arreglos estáticas, en última instancia se hará mención de los arreglos multidimensionales, los cuales se caracterizan por tener más de dos dimensiones. Algunas aplicaciones hacen necesario el uso de este tipo de arreglos, en la mayoría de los casos basta con un arreglo de 3 dimensiones, éste se puede imaginar como si fuera un cubo, donde cada cara es una matriz de nfilas x ncolumnas. La sintaxis de la declaración de este tipo de arreglos es la siguiente:

```
tipoDatonombreArreglo [ ] [ ] [ ] = new tipoDato [cantMatrices] [filas]
[columnas];
```

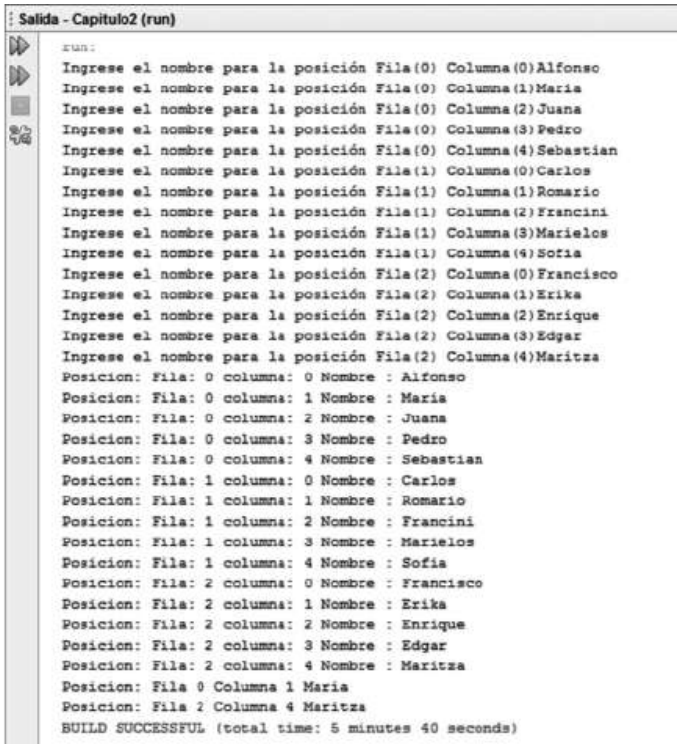


Figura 2.4 Salida del programa del listado 2.2

Para comprender este tipo de arreglos se puede pensar en un juego de bingo: si se pretende generar uno de los cartones del juego, se podría usar una matriz de 2 dimensiones (filas x columnas), pero si se necesita crear 100 cartones de 5 filas por 5 columnas (tamaño del cartón) ya no es factible crear 100 matrices de 2 dimensiones, sino que se puede usar un vector de matrices (arreglo de tres dimensiones), donde el primer índice definirá el número de cartón, mientras que el segundo y tercero indicarán la celda del cartón representado por la combinación, la fila y columna.

Es importante resaltar que en un cartón cada columna tiene un rango de números distinto: la primera va de 1 a 15, la segunda de 16 a 30, la tercera de 31 a 45, la cuarta de 46 a 60 y la última de 61 a 75.

Para implementar la funcionalidad de las matrices multidimensionales de 3 dimensiones se crea un programa de un bingo, el cual se muestra en el listado 2.3.

Listado No. 2.3 Programación de matrices de tres dimensiones en Java

```
1.  package ejercicio3;
2.  /**
3.   *
```

```
4.  * @author sunombre
5.  */
6.  public class Ejercicio3 {
7.    static int carton[ ][ ][ ] = new int[4][5][5];
8.    public static void main(String[ ] args) {
9.       int inicio;
10.     int num;
11.     //Llenando los cartones.
12.     for (int numCarton = 0; numCarton < 4; numCarton++) {
13.       inicio = 1;
14.       for (int fil = 0; fil < 5; fil++) {
15.         for (int col = 0; col < 5; col++) {
16.           num = (int) (Math.random( ) * 15) + inicio;
17.           while (buscaNum(numCarton, num)) {
18.              num = (int) (Math.random( ) * 15) + inicio;
19.           }
20.           carton[numCarton][col][fil] = num;
21.         }
22.          inicio = inicio + 15;
23.       }
24.        carton[numCarton][2][2] = 0;
25.    }
26.    mostrarCartones( );
27.  }
28.  //Método que verifica que el número no se repita en el cartón
29.  public static boolean buscaNum(int numCarton, int valor) {
30.    for (int f = 0; f < 5; f++) {
31.      for (int col = 0; col < 5; col++) {
32.         if (valor == carton[numCarton][f][col]) {
33.           return true;
34.         }
35.      }
36.    }
37.     return false;
38.  }
39.  //Muestra todos los cartones que existen en el arreglo multidimensional
40.   public static void mostrarCartones( ) {
```

```
41.      for (int numCarton = 0; numCarton < 4; numCarton++) {
42.        System.out.println("Carton: " + numCarton);
43.        for (int fil = 0; fil < 5; fil++) {
44.          for (int col = 0; col < 5; col++) {
45.            System.out.print(carton[numCarton][fil][col] + "\t");
46.          }
47.          System.out.println("");
48.        }
49.        System.out.println("");
50.      }
51.    }
52. }//Fin class Ejercicio3
```

Sobre el código del listado 2.3:

Cabe aclarar algunos conceptos contenidos en dicho listado:

- En la línea 7 se declara un arreglo multidimensional denominado carton de la forma `static int carton[ ][ ][ ] = new int[4][5][5]`; es importante aclarar que el primer índice señala cuántos cartones se crearán en el juego, el segundo y tercero muestran que cada cartón tendrá una matriz de 5 filas por 5 columnas.

- En la línea 9 se declara la variable inicio, la cual denota el comienzo por rango para cada columna de cada uno de los cartones. En la línea 13 se inicia la variable en 1 para cada cartón, que es el inicio del primer rango (1-15) y luego en la línea 22 se aumenta la variable en 15 para cambiar de rango en una nueva columna.

- En la línea 12 se inicia un ciclo for, el cual maneja el número del cartón a llenar, éste se denota por la variable local numCarton que inicia en 0 y termina en 3, para un total de 4 cartones.

- En la línea 16, `num = (int) (Math.random( ) * 15) + inicio`; se genera un número aleatorio a través del método random de la clase Math. Este número se produce en un rango que inicia en el valor de la variable inicio y que se extiende hasta 15 números hacia adelante, por ejemplo, si inicio=16, el rango en el que se generará el número aleatorio será de 16 a 30.

- La línea 17 permite buscar el número aleatorio producido en el `numCarton` actual, a través del método `buscaNum( )`, el cual devuelve falso si el número aún no existe en el cartón y verdadero en caso de que el mismo ya se haya generado. En este último escenario, por medio del ciclo while, se produce un nuevo número en el rango hasta

que se encuentre uno no repetido, ya que un número puede aparecer una sola vez en cada cartón.

- La línea 20 muestra la asignación del número generado en el cartón `numCarton`, en la fila fil y en la columna col.

- Respecto a la línea 24, se establece en la celda central del cartón un 0 ya que es común en los cartones de bingo que ésta aparezca vacía.

- Por último, en la línea 26, se llama al método `mostrarCartones( )`; el cual imprime por pantalla todos los cartones generados que existen en el arreglo tridimensional de cartones.

La salida que mostraría la ejecución del programa del listado 2.3 se puede apreciar en la **figura 2.5**.

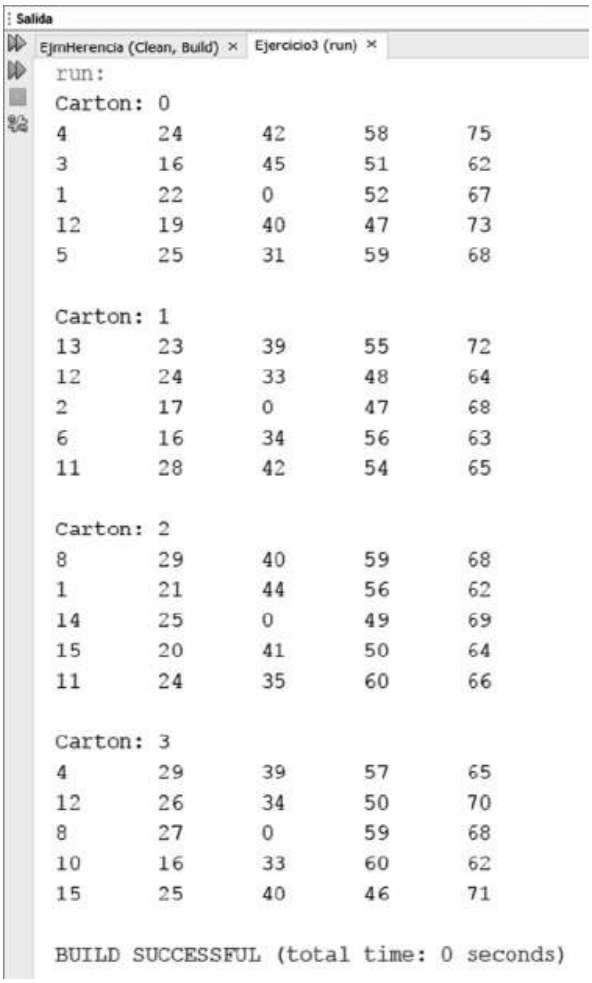


Figura 2.5 Salida del programa del listado 2.3

Para mayor más información acerca del manejo de arreglos puedes visitar los sitios:

- http://elvex.ugr.es/decsai/java/pdf/6A-Arrays.pdf
- https://sites.google.com/a/espe.edu.ec/programacion-ii/home/a1-arreglos/declaracion-manipulacion-y-asignacion-de-vectores
- http://www.mundojava.net/matrices-arrays-o-vectores-en-java.html?Pg=java_inicial_4_4_7.html
- http://puntocomnoesunlenguaje.blogspot.com/2012/12/matriz-en-java.html

2.3 Estructuras predefinidas en Java

Las estructuras de datos complejas como un vector o una matriz permiten manejar una gran cantidad de datos en una sola estructura. Sin embargo, por su construcción, son estructuras estáticas que tienen un tamaño limitado: hasta n filas en un arreglo unidimensional o n filas y m columnas finitas para una matriz; ello limita la cantidad de datos que se pueden almacenar en un programa y por ende la funcionalidad.

Por lo anterior, los lenguajes de programación han implementado en sus compiladores algunas estructuras complejas como ArrayList o HashMap, las cuales permiten crear estructuras dinámicas, sin límites de la cantidad de datos para almacenar o gestionar en ellas.

Java proporciona una serie de clases, éstas almacenan secuencias de objetos de diferente tipo, las cuales a su vez son llamadas **colecciones**. La diferencia entre unas y otras radica en la forma de organizar los objetos y, por lo tanto, en la manera en que se pueden recuperar. En las siguientes secciones se estudiarán las colecciones o arreglos dinámicos ArrayList y HashMap.

2.3.1 Clase ArrayList

La clase **ArrayList** de Java implementa la interfaz List. Se trata de una lista dinámica que no tiene por qué declararse de un tamaño fijo, como ocurre con los vectores o matrices descritos en la sección anterior. Una de las ventajas del ArrayList es que se indexan sus valores, por lo que es posible acceder directamente a una posición del mismo.

Los ArrayList, al ser una lista redimensionable, disponen de métodos habituales para su operación; esto significa que se pueden añadir objetos en una posición dada, recuperar desde una ubicación determinada, eliminarlos, etc. La tabla 2.1 muestra algunas de las operaciones posibles en un ArrayList.

Tabla 2.1 Operaciones ArrayList en Java

FUNCIONALIDAD	DESCRIPCIÓN
`size( )`	Devuelve un entero del número de elementos del ArrayList.
`add(x)`	Agrega el objeto x al final del ArrayList. Devuelve true.
`add(x,y)`	Agrega el objeto y en la posición dada por x.
`get(posicion)`	Devuelve el elemento que se encuentra en la posición indicada.
`remove(posicion)`	Elimina el elemento que se encuentra en la posición indicada.
`remove(x)`	Elimina la primera ocurrencia del objeto x. Devuelve true.
`clear( )`	Elimina todos los elementos del ArrayList.
`set(posicion,x)`	Sustituye el elemento que se encuentra en la posición indicada por el objeto x. Devuelve el elemento que se sustituye.
`constains(x)`	Verifica la existencia del elemento x. Devuelve true o false.
`indexof(x)`	Devuelve la posición de la primera ocurrencia del objeto x, de lo contrario devuelve -1.
`lastIndexof(x)`	Devuelve la posición (índice) en que se encuentra el valor de búsqueda x de la última ocurrencia, de lo contrario devuelve -1.
`isEmpty( )`	Devuelve True si el ArrayList está vacío; si no, devuelve False.

Un aspecto importante en el uso y manejo de los ArrayList son los iteradores (Iterator). La función de éstos es recorrer el ArrayList para poder trabajar con ellos. Para operar con la clase Iterator existen tres opciones, las cuales incluyen los métodos mostrados en la **tabla 2.2.**

Tabla 2.2 Métodos iteradores en Java

ITERADOR	DESCRIPCIÓN
`hashNext ( )`	Comprueba si quedan elementos en el arreglo. Devuelve verdadero si los hay.
`next( )`	Devuelve el siguiente elemento en la iteración.
`remove( )`	Sirve para eliminar el elemento del iterador.

Declaración y creación de un ArrayList

Existen dos formas de declarar y crear un ArrayList, su uso depende de la necesidad del programador. La primera es factible si se desea almacenar objetos de diferentes clases en el mismo ArrayList. En este caso, la declaración se haría:

```
  ArrayList nomArrayList = new ArrayList( );
//Crea una lista vacía, llamada nomArrayList
```

A continuación se muestra un ejemplo donde se visualiza la funcionalidad del ArrayList. Se crea el ArrayList denominado lista:

```
ArrayList lista = new ArrayList( );
```

Se pueden agregar elementos de la siguiente manera:

```
a) lista.add("Lista dinámica");
b) lista.add(36);
c) lista.add('j');
d) lista.add(31.1);
```

Tras analizar el código anterior, se puede observar que el primer objeto que se agregó fue un String "Lista dinámica", el resto de elementos 36, j, 31.1, no son objetos sino datos primitivos, por lo que antes de agregarlos al ArrayList el compilador los convierte en objetos. En conclusión, en el ejemplo se agregaron objetos de diferentes tipos.

La segunda forma de declaración de un ArrayList es factible cuando se desea almacenar en éste objetos del mismo tipo, su declaración es la siguiente:

```
ArrayList<tipo> nombreArray = new ArrayList<tipo>( );
//Donde tipo debe ser una clase.
```

A continuación se muestra un ejemplo:

```
ArrayList<Integer> edades = new ArrayList<Integer>( );
```

En este caso se crea un ArrayList de números enteros denominado edades, que almacena edades de personas.

Recorrido de un ArrayList

Para recorrer un ArrayList existen diferentes formas: la primera es la tradicional con un bucle for.

```
for(int i = 0;i<lista.size( );i++){
            System.out.println(lista.get(i));
}
```

La segunda forma es mediante un ciclo foreach. Tras suponer que se necesita recorrer el array de enteros ya creado, se hace de la siguiente manera:

```
for(Integer i: edades){
           System.out.println(i);
}
```

El tercer camino para recorrer una lista de tipo ArrayList es bajo el supuesto de que existe una lista objetos de distintos tipos, lo cual se hace de la siguiente manera:

```
for(Object o: lista){
            System.out.println(o);
}
```

Por último, se pueden utilizar los iteradores descritos en la tabla 2.2. El beneficio de usar un objeto de la clase Iterador es que no se necesita especificar el tipo de objeto que contiene el ArrayList, por ejemplo:

```
ArrayList<Integer> edades = new ArrayList<Integer>( ); //se declara el array
de enteros
Iterator itera = edades.iterator( ); //se crea el iterador itera para el array edades

//se agregan los elementos a la lista
edades.add(15);
edades.add(23);
edades.add(55);
edades.add(9);
edades.add(3);

//recorriendo el ArrayList para mostrarlo por pantalla
while(itera.hasNext( )){          //mientras queden elementos
     System.out.println(itera.next( ));   //se obtienen y se muestran
 }
```

Uso del ArrayList como parámetro de un método

Un ArrayList puede ser enviado como parámetro a un método y, al mismo tiempo, un ArrayList puede ser devuelto por un método.

La sintaxis para su declaración es la siguiente:

```
public static ArrayList<Integer> ordenar(ArrayList<Integer> numeros) {
     //Cuerpo del método
}
```

En el ejemplo anterior se tiene un método denominado ordenar, que recibe por parámetro un ArrayList de enteros; el parámetro que recibe la lista se llama números. Por otro lado, se observa que el método ordenar devuelve un ArrayList de enteros.

Para implementar la funcionalidad del ArrayList explicada en esta sección, se creará un programa en Java que almacena nombres de empleados de una empresa; el código se muestra en el listado 2.4.

Listado No. 2.4 Almacenamiento de datos en lista dinámica a través de ArrayList en Java

```
1.  package ejercicio4;
2.      /**
3.      * @author sunombre
4.  */
5.  import java.util.ArrayList;
6.  import java.util.Scanner;
7.  public class Ejercicio4 {
8.    static ArrayList<String> listaEmpl = new ArrayList<String>( );
9.    public static void main(String[ ] args) {
10.     Scanner entrada = new Scanner(System.in);
11.     int opcion;
12.     String empl;
13.     do {
14.       System.out.println("\n");
15.       menu( );
16.       opcion = entrada.nextInt( );
17.       switch (opcion) {
18.         case 1://Insertando
19.                     System.out.println("Insertando un Empleado");
20.                     entrada.nextLine( ); //Limpia el buffer
21.                     System.out.println("Nombre: ");
22.                     empl = entrada.nextLine( );
23.                     listaEmpl.add(empl); //Se agrega el objeto Empleado a la lista
```

```
24.              break;
25.           case 2://Actualizando
26.                        String nuevoEmp = new String( );
27.                        boolean bandera = false;
28.                        System.out.println("\n");
29.                        entrada.nextLine( );//Limpia el bufer
30.                        System.out.println("Nombre del empleado a modificar: ");
31.                        empl = entrada.nextLine( );
32.                        for (int pos = 0; pos < listaEmpl.size( ); pos++) {
33.                          if (empl.compareTo(listaEmpl.get(pos)) == 0) {
34.                                System.out.println("Nuevo nombre del empleado: ");
35.                                nuevoEmp = entrada.nextLine( );
36.                                listaEmpl.set(pos, nuevoEmp);
37.                                System.out.println("\n\nRegistro Actualizado");
38.                                bandera = true;
39.                                break;
40.                           }
41.                      }
42.                      if (bandera == false) {
43.                          System.out.println("El empleado no existe");
44.                      }
45.                      break;
46.           case 3://Eliminando
47.                    bandera = false;
48.                    char resp;
49.                    System.out.println("\n");
50.                    entrada.nextLine( );
51.                    System.out.println("Nombre del empleado a eliminar: ");
52.                    empl = entrada.nextLine( );
53.                    for (String e : listaEmpl) {
54.                    if (e.equals(empl)) {
55.                         System.out.println("Desea eliminar el empleado  (s/n)");
56.                         resp = entrada.nextLine( ).toLowerCase( ).charAt(0);
57.                         if (resp == 's') {
58.                               listaEmpl.remove(e);
59.                            System.out.println("\n\nEmpleado Eliminado");
```

```
60.                                 bandera = true;
61.                             }
62.                             break;
63.                          }
64.                  }
65.                  if (bandera == false) {
66.                         System.out.println("El empleado no existe");
67.                     }
68.                     break;
69.                 case 4://Mostrar Empleados
70.                     System.out.println("\nLista de Empleados\n");
71.                     if (listaEmpl.isEmpty( )) {
72.                         System.out.println("No hay registros en la lista");
73.                     } else {
74.                         listaEmpl.forEach((e) -> {
75.                             System.out.println(e);
76.                         });
77.                     }
78.                     break;
79.                 case 5://Salir
80.                     break;
81.                 default://Sino es ninguna de las opciones anteriores
82.                     System.out.println("Opción Inválida");
83.                     break;
84.             }
85.         } while (opcion != 5);
86.     }
87.     public static void menu( ) {
88.         System.out.println("Menú Principal");
89.         System.out.println("1. Incluir Empleado");
90.         System.out.println("2. Actualizar Empleado");
91.         System.out.println("3. Eliminar Empleado");
92.         System.out.println("4. Mostrar Empleados");
93.         System.out.println("5. Salir");
94.         System.out.println("Opción: ");
95.     }
96. }
```

Sobre el código del listado 2.4:

Cabe aclarar algunos conceptos contenidos en dicho listado:

- En la línea 2, es vital importar la clase ArrayList para hacer uso de ella en nuestro programa.
- En la línea 5 se declara el ArrayList listaEmpl utilizado en el ejemplo para almacenar a los empleados.
- En la línea 20 se agrega el empleado al ArrayList a través del método add.
- La variable denominada bandera, que aparece en la línea 24, se usa en la modificación para verificar la existencia del elemento que se pretende actualizar o borrar; si existe su valor es true y procede con la operación, pero si es false significa que el elemento que se pretende actualizar o eliminar no existe en la lista.
- Con la línea 33 se actualiza al empleado a través del método set, que se encuentra en la posición pos por el contenido en la variable nuevoEmp.
- Como se observa en la línea 55, se usa el método remove para eliminar el elemento de la lista.
- Por último, en la línea 71 se usa un bloque forEach para recorrer la lista y mostrar por pantalla a los empleados existentes.

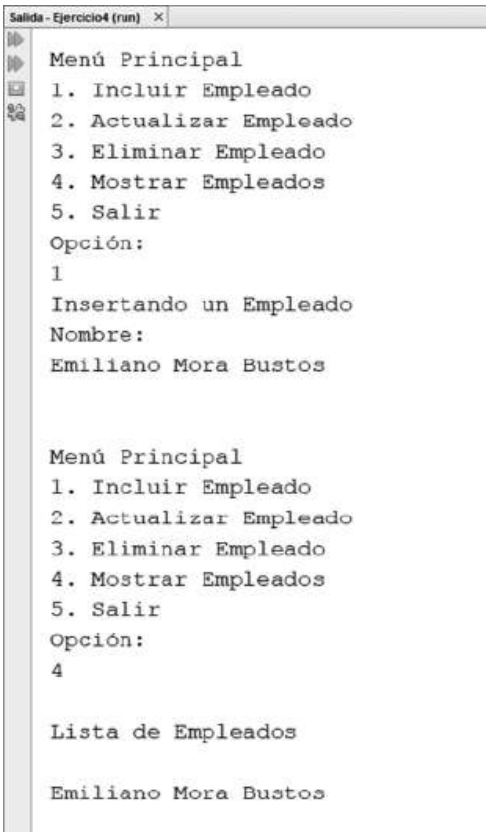


Figura 2.6 Salida del programa del listado 2.4

La ejecución del programa del listado 2.4 y su salida se puede apreciar en la **figura 2.6**.

2.3.2 HashMap

Es una estructura de datos dinámica que implementa la interfaz Map y asocia claves con valores, en ella se pueden almacenar de manera dispersa (sin un orden) pares de datos de la forma clave-valor, de modo que para una clave sólo se tiene un valor y viceversa; además, las claves no se pueden repetir (si se agrega un elemento con la misma, el valor se sobrescribirá). Estas estructuras poseen métodos ya programados que permiten insertar, buscar, modificar y eliminar con facilidad.

Se llama **HashMap** porque hace referencia a una técnica de organización de archivos denominada dispersión o hashing, ya que se puede acceder a los elementos almacenados mediante una clave generada (hash) a partir de una función hash, éste indica la posición donde está almacenado el elemento.

La principal función de este tipo de estructuras es su método de búsqueda eficiente sin necesidad de tener los datos ordenados, ya que puede acceder aleatoriamente al valor por medio de la clave.

Para entender la diferencia entre el ArrayList y el HashMap se puede hacer la siguiente analogía: un ArrayList visto como una fila de estudiantes esperando ser atendidos en una cafetería, mientras que HashMap es el salón de clases lleno de estudiantes dispersos a los cuales el profesor llama por su nombre (clave-valor) para hacer entrega de un examen calificado.

La **tabla 2.3** muestra algunos de los métodos que posee el HashMap

Tabla 2.3 Métodos de HashMap en Java

FUNCIONALIDAD	DESCRIPCIÓN
`clear( )`	Elimina todos los elementos del HashMap
`containsKey(Object clave)`	Devuelve true si el mapa contiene la clave pasada como parámetro
`get(Object clave)`	Retorna el valor del HashMap para la clave pasada como parámetro
`isEmpty( )`	Devuelve True si el HashMap está vacío, de lo contrario devuelve False
`put(Object clave, V value)`	Inserta un par clave/valor en el mapa mediante los valores pasados como parámetros

FUNCIONALIDAD	DESCRIPCIÓN
`remove(Object clave)`	Elimina el elemento del mapa denotado por la clave pasada por parámetro
`size( )`	Retorna el número de elemento clave/valor que tiene el HashMap

Declaración y creación de un HashMap

Para declarar y crear un HashMap, se puede hacer de la siguiente forma:

```
HashMap<Clave, Valor> nomHashMap = new HashMap<>( );
```

En la declaración se crea un HashMap denominado nomHashMap, que almacenará un par de elementos de la forma clave-valor, los cuales están denotados en la declaración por un tipo de dato o clase Clave, para el caso de la clave y por un tipo de dato o clase Valor para el valor que será almacenado. Un ejemplo de ello:

```
HashMap<String, Float> listaProductos = new HashMap<>( );
```

En el HashMap listaProductos, se almacenarán pares clave-valor donde la clave debe ser un valor String y el valor será un número decimal Float.

Para ejemplificar ampliamente el uso de HashMap, imagine que es necesario tener una estructura en la que se almacenan los productos de un supermercado y sus respectivos precios. La finalidad es tener una lista resumida de los productos (nombre-precio) en la que se puedan consultar eficientemente los precios ofrecidos, así como su cambio si se requiere. En este ejemplo la clave será el nombre del producto y el valor será su precio. En el listado 2.5 se muestra el código fuente de dicho programa.

Listado No. 2.5 Ejemplo del uso de HashMap en Java

```
1.  package ejercicio5;
2.  /**
3.   * @author sunombre
4.  */
5.  import java.util.HashMap;
6.  import java.util.Iterator;
7.  import java.util.Scanner;
8.  public class Ejercicio5 {
9.      public static void main(String[ ] args) {
```

```
10.      HashMap<String, Float> listaProductos = new HashMap<>( ); //Declaración
11.        Scanner sc = new Scanner(System.in); //objeto sc para lectura en consola
12.          int opcionElegida = 0;
13.          float precio;
14.          String codigo;
15.          while (opcionElegida != 5) {
16.              System.out.println("Menú de opciones:");
17.              System.out.println("1.- Insertar producto");
18.              System.out.println("2.- Modificar precio");
19.              System.out.println("3.- Mostrar todos los productos");
20.              System.out.println("4.- Eliminar producto");
21.              System.out.println("5.- Salir");
22.             opcionElegida = sc.nextInt( ); //Se lee la opción elegida del menú
23.               switch (opcionElegida) {
24.          case 1://Insertar un producto al HashMap
25.            System.out.println("Digite el códido del producto:");
26.                       codigo = sc.next( );
27.                       System.out.println("Digite el precio del producto:");
28.                       precio = sc.nextFloat( );
29.                        guardarProducto(codigo, precio, listaProductos); //Mé-
todo que inserta
30.                       break;
31.          case 2://Modifica un producto ya existente
32.           System.out.println("Introduce el códido del producto a cambiar:");
33.                       codigo = sc.next( );
34.                        modificaPrecio(codigo, listaProductos);//Método que mo-
difica precio
35.                       break;
36.          case 3://Muestra la lista de productos
37.            mostrarProductos(listaProductos);//Muestra todos los productos
38.                       break;
39.          case 4://Elimina un producto existente
40.                      System.out.println("Introduce el código del producto a
eliminar:");
41.                       codigo = sc.next( );
42.                     eliminaProducto(codigo, listaProductos);//Método que elimina
43.                       break;
```

```
44.         case 5:
45.            break;   // Si la opcion es 5 no se hace nada
46.         default:
47.                     System.out.println("Opción inválida");
48.             }
49.             System.out.println("\n");
50.         }
51.     }//Fin del main
52.     public static void guardarProducto(String codigo, float precio, HashMa-
    p<String,
    Float> listaProductos) {
53.         if (listaProductos.containsKey(codigo)) { //No permite duplicados
54.             System.out.println("Imposible introducir el producto. El código
    repetido.");
55.         } else {
56.              listaProductos.put(codigo, precio);//Guarda el producto en el
    HashMap
57.         }
58.     }
59.     public static void modificaPrecio(String codigo, HashMap<String, Float>
    listaProductos) {
60.         Scanner sc = new Scanner(System.in);
61.         if (listaProductos.containsKey(codigo)) {
62.             System.out.println("Introduce el nuevo precio del producto:");
63.             listaProductos.put(codigo, sc.nextFloat( )); //Actualiza el precio
64.         } else {
65.             System.out.println("No hay ningun producto con ese código.");
66.         }
67.     }
68.     public static void mostrarProductos(HashMap<String, Float> listaProduc-
    tos) {
69.         String clave;
70.         Iterator<String> productos = listaProductos.keySet( ).iterator( );
71.         System.out.println("Listado de productos:\n");
72.         System.out.println("Código\t Precio");
73.         while (productos.hasNext( )) {
74.             clave = productos.next( );
75.             System.out.println(clave + "\t " + listaProductos.get(clave));
```

```
76.         }
77.     }
78.     public static void eliminaProducto(String codigo, HashMap<String, Float>
        listaProductos) {
79.         if (listaProductos.containsKey(codigo)) {
80.             listaProductos.remove(codigo);
81.       } else {
82.             System.out.println("No hay ningun producto con ese código.");
83.         }
84.       }
85.     } //Fin clase Ejercicio5
```

Sobre el código del listado 2.5:

Algunos comentarios en relación con dicho código se citan a continuación:

- En la línea 29 se llama al método guardarProducto(codigo, precio, listaProductos); el cual permite guardar el código en la clave y el precio en el valor del HashTable listaProductos. Esto valores son pasados por parámetro al método en el cual se efectúa la operación en el HashTable.

- En la línea 34 se invoca al método modificaPrecio(codigo, listaProductos); el cual permite actualizar el precio de un producto identificado por el parámetro clave codigo por un nuevo valor en el HashTable listaProductos.

- En la línea 53 se utiliza el método listaProductos.containsKey(codigo), el cual verifica que el código que se intenta insertar no exista con anterioridad en el HashMap, si existe impide la sobre escritura y se imprime un mensaje.

- En la línea 56 y 63 el método put (clave, valor) permite insertar un par clave-valor en la primera, o en la 63 permite actualizar el valor de la clave que recibe por parámetro.

- El código mostrado en la línea 70 permite crear un iterador para recorrer el HashMap listaProductos.

- Por último, en la línea 80, mediante el método remove y la clave pasada por parámetro, se puede eliminar el elemento del HashMap.

La muestra de salida de ejecución del programa del listado 2.5 se puede apreciar en la **figura 2.7**.

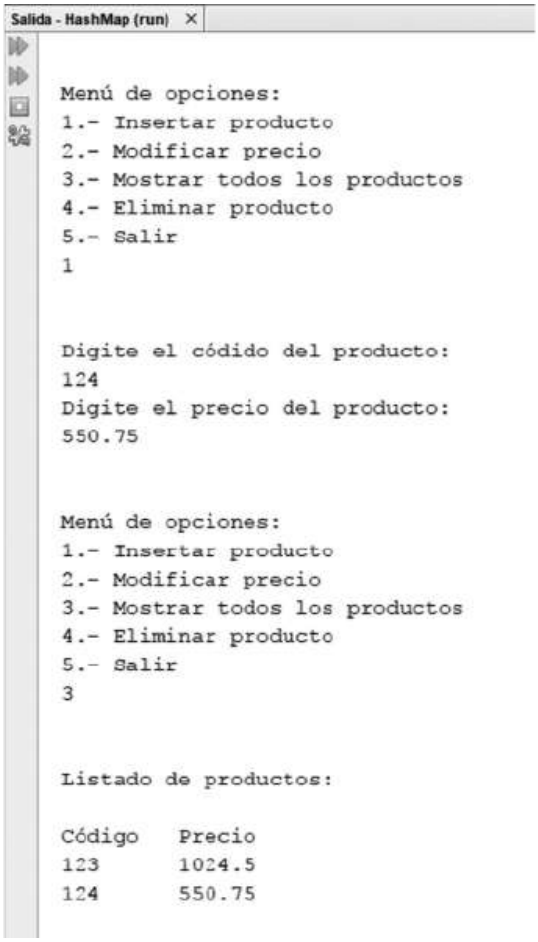


Figura 2.7 Salida del programa del listado 2.5

2.4 Manejo de concurrencia en Java

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

Se puede afirmar que un **hilo** es una secuencia de instrucciones controlada por un planificador del sistema operativo: este planificador controla el tiempo de ejecución del hilo en el procesador y asigna 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. En el caso de un navegador web, es posible que una aplicación permita 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 de que los primeros se ejecutan dentro del contexto de un programa y los procesos lo hacen en su propio espacio de direcciones y datos. Es decir, los hilos dependen de los recursos de un programa superior (o padre) mientras que un proceso no (tiene sus propios recursos).

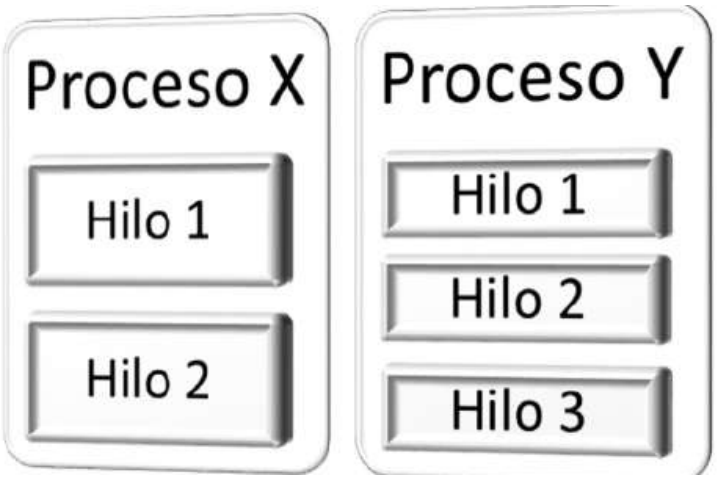


Figura 2.8 Relación y diferencia entre procesos e hilos

En la **figura 2.8** se muestra la relación existente entre procesos e hilos (o de cómo los hilos existen dentro de los procesos).

2.4.1 La programación concurrente en Java

Java 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 *start()* para hacer que funcione 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, ésta y las clases se resumen en la **tabla 2.4** mostrada a continuación.

Tabla 2.4 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 a través del método run y hereda otros métodos como start, stop (no recomendado), 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, pues se pretende la gestión eficiente en la ejecución de varios hilos. Asimismo, se proporcionan los métodos stop, suspend y resume para controlar la ejecución.
Object	Esta clase 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 avisa al hilo que está suspendido mediante wait que continúe su ejecución. Finalmente, notifyAll es similar a notify, con la diferencia de que la notificación es a todos los hilos que están suspendidos.

Como se señaló anteriormente, los hilos comparten el mismo espacio de memoria y los recursos de un proceso determinado. Además, 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 vía (solo procesos).

Para crear hilos en Java existen dos maneras: con la interfaz Runnable o heredando de la clase Thread. A pesar de que Runnable es más recomendable, en algunos procesos se utilizará la derivación de Thread para crear ejemplos propios de hilos.

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

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

En el ejemplo anterior se crea una clase denominada MiHilo, la cual hereda de la clase Thread; al mismo tiempo se sobrescribe el método Thread.run() por su propia implementación. En el método run ( ) se realizará todo el trabajo de la clase.

Si recordamos qué sucede cuando una clase derivada hereda de una padre, se puede decir que ésta recibe de su padre tanto atributos como métodos, pero con la limitación de que en Java sólo existe la herencia simple, es decir, sólo se puede heredar de un padre. Esta limitación del lenguaje es solucionada por la segunda forma de creación de hilos, la cual consiste en implementar la interfaz Runnable.

La sintaxis para crear hilos en Java mediante la interfaz Runnable es la siguiente:

```
public class MiHilo implements Runnable {
    Thread t;
           public void run( ) {
           // Se ejecuta el hilo después de crearlo.
       }
 }
```

En el ejemplo anterior se crea una clase llamada MiHilo, que implementa la interfaz Runnable. Como se aprecia, es necesario generar una instancia de Thread, en este caso representada por t, para luego ejecutar el proceso como un hilo.

Por otro lado, también es necesario implementar el método run ( ) de la interfaz Runnable. Esta última manera es mucho más flexible que la primera. Se debe recordar que una interfaz es sencillamente la especificación de la estructura de diseño de una clase, en este caso de la clase Thread.

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 son: New, Runnable, Not running o Dead, los mismos se describen a continuación.

- **New**: Es el estado inicial. Se usa cuando se crea inicialmente un hilo y posteriormente es llamado a ejecución por el método start().

- **Runnable**: Una vez que se invoca a start(), el método run() es llamado y el hilo entra en un estado de ejecución (runnable).

- **Not running**: Se aplica a todos los hilos detenidos por alguna circunstancia. En este caso, está a la espera de ser llamado para volver a ejecutarse. Los posibles estados de

que no se encuentre en ejecución es porque ha sido suspendido (mediante suspend), está 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 el método main() es la primera función que se invoca tras ejecutarse un programa. Existe sólo un main() por programa. En esta etapa 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) );
```

La instrucción anterior permite la creación de un hilo nuevo denominado hilo basado en la implementación de la clase HiloUno. Los parámetros pasados a la clase son "Hilo 1" e (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 de programación concurrente

A continuación se desarrollará un ejemplo sencillo de hilos en Java, el cual se basará en las especificaciones para un proyecto programado que se utiliza frecuentemente en los cursos de programación. El enunciado es el siguiente:

Para realizar este proyecto es aconsejable estudiar los capítulos 3 (programación orientada a objetos) y el capítulo 4 (aplicaciones de escritorio). En el capítulo de aplicaciones de escritorio se explica el uso de paquetes en Java (packages). El proyecto presentado a continuación está basado en interfaces gráficas (formularios Windows®) y el uso de clases y objetos.

Enunciado del proyecto:

Se requiere programar el comportamiento de un juego que ejecute 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 se repite alguna posición generada en el vector, se debe hacer nuevamente ya que deben resultar tres campos del vector llenos con -1.

Nota: *La posición 0 y la posición 24 no deben quedar con -1.*

	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 comienzan a ejecutarse.

d) Participarán 3 jugadores, cada uno de los cuales es un hilo e inicia en la posición cero del JTable. El jugador lleva el control de su posición en la JTable, y cada uno tiene un nombre y un número asignados. Se realizarán las siguientes actividades:

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

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

- El programa deberá cerciorarse de que el valor de la variable avance con respecto a la posición actual del jugador, ya que ésta 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 **avanza** contiene un **-1** éste debe devolverse a la **posición 0**.
  iii. Una vez que finaliza el turno del jugador, éste 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 competidores.

1.1 Programe la clase padre correspondiente al Juego. Debe programarse 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.

1.3 Programe el formulario. Se deben declarar e inicializar los hilos, además de mostrar su funcionamiento.

Una propuesta de solución se observa con la creación del siguiente programa.

1. Ingrese a NetBeans y cree un proyecto tipo Java Application de la categoría Java. Nombre este proyecto *dskHilos*.

2. Haga clic derecho sobre el nombre del proyecto dskHilos y en el menú de contexto que se muestra seleccione Java Package de la opción *Nuevo*. Use *Logica* como nombre de esta carpeta.

3. Repita el paso anterior y cree otro paquete (carpeta) denominado *Fisico*.

4. Una vez creados los paquetes pulse con el botón derecho del mouse sobre el paquete *Fisico* y cree un nuevo objeto *JFrame* de nombre *FormaHilos*. La interfaz que debe tener este formulario se muestra en la **figura 2.9**.

Figura 2.9 Formulario FormaHilos

Se observa que los objetos (controles) aparecen con su nombre y tipo en la figura 2.9, abajo se ubica una serie de etiquetas y botones. Las tres etiquetas que aparecen debajo de los JTable se denominan *lblIntJugador1*, *lblIntJugador2* y *lblIntJugador3*, respectivamente. Sin embargo, debajo de éstas aparecen 3 botones de nombre: *btnJugar*, *btnReinicioJuego* y *btnGenerarTabla*.

5. Ahora se crean las clases que permitirán el funcionamiento de los tres jugadores; se deben crear dentro del JavaPackage Logica. Generar la primera clase y nombrarla clsJugador1. El código se muestra en el listado 2.6:

Listado No. 2.6 Código fuente de la clase clsJugador1

```
1.  package Logica;
2.  import javax.swing.JOptionPane;
3.  //se crea la clase clsJugador1 y toda la funcionalidad de la misma con
    herencia de la clase Thread
```

```
4.  public class clsJugador1 extends Thread {
5.  private int numero; //atributo para definir el número de jugador.
6.  private String nombre; //atributo para definir el nombre del jugador
7.  private boolean continuar = true;
    //flag para saber si continua el procedimiento
8.  public clsJugador1( ) {  //implementación del constructor.
9.     numero = 1;  //le asigna el numero 1 al jugador 1.
10.    nombre = JOptionPane.showInputDialog(null,"Digite el nombre del Jugador 1.");
11. }
12. public int getNumero( ){ return numero; }
    //para devolver el número del jugador
13. public String getNombre( ){return nombre;}
14. public Fisico.FormaHilos formHilo;
    //declara una variable objeto de tipo FormaHilos
15. public clsJugador1(Fisico.FormaHilos aThis) { //constructor que recibe por
    defecto el formulario FormaHilos// y lo asigna a formHilo.
16.     formHilo = aThis;
17. }
18. public void detenerHilo( ){ //método que detiene la ejecución del hilo
19.     continuar = false;
20. }
21. @Override //se sobrescribe el método run
22. public void run( ) {
23.     while (continuar) {   //hágase mientras continuar sea verdadero
24.         try {
25.             int avance = (int) (Math.random( ) * 4);
                //establece el avance de jugador 1
26.             formHilo.moverJugador1(avance);
                //invoca al método moverJugador1 que se
                //encuentra en el formulario FormaHilos
27.             continuar = formHilo.seguir;
                //para validar si se debe continuar
28.             Thread.sleep(150); //descansa por 15 segundos...
29.         } catch (InterruptedException ex) {
30.         }
31.     }
32.  }
33. }
```

6. Se crea la clase clsJugador2. El código fuente para esta clase se muestra en el listado 2.7:

Listado No. 2.7 Código fuente de la clase clsJugador2

```
1.  public class clsJugador2 extends Thread {
2.      private int numero;  //atributo para definir el número de jugador.
3.      private String nombre; //atributo que define el nombre del jugador
4.      private boolean continuar = true;
        //flag para saber si continua el procedimiento
5.      public clsJugador2( ) {   //implementación del constructor.
6.          numero = 2;  //le asigna el numero 2 al jugador 2.
7.          nombre = JOptionPane.showInputDialog(null,"Digite el nombre
            del Jugador 2.");
8.      }
9.      public int getNumero( ){ return numero; }
        //para devolver el número del jugador
10.     public String getNombre( ){ return nombre; }
11.     public Fisico.FormaHilos formHilo;
        //declara una variable de tipo FormaHilos
12.     public clsJugador2( Fisico.FormaHilos aThis ) {
        //constructor que recibe por defecto el formulario FormaHilos
        // y lo asigna a formHilo.
13.         formHilo = aThis;
14.     }
15.     public void detenerHilo( ){
        //método para detener la ejecución del hilo.
16.         continuar = false;
17.     }
18.     @Override //se sobrescribe el método run
19.     public void run( )   {
20.         while (continuar) {  //hágase mientras continuar sea verdadero
21.             try {
22.                 int avance = (int) (Math.random( ) * 4);
                    //establece el avance de jugador 2
23.                 formHilo.moverJugador2(avance);
                    //invoca al método moverJugador2 que se
                    //encuentra en el formulario FormaHilos
24.                 continuar = formHilo.seguir;
                    //para validar si se debe continuar.
25.                 Thread.sleep(150); //descansa por 15 segundos...
26.             } catch (InterruptedException ex) {
27.             }
28.         }
29.     }
30. }
```

7. La clase para el jugador 3 posee el mismo código que para las dos clases anteriores. Dicho código se visualiza en el listado 2.8:

Listado No. 2.8 Código fuente de la clase clsJugador3

```
1.  public class clsJugador3 extends Thread {
2.      private int numero;  //atributo para definir el número de jugador.
3.      private String nombre; //atributo para definir el nombre del jugador
4.      private boolean continuar = true;
        //flag para saber si continua el procedimiento
5.      public clsJugador3( ) {   //implementación del constructor.
6.          numero = 3;  //le asigna el numero 3 al jugador 3.
7.          nombre = JOptionPane.showInputDialog(null,"Digite el nombre
            del Jugador 3.");
8.      }
9.      public int getNumero( ){ return numero; }
        //para devolver el número del jugador
10.     public String getNombre( ){ return nombre; }
11.     public Fisico.FormaHilos formHilo;
        //declara una variable de tipo FormaHilos
12.     public clsJugador3( Fisico.FormaHilos aThis ) {
        //constructor que recibe por defecto el formulario FormaHilos
13.         formHilo = aThis;     // y lo asigna a formHilo.
14.     }
15.     public void detenerHilo( ){
        //método para detener la ejecución del hilo.
16.         continuar = false;
17.     }
18.     @Override //se sobrescribe el método run
19.     public void run( )   {
20.         while (continuar) {   //hágase mientras continuar sea verdadero
21.             try {
22.                 int avance = (int) (Math.random( ) * 4);
                    //establece el avance de jugador 3
23.                 formHilo.moverJugador3(avance);
                    //invoca al método moverJugador3 que se
                    //encuentra en el formulario FormaHilos
24.                 continuar = formHilo.seguir;
                    //para validar si se debe continuar.
25.                 Thread.sleep(150); //descansa por 15 segundos...
26.             } catch (InterruptedException ex) {
27.             }
28.         }
29.      }
30. }
```

8. Una vez que se ha escrito el código para las tres clases que procesarán a cada jugador, se escribe el código que radicará en el formulario principal (FormaHilos.Java), el cual se observa en el listado 2.9:

Listado No. 2.9 Código fuente de la clase FormaHilos

```
1.  package Fisico;
2.  import Logica.clsJugador1; //se importa la clase clsJugador1
3.  import Logica.clsJugador2; //se importa la clase clsJugador2
4.  import Logica.clsJugador3; //se importa la clase clsJugador3
5.  import javax.swing.JOptionPane;
6.  //se importa la clase para el manejo de modelo de tablas (JTable)
7.  import javax.swing.table.DefaultTableModel;
8.  public class FormaHilos extends javax.swing.JFrame {
9.      //se crea variable modeloTabla para apoyar el manejo de JTable tabla
10.     //vea figura 2.9
11.     javax.swing.table.DefaultTableModel modeloTabla = new javax.swing.
        table.DefaultTableModel( );
12.     //se crea variable modeloTabla para apoyar el manejo de JTable
          tablaJugador1
13.     // vea figura 2.9
14.     avax.swing.table.DefaultTableModel modTablaJugador1 = new javax.swing.
        table.DefaultTableModel( );
15.     //se crea variable modeloTabla para apoyar el manejo de JTable
    tablaJugador2
16.     // vea figura 2.9
17.     javax.swing.table.DefaultTableModel modTablaJugador2 = new javax.
        swing.table.DefaultTableModel( );
18.     //se crea variable modeloTabla para apoyar el manejo de JTable
        tablaJugador3
19.     // vea figura 2.9
20.     javax.swing.table.DefaultTableModel modTablaJugador3 = new javax.
        swing.table.DefaultTableModel( );
21.     //se crea un objeto "filas" de 24 elementos para
        almacenar las columnas de la tabla "tabla"
22.     Object[ ] filas = new Object[24]; //para datos del jugador1
23.     Object[ ] datosJugador1 = new Object[3];
24.     //para manejo de las columnas celdaAvance, Contenido,nuevaCelda
25.     Object[ ] datosJugador2 = new Object[3]; //para datos del jugador2
26.     Object[ ] datosJugador3 = new Object[3]; //para datos del jugador3
```

```
27.     public Boolean seguir = true;
28.     int celdaJugador1 = 0; //para llevar el control de las
                               //celdas para el jugador1
29.     int celdaJugador2 = 0; //para llevar el control de las
                               //celdas para el jugador2
30.     int celdaJugador3 = 0; //para llevar el control de las
                               //celdas para el jugador3
31.     int p = 0;
32.     int q = 0;
33.     int m = 0;
34.     clsJugador1 jug1 = new clsJugador1( );
        //se crea instancia de la clase clsJugador1
35.     clsJugador2 jug2 = new clsJugador2( );
36.     //se crea instancia de la clase clsJugador2
37.     clsJugador3 jug3 = new clsJugador3( );
        //se crea instancia de la clase clsJugador3
38.     public FormaHilos( ) {
39.         initComponents( );
40.         cargarTabla( );  //se carga la table con los títulos
41.         //se obtienen los nombres de los jugadores
            //y se cargan en las respectivas etiquetas.
42.         jLabel1.setText("Jugador : " + jug1.getNombre( ).toUpperCase( ));
43.         jLabel2.setText("Jugador : " + jug2.getNombre( ).toUpperCase( ));
44.         jLabel3.setText("Jugador : " + jug3.getNombre( ).toUpperCase( ));
45.     }
46.     //se carga el detalle de las tablas con los valores 0 y -1
47.     void cargarDetalle( ) {
48.         for (int i = 0; i < 24; i++) {
49.             filas[i] = 0;
50.         }
51.         for (int x = 1; x <= 3; x++) {
52.             int numero = (int) (Math.random( ) * 22 + 1);
                //para generar los numeros.
53.             filas[numero] = -1;
54.         }
55.         modeloTabla.addRow(filas);
            //carga el modelo con los datos obtenidos
56.         //carga el modelo en JTable
57.         tabla.setModel(modeloTabla);
58.     }
```

```
59.
60.     void cargarTabla( ) {
a.        for (int i = 1; i <= 24; i++) {
b.            modeloTabla.addColumn(String.valueOf(i));
c.        } //cargar 24  valores (encabezado)
d.        cargarDetalle( );
61.     } //cargar el detalle de la table (0 y -1)
62.     public void limpiarTabla(javax.swing.JTable miTabla) {
        //para limpiar la tabla
63.         DefaultTableModel modelo = (DefaultTableModel) miTabla.getModel( );
64.         //se elimina todas las filas del detalle.
65.         while (modelo.getRowCount( ) > 0) {
66.             modelo.removeRow(0);
67.         }
68.         celdaJugador1 = 0;
69.         celdaJugador2 = 0;
70.         celdaJugador3 = 0;
71.         seguir = true;
72.     }
73.     //se configura la tabla según sus columnas.
74.     void confTablaJugadores(javax.swing.table.DefaultTableModel Modelo)
        {
a.          Modelo.addColumn("Celda Avance");
75.         Modelo.addColumn("Contenido de la Celda");
76.         Modelo.addColumn("Celda Actual");
77.     }
78.     public void moverJugador1(int celda) {
79.         try {
80.             celdaJugador1 += celda;
81.             if (celdaJugador1 >= 24 && seguir == true) {
82.                 celdaJugador1 = 24;
83.                 seguir = false;
84.                 JOptionPane.showMessageDialog(null, "Juego Terminado...
                    Ganó el Jugador No." + String.valueOf(jug1.getNumero( ))+",
                    " + jug1.getNombre( ).toUpperCase( ));
85.             }
86.             if (celdaJugador1 <= 24) {
87.                 //se obtiene el contenido de la celda seleccionada
                    //para jugador1
```

```
88.                 Object valorCelda = tabla.getValueAt(0, celdaJugador1);
89.                 //si en esa celda existe un -1
90.                 if ("-1".equals(valorCelda.toString())) {
91.                 celdaJugador1 = 0;
92.             }
93.             //permite configurar solo una vez la tabla del jugador1
94.             if (p == 0) {
95.                  confTablaJugadores(this.modTablaJugador1);
96.                  p = 1;
97.             }
98.             //se llenan las columnas del detalle de la table del jugador1
99.             datosJugador1[0] = String.valueOf(celda);
100.            datosJugador1[1] = valorCelda;
101.            datosJugador1[2] = celdaJugador1;
102.            modTablaJugador1.addRow(datosJugador1);
103.            tablaJugador1.setModel(modTablaJugador1);
104.            lblIntJugador1.setText("Interacciones: "+ String.valueOf
                (modTablaJugador1.getRowCount()));
105.    }
106.  } catch (Exception ex) {
107.      seguir = false;
108.  }
109. }
110. public void moverJugador2(int celda) {
111.    try {
112.        celdaJugador2 += celda;
113.        if (celdaJugador2 >= 24 && seguir == true) {
114.            celdaJugador2 = 24;
115.            seguir = false;
116.            JOptionPane.showMessageDialog(null, "Juego Terminado... Ganó
                el Jugador No." + String.valueOf(jug2.getNumero( ))+ ", " +
                jug2.getNombre().toUpperCase( ));
117.        } //aqui gana el jugador...
118.        if (celdaJugador2 <= 24) {
119.            Object valorCelda = tabla.getValueAt(0, celdaJugador2);
120.            if ("-1".equals(valorCelda.toString( ))) {
121.                celdaJugador2 = 0;
122.            }
```

```
123.            if (q == 0) {
124.                confTablaJugadores(this.modTablaJugador2);
125.                q = 1;
126.            }
127.            datosJugador2[0] = String.valueOf(celda);
128.            datosJugador2[1] = valorCelda;
129.            datosJugador2[2] = celdaJugador2;
130.            modTablaJugador2.addRow(datosJugador2);
131.            tablaJugador2.setModel(modTablaJugador2);
132.            lblIntJugador2.setText("Interacciones:"+ String.valueOf
                (modTablaJugador2.getRowCount( )));
133.        }
134.    } catch (Exception ex) {
135.        seguir = false;
136.    }
137. }
138. public void moverJugador3(int celda) {
139.    try {
140.        celdaJugador3 += celda;
141.        if (celdaJugador3 >= 24 && seguir == true) {
142.            celdaJugador3 = 24;
143.            seguir = false;
144.            JOptionPane.showMessageDialog(null, "Juego Terminado...Ganó el
                Jugador No." + String.valueOf(jug3.getNumero( ))+ ", " + jug3.
                getNombre( ).toUpperCase( ));
145.        } //aqui gana el jugador...
146.        if (celdaJugador3 <= 24) {
147.            Object valorCelda = tabla.getValueAt(0, celdaJugador3);
148.            if ("-1".equals(valorCelda.toString( ))) {
149.                celdaJugador3 = 0;
150.            }
151.            if (m == 0) {
152.                confTablaJugadores(this.modTablaJugador3);
153.                m = 1;
154.            }
155.            datosJugador3[0] = String.valueOf(celda);
156.            datosJugador3[1] = valorCelda;
157.            datosJugador3[2] = celdaJugador3;
158.            modTablaJugador3.addRow(datosJugador3);
```

```
159.             tablaJugador3.setModel(modTablaJugador3);
160.             lblIntJugador3.setText("Interacciones: " +   String.valueOf
                 (modTablaJugador3.getRowCount( )));
161.         }
162.     } catch (Exception ex) {
163.         seguir = false;
164.     }
165. }
166. private void initComponents( ) {
167. //Este código lo genera netbeans cuando se agregan los controles
168. }// </editor-fold>
169. private void btnGenerarTablaActionPerformed( java.awt.event.ActionEvent
     evt) {
170.     //se cargan los nuevos detalles (nuevos 0 y -1 generados)
171.     modeloTabla.removeRow(0);
         //Elmina toda la fila 0 donde está el encabezado
172.     tabla.setModel(modeloTabla); //se actualiza la table
173.     cargarDetalle( );
174. }
175. private void btnJugarActionPerformed(java.awt.event.ActionEvent evt) {
176.     clsJugador1 j1;  //se crea una instancia de clsJugador1
177.     j1 = new clsJugador1(this);
178.     clsJugador2 j2;
179.     j2 = new Logica.clsJugador2(this);
180.     clsJugador3 j3;
181.     j3 = new Logica.clsJugador3(this);
182.     j1.start( );
183.     j2.start( );
184.     j3.start( );
185.     if (seguir == false) {
186.         j1 = null;
187.         j2 = null;
188.         j3 = null;
189.     }
190.     btnJugar.setEnabled(false);
191.     btnReinicioJuego.setEnabled(true);
192.   }
193.   private void btnReinicioJuegoActionPerformed (java.awt.event.ActionEvent
     evt) {
```

```
194.     limpiarTabla(this.tablaJugador1);
195.     limpiarTabla(this.tablaJugador2);
196.     limpiarTabla(this.tablaJugador3);
197.     btnJugar.setEnabled(true);
198.     btnReinicioJuego.setEnabled(false);
199.     lblIntJugador1.setText("Interacciones: 0");
200.     lblIntJugador2.setText("Interacciones: 0");
201.     lblIntJugador3.setText("Interacciones: 0");
202. }
203. public static void main(String args[ ]) {
204.     //Código generado cuando se crea el JFrame
205. }
206. // Variables declaration - do not modify /Este código se genera al agregar
     controles
207. private javax.swing.JButton btnGenerarTabla;
208. private javax.swing.JButton btnJugar;
209. …
210. } //Fin de la clase FormaHilosw
```

9. Una vez escrito todo el código de la aplicación ya se puede ejecutar. Para ello, se debe pulsar con el botón derecho del mouse sobre el nombre del formulario *FormaHilos* y seleccionar *Ejecutar Archivo*. Se mostrará una pantalla similar a la **figura 2.10**.

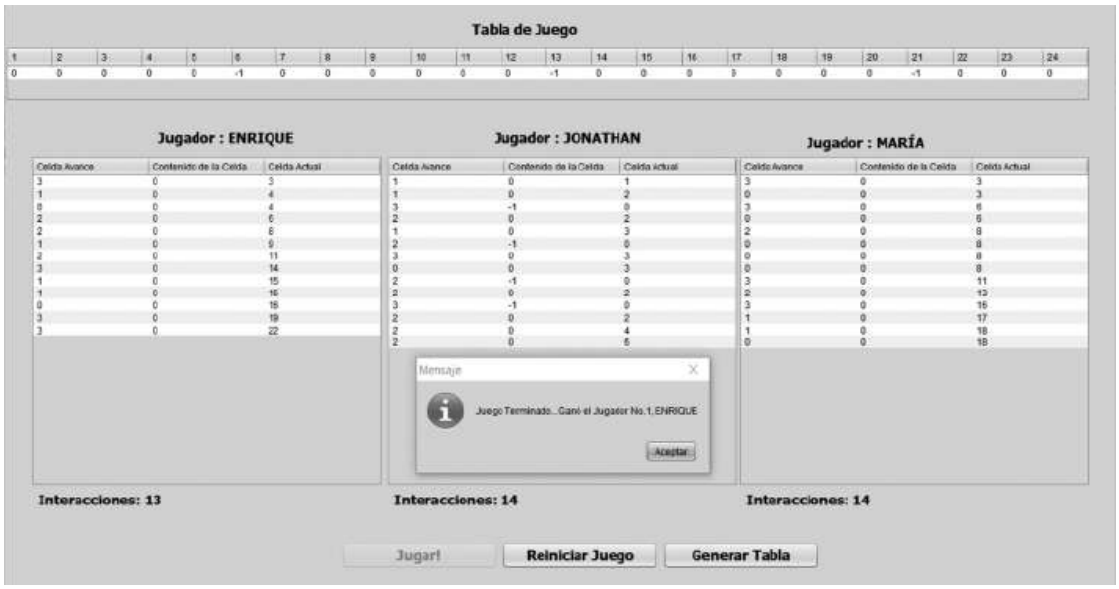


Figura 2.10 Ejecución del formulario FormaHilos

Actividades para el lector

Utilizar NetBeans para desarrollar un programa en Java, el cual 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:

- Jugarán los 4 equipos al mismo tiempo.
- No se podrán enfrentar consigo mismos.
- El equipo que gane 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 diferencial (goles a favor menos goles en contra).

Resumen

En este capítulo se ha desarrollado someramente el tema de arreglos estáticos; dentro de ellos, se exploraron los de una sola dimensión, los cuales poseen un solo índice y son planos. También se estudiaron, mediante la teoría y práctica, los arreglos bidimensionales a los que se les conoce como matrices; este tipo de arreglos tiene dos dimensiones de la forma fila-columna.

El último tipo de arreglos estudiado fue sobre los multidimensionales (tres dimensiones), en el cual se utilizó un ejemplo de generación de múltiples cartones de 5 filas por 5 columnas para un juego de bingo. Estas estructuras estáticas nos son suficientes para resolver todos los problemas enfrentados al programar, por lo que también se desarrolló el tema de las estructuras dinámicas en Java; en este caso se abordaron el ArrayList y el HashMap, los cuales no poseen un tamaño fijo como los anteriores.

En la última sección se estudió la concurrencia de Java, mediante el uso de hilos para cumplir con esta función tan importante en las aplicaciones. En resumen, se trataron los siguientes temas:

- La sintaxis y uso de los arreglos unidimensionales.
- La sintaxis y manejo de las matrices.
- La sintaxis e implementación de los arreglos multidimensionales de tres dimensiones.
- La clase ArrayList, su sintaxis, usos e implementación.
- La clase HashMap, su sintaxis, usos e implementación.
- La concurrencia en Java a través del uso de hilos de ejecución.

Evidencia

a) Realizar una investigación acerca de otras estructuras dinámicas existentes en Java. Enumerarlas e identificar sus principales características, usos y sintaxis.

b) Pensar en problemas de programación que pueden ser resueltos mediante arreglos estáticos.

c) Plantear algunos enunciados de ejercicios que puedan solucionarse con matrices y arreglos multidimensionales.

d) Recordar algún juego en el que se considere el uso de la concurrencia con el manejo de hilos de ejecución.

Preguntas de autoevaluación

1. Explicar la diferencia entre una estructura estática y una dinámica

2. ¿Qué es un vector?

3. ¿Qué es una matriz?

4. ¿Cuántas dimensiones posee una matriz?

5. ¿Para qué es útil la concurrencia cuando se desarrolla software?

6. ¿Cuál es la diferencia entre procesos e hilos?

Respuestas a las preguntas de autoevaluación

1. Una estructura estática posee un tamaño fijo para el almacenamiento de datos en memoria, mientras que una dinámica crece en tiempo de ejecución conforme la necesidad.

2. Un vector es un arreglo de datos que posee una sola dimensión, es finito y tiene un conjunto de elementos del mismo tipo de datos.

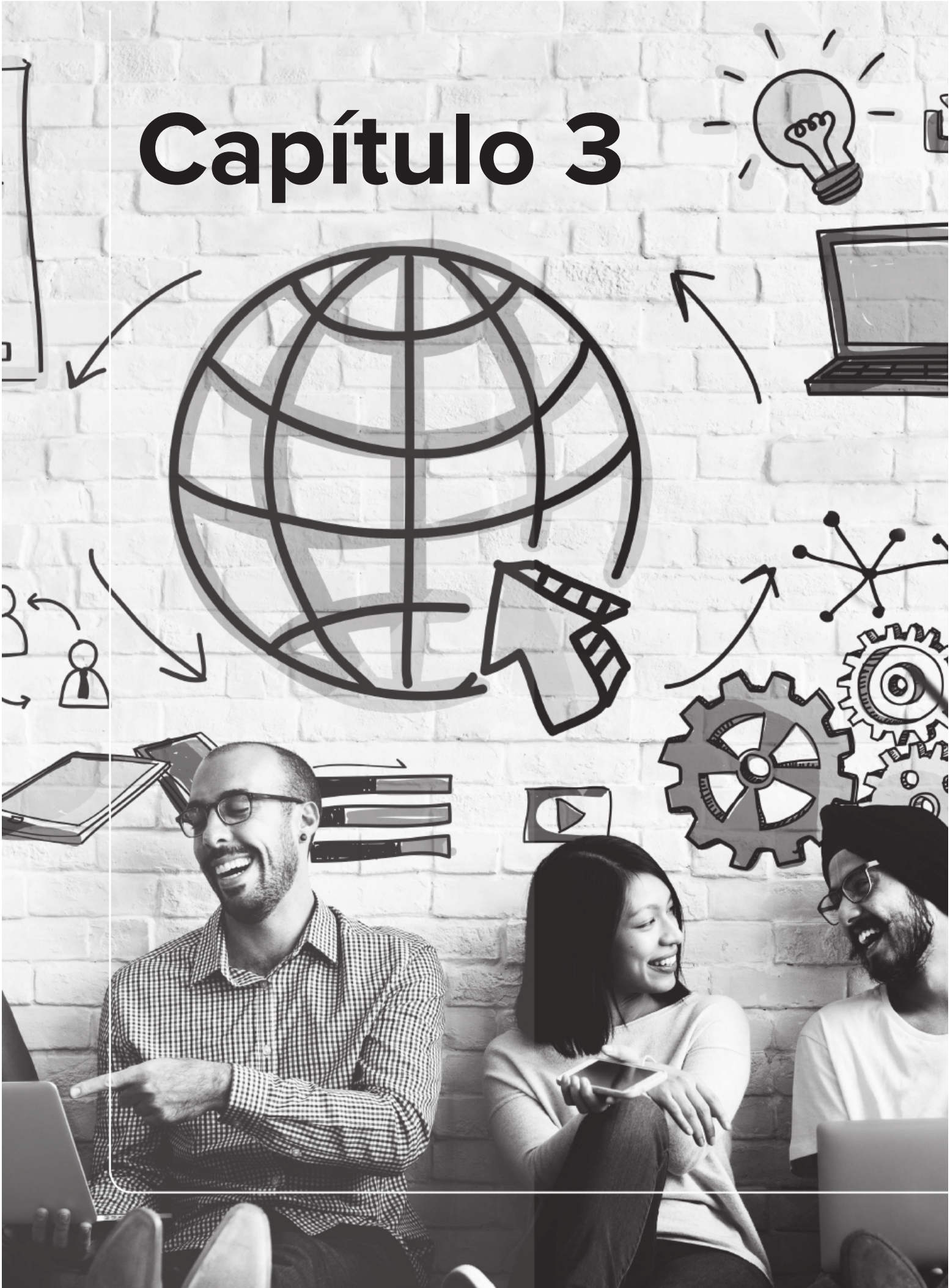
3. Una matriz en un arreglo de datos de dos dimensiones, posee filas y columnas; los elementos que se almacenan son del mismo tipo de datos y se alojan en las celdas que se forman por la intersección de las filas y columnas.

4. Una matriz posee dos dimensiones representadas por filas y columnas.

5. Es útil para realizar múltiples tareas o trabajos desde una misma aplicación de manera simultánea.

6. Cada aplicación que se inicia en una computadora es un proceso, el cual cuenta con recursos propios asignados, por ejemplo el espacio en RAM. Un proceso puede estar compuesto por uno o varios hilos de ejecución; entonces, los hilos forman parte de un proceso y se pueden ejecutar de manera simultánea mientras realizan tareas distintas que la aplicación principal (proceso) necesita, los hilos por tanto no poseen recursos propios sino que comparten los que fueron asignados.

Capítulo 3



Programación orientada a objetos con Java

Reflexione y responda
las siguientes preguntas

¿A qué se le denomina paradigma de programación?

¿Es la programación orientada 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

3.1 Introducción
- **3.1.2** Los paradigmas

3.2 Programación orientada a objetos
- **3.2.1** Introducción a la programación orientada a objetos
- **3.2.2** Conceptos básicos de la programación orientada a objetos

3.3 Interfaces

3.4 Polimorfismo

Expectativa

La **programación orientada a objetos** es un paradigma de programación vigente en nuestros días, éste tiene como meta el desarrollo de software para resolver problemas por medio de colecciones de objetos que se interrelacionan y trabajan conjuntamente.

Mediante este paradigma se pueden diseñar sistemas y aplicaciones informáticas que permiten la especificación funcional y la reutilización de códigos y componentes. Esto implica un alto grado de cohesión y acoplamiento de las aplicaciones a las reglas de negocios. Por ende, conocer este paradigma se convierte en una necesidad para todos los que se dedican al desarrollo de software.

Después de estudiar este capítulo, el lector será capaz de:

- Entender la importancia del paradigma de programación orientada a objetos en el desarrollo de software.
- Conocer los componentes de la programación orientada a objetos para conceptualizarla dentro del marco del desarrollo de aplicaciones informáticas.
- Comprender el paradigma de la programación orientada a objetos y su implementación en Java.
- Desarrollar aplicaciones en Java que utilicen la programación orientada a objetos como herramienta para resolver problemas.
- Aplicar la reutilización de código y componentes en el desarrollo de aplicaciones orientadas a objetos en Java.

3.1 Introducción

En la actualidad, el desarrollo de software es una actividad que genera millones de dólares y se vislumbra un crecimiento aún mayor con la llegada del Internet de las Cosas (IoT), donde millones de dispositivos estarán presentes en Internet y necesitarán software para cumplir sus funciones.

Tales ingresos se obtienen del uso del software y de la creación del mismo. En el primer caso, ya se sabe que la vida comercial y financiera se ve comprometida con el uso intensivo del software para la realización de transacciones o generación de información. En segunda instancia, las empresas desarrolladoras se enfrascan en una lucha feroz por ser parte del mercado de la venta de aplicaciones comerciales, financieras, móviles, juegos, entre otros.

Por lo anterior, es necesario contar con tecnología de punta y las mejores técnicas para el rápido desarrollo de aplicaciones con garantía de calidad y seguridad mediante buenas prácticas. El paradigma de orientación a objetos permite satisfacer muchas de estas necesidades.

3.1.2 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 idea para la resolución de un problema relacionado con el tema.

Paradigma de programación

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

Tipos de paradigmas de programación

En la historia de la computación han existido diversos paradigmas de programación. Actualmente, la orientada a objetos es la que está vigente y la que más compañías de software usan 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 en su funcionalidad.

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

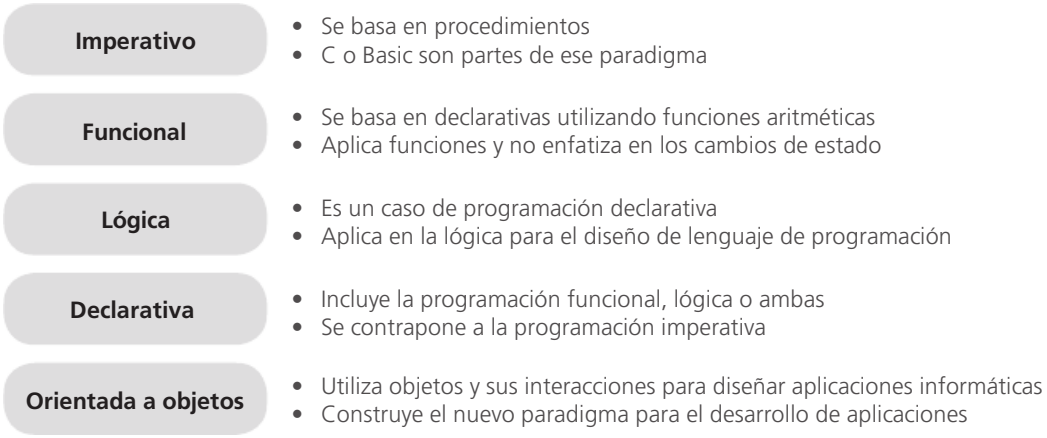


Figura 3.1 Tipos de paradigmas de programación

3.2 Programación orientada a objetos

Como se mencionó, la programación orientada a objetos utiliza a éstos y a las interacciones que se pueden establecer entre ellos. Incluye varias técnicas, entre las que se destacan la abstracción, la herencia, el polimorfismo y el encapsulamiento, mismas que, de utilizarse en forma adecuada y combinada, permiten diseñar y desarrollar sistemas informáticos de calidad respecto de arquitectura y desempeño, además de reutilizar el código implicado.

En la actualidad, existen diversos lenguajes que soportan la programación orientada a objetos, es raro encontrar uno que no lo haga. Muchas suites de desarrollo, además de la herramienta de creación de la aplicación, ofrecen elementos que permiten el diseño conceptual de las mismas. Entre estos utilitarios se encuentran diseñadores de diagramas UML implementados por diseñadores de código e interfaces que facilitan el desarrollo de aplicaciones.

El **Lenguaje Unificado de Modelado** (UML) es un estándar mundial que sirve para modelar (dar forma) software que se desarrolla mediante el paradigma de programación orientada a objetos. Esta poderosa herramienta dota a los desarrolladores de una serie de diagramas con las que pueden visualizar, especificar, diseñar, construir y documentar el software. Entonces, se puede decir que UML y la programación orientada a objetos son el complemento perfecto para crear aplicaciones en la actualidad.

Algunas ventajas de la programación orientada a objetos son las siguientes:

- Se desarrolla más rápido
- Aumento de la calidad del producto final
- El mantenimiento es más sencillo

- Permite la reutilización de software
- Se asemeja a la forma humana de pensar, existe una mayor naturalidad

3.2.1 Introducción a la programación orientada a objetos

La programación orientada a objetos (POO) permite modelar un sistema, identifica los objetos que existen en el mundo acerca del problema que se pretende solucionar, cómo son, cómo se comportan y cómo se relacionan entre sí.

Los **objetos** son las unidades básicas en la solución de problemas al usar POO: son conceptos, abstracciones o cosas discernibles que comparten las mismas características y tienen un significado para el problema tratado. 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 (valores de dichos atributos), mientras que el comportamiento se refiere a 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 (métodos que puede ejecutar un objeto).

La identidad es otra característica presente en los objetos y se orienta a la identificación unívoca del mismo dentro de la colección de objetos del aplicativo. Esta identidad se refiere al nombre del objeto en tiempo de programación.

Finalmente, el tiempo de vida del objeto alude a la duración en la ejecución, es decir, se refiere al lapso de vida general que tiene la aplicación.

La **figura 3.2** muestra las características básicas de un objeto.

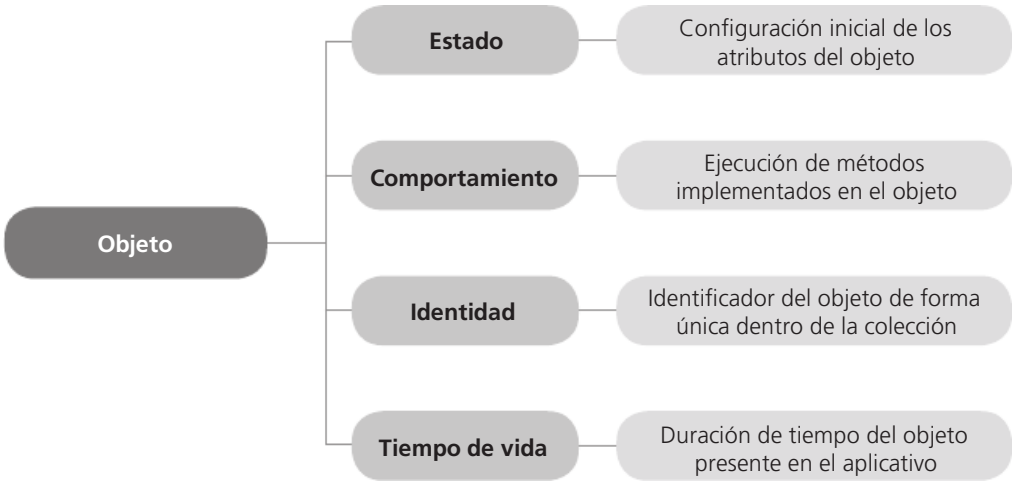


Figura 3.2 Características básicas de un objeto

En la **figura 3.3** se ilustran estas características:

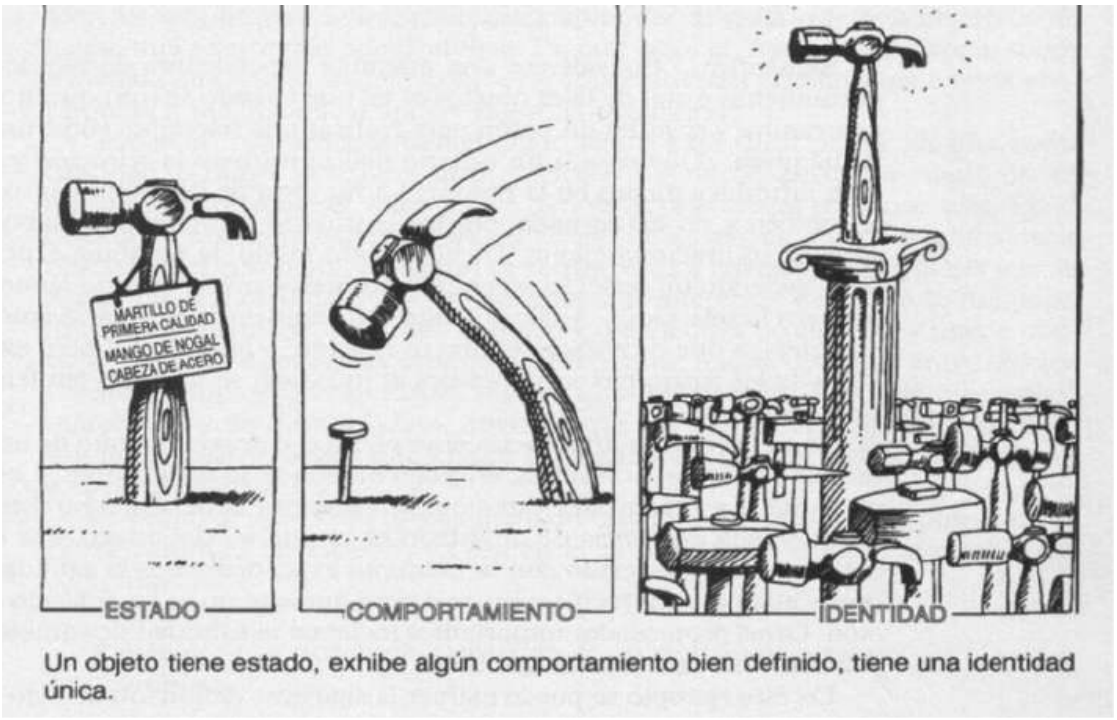


Figura 3.3 Ilustración de las características de un objeto

3.2.2 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, detallados en la **tabla 3.1**.

Tabla 3.1 Conceptos básicos de la programación orientada a objetos

CONCEPTO	DESCRIPCION
Clase	Una clase conjunta elementos que comparten las mismas propiedades y métodos, constituye una plantilla para la creación de objetos. La instanciación de una clase crea lo que se denomina un objeto. Existen superclases y subclases. Con ello se crea lo que se llaman 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. 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 dos o más clases superiores. En Java y .NET sólo existe la herencia simple.

CONCEPTO	DESCRIPCION
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. Es el algoritmo que se implementa para realizar una determinada operación sobre los atributos de la clase o de una clase derivada. Se puede decir que un método es una función o procedimiento que realiza alguna función para el objeto.
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 establecida entre objetos para intercambiar parámetros ya 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.

Es preciso agregar cuáles son las propiedades principales de la programación orientada a objetos; se muestran en la **tabla 3.2**.

Tabla 3.2 Propiedades principales de la programación orientada a objetos

CONCEPTO	DESCRIPCION
Abstracción	Representa las características esenciales de un objeto, sin tener la preocupación por las demás características que no lo son. Se interesa 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. Un ejemplo de abstracción es un motor, el cual puede ser muy genérico y comportarse de tal forma cuando es implementado por un motor a explosión, eléctrico o a vapor.
Encapsulamiento	Es un mecanismo que organiza datos y métodos, evita el acceso a estos datos por cualquier otro medio diferente a los especificados. 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).

CONCEPTO	DESCRIPCION
Ocultamiento de información	Es generalmente el sistema de aislamiento de las propiedades (atributos) de un objeto, usado 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 de manera directa. Se aplica de acuerdo con la Ley de Demeter. Tiene que ver con los niveles de acceso: público, protegido y privado.
Polimorfismo	Se refiere a la capacidad de los objetos de comportarse de acuerdo con la funcionalidad requerida. Es decir, establece diferentes comportamientos para los métodos del objeto, de manera que implementan diversas funcionalidades en relación 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.

Creación de una clase en Java

Una clase describe cómo se estructuran internamente los objetos (atributos) y cuáles servicios proveen (métodos). Para crear una clase en Java se tomará en cuenta la siguiente estructura genérica, la cual está distribuida en 5 pasos:

```
// Paso 1. Se crea la clase.
public class NombreClase {
   // Paso 2. Se declaran los atributos de la clase.
   private tipoDato nomAtributo;
   // Paso 3. Declaración del o los constructores de la clase.
   public NombreClase( ) { //El constructor se llama igual que la clase
        //Se asigna valores iniciales a los atributos.
   }
   // Paso 4. Declaración de los métodos set( ) y get( ) de los atributos.
   public tipoRetorno getNomAtributo( ) {
        return nomAtributo;
   }
   public void setNomAtributo(tipoDato nomAtributo) {
     this.nomAtributo = nomAtributo;
   }
   // Paso 5. Se declaran los métodos de la clase, los cuales definen el
   //comportamiento de los objetos de la clase.
} //Fin de la clase
```

Tomando en cuenta la estructura anterior, se definirán algunos conceptos importantes que se deben conocer al crear una clase.

Modificadores de acceso

Determinan la visibilidad y acceso que puede tener un atributo o un método desde otras clases en las que se cree una instancia de ellas.

- **private:** El atributo o método es accesible sólo desde la misma clase en la que se declare
- **public:** El atributo o método es accesible desde cualquier clase, sin restricciones
- **protected:** El atributo o método es accesible desde la clase actual, las clases que deriven de ella y las que estén dentro del mismo paquete

Declaración de atributos

Para garantizar el ocultamiento de la información, el cual es una característica importante en la POO, se recomienda que los atributos de las clases sean privados. Los atributos sólo se declaran, sin inicializarlos.

Método constructor y destructor

El **constructor** es un método que se llama igual que la clase y permite la construcción del objeto de la misma, tiene la estructura public `NombreClase( ){ }`. Este método se invoca cuando se crea un objeto de dicha clase, permite el uso de parámetros que comúnmente se utilizan para dar valores iniciales a los atributos del objeto. Puede que existan varios constructores. Otras características son: se declaran como `public` y no devuelve ningún valor debido a su naturaleza particular. En Java no se declaran los métodos **destructores** debido a la existencia del **Garbage Collector** (este concepto se usa para describir la administración automática de memoria en Java), el cual se encarga de eliminar de la memoria todos aquellos objetos que ya no se necesiten.

Sobrecarga de métodos

La **sobrecarga de métodos** es un tipo de polimorfismo que permite que existan varios métodos con el mismo nombre pero con diferente firma, ésta se refiere a los parámetros del método, en los cuales se pueden usar diferentes combinaciones de tipos de datos y el orden en que se declara; la creación de uno o varios constructores obedece a la necesidad del programador.

En el o los constructores se suele dotar a los atributos de valores iniciales que podrán ser cambiados por medio de los métodos `set( )` y que se podrán consultar a través de los métodos `get( )`.

Método set y get

Estos métodos permiten modificar y recuperar el valor de los atributos (que fueron encapsulados) desde afuera de la clase en la que están declarados. Como se puede observar, en la estructura básica de una clase el método `set( )` permite cambiar el valor de atributo para un objeto específico; este método no retorna ningún valor, pero sí recibe por parámetro el nuevo valor dirigido al atributo para el que se creó el set, y se nombra de la forma `setNomAtributo`.

Por otro lado, el método `get( )` permite obtener el valor de un determinado atributo de un objeto; no recibe parámetros pero sí devuelve el valor actual del atributo que se define en su nombre. El nombre por convección tiene la siguiente forma: getNomAtributo.

La única manera para trabajar con los atributos de un objeto es por medio de los métodos `set( )` y `get( )`, esto debido a la encapsulación y el ocultamiento de la información.

Métodos de clase

En esta sección de la estructura básica de una clase se pueden crear todos los métodos necesarios que le permitan a un objeto tener diferentes comportamientos, esto a través de una gama de operaciones.

Se pondrá como ejemplo la creación de la clase Empleado para desarrollar su estructura básica.

Listado No. 3.1 Estructura básica para la creación de la clase Empleado

```
1.  //Paso 1. Se crea la clase.
2.  public class Empleado {
3.      //Paso 2. Se declaran los atributos de la clase.
4.      private int idEmpleado;
5.      private String nombre;
6.      private String genero;
7.      private String puesto;
8.       //Se pueden declarar todos los atributos necesarios
9.
10.     //Paso 3. Declaración del o los constructores de la clase.
11.     public Empleado(int idEmpleado, String nombre, String genero, String puesto) {
12.         //Se inicializan atributos con los valores de los parámetros
13.         this.idEmpleado = idEmpleado;
```

```
14.          this.nombre = nombre;
15.          this.genero = genero;
16.          this.puesto = puesto;
17.  }
18.      public Empleado( ) {    //Sobrecarga del constructor
19.          //Se inicializan atributos con los valores iniciales //de ceros o comillas vacía para los strings.
20.          this.idEmpleado = 0;
21.          this.nombre = “”;
22.          this.genero = “”;
23.          this.puesto = “”;
24.        }
25.      //Paso 4. Declaración de los métodos set( ) y get( ) de los atributos.
26.      public int getIdEmpleado( ) {
27.          return idEmpleado;
28.      }
29.      public void setIdEmpleado(int idEmpleado) {
30.          this.idEmpleado = idEmpleado;
31.      }
32.      public String getNombre( ) {
33.          return nombre;
34.      }
35.      public void setNombre(String nombre) {
36.          this.nombre = nombre;
37.      }
38.      public String getGenero( ) {
39.          return genero;
40.      }
41.      public void setGenero(String genero) {
42.          this.genero = genero;
43.      }
44.      public String getPuesto( ) {
45.          return puesto;
```

```
46.     }
47. public void setPuesto(String puesto) {
48.     this.puesto = puesto;
49.     }
50. //Paso 5. Se declaran los métodos de la clase.
51. public void imprimirEmpleado( ){
52.         System.out.println("-----------------------------------------
    ----------");
53.         System.out.println("         DATOS DEL EMPLEADO");
54.         System.out.println("-------------------------------------------------
    --------");
55.         System.out.println("Identificación: "+ this.idEmpleado);
56.         System.out.println("Nombre: "+ this.nombre);
57.         System.out.println("Género: "+ this.genero);
58.         System.out.println("Puesto: "+ this.puesto);
59.         System.out.println("-----------------------------------------------
    ---------");
60.     }
61. }
```

Como se puede ver en el listado anterior, se toma como base la estructura genérica para crear una clase en Java, descrita anteriormente, que a su vez está basada en los 5 pasos propuestos como documentación en el código fuente del ejemplo.

Paso 1. Se crea la clase

Paso 2. Se declaran los atributos de la clase

Paso 3. Declaración del o los constructores de la clase

Paso 4. Declaración de los métodos `set( )` y `get( )` de los atributos

Paso 5. Se declaran los métodos de la clase

Herencia y tipos de clases

La herencia permite crear una clase nueva a partir de otra ya existente. Una clase padre puede heredar atributos y métodos a sus clases hijas. Java sólo permite la herencia simple, o sea, una

clase sólo puede tener un padre o sólo puede heredar de una sola clase. Con la herencia se logra la reutilización del código fuente.

Existen tres tipos de clases: **abstracta**, **base** y **derivada**. A continuación se describen.

Clases abstractas

Una clase abstracta implementa la estructura genérica de la funcionalidad que deberá ser aplicada en una clase derivada. En realidad, una clase abstracta 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 se crea la instanciación.

Las clases abstractas se suelen crear o definir cuándo se necesita englobar objetos de tipos diferentes y se quiere implementar polimorfismo. Supóngase que se requiere crear un sistema que sea lo suficientemente genérico (parametrizable), pero que establezca los métodos que obligatoriamente deben realizarse y los atributos básicos requeridos. Entonces, entran en juego las clases abstractas. La reutilización de código en este contexto es claramente demostrable, pues es usado como clase base.

Clases base

También se denomina superclase y se refiere a aquélla de la cual heredaran otras clases; a diferencia de la abstracta, este tipo de clase sí permite instancias (crear objetos de sí misma). En el listado 3.1 se muestra un ejemplo.

Clases derivadas

También llamada subclase, una **clase derivada** es aquella que hereda atributos y métodos de otra clase base o abstracta.

A continuación se muestra un ejemplo de implementación de clases abstractas y derivadas representado por los listados 3.2, 3.3, 3.4 y 3.5, y por las figuras 3.4 y 3.5.

1. Inicie con la apertura de NetBeans
2. Cree un nuevo proyecto Java, como se mostró en las figuras 1.5, 1.6 y 1.7 del capítulo primero
3. El nombre del proyecto será **ClaseAbstracta**
4. Una vez que se ha creado el proyecto, pulse con el botón derecho sobre el nombre del mismo (ClaseAbstracta) y seleccione **Nuevo** y **Clase Java**. Nombre esta clase como `Vehiculo` y escriba el código siguiente:

Listado No. 3.2 Creación de la clase abstracta `Vehiculo`

```
1.  import java.util.Scanner;
2.  public abstract class Vehiculo {
3.      protected int modelo;
4.      protected String color;
5.      protected String marca;
6.      protected double precio;
7.      Scanner Lector = new Scanner(System.in);          //para procesar lecturas de
8.                                          //teclado en clases derivadas
9.      public Vehiculo(int modelo, String color, String marca, double precio) {
10.             this.modelo = modelo;   //Este es un constructor de la clase
11.             this.color = color;
12.             this.marca = marca;
13.             this.precio = precio;
14. }
15.     public Vehiculo( ) {  //declara un constructor vacío.
16.             this.modelo = 0;
17.             this.color = "";
18.             this.marca = "";
19.             this.precio = 0;
20.     }
21.     public abstract void registrarVehiculo( ); //Declaración de un método abstracto
        el cual debe ser
22.     // implementado en la clase derivada.
23. }
```

Acerca del código del listado 3.2:

Primero, es preciso aclarar algunos conceptos:

- Como se observa en la línea 2, se ha declarado la clase pública Vehiculo y antepuesto la sentencia abstract con lo cual se crea una clase abstracta.

- En las líneas 3, 4, 5 y 6 se declaran los atributos modelo, color, marca y precio de tipo protected. Estos atributos serán accesibles desde la clase donde se definen, así como en las clases que se derivan de ella (subclases). Esto para que sus atributos sean directamente accesibles desde las clases derivadas.

- Observe la línea 9, esta línea es uno de los constructores de la clase (porque también hay otro en la línea 15; la clase se llama igual: Vehiculo). El constructor de la clase recibe los parámetros modelo, color, marca y precio; sin embargo, al observar la línea 10 se notará 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 se trata de una referencia al objeto que ejecuta 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.

- En la línea 21 se observa la declaración de un método abstracto denominado `registrarVehiculo( )`, pero 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 se crea una nueva clase que será derivada (heredará) de la anterior, en esta nueva clase será donde se implemente el código del método abstracto; se reutilizarán los atributos protegidos de la clase abstracta. Se debe proceder como en el punto 4: crear una nueva clase de nombre Automovil.

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

```
public class Automovil {
por:
public class Automovil extends Vehiculo {
```

La primera línea crea una clase normal denominada Automovil. La segunda crea la clase Automovil y le indica que herede (mediante la palabra reservada extends) de la clase abstracta Vehiculo. Una vez que se escriba lo anterior, NetBeans se dará cuenta que se está implementando una clase abstracta, por ende solicitará que se incluyan y sobrescriban todos los métodos abstractos de dicha clase. La **figura 3.4** muestra esta situación.

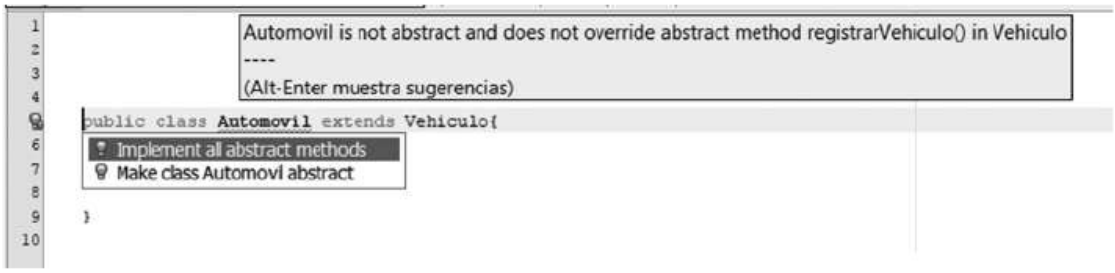


Figura 3.4 Implementación de los métodos abstractos de la clase abstracta

6. Se debe pulsar sobre el ícono de la lámpara en la opción Implementar todos los métodos abstractos para que dichos métodos de la clase aparezcan en el código de la nueva.

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

Listado No. 3.3 Creación de la clase Automovil que reescribe el método abstracto

```
1.  public class Automovil extends Vehiculo {
2.      //Se declaran atributos propios
3.      private float cilindraje;
4.      //Contructores de la clase
5.      public Automovil(float cilindraje, int modelo, String color, String
        marca, double precio) {
6.          super(modelo, color, marca, precio); //Llamada al constructor del padre
7.          this.cilindraje = cilindraje;
8.      }
9.      public Automovil( ) {
10.         super( );
11.         this.cilindraje = 0;
12.     }
13.     //Métodos set y get de los atributos
14.     public float getCilindraje( ) {
15.         return cilindraje;
16.     }
17.     public void setCilindraje(float cilindraje) {
18.         this.cilindraje = cilindraje;
19.     }
20.
21.     //Métodos de clase
22.     @Override  //Anotación de sobreescritura
23.     public void registrarVehiculo( ) {
24.         System.out.print("Modelo del automóvil: ");
25.         modelo = Lector.nextInt( );
26.         System.out.print("Color del automóvil: ");
```

```
27.          color = Lector.next( );
28.          System.out.print("Marca del automóvil: ");
29.          marca = Lector.next( );
30.          System.out.print("Precio del automóvil: ");
31.          precio = Lector.nextInt( );
32.          System.out.print("Cilindraje del automóvil: ");
33.          cilindraje = Lector.nextFloat( );
34.     }
35. }
```

En la líneas 6 y 10 se hace la llamada al método construtor del padre (clase Vehiculo), a través del método super( ), en éstas líneas de código se puede ver la implementación de la herencia.

7. Ahora se crea otra clase denominada Bicicleta con el siguiente código:

Listado No. 3.4 Clase Bicicleta que implementa la funcionalidad de la clase abstracta

```
1.  public class Bicicleta extends Vehiculo {
2.       //Se declaran atributos propios
3.       private float numeroRueda;
4.       //Contructores de la clase
5.       public Bicicleta(float numeroRueda, int modelo, String color, String
         marca, double precio) {
6.            super(modelo, color, marca, precio);
7.            this.numeroRueda = numeroRueda;
8.       }
9.       public Bicicleta( ) {
10.           super( ); //Llamada al constructor vacío del padre
11.           this.numeroRueda = 0;
12.      }
13.      //Métodos set y get de los atributos
14.      public float getNumeroRueda( ) {
15.           return numeroRueda;
16.      }
17.      public void setNumeroRueda(float numeroRueda) {
18.           this.numeroRueda = numeroRueda;
19.      }
```

```
20.      //Métodos de clase
21.      @Override  //Anotación de sobreescritura
22.      public void registrarVehiculo( ) {
23.           System.out.print("Modelo de la bicicleta: ");
24.           modelo = Lector.nextInt( );
25.           System.out.print("Color de la bicicleta: ");
26.           color = Lector.next( );
27.           vSystem.out.print("Marca de la bicicleta: ");
28.           marca = Lector.next( );
29.           System.out.print("Precio de la bicicleta: ");
30.           precio = Lector.nextInt( );
31.           System.out.print("Número de rueda de la bicicleta: ");
32.           numeroRueda = Lector.nextFloat( );
33.    }
34. }
```

8. Sobre este código se puede agregar que en la línea 21 existe la palabra `@Override`, una anotación que significa la sobrescritura de un método de la clase padre (es decir, se está sobrescribiendo el método `registrarVehiculo( )` en `Bicicleta` que es implementada desde la clase padre `Vehiculo`). Considérese otro ejemplo similar.

Supóngase que se necesita escribir una clase llama Comida que contenga una operación llamada cocinarAlimento(). Sin embargo, también se sabe que con dicha operación se desea procesar cualquier tipo de alimento. Atributos genéricos podrían ser `tiempoCoccion` y temperaturaCoccion. En un determinado caso, podría ser que se cree una clase llamada ComidaChina que requiere preparar cierto tipo de alimento, o la clase ComidaCasera que es otro tipo. Ambas necesitan una operación en común: `cocinarAlimento( ).` Esta operación requiere los atributos `tiempoCoccion` y `temperaturaCoccion`. En tal caso, la clase Comida se podría declarar como abstracta con la operación de tipo abstracto (vacío) `cocinarAlimento`(), y las clases `ComidaChina` y `ComidaCasera` heredaran de ella, e implementarán el método `cocinarAlimento`() según sus necesidades.

Actividades para el lector

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

9. Ahora se implementará la clase PruebaClase que tiene el método main(), el cual que permite ejecutar los programas del proyecto.

Listado No. 3.5 Creación de la clase principal PruebaClase

```
1.  public class PruebaClase {
2.      public static void main(String[ ] args) {
3.              Bicicleta bicicleta = new Bicicleta( );
4.              bicicleta.registrarVehiculo( );
5.              Automovil automovil = new Automovil( );
6.              automovil.registrarVehiculo( );
7.      }
8.  }
```

Comentarios sobre el código del listado 3.5:

- Para crear un objeto en Java (instancia de una clase no abstracta) se puede seguir la fórmula: `NombreClase nombreObjeto = new nombreConstructor( );`

- Dos ejemplos de la fórmula anterior se observan en las líneas 3 y 5 dentro del main, en ellas se instancian las clases `Bicicleta` y `Automovil`, mediante la creación de los objetos `bicicleta` y `automóvil`, respectivamente. Hay que recordar que ambas clases implementan el método abstracto `registrarVehiculo( )` que se creó en la clase abstracta `Vehiculo`.

- Para acceder a las funcionalidades u operaciones (métodos) que posee una clase, se hace por medio del nombre del objeto, de la forma `nombreObjeto.operacion(valoresParaParametros)`. Los ejemplos para este caso se pueden observar en las líneas 4 y 6.

10. En la **figura 3.5** se muestra la ejecución típica de este ejemplo.

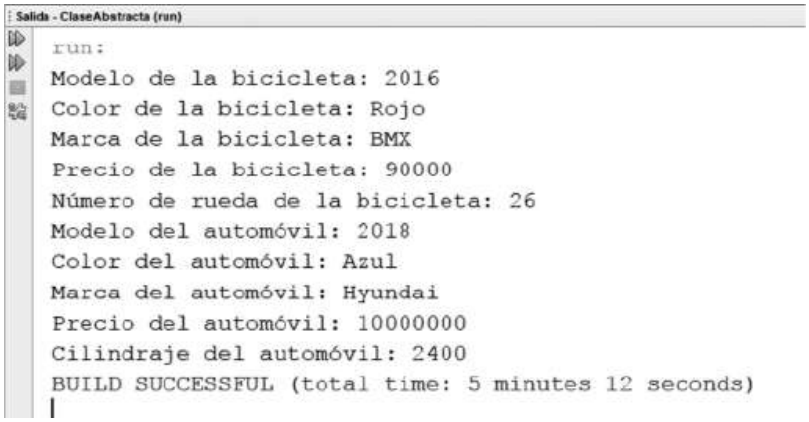


Figura 3.5 Ejecución del ejemplo de ClaseAbstracta

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, se debe considerar que la mayoría de lenguajes de programación (si no es que todos) implementan la herencia simple solamente.

La instrucción escrita en el listado 3.3 en la línea 1 es:

```
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` tanto los atributos cómo los métodos. En este sentido, se establecen las siguientes reglas (en el caso de que Vehiculo no fuera una clase abstracta):

- `Automovil` hereda de `Vehiculo` todos sus métodos u operaciones 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 ellos, sino que lo hace a través del método público heredado, en este caso los métodos `set( )` y `get( )`. En el ejemplo anterior se declararon protegidos para que las clases derivadas tuvieran acceso directo.

- La instanciación se realiza sobre la clase `Automovil` para que herede de `Vehiculo`. Cabe recordar que no se puede utilizar directamente una clase en un programa, es necesario crear un objeto antes.

Actividades para el lector

Investigar y documentar para qué se utiliza la instrucción super

3.3 Interfaces

Una **interfaz** es una colección de atributos constantes y métodos abstractos; es como una plantilla genérica, ya que se especifica qué se debe hacer pero no se detalla cómo, es decir, su implementación.

Mediante el uso de las interfaces se puede cubrir la necesidad de que una clase herede de otras (dos o más). 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 o más superclases. Se puede concluir que una clase puede implementar dos o más interfaces, o bien, como se podría decir en el lenguaje C++, que una clase puede heredar de dos o más de ellas.

Se debe considerar el diagrama de clases de la **figura 3.6**.

Figura 3.6 Diagrama de clases simulando herencia múltiple con interfaces

Este diagrama demuestra la necesidad de asociar la clase `Empleado` con las interfaces `iDepartamento` e `iOperadoraPension`. 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 de manera genérica y, posteriormente, la clase `Empleado` introduce las interfaces.

1. Abrir NetBeans y desarrollar un nuevo proyecto llamado "Usando interfaces", como se muestra en las figuras 1.5, 1.6 y 1.7 del capítulo primero, basado en el diagrama de clases de la figura anterior.

2. Una vez que se ha creado el proyecto, en lugar de crear una clase, seleccionar una interfaz, pulsar con el botón derecho sobre el nombre del proyecto y seleccionar **Nuevo** y **Java Interface**... Nombrar esta interfaz como `iDepartamentos` y escribir el código que se muestra en el listado 3.6.

Listado No. 3.6 Creación de la interfaz iDepartamentos

```
1.  public interface iDepartamentos {
2.      //se crea un vector de cadena y se inicializa con sus respectivos valores
3.      public static final String DPTOS[ ] = {“Recursos Humanos”,”Informática”,
        “Financiero”, “Contabilidad”, “Ventas”};
4.      //se inicializa un vector de enteros con sus respectivos valores
5.      public static final int EXTS[ ] = {123, 345, 324, 125, 423};
6.      //se crear un método vacío (sin implementar)
7.      void asignarDepto( );
8.  }
```

Comentarios sobre el código del listado 3.6:

- En la línea 1 se crea la interfaz Java llamada iDepartamentos.
- En las líneas 3 y 5 se declaran atributos vectores constantes, como lo indica la definición de interfaces antes vista.
- La línea 7 muestra la declaración del método `asignarDepto( )`, el cual deberá ser implementado en cualquier clase que implemente la interfaz iDepartamentos.

3. Luego, se debe crear la interfaz iOperadoraPension; se procede de la misma manera que en el punto anterior. El código de esta interfaz se muestra en el listado 3.7:

Listado No. 3.7 Creación de la interfaz `iOperadoraPension`

```
1.  public interface iOperadoraPension {
2.      //se crea un vector de cadena con sus respectivos valores
3.      public static final String ENTIDADES[ ] = {“Banco de Pensiones”,”Pensiones
        Costa Rica”, “Banco Municipal”,
        “Sistema de Pensiones”, “Pensiones Acme”};
4.      //se crea un vector de tipo double con sus respectivos valores
5.      public static final double TASA_RINDE[ ] = {0.12, 0.13, 0.12, 0.12, 0.12};
6.      void asignarOperadora( );
7.  }
```

4. Ahora se crea la clase Empleado y se implementan estas interfaces en la misma. Para ello, se debe escribir el siguiente código del listado 3.8:

Listado No. 3.8 Creación de la clase Empleado e implementación de las interfaces

```
1.  import java.util.Scanner;
2.  //La clase Empleado implementa las interfaces antes creadas.
3.  public class Empleado implements iDepartamentos, iOperadoraPension {
4.      //se crea una variable tipo Scanner para realizar lecturas desde teclado
5.      Scanner Lector = new Scanner(System.in);
6.
7.      @Override //para implementar el método asignarDepto
8.      public void asignarDepto( ) {
9.          int dpto, ext;
10.         String nombre;
11.         System.out.print("Escriba Nombre del empleado: ");
12.         nombre = Lector.next( );  //se lee de teclado el contenido para nombre
13.         for (int i = 0; i < DPTOS.length; i++){ //se recorre el vector DPTOS.
14.             System.out.println(i + 1 + " " + DPTOS[i]);    //se muestran valores del vector
15.         }
16.         System.out.print("Seleccione Departamento : ");
17.         dpto = Lector.nextInt( ); //lee por teclado una variable de tipo entera
18.         System.out.println("Departamento seleccionado : " + DPTOS[dpto - 1]);
19.         for (int i = 0; i < EXTS.length; i++) {//se recorre el vector de
    extensiones
20.             System.out.println(i + 1 + " " + EXTS[i]); //despliega cada va-
    lor del vector
21.         }
22.         System.out.print("Seleccione la Extensión : ");
23.         ext = Lector.nextInt( );
24.         System.out.println("Extensión : " + EXTS[ext - 1]);
25. }
26.
27.     @Override
28.     public void asignarOperadora( ) {
29.         int opera, tasa;
```

```
30.          for (int i = 0; i < ENTIDADES.length - 1; i++) {
31.             System.out.println(i + 1 + " " + ENTIDADES[i]);
32.          }
33.          System.out.print("Seleccione la Operadora : ");
34.          opera = Lector.nextInt( );
35.          System.out.println("Departamento seleccionado : " +
             ENTIDADES[opera - 1]);
36.          for (int i = 0; i < TASA_RINDE.length - 1; i++) {
37.             System.out.println(i + 1 + " " + TASA_RINDE[i]);
38.          }
39.          System.out.print("Seleccione la tasa de rendimiento: ");
40.          tasa = Lector.nextInt( );
41.          System.out.println("Extensión : " + TASA_RINDE[tasa - 1]);
42.      }
43. }
```

Obsérvese cómo se implementan las interfaces iDepartamentos e iOperadoraPension en la línea 3 de la clase Empleado. También se implementan los métodos `asignarDepto( )` y `asignarOperadora( )` de las interfaces.

5. Finalmente, se prueba el uso de la clase Empleado que implementa las dos interfaces por medio de la creación de una instancia de dicha clase. El código en el programa principal (que contiene el main) se muestra en el listado 3.9:

Listado No. 3.9 Programación del main para probar la clase Empleado

```
1.  public class PruebaInterfaces {
2.      public static void main(String[ ] args) {
3.        Empleado aux = new Empleado( );
4.        aux.asignarDepto( );
5.        aux.asignarOperadora( );
6.      }
7.  }
```

6. Una corrida de la aplicación se muestra en la **figura 3.7**.

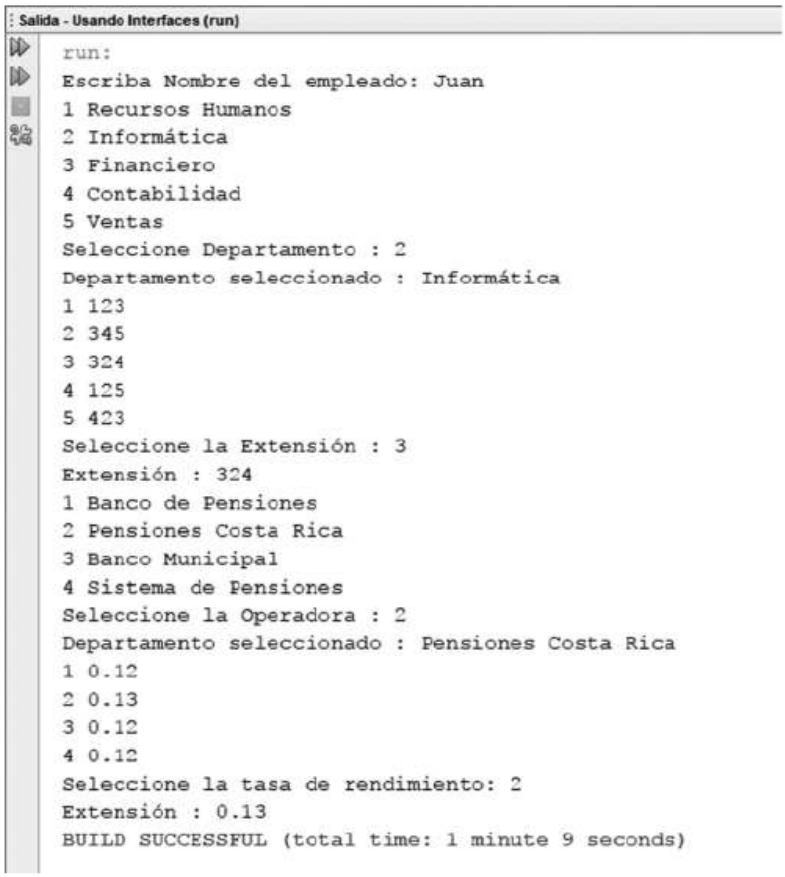


Figura 3.7 Corrida típica de aplicación de interfaces

Actividades para el lector

Desarrolle un programa Java utilizando NetBeans que implemente las interfaces necesarias para el caso de Medico, Empleado, Persona.

3.4 Polimorfismo

El **polimorfismo** (llamado también **upcasting**) brinda la posibilidad 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 de modo diferente.

Por ejemplo, considérese 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 llamado 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 incluye especificación alguna sobre el tipo de datos en el que opera; por ende, se puede utilizar en cualquier dato que sea compatible. Al polimorfismo dinámico se le conoce también como programación genérica.

El **polimorfismo estático** (también denominado ad hoc) es aquél donde los tipos de datos deben ser explícitos y declarados en forma individual antes de ser utilizados.

Supóngase que se tiene una clase que implementa tres métodos con el mismo nombre pero que éstos reciben parámetros de distinto tipo y devuelven resultados también diferentes. Se crea el siguiente ejemplo:

1. Crear un nuevo proyecto llamado "Usando polimorfismo", agregar una nueva clase (Java Class) al proyecto que se acaba de crear y denominarla `clsPolimorfismo`. El código se muestra en el listado 3.10:

Listado No. 3.10 Programación de la clase `ClsPolimorfismo`

```
1.  public class ClsPolimorfismo {
2.
3.  int sumar(int a, int b) {
4.      return a + b;
5.    }//retorna un valor de tipo entero
6.
7.  double sumar(double a, double b) {
8.      return a + b;
9.    } //retorna un valor de tipo doublé
10.
11.   String sumar(String a, String b) {
12.     return a + b;
13.   } //retorna  un valor de tipo cadena
14. }
```

Se debe observar que hay tres métodos llamados igual (sumar), con la diferencia de los tipos de datos que se reciben como parámetro y los de retorno del método.

La llamada a estas funciones dependerá del tipo de dato que envía aquélla (quien la llama o invoca), es decir, si se envían datos de tipo cadena se ejecutará la función que reciben esos parámetros o, si por el contrario, se hace la llamada y se envían valores enteros, se invocará el método que recibe parámetros de tipo entero.

2. Crear una clase Java Principal (Java main class) llamada `PruebaPolimorfismo`, en ésta se ejecutará la funcionalidad que se programó en la clase anterior, el listado 3.11 muestra el código de la misma:

Listado No. 3.11 Programación de la clase `PruebaPolimorfismo`

```
1.  public class PruebaPolimorfismo {
2.     public static void main(String[ ] args) {
3.     ClsPolimorfismo aux = new ClsPolimorfismo( );
4.     System.out.println("Suma de Enteros: " + aux.sumar(2, 3));
5.     System.out.println("Suma de Cadenas: " + aux.sumar("Concatena", " Palabras"));
6.     System.out.println("Suma de Doubles: " + aux.sumar(2.5, 3.3));
7.    }
8.  }
```

La **figura 3.8** muestra la salida de la ejecución de este programa.

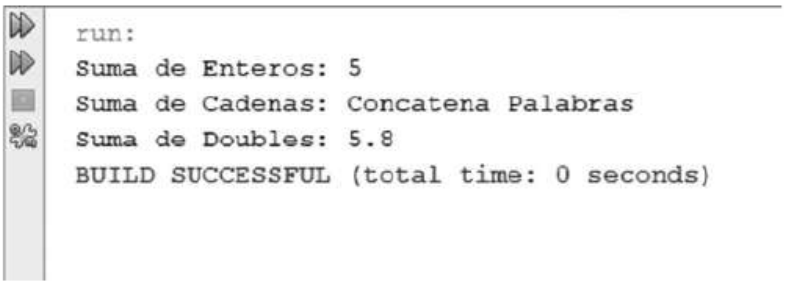


Figura 3.8 Ejecución del ejemplo de polimorfismo

Actividades para el lector

Desarrollar un programa Java que por medio de NetBeans abstraiga el pago del salario de un empleado en una empresa A, la cual brinda servicios de salud, y el pago de un empleado de una empresa B dedicada a brindar educación preescolar, tomando en cuenta que el cálculo de salarios de cada compañía es distinto y considera diferentes factores; por ejemplo, en la empresa de salud, el pago de disponibilidad a médicos especialistas.

En conclusión, en el presente capítulo se describieron los elementos básicos de la programación orientada a objetos mediante el lenguaje base de programación Java, aunque en su mayoría aplican para cualquier lenguaje orientado a objetos. Existe mucha bibliografía sobre el tema; para profundizar, se recomienda consultar la que se sugiere al final de esta obra, así como manuales de uso libre.

Resumen

En este capítulo se han desarrollado los fundamentos de la programación orientada a objetos en Java, los cuales constituyen la base para el desarrollo de aplicaciones con esta orientación. Básicamente, se trataron los siguientes temas:

- Definición de los paradigmas existentes en nuestro medio
- Resumen de los paradigmas existentes en la programación de sistemas computacionales
- Síntesis de los tipos de programación existentes
- Fundamentos de los conceptos básicos de la programación orientada a objetos
- Desarrollo de ejemplos prácticos y sencillos sobre el formato de una clase genérica en 5 pasos, clases abstractas, interfaces y polimorfismo

Preguntas de autoevaluación

1. Definir paradigma

2. ¿Qué es un paradigma de programación?

3. Citar los tipos de paradigmas de programación que han existido hasta nuestros días

4. ¿Cuál es la diferencia entre el estado y el comportamiento de un objeto?

5. Analizar la diferencia entre clase base, abstracta y derivada

6. Citar las características básicas de un objeto

7. Definir qué es un método

8. ¿Para qué sirven los modificadores de acceso?

9. ¿Cuál es la importancia de la programación orientada a objetos?

Respuestas a las preguntas de 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. Una clase base sirve para heredar a otra, pero también se pueden crear instancias de ella misma. La clase abstracta se usa exclusivamente para que herede a otras clases derivadas, y no permite instanciación. La clase derivada puede heredar tanto de una clase base normal como de una abstracta, hereda los atributos y los métodos.

6. Estado, comportamiento, identidad y tiempo de vida.

7. Un método es la implementación de la lógica de un objeto. Son los algoritmos que se crean en una clase cuyo objetivo es manipular los atributos de la misma.

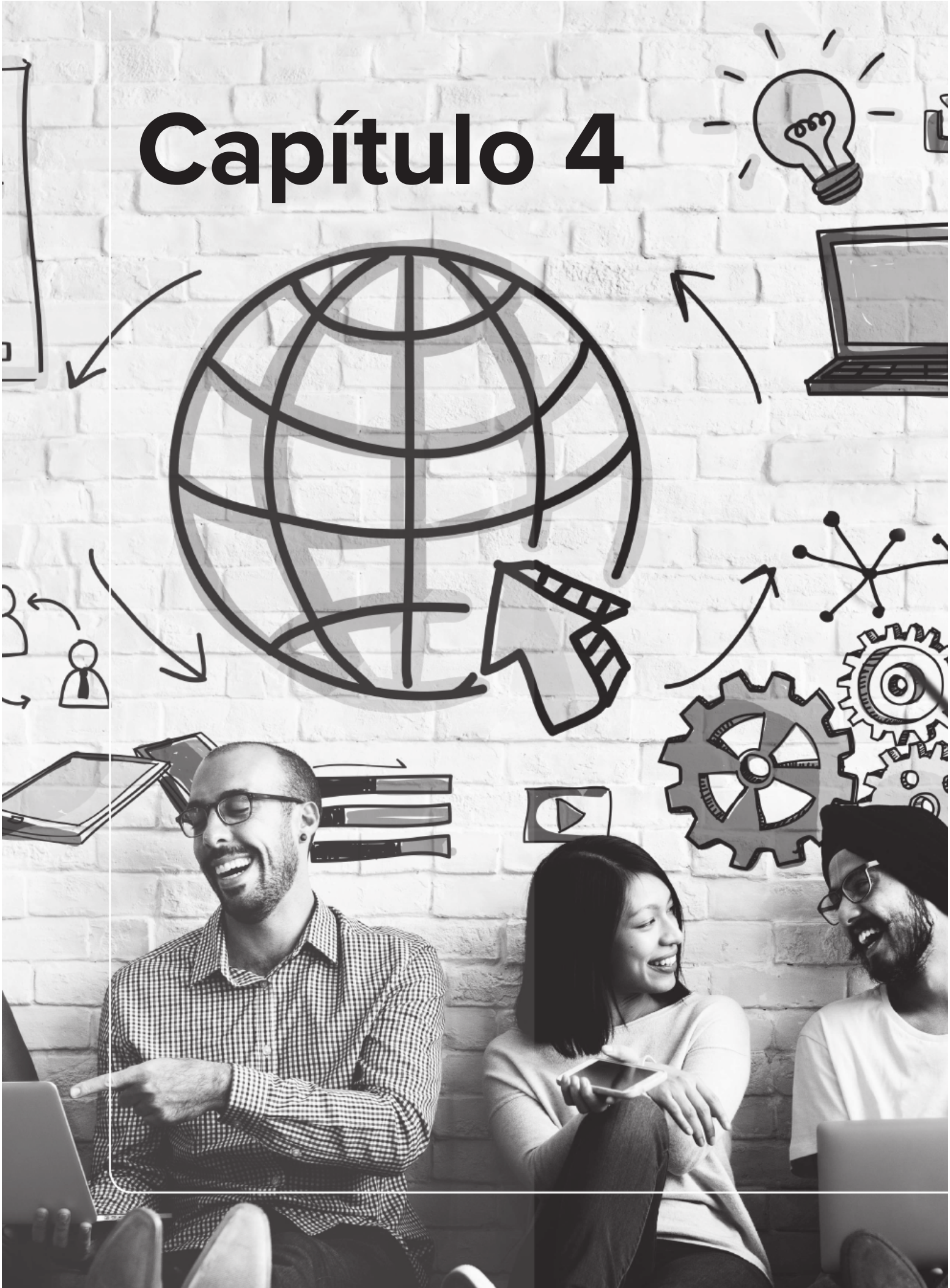
8. Los modificadores de acceso sirven para determinar la visibilidad y acceso que puede tener un atributo o un método desde otras clases en las que se cree instancia de ellas.

9. La importancia de la programación orientada a objetos es que permite resolver problemas de una manera más semejante a la forma en que los seres humanos lo hacen por medio de abstracciones. También es importante porque facilita la programación y la reutilización de código.

Evidencia

- [ ] Desarrolló un programa Java en el que, utilizando NetBeans, implementó las interfaces necesarias para el caso de Medico, Empleado y Persona
- [ ] Desarrolló un programa Java con NetBeans, que implementa la solución del problema de comida china y comida casera
- [ ] Investigó y documentó para qué se utiliza la instrucción super
- [ ] Desarrolló un programa Java con NetBeans para abstraer el cálculo del salario de dos empresas distintas, cuyos procedimientos para calcularlo y los factores a tomar en cuenta fueron diferentes

Capítulo 4



Aplicaciones de escritorio con Java

Reflexione y responda
las siguientes preguntas

¿A qué se le denomina aplicaciones de escritorio?

¿Qué herramientas o controles provee NetBeans para desarrollar aplicaciones de escritorio?

¿Son superadas las aplicaciones de escritorio por los 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

4.1 Introducción
4.2 Componentes de un aplicativo de escritorio
4.3 Paquetes (Packages) en NetBeans

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 hablaba sobre redes Novell o Windows.

Entonces se practicaba el desarrollo de aplicaciones cliente-servidor, donde los archivos de datos radicaban en un servidor y las aplicaciones también, o se instalaba en cada cliente, cuando esto último ocurría 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, y se han llamado aplicaciones de escritorio (*desktop*). Una **aplicación de escritorio** es aquella que está instalada en la computadora del usuario y es ejecutada por el sistema operativo local, de manera que su rendimiento depende de los recursos de hardware (RAM, procesador, disco duro, video) de la PC donde residen. 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), asimismo, son muchas las organizaciones que han migrado sus sistemas de escritorio a aplicaciones basadas en la 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 Java con NetBeans.
- Comprender la funcionalidad de los componentes principales que ofrece el IDE NetBeans 8 para el desarrollo de interfaces gráficas de usuario.
- Desarrollar aplicativos de escritorio (desktop) utilizando el lenguaje de programación Java por medio del IDE NetBeans.

4.1 Introducción

Desarrollar una aplicación de escritorio (desktop) en Java con NetBeans es una tarea que puede pasar de sencilla a compleja, de acuerdo con el problema que se enfrente. Sin embargo, NetBeans ofrece una serie de componentes, librerías y utilitarios que hacen 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, com, Jar, entre otros). Todos los usuarios comparten los mismos datos (servidor de base de datos), pero la aplicación reside en cada ordenador, por lo que el aplicativo debe manejar eficientemente la concurrencia que se presenta en este tipo de soluciones.

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

4.2 Componentes de un aplicativo de escritorio

Desde sus inicios, Java contaba con una librería de soporte del desarrollo de aplicaciones con interfaces gráficas, esta librería se conocía como AWT (kit de herramientas de ventanas abstracta) y no tuvo éxito principalmente porque dependía fuertemente de los componentes nativos del sistema operativo en el que corría la aplicación, además de que por depender del sistema operativo, el comportamiento de los controles gráficos se volvía variable dependiendo de cada sistema, lo que comprometía la portabilidad de las aplicaciones.

En vista de tales problemas, se desarrolló una nueva API para las interfaces gráficas en Java la cual se denominó SWING, esta librería trajo ventajas con respecto a la anterior; por ejemplo, no posee dependencia de plataforma, el diseño de los controles es más atractivo, provee más variedad de controles y con características que los hacen más flexibles para el programador y, por último, crea aplicaciones más ligeras.

Los componentes de un aplicativo de escritorio (desktop) van desde los controles que NetBeans 8 proporciona (tales como etiquetas, cajas de texto, botones de comando, listas, tablas, entre otros), así 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 que se categorizan estos controles.

Tabla 4.1 Principales controles que proporciona NetBeans 8

BIBLIOTECA	COMPONENTES	DESCRIPCIÓN
Contenedores Swing	• Panel (JPanel) • Panel con pestañas (JTabbedPane) • Panel divisor (JSplitPane) • Panel de desplazamiento (JScrollPane) • Barra de herramientas (JToolBar) • Panel de escritorio (JDesktopPane) • Ventana interna (JInternalPane) • 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 agregar los contenedores de peso ligero.
Controles Swing	• Etiqueta: JLabel • Botón: JButton • Botón de 2 posiciones: JToogleButton • Casillas de activación: JCheckBox • Botón de opción: JRadioButton • Grupo de botones: ButtonGroup • Lista desplegable: JComboBox • Lista: JList • Campo de texto: JTextField • Área de texto: JTextArea • Barra de desplazamiento: JScrollBar • Deslizador: JSlider • Barra de progreso: JProgressBar • Cuadro formateado: JFormattedTextField • Cuadro de contraseña: JPasswordField • Spinner: JSpinner • Separador: JSeparator • Panel de texto: JTextPane • Panel editor: JEditorPane • Árbol: JTree • Tabla: JTable	Los controles Swing brindan la funcionalidad requerida para una aplicación de tipo escritorio (desktop). Entre estos componentes se tienen 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 para desarrollar una barra de menús de una aplicación. Desde la barra de menús principal, 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), rellenadores swing, componentes AWT y otras bibliotecas. Nos interesan, básicamente, las bibliotecas citadas en la **tabla 4.1**.

Con este conjunto de controles gráficos (*swing*) es posible desarrollar interfaces gráficas poderosas y que puedan satisfacer todas las necesidades imperantes en los usuarios finales de aplicaciones; aunque no sólo proveen funcionalidad a las aplicaciones, sino que hacen atractivas las ventanas de aquéllas de escritorio, dotando a los desarrolladores de herramientas valiosas para que implementen su arte y creatividad en el desarrollo de aplicaciones de este tipo.

En la **figura 4.1** se muestra la paleta de controles que provee NetBeans para desarrollar aplicaciones Java, las cuales se analizaron en la **tabla 4.1**.

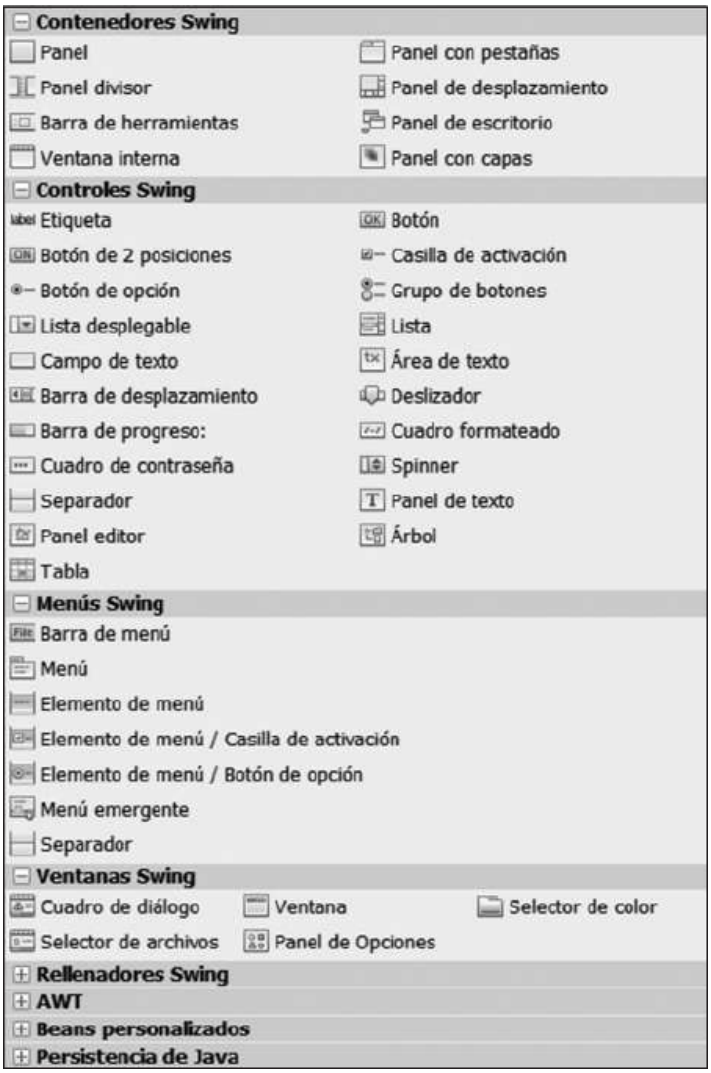


Figura 4.1 Controles que ofrece NetBeans 8

Ejemplo 1: Aplicación al estilo MDI

MDI significa Interfaz de Múltiples Documentos, y se refiere a los programas gráficos cuyas ventanas se encuentran dentro de una ventana padre.

A continuación, se desarrollará un aplicativo que utiliza algunos de estos controles.

1. Ingrese a NetBeans.

2. Una vez en el IDE, seleccione Archivo en el menú principal y de ahí Proyecto Nuevo, o presione conjuntamente las teclas [Ctrl] + [Mayúsculas] + [N] para crear un proyecto NetBeans.

3. Cree un proyecto nuevo de tipo Java Application, tal como se muestra en la **figura 4.2**:

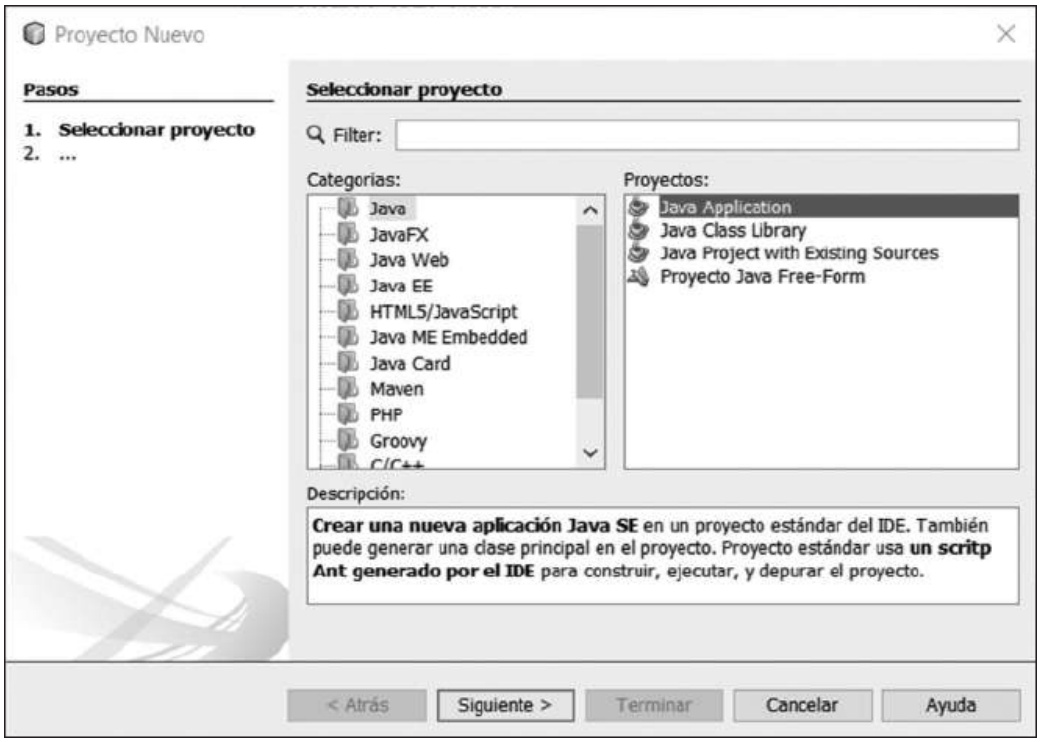


Figura 4.2 Proyecto Java Application

4. Pulse el botón Siguiente y en la pantalla que viene nombre el proyecto como DskCapitulo4. Por último, pulse el botón Terminar. Con ello se crea el nuevo proyecto que puede ser visualizado en el panel izquierdo de NetBeans.

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 4.3**.

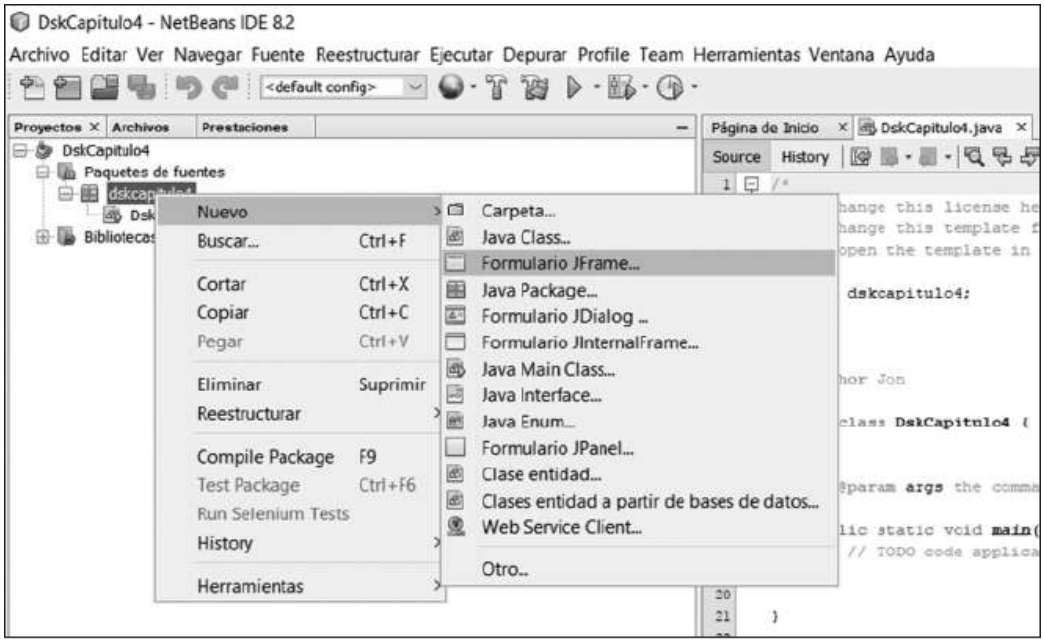


Figura 4.3 Creación del primer formulario JFrame

6. Una vez creado el formulario, debe seleccionar Archivo, Proyecto Properties (propiedades del proyecto) y en las opciones que aparecen seleccionar Ejecutar, y Seleccionar frmMenu como clase principal (Main Class). Puede pulsar el botón Examinar para buscar el formulario que servirá como el principal de la aplicación. La **figura 4.4**. muestra cuál sería nuestro formulario inicial.

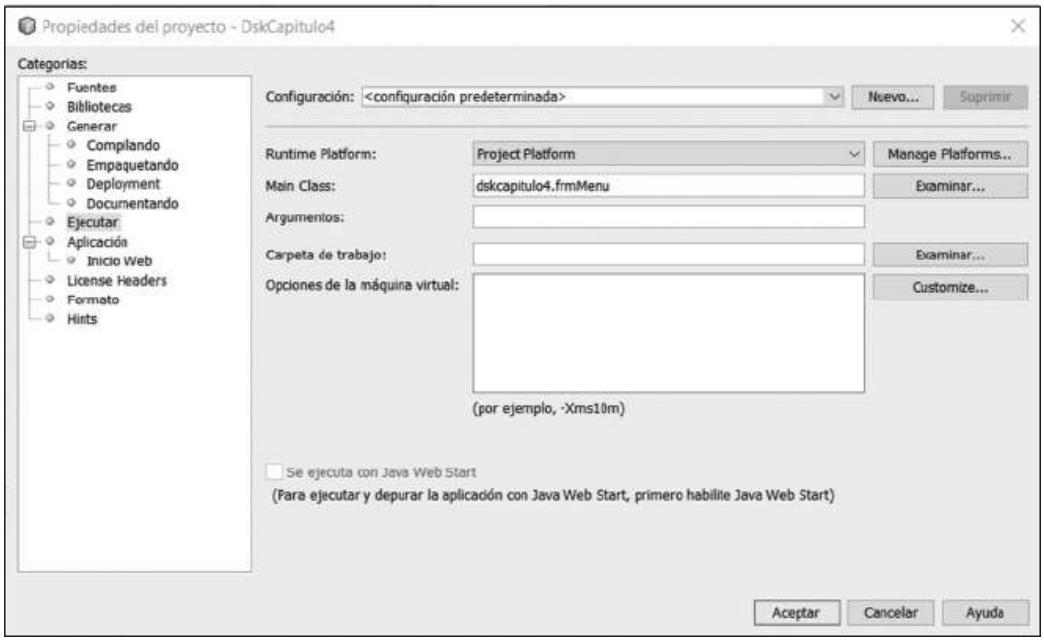


Figura 4.4 Definición de formulario inicial del aplicativo

Si necesitáramos cambiar el nombre del archivo que hemos creado mediante el proceso indicado en la **figura 4.4**, basta con pulsar con el botón derecho del mouse sobre el nombre de dicho archivo y seleccionar Reestructurar, posteriormente Cambiar nombre (o pulsar las teclas CTRL+R).

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 (por lo general es el tipo de diseño por defecto). Esto permitirá dividir la ventana del JFrame en cinco zonas: norte, sur, este, oeste y centro.

8. Agregue una barra de menú pulsando sobre el cuerpo del formulario con el botón derecho del mouse, luego añadir paleta seguido de Menús Swing. Otra forma es desde la paleta gráfica, como la que se muestra en la figura 4.1 en la sección Menús Swing. 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 al pulsar sobre el mismo y editar el texto que presenta. Otra forma de editar el texto es pulsar 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.

En NetBeans, este tipo de panel se utiliza como contenedor de las ventanas internas de una aplicación MDI (Multiple Document Interface). Se muestra de color oscuro para diferenciarlo de los paneles normales. Observe la **figura 4.5**, donde se ve este tipo de formulario.

Figura 4.5 Un JFrame al estilo MDI

11. Cambie el nombre a este panel pulsando sobre el mismo con el botón derecho del mouse y seleccionando la opción Cambiar nombre de variable. Se le puede poner 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 4.6** acerca de este procedimiento.

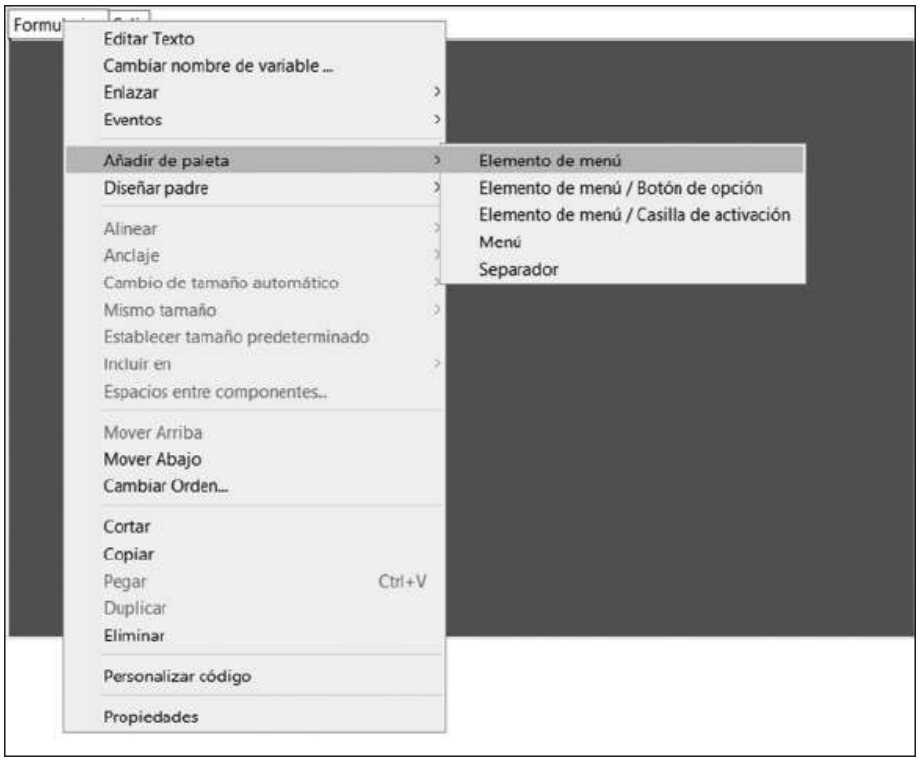


Figura 4.6 Agregar elementos de menú a la barra

Observe algunas funcionalidades que se muestran en la **figura 4.6**, por ejemplo, las opciones Desplazar hacia abajo y Desplazar hacia arriba, las cuales desplazan los elementos de la forma en que lo indican.

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

14. A continuación, pulse con el botón derecho del mouse sobre el formulario frmMantEmpleados y en el menú de contexto que aparece seleccione Añadir de paleta, Contene-

dores Swing y luego Panel. Coloque este objeto en la parte superior del formulario, y otro similar en la segunda mitad de la parte inferior.

15. Seleccione el panel de la primera mitad superior y pulsando el botón derecho del mouse elija Propiedades, luego pulse sobre la propiedad border. Seleccione Borde con título. En el campo 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 4.7** muestra el modo en que hasta el momento nuestro formulario está diseñado.

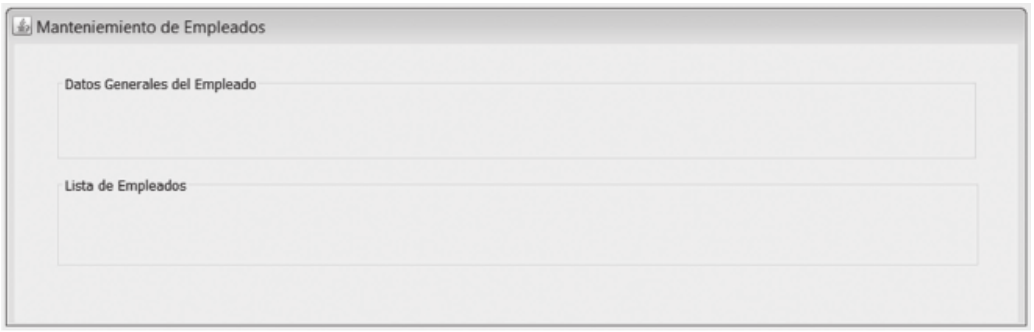


Figura 4.7 Agregar dos objetos panel a nuestro formulario

16. El producto terminado se muestra en la **figura 4.8** con los objetos creados para este formulario. Observe las flechas numeradas del 1 al 17 con los objetos que componen el formulario y que se detallan en la **tabla 4.2**.

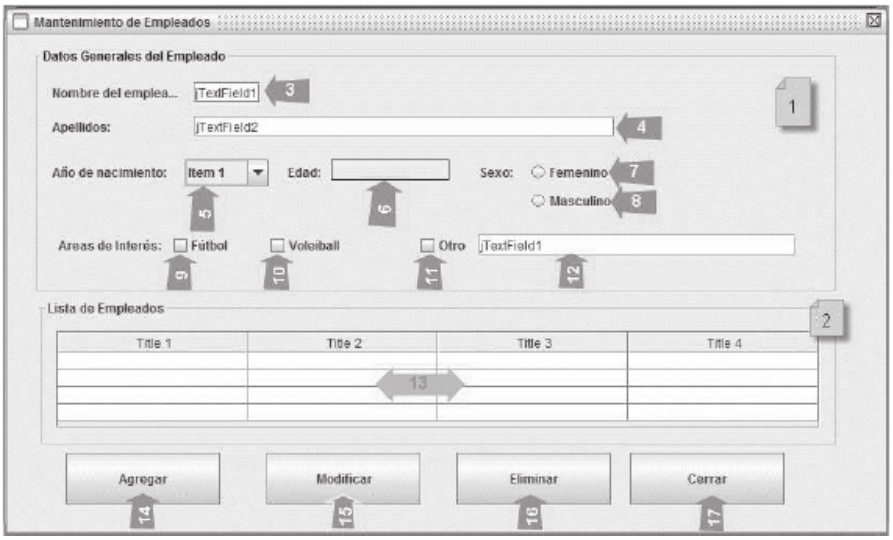


Figura 4.8 El formulario frmMantEmpleados terminado

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: chkVoleiball
11	JCheckBox	Name: chkOtro
12	JTextField	Name: txtOtro
13	JTable	Name: tabEmpleados
14	JButton	Name: btnAgregar
15	JButton	Name: btnModificar
16	JButton	Name: btnEliminar
17	JButton	Name: btnCerrar

Nota: Las que no están numeradas son controles tipo JLabel: Etiquetas

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).

Listado No. 4.1 Variables principales en el formulario frmMantEmpleados

```
1. private int anyoActual, fila;
2. Object[ ] filas = new Object[6];
3. javax.swing.table.DefaultTableModel modeloTabla =
new javax.swing.table.DefaultTableModel( );
```

Nota: En NetBeans puede usar la combinación de teclas Alt + Shift + F, para ordenar y tabular el código fuente de manera automática.

Comentarios sobre el código del listado 4.1:

- La instrucción de la línea 2 del listado 4.1, `Object[] filas = new Object[6]` declara un vector de objetos de 6 posiciones (6 celdas que conformarán los atributos de un empleado en una fila de la tabla). 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 vector Object que puede contener los atributos de un empleado de cada fila (en este caso son 1 filas con 6 campos), tipos de datos tales como int, String o double.

Se declara filas como Object, siendo un vector que en cada celda almacena un valor de diferente tipo; lo cual, en un vector de un tipo de dato específico, no sería posible.

- En la línea 3, 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 del vector 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 declarando una variable de tipo Object en el que se cargan las entradas del usuario (una fila para la tabla), posteriormente una variable de tipo JTable, JComboBox o JList, y se trasladan ahí los datos de la variable Object.

Por último, de la variable JTable, JComboBox o JList se cargan los datos al objeto propiamente (el control) JTable, JComboBox o JList. Vea la **figura 4.9**, donde se muestra el procedimiento para la carga de datos en uno de estos componentes.

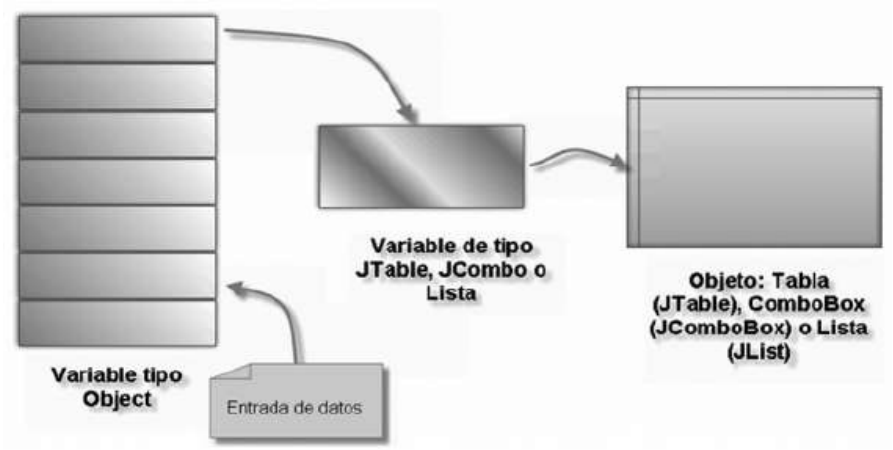


Figura 4.9 Manejo del proceso de carga de un objeto JTable, JComboBox o JList

18. Ahora en el listado 4.2 se procede a programar el evento inicializador de nuestro formulario frmMantEmpleados con los procedimientos que se ejecutarán cuando éste se cargue:

Listado No. 4.2 Inicializar el formulario `frmMantEmpleados`

```
1. public frmMantEmpleados( ) {
2.     initComponents( ); //inicialización de componentes en el formulario
3.     CargarAnyos( ); //método para cargar años en el ComboBox
4.     configurarModelo( ); //método para configurar títulos del encabezado de la tabla (JTable)
5.     detEdad( ); //método para calcular la edad de un empleado
6. }
```

19. Seguido de ello, en el listado 4.3 se programa el método `CargarAnyos( )` con el siguiente código:

Listado No. 4.3 Programación del método `CargarAnyos( )`

```
1. void CargarAnyos( )   {
2.     int i;
3.     javax.swing.DefaultComboBoxModel modeloCombo = new javax.swing.
       DefaultComboBoxModel( );
4.     java.util.Calendar fecha = java.util.Calendar.getInstance( );
5.     anyoActual = fecha.get(java.util.Calendar.YEAR); //obtener año actual
6.     for (i=1950;i<anyoActual;i++) {
7.       modeloCombo.addElement(i); //Carga un elemento al modelo(un año)
8.   }
9.   cmbAnyos.setModel(modeloCombo);  //se carga el combobox desde el // modelo
10. }
```

20. Ahora, en el listado 4.4 se procede a crear el código para el método `configurarModelo ( ).`

Listado No. 4.4 Programación del método `configurarModelo ( )`

```
1. void configurarModelo( ) { //carga los títulos de la cabecera de la tabla
2.     modeloTabla.addColumn("Nombre"); //agrega la columna Nombre
3.     modeloTabla.addColumn("Apellidos"); //agrega la columna Apellidos
4.     modeloTabla.addColumn("Año"); //agrega la columna Año
5.     modeloTabla.addColumn("Edad"); //agrega la columna Edad
6.     modeloTabla.addColumn("Sexo"); //agrega la columna Sexo
7.     modeloTabla.addColumn("Áreas"); //agrega la columna Áreas
8. }
```

21. El método `detEdad( )` permite determinar la edad del empleado. El código se muestra en el listado 4.5.

Listado No. 4.5 Programación del método `detEdad ( )`

```
1. void detEdad( )   {
2. int edad =
       anyoActual Integer.parseInt(cmbAnyos.getSelectedItem().
toString( ));
3.      lblEdad.setText(String.valueOf(edad));
4. }
```

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. Dentro de ese evento haga un llamado al método `detEdad( )`.

23. En el botón Agregar (`btnAgregar`), pulse con el botón derecho del mouse y seleccione la opción Eventos, Action y elija la opción actionPerformed, dentro del método (Evento) que se autogenera escriba el siguiente código que se observa en el listado 4.6:

Listado No. 4.6 Programación del botón agregar (`btnAgregar`)

```
1. detDatos( );   //llamada al método detDatos()
2. modeloTabla.addRow(filas); //carga el modelo con los datos //obtenidos por el método
   detDatos( )
3. tabEmpleados.setModel(modeloTabla); //carga el JTable //tabEmpleados con los datos
   contenidos en el modelo.
```

24. Observe que el procedimiento `detDatos( )` aún no ha sido definido. Por tanto, escriba el código para ese procedimiento como sigue en el listado 4.7:

Listado No. 4.7 Programación del método `detDatos ( )`

```
1. void detDatos( ){
2.   String AInteres="";
3.   filas[0] = txtNombre.getText( ); //carga el nombre en el vector de Object
4.   filas[1] = txtApellidos.getText( ); //carga el apellido
5.   filas[2] = cmbAnyos.getSelectedItem( ).toString( ); //carga el año de nacimiento
6.   filas[3] = lblEdad.getText( ); //carga la edad
7.   if (rdMasculino.isSelected( )) {filas[4] = "Masculino";} //si es masculino
8.   else {filas[4] = "Femenino";} //o si es femenino
9.   if (chkFutbol.isSelected( )) AInteres=chkFutbol.getText() + ", ";
10.  if (chkVoleiball.isSelected( )) AInteres = AInteres + chkVoleiball.
     getText( )+ ", ";
```

```
11.  if (chkOtro.isSelected( )) AInteres = AInteres + txtOtro.getText( );
12.  filas[5] = AInteres; //agrega en la columna 5 la variable AInteres
13.}
```

25. En el botón Modificar (`btnModificar`) en el evento actionPerformed, escriba el código que se despliega en el listado 4.8:

Listado No. 4.8 Programación del botón modificar (`btnModificar`)

```
1. detDatos( );
2. for (int i=0;i<6;i++) //para las 6 columnas de la tabla
3.      modeloTabla.setValueAt (filas[i], fila, i); //cambie en el modelo por lo
        // que tiene  almacenado el vector filas
4.   tabEmpleados.setModel(modeloTabla);  //actualiza la tabla   //(JTable) con el modelo
```

26. En el botón Eliminar (`btnEliminar`) en el evento actionPerformed, escriba el código del listado 4.9:

Listado No. 4.9 Programación del botón eliminar (btnEliminar)

```
1. modeloTabla.removeRow(fila); //elimina la fila determinada.
2. tabEmpleados.setModel(modeloTabla);
```

27. Pulse con el botón derecho del mouse sobre la tabla del panel de abajo del formulario frmMantEmpleados y asegúrese de seleccionar Eventos, Mouse y MouseClicked y escriba ahí dentro del método-evento, 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 elegirse femenino y viceversa (selección única). Para estos efectos, pulse sobre el formulario con el botón derecho del mouse y seleccione Anadir de Paleta, Controles Swing y ahí elija el control llamado Grupo de botones. Se agregará automáticamente un grupo de botones que no tendrá visibilidad (es como crear una variable o un control oculto).

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 si no lo ha renombrado).

Con ello se enlaza el radioButton a este grupo de botones. Seleccione también la propiedad Selected habilitando la caja de check que presenta. Así, se mostrará que

este radioButton inicialmente está seleccionado. Haga el mismo proceso de enlazar el siguiente radioButton (rdFemenino) al buttonGroup. Ahora ambos radioButton están sincronizados, es decir, sólo uno de ellos se podrá seleccionar al mismo tiempo.

El primero:
En el evento btnCerrarActionPerformed( ) del botón Cerrar escriba la siguiente línea de código: this.dispose();

El segundo:
Abra el formulario frmMenu, diríjase a la pestaña de diseño para desplegar el formulario principal y allí haga clic en el menú Formularios y en la opción de Menú Mantenimiento de Empleados; estando allí dé doble clic sobre esta opción y dentro del evento ActionPerformed del ítem del menú, escriba el siguiente código:

```
1. frmMantEmpleados frmEmpl = new frmMantEmpleados();
2. this.panelInterno.add(frmEmpl);
3. frmEmpl.setVisible(true);
```

29. Ejecute el ejemplo y podrá agregar, modificar y eliminar registros desde la tabla. Una ejecución común de esta aplicación se muestra en la **figura 4.10**.

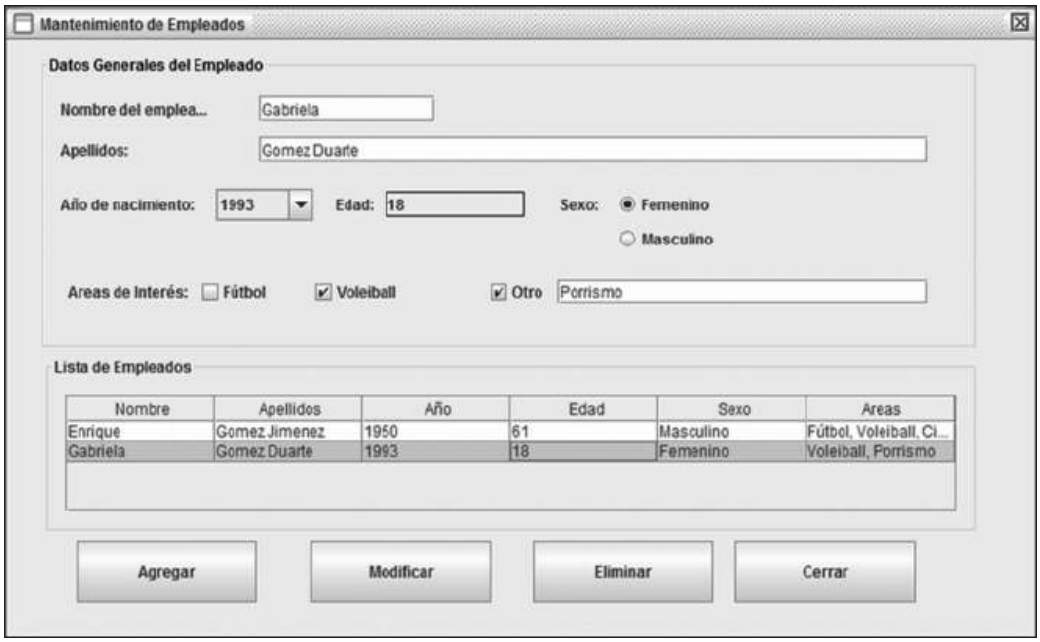


Figura 4.10 Ejecución de nuestro formulario frmEmpleados

Observe también la **figura 4.11**, el formulario frmEmpleados de la **figura 4.10** queda dentro del entorno del formulario de menú. Este concepto indica que JFrame del menú es el padre. Todos los formularios que se creen como éste, toman como padre a frm-Menu y nuestra aplicación tendrá un aspecto similar siempre. Se trata de simular una metáfora de escritorio.

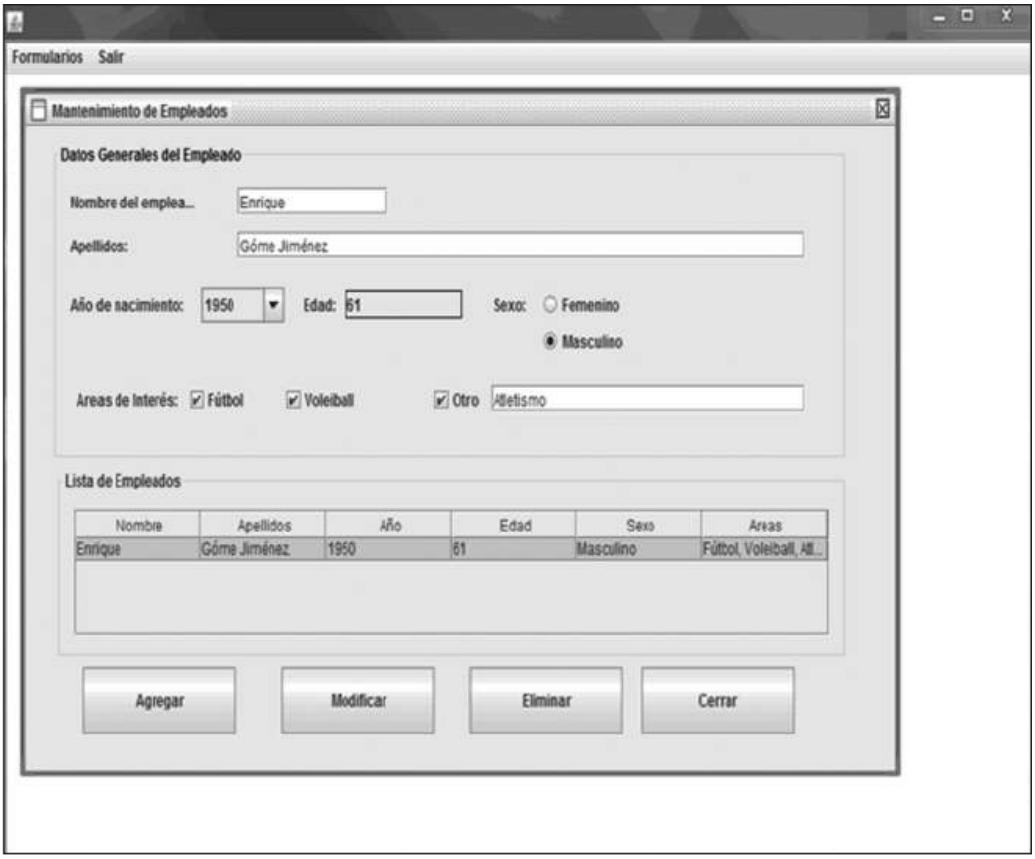


Figura 4.11 Aplicación estilo MDI

Actividades para el lector

Desarrolle programas en Java con NetBeans para implementar:

- El mantenimiento de departamentos del empleado. Los campos serán nombre del departamento, número de teléfono y nombre director. La interfaz 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 interfaz debe ser similar a la desarrollada en el ejemplo anterior.

4.3 Paquetes (Packages) en NetBeans

Las clases en Java pueden agruparse en familias lógicas, a esto se le denomina paquetes. Al igual que en .NET de Microsoft se utilizan los espacios de nombres (namespaces), en Java es posible agrupar clases con funcionalidades lógicas determinadas. Por defecto, 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 nombreVariable.

¿Cuál es la utilidad de los paquetes? Básicamente es el ordenamiento de las clases en grupos funcionales. Asimismo, evita el infierno de las dll que se producía hace unos años, las colisiones por nombres similares en las dll (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 de otros paquetes.

Una vez que se han acumulado muchas clases dentro de un paquete, es importante generar un archivo que se pueda importar en sus aplicaciones, cuyo uso sea fácil y práctico. Para ello es posible generar los denominados archivos JAR (*Java Archives*)

La variable CLASSPATH permite determinar qué librerías se encuentran disponibles para generar el ejecutable. Con CLASSPATH se determina la ubicación de las librerías en un aplicativo que 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. Cree un proyecto nuevo en NetBeans de tipo Biblioteca de clases Java (Java Class Library), como se muestra a continuación en la **figura 4.12**.

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 en el momento necesario de incluirlo en otro proyecto.

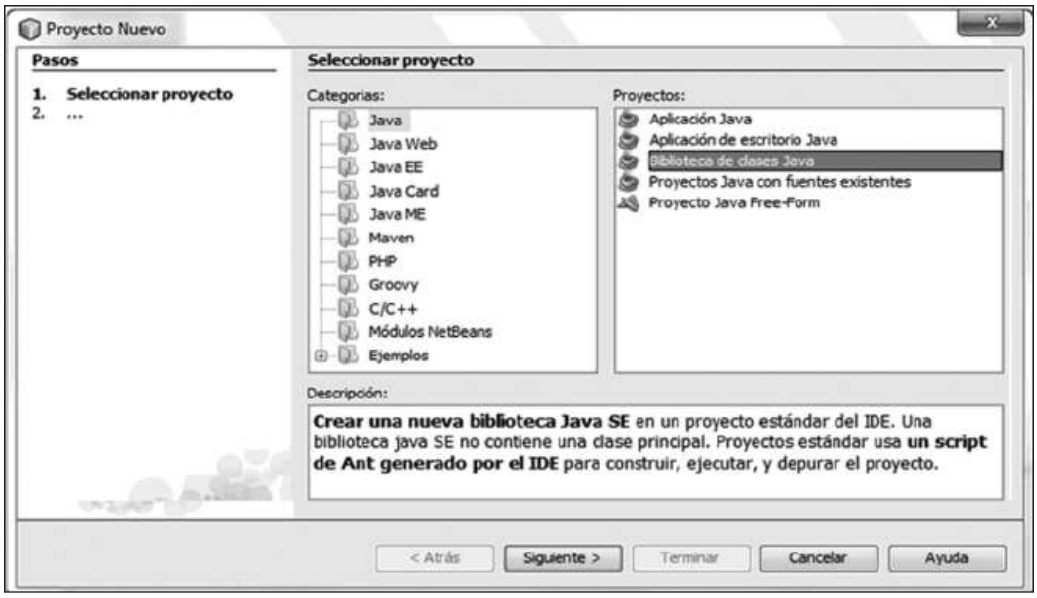


Figura 4.12 Creación de una biblioteca de clases

4. Pulse sobre el nombre del proyecto y con el botón derecho del mouse seleccione Nuevo y la opción Paquete Java, como se muestra en la **figura 4.13**.

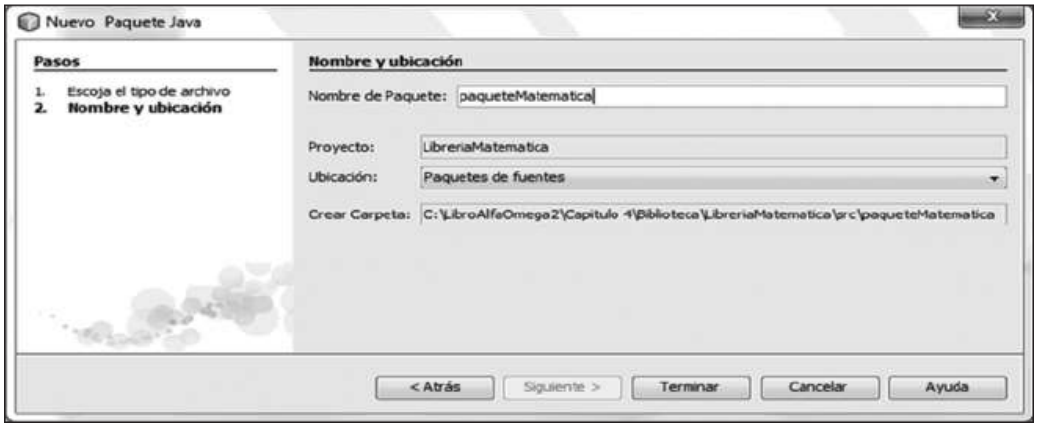


Figura 4.13 Creación de nuestro paquete (package)

5. Observe que el nombre del paquete es paqueteMatematica. Pulse el botón Terminar.

6. Ahora dé clic sobre el nombre del paquete, con el botón derecho del mouse seleccione la creación de una nueva clase (Clase Java) denominada Aritmetica, como se muestra en la **figura 4.14**:

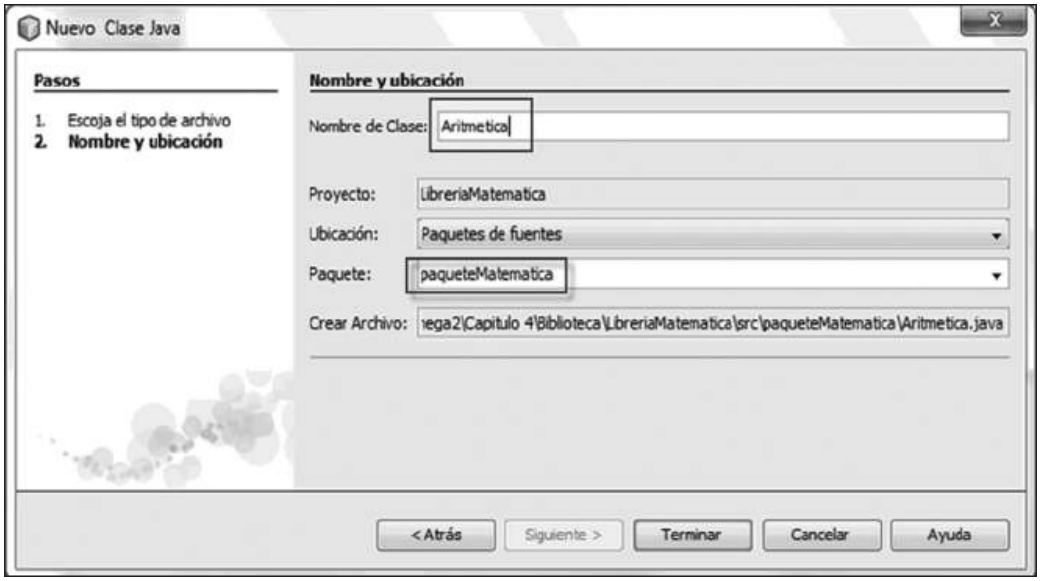


Figura 4.14 Agregar una clase a nuestro paquete (package)

7. Pulse el botón Terminar. A continuación, programe la clase Aritmética, como se muestra en el listado 4.10.

Listado No. 4.10 Creación de la clase Aritmetica

```
1. public class Aritmetica {
2.     public double sumar (double a, double b){return a+b;}
3.     public double restar (double a, double b){return a-b;}
4.     public double multiplicar (double a, double b){return a*b;}
5.     public double dividir (double a, double b){return a/b;}
6. }
```

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

Listado No. 4.11 Creación de la clase `Calculo`

```
1. public class Calculo {
2.     public double raiz(double x){return Math.sqrt(x);}
3.    public double potencia (double x, double y) {return Math.pow(x, y);}
4. }
```

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 proyecto y seleccione la opción Generar. Con ello ha creado 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. Nombremos este proyecto como Calculadora.

12. Ubique este proyecto en la ruta que desee y luego pulse el botón Terminar.

13. A continuación, agregue la librería de clase (Agregar proyeto…) pulsando sobre la carpeta Librería de su proyecto, como se muestra en la **figura 4.15**:

Figura 4.15 Agregar nuestra librería (.JAR) a una aplicación de escritorio

14. Una vez que pulse Agregar Proyecto se mostrará la ventana de selección del JAR que creó (localice el proyecto `LibreriaMatematica` que creó anteriormente en la ruta donde lo guardó). En este caso, se encuentra en la carpeta que se muestra en la **figura 4.16**:

Figura 4.16 Agregar nuestra librera JAR

15. Pulse el botón Agregar al proyecto archivos JAR. Ya puede ubicar el archivo JAR (nuestra biblioteca) en la aplicación. Observe la **figura 4.17**.

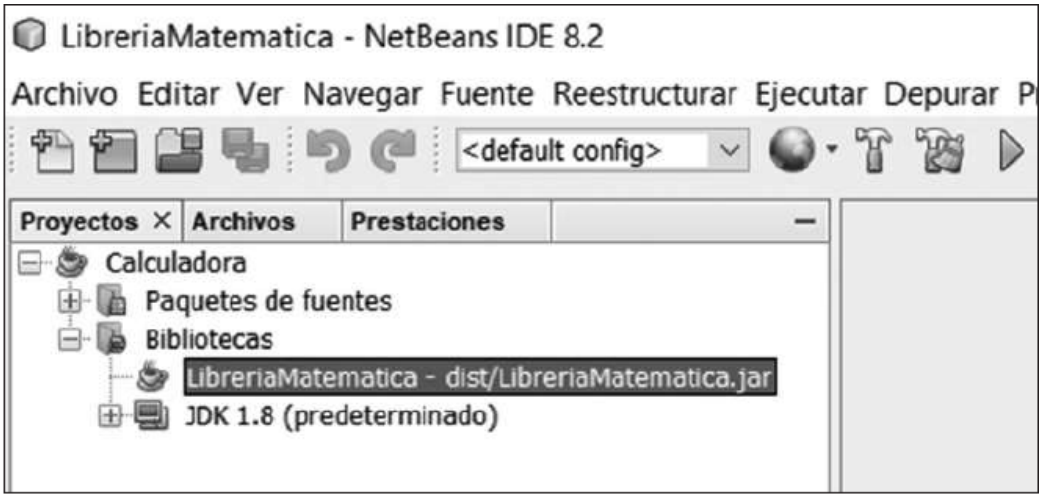


Figura 4.17 Librería JAR habilitada

16. Una vez vinculada la librería al proyecto, cree un paquete para alojar la interfaz gráfica. Haga clic derecho sobre el proyecto y elija la opción Java Package. Se le llamará Vista, como se muestra en la **figura 4.18**. Pulse el botón Terminar.

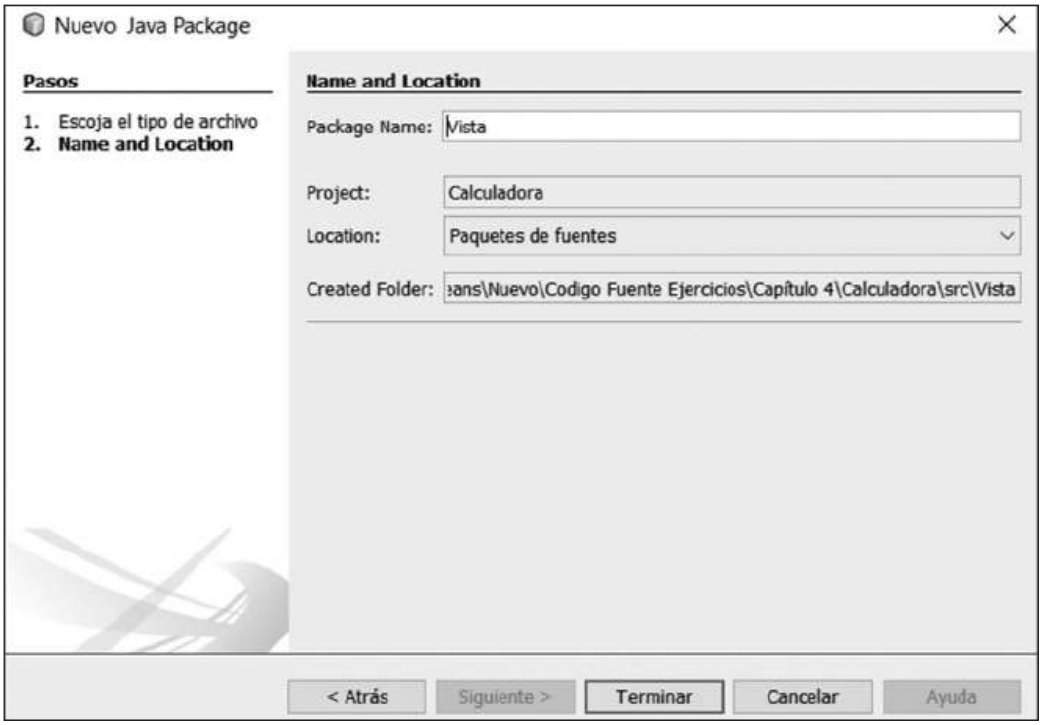


Figura 4.18 Creación de paquete Vista

17. En este proyecto se crea una calculadora muy sencilla para trabajar con números enteros, que permite la realización de operaciones aritméticas básicas y el cálculo de una potencia y raíces cuadradas. El propósito principal es demostrar el uso de la biblioteca de clase que se creó en el ejemplo anterior, en el proyecto Calculadora.

18. Luego, cree un nuevo Formulario JFrame con clic derecho sobre el paquete Vista en la opción Nuevo. Al nuevo formulario se le llamará CalculadoraApp.

19. Diseñe el formulario principal CalculadoraApp (JFrame) de nuestra aplicación de escritorio, con base en la **figura 4.19**.

Figura 4.19 Formulario JFrame CalculadoraApp

20. En la siguiente tabla se describirán todos los controles gráficos que se usarán en el formulario anterior.

Tabla 4.3 Controles Swing del formulario `CalculadoraApp`

No.	TIPO DE COMPONENTE	ATRIBUTOS PRINCIPALES
1	JTextField1	Name: txtPantalla
2	JButton	Name: btnAc
3	JButton	Name: btnDiv
4	JButton	Name: btnMulti
5	JButton	Name: btnBorrar
6	JButton	Name: btn7

No.	TIPO DE COMPONENTE	ATRIBUTOS PRINCIPALES
7	JButton	Name: btn8
8	JButton	Name: btn9
9	JButton	Name: btnResta
10	JButton	Name: btn4
11	JButton	Name: btn5
12	JButton	Name: btn6
13	JButton	Name: btnSum
14	JButton	Name: btn1
15	JButton	Name: btn2
16	JButton	Name: btn3
17	JButton	Name: btnPotenc
18	JButton	Name: btnNegat
19	JButton	Name: btn0
20	JButton	Name: btnRaiz
21	JButton	Name: btnIgual

21. Dé clic derecho sobre cada control del formulario y elija Cambiar nombre de variable. Asigne a cada control el nombre respectivo según la **tabla 4.3**.

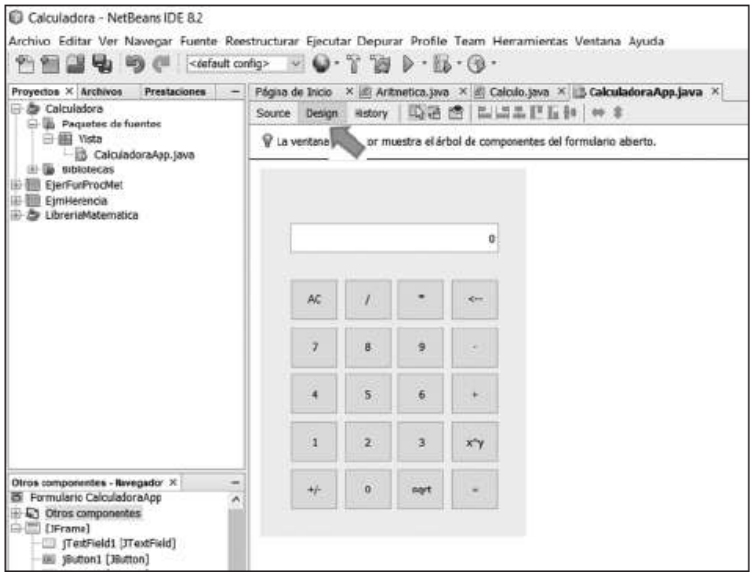


Figura 4.20 Vista de diseño de un formulario en NetBeans

22. Cuando se crea un formulario en NetBeans se tienen 2 vistas para trabajarlo, la primera en la vista gráfica de diseño (Design) y la segunda (Source) donde se tiene acceso al código fuente del formulario. En la **figura 4.20** y **4.21** se ilustran estas 2 vistas.

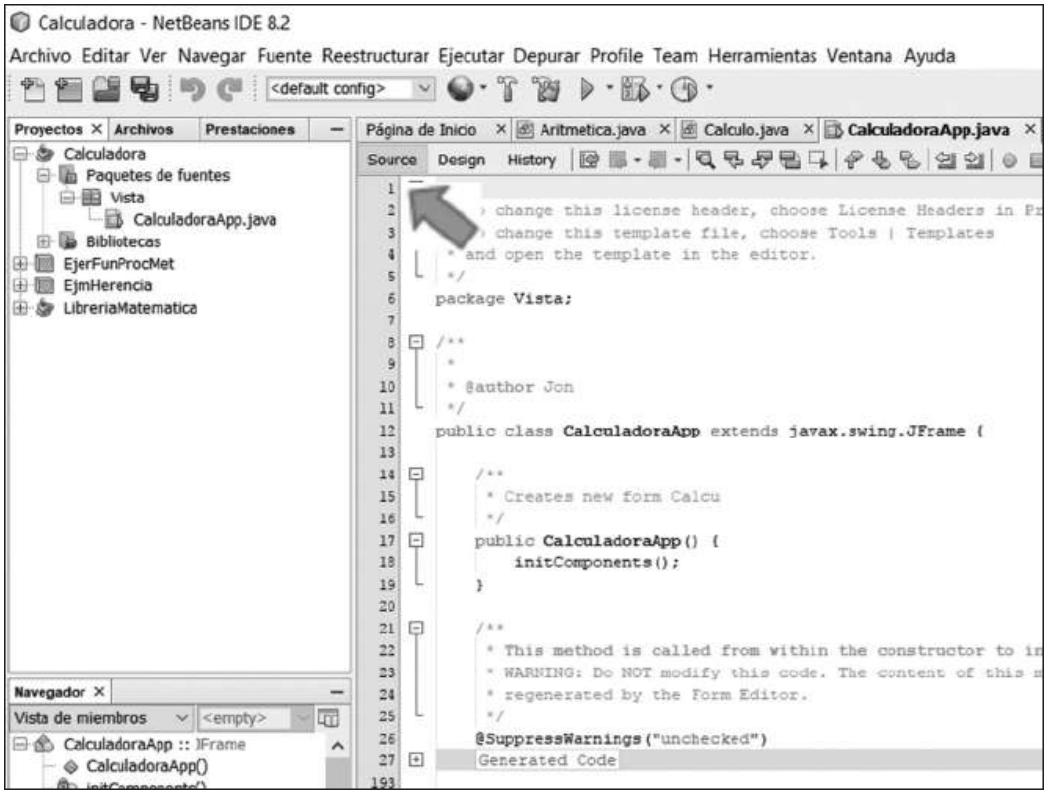


Figura 4.21 Vista del código fuente de un formulario en NetBeans

23. Haga clic derecho sobre el formulario Jframe CalculadoraApp y elija la opción Propiedades, en el atributo `title` establezca Calculadora como título del formulario.

24. Luego, sitúese en el cuadro de Texto de la Pantalla en vista de diseño y haga clic derecho, seleccione la opción propiedades, busque el atributo text y establézcalo en 0, luego busque el atributo `horizontalAlignment` de la ventana de propiedades y elija la opción RIGHT, para que el texto de la caja se alinee a la derecha, simulando una calculadora.

25. En la clase principal del formulario de la **figura 4.19** se deben instanciar las clases `Aritmetica` y `Calculo` de la biblioteca que se agregó al proyecto. Observe la **figura 4.22** donde se muestra la implementación de estas instanciaciones.

26. El código para utilizar estas instanciaciones se muestra en el listado 4.12:

Listado No. 4.12 Creación de instancias de las clases del paquete `Matematica`

```
paqueteMatematica.Aritmetica operaArit = new paqueteMatematica.Aritmetica();
paqueteMatematica.Calculo operaCalc = new paqueteMatematica.Calculo();
```

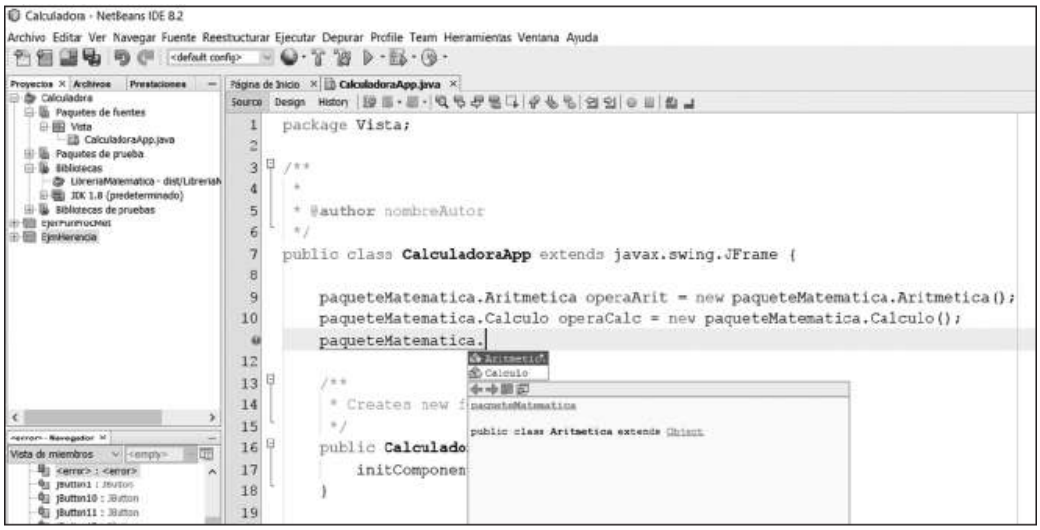


Figura 4.22 Instanciaciones de clases contenidas en PaqueteMatematica

27. Después de haber creado las instancias anteriores, declare las siguientes variables, según la **figura 4.23**.

```
7   public class CalculadoraApp extends javax.swing.JFrame {
8
9       paqueteMatematica.Aritmetica operaArit = new paqueteMatematica.Aritmetica();
10      paqueteMatematica.Calculo operaCalc = new paqueteMatematica.Calculo();
11      Double num1 = 0.0;
12      Double num2 = 0.0;
13      Double result = 0.0;
14      Integer operacion = 0; //1. Suma  2. Resta  3. Div  4. Multip 5.Potencia
15
```

Figura 4.23 Declaración de variables para la calculadora

28. La variable `num1` almacenará el primer número de la operación, al igual que `num2` almacenará el segundo número necesario para realizar la operación que el usuario solicite. La variable `result` almacenará el resultado de la operación que se ejecuta, cuyo valor luego se mostrará en `txtPantalla`. Por último, a la variable operación se le dará un número del 1 al 5, dependiendo de la operación que el usuario haya presionado.

29. Sitúese antes de la declaración del método public static void main(String args[]) y escriba el método que se muestra en el listado 4.13 y en el 4.14.

Listado No. 4.13 Creación del método `escribeNum ( )`

```
1. public void escribeNum (int num){
2.       if(txtPantalla.getText().equals("0")){
3.          txtPantalla.setText("");
4.       }
5.       txtPantalla.setText(txtPantalla.getText()+ String.valueOf(num));
6. }
```

30. En el listado anterior, en el `if`, si la pantalla sólo tiene un cero se limpia, si no se concatena lo que tiene con el número que se recibe por pantalla, de esta manera se puede escribir en ella con los botones, números de más de una cifra. Este método se debe llamar desde cada botón de números enviando por parámetro el número presionado.

Listado No. 4.14 Creación del método `limpiarVariables ( )`

```
1. public void limpiarVariables() {
2.       num1 = 0.0;
3.       num2 = 0.0;
4.       txtPantalla.setText("0");
5.       operacion = 0;
6. }
```

31. El método `limpiarVariables()` del listado 4.14 tiene como objetivo reinicializar las variables usadas por la calculadora para prepararla para un nuevo cálculo; al mismo tiempo se inicializa la pantalla en 0.

32. De aquí en adelante se programará el código fuente necesario para cada uno de los botones númericos del 0 al 9. Se inicia por el cero. Haga clic derecho en el botón cero y elija Eventos – Action – ActionPerformed, de manera automática se pasará a la vista de código fuente, como se muestra en la **figura 4.24**.

Figura 4.24 Acceso al evento ActionPerformed del botón 0

33. En cada botón numérico se debe llamar al método `escribeNum()`, y se le envía por parámetro el número del botón presionado. En la **figura 4.25** se muestra el código fuente de los botones del 0 al 3.

```
242    private void btn0ActionPerformed(java.awt.event.ActionEvent evt) {
243        // TODO add your handling code here:
244        if (!txtPantalla.getText().equals("0")){
245            txtPantalla.setText(txtPantalla.getText()+ "0");
246        }
247    }
248
249    private void btn1ActionPerformed(java.awt.event.ActionEvent evt) {
250        // TODO add you dling code here:
251        escribeNum(1);
252    }
253
254    private void btn2ActionPerformed(java.awt.event.ActionEvent evt) {
255        // TODO add you dling code here:
256        escribeNum(2);
257    }
258
259    private void btn3ActionPerformed(java.awt.event.ActionEvent evt) {
260        // TODO add you dling code here:
261        escribeNum(3);
262    }
```

Figura 4.25 Código fuente del evento ActionPerformed de los botones del 0 al 3

34. Observe las flechas de la **figura 4.25**, se llama al método escribeNum () dentro de cada evento por botón y se le envía por parámetro el número representado por cada uno. Desarrolle el código de los botones 4 al 9 con base en el código que se muestra en la figura 4.25, copie y pegue el código en el evento `ActionPerformed` de cada botón numérico, con la diferencia, como ilustra la flecha de la figura, de que debe cambiar el número al representado por cada botón.

35. Ahora es preciso continuar con la programación de los botones de operaciones aritméticas. Seleccione el evento actionPerfomed del botón Sumar, Restar, Dividir, Multiplicar y escriba el código que se muestra en la **figura 4.26** dentro de cada evento.

36. En la **figura 4.27** se agrega el código del botón Potencia (x^y).

37. Proceda a crear el código del botón Ac, el cual limpia la pantalla y del botón borrar. Este código se visualiza en la **figura 4.28**.

```
349     private void btnSumActionPerformed(java.awt.event.ActionEvent evt) {
350         // TODO add your handling code here:
351         operacion = 1; // 1 significa operación Sumar
352         num1 = Double.parseDouble(txtPantalla.getText());
353         txtPantalla.setText("0");
354     }
394     private void btnDivActionPerformed(java.awt.event.ActionEvent evt) {
395         // TODO add your handling code here:
396         operacion = 3; // 3 significa operación Dividir
397         num1 = Double.parseDouble(txtPantalla.getText());
398         txtPantalla.setText("0");
399     }
400
401     private void btnMultiActionPerformed(java.awt.event.ActionEvent evt) {
402         // TODO add your handling code here:
403         operacion = 4; // 4 significa operación Multiplicar
404         num1 = Double.parseDouble(txtPantalla.getText());
405         txtPantalla.setText("0");
406     }
407
408     private void btnRestaActionPerformed(java.awt.event.ActionEvent evt) {
409         // TODO add your handling code here:
410         operacion = 2; // 2 significa operación Resar
411         num1 = Double.parseDouble(txtPantalla.getText());
412         txtPantalla.setText("0");
413     }
```

Figura 4.26 Código fuente del evento ActionPerformed de los botones +, -, /, *

```
438     private void btnPotencActionPerformed(java.awt.event.ActionEvent evt) {
439         // TODO add your handling code here:
440         operacion = 5; // 5 significa operación Potencia
441         num1 = Double.parseDouble(txtPantalla.getText());
442         txtPantalla.setText("0");
443     }
```

Figura 4.27 Código fuente del evento ActionPerformed del botón Potencia

```
415     private void btnAcActionPerformed(java.awt.event.ActionEvent evt) {
416         // TODO add your handling code here:
417         txtPantalla.setText("0");
418         limpiarVariables();
419         result = 0.0;
420     }
421
422     private void btnBorrarActionPerformed(java.awt.event.ActionEvent evt) {
423         // TODO add your handling code here:
424         if (txtPantalla.getText().length() > 1) {
425             txtPantalla.setText(txtPantalla.getText().substring(0,
426                     txtPantalla.getText().length() - 1));
427         } else {
428             txtPantalla.setText("0");
429             limpiarVariables();
430             result = 0.0;
431         }
432     }
```

Figura 4.28 Código fuente del evento ActionPerformed del botón Ac y borrar

38. Continúe con el desarrollo del código del evento `ActionPerformed` del botón Negativo o Cambiar signo, como se muestra en la **figura 4.29**.

```
434     private void btnNegatActionPerformed(java.awt.event.ActionEvent evt) {
435         // TODO add your handling code here:
436         if (txtPantalla.getText().charAt(0) != '-') {
437             txtPantalla.setText("-" + txtPantalla.getText());
438         }else{
439             txtPantalla.setText(txtPantalla.getText().substring(1,
440                     txtPantalla.getText().length()));
441         }
442     }
```

Figura 4.29 Código fuente del evento ActionPerformed del botón btnNegat

39. El código que se debe programar en el evento `ActionPerformed` del botón igual se visualiza en la **figura 4.30**.

```
356     private void btnIgualActionPerformed(java.awt.event.ActionEvent evt) {
357         // TODO add your handling code here:
358         num2 = Double.parseDouble(txtPantalla.getText());
359         switch (operacion) {
360             case 1: //Operación Sumar
361                 result = operaArit.sumar(num1, num2);
362                 limpiarVariables();
363                 break;
364             case 2: //Operación Resta
365                 result = operaArit.restar(num1, num2);
366                 limpiarVariables();
367                 break;
368             case 3: //Operación División
369                 result = operaArit.dividir(num1, num2);
370                 limpiarVariables();
371                 break;
372             case 4: //Operación Multiplicación
373                 result = operaArit.multiplicar(num1, num2);
374                 limpiarVariables();
375                 break;
376             case 5://Operación Potencia
377                 result = operaCalc.potencia(num1, num2);
378                 limpiarVariables();
379                 break;
380         }
381         txtPantalla.setText(String.valueOf(result));
382         result = 0.0;
383     }
```

Figura 4.30 Código fuente del evento ActionPerformed del botón Igual

40. En la figura anterior se puede ver cómo se usan los objetos `operaArit` y `operaCalc` para poder realizar las operaciones suma, resta, división, multiplicación y potencia, in-

vocando a los respectivos métodos de la librería `Matematica` que se incluyó en el proyecto de la `CalculadoraApp`.

41. Aún resta incluir el código necesario para el evento `ActionPerformed` del botón `Raiz`. Éste se muestra en la **figura 4.31**.

```
385    private void btnRaizActionPerformed(java.awt.event.ActionEvent evt) {
386        // TODO add your handling code here:
387        num1 = Double.parseDouble(txtPantalla.getText());
388        result = operaCalc.raiz(num1);
389        limpiarVariables();
390        txtPantalla.setText(String.valueOf(result));
391        result = 0.0;
392    }
```

Figura 4.31 Código fuente del evento ActionPerformed del botón Raiz

42. Cuando concluya la codificación del código fuente del ejemplo, es tiempo de correr la aplicación. La **figura 4.32** muestra una ejecución típica.

Figura 4.32 Ejecución de aplicación CalculadoraApp

43. Recuerde que el proyecto de la calculadora es básico y tiene el fin primordial de usar la biblioteca `Matematica`. No permite introducir número decimal ni hacer operaciones acumulativas, por ejemplo 3+2-4.

Actividades para el lector

- Utilice NetBeans para desarrollar un programa en Java que cree un paquete que permita procesar los cálculos empleados en la actividad para lector del ejemplo 1.
- Redefina la aplicación anterior del ejemplo 1 para el uso de este nuevo paquete que va a crear.

Con esto se llega al final del presente capítulo. Se pretende 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 capítulo se han desarrollado los fundamentos de la programación orientada al escritorio (desktop). A pesar de no detallar gran parte de la teoría, se desarrollan dos ejemplos claros y sencillos de cómo utilizar algunos componentes usuales de este tipo de aplicaciones, y cómo crear y utilizar un paquete (packages) en un aplicativo desktop. Además, se brindó una introducción al uso de hilos en Java en el capítulo 2. Básicamente, se trataron los siguientes temas:

- Explicar los principales componentes que conforman un aplicativo desktop o de escritorio
- Desarrollar una aplicación de escritorio (desktop)
- Explicar el funcionamiento de los paquetes (packages) en NetBeans
- Desarrollar un paquete (package) en NetBeans y el aplicativo de escritorio (desktop) que lo utiliza
- Uso de eventos y propiedades de algunos controles más usados
- Intercambiar entre la vista de diseño y la de código fuente en un formulario

Evidencia

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

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

☐ Implementó un programa Java con NetBeans para procesar los cálculos empleados en el ejercicio anterior y redefinió el aplicativo para que admita el uso de este paquete.

☐ Diseñó y programó una calculadora gráfica que pone a prueba y en práctica los conocimientos en el manejo de aplicaciones de escritorio.

☑ Preguntas de autoevaluación

1. ¿Qué es una aplicación de escritorio (desktop)?

2. ¿Qué son los componentes de un aplicativo 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 un objeto JTable, JComboBox o JList?

6. ¿Qué son los paquetes (packages) en NetBeans?

7. ¿Para qué sirve el evento ActionPerformed de un botón?

8. ¿Cómo se logra extraer un fragmento de una cadena de texto?

☑ Respuestas a las preguntas de autoevaluación

1. Son aquellas que se desarrollan para utilizarse en un ambiente de red local (LAN) de una empresa o de manera stand alone. Se instalan en la computadora.

2. Son los controles que aporta NetBeans a través de la librería gráfica Swing de Java, para la creación de aplicaciones de escritorio (desktop).

3. Para incluir (agrupar) en ellos los componentes de peso ligero que se agregan en NetBeans tales como cajas de texto, etiquetas, entre otros.

4. Para acoplarse en una aplicación de escritorio (desktop) que maneja una interfaz al estilo MDI.

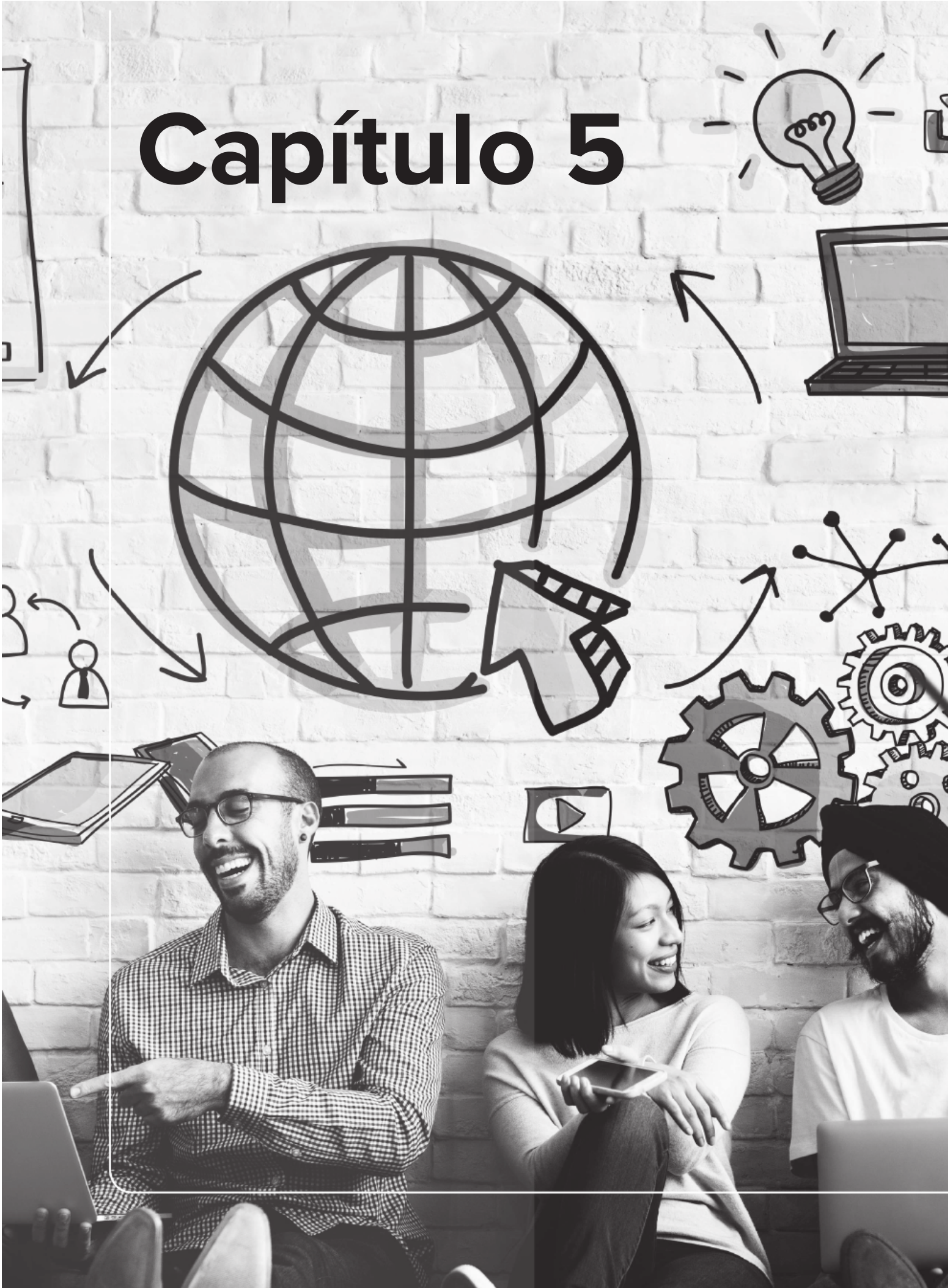
5. Primero se carga un vector de tipo Object, los datos se pasan a una variable del tipo requerido (JTable, JComboBox o JList) y finalmente de éste se pasa al objeto JTable, JComboBox o JList.

6. Son librerías que agrupan clases funcionales y que pueden ser utilizadas en cualquier programa que se desarrolle con NetBeans, generando un .JAR.

7. Sirve para realizar alguna acción en el programa cuando se haga clic al botón.

8. Recordemos que todos los lenguajes de programación poseen funciones predefinidas para fecha, texto, conversiones, operaciones matemáticas, entre otras. En este caso, se puede extraer un fragmento del texto de un String usando una función predefinida llamada substring (inicio, fin).

Capítulo 5



Manejo de archivos de texto y binarios en Java

Reflexione y responda
las siguientes preguntas

¿En qué situaciones se pueden utilizar los archivos de texto si las bases de datos ofrecen una mejor opción de gestión de los mismos?

¿Qué nivel de experiencia en archivos de texto y binarios se requiere para desarrollar nuestras labores técnicas de programación de computadoras?

¿Conoce la diferencia entre un archivo de texto y un archivo binario?

Contenido

5.1 Introducción
5.2 Archivos de texto
- **5.2.1** Creación de un archivo de texto
- **5.2.2** Lectura de un archivo de texto

5.3 Aplicación para el manejo de archivos de texto
- **5.3.1** Diseño y programación del formulario mantCooperativas.java
- **5.3.2** Diseño y programación del formulario mantPersonas.java
- **5.3.3** Diseño y programación del formulario movTaxis.java

5.4 Archivos binarios en Java
- **5.4.1** Creación de un archivo binario en Java

Expectativa

El manejo de archivos de texto o binarios nos remonta a las primeras clases de programación de computadoras, cuando se usaba Pascal o C/C++. Las bases de datos en aquellos tiempos distaban de la complejidad interna y la facilidad de manejo externa que tienen actualmente. Tanto es así que la utilización de archivos de texto o binarios son cosas del pasado o simplemente se usan como un readme.txt o en algunos programas antiguos de Cobol, BBX, Fortran u otros.

Asimismo, algunos textos que manejan poca información como la de configuración de un driver o un programa; la descripción de una ayuda breve, entre otras coberturas.

En este capítulo se abordará someramente el tema de la gestión de archivos de texto y binarios en Java, dada la importancia que podría significar en la creación de un programa que no requiera el uso de bases de datos, pero sí de una persistencia básica de la información.

Después de estudiar este capítulo, el lector será capaz de:

- Describir la estructura básica de un archivo de texto.
- Generar programas para la creación, escritura y lectura de datos en un archivo de texto.
- Describir la estructura básica de un archivo binario.
- Generar programas para la creación, escritura y lectura de datos y objetos en un archivo binario.

5.1 Introducción

Se inicia este capítulo con la muestra del funcionamiento de los archivos de texto y binarios. La idea principal es que el lector cree, escriba y lea archivos de texto y binarios mediante un programa Java utilizando NetBeans como herramienta de desarrollo. También es importante conocer la teoría básica que soporta el uso de este tipo de archivos; a pesar de que este uso es limitado, es muy importante conocer su funcionamiento, dado que en cualquier momento se podría requerir un programa que no necesite una base de datos para gestionar la información que contiene, pero sí la persistencia de datos.

Los archivos de texto fueron muy importantes en el pasado dado que mediante éstos se podía almacenar información que necesitaba un aplicativo de computadora. En estos archivos se guardaban desde datos de configuración hasta información relacionada con una cuenta bancaria, un cliente o una factura. Así que cuando la computadora se apagara, esa información no se perdería sino que se almacenaba en archivos planos y luego podía ser recuperada. Los archivos binarios se utilizaron con fines similares con la diferencia de su formato, que hacía mucho más difícil su lectura o manipulación que la de un archivo de texto.

5.2 Archivos de texto

Un archivo de texto es un conjunto de datos que se almacena en un dispositivo secundario tal como un disco duro, CD, DVD, llave USB, entre otros. La información guardada se agrupa en un conjunto de datos denominado fichero o archivo. En sí, es una cadena de bytes consecutivos cuyo final está representado por un carácter especial denominado EOF (End of File, por sus siglas en inglés).

La responsabilidad de interpretar esas cadenas de bytes corresponde a los programas o aplicaciones adecuados. En caso de archivos de texto, programas como Notepad, EditPad, Writebox, Writer, Simplenote, entre otros, pueden crear y gestionar datos. La **figura 5.1** muestra la estructura de un archivo de texto, donde se puede observar que cada "celda" constituye un byte de cierto tamaño y la conjunción de todos ellos conforma un registro. Al final se muestra el carácter especial EOF.

Figura 5.1 Estructura de un archivo de texto

Tanto los archivos de texto como los binarios pueden realizar operaciones básicas que les permiten crear y gestionar la información para los cuales fueron implementados. Estas operaciones se llevan a cabo a través de programas por los que a su vez se crearon. Tales operaciones básicas son:

a. Creación
b. Apertura
c. Lectura
d. Escritura
e. Recorrido
f. Cierre

5.2.1 Creación de un archivo de texto

Los archivos de texto pueden crearse mediante la clase FileWriter. Esta clase permite tener acceso a un archivo de texto en modo escritura. Sin embargo, la escritura puede ser muy costosa. Por ejemplo, si se desea escribir 10 líneas de texto, se tendrán que realizar 10 accesos al disco, uno por cada línea por escribir. Esto se debe a que el buffer utilizado por la clase es más pequeño. La solución sería utilizar `BufferedWriter`, dado que mediante esta clase podemos escribir un buffer intermedio y luego enviar las 10 líneas completas al archivo de texto, con un solo acceso al disco. La sintaxis básica de `FileWriter` se muestra en la siguiente línea de texto:

```
FileWriter archivo = new FileWriter("C:\\archivoTexto.txt");
```

En la línea anterior se puede ver que se establece la variable archivo como referencia a un archivo físico denominado `archivoTexto.txt` que se encuentra en el disco C:. Si no se encuentra entonces automáticamente lo crea vacío.

Otra clase importante en la generación de archivos de texto es `PrintWriter`, la cual se utiliza para imprimir la representación formateada de objetos en la secuencia de salida de texto. El siguiente fragmento de código muestra esta combinación entre `FileWriter` y `PrintWriter`.

```
FileWriter archivo = null;
archivo = new FileWriter("C:\\archivoTexto.txt");
pw = new PrintWriter(archivo);
```

En el fragmento de código anterior se observa cómo se crea la referencia a un archivo de texto con `FileWriter` y luego `PrintWriter`, se utiliza para manejar la secuencia de salida de texto de ese archivo.

5.2.2 Lectura de un archivo de texto

Es un procedimiento similar al de la escritura de un archivo de texto. En este caso, se utilizan las clases `FileReader` y `BufferedReader`. La diferencia que existe en este contexto es que la clase `FileReader` crea el flujo principal de lectura y `BufferedReader` el flujo intermedio. El

siguiente fragmento de código muestra cómo se declaran estas clases para la lectura de un archivo de texto.

```
archivo = new FileWriter("C:\\archivoTexto.txt");
fr = new FileReader(archivo);
br = new BufferedReader(fr);
```

En las líneas de código anteriores se observa cómo primero se crea la referencia al archivo de texto con `FileWriter`, luego se crea un puntero de tipo `FileReader` que lo prepara para ser leído. Finalmente, con `BufferedReader` se procede a llenar el buffer requerido con la información del archivo de texto a ser desplegado.

Actividades para el lector

Para más información visitar los sitios web:

- http://chuwiki.chuidiang.org/index.php?title=Lectura_y_Escritura_de_Ficheros_en_Java
- https://www.javatpoint.com/java-printwriter-class
- http://www.ecodeup.com/como-escribir-y-leer-archivos-de-texto-plano-en-java/
- https://www.discoduroderoer.es/clases-filereader-y-filewriter-para-ficheros-de-texto-en-java/

Se ha tenido ya un corto acercamiento al manejo de archivos de texto en Java. Ahora se pondrá a prueba ese conocimiento con la creación de un aplicativo sencillo. Sólo se requerirá el uso de vectores y las clases explicadas líneas arriba en este capítulo.

5.3 Aplicación para el manejo de archivos de texto

Se desarrollará una pequeña aplicación de una asociación de cooperativas de taxis. Cada cooperativa se registra en la sociedad y cada taxi pertenece a una sociedad determinada.

1. Abra NetBeans y cree un proyecto de tipo Java Appication (véase figuras 1.5, 1.6 y 1.7 del capítulo 1). El nombre de este proyecto será *Taxis*.

2. El primer paso es crear una serie de paquetes necesarios para guardar en ellos los archivos requeridos para desarrollar este aplicativo. Para generar un paquete Java se debe pulsar con el botón derecho del mouse sobre el nombre del proyecto (en este caso el proyecto se llama *Taxis*) y seleccionar Java Package…, tal como se observa en la **figura 5.2**.

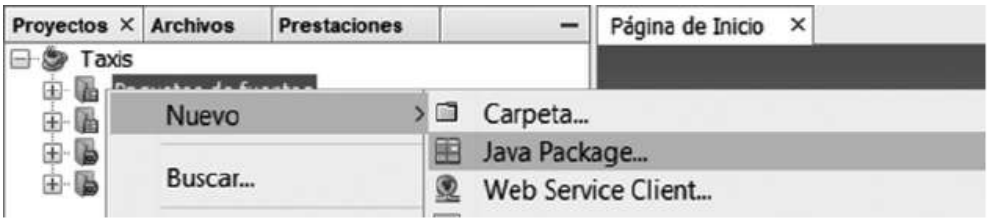


Figura 5.2 Creación de un paquete Java

3. Los paquetes a crear (`Archivos`, `Clases`, `Formularios`, `Imagenes` y `Taxis`) son los que se muestran en la **figura 5.3**.

Figura 5.3 Paquetes Java requeridos

4. Una vez creados los paquetes Java, se debe generar una clase nueva haciendo clic derecho sobre el paquete `Clases` en la opción Nuevo – Java Class, la cual se denominará `Utilitarios`.java. Esta clase servirá para crear los archivos de texto necesarios para el aplicativo, así como para leerlos. El listado 5.1 muestra el código a generar en esta clase.

Listado No. 5.1 Programación de la clase `Utilitarios.java`

```
1.  package Clases;
2.  import java.io.BufferedReader;
3.  import java.io.File;
4.  import java.io.FileReader;
5.  import java.io.FileWriter;
6.  import java.io.IOException;
7.  import java.io.PrintWriter;
8.  import javax.swing.JOptionPane;
9.  import javax.swing.table.DefaultTableModel;
10. public class Utilitarios {
11. public static String ruta =  System.getProperty("user.dir")+"\\src\\Archivos";
```

```
12.
13. public void TablaArchivo(int nColumnas,String nombreArchivo,
14. DefaultTableModel tabla)
15. {
16. File archivo = null;
17. FileReader fr = null;
18. BufferedReader br = null;
19. Object[ ] filas = new Object[nColumnas];
20. try
21.      {
22.      // crea un objeto file, el constructor recibe la
23.      // ruta del archivo del cual quiero saber sus propiedades
24.      archivo = new File (ruta+"\\"+nombreArchivo);
25.      fr = new FileReader(archivo);
26.      br = new BufferedReader(fr);
27.      String linea;
28.      while((linea=br.readLine( ))!=null)    {
29.        filas[0]=linea;
30.        for (int j=1;j<nColumnas;j++)
31.        {
32.          filas[j] = br.readLine( );
33.        }
34.        tabla.addRow(filas);
35.      }
36. }
37. catch (IOException e)
38. {
39.      JOptionPane.showMessageDialog(null, "Agregue un registro para crear
         el archivo");
40. }
41. finally
42. {
43.      try
44.      {
45.        if(null != fr)
46.          fr.close( );
47.      }
48.      catch (IOException e2)
49.      {
```

```
50.         JOptionPane.showMessageDialog(null, e2.getMessage( ));
51.       }
52.    }
53. }
54. public void EscribirEnArchivo(String nombreArchivo, DefaultTableModel ta-
    bla)
55. {
56. FileWriter archivo = null;
57. PrintWriter pw = null;
58. try
59. {
60.       archivo = new FileWriter(Utilitarios.ruta + "\\"+nombreArchivo);
61.       pw = new PrintWriter(archivo);
62.       for (int filas = 0; filas < tabla.getRowCount( ); filas++) {
63.        for (int columnas = 0; columnas < tabla.getColumnCount( ); columnas++) {
64.           pw.println(tabla.getValueAt(filas, columnas));
65.       }
66.    }
67. } catch (IOException ex) {
68.    JOptionPane.showMessageDialog(null, ex.getMessage( ));
69. } finally {
70.       try {
71.         if (null != archivo) {
72.            archivo.close( );
73.         }
74.     } catch (IOException e) {
75.       JOptionPane.showMessageDialog(null, e.getMessage( ));
76.        }
77.      }
78.    }
79. }
```

Comentarios sobre el código del listado 5.1:

Es preciso aclarar algunos conceptos de dicho código:

- De las líneas 2 a la 9 se importan todas las librerías requeridas para el programa. Destacan las librerías `Java.IO.FileReader`, `Java.IO.FileWriter` y `Java.IO.PrintWriter` que ya se han visto someramente en este capítulo. En la línea 8 se importa la librería `Java.Swing.JOptionPane` que servirá para crear ventanas de mensajes en

nuestra aplicación. En la fila 9 se importa la librería `Java.Swing.Table.DefaultModel` que se utilizará para crear un objeto tipo table (con filas y columnas) para almacenar información que luego se enviará a un archivo de texto o se almacenará en él una vez recuperada la información desde el archivo de texto.

- En la línea 11 se creará la variable pública `ruta` que permitirá establecer la ruta física donde se encuentran los archivos de texto.

- En las líneas 13 y 14 se crea el método `TablaArchivo` que servirá para leer la información contenida en un archivo. Para ello recibe como parámetros `nColumnas` para saber cuántas columnas tiene el archivo de texto y así crear un objeto de nombre Filas para luego almacenar en él los valores obtenidos. También recibe el parámetro `nombreArchivo` que servirá para conocer al archivo que se va a leer. Finalmente, el parámetro tabla nos servirá para modelar el archivo en memoria, agregándole filas al objeto `Filas`.

- En las filas 16 a 19 se crean instancias de File, `FileReader` y `BufferReader`, para el manejo de archivos de texto de nuestra aplicación. También se declara la variable `filas` de tipo `Object` que será inicialmente de una sola fila pero con las columnas que se indiquen en nColumnas, que es recibido por parámetro.

- La fila 24 establece la variable `archivo` con la ruta física donde se encuentran nuestros archivos de texto más el nombre del archivo a procesar que se recibe por parámetro.

- De las líneas 28 a 35 se crea un ciclo `while` donde se lee una línea de datos, almacenándolas en el buffer `br` y si ésta es diferente de `null` se carga en la variable cadena denominada linea (la cual se declara en la línea 27). Si la variable linea es llenada desde el buffer de datos br entonces se entra en un ciclo for, delimitado por el número de columnas dado por `nColumnas` y se incrementa de acuerdo con el número de columnas rescatadas del archivo de texto. Así, por cada fila (controlada por el while) se agregan las columnas respectivas según `nColumnas` (y controladas por el `for`) para, finalmente, agregar una fila completa (con todas sus columnas) en la variable tabla (línea 34).

- En la fila 54 se declara el método `EscribirEnArchivo`, el cual tiene como objetivo guardar en un archivo de texto los datos cargados en una tabla de tipo `JTable`. Los parámetros que recibe son: una cadena con el nombre del archivo a procesar y una tabla con los datos a almacenar en el archivo de texto.

- Las filas 56 y 57 declaran las variables `archivo` y `pw` para el manejo de escritura en archivos de texto.

- Las filas 60 a 64 permiten ubicar físicamente el archivo en disco (línea 60), preparar el archivo para ser modificado creando una referencia al mismo (fila 61) y procesar todas

las filas y columnas que están en la variable `tabla`, la cual ha sido pasada por parámetro al procedimiento.

La clase anterior, al ser genérica permitirá leer y escribir archivos de texto. La idea de la modificación y eliminación de datos en el archivo de texto es simple: se cargarán los datos en una tabla JTable y ahí se eliminarán o se modificarán. Una vez que se realice cualquiera de estas operaciones se procederá a sustituir el archivo de texto existente por los datos cargados en el JTable. No se entrará en una codificación más compleja de este tipo de archivos.

A continuación, se procederá a crear un formulario que se denominará Menu. Para ello, se pulsará con el botón derecho del mouse sobre el paquete java denominado Formularios y se seleccionará Nuevo; posteriormente Formulario `JFrame`…, como se muestra en la **figura 5.4**.

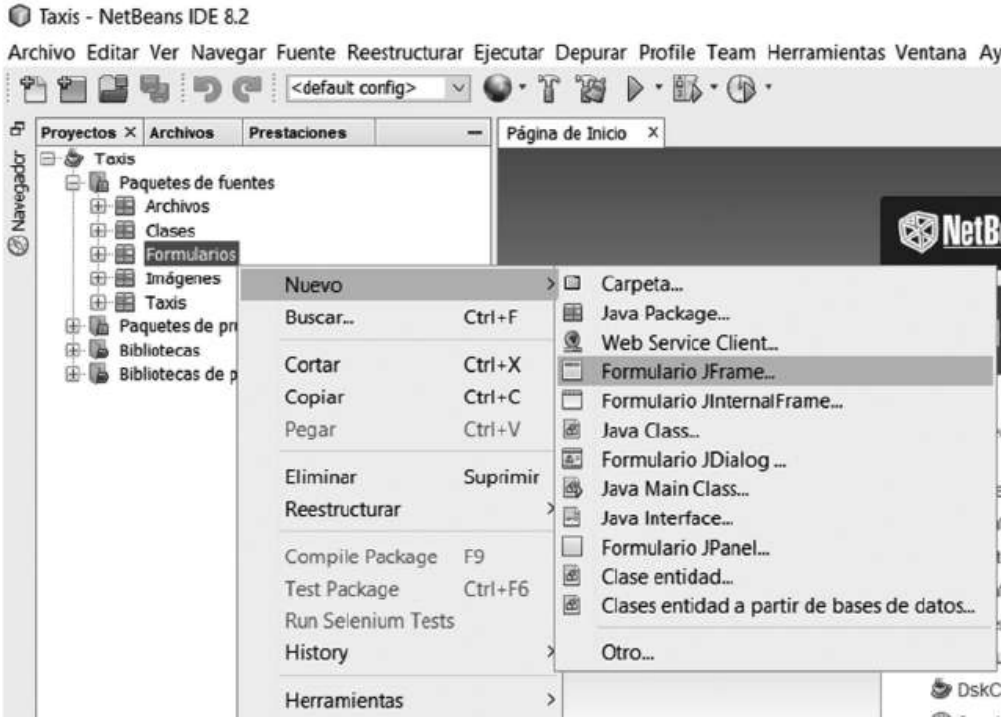


Figura 5.4 Creación de formulario JFrame "Menu" para proyecto Taxis

5. Una vez creado el formulario, habilite la barra de herramientas (Ventana, Herramientas IDE, Paleta o simplemente pulse los botones CTRL+Mayúscula+8). Seleccione ahí la pestaña Menús Swing y arrastre el control Barra de menú, como se observa en la **figura 5.5**.

6. Una vez colocado el componente Barra de menú pulse sobre la opción File y cambie el texto por Mantenimiento. Luego pulse sobre la opción Edit y cámbiela por Movimientos. Hasta este momento se tiene lo que se muestra en la **figura 5.6**.

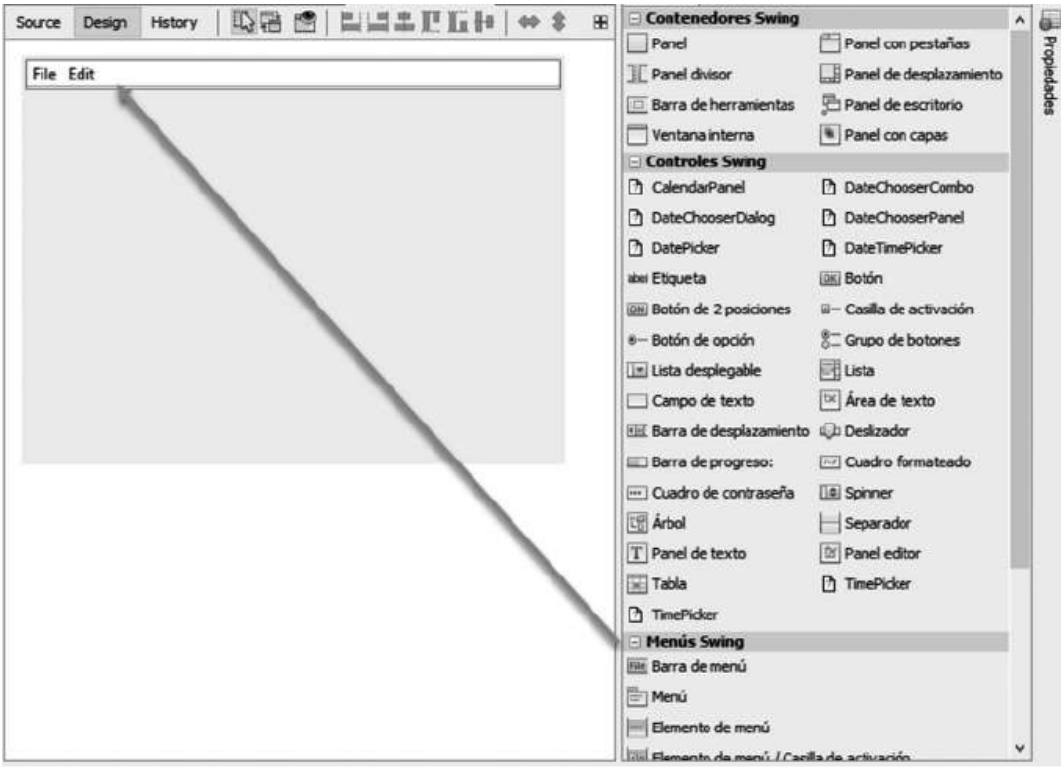


Figura 5.5 Arrastrar un componente Barra de menú al formulario Menú

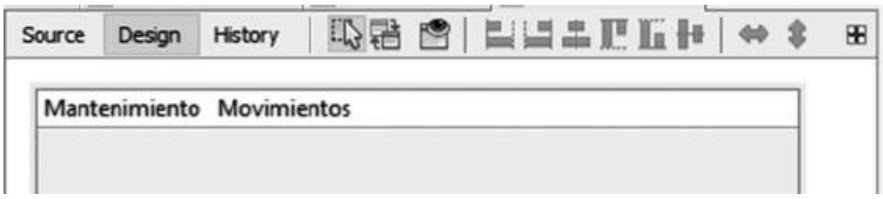


Figura 5.6 Crear las opciones horizontales del menú

7. Ahora se procede a crear los elementos de cada opción del menú, arrastrando desde la barra de herramientas el control Elemento de menú, como se muestra en la **figura 5.7.**

8. Igual que las opciones horizontales del menú, se editan los elementos de menús con clic derecho sobre cada `jMenuItem` y se selecciona la opción Editar texto en el menú de contexto que aparece. También se puede editar pulsando directamente sobre el texto y cambiándolo manualmente. Agregue en el Menú `Mantenimiento 3 jMenuItem`, a los cuales editará el texto por Personas, Taxis, Cooperativas. Por último, en el menú Movimientos agregue un `jMenuItem` al cual le editará el nombre por `Entradas` y `Salidas`.

9. Una vez creadas las opciones de menú, cambie el nombre de variable con un clic derecho en cada opción de menú en la opción Cambiar nombre de variable. Los nombres que le

pondrá son: `mnuPersonas`, `mnuTaxis` y `mnuCooperativas`, para el caso del menú mantenimiento y `mnuEntradaSalida` para el ítem de menú del menú Movimientos.

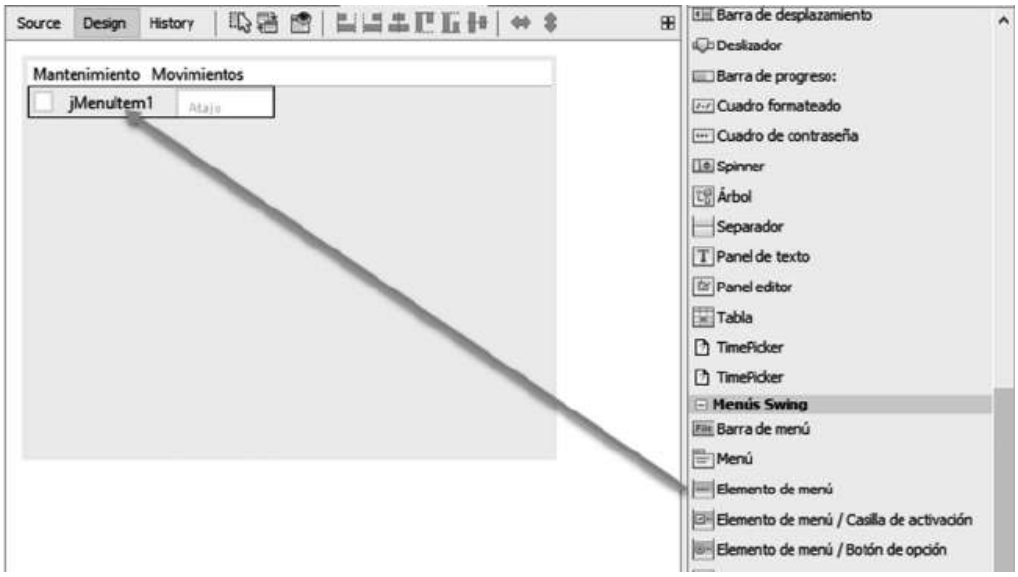


Figura 5.7 Creación de los submenús del menú

10. En el medio del formulario se deberá cargar una etiqueta (label), arrastrándola desde la barra de herramientas y en la propiedad `Icons` de la misma cargar la imagen que se desee. Sin embargo, esta característica es opcional, por lo que no se documenta detalladamente.

11. Hacer clic derecho sobre una sección vacía del `JFrame` y elegir propiedades, localizar la propiedad `title` y establecerla en `Sistema de Control de Taxis`.

El diseño final del menú JFrame se muestra en la **figura 5.8**.

12. Proceda a crear los formularios restantes de la aplicación. Haga clic derecho en el paquete `Formularios` en la opción Nuevo – Formulario JDialog… (Nota: Si no aparece en la lista, elija Otro y en el cuadro de diálogo que se muestra seleccione Categorías, Formulario de interfaz gráfica de swing, en el tipo de archivo que se muestra a la derecha, seleccione la opción Formulario `JDialog`). Establezca los siguientes nombres para los formularios `JDialog` a crear: `mantCooperativas`, `mantPersonas`, `mantTaxis` y `movTaxis`, en total se deben crear 4 formularios `JDialog`.

13. Modifique la propiedad `title` de cada uno de los formularios `JDialog`, para `mantCooperativas` establezca esta propiedad en Gestión de Cooperativas; para `mantPersonas` establezcala en Gestión de Personas, para `mantTaxis` el title será Gestión de Taxis y, por último, para movTaxis se establece el `title` en Movimientos de Taxis.

Figura 5.8 Formulario Menú final

14. Una vez creada la interfaz del menú, se debe programar la funcionalidad de la misma. Para ello, se selecciona la opción Source del diseñador, como se muestra en la **figura 5.9**.

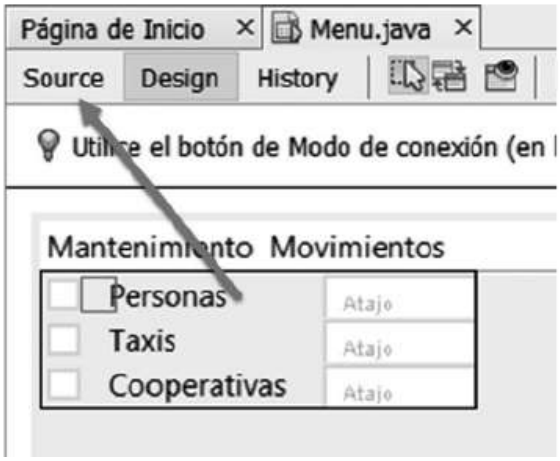


Figura 5.9 Programación de la funcionalidad del menú

15. Una vez seleccionada la opción indicada en la **figura 5.9** aparecerá el editor de código, donde se procederá a crear la funcionalidad del menú. El listado 5.2 muestra el código a programar para el menú.

Listado No. 5.2 Programación de la funcionalidad del menú

```
1.  package Formularios;
2.  public class Menu extends javax.swing.JFrame {
```

```
3.
4.  public Menu( ) { initComponents( ); }
5.  private void mnuPersonasActionPerformed(java.awt.event.ActionEvent evt)
6.  {
7.            mantPersonas frm = new mantPersonas(this, true);
8.            frm.setVisible(true);
9.  }
10. private void mnuTaxisActionPerformed(java.awt.event.ActionEvent evt)
11. {
12.           mantTaxis frm = new mantTaxis(this, true);
13.           frm.setVisible(true);
14. }
15. private void mnuEntradaSalidaActionPerformed(java.awt.event.ActionEvent evt)
16. {
17.           movTaxis frm = new movTaxis(this, true);
18.           frm.setVisible(true);
19. }
20. private void mnuCooperativasActionPerformed(java.awt.event.ActionEvent evt)
21. {
22.           mantCooperativas frm = new mantCooperativas(this, true);
23.           frm.setVisible(true);
24. }
25.    public static void main(String args[ ]) {
26.           java.awt.EventQueue.invokeLater(new Runnable( )
27.           {
28.               public void run( ) {
29.                   new Menu( ).setVisible(true);
30.           } });
31.    }
32. }
```

Comentarios sobre el código del listado 5.2:

A continuación se aclaran algunas funcionalidades del listado:

- De las líneas 5 a la 9 se crea un evento relativo al accionar mnuPersonas. Al hacer clic en el elemento de submenú `Personas` del menú Mantenimiento se creará una instancia del formulario llamada `mantPersonas`, a su vez denominada frm (línea 7, en esta línea para crear la instancia se llama al constructor `mantPersonas(this, true)`, como parámetro se tiene this el cual hace referencia al formulario padre, en este caso el `JFrame Menu`;

por otro lado, el parámetro true se refiere a si el formulario hijo `JDialog` será modal o no modal; el modal establecido en true significa que no se podrá volver al formulario padre (estará bloqueado) hasta que se cierre el formulario hijo, en este caso `mantPersonas`), después de crear la instancia del `JDialog` éste se hará visible mediante la instrucción que se observa en la línea 8.

- Las mismas funcionalidades tienen las líneas de la 10 a la 14, 15 a la 19 y 20 a la 24 para poder visualizar los formularios `mantTaxis`, `movTaxis` y `mantCooperativas`. Estos formularios se diseñarán a continuación.

- Finalmente, la línea 29 es la que permite que el formulario `Menu` se ejecute y se visualice.

5.3.1 Diseño y programación del formulario mantCooperativas.java

Ahora corresponde diseñar y programar el formulario `mantCooperativas` (antes creado). Se iniciará con la determinación de los objetos que contendrá el mismo, los cuales se indican en la **tabla 5.1** y se relacionan con la **figura 5.10**.

Tabla 5.1 Objetos a implementar en el formulario `mantCooperativas`

NÚMERO	DESCRIPCIÓN
1	Todos los componentes que están en la figura 5.10 con este número representan `JLabel`. No tienen un nombre de objeto asignado.
2	El componente que está en la figura 5.10 con este número es un `JTextArea`. Su nombre de variable es `txtDireccion`.
3	Todos los componentes que están en la figura 5.10 con este número son JTextField. Sus nombres de variables son: `txtCedulaJuridica`, `txtNombre` y `txtTelefono`.
4	Todos los componentes que están en la figura 5.10 con este número son `JButton`. Sus nombres de variable son: `btnIncluirTabla`, `btnModificar`, `btnEliminar`, `btnCerrar` y `btnGuardarArchivo`.
5	El componente que está en la figura 5.10 con este número es un `JTable`. Su nombre de variable es tabla. Para modificar los títulos de filas y columnas se debe pulsar sobre el elegido y seleccionar la opción Contenido. Se mostrará una pantalla como en la figura 5.11.

Figura 5.10 Interfaz del formulario `mantCooperativas`.java

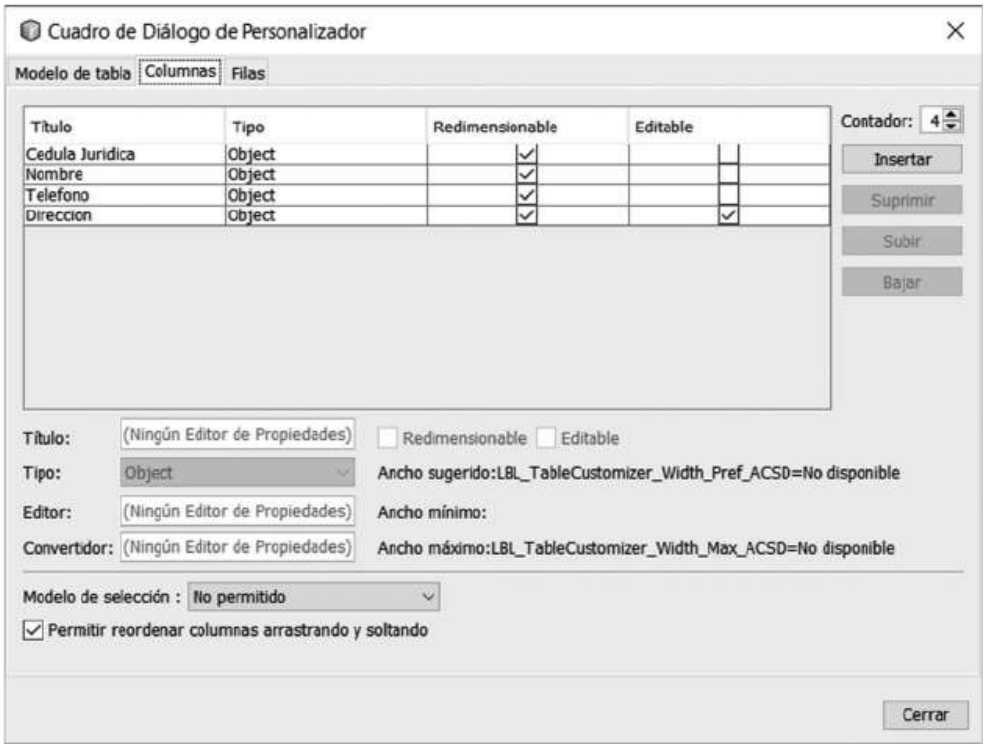


Figura 5.11 Configuración de filas y columnas de un objeto `JTable`

16. El código fuente para el formulario `mantCooperativas` se muestra en el **listado 5.3**.

Listado No. 5.3 Programación de la funcionalidad del formulario `mantCooperativas`

```
1.  package Formularios;
2.  import Clases.Utilitarios;
3.  import javax.swing.JOptionPane;
4.  import javax.swing.table.DefaultTableModel;
5.  /**
6.   * @author su nombre
7.   */
8.  public class mantCooperativas extends javax.swing.JDialog {
9.      static Object[ ] filas = new Object[4];
10.     static DefaultTableModel modeloTabla = new DefaultTableModel( );
11.     private int fila;
12.     Utilitarios util = new Utilitarios( );
13.     public mantCooperativas(java.awt.Frame parent, boolean modal) {
14.        super(parent, modal); initComponents( );
15.        configurarModelo( );
16.        cargarArchivo( );
17.     }
18.     private void cargarArchivo( )
19.     {
20.        util.TablaArchivo(4, "Cooperativas.txt",modeloTabla);
21.        tabla.setModel(modeloTabla);
22.     }
23.     private void  configurarModelo( ){
24.        modeloTabla = new DefaultTableModel( );
25.        modeloTabla.addColumn("No. Cédula");
26.        modeloTabla.addColumn("Nombre");
27.        modeloTabla.addColumn("Teléfono");
28.        modeloTabla.addColumn("Dirección");
29.        tabla.setModel(modeloTabla);
30.    }
31.    @SuppressWarnings("unchecked")
32.    private void cargarTabla( )
33.        {
34.           try
35.           {
36.              filas[0] = txtCedulaJuridica.getText( );
37.              filas[1] = txtNombre.getText( );
```

```
38.              filas[2] = txtTelefono.getText( );
39.              filas[3] = txtDireccion.getText( );
40.           }
41.           catch(Exception ex)
42.           {
43.               JOptionPane.showMessageDialog(rootPane, ex.toString( ));
44.           }
45.        }
46.           private void btnIncluirTablaActionPerformed (java.awt.event.
              ActionEvent evt) {
47.           cargarTabla( );
48.           modeloTabla.addRow(filas);
49.           tabla.setModel(modeloTabla);
50.           btnGuardarArchivo.setEnabled(true);
51.        }
52.           private void btnGuardarArchivoActionPerformed (java.awt.event.
              ActionEvent evt) {
53.           util.EscribirEnArchivo("Cooperativas.txt", modeloTabla);
54.            btnGuardarArchivo.setEnabled(false);
55.        }
56.           private void btnModificarActionPerformed (java.awt.event.
              ActionEvent evt) {
57.            cargarTabla( );
58.           for (int i = 0; i < 4; i++) //para las 4 columnas de la table
59.           {
60.                modeloTabla.setValueAt(filas[i], fila, i);
61.            }
62.            tabla.setModel(modeloTabla);
63.            btnGuardarArchivo.setEnabled(true);
64.        }
65.           private void btnEliminarActionPerformed (java.awt.event.
              ActionEvent evt) {
66.            modeloTabla.removeRow(fila);
67.            tabla.setModel(modeloTabla);
68.            btnGuardarArchivo.setEnabled(true);
69.        }
70.        private void seleccionarRegistro( )
71.        {
72.          txtCedulaJuridica.setText(String.valueOf (tabla.getValueAt(fila, 0)));
73.          txtNombre.setText((String.valueOf (tabla.getValueAt(fila, 1))));
74.          txtTelefono.setText((String.valueOf (tabla.getValueAt(fila, 2))));
75.          txtDireccion.setText((String.valueOf (tabla.getValueAt(fila, 3))));
```

```
76.         }
77.         private void tablaMouseClicked (java.awt.event.MouseEvent evt) {
78.             fila = tabla.rowAtPoint(evt.getPoint( ));
79.             seleccionarRegistro( );
80.         }
81.         private void btnCerrarActionPerformed (java.awt.event.ActionEvent evt) {
82.            this.dispose( );
83.         }
84.         public static void main(String args[ ]) {
85.          java.awt.EventQueue.invokeLater(new Runnable( ) {
86.              public void run( ) {
87.                 mantCooperativas dialog = new mantCooperativas(new javax.
                    swing.JFrame( ), true);
88.                 dialog.addWindowListener(new java.awt.event.WindowAdapter( ) {
89.                 @Override
90.                 public void windowClosing (java.awt.event.WindowEvent e) {
91.                       System.exit(0);
92.                       }
93.                 });
94.                 dialog.setVisible(true);
95.          }
96.         });
97.  } //Fin de la clase mantCooperativas
```

Comentarios sobre el código del listado 5.3:

La explicación de algunas líneas que se incluyen en el listado 5.3 se listan seguidamente:

- La fila 9 declara una variable vector denominada filas que se usa para el almacenamiento de cuatro valores o columnas listados en el `JTable`(como si fueran las 4 columnas que conforman el archivo de texto `Cooperativas.txt`: cédula, nombre, teléfono y dirección).

- En la fila 12 se crea una instancia de la clase `Utilitarios` para utilizar los métodos de lectura y escritura de archivos de texto.

- En las filas 18 a 22 a lo interno del método `cargarArchivo( )`, se hace un llamado al método `TablaArchivo` mediante la instancia `util` de la clase `Utilitarios`, pasándole por parámetro el número 4 (en referencia a las cuatro columnas que requiere el objeto `JTable tabla` y que se cargan con el método `configurarModelo( )`, en las líneas 23 a 30), la cadena `Cooperativas.txt` que referencia al archivo de texto a leer y `modelo-Tabla` donde se cargarán los datos.

- En las líneas 32 a 45 se crea el método cargarTabla que carga en la variable filas (el cual es un vector) los valores de los objetos (cajas de texto) `txtCedula.getText( )`, `txtNombre.getText( )`, `txtTelefono.getText( )` y `txtDireccion.getText( )`, que serán los que posteriormente se cargarán en el objeto tabla del formulario.

- En las líneas 46 a 51 se crea el método `btnIncluirTablaActionPerformed ( )`, que se ejecuta cuando se hace clic en el botón Incluir en Tabla, el cual permite agregar el registro al vector `filas` en el `JTable` tablas del formulario.

- En las filas 52 a 55 se crea el método de guardar (`btnGuardarArchivoActionPerformed ( )`), el cual permite invocar a `EscribirEnArchivo` enviándole los parámetros "`Cooperativa.txt`" es decir el archivo de texto que se sobrescribirá y `modeloTabla`, el componente que contienen los datos.

- De las filas 56 a 64 se implementa el método (`btnModificarActionPerformed ( )`) para modificar los valores que se encuentran almacenados en el `JTable` tabla del formulario con los valores que se encuentran en los objetos `JTextBox` del mismo. El ciclo for carga las 4 columnas del objeto filas que modifica el modelo sobre el cual se basa el JTable tabla con los datos del modelo `modeloTabla`.

- En las filas 65 a 69 se crea el método `btnEliminarTablaActionPerformed ( )`, que permite eliminar una fila en `modeloTabla` y, a su vez, cambia el `JTable tabla`.

- En las filas 70 a 76 se modifican los objetos `JTextBox` del formulario con los valores de una fila seleccionada de `l JTable tabla`. Esto para que los valores de los `textbox` y `textarea` del formulario sean iguales a la fila que el usuario selecciona con un clic en la tabla del formulario.

- En las filas 81 a 83 se permite cerrar el formulario `JDialog` (`mantCooperativas`) para volver al formulario `JFramen` principal Menu.

5.3.2 Diseño y programación del formulario mantPersonas.java

El diseño y programación de este formulario es similar al de `mantCooperativas.java`. Por tanto, sólo se mostrará la interfaz del formulario, el código programado y se explicarán algunas partes de código diferente.

Se diseña un formulario `JDialog` similar al que se muestra en la **figura 5.12**.

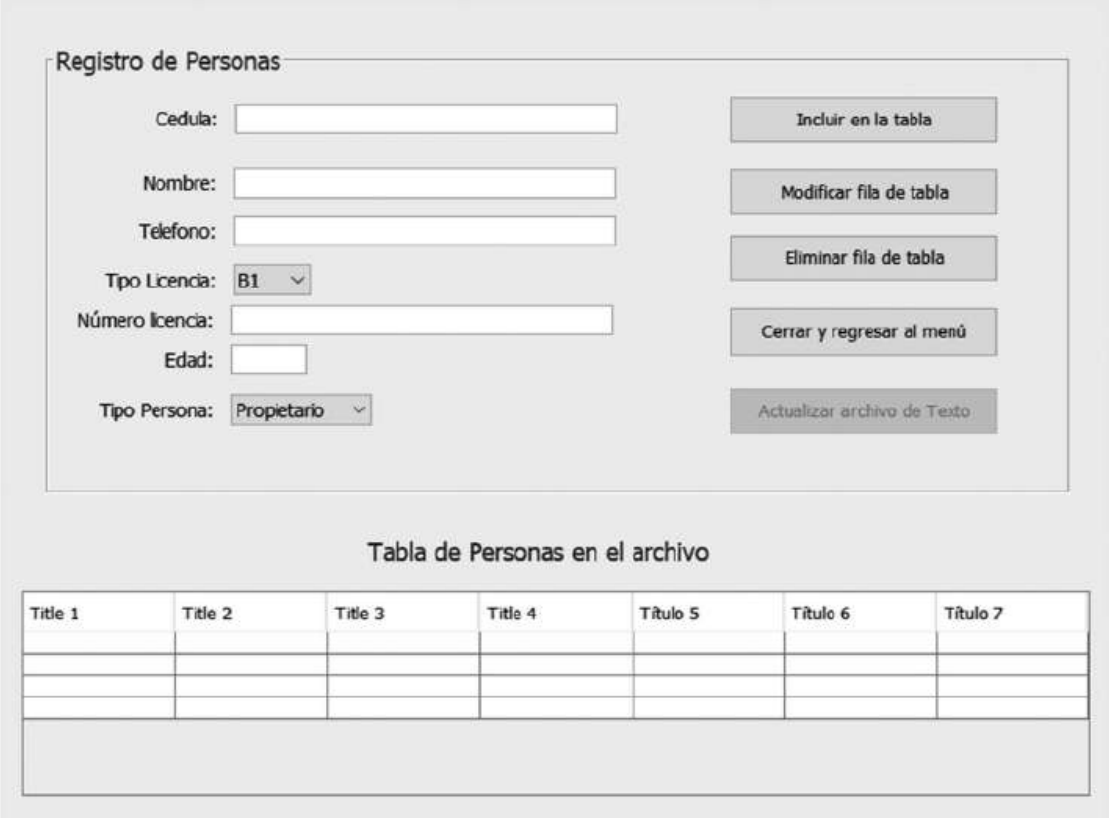


Figura 5.12 Interfaz del formulario `mantPersonas.java`

En la **figura 5.12** cédula, nombre, teléfono, número de licencia y edad son objetos `JTextField` (`textbox`). Los componentes de Tipo Licencia y Tipo Persona son `JComboBox` (lista desplegable). Existen 5 controles de tipo `JButton` y el `JTable` que nos permitirán cargar los datos del archivo de texto para ser visualizados en el formulario.

Es importante destacar que se deben cargar datos estáticos en los dos `JComboBox` del formulario (evite cargarlos desde un archivo de texto). Este procedimiento se hará con la modificación de la propiedad model de cada uno de estos dos componentes. La **figura 5.13** muestra cómo cargar los datos B1, B2, B3, C1, C2 y C3 en la propiedad model del JComboBox de los tipos de licencias. Para el JComboBox de tipos de personas se realiza el mismo procedimiento, cargando los valores estáticos `Propietario` y `Conductor`.

A continuación, el **listado 5.4** muestra el código fuente para el funcionamiento del formulario `mantPersonas`.

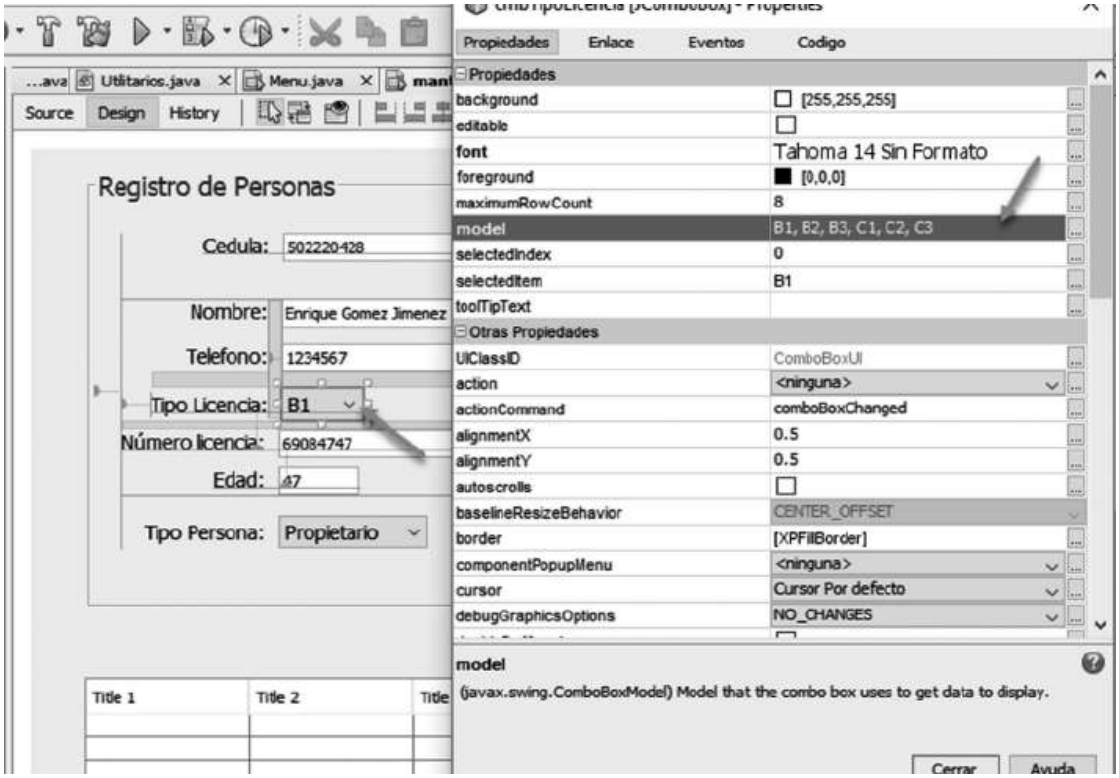


Figura 5.13 Cargar datos estáticos en el `JComboBox` tipo licencia

Listado No. 5.4 Programación de la funcionalidad del formulario `mantPersonas`

```
1.  package Formularios;
2.  import Clases.Utilitarios;
3.  import javax.swing.JOptionPane;
4.  public class mantPersonas extends javax.swing.JDialog{
5.      Object[ ] filas = new Object[7];
6.      javax.swing.table.DefaultTableModel modeloTabla =
        new javax.swing.table.DefaultTableModel( );
7.      Utilitarios util = new Utilitarios( );
8.      private int fila;
9.      private void  configurarModelo( ){
10.         modeloTabla = new javax.swing.table.DefaultTableModel( );
11.         modeloTabla.addColumn("Cédula");
12.         modeloTabla.addColumn("Nombre");
13.         modeloTabla.addColumn("Teléfono");
14.         modeloTabla.addColumn("Tipo Licencia");
15.         modeloTabla.addColumn("# Licencia");
16.         modeloTabla.addColumn("Edad");
```

```
17.          modeloTabla.addColumn(“Tipo”);
18.          tabla.setModel(modeloTabla);
19.      }
20.      private void cargarTabla( )
21.      {
22.          try
23.          {
24.              filas[0] = txtCedula.getText( );
25.              filas[1] = txtNombre.getText( );
26.              filas[2] = txtTelefono.getText( );
27.              filas[3] = cmbTipoLicencia.getSelectedItem( ).toString( );
28.              filas[4] = txtNumeroLicencia.getText( );
29.              filas[5] = txtEdad.getText( );
30.              filas[6] = cmbPropietario.getSelectedItem( ).toString( );
31.          }
32.          catch(Exception ex)
33.          {
34.              JOptionPane.showMessageDialog(rootPane, ex.toString( ));
35.          }
36.      }
37.         public mantPersonas(java.awt.Frame parent, boolean modal) {
38.              super(parent, modal); initComponents( );
39.              configurarModelo( );
40.              cargarArchivo( );
41.         }
42.         @SuppressWarnings(“unchecked”)
43.         private void cargarArchivo( )
44.         {
45.              util.TablaArchivo(7, “Personas.txt”,modeloTabla);
46.              tabla.setModel(modeloTabla);
47.         }
48.         private void btnIncluirTablaActionPerformed (java.awt.event.
            ActionEvent evt) {
49.              int nLic = Integer.parseInt(txtNumeroLicencia.getText( ));
50.              int Edad = Integer.parseInt(txtEdad.getText( ));
51.              cargarTabla( );
52.              modeloTabla.addRow(filas);
53.              tabla.setModel(modeloTabla);
54.              btnGuardarArchivo.setEnabled(true);
55.         }
```

```
56.         private void btnGuardarArchivoActionPerformed (java.awt.event.
            ActionEvent evt) {
57.             util.EscribirEnArchivo("Personas.txt", modeloTabla);
58.             btnGuardarArchivo.setEnabled(false);
59.         }
60.         private void tablaMouseClicked (java.awt.event.MouseEvent evt) {
61.             fila = tabla.rowAtPoint(evt.getPoint( ));
62.             txtCedula.setText(String.valueOf (tabla.getValueAt(fila,0)));
63.             txtNombre.setText((String.valueOf (tabla.getValueAt(fila,1))));
64.             txtTelefono.setText((String.valueOf (tabla.
                getValueAt(fila,2))));
65.             String valor1 = (String) tabla.getValueAt(fila,3);
66.             for(int i=0;i<cmbTipoLicencia.getItemCount( );i++)
67.             {
68.               cmbTipoLicencia.setSelectedIndex(i);
69.             if  (valor1 == null ? cmbTipoLicencia.getSelectedItem( ).
                toString( ) == null :
                valor1.equals (cmbTipoLicencia.getSelectedItem( ).toString())){
70.                   i = cmbTipoLicencia.getItemCount( );  }
71.          }
72.          txtNumeroLicencia.setText (String.valueOf(tabla.
             getValueAt(fila,4)));
73.          txtEdad.setText(String.valueOf(tabla.getValueAt(fila,5)));
74.          String valor2 = (String) tabla.getValueAt(fila,6);
75.          for(int j=0;j<cmbPropietario.getItemCount( );j++){
76.                cmbPropietario.setSelectedIndex(j);
77.                if(valor2.equals (cmbPropietario.getSelectedItem( ).
                   toString( )))  {
78.                   j = cmbPropietario.getItemCount( );   }
79.          }
80.      }
81.      private void btnModificarActionPerformed (java.awt.event.ActionEvent
         evt) {
82.          cargarTabla( );
83.          for (int i=0;i<7;i++){   //para las 7 columnas de la table
84.              modeloTabla.setValueAt (filas[i], fila, i);  }
85.          tabla.setModel(modeloTabla);
86.          btnGuardarArchivo.setEnabled(true);
87.      }
88.      private void btnEliminarActionPerformed (java.awt.event.ActionEvent
          evt) {
```

```
89.          modeloTabla.removeRow(fila);
90.          tabla.setModel(modeloTabla);
91.          btnGuardarArchivo.setEnabled(true);
92.       }
93.       private void btnCerrarActionPerformed (java.awt.event.ActionEvent evt) {
94.             this.dispose( );
95.         }
96.         public static void main(String args[ ]) {
97.          java.awt.EventQueue.invokeLater(new Runnable( ) {
98.              public void run( ) {
99.                 mantPersonas dialog = new mantPersonas(new javax.swing.
                    JFrame( ), true);
100.                dialog.addWindowListener(new
                    java.awt.event.WindowAdapter( ) {
101.                     @Override
102.                     public void windowClosing(java.awt.event.WindowEvent e) {
103.                         System.exit(0);
104.             }
105.              });
106.              dialog.setVisible(true);
107.            }
108.       });
109.    }
110. } //Fin de la clase mantPersonas
```

Comentarios sobre el código del listado 5.4:

Algunas líneas se explican de la siguiente manera:

- En la línea 30 en la posición 6 del objeto `filas` (que en realidad representa la columna 6 de la fila) se almacena el valor que el usuario selecciona del `JComboBox cmbPropietario`.

- En las filas 65 a 71 se realiza un proceso de recorrido del `JComboBox cmbTipoLicencia` buscando el valor dado por valor1, el cual es seleccionado de la columna 3 de la fila respectiva en el `JTable tabla`. Una vez encontrado, queda habilitada dicha opción en el `JComboBox cmbTipoLicencia`. Si por ejemplo, se selecciona la fila 2 y en la misma se encuentra la licencia B3 entonces este valor se buscará en la lista de elementos del `JComboBox cmbTipoLicencia`, recorriéndolo de la posición n hasta la última. Una vez que lo encuentre se detiene, quedando seleccionado ese valor en el `JComboBox`.

- En las filas 75 a la 80 se repite el proceso anterior para seleccionar el tipo de propietario.

Actividades para el lector

En el ejemplo anterior se desarrollaron los formularios mantCooperativas y mantPersonas; sin embargo falta realizar mantTaxis.

- Tome como guía los dos que ya se programaron y realice el mantenimiento del formulario mantTaxis, el cual tendrá un diseño similar al que se muestra en la figura 5.14.

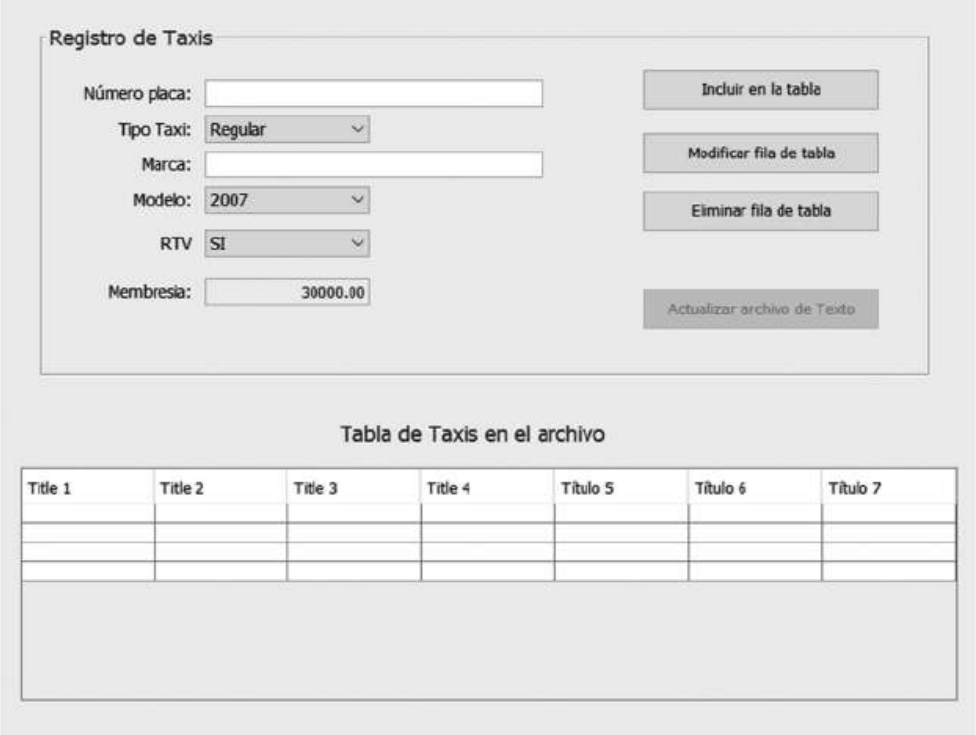


Figura 5.14 Vista de diseño del formulario `JDialog mantTaxis`

5.3.3 Diseño y programación del formulario movTaxis.java

Ahora se diseñará y desarrollará el código requerido para registrar los movimientos de taxis (entradas y salidas). Para este ejercicio se requerirá el uso de un archivo zip denominado DateChoose.jar, el cual se encargará de gestionar datos de tipo fecha. Este archivo se encuentra en http://plugins.netbeans.org/plugin/658/jdatechooser-1-2 también se puede utilizar el control jdatepicker.jar que brinda una funcionalidad similar y se puede descargar desde http://www.codejava.net/java-se/swing/how-to-use-jdatepicker-to-display-calendar-component. El diseño de este formulario se puede visualizar en la **figura 5.15**.

El código a utilizar en este formulario para el mantenimiento del archivo Movimientos.txt es similar a los ejemplos desarrollados anteriormente. El código a implementar en este formulario es el que se muestra en el **listado 5.5**.

Figura 5.15 Diseño del formulario `movTaxis`.java

Listado No. 5.5 Programación de la funcionalidad del formulario movTaxis

```
1.  package Formularios;
2.  import Clases.Utilitarios;
3.  import java.io.*;
4.  import javax.swing.JOptionPane;
5.  public class movTaxis extends javax.swing.JDialog {
6.      Object[ ] filas = new Object[5];
7.      Object[ ] otaxis = new Object[6];
8.      int fila;
9.      javax.swing.table.DefaultTableModel modeloTabla = new
        javax.swing.table.DefaultTableModel( );
10.     public movTaxis(java.awt.Frame parent, boolean modal) {
11.         super(parent, modal);      initComponents( );
12.         configurarModelo( );
13.         cargarTaxis( );
14.         cargarMovimientos( );
15.     }
16.     @SuppressWarnings("unchecked")
17.
18.     private void  configurarModelo( )
```

```
19.     {
20.          modeloTabla.addColumn("# Placa");
21.          modeloTabla.addColumn("Tipo Movim.");
22.          modeloTabla.addColumn("Fecha");
23.          modeloTabla.addColumn("Hora");
24.          modeloTabla.addColumn("Estado");
25.          tabla.setModel(modeloTabla);
26.     }
27.     private void cargarTaxis( )
28.     {
29.          File archivo = null;
30.          FileReader fr = null;
31.          BufferedReader br = null;
32.          try
33.          {
34.                archivo = new File (Utilitarios.ruta + "\\Taxis.txt");
35.                fr = new FileReader(archivo);
36.                br = new BufferedReader(fr);
37.                String linea;
38.                while((linea=br.readLine( ))!=null)
39.                {
40.                   otaxis[0]=linea;
41.                    for (int j=1;j<6;j++)
42.                    {
43.                            otaxis[j] = br.readLine( );
44.                    }
45.                    cmbPlacas.addItem(otaxis[0]);
46.                }
47.           }
48.           catch (Exception e)
49.           {
50.                JOptionPane.showMessageDialog(rootPane, e.getMessage( ));
51.           }
52.           finally
53.           {
54.                try
55.                {
56.                      if(null != fr) {
57.                          fr.close( );  }
```

```
58.                 }
59.                 catch (Exception e2)
60.                 {
61.                 }
62.             }
63.     }
64.     private void cargarMovimientos( )
65.     {
66.         File archivo = null;
67.         FileReader fr = null;
68.         BufferedReader br = null;
69.         try
70.         {
71.         archivo = new File (Utilitarios.ruta + "\\Movimientos.txt");
72.         fr = new FileReader(archivo);
73.         br = new BufferedReader(fr);
74.         String linea;
75.         while((linea=br.readLine( ))!=null)
76.         {
77.                 filas[0]=linea;
78.                 for (int j=1;j<5;j++)
79.                 {
80.                      filas[j] = br.readLine( );
81.                 }
82.                 modeloTabla.addRow(filas);
83.             }
84.             tabla.setModel(modeloTabla);
85.      }
86.      catch (Exception e)
87.      {
88.          JOptionPane.showMessageDialog(rootPane, e.getMessage( ));
89.      }
90.      finally
91.      {
92.          try
93.          {
94.              if(null != fr)
95.                  { fr.close( );  }
96.          }
```

```
97.           catch (Exception e2)
98.           {
99.           }
100.   }
101. }
102. private void cmbPlacasActionPerformed(java.awt.event.ActionEvent evt) {
103. }
104. private void cargarTabla( )
105. {
106.     try
107.     {
108.         filas[0] = cmbPlacas.getSelectedItem( ).toString( );
109.         filas[1] = cmbTipo.getSelectedItem( ).toString( );
110.         filas[2] = fecha.getText( );
111.         filas[3] = txtHora.getText( ) + ":" + txtMinutos.getText( );
112.         filas[4] = cmbEstado.getSelectedItem( ).toString( );
113.     }
114.     catch(Exception ex)
115.     {
116.         JOptionPane.showMessageDialog(rootPane, ex.toString( ));
117.     }
118. }
119. private void btnIncluirTablaActionPerformed (java.awt.event.ActionEvent
     evt) {
120.     String hora = txtHora.getText( )+":"+txtMinutos.getText( );
121.     //  if (cmbTipo.getSelectedItem( )=="Entrada")
122.     //  oTaxis.agMovimientos (cmbPlacas.getSelectedItem( ).toString( ),
        hora,"00:00", fecha.getText( ),cmbEstado.getSelectedItem( ).toString( ));
123.     //  else
124.     // oTaxis.agMovimientos (cmbPlacas.getSelectedItem( ).toString( ),
        "00:00",hora, fecha.getText( ),cmbEstado.getSelectedItem( ).toString( ));
125.     cargarTabla( );
126.     modeloTabla.addRow(filas);
127.     tabla.setModel(modeloTabla);
128.     btnGuardarArchivo.setEnabled(true);
129. }
130. private void btnGuardarArchivoActionPerformed (java.awt.event.ActionEvent
     evt) {
131.     FileWriter archivo = null;
132.     PrintWriter pw = null;
```

```
133.     try {
134.         archivo = new FileWriter(Utilitarios.ruta + "\\Movimientos.txt");
135.         pw = new PrintWriter(archivo);
136.         for (int filas = 0; filas< tabla.getRowCount( ); filas++) {
137.             for (int columnas = 0; columnas < tabla.getColumnCount( );
                 columnas++) {
138.                 pw.println(tabla.getValueAt(filas, columnas));
139.              }
140.         }
141.     } catch (Exception ex) {
142.         JOptionPane.showMessageDialog(rootPane, ex.getMessage( ));
143.     } finally {
144.         try {
145.             if (null != archivo) {
146.                 archivo.close( );
147.             }
148.         } catch (Exception e) {
149.             JOptionPane.showMessageDialog(rootPane, e.getMessage( ));
150.         }
151.     }
152.     btnGuardarArchivo.setEnabled(false);
153. }
154. private void btnModificarActionPerformed (java.awt.event.ActionEvent evt) {
155.     cargarTabla( );
156.     for (int i = 0; i < 5; i++) //para  las 6 columnas de la table
157.     {
158.         modeloTabla.setValueAt(filas[i], fila, i); //cambie el
159.         //modelo por lo que tiene almacenado el vector filas
160.     }
161.     tabla.setModel(modeloTabla);  //actualice la tabla (JTable)
162.                                   //con el modelo
163.     btnGuardarArchivo.setEnabled(true);
164. }
165. private void btnEliminarActionPerformed(java.awt.event.ActionEvent evt) {
166.     modeloTabla.removeRow(fila);
167.     tabla.setModel(modeloTabla);
168.     btnGuardarArchivo.setEnabled(true);
169. }
170. private void seleccionarRegistro( )
171. {
```

```
172.     String valor1 = (String) tabla.getValueAt(fila, 0);
173.     for (int j = 0; j < cmbPlacas.getItemCount( ); j++) {
174.         cmbPlacas.setSelectedIndex(j);
175.         if(valor1.equals(cmbPlacas.getSelectedItem( ).toString( ))){
176.             j = cmbPlacas.getItemCount( );
177.         }
178.         String valor2 = (String) tabla.getValueAt(fila, 1);
179.         for (int i = 0; i < cmbTipo.getItemCount( ); i++) {
180.             cmbTipo.setSelectedIndex(i);
181.             if (valor2 == null ? cmbTipo.getSelectedItem( ).toString( ) ==
                 null
182. :           valor2.equals (cmbTipo.getSelectedItem ( ).toString( ))) {
183.                 i = cmbTipo.getItemCount( );
184.             }
185.         }
186.         fecha.setText((String) tabla.getValueAt(fila, 2));
187.         String cad = (String) tabla.getValueAt(fila, 3);
188.         int posi = cad.indexOf(":");
189.         String hora = cad.substring(0, posi);
190.         String minutos = cad.substring(posi+1,cad.length( ));
191.         txtHora.setText(hora);
192.         txtMinutos.setText(minutos);
193.         String valor3 = (String) tabla.getValueAt(fila, 4);
194.         for (int k = 0; k < cmbEstado.getItemCount( ); k++) {
195.             cmbEstado.setSelectedIndex(k);
196.             if (valor3.equals (cmbEstado.getSelectedItem( ).toString( ))) {
197.                 k = cmbEstado.getItemCount( );
198.             }
199.         }
200.       }
201.     }
202.      private void tablaMouseClicked(java.awt.event.MouseEvent evt) {
203.          fila = tabla.rowAtPoint(evt.getPoint( ));
204.          seleccionarRegistro( );
205. }
206. private void txtHoraKeyPressed(java.awt.event.KeyEvent evt) {
207.     char c;
208.     //capturar el caracter digitado
209.     c=evt.getKeyChar( );
```

```
210.     if(c<'0'|| c>'9')
211.     {
212.        evt.consume( );//ignora el caracter digitado
213.     }
214. }
215. private void btnCerrarActionPerformed(java.awt.event.ActionEvent evt) {
216.     this.dispose( );
217.
218.
219. }
220. private void txtMinutosKeyPressed(java.awt.event.KeyEvent evt) {
221.     char c;
222.     //capturar el caracter digitado
223.     c=evt.getKeyChar( );
224.     if(c<'0'|| c>'9')
225.     {
226.         evt.consume( );//ignora el caracter digitado
227.     }
228. }
229. private void txtMinutosKeyReleased(java.awt.event.KeyEvent evt) {
230.     int numEntero = Integer.parseInt(txtMinutos.getText( ));
231.     if ( numEntero > 60)    {  txtMinutos.setText("59");  }
232. }
233. private void txtHoraKeyReleased(java.awt.event.KeyEvent evt) {
234.     int numEntero = Integer.parseInt(txtHora.getText( ));
235.     if ( numEntero > 12) { txtHora.setText("12");  }
236. }
237. public static void main(String args[ ]) {
238.     java.awt.EventQueue.invokeLater(new Runnable( ) {
239.         public void run( ) {
240.           mantPersonas dialog = new movTaxis(new
              javax.swing.JFrame( ), true);
241.           dialog.addWindowListener(new java.awt.event.WindowAdapter( ) {
242.          @Override
243.             public void windowClosing(java.awt.event.WindowEvent e) {
244.                     System.exit(0);
245.                          }
246.          });
247.          dialog.setVisible(true);
248.        }
```

```
249.     });
250.   }
251. } //Fin de la clase movTaxis
```

La ejecución de este formulario se puede observar en la **figura 5.16.**

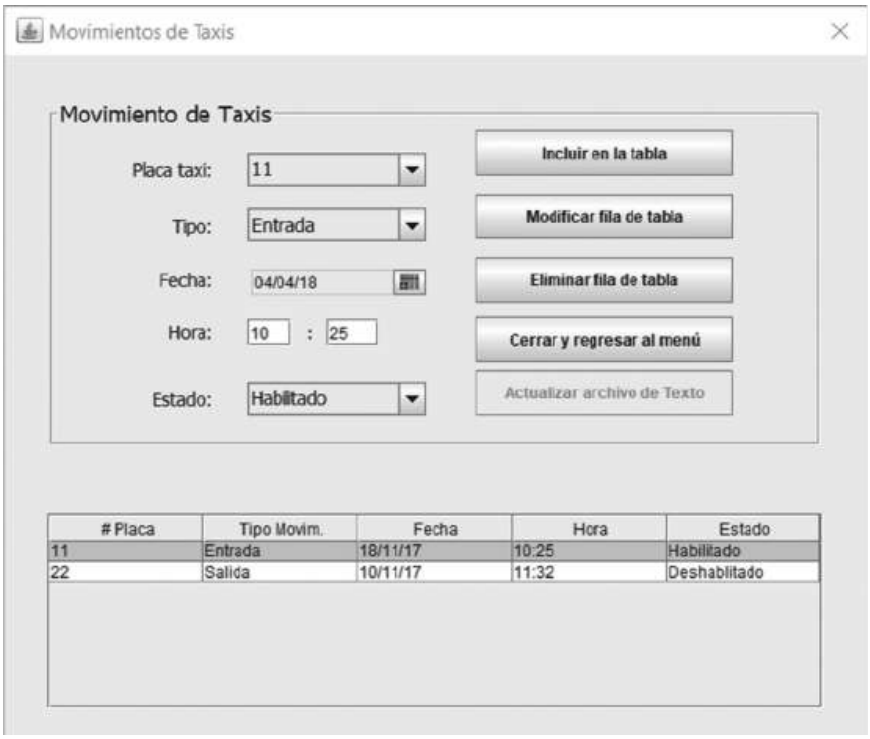


Figura 5.16 Ejecución del formulario `movTaxis.java`

5.4 Archivos binarios en Java

Un archivo binario o de datos está formado por una secuencia de bytes, codificada y agrupada en 8 dígitos binarios que son comunes, estos ficheros pueden almacenar tipos primitivos como `int`, `float`, `double`, `char`, entre otros, pero también es posible almacenar en ellos objetos.

Esta clase de ficheros puede almacenar cualquier cosa que no sea caracteres de texto; por ejemplo, es posible almacenar números, imágenes, sonidos, archivos de programas fuentes, compilados, entre otros. Es decir, se puede almacenar cualquier tipo de información la cual en el momento de escribirse en el archivo se guardará como dígitos binarios. A diferencia de los archivos de texto, si se intenta abrir un fichero binario con el editor de texto lo que se podrá visualizar son símbolos no legibles, ya que los archivos binarios suelen contener información con formatos no estandarizados que sólo pueden ser leídos por el software que los creó.

Como se mencionó en la sección anterior referente a los archivos de texto, también en los archivos binarios es posible realizar operaciones básicas que les permiten crear y gestionar la información para los cuales fueron implementados. Estas operaciones se llevan a cabo mediante los programas que los crearon. Estas operaciones básicas para un archivo binario son:

a. Creación
b. Apertura
c. Lectura
d. Escritura
e. Recorrido
f. Cierre

5.4.1 Creación de un archivo binario en Java

Se crea como si se fuera a instanciar una clase cualquiera, en este caso `DataOutputStream`, a la cual en su constructor se transfiere un objeto de la clase `FileInputStream`, en cuyo constructor se envía la ruta y el nombre de archivo. Enseguida se muestra un ejemplo:

```
String ruta = System.getProperty("user.dir") + "\\src\\Archivo\\
Personas.dat";
DataOutputStream crea = new DataOutputStream(new FileOutputStream
(ruta));
```

Se puede observar en las líneas anteriores qué ruta posee un directorio válido en el sistema de archivos del sistema operativo donde se almacenará el archivo que se especifica.

Por último, si el archivo que se desea crear es para escribir objetos de una clase en particular, en lugar de `DataOutputStream` se usará la clase `ObjectOutputStream` en la creación del archivo.

Escritura en un archivo binario en Java

Para escribir en un archivo binario se debe definir el tipo de dato de cada columna o elemento que se guardará. Por ejemplo, si se desea almacenar los meses del año representados por números del 1 al 12, se escribirá un entero en el archivo, pero si lo que se quiere almacenar es el promedio de ventas diario de un supermercado, se debe escribir en el fichero un float o double, dependiendo del volumen de ventas.

Java, a través del paquete Java.io, ofrece una serie de clases que permiten y facilitan el uso de este tipo de archivos. Para la escritura en ficheros binarios se tiene la clase `DataOutputStream`, la cual proporciona métodos `(writeTipo( )`, donde Tipo es el dato que se quiere escribir, por ejemplo `writeFloat)` para datos primitivos; estos métodos indican el tipo de dato

a ser escrito en el archivo (se escriben bytes), es decir, si se quiere escribir un entero se usará la función `writeInt(numAescribir)`.

Para crear un objeto de la clase `DataOutputStream` se le debe pasar por parámetro la clase `FileOutputStream`, a continuación se muestra un ejemplo:

```
DataOutputStream escribir = new DataOutputStream(new
FileOutputStream ("D:\\nom_fich_binario.dat"));
```

En la sección anterior se mencionó que la línea de código anterior crea el archivo, pero si éste ya existe abre el archivo para lectura y escritura.

A parte de escribir tipos de datos primitivos en un archivo binario, también se puede escribir objetos; para esto, en lugar de usar la clase `DataOutputStream`, se utiliza `ObjectOutputStream` de la siguiente manera:

```
ObjectOutputStream escribir = new ObjectOutputStream(new
FileOutputStream ("D:\\nom_fich_binario.dat"));
```

Mediante esta clase se pueden escribir objetos en el archivo usando los métodos `writeObject(objAescribir)` o con el método `writeUnshared(objAescribir)`. Por ejemplo, `escribir.writeObject(persona)`; donde persona es un objeto de la clase Persona.

Lectura de un archivo binario en Java

Para leer un archivo binario se debe conocer muy bien la estructura interna del mismo, es decir, se debe saber cómo se han escrito los datos. Por ejemplo, si hay datos de tipo double, enteros, char, por otro lado hay que conocer el orden en el que fueron escritos. Si se desconoce esta información existe otra alternativa, la cual es leerla byte a byte.

Java, a través de del paquete Java.io, ofrece una serie de clases que permiten y facilitan el uso de este tipo de archivos. Para la lectura de ficheros binarios se tiene la clase `DataInputStream`, en la cual se debe indicar el tipo de dato a ser leído. La clase proporciona métodos `readTipo( )` donde Tipo es el nombre del tipo de dato primitivo a ser leído.

Se deben leer los datos en el mismo orden en que fueron escritos, por ejemplo, un número flotante, un entero, una cadena.

Para crear un objeto de la clase `DataInputStream`, se le debe pasar por parámetro la clase `FileInputStream`, a continuación se muestra un ejemplo:

```
DataInputStream leer = new DataInputStream(new FileInputStream (“D:\\
nom_fich_binario.dat”));
```

La línea anterior abre el archivo `nom_fich_binario.dat` para lectura.

Si lo que se escribió en el archivo binario no fueron datos primitivos sino objetos de alguna clase, se usará la clase `ObjectInputStream` y su método `readObject( )` para leer cada uno de ellos.

Aplicación para el manejo de archivo binario

Se creará el código necesario para el mantenimiento del archivo binario `Personas.dat`. Solamente se generará la interfaz y la codificación para el mantenimiento de personas. Ya se tiene en el proyecto la implementación del manejo de archivos de texto para ello. Ahora se creará una segunda opción para demostrar el uso de archivos binarios.

Inicialmente se presentará en el listado 5.6 un ejemplo sencillo donde se escriben uno por uno los atributos de Persona en el archivo, dependiendo del tipo de dato de cada atributo. Para la lectura ocurre lo mismo, se debe leer en el mismo orden en que se escribió y el mismo tipo de datos. Cree un proyecto nuevo y llámelo `ArchivoBinario1` luego, cree una Java Main Class y nómbrela `ArchivoBinario1`, escriba el siguiente código.

Listado No. 5.6 Ejemplo de archivos binarios, lectura y escritura de datos individuales

```
1.  import java.io.*;
2.  public class ArchivoBinario1 {
3.      public static void main(String[ ] args) {
4.        String ruta = System.getProperty(“user.dir”) + “\\src\\Persona.dat”;
5.        try (DataOutputStream dos = new DataOutputStream(new FileOutput
          Stream(ruta));) {
6.        //Se escribe la cédula, un entero
7.        dos.writeInt(5315123);
8.        //Se escribe una cadena, el nombre
9.        dos.writeUTF(“Fernando”);
10.       //Se escribe un número, el teléfono
11.       dos.writeInt(88287548);
12.       // Se escribe una cadena, tipo de licencia
13.       dos.writeUTF(“B1”);
14.       // Se escribe un número, número de licencia
15.       dos.writeInt(264512);
16.       // Se escribe una cadena, tipo de persona
```

```
17.         dos.writeUTF("Conductor");
18.
19.         //Leer datos e imprimirlos por pantalla
20.         DataInputStream lect = new DataInputStream(new FileInputStream(ruta));
21.         System.out.println("Datos de la Persona");
22.         System.out.println(lect.readInt( )); //Se lee la cédula y se imprime
23.         System.out.println(lect.readUTF( )); //Se lee el nombre y se imprime
24.         System.out.println(lect.readInt( )); //Se lee el teléfono y se imprime
25.        System.out.println(lect.readUTF( )); //Se lee tipo de licencia y se imprime
26.         System.out.println(lect.readInt( )); //Se lee el núm licencia y se imprime
27.         System.out.println(lect.readUTF( ); //Se lee el tipo persona y se imprime
28.         } catch (IOException e) {
29.                   System.out.println("Error E/S");
30.           }
31.      }
32.  }
```

Enseguida se mostrará un ejemplo más completo en el que se escribirán y leerán objetos de tipo `Persona` en el archivo binario `Personas.dat`.

1. Cree un nuevo proyecto denominado `ArchivoBinario2`.

2. Cree los siguientes paquetes: Archivo, Formulario, Clases.

3. Haga clic derecho sobre el paquete `Clases`, elija la opción `Nuevo`, `Java Class`. Establezca por nombre `Persona`. Haga que la clase Persona implemente la interfaz `Serializable` (`implements java.io.Serializable`).

4. La clase `Persona` debe llevar el código que se muestra en el listado 5.7.

Listado No. 5.7 Programar la clase Persona

```
1.  package Clases;
2.  public class Persona implements java.io.Serializable{
3.      private int cedula;
4.      private String nombre;
5.      private int telefono;
6.      private String tipoLicencia;
7.      private int numLicencia;
8.      private int edad;
9.      private String tipoPersona;
```

```
10.             public Persona(int cedula, String nombre, int telefono, String
                tipoLicencia, int numLicencia, int edad, String tipoPersona) {
11.               this.cedula = cedula;
12.               this.nombre = nombre;
13.               this.telefono = telefono;
14.               this.tipoLicencia = tipoLicencia;
15.               this.numLicencia = numLicencia;
16.               this.edad = edad;
17.               this.tipoPersona = tipoPersona;
18. }
19. public Persona( ) {
20.     this.cedula = 0;
21.     this.nombre = "";
22.     this.telefono = 0;
23.     this.tipoLicencia = "";
24.     this.numLicencia = 0;
25.     this.edad = 0;
26.     this.tipoPersona = "";
27. }
28. public int getCedula( ) {
29.     return cedula;
30. }
31. public void setCedula(int cedula) {
32.     this.cedula = cedula;
33. }
34. public String getNombre( ) {
35.     return nombre;
36. }
37. public void setNombre(String nombre) {
38.     this.nombre = nombre;
39. }
40. public int getTelefono( ) {
41.     return telefono;
42. }
43. public void setTelefono(int telefono) {
44.     this.telefono = telefono;
45. }
46. public String getTipoLicencia( ) {
47.     return tipoLicencia;
```

```
48.     }
49.     public void setTipoLicencia(String tipoLicencia) {
50.     this.tipoLicencia = tipoLicencia;
51. }
52. public int getNumLicencia( ) {
53.     return numLicencia;
54. }
55. public void setNumLicencia(int numLicencia) {
56.     this.numLicencia = numLicencia;
57. }
58. public int getEdad( ) {
59.     return edad;
60. }
61. public void setEdad(int edad) {
62.     this.edad = edad;
63. }
64. public String getTipoPersona( ) {
65.     return tipoPersona;
66. }
67. public void setTipoPersona(String tipoPersona) {
68.     this.tipoPersona = tipoPersona;
69.     }
70. }
```

5. Haga clic derecho sobre el paquete Clases en la opción Nuevo, elija Java Class, ponga por nombre `MiObjectOutputStream`.

6. En el listado 5.8 se muestra el código fuente necesario para esta clase.

Listado No. 5.8 Programación de la clase `MiObjectOutputStream.java`

```
1.  package Clases;
2.  import java.io.IOException;
3.  import java.io.ObjectOutputStream;
4.  import java.io.OutputStream;
5.  //Redefinición de la clase ObjectOuputStream para que no escriba una cabecera
    al inicio del Stream de datos.
6.  public class MiObjectOutputStream extends ObjectOutputStream
7.      public MiObjectOutputStream(OutputStream out) throws IOException
8.      {
9.         super(out);
```

```
10.      }
11.      protected MiObjectOutputStream( ) throws IOException, SecurityException
12.      {
13.         super( );
14.      }
15.      //Redefinir método de escribir la cabecera para que no haga nada.
16.      protected void writeStreamHeader( ) throws IOException
17.      {
18.      }
19.  }
```

La clase anterior se crea con el objetivo de eliminar unas cabeceras que el archivo genera cuando se agregan datos a un archivo ya existente.

7. Haga clic derecho sobre el paquete Clases en la opción Nuevo, elegir Java Class y póngale por nombre Utilitarios.

8. En el listado 5.9 se muestra el código fuente necesario para esta clase.

Listado No. 5.9 Programar la clase `Utilitarios.java`

```
1.  package Clases;
2.  import java.io.*;
3.  import javax.swing.JOptionPane;
4.  import javax.swing.table.DefaultTableModel;
5.  public class Utilitarios {
6.      public static String ruta = System.getProperty("user.dir") + "\\src\\
        Archivo\\Personas.dat";
7.      public void TablaArchivo(int nColumnas, DefaultTableModel tabla)
        throws IOException {
8.          Object[ ] filas = new Object[nColumnas];
9.          Persona p1;
10.         try {
11.            ObjectInputStream lect = new ObjectInputStream(new
               FileInputStream(ruta));
12.            // Se lee el primer objeto
13.            Object aux = lect.readObject( );
14.            while (aux != null) {
15.               if (aux instanceof Persona) {
16.                  p1 = (Persona) aux;
17.                  filas[0] = p1.getCedula( );
18.                  filas[1] = p1.getNombre( );
```

```
19.                 filas[2] = p1.getTelefono( );
20.                 filas[3] = p1.getTipoLicencia( );
21.                 filas[4] = p1.getNumLicencia( );
22.                 filas[5] = p1.getEdad( );
23.                 filas[6] = p1.getTipoPersona( );
24.                 tabla.addRow(filas);
25.              }
26.              aux = lect.readObject( );
27.           }
28.           lect.close( );
29.     } catch (ClassNotFoundException | FileNotFoundException | EOFException ex) {
30.     }
31.  }
32.  public void crearArchivo(Persona p1){
33.     try {
34.           ObjectOutputStream crea = new ObjectOutputStream(new FileOutputS
              tream(ruta));
35.           crea.writeObject(p1);
36.           crea.close( );
37.       } catch (FileNotFoundException ex) {
38.       } catch (IOException ex) {
39.       }
40. }
41. public void EscribirEnArchivo(Persona p1) {
42.     try {
43.        MiObjectOutputStream escri = new MiObjectOutputStream(new
          FileOutputStream(ruta, true));
44.        escri.writeUnshared(p1);
45.        escri.close( );
46.     } catch (IOException ex) {
47.        JOptionPane.showMessageDialog(null, ex.getMessage( ));
48.     }
49.   }
50. }
```

Comentarios sobre el código del listado 5.9:

Es posible explicar algunas líneas de la siguiente manera:

- En la línea 6 se establece la ruta donde se creará, leerá o actualizará el archivo binario, ahí también, aparte de su ruta, se establece el nombre del fichero.

- En la línea 7 y hasta la línea 31 se muestra el método `TablaArchivo(int nColumnas, DefaultTableModel tabla)`, el cual recibe por parámetro la cantidad de columnas de la tabla, es decir, la cantidad de atributos de un objeto Persona, también se recibe un parámetro llamado tabla de tipo `DeafultTableModel` en el cual se agregarán como filas de una matriz todos los objetos que se encuentren almacenados en el archivo, mientras éste se recorre secuencialmente para luego mostrarlo al usuario por medio de `JTable` de la interfaz gráfica.

- En las líneas 32 a 40 se tiene el método `crearArchivo(Persona p1)`, que genera el archivo binario y guarda en el mismo el primer objeto tipo persona.

- Finalmente, de la línea 41 a 49 se tiene el método `EscribirEnArchivo(Persona p1)`, el cual escribe un objeto tipo Persona en el archivo al final del mismo, justo después del último objeto escrito. Es importante recalcar que en la línea 43 se agrega un parámetro más al constructor de la clase `FileOutputStream(ruta, true)`, este parámetro es true y su fin es permitir la adición de registros al final del archivo sin sobrescribirlo.

- Como una buena práctica después de usar el archivo es recomendable cerrarlo con el método `close( )`.

9. Haga clic derecho sobre el paquete Formulario, eleja la opción Nuevo, luego Formulario `JFrame`. Establezca por nombre mantPersonas.

10. Luego se crea la interfaz gráfica de mantPersonas para gestionar el archivo Personas.dat; la misma se muestra en la **figura 5.17** y su diseño es similar a la creada en la **figura 5.12** aunque con algunos cambios en los botones del formulario.

Figura 5.17 Creación de la interfaz gráfica `mantPersonas`

En el listado 5.10 se mostrará el código fuente del formulario mantPersonas.

Listado No. 5.10 Programación de mantPersonas con archivos binarios

```
1.  package Formulario;
2.  import Clases.Persona;
3.  import Clases.Utilitarios;
4.  import java.io.IOException;
5.  import java.util.logging.Level;
6.  import java.util.logging.Logger;
7.  import javax.swing.JOptionPane;
8.  public class mantPersonas extends javax.swing.JFrame {
9.      Persona person = new Persona( );
10.     Object[ ] filas = new Object[7];
11.     javax.swing.table.DefaultTableModel modeloTabla = new javax.swing.
        table.DefaultTableModel( );
12.     Utilitarios util = new Utilitarios( );
13.     private int fila;
14.     private void configurarModelo( ) {
15.         modeloTabla.addColumn("Cedula");
16.         modeloTabla.addColumn("Nombre");
17.         modeloTabla.addColumn("Telefono");
18.         modeloTabla.addColumn("Tipo Licencia");
19.         modeloTabla.addColumn("# Licencia");
20.         modeloTabla.addColumn("Edad");
21.         modeloTabla.addColumn("Tipo");
22.         tabla.setModel(modeloTabla);
23.     }
24.     private void limpiarForm( ) {
25.         txtCedula.setText("");
26.         txtNombre.setText("");
27.         txtTelefono.setText("");
28.         cmbTipoLicencia.setSelectedIndex(0);
29.         cmbPropietario.setSelectedIndex(0);
30.         txtNumeroLicencia.setText("");
31.         txtEdad.setText("");
32.     }
```

```
33. private void cargarTabla( ) {
34.     try {
35.         filas[0] = txtCedula.getText( );
36.         filas[1] = txtNombre.getText( );
37.         filas[2] = txtTelefono.getText( );
38.         filas[3] = cmbTipoLicencia.getSelectedItem( ).toString( );
39.         filas[4] = txtNumeroLicencia.getText( );
40.         filas[5] = txtEdad.getText( );
41.         filas[6] = cmbPropietario.getSelectedItem( ).toString( );
42.     } catch (Exception ex) {
43.         JOptionPane.showMessageDialog(rootPane, ex.toString( ));
44.     }
45. }
46. public mantPersonas( ) {
47.     initComponents( );
48.     configurarModelo( );
49.     cargarArchivo( );
50. }
51. @SuppressWarnings(“unchecked”)
52. Generated Code
53. private void cargarArchivo( ) {
54.     try {
55.         util.TablaArchivo(7, modeloTabla);
56.     } catch (IOException ex) {
57.         JOptionPane.showMessageDialog(rootPane, ex.toString( ));
58.     }
59.     tabla.setModel(modeloTabla);
60. }
61. private void btnIncluirTablaActionPerformed(java.awt.event.ActionEvent evt) {
62.     Persona p1 = new Persona( );
63.     cargarTabla( );
64.     modeloTabla.addRow(filas);
65.     tabla.setModel(modeloTabla);
66.     p1.setCedula(Integer.parseInt(txtCedula.getText( )));
67.     p1.setNombre(txtNombre.getText( ));
68.     p1.setTelefono(Integer.parseInt(txtTelefono.getText( )));
```

```
69.      p1.setTipoLicencia(cmbTipoLicencia.getSelectedItem( ).toString( ));
70.      p1.setNumLicencia(Integer.parseInt(txtNumeroLicencia.getText( )));
71.      p1.setEdad(Integer.parseInt(txtEdad.getText( )));
72.      p1.setTipoPersona(cmbPropietario.getSelectedItem( ).toString( ));
73.      if (modeloTabla.getRowCount( ) == 1) {
74.          util.crearArchivo(p1);
75.      } else {
76.          util.EscribirEnArchivo(p1);
77.      }
78.      limpiarForm( );
79.  }
80.  private void tablaMouseClicked(java.awt.event.MouseEvent evt) {
81.      fila = tabla.rowAtPoint(evt.getPoint( ));
82.      txtCedula.setText(String.valueOf(tabla.getValueAt(fila, 0)));
83.      txtNombre.setText((String.valueOf(tabla.getValueAt(fila, 1))));
84.      txtTelefono.setText((String.valueOf(tabla.getValueAt(fila, 2))));
85.      String valor1 = (String) tabla.getValueAt(fila, 3);
86.      for (int i = 0; i < cmbTipoLicencia.getItemCount( ); i++) {
87.          cmbTipoLicencia.setSelectedIndex(i);
88.          if (valor1 == null ? cmbTipoLicencia.getSelectedItem( ).toString()
             == null
89.          : valor1.equals(cmbTipoLicencia.getSelectedItem( ).toString( ))) {
90.             i = cmbTipoLicencia.getItemCount( );
91.          }
92.  }
93.      txtNumeroLicencia.setText(String.valueOf(tabla.getValueAt(fila, 4)));
94.      txtEdad.setText(String.valueOf(tabla.getValueAt(fila, 5)));
95.      String valor2 = (String) tabla.getValueAt(fila, 6);
96.      for (int j = 0; j < cmbPropietario.getItemCount( ); j++) {
97.          cmbPropietario.setSelectedIndex(j);
98.          if (valor2.equals(cmbPropietario.getSelectedItem( ).toString( ))) {
99.             j = cmbPropietario.getItemCount( );
100.         }
101.     }
102. }
103. private void btnModificarActionPerformed(java.awt.event.ActionEvent evt) {
```

```
104.     Persona p1 = new Persona( );
105.     cargarTabla( );
106.     for (int i = 0; i < 7; i++) //par alas 7 columnas de la table
107.  {
108.         modeloTabla.setValueAt(filas[i], fila, i); //cambie en el modelo por
             lo que tiene
109.  }
110.     tabla.setModel(modeloTabla);  //actualice la tabla (JTable) con el modelo
111.     for (int x = 0; x < modeloTabla.getRowCount( ); x++) {
112.         p1.setCedula(Integer.parseInt(modeloTabla.getValueAt(x, 0).toString( )));
113.         p1.setNombre(modeloTabla.getValueAt(x, 1).toString( ));
114.         p1.setTelefono(Integer.parseInt(modeloTabla.getValueAt(x,
             2).toString( )));
115.         p1.setTipoLicencia(modeloTabla.getValueAt(x, 3).toString( ));
116.         p1.setNumLicencia(Integer.parseInt(modeloTabla.getValueAt
             (x, 4).toString( )));
117.         p1.setEdad(Integer.parseInt(modeloTabla.getValueAt
             (x, 5).toString( )));
118.         p1.setTipoPersona(modeloTabla.getValueAt(x, 6).toString( ));
119.         if (x == 0) {
120.            util.crearArchivo(p1);
121.         } else {
122.            util.EscribirEnArchivo(p1);
123.         }
124.         limpiarForm( );
125.     }
126. }
127. private void btnEliminarActionPerformed(java.awt.event.ActionEvent evt) {
128.     Persona p1 = new Persona( );
129.     int opcion = JOptionPane.showConfirmDialog(this, "¿Desea eliminar el
         registro?");
130.     if (opcion == 0) {
131.         modeloTabla.removeRow(fila);
132.         tabla.setModel(modeloTabla);
133.         for (int x = 0; x < modeloTabla.getRowCount( ); x++) {
134.            p1.setCedula(Integer.parseInt(modeloTabla.getValueAt
               (x, 0).toString( )));
```

```
135.            p1.setNombre(modeloTabla.getValueAt(x, 1).toString( ));
136.            p1.setTelefono(Integer.parseInt(modeloTabla.getValueAt
                (x, 2).toString( )));
137.            p1.setTipoLicencia(modeloTabla.getValueAt(x, 3).toString( ));
138.            p1.setNumLicencia(Integer.parseInt(modeloTabla.getValueAt(x,
                4).toString( )));
139.            p1.setEdad(Integer.parseInt(modeloTabla.getValueAt(x,
                5).toString( )));
140.            p1.setTipoPersona(modeloTabla.getValueAt(x, 6).toString( ));
141.            if (x == 0) {
142.               util.crearArchivo(p1);
143.            } else {
144.               util.EscribirEnArchivo(p1);
145.            }
146.         }
147.         limpiarForm( );
148.     }
149. }
150. public static void main(String args[ ]) {
151.     java.awt.EventQueue.invokeLater(new Runnable( ) {
152.        @Override
153.        public void run( ) {
154.           new mantPersonas( ).setVisible(true);
155.        }
156.  });
157.   }
158.   // Variables declaration - do not modify
159. }
```

En la **figura 5.18** se muestra una corrida típica del formulario mantPersonas con el uso de archivos binarios.

Figura 5.18 Corrida típica de mantPersonas

Con el listado anterior se concluye el ejemplo práctico básico acerca del uso de archivos binarios con programación orientada a objetos; esta herramienta es estimulante para investigar aún más sobre el tema.

Actividades para el lector

En el ejemplo anterior sólo se desarrolló el formulario mantPersonas para completar el proyecto de Taxis:

Desarrollar el resto de formularios que se programaron en el ejemplo práctico de archivos de texto, entre ellos: mantCooperativas, mantTaxis, movTaxis, con esto el lector pondrá a prueba sus conocimientos y desarrollará más habilidades en la programación orientada a objetos con almacenamientos en archivos binarios.

Resumen

En este capítulo se desarrolló someramente el tema de archivos secuenciales de texto y archivos secuenciales binarios en Java. Con los primeros se exploró el concepto y uso de este tipo de archivos, así como la lógica y sintaxis esencial para crear, abrir, recorrer, leer y escribir en un archivo tal. Se concluyó con un ejemplo muy completo que muestra detalladamente un uso práctico de estos archivos.

En lo referente a los archivos binarios el abordaje fue similar: se hizo un recorrido por el concepto, la creación, apertura, lectura y escritura en el archivo; sin embargo, en esta sección el ejemplo práctico se trabajó con la escritura en el archivo tanto de datos primitivos, como con la escritura de objetos de una clase en el mismo como si fueran registros. Básicamente, se trataron los siguientes temas:

- Concepto de archivo de texto
- Usos de los archivos de texto
- Definición de archivos binarios y sus usos
- Diferencia entre archivos de texto y binarios
- Creación y escritura en archivos de texto
- Recorrido y lectura de archivos de texto
- Creación y escritura en archivos binarios
- Recorrido y lectura de archivos binarios

Evidencia

a) Realizar una investigación acerca del uso de archivos de acceso aleatorio (RandomAccessFile), ya que los ejemplos planteados en este capítulo fueron desarrollados con archivos de acceso secuencial. Determinar el concepto, diferencias respecto de los de acceso secuencial, sintaxis, creación, apertura, lectura y escritura.

b) Pensar en qué tipos de problemas pueden ser resueltos mediante el uso de archivos como forma de almacenamiento permanente.

c) Plantear un problema cercano que amerite ser resuelto mediante el uso de archivos ya sea binarios o de texto, e intentar resolverlo con programación de archivos en Java.

d) Recordar si se ha interactuado o visto en ejecución algún software en que se hayan usado archivos binarios o de texto como solución para la persistencia de datos.

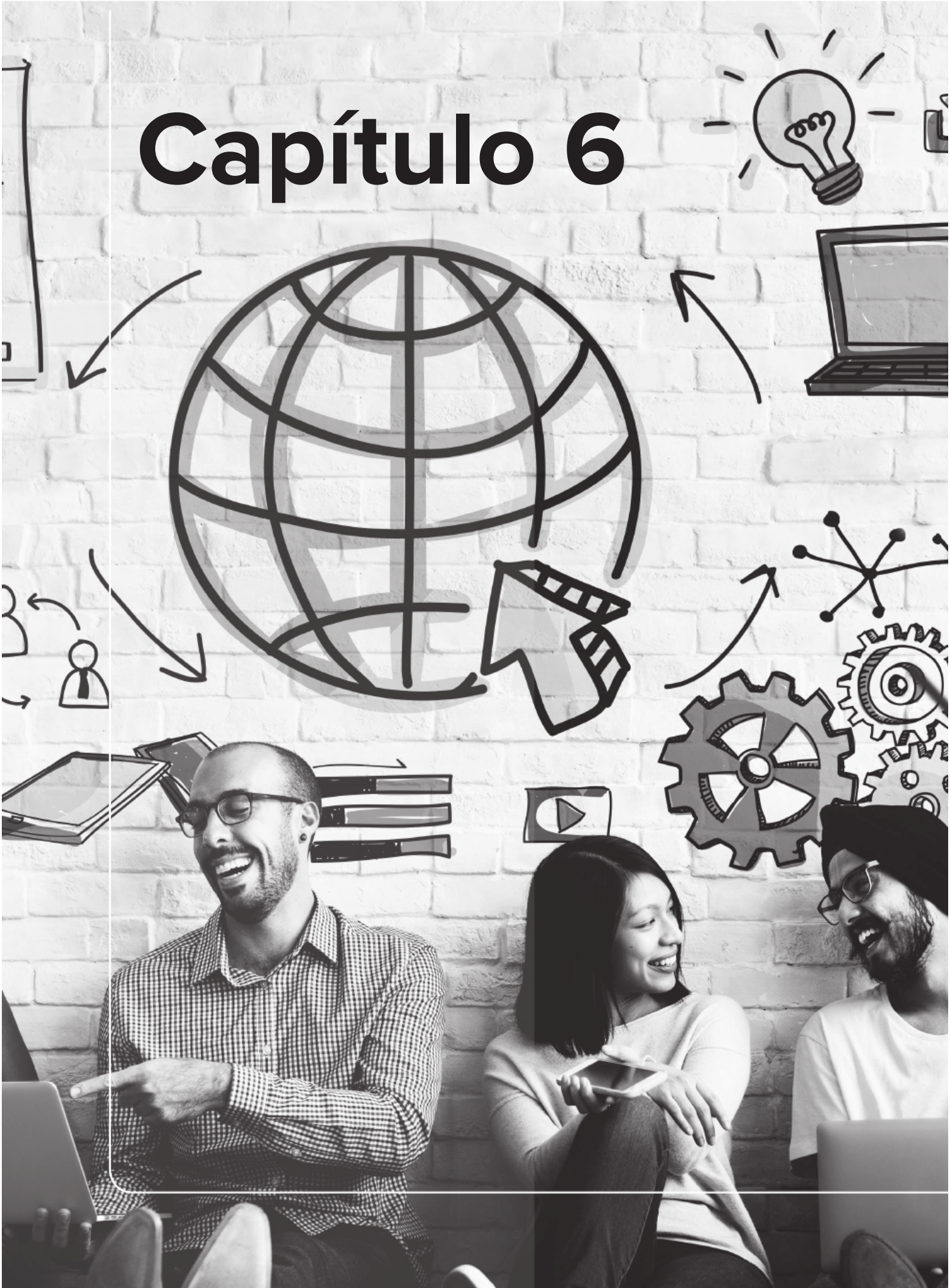
☑ Preguntas de autoevaluación

1. ¿Qué es un archivo de texto?
2. Definir archivo binario
3. Explicar la diferencia entre un archivo de texto y otro binario
4. ¿Cuál es clase que permite crear archivos de texto en Java?
5. Crear la sentencia necesaria para un archivo de texto llamado agenda en la siguiente dirección: "C:\\agenda.txt".
6. ¿Mediante cuál clase se puede crear un archivo binario que permita el almacenamiento de objetos en Java?
7. Crear la línea de código necesaria para un archivo binario llamado agenda en la siguiente dirección: "C:\\agenda.dat", que permita el almacenamiento de objetos

☑ Respuestas a las preguntas de autoevaluación

1. Un archivo de texto es un conjunto de datos que se almacena en un dispositivo secundario tal como un disco duro, CD, DVD, llave USB, entre otros. También se puede decir que es una cadena de bytes consecutivos cuyo final está representado por un carácter especial denominado EOF. Como su nombre lo dice, se almacena texto.
2. Un archivo binario está formado por una secuencia de bytes, codificada y agrupada en 8 dígitos binarios que son comunes; estos ficheros pueden almacenar datos de tipos primitivos como int, float, double, char, entre otros, así como objetos de una clase. Este tipo de archivo puede almacenar cualquier información que no sea texto.
3. La diferencia es que un archivo binario puede almacenar cualquier cosa que no sea caracteres de texto, por ejemplo, es posible almacenar números, imágenes, sonidos, archivos de programas, fuentes, compilados, entre otros. Entonces, se puede decir que es posible guardar cualquier tipo de información, la cual en el momento de escribirse en el archivo se almacenará como dígitos binarios. A diferencia de los archivos de texto, si se intenta abrir un fichero binario con un editor de texto lo que se podrá visualizar son símbolos no legibles, ya que los archivos binarios suelen contener información con formatos no estandarizados que sólo pueden ser leídos por el software que los creó.
4. `FileWriter`
5. `FileWriter agenda = new FileWriter ("C:\\agenda.txt");`
6. `ObjectInputStream` y debe de recibir como parámetro la clase `FileInputStream`.
7. `ObjectInputStream agenda = new ObjectInputStream(new FileInputStream("C:\\agenda.txt")););`

Capítulo 6



Gestión de base de datos MySQL con Java mediante NetBeans

Reflexione y responda
las siguientes preguntas

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

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

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

Contenido

6.1 Introducción
6.2 Instalación de MySQL
- **6.2.1** Instalación de front-end dbforgemysqlfree
- **6.2.2** Creación de base de datos con dbforgemysqlfree
- **6.2.3** Introducción a SQL

6.3 Gestión de datos en MySQL con NetBeans
- **6.3.1** Arquitectura JDBC
- **6.3.2** Conectividad a una base de datos

Expectativa

Las bases de datos representan actualmente el repositorio de información más importante para los negocios, el gobierno y las personas. Existen muchos tipos de información que pueden ser utilizados, de los que destacan comerciales, financieras, militares, personales, crediticias, entre otras. Por ende, una buena gestión de datos representa una tarea sumamente crítica para cualquier empresa, entidad de gobierno o persona particular. El correcto y eficiente manejo de la información puede significar inclusive la quiebra de una empresa, la pérdida de un negocio, el mejoramiento de las actividades de una entidad, entre otros beneficios o perjuicios.

Después de estudiar este capítulo, el lector será capaz de:

- Comprender la arquitectura que se maneja en Java para la gestión de bases de datos con el uso de un motor de datos MySQL.
- Desarrollar aplicaciones que involucren la gestión de bases de datos MySQL con Java como lenguaje de desarrollo.

6.1 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, actualmente se hace por medios electrónicos. No es posible imaginarse un mundo sin los cajeros automáticos, sin las consultas electrónicas de estados de cuentas o sin las transferencias electrónicas de fondos. Un entorno global 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.

Existen muchos sistemas administradores de bases de datos, entre los cuales destacan aquellos que son propietarios (como Oracle® o SQL Server de Microsoft®) o los que son Open Source (como MySQL, MariaDB, PostgreSQL, entre otros). Todos ellos han mejorado con el tiempo ofreciendo seguridad, rapidez, alto desempeño, y demás factores técnicos de gran beneficio para el mundo y las empresas.

Los lenguajes de programación también deben poner su aporte mediante la gestión a realizar sobre dichos datos. La gestión de datos, por ende, es un asunto crucial para los diseñadores de bases, quienes muchas veces se enfrascan en una lucha agresiva para ser los mejores.

6.2 Instalación de MySQL

Se iniciará este capítulo con la instalación de MySQL, versión 5.7.20, el cual se podrá descargar desde https://dev.mysql.com/downloads/mysql/.

MySQL es un software tipificado como Open Source y orientado para su uso general en una variedad importante de aplicaciones de gestión de datos. Se considera no destinado para su uso en aplicaciones riesgosas, entre las que se incluyen las que representan un riesgo de lesiones personales. En caso de que se utilice en estos escenarios, el desarrollador se responsabiliza de tomar las precauciones si existen las pruebas de fallos, copias de seguridad, redundancia, entre otras que se incluyen en el sitio de Oracle®, responsable del diseño y desarrollo de la herramienta.

Sin mayores preámbulos, se inicia la instalación del producto.

1. Se descarga el producto desde https://dev.mysql.com/downloads/mysql/. A continuación se busca la opción que indica Windows (x86 32 & 64-bit), MSI Installer (mysql-installer-community-5.7.20.msi y se descarga. La **figura 6.1** muestra el sitio web de descarga de esta base de datos.

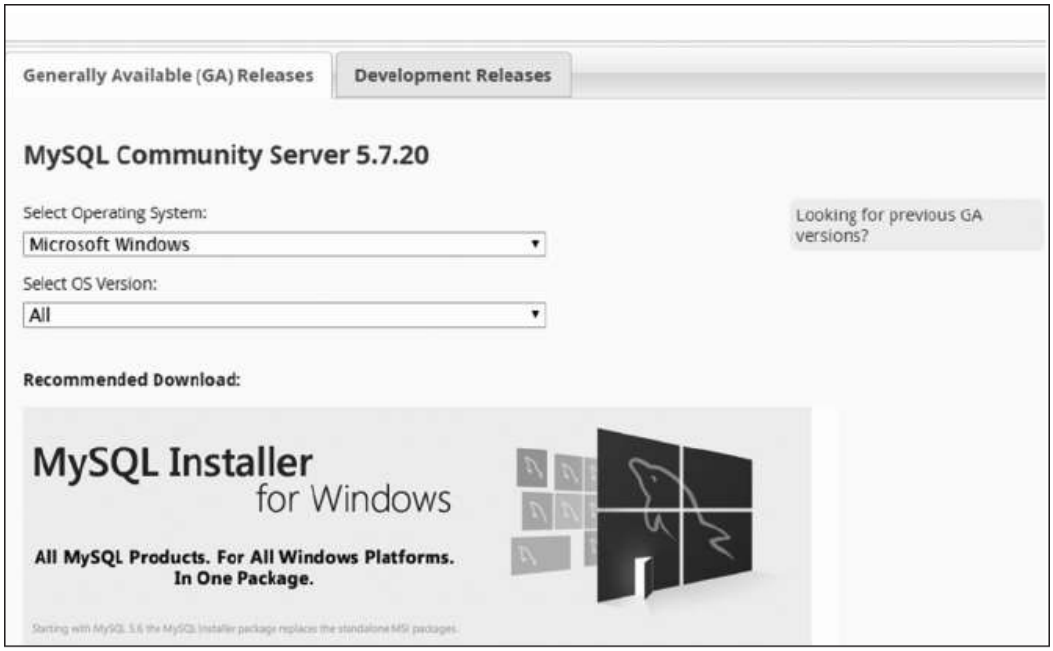


Figura 6.1 Sitio web para descarga de MySQL

2. Se instala MySQL pulsando dos veces con el puntero del mouse sobre el archivo mysql-installer-community-5.7.20.0.msi el cual se descargó del sitio web mostrado en la **figura 6.1**. Una pantalla similar a la **figura 6.2** se muestra enseguida.

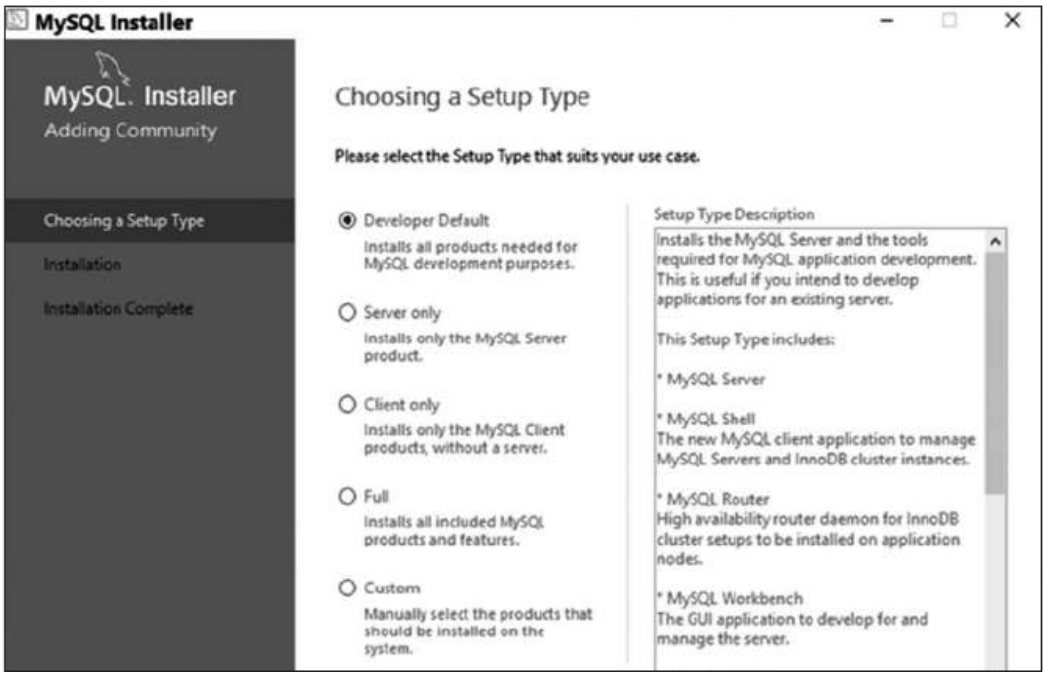


Figura 6.2 Instalando MySQL

3. En la tercera pantalla se podrían mostrar algunos errores detectados en nuestra computadora. Sin embargo, no se consideran y se pulsa la tecla Next (siguiente), como aparece en la **figura 6.3**.

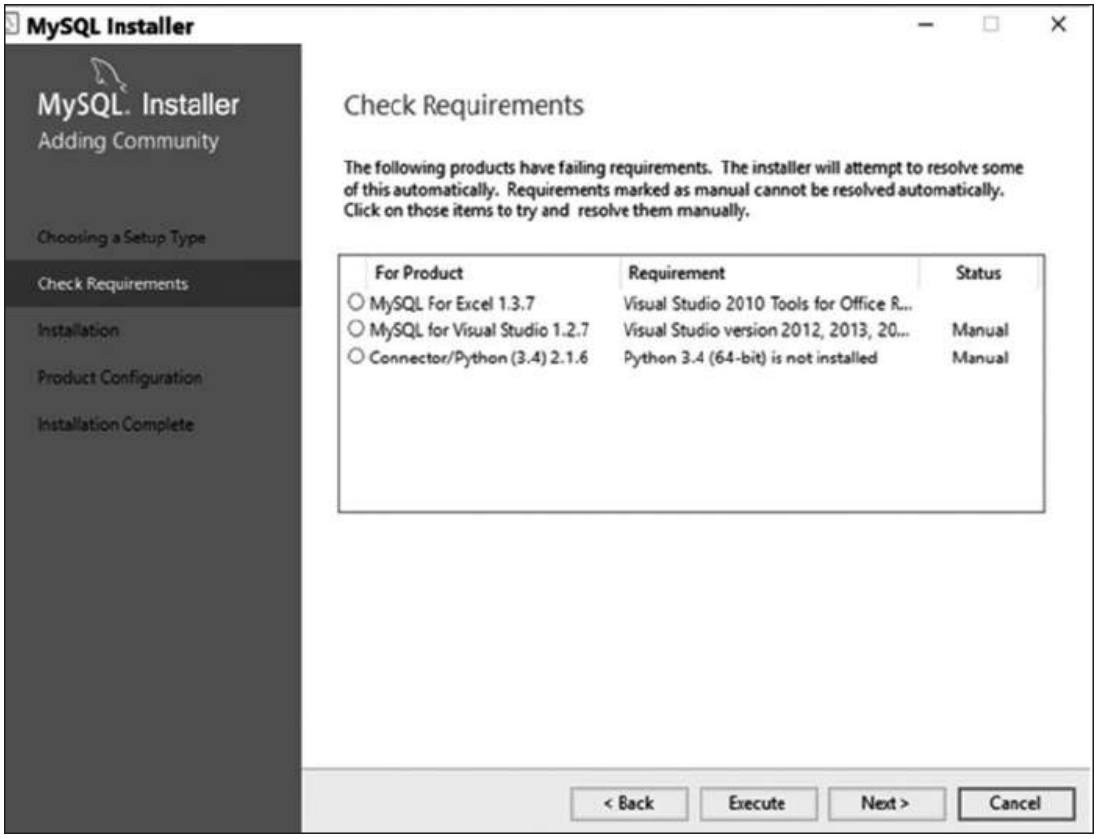


Figura 6.3 Muestra de problemas detectados

4. En la pantalla que aparece se deben aplicar todas las instalaciones ofrecidas. Pulsar el botón Execute, como se muestra en la **figura 6.4**, después hacer clic en Next (siguiente).

5. La pantalla que se presenta a continuación se denomina `Product Configuration`. Es informativa acerca de lo que se realizará en la instalación; por tanto, se debe seleccionar el botón Next (siguiente) para continuar.

6. Seguidamente se presentará la opción Type and Networking. Seleccionar la opción Standalone MySQL Server/Classic MySQL Server Replication.

7. En la pantalla siguiente se mostrará la opción de configurar el tipo de servidor. Seleccionar Development Machine y los valores que tiene por defecto. Las opciones se muestran en la **figura 6.5**.

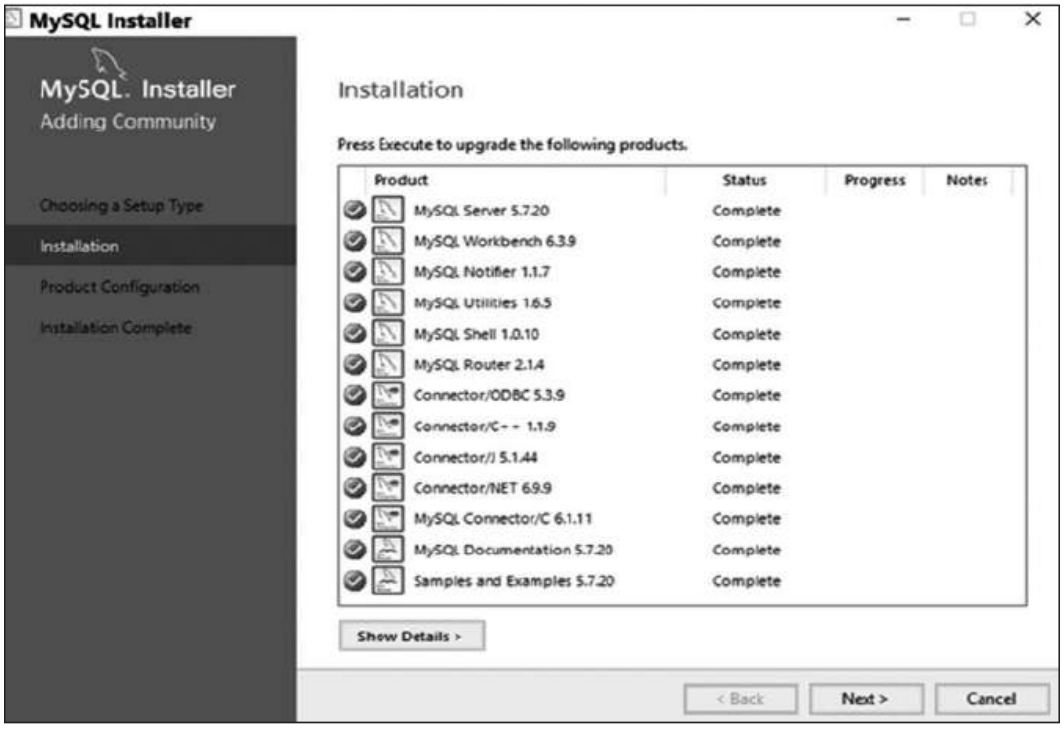


Figura 6.4 Instalación de las opciones de MySQL requeridas

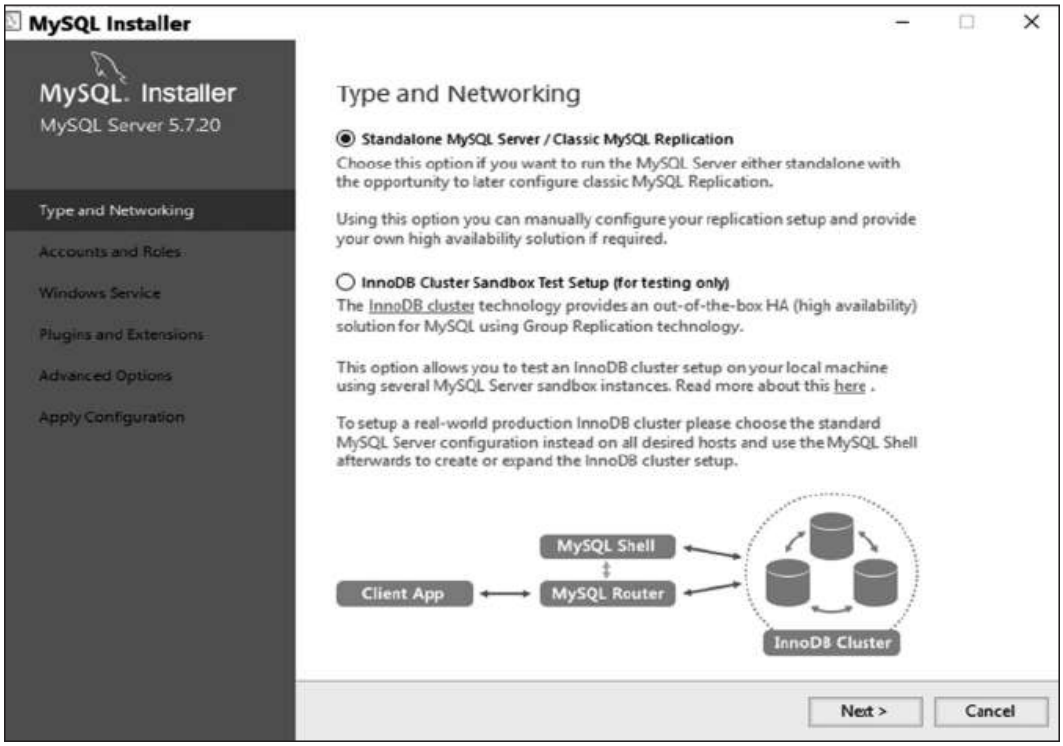


Figura 6.5 Configuración del tipo de servidor a utilizar

8. A continuación, se presentará una pantalla para configurar la cuenta y los roles para el servidor. En este caso se escribe una contraseña no adecuada pero fácil para utilizar en ambientes académicos (password = 12345). Luego se pulsa el botón Next (siguiente), como se muestra en la **figura 6.6**.

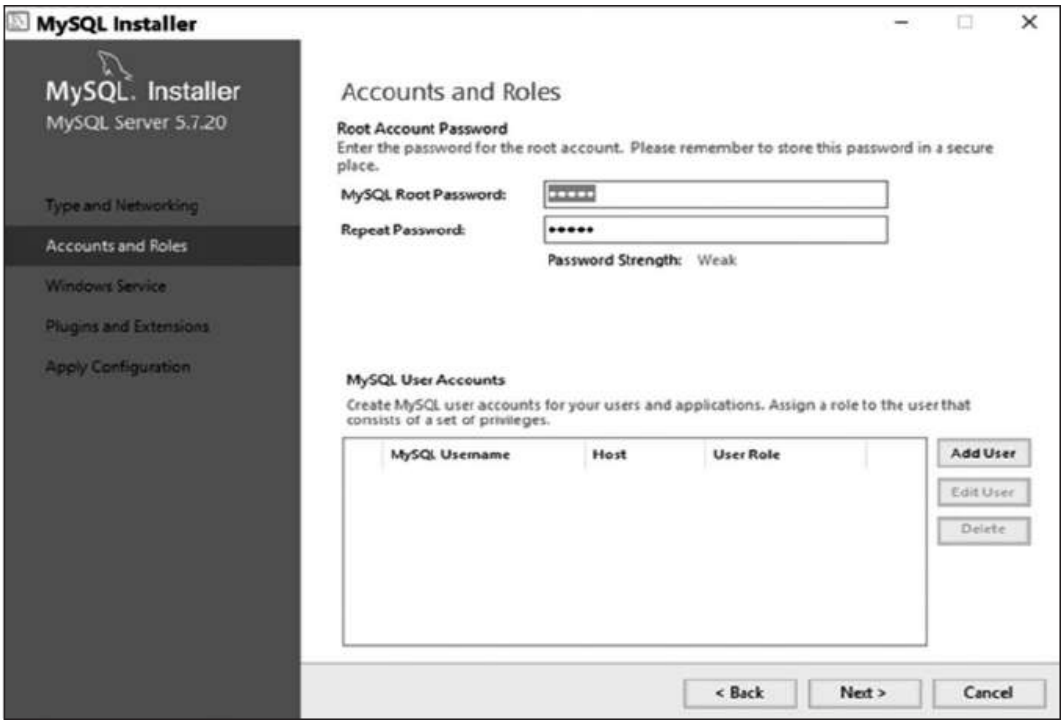


Figura 6.6 Configuración de la cuenta raíz en el servidor

9. En la pantalla siguiente se presenta la opción Windows Service que básicamente configura la instancia de MySQL en el sistema operativo. Se debe dejar como aparece en la pantalla y pulsar el botón Next (siguiente).

10. Las pantallas siguientes son informativas. Es preciso seguir las instrucciones que muestran cada pantalla hasta finalizar la instalación.

6.2.1 Instalación de front-end dbforgemysqlfree

Una vez instalado MySQL se observa que éste 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 cual, 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 cómo se hace.

Al realizar la instalación anterior también se ha instalado el IDE MySQL WorkBench (software de administración del motor de base de datos MySql). En este IDE se puede trabajar la implementación de una base de datos desde una interfaz gráfica. La **figura 6.7** muestra la pantalla inicial de este IDE. Cabe recordar que se configuró el servidor con el usuario root y la contraseña 12345.

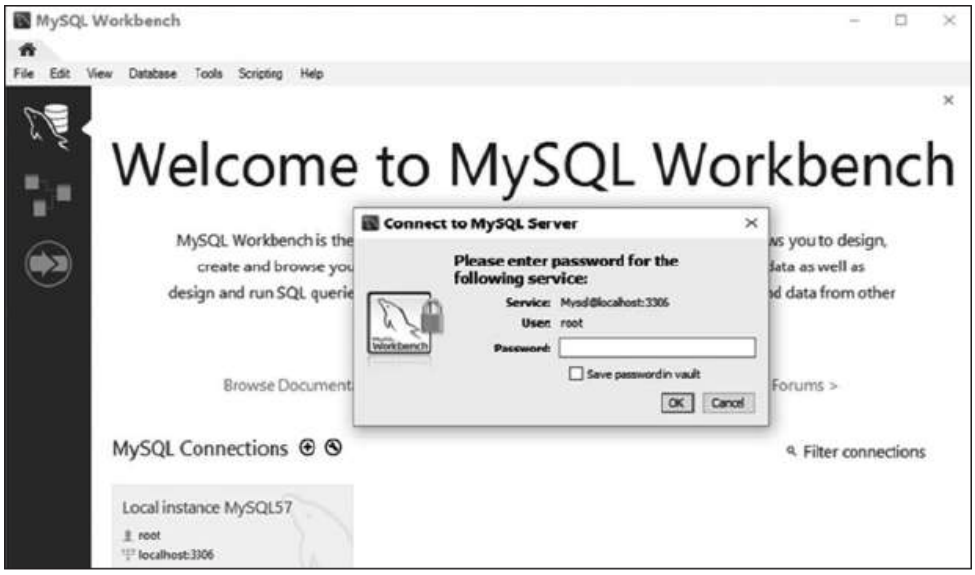


Figura 6.7 Ingreso al servidor MySQL

Una vez que se ingresa se puede observar el entorno de trabajo, según la **figura 6.8**.

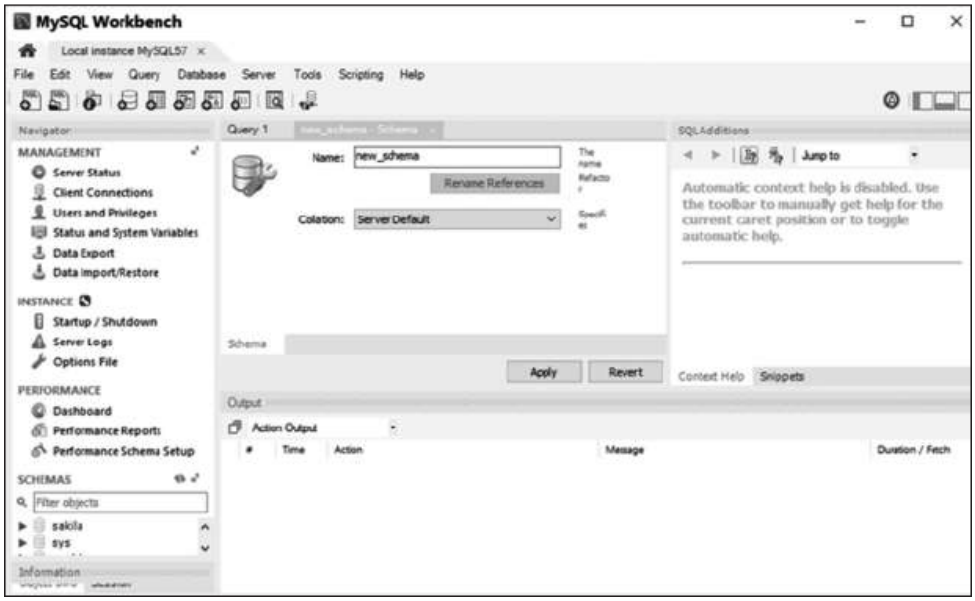


Figura 6.8 Entorno de trabajo con MySQL Workbench

De manera particular se ha decidido utilizar otro IDE. La interfaz usada como front-end, es decir, cómo se hará la comunicación con MySQL será a través del software dbforgemysqlfree en su versión exprés. Se puede descargar desde http://www.devart.com/dbforge/mysql/studio/ La versión a descargar es **dbForge Studio for MySQL, v7.2 Express**. Se inicia entonces con el proceso de instalación del software.

1. Se inicia 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. Se da doble clic sobre este archivo, el cual se incluye en el presente libro. La primera pantalla se observa en la **figura 6.9**.

Figura 6.9 Instalación del front-end dbForge para MySQL

3. El asistente llevará paso a paso por el instalador. Al final solicitará si se desea su ejecución. Responder que sí y se mostrará una pantalla como en la **figura 6.10**.

4. Una vez digitada la clave (12345) se mostrará una pantalla similar a la **figura 6.11**.

Ya está instalado MySQL y el IDE de gestión dbForge. Por tanto, se diseñará una base de datos con este IDE y posteriormente se desarrollará toda la teoría que le da soporte a la gestión de datos en Java.

Figura 6.10 Ingreso a dbForge

Figura 6.11 dbForge abierto

6.2.2 Creación de base de datos con dbforgemysqlfree

1. Ingresar nuevamente a dbForge Studio Express. Conectarse al servidor como se muestra en la **figura 6.10**.

2. Una vez conectado, crear la primera base de datos. Se denominará *dbTalleres*. Para crear una base de datos se 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 las **figuras 6.12** y **6.13**.

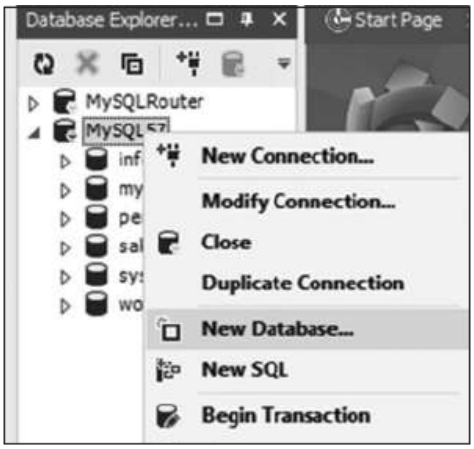


Figura 6.12 Creación de la base de datos *dbTalleres*

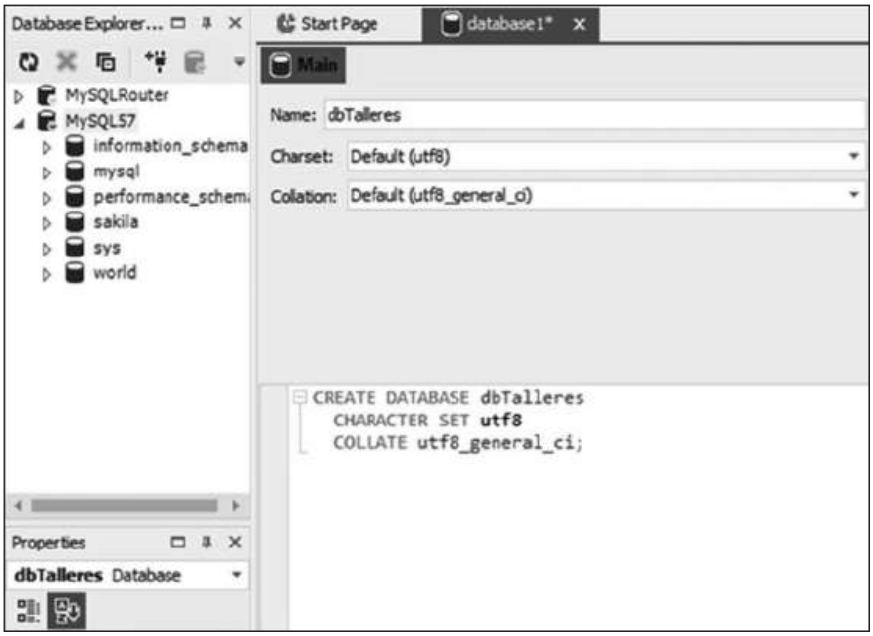


Figura 6.13 Creación de la base de datos *dbTalleres*

3. Para finalizar la creación de la base de datos se presiona el botón Update Database. En este momento ya está creada la base de datos, como se puede observar en la **figura 6.14**.

Figura 6.14 Base de datos dbTalleres creada

4. Una vez generada la base de datos *dbTalleres*, se creará una tabla en ella. Si se pulsa sobre el nombre de la base de datos se mostrará un menú de contexto que permitirá agregar una tabla. Diseñar esta tabla como se muestra en la **figura 6.15**.

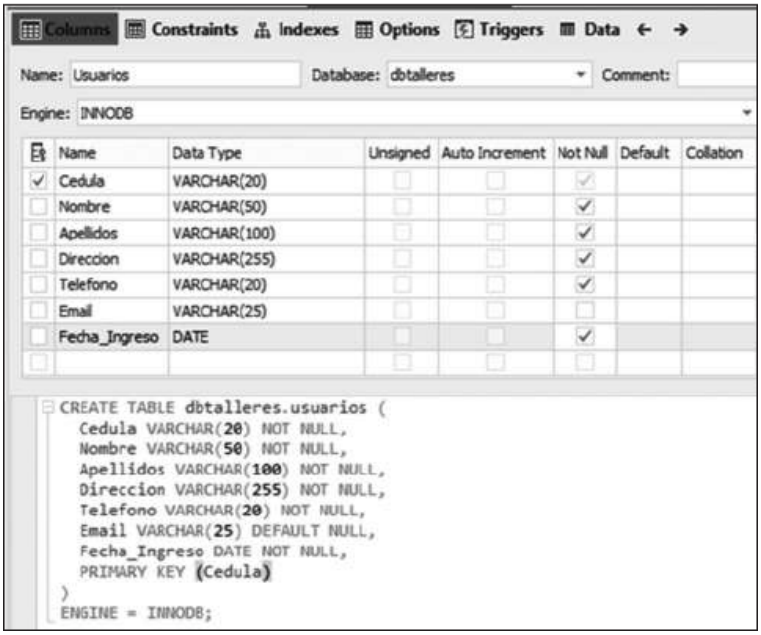


Figura 6.15 Creación de la tabla Usuarios

5. Una vez diseñada la tabla se da clic en el botón Upgrade Database y aquella aparecerá como creada.

Se ha creado una base de datos que se utilizará para demostrar el uso de JDBC (Java Data Base Connectivity) en aplicaciones NetBeans–Java con el motor de datos MySQL. Ahora es necesario conocer algunos conceptos básicos de SQL (Structured Query Language).

Una base de datos relacional está conformada por tablas relacionadas entre sí y cada una de ellas contiene filas y columnas que guardan cierta información. A las filas se les denomina registros o tuplas y están conformadas por varias columnas de una tabla (por ejemplo, una fila o tupla podría integrarse por las columnas identificador, nombre, apellidos, dirección y teléfono). De acuerdo con esto, una columna es de un mismo tipo de datos e identifica un campo de una tabla (por ejemplo, la columna nombre identifica el nombre de un cliente, un proveedor o de un empleado). La colección de filas, tuplas o registros conforma una tabla.

6.2.3 Introducción a SQL

SQL (Structured Query Language) es un lenguaje que permite interactuar con una base de datos relacional: crear archivos y gestionar los datos dentro de ellos. A través de un motor de bases de datos relacional o DBMS (DataBase Management System) es posible crear estructuras de datos (bases de datos, tablas, índices, llaves, entre otros), almacenar o extraer información de ellas, entre otras funcionalidades.

A continuación se dará un breve recorrido por la estructura y sintaxis del SQL como lenguaje de consulta estándar.

Componentes de SQL

Está compuesto por comandos, cláusulas, operadores y funciones de agregado. Estos componentes se combinan en las instrucciones, de manera que se puede crear, actualizar, consultar y eliminar las bases de datos y los objetos que conforman alguna base de datos, como las tablas.

Comandos de SQL

Existen dos lenguajes en las bases de datos relacionales:

LDD (Lenguaje de definición de datos). También se le conoce con DDL por sus siglas en inglés. Proporciona órdenes para la definición de esquemas de relación, borrado de relaciones, creación de índices y modificación de esquemas de relación. En general, tiene que ver con lo relativo a la creación, actualización y borrado de todos los objetos que formarán la base de datos.

LMD (Lenguaje de manipulación de datos). Se denomina DML por sus siglas en inglés. Incluye órdenes para insertar, borrar, consultar y actualizar tuplas o registros de una base de datos ya existente.

Cada uno de estos lenguajes posee sus propios comandos, los cuales se muestran en la **tabla 6.1** y **6.2**.

Tabla 6.1 Comandos del lenguaje DDL.

COMANDO	DESCRIPCIÓN
CREATE	Utilizado para crear nuevas bases de datos, tablas, vistas, procedimientos almacenados, entre otros.
DROP	Permite eliminar bases de datos, tablas y todos los objetos que hayan sido creados con el comando CREATE.
ALTER	Usado para modificar las bases de datos, las tablas y los objetos que han sido creados con el comando CREATE.

Tabla 6.2 Comandos del lenguaje DML

COMANDO	DESCRIPCIÓN
SELECT	Utilizado para consultar registros de la base de datos que satisfagan un criterio determinado.
INSERT	Se usa para cargar lotes de datos en la base de datos en una sola operación. Se insertan registros completos.
UPDATE	Usado para modificar los valores de los campos y registros especificados.
DELETE	Utilizado para eliminar registros de una tabla de una base de datos.

Cláusulas SQL

Son condiciones de modificación utilizadas para definir los datos que se desea seleccionar o manipular. Algunas de las cláusulas más usadas se muestran en la **tabla 6.3**.

Tabla 6.3 Cláusulas SQL.

COMANDO	DESCRIPCIÓN
FROM	Utilizada para especificar la tabla de la cual se van a seleccionar los registros.
WHERE	Se usa para especificar las condiciones que deben reunir los registros que se van a seleccionar.
ORDER BY	Permite ordenar los registros seleccionados de acuerdo con un orden específico, el cual puede ser ascendente o descendente.

Consultas de selección

Se utilizan para indicar al motor de datos que devuelva información de las bases de datos, la cual se regresa en forma de conjunto de registros que se pueden almacenar en un objeto recordset. Este conjunto de registros puede ser modificable.

La sintaxis básica de este tipo de consultas es:

```
SELECT campos
FROM tabla
WHERE condiciones;
```

En la estructura sintáctica anterior, campos es la lista de aquellos (separados por coma) que se deseen recuperar y mostrar; tabla es el origen de los mismos y condiciones son los criterios de filtro que se aplican para desplegar los registros deseados. A continuación se muestra un ejemplo de una consulta sencilla:

```
SELECT nombre, telefono
FROM Cliente
WHERE provincia=”Guanacaste”;
```

En este ejemplo se observa un listado de los nombres y teléfonos de todos los clientes de la tabla Cliente que vivan en la provincia de Guanacaste.

Consultas con predicado

El predicado se incluye entre la cláusula y el primer nombre del campo a recuperar. En la siguiente tabla se muestran algunos de los predicados que se pueden usar y su descripción.

Tabla 6.4 Predicados en SQL.

PREDICADO	DESCRIPCIÓN/EJEMPLO
ALL (*)	Devuelve todos los campos de la tabla de resultados.
DISTINCT	Elimina en el resultado los registros duplicados.

Consultas de acción

Son aquellas que no devuelven ningún registro; se encargan de acciones como añadir, borrar y modificar registros.

DELETE: Consulta que elimina los registros de una o más de las tablas listadas en la cláusula FROM y que satisfacen los criterios de la cláusula WHERE. Este comando elimina registros completos, no es posible sólo borrar campos. Es importante destacar que una operación borrar en firme no se puede deshacer.

Su sintaxis es:

```
DELETE FROM Tabla WHERE criterio
```

Un ejemplo sencillo para el DELETE es:

```
DELETE
FROM Empleados
WHERE  cargo = 'Vendedor';
```

En el ejemplo anterior se eliminarán todos los registros de la tabla Empleados donde el cargo del empleado sea Vendedor.

INSERT INTO: Agrega un registro en una tabla.

Su sintaxis básica es:

```
INSERT INTO Tabla (campo1, campo2, ..., campoN)
VALUES (valor1, valor2, ..., valorN)
```

Esta consulta graba en el campo1 el valor1, en el campo2 el valor2 y así sucesivamente. Un ejemplo de esta operación es:

```
INSERT INTO Empleados (nombre, apellido, cargo)
VALUES ('Luis', 'Sánchez', 'Becario');
```

En el ejemplo se agrega un Nuevo registro en la tabla Empleados, en el campo nombre el valor insertado es 'Luis'; para el campo apellido el valor es 'Sánchez' y por último, el campo cargo recibe el valor de 'Becario'.

UPDATE: Crea una consulta de actualización que cambia los valores de los campos de una tabla especificada basándose en un criterio determinado. Su sintaxis es:

```
UPDATE Tabla SET campo1=Valor1, campo2=Valor2, campoN=ValorN
WHERE Criterio;
```

Es especialmente útil cuando se desea cambiar un gran número de registros o cuando éstos se encuentran en múltiples tablas. Un ejemplo del uso de este tipo de sentencias de actualización es:

```
UPDATE Productos
SET precio = precio * 1.1, porcUtilidad = porcUtilidad * 1.20
WHERE paisOrigen = 'CR';
```

En este ejemplo se incrementa el precio y el porcentaje de utilidad para todos aquellos registros de la tabla Productos que se producen en Costa Rica (CR). En el caso del precio se aumenta un 10% y el porcentaje de utilidad sufre una alza de 20%.

Consultas de unión interna

Se realizan mediante la cláusula **INNER JOIN** que combina registros de dos tablas siempre que haya concordancia de valores en un campo común. Por lo general, este campo en común se da porque existe una relación entre las 2 tablas que se pretende combinar, en un lado el campo común será clave primaria y en el otro lado de la relación será clave foránea. La sintaxis de este tipo de consultas es:

```
SELECT campos
FROM tb1 INNER JOIN tb2 ON tb1.campo1 comp tb2.campo2
```

En la **tabla 6.5** se explica el contenido de la estructura sintáctica anterior.

Tabla 6.5 Sintaxis del INNER JOIN

PARTE	EXPLICACIÓN
tb1, tb2	Son los nombres de las tablas desde las que se combinan los registros.
campo1, campo2	Son los nombres de los campos que se combinan (campos en común o relacionados). Si no son numéricos, los campos deben ser del mismo tipo de datos y contener lo mismo, pero no necesariamente deben tener el mismo nombre.
Comp	Es cualquier operador de comparación relacional: =, <, < >, <=, =>, ó >. Lo más común es usar el =.

Se puede utilizar una operación **INNER JOIN** en cualquier cláusula **FROM**. Esto crea una combinación por equivalencia conocida también como unión interna.

Las combinaciones equivalentes son las más comunes; combinan los registros de dos tablas siempre que haya concordancia de valores en un campo común a ambas.

El ejemplo siguiente muestra cómo se podrían combinar las tablas `Categorias` y `Productos` basándose en el campo `idCategoria`, donde `Categoria.idCategoria` es la clave primaria y `Producto.idCategoría` es una clave foránea:

```
SELECT nombreCategoria, nombreProducto
FROM Categorias INNER JOIN Productos
  ON Categorias.idCategoria = Productos.idCategoria
```

En este ejemplo, `idCategoria` es el campo combinado pero no está incluido en la salida de la consulta ya que no está integrado en la instrucción SELECT. Como se puede observar, el nombre del producto se obtiene de la tabla Productos, pero el nombre de la categoría se encuentra en la tabla Categorías, por lo tanto, es necesario hacer la combinación mediante INNER JOIN.

Aquí termina el breve recorrido en el tema de SQL como lenguaje de consulta de las bases de datos relacionales. Con esta breve introducción se tienen los conocimientos necesarios para continuar con la siguiente sección. Para realizar consultas más complejas es necesario profundizar en la temática, por lo que se insta a aprender más de este poderoso lenguaje y así sacar mayor provecho de los datos almacenados en las bases de datos y en pro del beneficio de las organizaciones.

Enseguida se analizará la gestión de datos de una base MySql con el uso de Java como lenguaje de programación conectado a una base de datos.

6.3 Gestión de datos en MySQL con NetBeans

La persistencia de datos se refiere a la capacidad de los mismos para permanecer almacenados en una computadora para luego utilizarse nuevamente. Es decir, los datos tienen una vida efímera en un aplicativo y por tanto su persistencia se ve amenazada una vez que se cambian sus valores en la memoria principal de la máquina. A partir de ese momento su persistencia se pierde. Más aún si la máquina se apaga, desaparece con ello los datos. Por eso es necesario almacenarlos en un medio permanente, por ejemplo en un disco duro o inclusive en la nube.

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:

Tiene como objetivo preservar la integridad referencial de los objetos con la desventaja de que no soportan la concurrencia de muchos usuarios que acceden a los datos. Mediante el proceso de serialización se crean flujos de bytes independientes de la arquitectura del hardware donde se utilizan, y del lenguaje de programación que se utilice para crearlos o recrearlos.

Un problema de la serialización es que los objetos al contener apuntadores hacia otros o hacia datos en memoria y al ser enviados, puede que suceda que las direcciones referenciadas por los apuntadores no concuerden con las que se están en el equipo donde se reconstruyen, ni siquiera se garantiza que se hayan enviado los objetos a los que se referencian.

JDBC:

JDBC constituye una interfaz de programación de aplicaciones (API) desarrollado para el lenguaje de programación Java y que permite el acceso de una interfaz cliente a una base de datos. JDBC está conformada por métodos orientados a realizar operaciones CRUD (Create, Read, Update y Delete) y definir esquemas de una base de datos. Constituye parte de la plataforma JSE (Java Standard Edition) de la empresa Oracle.

En este contexto JDBC le da el trabajo al programador para que maneje el estado de los objetos y la proyección en las relaciones que establezcan con el sistema administrador de la base de datos. Entonces, se puede desde crear el esquema de la base de datos (el esqueleto) hasta las operaciones de insertar, actualizar, eliminar y buscar/listar datos desde una base de datos relacional.

SQLJ:

Tiene como propósito que Java embeba el código SQL estático en la programación, facilitando así 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).

En este contexto, se centrará el 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, tal 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.

6.3.1 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 utilice, 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 observa la arquitectura Java en la **figura 6.16**.

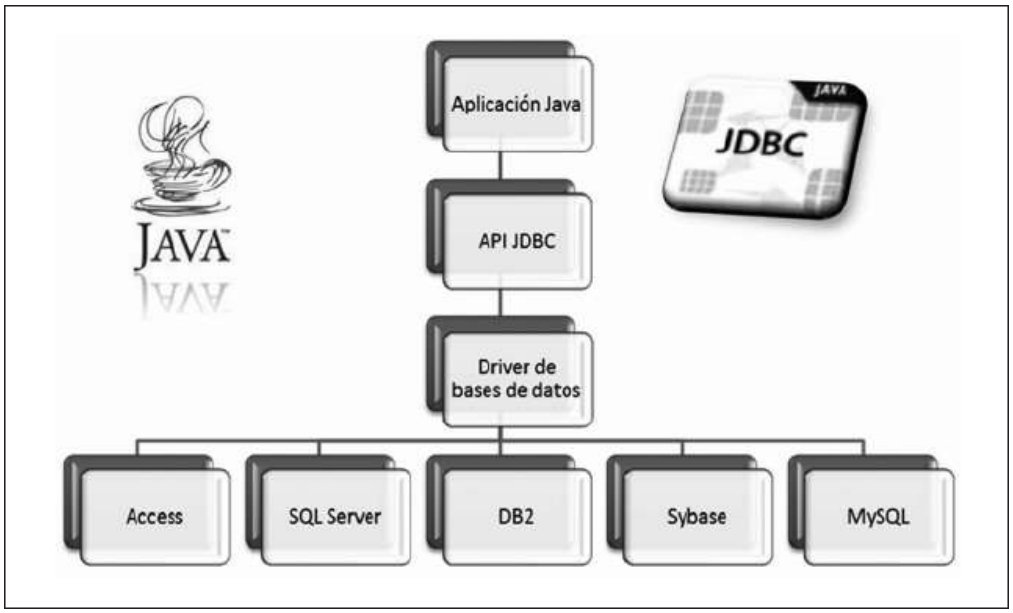


Figura 6.16 Arquitectura JDBC

Como se puede observar en la **figura 6.16** en la parte de arriba de la estructura se encuentra la aplicación JAVA, debajo de ella la API JDBC que permite la interacción entre la aplicación y el driver de la base de datos. Abajo de la API JDBC se encuentran los diferentes drivers de base de datos 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 drivers, 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 a cualquier base de datos, dado que los controladores ODBC de la base 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 que 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 de que no realiza el proceso de traducción mediante el puente–controlador. Ofrece 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, siendo inútil su uso para aplicativos web. La portabilidad no existe para aplicativos 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 por lo que es portable y adecuado para aplicativos web. Su única desventaja quizás 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 aplicativos web. Se necesita un controlador para cada base de datos.

A continuación, la **figura 6.17** muestra la API de JDBC. Se observa que existen muy pocas capas y componentes, lo que la hace muy versátil, sencilla, pero poderosa a la vez.

Los componentes de la API JDBC se describen en la **tabla 6.6**.

Tabla 6.6 Componentes de la API JDBC

COMPONENTE	FUNCIÓN
DriverManager	Se utiliza para cargar un driver.
Statement	Usado para crear consultas y enviarlas a la base de datos.
Connection	Permite conectarse a una fuente de datos.
Resultset	Se utiliza para almacenar el resultado de la consulta.

COMPONENTE	FUNCIÓN
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.

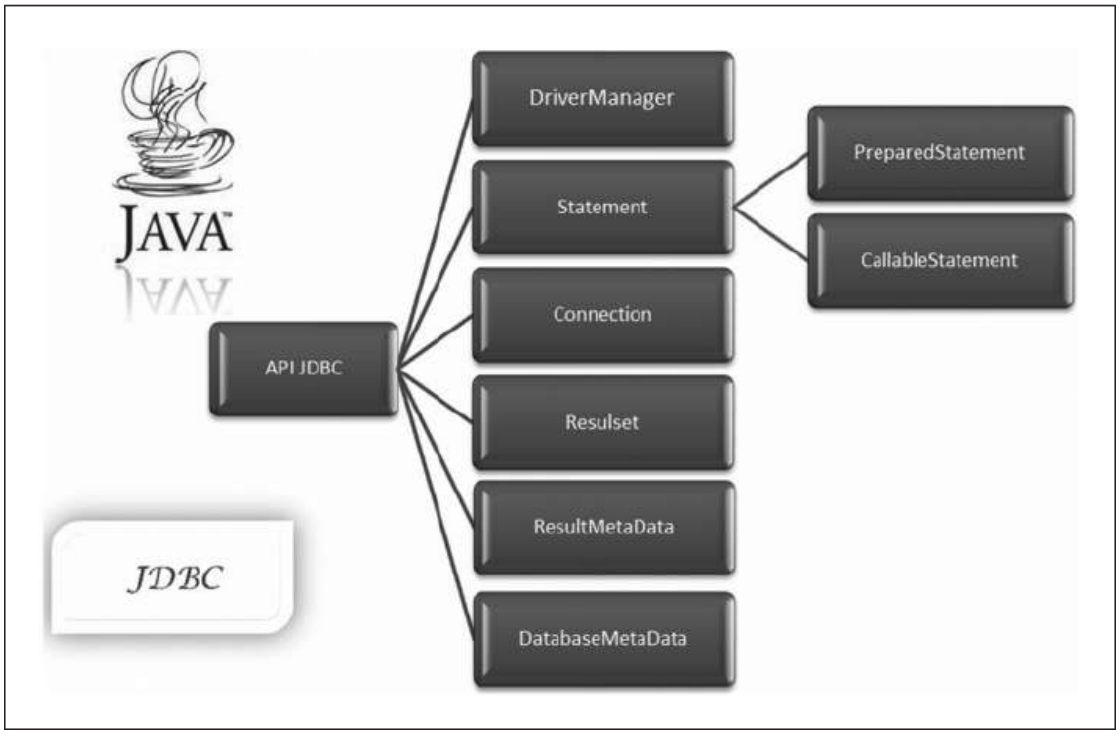


Figura 6.17 Arquitectura JDBC

6.3.2 Conectividad a una base de datos

Para conectarse a una base de datos se necesitan los componentes mostrados en la **figura 6.17** y brevemente explicados en la tabla 6.6. A continuación se detallan más dichos componentes para construir 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 a utilizar en la máquina virtual de Java (JVM), JDBC supone que todos los drivers de la base 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. Agregar a la variable CLASSPATH el archivo que contiene las clases del driver de la base de datos a utilizar. Es posible que también resulte agregarlo al directorio Jre\lib\ext ubicado en el directorio raíz del JDK Standard Edition.

2. Utilización del 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 conjuga la carga del driver de una base de datos MySQL y su respectiva conexión.

Listado No. 6.1 Carga de un Driver MySQL y establecimiento de la conexión a la base de datos.

```
1.  static Connection conn=null; //declaración de un objeto de tipo conexión.
2.  static Statement stm = null; declaración de un objeto de tipo sentencia SQL
3.  static ResultSet rs = null; //declaración de un cursor para almacenar los datos
4.  static String SQL; //declaración de una cadena para almacenar sentencias SQL
5.  static String user = "usuario"; //se establece un nombre de usuario
6.  static String pwd = "12345"; //se estable la contraseña
7.  static String bd="dbTalleres";  //nombre de la base de datos previamente creada
8.  static String url ="jdbc:mysql://localhost:3306/" & bd; //establezco
9.                                  //el string que establece la conexión de la base de datos.
10. /**Creación de una función para establecer la conexión a la base de datos**/
11. public static Connection Enlace(Connection conn) throws SQLException
12. {
13.     try {
14.         Class.forName("com.mysql.jdbc.Driver");  //carga del driver…
```

```
15.          conn = (Connection) DriverManager.getConnection(url,user,pwd); //Se
16.                              //establece la conexión a través del DriverManager
17.      } catch (ClassNotFoundException ex) {
18.          javax.swing.JOptionPane.showMessageDialog(null,ex);
19.      }
20.      return conn;
21.  }
```

Statement:

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 utiliza asociado al método `executeQuery()`. En este caso, los registros que retornan a través de la consulta deben ser cargados en un objeto `ResultSet`.

La sintaxis para crear un Statement es la siguiente:

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

Y para utilizar el Statement conjuntamente con un ResultSet:

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

Ejemplo:

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

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

```
Statement stm = conn.createStatement();
ResultSet rst = stm.executeUpdate("Insert Into Usuarios(Cedula, Nombre,
Apellidos, Direccion, Telefono, Email, Fecha_Ingreso) Values ('123456',
'Cristobal','Núñez Jiménez', 'Barrio La Cruz', '654321', 'cristobal@
antes.com', '2018-06-19 18:00:00'))";
```

PreparedStatement

`PreparedStatement` es un objeto que permite ejecutar sentencias precompiladas, es decir, se ha construido su árbol de ejecución desde la primera vez que se aplicó y se dinamizan en cuanto a los valores de los parámetros que utiliza en la sentencia SQL. Generalmente se usa para ejecutar sentencias que se valen de 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 Usuarios set
email = 'Núñez@antes.com' set codigo = ?")
pstmt.setInt(1,123456);
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 aplicativo de esta condición es la siguiente:

```
preparedStatement pstmt = conn.prepared.Statement("Select *From
Usuarios Where Cedula  = ?");
pstmt.setString(1,"123456");
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 Usuarios WHERE Apellidos = ? And Fecha_Ingreso > ?";
smt.setString(1,"Colón");
smt.setString(2,'2018-06-19');
```

Puede ser que la consulta anterior no tenga mucha lógica puesto que las comparaciones deberían ser entre campos que son disímiles y fácilmente procesables entre los registros, por ejemplo fechas, tipo de usuario, código de provincia, entre otros. Los tipos de datos para los parámetros pueden ser `setBoolean`, `setDate`, `setDouble`, `setFloat`, `setString` y `setInt`.

CallableStatement:

Con CallableStatement se permite la ejecución de procedimientos almacenados (stored procedures) que se crean 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 `dbTalleres` como se muestra en el listado 6.2.

Listado No. 6.2 Creación de un procedimiento almacenado en MySQL

```
1.  CREATE DEFINER = 'egomez'@'localhost'
2.  PROCEDURE dbTaller.spConsultarUsuario(IN _Cedula CHAR(20))
3.  BEGIN
4.      SELECT *FROM tbUsuarios WHERE Cedula = _Cedula;
5.  END
```

Y la ejecución desde Java con un `CallableStatement` se muestra en el listado 6.3.

Listado No. 6.3 Utilización de `CallableStatement` para llamar un procedimiento almacenado en MySQL

```
1.  CallableStatement cStmt = (CallableStatement) conn.prepareCall("{call sp-
    ConsultarUsuario(?)}");
2.  cStmt.setString(1, txtCedula.getText());
3.  cStmt.execute();
4.  rs = cStmt.getResultSet();
5.  while(rs.next()){
6.  …
7.  }
```

ResultSet:

Mediante `ResultSet` se obtienen los resultados de la ejecución de una consulta usando Statement. Se asocia con la sentencia Select del 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 de 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 en el listado 6.4.

Listado No. 6.4 Utilización de `ResultSet` para recibir el resultado de la ejecución de una consulta en MySQL:

```
1.  String SQL="SELECT *  FROM Usuarios";
2.  Statement stmt  = conn.createStatemet( );
3.  ResultSet resultado  = stmt.executeQuery(SQL);
4.  while(resultado.next( )){
5.       System.out.println(" Cédula : "+resultado.getString("Cedula"));
6.       System.out.println(" Nombre : "+resultado.getString("Nombre"));
7.  }
```

Se observa cómo en la primera línea se crea un SQL que permite obtener todo el listado de usuarios encontrados en la tabla Usuarios. La siguiente línea crea un objeto Statement en la 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 se finalizará el capítulo con la puesta de todo esto en práctica, es decir, con la creación de un sencillo ejemplo. Para ello se utilizará la base de datos *dbTalleres* que se creó al iniciar este capítulo. Entonces, se da por hecho que esa parte ya se tiene, ahora se genera la aplicación.

Ejemplo No. 1: Creación de aplicación Java de mantenimiento de datos MySql:

1. En el explorador de Windows (en este es caso es Windows 10) se crea una carpeta denominada dskTalleres en la ubicación deseada para almacenar los archivos necesarios.

2. Lo primero que se debe hacer es descargar un archivo JAR que permita manejar las fechas a utilizar en los formularios de la aplicación. Para ello, se descarga el siguiente archivo *jdatechooser.JAR* de la dirección http://jdatechooser.sourceforge.net/. Se guarda en una carpeta tal como *dskTaller* es que se acaba de crear o en aquella que se considere fácil de ubicar para utilizar el componente descargado.

Se debe extraer el archivo zip descargado para obtener la librería `DateChooser.jar` requerida en el proyecto.

3. Enseguida es preciso descargar dentro de la misma carpeta dskTalleres otra librería que se usará también en el manejo de fechas y calendarios en la interfaz gráfica. El nombre del archivo librería a descargar es LGoodDatePicker-10.3.1.jar. La dirección es la siguiente: https://github.com/LGoodDatePicker/LGoodDatePicker/releases/tag/v10.3.1-Standard

Una vez descargado el archivo se debe descomprimir para usarlo posteriormente.

4. Para conectar la aplicación Java a la base de datos en MySql se debe descargar un driver de conexión denominado (mysql-connector-java-5.1.45.zip), el cual es otra librería que se puede descargar de https://dev.mysql.com/downloads/connector/j/5.1.html.

5. Descomprimir el archivo zip anterior en la carpeta `dskTalleres`.

6. Una vez descargadas las librerías necesarias se debe crear un proyecto nuevo en NetBeans, pero esta vez será de tipo escritorio (desktop), como se muestra en la **figura 6.18**. La carpeta que se utilizará para guardar este proyecto será la recién creada dskTalleres.

7. Después de seleccionar Java Application en el tipo de proyecto, pulsar el botón Siguiente. En la pantalla mostrada dar el nombre (dskTalleres) al proyecto creado y ubicarlo en la carpeta dskTalleres, como se muestra en la **figura 6.19**.

8. Pulsar la tecla Terminar.

9. Una vez creado el proyecto se genera una carpeta (paquete) donde se almacenan los formularios. La **figura 6.20** muestra la forma de crear estos paquetes.

10. El paquete a crear se denominará Formularios. La **figura 6.21** muestra cómo se crea un formulario posicionándose sobre el paquete Formularios.

11. De la misma manera que se creó el paquete Formularios, crear los siguientes paquetes: Datos, Logica.

Figura 6.18 Creación de aplicación tipo desktop

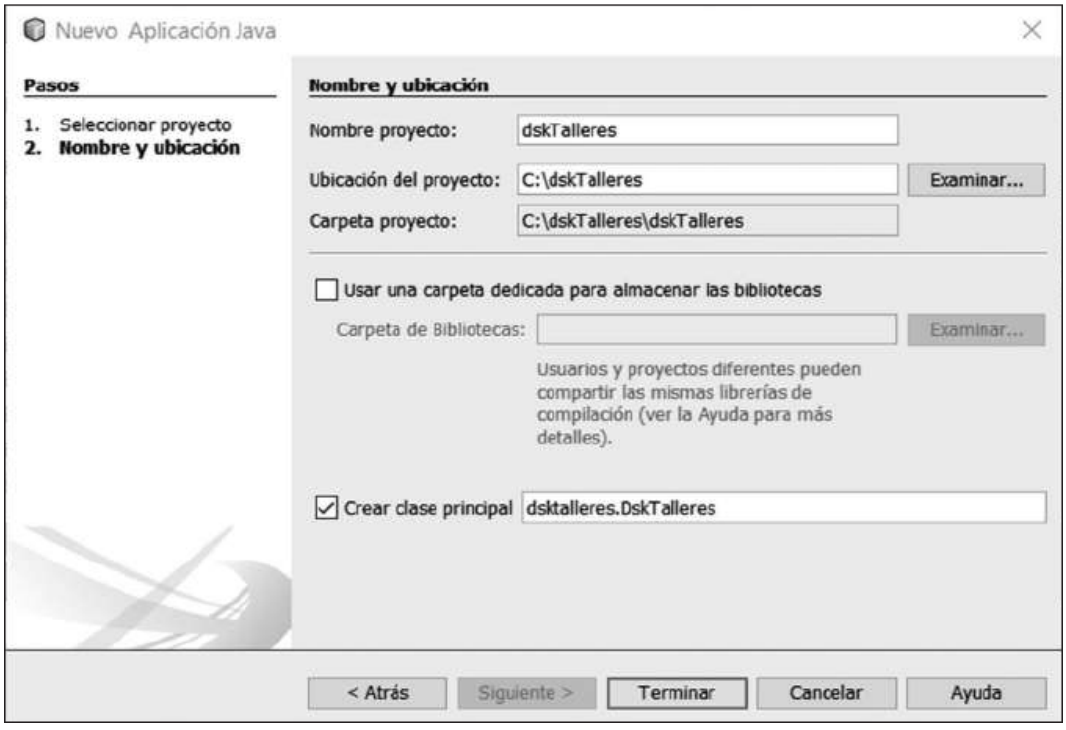


Figura 6.19 Asignación de nombre de la aplicación tipo desktop

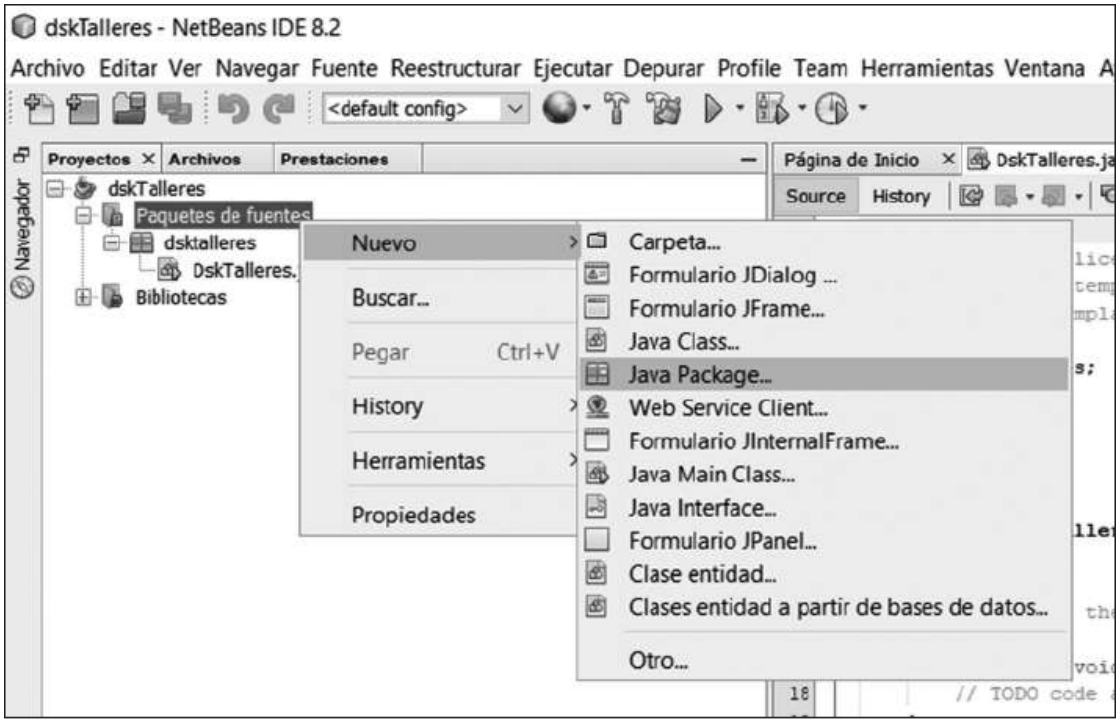


Figura 6.20 Creación de paquetes para organizar la aplicación de escritorio

Los paquetes (packages) en Java permiten agrupar lógicamente los componentes que se van a crear en una aplicación. Esto significa incluir en paquetes clases, interfaces, archivos de texto, entre otros. Con ello se organiza adecuadamente una aplicación.

12. Una vez generados nuestros paquetes se crearán los formularios dentro del paquete Formularios.

En la **figura 6.21** se observa la opción JFrame. Los JFrame permiten crear formularios tipo escritorio donde se pueden agrupar componentes tales como cajas de texto, etiquetas, listas, botones, entre otros. En este caso se selecciona un JFrame que se nombra frmUsuarios (*frm* significaría formulario y *Usuarios* el objeto a representar en el formulario). Gran parte de esto ya se explicó en el capítulo 4, que trató sobre aplicaciones de escritorio.

13. Antes de continuar, se agrega a la librería de componentes que se descarga según el punto 2 de este ejercicio. En la **figura 6.22** se muestra cómo se agrega a las librerías de componentes descargadas el grupo Controles Swing.

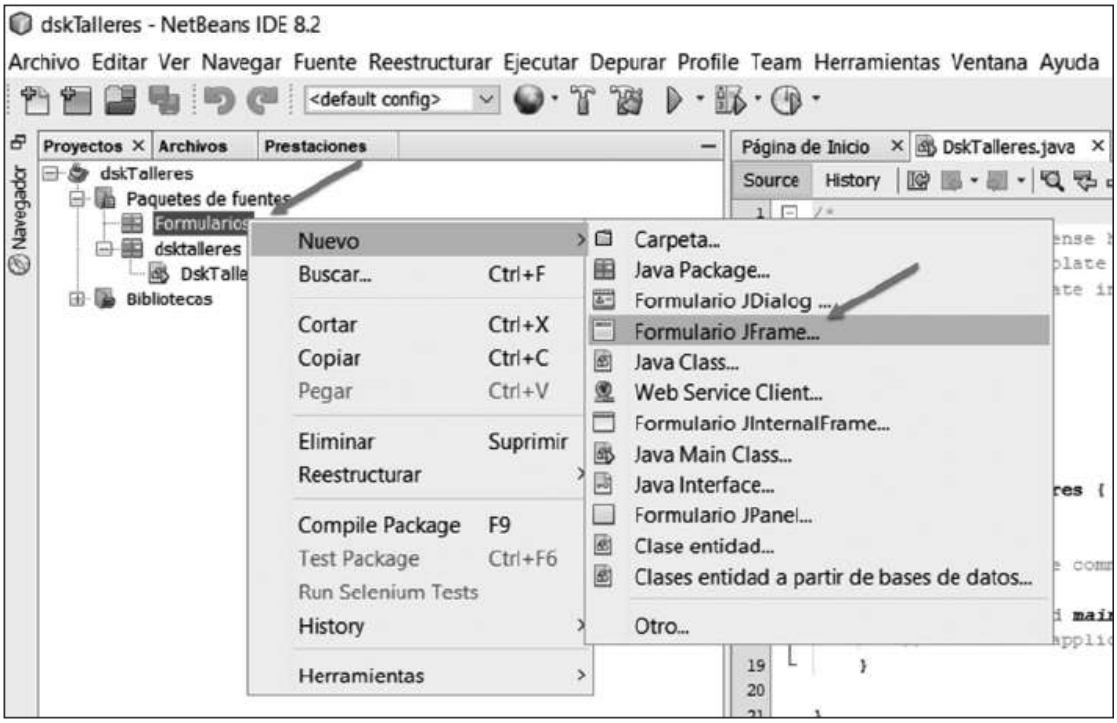


Figura 6.21 Creación de formulario `frmUsuarios`

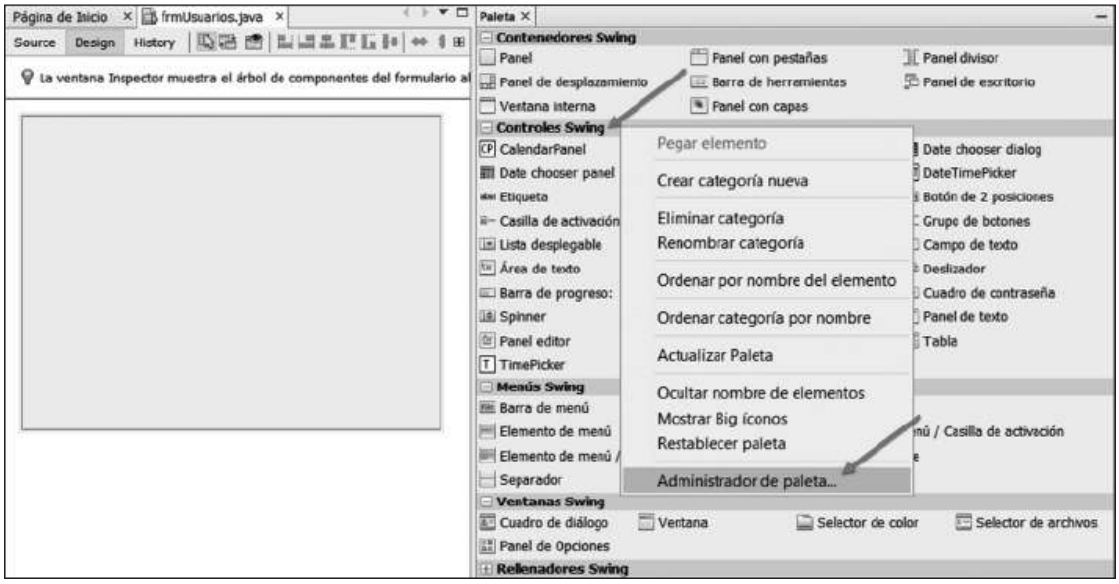


Figura 6.22 Agregar un conjunto de componentes a la paleta

14. Si se pulsa Administrador de paleta se mostrará una caja de diálogo que indica de dónde se obtendrán los componentes. En este caso, se debe seleccionar la opción Añadir

de archivos Jar. Es preciso buscar el archivo DateChooser.jar que se crea en el punto 2 de este ejercicio. Se observa la **figura 6.23** que muestra este procedimiento.

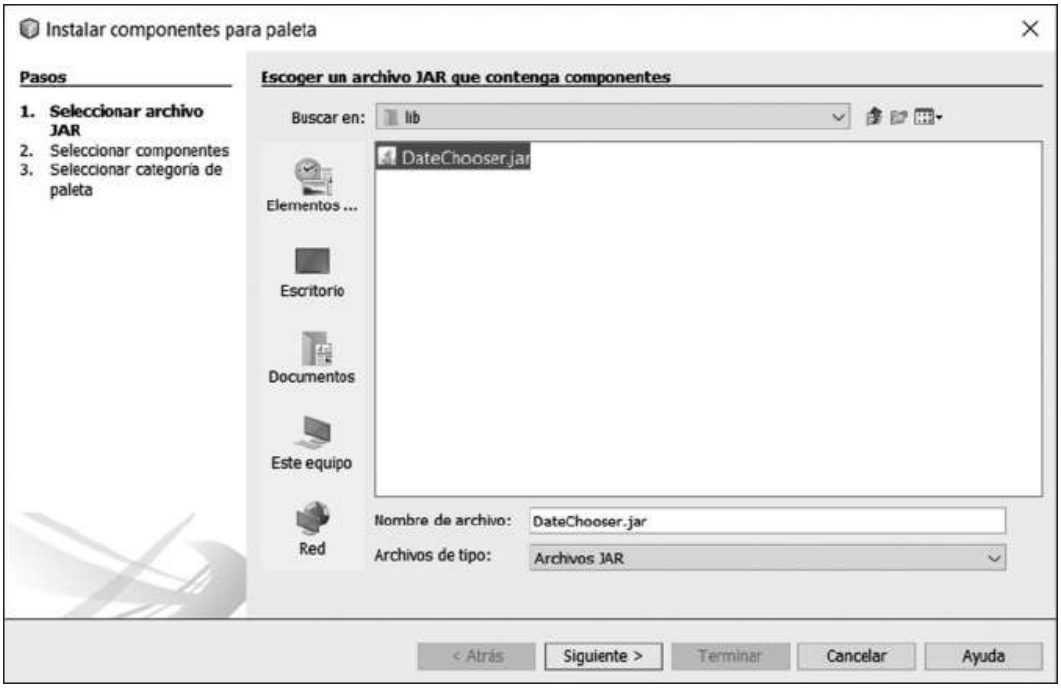


Figura 6.23 Agregar el componente DateChooser al proyecto

15. La pantalla que se muestra a continuación, luego de pulsar el botón Siguiente, solicitará que se seleccionen los componentes requeridos para la aplicación.

La **figura 6.24** permitirá observar todos los componentes contenidos en esta librería. De todos ellos, se deben seleccionar sólo los que se muestran en la **figura 6.24**. Estos componentes son `DateChooserCombo`, `DateChooserDialog` y `DateChoosePanel`.

16. Una vez pulsado el botón Siguiente mostrado en la figura 6.24, se solicitará que decida dónde se colocarán los componentes. Se debe optar por la opción Controles Swing. Pulsar la tecla Terminar y luego Cerrar en la pantalla siguiente. Se mostrarán los componentes instalados en el cuadro de controles, como se muestra en la **figura 6.25**.

17. Replicar el paso anterior para la librería LGoodDatePicker-10.3.1.jar, seleccionar todos los componentes disponibles y agregarlos a la paleta swing.

18. Agregar ahora al proyecto las tres librerías. Hacer clic derecho sobre el proyecto dskTalleres, elegir la opción Propiedades y luego Bibliotecas; dar clic en el botón Añadir JAR/Carpeta, se mostrará enseguida un cuadro de diálogo donde se deben buscar y elegir las tres librería que se descargaron en los puntos 2, 3 y 4. Hacer clic en Abrir y luego en Aceptar.

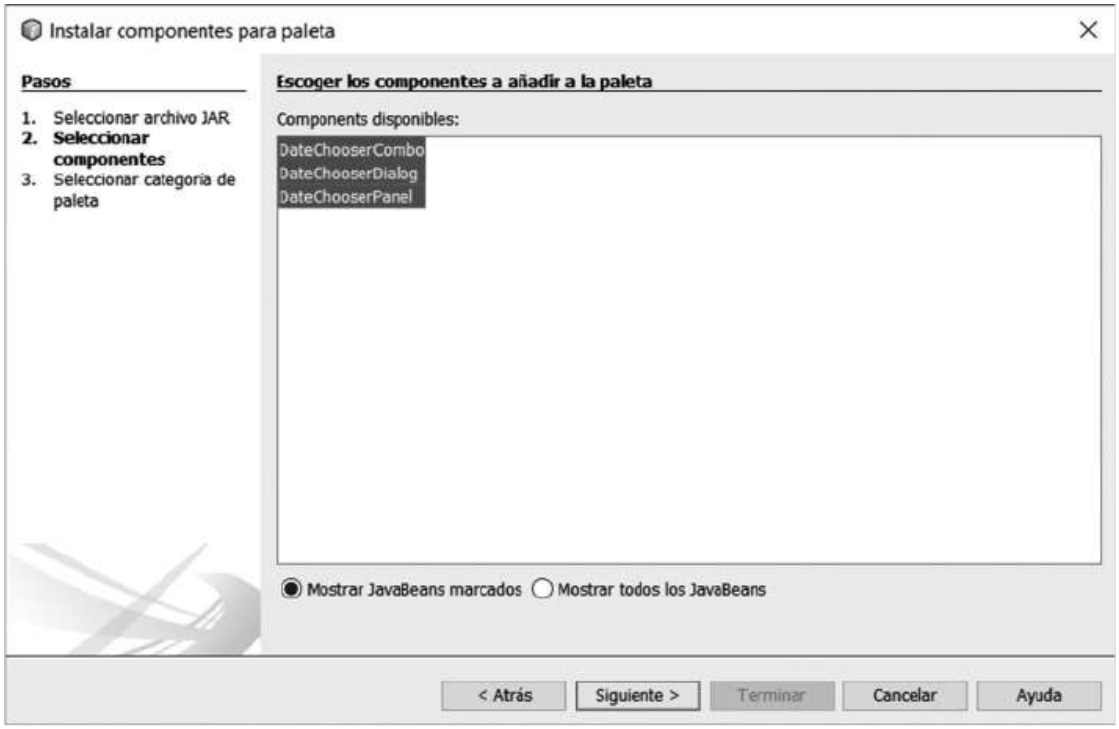


Figura 6.24 Seleccionar los componentes de la librería DateChooser

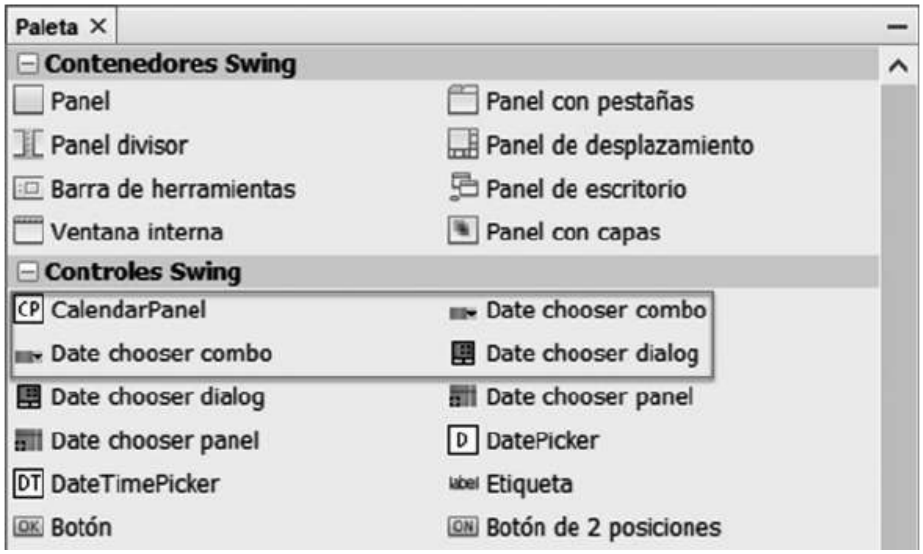


Figura 6.25 Componentes de la librería DateChooser instalados

19. Ahora que ya se tienen los componentes requeridos, se crea la arquitectura del aplicativo de escritorio. Como ya se sabe, ya se tienen todos los paquetes necesarios para organizar las clases en el proyecto, como se muestra en la **figura 6.26**.

Figura 6.26 Estructura de paquetes Java de la aplicación dskTalleres

20. Enseguida se crean 3 formularios de tipo JDialog para realizar las operaciones de Nuevo registro, Modificar y Eliminar.

21. Hacer clic derecho sobre el paquete Formularios, la opción Nuevo, Formulario JDialog, crear un nuevo formulario denominado frmNuevoUsuario.java.

22. Luego, generar a frmEditarUsuario.java y frmEliminarUsuario.java de la misma manera en que se crearon los anteriores. De este modo, ya se tienen creados todos los formularios, pronto se retomarán para diseñarlos y agregarles a cada uno el código respectivo.

23. Ingresar a la clase dskTalleres.java que está dentro del paquete dskTalleres e introducir el siguiente código dentro del método main (), como se muestra en el listado 6.5.

Listado No. 6.5 Desarrollo de la clase principal dskTalleres.java

```
1.  package dsktalleres;
2.  import Formularios.frmUsuarios;
3.  public class DskTalleres {
4.  public static void main(String[] args) {
5.        frmUsuarios form = new frmUsuarios();
6.        form.setVisible(true);
7.  }
```

Como se observa en el listado anterior, en la línea 5 se crea una instancia del formulario frmUsuarios y en la línea 6 se hace visible para que se muestre en pantalla.

24. Ahora se crean las clases que se guardarán en el paquete Datos, la primera es la clase DatosUsuario.java. Hacer clic derecho sobre el paquete Datos, seleccionar la opción Nuevo del menú contextual, luego elegir Java Class, establecer como nombre `DatosUsuario`. Enseguida ingresar a la clase recién creada e introducir el código fuente mostrado en el listado 6.6 a continuación.

Listado No. 6.6 Programación de la clase DatosUsuario

```
1.  package Datos;
2.  import java.sql.Date;
3.  public class DatosUsuario {
4.      private String cedula;
5.      private String nombre;
6.      private String apellidos;
7.      private String direccion;
8.      private String telefono;
9.      private String email;
10.     private Date fecha_ing;
11.     public DatosUsuario( ) {   }
12.     public DatosUsuario(String cedula, String nombre, String apellidos,
        String direccion, String telefono, String email, Date fecha_ing) {
13.             this.cedula = cedula;
14.             this.nombre = nombre;
15.             this.apellidos = apellidos;
16.             this.direccion = direccion;
17.             this.telefono = telefono;
18.             this.email = email;
19.             this.fecha_ing = fecha_ing;
20.     }
21.     public String getCedula( ) {  return cedula; }
22.     public String getNombre( ) {  return nombre;   }
23.     public String getApellidos( ) {  return apellidos;  }
24.     public String getDireccion( ) {  return direccion;  }
25.     public String getTelefono( ) {  return telefono;  }
26.     public String getEmail( ) {  return email;  }
27.     public Date getFecha( ) {  return fecha_ing;  }
```

```
28.     public void setCedula(String cedula) { this.cedula = cedula; }
29.     public void setNombre(String nombre) { this.nombre = nombre; }
30.     public void setApellidos(String apellidos) { this.apellidos = apellidos; }
31.     public void setDireccion(String direccion) { this.direccion = direccion; }
32.     public void setTelefono(String telefono) { this.telefono = telefono; }
33.     public void setEmail(String email) { this.email = email; }
34.     public void setFecha(Date fecha_ing) { this.fecha_ing = fecha_ing; }
35. }
```

Comentarios sobre el código del listado 6.6:

- En la línea 3 se declara la clase `DatosUsuario`.
- En las líneas 4 a 10 se declaran los atributos de la clase los cuales son privados, según las reglas de la programación orientada a objetos.
- En la línea 11 se declara un constructor vacío de la clase.
- Desde la línea 12 a 20 se declara otro constructor, el cual recibe como parámetro un valor para cada atributo y se los asigna a cada uno respectivamente.
- De la línea 21 hasta la 34 se declaran los métodos set y get para tener acceso a los atributos desde fuera de la clase `DatosUsuario`.

25. En el mismo paquete de Datos hacer clic derecho y crear una nueva Java Class a la que se llamará `Conexion`. Agregar el código que se muestra en el listado 6.7. Esta clase usará la librería mysql-connector-java agregada al proyecto en pasos anteriores, con el fin de crear el código necesario para establecer la conexión de la aplicación con la base de datos MySql.

Listado No. 6.7 Programación de la clase `Conexion`

```
1.  package Datos;
2.  import java.sql.Connection;
3.  import java.sql.DriverManager;
4.  import java.sql.SQLException;
5.  import java.util.logging.Level;
6.  import java.util.logging.Logger;
7.  import javax.swing.JOptionPane;
8.  public class Conexion {
9.      private final String db="dbTalleres"; //nombre de la base de datos.
10.     private final String url="jdbc:mysql://localhost:3306/" +db;
```

```
11.     private final String user="root"; //Usuario de la base de datos
12.     public String pass="12345";  //Contraseña para conectarse a la B.D.
13.     public Conexion() {    }
14.     public Connection conectar(){
15.         Connection dbConn=null;
16.         try {
17.         Class.forName("com.mysql.jdbc.Driver");
18.         dbConn=DriverManager.getConnection(this.url, this.user, this.pass);
19.     } catch (SQLException e) {
20.         JOptionPane.showConfirmDialog(null, e);
21.     }      catch (ClassNotFoundException ex) {
22.         Logger.getLogger(Conexion.class.getName()).log(Level.SEVERE,null,ex);
23.     }
24.     return dbConn;
25.   }
26. }
```

Comentarios sobre el código del listado 6.7:

- En la línea 9 se guarda en la variable db el nombre de la base de datos a la que nos conectaremos.
- En la línea 10 se hace referencia al driver de conexión, así como el nombre o dirección IP del servidor de bases de datos a la que nos conectaremos.
- Con respecto a la línea 11 se puede ver que se estable el nombre del usuario con el cual nos conectaremos a la base de datos, en este caso root.
- En la línea 12 se establece el password para el usuario con que nos conectaremos a la base de datos. Si usted estableció otra contraseña, debe cambiarla también en esta línea de código.
- En las líneas 14 hasta la 25 se muestra el código necesario para conectarse a la base de datos con los parámetros que se definieron en las líneas anteriores. Específicamente en la línea 18 se establece la conexión.

26. El siguiente paso será generar la clase Usuario.java que va dentro del paquete Logica; hacer clic derecho sobre el paquete Logica en la opción Nuevo y crear una Java Class nueva a la que se llamará Usuario. Una vez creada la clase, ingresar al código fuente de la misma y agregar el siguiente código que se despliega en el listado 6.8.

Listado No. 6.8 Programación de la clase `Usuario`

```
1.  package Logica;
2.  import Datos.Conexion;
3.  import Datos.DatosUsuario;
4.  import java.sql.Connection;
5.  import java.sql.PreparedStatement;
6.  import java.sql.ResultSet;
7.  import java.sql.SQLException;
8.  import java.sql.Statement;
9.  import javax.swing.JOptionPane;
10. import javax.swing.table.DefaultTableModel;
11. public class Usuario {
12.     private final Conexion cnx = new Conexion( );
13.     private String cSQL = "";
14.     int totReg;
15.     String[ ] titulos = {"Cédula", "Nombre", "Apellidos", "Dirección",
                "Teléfono", "Email", "Fecha Ingreso"};
16.     Object [ ] registro = new Object [7];
17.     public DefaultTableModel Filtrado(String apellidos) {
18.        DefaultTableModel modelo;
19.        Connection cn = cnx.conectar( );
20.        modelo = new DefaultTableModel(null, titulos);
21.        cSQL = "SELECT * FROM usuarios Where apellidos LIKE '%" +
           apellidos.trim( ) + "%'";
22.        try {
23.           private final Conexion cnx = new Conexion( );
24.           Statement st = cn.createStatement( );
25.           ResultSet rs = st.executeQuery(cSQL);
26.           while (rs.next( )) {
27.              registro[0] = rs.getString("Cedula");
28.              registro[1] = rs.getString("Nombre");
29.              registro[2] = rs.getString("Apellidos");
30.              registro[3] = rs.getString("Direccion");
31.              registro[4] = rs.getString("Telefono");
```

```
32.              registro[5] = rs.getString(“Email”);
33.              registro[6] = rs.getString(“Fecha_Ingreso”);
34.              modelo.addRow(registro);
35.           }
36.           rs.close( );
37.           st.close( );
38.           cn.close( );
39.           return modelo;
40.        } catch (Exception e) {
41.           JOptionPane.showConfirmDialog(null, e + “ error mostrar”);
42.           return null;
43.        }
44.     }
45.     public DefaultTableModel BuscarCedula(String cedula) {
46.        DefaultTableModel modelo;
47.        totReg = 0;
48.        modelo = new DefaultTableModel(null, titulos);
49.        cSQL = “SELECT * FROM usuarios Where cedula LIKE ‘%” + cedula.
           trim() + “%’”;
50.        try {
51.           Connection cn = cnx.conectar( );
52.           Statement st = cn.createStatement( );
53.           ResultSet rs = st.executeQuery(cSQL);
54.           while (rs.next( )) {
55.              registro[0] = rs.getString(“Cedula”);
56.              registro[1] = rs.getString(“Nombre”);
57.              registro[2] = rs.getString(“Apellidos”);
58.              registro[3] = rs.getString(“Direccion”);
59.              registro[4] = rs.getString(“Telefono”);
60.              registro[5] = rs.getString(“Email”);
61.              registro[6] = rs.getString(“Fecha_Ingreso”);
62.              totReg += 1;
63.              modelo.addRow(registro);
64.           }
```

```
65.            rs.close( );
66.            st.close( );
67.            cn.close( );
68.            return modelo;
69.         } catch (Exception e) {
70.            JOptionPane.showConfirmDialog(null, e + " error mostrar");
71.            return null;
72.         }
73.     }
74.     public DefaultTableModel listarUsuarios( ) {
75.         DefaultTableModel modelo;
76.         totReg = 0;
77.         modelo = new DefaultTableModel(null, titulos);
78.         cSQL = "Select *From Usuarios";
79.         try {
80.            Connection cn = cnx.conectar( );
81.            Statement st = cn.createStatement( );
82.            ResultSet rs = st.executeQuery(cSQL);
83.            while (rs.next( )) {
84.               registro[0] = rs.getString("Cedula");
85.               registro[1] = rs.getString("Nombre");
86.               registro[2] = rs.getString("Apellidos");
87.               registro[3] = rs.getString("Direccion");
88.               registro[4] = rs.getString("Telefono");
89.               registro[5] = rs.getString("Email");
90.               registro[6] = rs.getString("Fecha_Ingreso");
91.               totReg += 1;
92.               modelo.addRow(registro);
93.            }
94.            rs.close( );
95.            st.close( );
96.            cn.close( );
97.            return modelo;
98.         } catch (SQLException e) {
```

```
99.            JOptionPane.showConfirmDialog(null, e + " error mostrar");
100.           return null;
101.        }
102.     }
103.     public boolean insertar_Usuario(DatosUsuario usuario) {
104.        cSQL = "Insert into usuarios (Cedula, Nombre, Apellidos,
           Direccion, Telefono, Email, Fecha_Ingreso) values
           (?,?,?,?,?,?,?)";
105.        try {
106.           Connection cn = cnx.conectar( );
107.           PreparedStatement pst = cn.prepareStatement(cSQL);
108.           pst.setString(1, usuario.getCedula( ));
109.           pst.setString(2, usuario.getNombre( ));
110.           pst.setString(3, usuario.getApellidos( ));
111.           pst.setString(4, usuario.getDireccion( ));
112.           pst.setString(5, usuario.getTelefono( ));
113.           pst.setString(6, usuario.getEmail( ));
114.           pst.setDate(7, usuario.getFecha( ));
115.           int n = pst.executeUpdate( );
116.           return n != 0;
117.        } catch (SQLException e) {
118.           JOptionPane.showMessageDialog(null, e);
119.           return false;
120.        }
121.     }
122.     public boolean editar_Usuario(DatosUsuario usuario) {
123.        cSQL = "Update usuarios  set Nombre = " + "?,"
                        + "Apellidos = " + "?,"
                        + "Direccion = " + "?,"
                        + "Telefono = " + "?,"
                        + "Email = " + "?,"
                        + "Fecha_Ingreso = " + "?"
                        + " Where Cedula = " + "?";
124.        try {
```

```
125.          Connection cn = cnx.conectar( );
126.          PreparedStatement pst = cn.prepareStatement(cSQL);
127.          pst.setString(1, usuario.getNombre( ));
128.          pst.setString(2, usuario.getApellidos( ));
129.          pst.setString(3, usuario.getDireccion( ));
130.          pst.setString(4, usuario.getTelefono( ));
131.          pst.setString(5, usuario.getEmail( ));
132.          pst.setDate(6, usuario.getFecha( ));
133.          pst.setString(7, usuario.getCedula( ));
134.          int n = pst.executeUpdate( );
135.          return n != 0;
136.       } catch (SQLException e) {
137.          JOptionPane.showMessageDialog(null, e);
138.          return false;
139.       }
140.    }
141.    public boolean eliminar_Usuario(String Cedula) {
142.       cSQL = “Delete From Usuarios where Cedula = “ + “?”;
143.       try {
144.          Connection cn = cnx.conectar( );
145.          PreparedStatement pstEdicion = cn.prepareStatement(cSQL);
146.          pstEdicion.setString(1, Cedula);
147.          int n = pstEdicion.executeUpdate( );
148.          pstEdicion.close( );
149.          cn.close( );
150.          return true;
151.       } catch (SQLException e) {
152.          JOptionPane.showMessageDialog(null, e);
153.          return false;
154.       }
155.    }
156. }
```

Comentarios sobre el código del listado 6.8:

- En la línea 12 se establece la conexión con la base de datos.
- En la línea 17 se declara un método denominado `Filtrado( )`, el cual recibe por parámetro un string que será un apellido a través de cual se puede buscar por apellido en la base de datos y mostrar en la pantalla principal sólo las coincidencias con el criterio de búsqueda.
- La línea 21 contiene la sentencia SQL que hace posible filtrar registros de la tabla por apellidos.
- En la línea 45 se declara el método `BuscarCedula( )`, el cual tiene como función hacer filtrado de registros por cédula, de manera tal que sólo los registros que tengan coincidencia con el criterio de búsqueda sean mostrados en la tabla de la pantalla principal.
- La línea 49 contiene la sentencia SQL que hace posible filtrar registros de la tabla por cédula. Esta consulta es ejecutada por medio de la línea 53.
- En el `while( )` de la línea 54 se recorre cada registro de la tabla resultante después de ejecutar la consulta; uno a uno se agregan al modelo, el cual luego será integrado a la tabla del formulario `frmUsuarios` para ser mostrado como resultado al usuario final.
- En la línea 74 se declara el método `listarUsuarios( )`, el cual tiene como objetivo mostrar en la tabla del formulario principal todos los registros existentes.
- Por medio de la línea 103 se declara el método `insertar_Usuario( )`, el cual tiene la función de insertar un registro en la base de datos con la información que se obtenga del usuario final a través del formulario `frmNuevoUsuario` de captura de datos.
- La línea 104 contiene la instrucción SQL para insertar un nuevo registro. Es importante señalar que los símbolos de pregunta al final de la instrucción representan los valores que serán insertados; éstos se agregan a la consulta desde la línea 108 hasta la 114 en el mismo orden en que cada símbolo de pregunta fue declarado; este procedimiento se hace por asuntos de seguridad, por ejemplo para evitar ataques de inyección SQL. Una vez seteados los valores, en la línea 115 se ejecuta la consulta tipo insert.

- Se puede observar en la línea 122 la presencia del método `editar_Usuario( )`, mismo que tiene el propósito de editar cualquier campo (excepto la cédula que es la clave primaria) de un registro ya existente en la base de datos.

- Por último, en la línea 141 se declara el método `eliminar_Usuario( )` el cual será el responsable de llevar a cabo el borrado de registros de la tabla de la base de datos.

27. Una vez creadas todas las clases del proyecto, se procede a generar el código de los formularios que darán funcionalidad a nuestra aplicación. Entrar al paquete Formularios y dar doble clic sobre `frmUsuarios.java`. Entrar a la vista de diseño y crear una interfaz gráfica similar a la que se muestra en la **figura 6.27**.

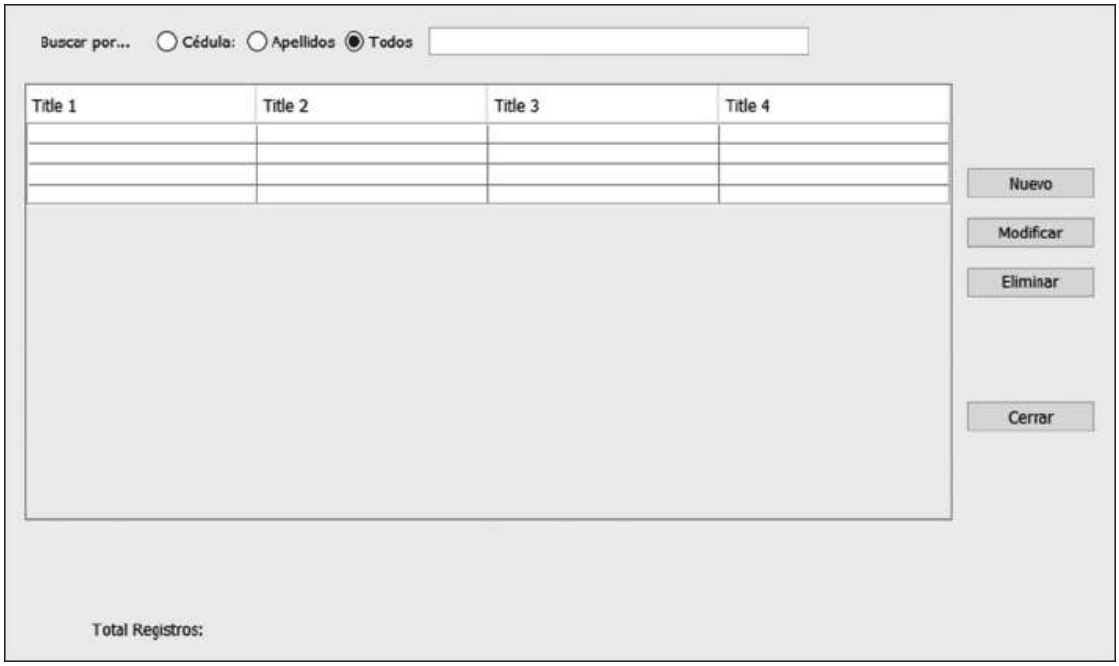


Figura 6.27 Diseño del formulario frmUsuarios

28. La **tabla 6.7** muestra la configuración de los componentes requeridos en `frmUsuarios`.

Tabla 6.7 Configuración de los controles de frmUsuarios

OBJETO	TEXT	NOMBRE (NAME)
JRadioButton	Cédula	rdCedula
JRadioButton	Apellidos	rdApellidos
JRadioButton	Todos	rdTodos
JTextField	Vacío	txtBusqueda
JLabel	Buscar por …	lblBuscar
JButton	Nuevo	btnNuevo
JButton	Modificar	btnModificar
JButton	Eliminar	btnEliminar
JButton	Cerrar	btnCerrar
JTable	No aplica	tablaUsuarios
JLabel	Total Registros	lblToltalRegistros
ButtonGroup	No aplica	buttonGroup1

29. Se debe recordar que el control `ButtonGroup` es un objeto que en la interfaz gráfica no es visible, por tanto cuando se agrega no se verá en el formulario. Su función es agrupar los `JRadioButton` para que trabajen como selección única.

30. Seleccionar cada uno de los 3 `JRadioButton` y en la propiedad `buttonGroup` elegir buttonGroup1.

31. Antes de continuar con la codificación, se pueden observar algunos consejos de manejo de NetBeans para la creación y configuración de los objetos que se generan en el formulario `frmUsuarios`.

Actividades para el lector

Algunos tips para trabajar el IDE de NetBeans al crear objetos:

- Para cambiar el texto de cualquier objeto pulsar sobre el mismo con el botón derecho del mouse y seleccionar Editar texto.
- Para cambiar el nombre de cualquier objeto pulsar sobre el mismo con el botón derecho del mouse y seleccionar Cambiar nombre de variable.
- Para crear objetos iguales, por ejemplo campos de texto, pulsar sobre el objeto con clic derecho del mouse y pulsar Duplicar en el menú de contexto.
- Para cambiar las propiedades del objeto, por ejemplo, el tamaño de letra, pulsar con el botón derecho del mouse y luego Propiedades en el menú de contexto

32. Ahora se agrega el código al formulario frmUsuario. Para esto, pasa de la vista de diseño (Design) a la de código fuente (Source) y se escribe el código mostrado en el listado 6.9

Listado No. 6.9 Programación de la clase frmUsuarios

```
1.  package Formularios;
2.  import Logica.Usuario;
3.  import javax.swing.JOptionPane;
4.  import javax.swing.table.DefaultTableModel;
5.  public class frmUsuarios extends javax.swing.JFrame {
6.      public DefaultTableModel modelo;
7.      public String[ ] registro = new String[7];
8.      Usuario usuario = new Usuario( );
9.      static public String valorColumna;
10.     final void mostrar( ) {
11.        try {
12.           modelo = usuario.listarUsuarios( );
13.           this.tablaUsuarios.setModel(modelo);
14.           lblTotalRegistros.setText("Total registros: " + Integer.to-
              String(modelo.getRowCount( )));
15.        } catch (Exception e) {
16.           JOptionPane.showConfirmDialog(rootPane, e);
17.        }
18.     }
19.     public frmUsuarios( ) {
20.        initComponents( );
21.        mostrar( );
22.     }
23.     @SuppressWarnings("unchecked")
24.     Generated Code
25.     private void txtBusquedaKeyReleased(java.awt.event.KeyEvent evt) {
26.        modelo = null;
27.        if (rdTodos.isSelected( )) {
28.           mostrar( );
29.        } else {
30.           if (rdApellidos.isSelected( )) {
```

```
31.              modelo = usuario.Filtrado(txtBusqueda.getText( ));
32.           }
33.           if (rdCedula.isSelected( )) {
34.              modelo = usuario.BuscarCedula(txtBusqueda.getText( ));
35.           }
36.           this.tablaUsuarios.setModel(modelo);
37.        }
38.        lblTotalRegistros.setText("Total registros: " + Integer.toString(-
           modelo.getRowCount( )));
39.     }
40.     private void btnNuevoActionPerformed(java.awt.event.ActionEvent evt)
        {
41.        frmNuevoUsuario frm = new frmNuevoUsuario(this, true);
42.        frm.setVisible(true);
43.     }
44.     private void btnModificarActionPerformed(java.awt.event.ActionEvent
        evt) {
45.        if (tablaUsuarios.getSelectedRow( ) != -1) {
46.           int fila = tablaUsuarios.getSelectedRow( );
47.           valorColumna = tablaUsuarios.getValueAt(fila, 0).toString( );
48.           frmEditarUsuario frm = new frmEditarUsuario(this, true);
49.           frm.setVisible(true);
50.        } else {
51.           JOptionPane.showMessageDialog(this, "Debe seleccionar un regis-
              tro");
52.        }
53.     }
54.     private void btnCerrarActionPerformed(java.awt.event.ActionEvent evt)
        {
55.        System.exit(0);
56.     }
57.     private void formWindowGainedFocus(java.awt.event.WindowEvent evt) {
58.        mostrar( );
59.     }
60.     private void btnEliminarActionPerformed(java.awt.event.ActionEvent
        evt) {
61.        if (tablaUsuarios.getSelectedRow( ) != -1) {
62.           int fila = tablaUsuarios.getSelectedRow( );
```

```
63.            valorColumna = tablaUsuarios.getValueAt(fila, 0).toString( );
64.            frmEliminarUsuario frm = new frmEliminarUsuario(this, true);
65.            frm.setVisible(true);
66.         } else {
67.            JOptionPane.showMessageDialog(this, “Debe seleccionar un regis-
               tro”);
68.         }
69.     }
70.     public static void main(String args[ ]) {
71.         java.awt.EventQueue.invokeLater(new Runnable( ) {
72.            public void run( ) {
73.               new frmUsuarios( ).setVisible(true);
74.            }
75.         });
76.     }
77. }
```

Comentarios sobre el código del listado 6.9:

- En la línea 10 se declara el método `mostrar( )`, el cual tiene la función de mostrar en pantalla y en la tabla todos los registros de Usuarios.

- En línea 25 se declara el evento `txtBusquedaKeyReleased( )`, el cual se activa cuando se ingresa texto en la caja de la búsqueda del formulario frmUsuarios. El método se invoca cada vez que se suelta una tecla, lo cual permite hacer una búsqueda dinámica que se actualiza con cada carácter introducido en la caja de texto. Los datos que se muestren en la tabla dependerán de la opción de RadioButton Cédula o Apellidos que se tenga seleccionada, por ejemplo, en la línea 28, si se seleccionan todos, se llamará al método `mostrar( )` declarado con anterioridad.

- Si no se despliegan todos los registros en las líneas 31 y 34, se tendrá la opción de filtrar la información a mostrar ya sea por apellidos o por cédula.

- Cuando se presiona el botón Nuevo en la línea 40, se invoca el evento `actionPerformed` del botón Nuevo, el cual crea una instancia del formulario `frmNuevoUsuario` (línea 41) y la hace visible en pantalla (línea 42).

- En la línea 44 se muestra el evento `actionPerformed` cuando se presiona el botón Modificar, el cual revisa si hay una fila de la tabla seleccionada; si existe alguna, busca ese registro en la base de datos y despliega en pantalla el formulario `frmEditarUsuario`

con la información del registro seleccionado. El usuario puede editar los campos del registro, excepto la cédula y guardar los cambios.

- En la línea 55 la instrucción `System.exit(0)` finaliza la ejecución del programa.

- En la línea 60 se muestra el evento `actionPerformed` del botón Eliminar, el cual revisa si hay una fila de la tabla seleccionada; si existe alguna, busca ese registro en la base de datos y despliega en pantalla el formulario frmEliminarUsuario con la información bloqueada del registro seleccionado. El usuario puede eliminar el registro completo si así lo desea.

33. Ahora se diseña la interfaz gráfica del formulario `frmNuevoUsuario`. Se verá similar a la de la **figura 6.28**.

Figura 6.28 Diseño del formulario `frmNuevoUsuario`

34. La **tabla 6.8** muestra la configuración de los componentes requeridos en `frmNuevoUsuario` .

Tabla 6.8 Configuración de los controles de frmNuevoUsuario

OBJETO	TEXT	NOMBRE (NAME)
jLabel	Cédula:	jLabel1
jLabel	Nombre:	jLabel2
jLabel	Apellidos:	jLabel3
jLabel	Dirección:	jLabel4
jLabel	Teléfono:	jLabel5
jLabel	Email:	jLabel6
jLabel	Fecha Ingreso:	jLabel7
JTextField	Nada	txtCedula
JTextField	Nada	txtNombre
JTextField	Nada	txtApellidos
JTextArea	Nada	txtDireccion
JTextField	Nada	txtTelefono
JTextField	Nada	txtEmail
DatePicker	No aplica	datePicker1
JButton	Guardar	btnGuardar
JButton	Cancelar	btnCancelar

35. Luego se agrega el código fuente al formulario `frmNuevoUsuario`. Para esto se pasa de la vista de diseño (Design) a la de código fuente (Source) y se escribe el código que se muestra en el listado 6.10:

Listado No. 6.10 Programación de la clase `frmNuevoUsuario`

```
1.  package Formularios;
2.  import Datos.DatosUsuario;
3.  import Logica.Usuario;
4.  import java.awt.event.KeyEvent;
5.  import java.sql.Date;
6.  import javax.swing.JOptionPane;
7.  import javax.swing.table.DefaultTableModel;
```

```
8.  public final class frmNuevoUsuario extends javax.swing.JDialog {
9.      DefaultTableModel modelo;
10.     public frmNuevoUsuario(java.awt.Frame parent, boolean modal) {
11.        super(parent, modal);
12.        initComponents( );
13.     }
14.     @SuppressWarnings("unchecked")
15.     Generated Code
16.     private void btnCancelarActionPerformed(java.awt.event.ActionEvent
        evt) {
17.        this.dispose( );
18.     }
19.     private void txtCedulaKeyPressed(java.awt.event.KeyEvent evt) {
20.        if ( evt.getKeyCode( ) == KeyEvent.VK_ENTER ) {
21.           if (txtCedula.getText( ).length( ) > 0) {
22.              Usuario usuario = new Usuario( );
23.              modelo = usuario.BuscarCedula(txtCedula.getText( ));
24.              for (int i = 0; i < modelo.getRowCount( ); i++) {
25.                 txtNombre.setText(modelo.getValueAt(i, 1).toString( ));
26.                 txtApellidos.setText(modelo.getValueAt(i, 2).toString( ));
27.                 txtDireccion.setText(modelo.getValueAt(i, 3).toString( ));
28.                 txtTelefono.setText(modelo.getValueAt(i, 4).toString( ));
29.                 txtEmail.setText(modelo.getValueAt(i, 5).toString( ));
30.                 String fecha = modelo.getValueAt(i, 6).toString( );
31.                 int longi = fecha.length( );
32.                 fecha = fecha.substring (8, longi) + "/" + fecha.
                    substring(5, longi - 3)
33.                                              + "/" + fecha.substring(0,
                    longi - 6);
34.                 datePicker1.setText(fecha);
35.              }
36.           }
37.        }
38.     }
39.     private void btnGuardarActionPerformed(java.awt.event.ActionEvent evt) {
```

```
40.         try {
41.            Usuario usuario = new Usuario( );
42.            DatosUsuario obj = new DatosUsuario( );
43.            obj.setCedula(txtCedula.getText( ));
44.            obj.setNombre(txtNombre.getText( ));
45.            obj.setApellidos(txtApellidos.getText( ));
46.            obj.setDireccion(txtDireccion.getText( ));
47.            obj.setTelefono(txtTelefono.getText( ));
48.            obj.setEmail(txtEmail.getText( ));
49.            obj.setFecha(Date.valueOf(datePicker1.getDate( )));
50.            if (usuario.insertar_Usuario(obj) == true) {
51.               JOptionPane.showMessageDialog(null, “Registro agregado!”);
52.            } else {
53.               JOptionPane.showMessageDialog(null, “Registro No
                  agregado!”);
54.            }
55.            this.dispose( );
56.         } catch (Exception e) {
57.            JOptionPane.showMessageDialog(null, e.getMessage( ));
58.         }
59.      }
60.
61.      public static void main(String args[ ]) {
62.         java.awt.EventQueue.invokeLater(new Runnable( ) {
63.            public void run( ) {
64.               frmNuevoUsuario dialog = new frmNuevoUsuario(new javax.
                  swing.JFrame( ), true);
65.               dialog.addWindowListener(new java.awt.event.WindowAdapter( ) {
66.                  @Override
67.                  public void windowClosing(java.awt.event.WindowEvent e) {
68.                     System.exit(0);
69.                  }
70.               });
71.               dialog.setVisible(true)
```

```
72.             }
73.         });
74.     }
75. }
```

Comentarios sobre el código del listado 6.10:

- En la línea 39 se declara el evento `actionPerformed` del botón Guardar, el cual permite insertar la información que el usuario ha introducido en el formulario de captura de datos.

- La línea 41 contiene la creación del objeto usuario de la clase Usuario, en la 42 se crea el objeto obj de la clase `DatosUsuario`.

- A partir de la línea 42 se setean los atributos del objeto obj con la información contenida en los componentes gráficos del formulario.

- En la línea 50 se invoca el método `insertar_Usuario( )`, el cual envía como parámetro al objeto obj construido con anterioridad para que sea insertado como un registro en la base de datos.

36. Ahora se diseña la interfaz gráfica del formulario `frmEditarUsuario`. La misma se verá similar a la de la anterior **figura 6.28**, sólo que se cambiará el título del borde del panel de Creación de un nuevo usuario por Editar usuario.

37. En la anterior **tabla 6.8** se muestra la configuración de los componentes requeridos en `frmNuevoUsuario`, utilizar la misma configuración para el formulario `frmEditarUsuario`.

38. Agregar el código fuente al formulario `frmEditarUsuario`. Para esto se pasa de la vista de diseño (Design) a la vista de código fuente (Source) y se escribe el código que se muestra en el listado 6.11.

Listado No. 6.11 Programación de la clase `frmEditarUsuario`

```
1.  package Formularios;
2.  import Datos.DatosUsuario;
3.  import Logica.Usuario;
4.  import java.sql.Date;
5.  import javax.swing.JOptionPane;
```

```
6.  import javax.swing.table.DefaultTableModel;
7.  public final class frmEditarUsuario extends javax.swing.JDialog {
8.     DefaultTableModel modeloEdicion;
9.     void cargarDatos( ) {
10.       frmUsuarios fr = new frmUsuarios( );
11.       String Cedula = fr.valorColumna;
12.       Usuario usuario = new Usuario( );
13.       modeloEdicion = usuario.BuscarCedula(Cedula);
14.       int NumFilas = modeloEdicion.getRowCount( );
15.       for (int i = 0; i < NumFilas; i++) {
16.          txtCedula.setText(Cedula);
17.          txtCedula.setEnabled(false);
18.          txtNombre.setText(modeloEdicion.getValueAt(i, 1).toString( ));
19.          txtApellidos.setText(modeloEdicion.getValueAt(i, 2).toString( ));
20.          txtDireccion.setText(modeloEdicion.getValueAt(i, 3).toString( ));
21.          txtTelefono.setText(modeloEdicion.getValueAt(i, 4).toString( ));
22.          txtEmail.setText(modeloEdicion.getValueAt(i, 5).toString( ));
23.          String fecha = modeloEdicion.getValueAt(i, 6).toString( );
24.          int longi = fecha.length( );
25.          fecha = fecha.substring(8, longi) + "/" + fecha.substring(5,
             longi - 3)
26.                               + "/" + fecha.substring(0, longi - 6);
27.          datePicker1.setText(fecha);
28.       }
29.    }
30.    public frmEditarUsuario(java.awt.Frame parent, boolean modal) {
31.       super(parent, modal);
32.       initComponents( );
33.       cargarDatos( );
34.    }
35.    @SuppressWarnings("unchecked")
36.    Generated Code
37.    private void btnCancelarActionPerformed(java.awt.event.ActionEvent evt)
       {
```

```
38.       this.dispose();
39.    }
40.    private void btnGuardarActionPerformed(java.awt.event.ActionEvent evt)
       {
41.       try {
42.          Usuario usuario = new Usuario( );
43.          DatosUsuario obj = new DatosUsuario( );
44.          obj.setCedula(txtCedula.getText( ));
45.          obj.setNombre(txtNombre.getText( ));
46.          obj.setApellidos(txtApellidos.getText( ));
47.          obj.setDireccion(txtDireccion.getText( ));
48.          obj.setTelefono(txtTelefono.getText( ));
49.          obj.setEmail(txtEmail.getText( ));
50.          obj.setFecha(Date.valueOf(datePicker1.getDate( )));
51.          if (usuario.editar_Usuario(obj) == true) {
52.             JOptionPane.showMessageDialog(null, “Registro actualizado!”);
53.          } else {
54.             JOptionPane.showMessageDialog(null, “Registro No
                actualizado!”);
55.          }
56.          this.dispose( );
57.        } catch (Exception e) {
58.           JOptionPane.showMessageDialog(null, e.getMessage( ));
59.        }
60.     }
61.     public static void main(String args[ ]) {
62.        java.awt.EventQueue.invokeLater(new Runnable( ) {
63.           public void run( ) {
64.              frmEditarUsuario dialog = new frmEditarUsuario(new javax.
                  swing.JFrame( ), true);
65.              dialog.addWindowListener(new java.awt.event.WindowAdapter( ) {
66.                 @Override
67.                 public void windowClosing(java.awt.event.WindowEvent e) {
68.                    System.exit(0);
69.                 }
```

```
70.              });
71.              dialog.setVisible(true)
72.           }
73.        });
74.     }
75. }
```

Comentarios sobre el código del listado 6.11:

- En la línea 9 se declara el método `cargarDatos( )`, el encargado de visualizar en el formulario el registro seleccionado para ser modificado.

- En la línea 40 se encuentra el botón `actionPerformed` del botón Guardar, el cual crea el objeto obj, lo setea con la información actualizada del usuario y luego guarda el registro en la base de datos con los cambios realizados.

39. Se debe diseñar la interfaz gráfica del formulario `frmEliminarUsuario`. La misma se verá similar a la de la anterior **figura 6.28**, sólo que cambiaremos el título del borde del panel de Creación de un nuevo usuario por Eliminar usuario, además, se debe cambiar el texto del botón Guardar por Eliminar.

40. En la anterior tabla 6.8 se muestra la configuración de los componentes requeridos en `frmNuevoUsuario`, se debe utilizar la misma configuración para el formulario `frmEliminarUsuario`.

41. Ahora se agrega el código fuente al formulario `frmEliminarUsuario`. Para esto se pasa de la vista de diseño (Design) a la vista de código fuente (Source) y se escribe el código que se muestra en el listado 6.12.

Listado No. 6.12 Programación de la clase `frmEliminarUsuario`

```
1.  package Formularios;
2.  import Logica.Usuario;
3.  import javax.swing.JOptionPane;
4.  import javax.swing.table.DefaultTableModel;
5.  public final class frmEliminarUsuario extends javax.swing.JDialog {
6.     DefaultTableModel modeloEdicion;
7.     void cargarDatos( ) {
```

```
8.        frmUsuarios fr = new frmUsuarios( );
9.        String Cedula = fr.valorColumna;
10.       Usuario usuario = new Usuario( );
11.       modeloEdicion = usuario.BuscarCedula(Cedula);
12.       int NumFilas = modeloEdicion.getRowCount( );
13.       for (int i = 0; i < NumFilas; i++) {
14.          txtCedula.setText(Cedula);
15.          txtNombre.setText(modeloEdicion.getValueAt(i, 1).toString( ));
16.          txtApellidos.setText(modeloEdicion.getValueAt(i, 2).toString( ));
17.          txtDireccion.setText(modeloEdicion.getValueAt(i, 3).toString( ));
18.          txtTelefono.setText(modeloEdicion.getValueAt(i, 4).toString( ));
19.          txtEmail.setText(modeloEdicion.getValueAt(i, 5).toString( ));
20.          String fecha = modeloEdicion.getValueAt(i, 6).toString( );
21.          int longi = fecha.length( );
22.          fecha = fecha.substring(8, longi) + "/" + fecha.substring(5,
             longi - 3)
                                        + "/" + fecha.substring(0, longi - 6);
23.          datePicker1.setText(fecha);
24.          datePicker1.setEnabled(false);
25.          txtCedula.setEnabled(false);
26.          txtNombre.setEnabled(false);
27.          txtApellidos.setEnabled(false);
28.          txtDireccion.setEnabled(false);
29.          txtTelefono.setEnabled(false);
30.          txtEmail.setEnabled(false);
31.       }
32.    }
33.    public frmEliminarUsuario(java.awt.Frame parent, boolean modal) {
34.       super(parent, modal);
35.       initComponents( );
36.       cargarDatos( );
37.    }
38.    @SuppressWarnings("unchecked")
39.    Generated Code
```

```
40.    private void btnCancelarActionPerformed(java.awt.event.ActionEvent evt) {
41.       this.dispose( );
42.    }
43.    private void btnEliminarActionPerformed(java.awt.event.ActionEvent evt) {
44.       try {
45.          Usuario usuario = new Usuario( );
46.          if (usuario.eliminar_Usuario(txtCedula.getText( )) == true) {
47.             JOptionPane.showMessageDialog(null, “Registro eliminado!”);
48.          } else {
49.             JOptionPane.showMessageDialog(null, “Registro No eliminado!”);
50.          }
51.          this.dispose( );
52.       } catch (Exception e) {
53.          JOptionPane.showMessageDialog(null, e.getMessage( ));
54.       }
55.    }
56.
57.     public static void main(String args[ ]) {
58.        java.awt.EventQueue.invokeLater(new Runnable( ) {
59.           public void run( ) {
60.              frmEliminarUsuario dialog = new frmEliminarUsuario(new
                  javax.swing.JFrame( ), true);
61.              dialog.addWindowListener(new java.awt.event.WindowAdapter( ) {
62.                 @Override
63.                 public void windowClosing(java.awt.event.WindowEvent e) {
64.                    System.exit(0);
65.                 }
66.              });
67.              dialog.setVisible(true)
68.           }
69.        });
70.     }
71. }
```

Comentarios sobre el código del listado 6.12:

- En la línea 43 se tiene el evento `actionPerformed` del botón Eliminar, el cual procederá a eliminar el registro que se muestra en el formulario `frmEliminarUsuario` y que previamente fue seleccionado para ser borrado.

Luego de haber diseñado y codificado la aplicación dskTalleres, es preciso observar las diferentes pantallas en ejecución del sistema.

La **figura 6.29** muestra la pantalla en ejecución del formulario JFrame `frmUsuarios`, la cual es la principal de la aplicación donde se observan todos los registros insertados en la tabla Usuario de la base de datos, además, se puede buscar por medio de un filtro de cédula o apellidos. Por otro lado, el botón Nuevo permite agregar un nuevo registro a la base de datos, mientras que si se selecciona alguno de los registros desplegados en la tabla es posible mediante los botones restantes eliminar o editar un registro ya existente.

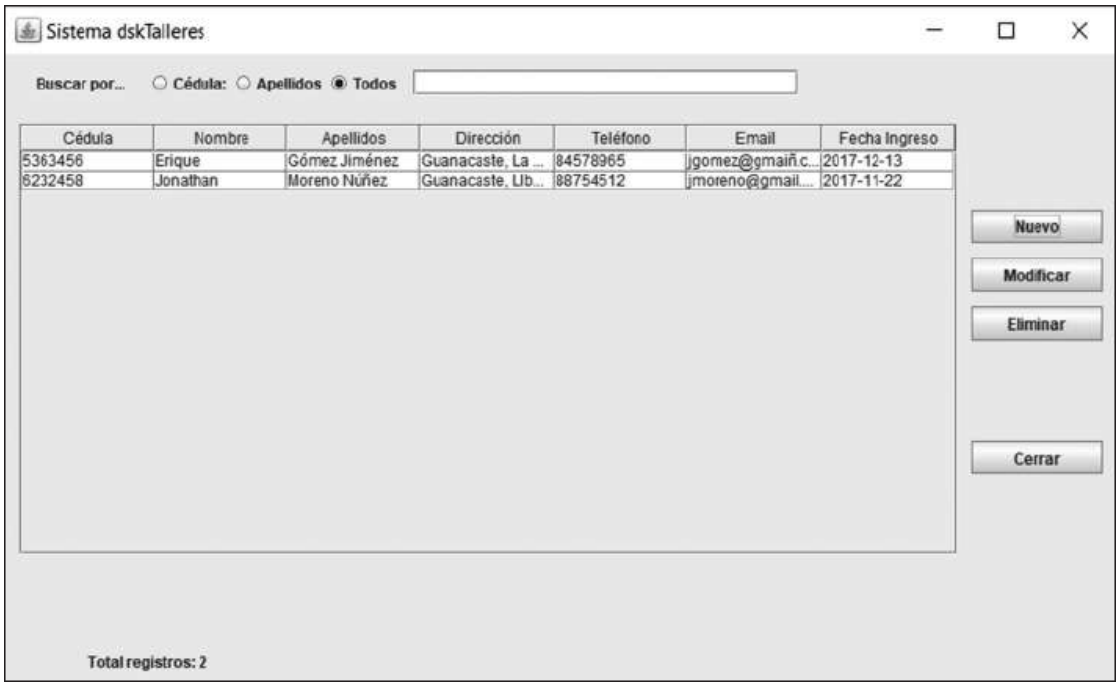


Figura 6.29 Formulario frmUsuario en ejecución

Enseguida, en la **figura 6.30** se muestra la pantalla `frmNuevoUsuario` en ejecución, en la cual se puede agregar un nuevo registro.

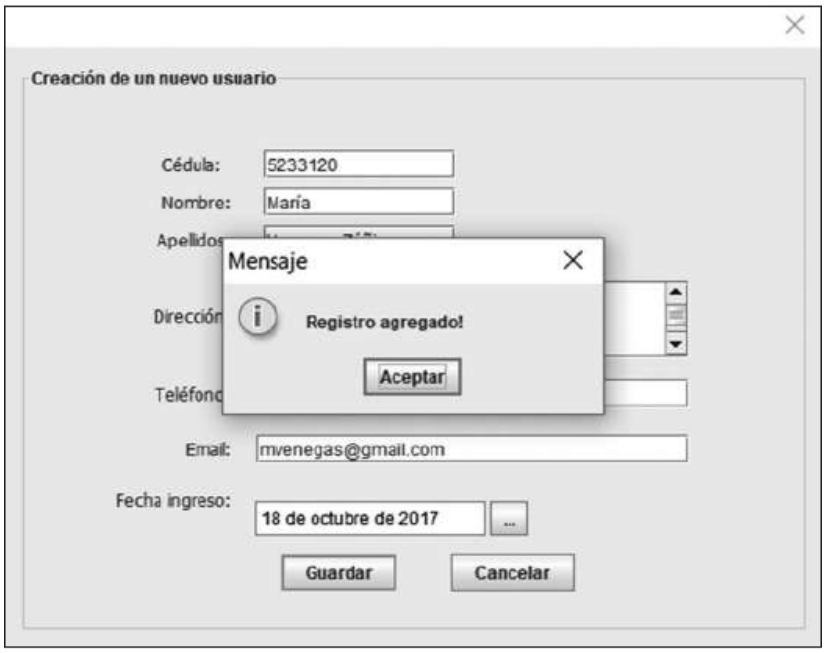


Figura 6.30 Formulario `frmNuevoUsuario` en ejecución

En la siguiente figura se puede apreciar el formulario `frmEditarUsuario` en ejecución. En este formulario es posible editar todos los campos excepto la cédula, debido a que es la clave primaria de la tabla.

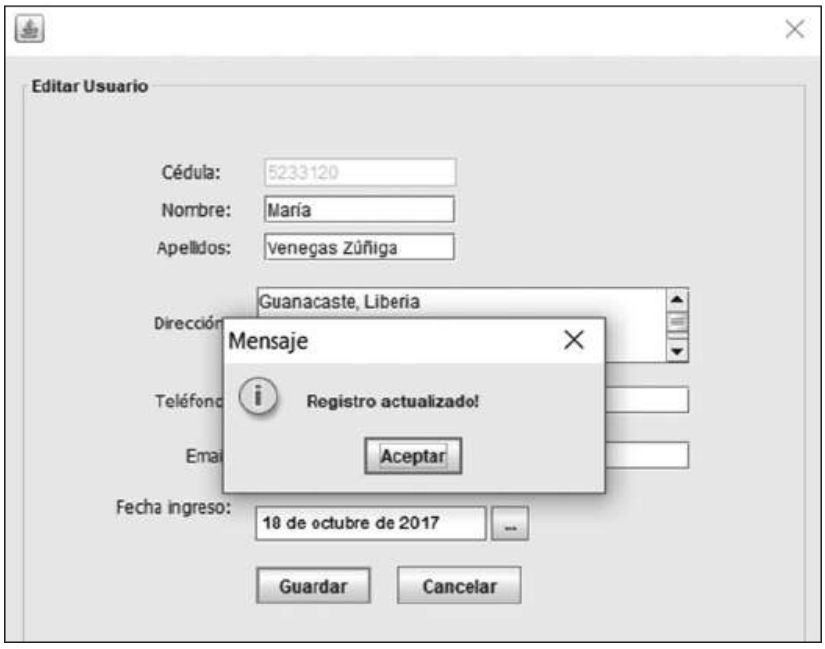


Figura 6.31 Formulario `frmEditarUsuario` en ejecución

Por último, en la **figura 6.32** se puede visualizar el formulario `frmEliminarUsuario`, en el cual es posible eliminar un registro de la base de datos previamente seleccionado.

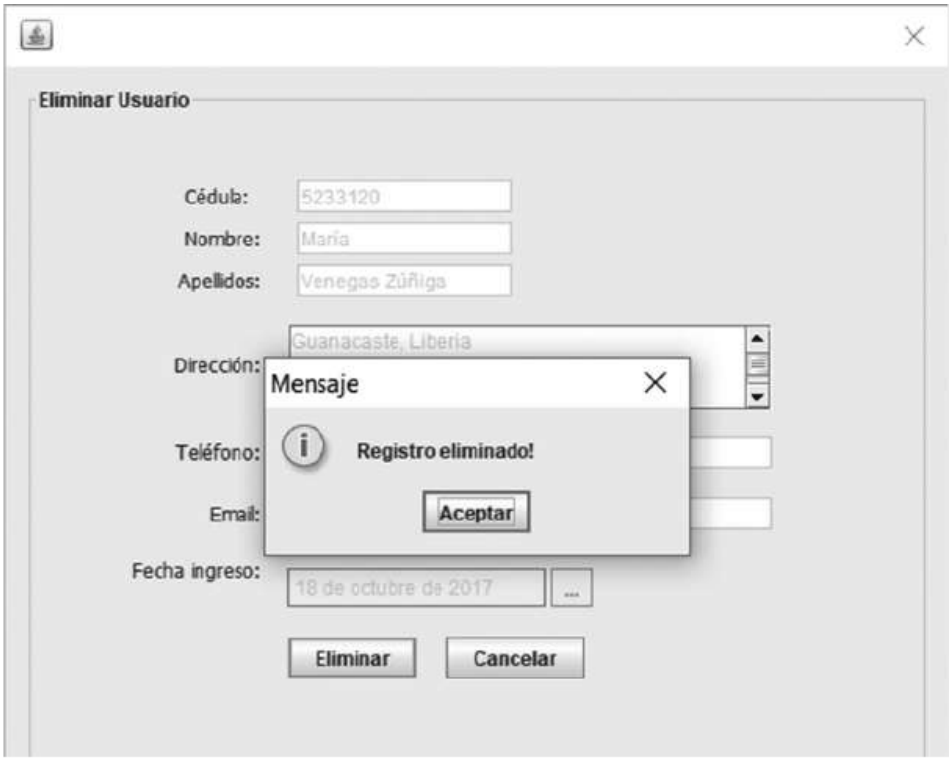


Figura 6.32 Formulario `frmEliminarUsuario` en ejecución

De esta manera, se concluye con el ejemplo práctico programado de la aplicación dskTalleres, la cual se conecta a una base de datos MySql haciendo uso de una librería que permite esta conexión; por otro lado, los formularios de la aplicación son las pantallas que interactúan con el usuario final y son las que se comunican con la clase Usuario.java, quien a su vez posee todos los métodos donde se utiliza el lenguaje LMD de SQL para realizar el CRUD por medio de los comandos SELECT, INSERT, UPDATE y DELETE.

Resumen

En este capítulo se desarrolló el tema de la gestión de una base de datos MySQL con Java, con el uso del IDE NetBeans. Se describió la parte conceptual de la arquitectura Java para la gestión de datos y cómo se implementa en MySQL, así como una breve introducción de consultas en SQL. Básicamente se trataron los siguientes temas:

- La instalación de MySQL como back-end y dbforgemysqlfree como front-end.
- La creación de una base de datos con el front-end y el back-end.

- Introducción a SQL.
- La gestión de datos con Java-NetBeans en base de datos MySQL.
- Arquitectura JDBC.
- Conectividad a una base de datos MySql con Java-NetBeans.
- Ejemplo de una aplicación CRUD de usuarios.

Evidencia

a) Implementar los ejemplos de `ResulsetMetaData` y `DataBaseMetaData` explicados en http://www.chuidiang.com/java/mysql/ResultSet-DataBase-MetaData.php

b) Desarrollar un aplicativo que brinde mantenimiento a una base de datos denominada *dbPresupuesto*. Las tablas que se deberán mantener son *tbCuentas* y *tbMovimientos*.

c) Crear las consultas adecuadas para el caso del inciso b.

d) Elaborar un procedimiento almacenado en la base de datos del inciso b y un formulario que permita su utilización.

Preguntas de autoevaluación

1. ¿A qué se denomina back-end? Citar un ejemplo.

2. ¿Qué es un front-end? Citar 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?

Respuestas a las preguntas de 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 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 de permanecer en la memoria de la computadora.

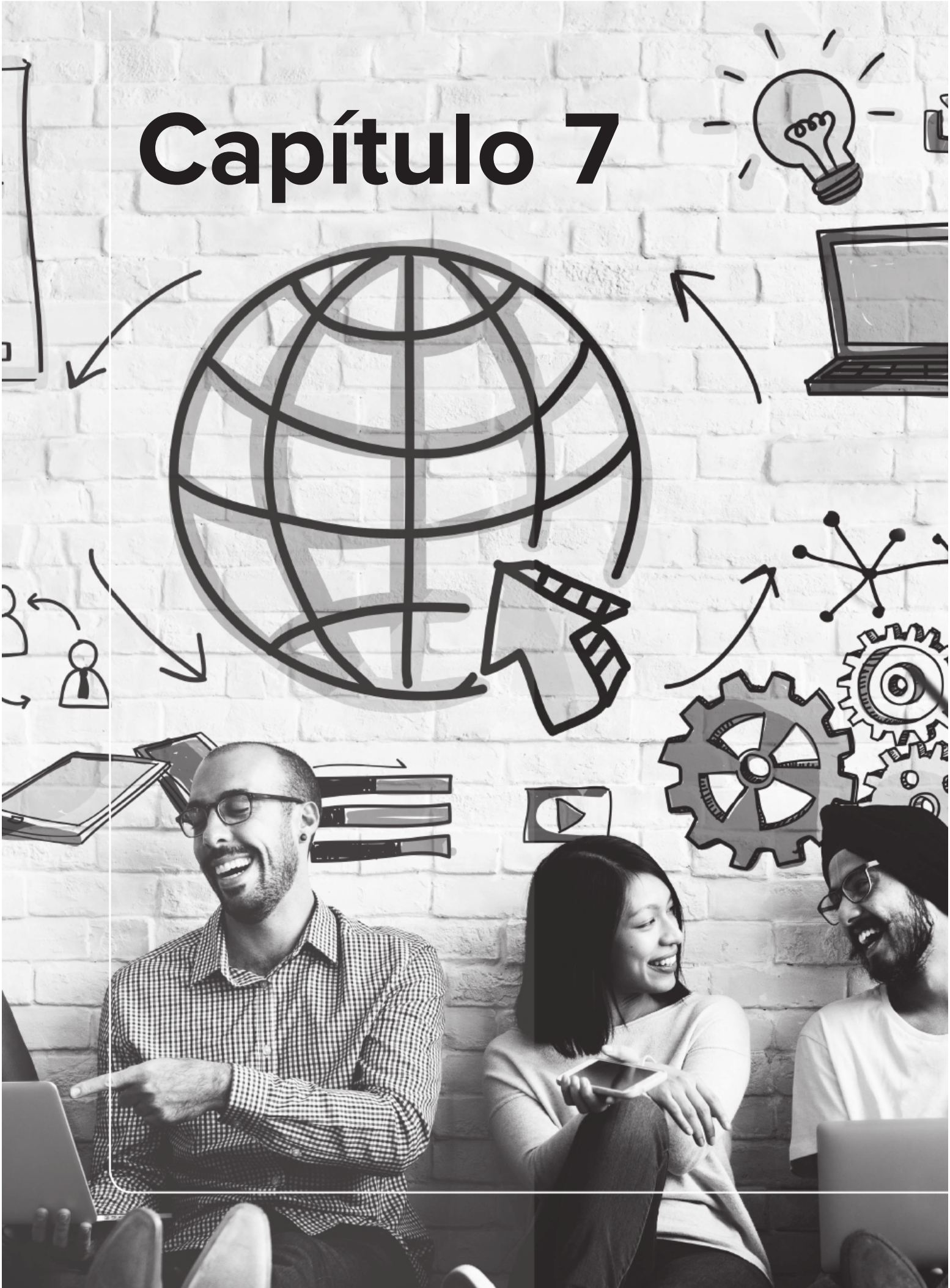
4. Es el mecanismo de preservación de la integridad referencial de los objetos.

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 la base de datos.

Capítulo 7



Introducción al front-end de una aplicación web Java

Reflexione y responda
las siguientes preguntas

¿Existe actualmente una división entre quienes desarrollan front-end y quienes lo hacen para back-end?

¿Cuánto se conoce de front-end en los desarrollos web?

¿Cómo han incidido las herramientas para diseño de front-end en el uso de la responsividad de las aplicaciones web?

Contenido

7.1 Introducción
- **7.1.2** Front-end

7.2 HTML
- **7.2.1** HTML 5
- **7.2.2** Canvas
- **7.2.3** Video y audio en HTML 5
- **7.2.4** Almacenamiento local y aplicaciones fuera de línea
- **7.2.5** Mejoras en formularios web

7.3 CSS3

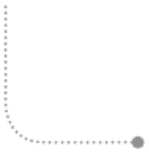
Expectativa

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 orientadas a los negocios, a la información o al mundo del entretenimiento, hacen que se convierta en un 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 de software 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 este desarrollo, sino son cada día más personas de distintas disciplinas las que participan de esta creatividad.

Para iniciar en el mundo de la programación web con Java, es necesario primero conocer algunos conceptos implícitos como HTML, CSS y JavaScript (o alguna librería similar). El tema de Javascript no se tratará en este libro.



Después de estudiar este capítulo, el lector será capaz de:

- Definir front-end y su importancia en el diseño de sitios web.
- Comprender la fundamentación de HTML 5 y CSS3 en el diseño de sitios web.
- Aplicar HTML 5 y CSS3 en el diseño de una página web.

7.1 Introducción

El mundo fascinante de Internet y todos los beneficios que depara su uso oculta a veces la complejidad detrás de una interfaz bonita y segura. Sitios web de altas prestaciones gráficas o poderosas aplicaciones de transacciones en línea facilitan muchas veces la vida al hacer solamente un clic sobre botones y selectores, pero internamente pueden esconder una impresionante complejidad de programación.

Existen varios factores que colaboran para que esta trama 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 la programación web orientados hacia el front-end. Es decir, a explicar los principios que rigen el diseño de las interfaces que el usuario observa cuando ingresa a un sitio web, así como las herramientas que permiten esta interacción.

7.1.2 Front-end

En tecnologías web las herramientas front-end son todas aquellas que se ejecutan del lado del cliente, es decir, en los navegadores de Internet. Son tres lenguajes los que hacen posible el diseño básico de las interfaces de un sitio web: HTML, CSS y JavaScript. Mediante estos lenguajes es posible diseñar las pantallas o interfaces requeridas en el navegador o explorador web que el usuario utiliza para realizar sus procesos y resolver sus necesidades.

Para extender la funcionalidad de estos tres lenguajes existen herramientas que posibilitan crear las interfaces de una forma más rápida y fácil. Entre ellas Bootstrap (https://getbootstrap.com/), Foundation (https://foundation.zurb.com/), Pure (https://purecss.io/), ink (http://ink.sapo.pt/), Metro UI CSS (https://metroui.org.ua/), HTML KickStart (http://www.99lime.com/elements/), como maquetadores. Estas herramientas permiten manejar la parte estática del diseño (HTML y CSS). Para el manejo dinámico, se tiene JavaScript y algunas herramientas similares como: AngularJS (https://angularjs.org/), ReactJS (https://reactjs.org/), NodeJS (https://nodejs.org/en/), entre otras.

Actividades para el lector

Para mayor más información acerca de herramientas front-end puedes visitar:

- http://chuwiki.chuidiang.org/index.php?title=Lectura_y_Escritura_de_Ficheros_en_Java
- https://www.1and1.es/digitalguide/paginas-web/desarrollo-web/5-alternativas-a-bootstrap-el-framework-de-twitter/
- http://blog.aulaformativa.com/framework-responsive-alternativas-a-bootstrap/
- https://hipertextual.com/archivo/2014/07/alternativas-twitter-bootstrap/

7.2 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.

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 puede representar un enlace a otro documento. Por tanto, no representa un concepto lineal ni secuencial en cuanto a su visualización.

Un documento hipermedia incluye además de texto; imágenes, audio y video (es como incluir la multimedia en los documentos de hipertexto). Actualmente las páginas web constituyen este tipo de documentos, lo cual, en conjunto, crean lo que denomina un sitio o un aplicativo web.

Para crear estas páginas de hipertexto o hipermedia se utilizará HTML como estándar. Existen diversos 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 respecto del 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 (Mozilla, Google Chrome, Microsoft Edge, entre otros) mediante el uso de marcas (etiquetas) o tags. Estos tags son normalizados internacionalmente, aunque algunos fabricantes de navegadores han incluido algunas nuevas capacidades que incompatibilizan con el resto.

7.2.1 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 la utilización abusiva 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. La nomenclatura HTML 5 implementa una estructura semántica a una página web, donde diversas secciones tienen un significado muy directo. La **figura 7.1** muestra la estructura semántica que debe tener un documento HTML 5.

Figura 7.1 Estructura semántica de un documento HTML 5

La **tabla 7.1** describe la funcionalidad de estos elementos semánticos HTML5.

Tabla 7.1 Algunos nuevos elementos semánticos en HTML 5

ELEMENTO SEMÁNTICO	DESCRIPCIÓN
<header>	Elemento utilizado para la definición del encabezado de la página web. Puede corresponder a la página completa o a 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 permite 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, 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 actualizan cada momento, sin necesidad de 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 a colocar a un lado de la página HTML podrían diseñarse mediante este elemento.
<hgroup>	Se usa para crear un grupo de encabezados: por ejemplo, un título y un subtítulo de una sección o una página HTML.

Puede descargarse, ver y analizar el código que demuestre el uso de HTML5 y CSS3 desde:

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

7.2.2 Canvas

Canvas es un elemento en HTML 5 orientado a la creación de dibujos en 2D para sitios 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 gráfico imaginable cuando se combina con el poder de JavaScript, entre otras API. 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

7.2.3 Video y audio en HTML 5

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 ambos en las páginas, 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 códecs de audio.

7.2.4 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 limitaciones de estas cookies, por ejemplo la capacidad de almacenamiento de alguna, el número máximo de cookies que un navegador puede mantener o el desperdicio de recursos que significa enviarlas con cada request (solicitud) que se haga al servidor.

HTML 5 ofrece una serie de API que permite resolver este problema. Básicamente, consiste en un almacenamiento local que lo separa del documento HTML 5. La información se alma-

cena 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 a 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 se caiga o se requiera trabajar en forma asincrónica. Lógicamente, esto es 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 se ejecute en el servidor, una vez que nuevamente se reconecte el aplicativo al servidor.

7.2.5 Mejoras en formularios web

Con HTML 4 se pueden crear controles de formularios 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 usar algunos de los productos comerciales conocidos actualmente, los cuales ofrecen esta funcionalidad, tal como Flex de Adobe, Silverlight de Microsoft o JavaFX presente en NetBeans/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 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 basado en casilla de texto en el momento en 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 muestra la interfaz y código creado desde:

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

Creación del primer documento HTML

Para entender el funcionamiento de los archivos HTML en NetBeans se desarrollará un ejemplo sencillo. Mediante su elaboración se explicarán algunos componentes de HTML5.

1. Ingresar a NetBeans.

2. Seleccionar Proyecto Nuevo desde el menú Archivo, o presionar CTRL+Mayúscula+N al mismo tiempo.

3. Seleccionar HTML5/JavaScript y HTML5/JS Application, como se muestra en la **figura 7.2**, y pulsar el botón Siguiente.

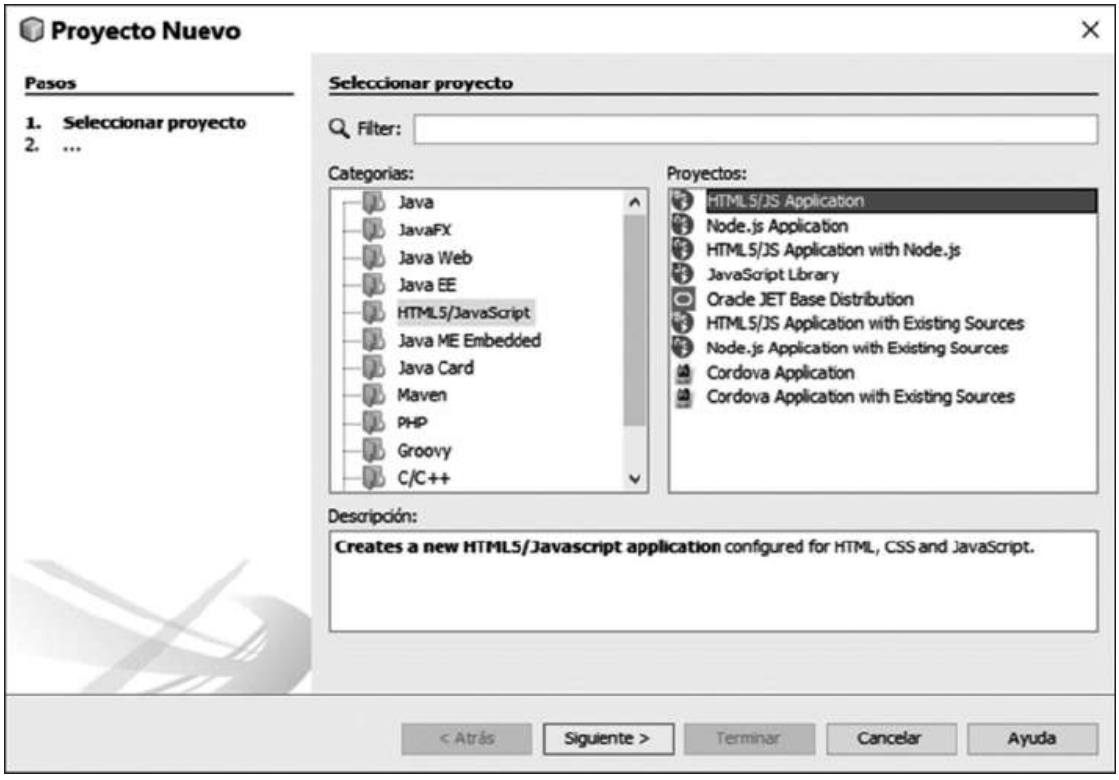


Figura 7.2 Creación de proyecto HTML5/JavaScript

4. Nombrar este proyecto como *HTML5Proyecto* en la pantalla que aparece a continuación. También se debe seleccionar una ubicación adecuada.

5. En la pantalla siguiente aparece la oportunidad de seleccionar un template o plantilla que se puede utilizar para el diseño. Estas plantillas pueden ser templates de Bootstrap, Angular o BoilerPlate de tipo responsivo (significa que tiene la capacidad de adaptar las dimensiones de contenido a distintos dispositivos con tamaños diversos de pantallas). Como no se tienen plantillas en este momento, se seleccionará la opción No site Template, como se muestra en la **figura 7.3**.

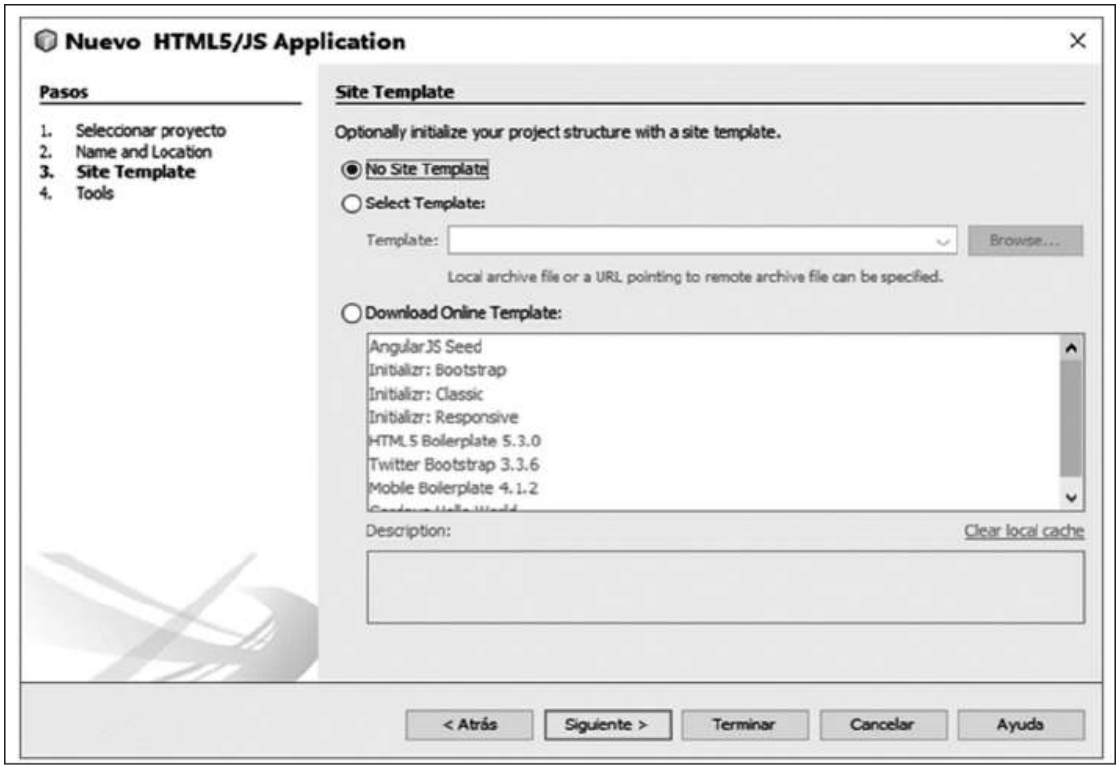


Figura 7.3 Selección de un template para HTML5/JavaScript

6. Una vez que se da clic en el botón Siguiente aparecerá una ventana con las opciones de creación de algunos archivos `json js`. Hacer caso omiso y pulsar el botón Finalizar. Después se indicará para qué son estos archivos.

7. Cerrar los archivos que aparecen abiertos en el proyecto. Sólo se debe dejar abierto el archivo `index.html`, el cual interesa programar en este momento. La **figura 7.4** muestra el código inicial de este archivo.

```
<!DOCTYPE html>
<!--
To change this license header, choose License Headers in Project Properties.
To change this template file, choose Tools | Templates
and open the template in the editor.
-->
<html>
    <head>
        <title>TODO supply a title</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
        <div>TODO write content</div>
    </body>
</html>
```

Figura 7.4 Código inicial de archivo index.html en el proyecto HTML5Proyecto

8. Es preciso asegurarse de que los archivos queden en la carpeta HTML (ver **figura 7.5**), donde se copiará el archivo index.html. Posteriormente, se creará una carpeta denominada `Imagenes`. En esta última se procede a descargar una imagen referente a un mall (centro comercial).

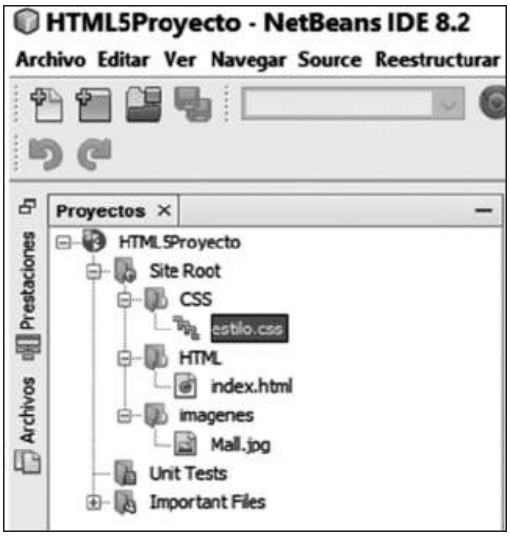


Figura 7.5 Estructura del proyecto *HTML5Proyecto*

9. Ahora se procede a modificar el archivo index.html para que sea similar al mostrado en el listado 7.1.

Listado No. 7.1 Código de la primera página HTML 5: index.html

```
1.  <!DOCTYPE html>
2.  <html lang="es">
3.      <head>
4.          <meta charset="UTF-8">
5.          <meta name="description" content="Ejemplo HTML 5">
6.          <meta name="keywords" content="HTML 5, CSS3, JavaScript">
7.          <meta name="author" content="Enrique Gómez | Jonathan Moreno">
8.          <meta name="robots" content="noindex, nofollow">
9.          <title>Nuestro City Mall</title>
10.         <meta name="viewport" content="width=device-width,
11.         user-scalable=no, initial-scale=1.0, maximum-scale=1.0,
12.         minimum-scale=1.0">
13.     </head>
```

```
14. <body>
15.     <nav>
16.       <ul>
17.         <li><a href="index.html">Inicio</a></li>
18.         <li><a href="#">Catálogo de tiendas</a></li>
19.         <li><a href="">Servicios</a></li>
20.         <li><a href="">Convenios comerciales</a></li>
21.         <li><a href="contactenos.html">Contáctenos</a></li>
22.       </ul>
23.     </nav>
24.  <div>
25.       <div>
26.                           </>
27.  </div>
28.                   <footer>Derechos Reservados &copy; 2016-2020</footer>
29.       </div>
30.     </body>
31. </html>
```

Comentarios sobre el código del listado 7.1:

- La línea 1 indica con el `<!DOCTYPE>` que es un documento HTML.

- La línea 2 indica que el idioma predeterminado para la página es el español: `<html lang="es">`

- De la línea 3 a la 13 está la sección `header`, que en realidad es la cabecera del documento HTML y que no se muestra en el explorador web. Dentro de esta sección se encuentra una serie de etiquetas que se resumen así:

  a) `<meta charset="UTF-8">`: Se utiliza para especificar la codificación usada en la página. En el caso de UTF-8 permitirá que letras acentuadas como á, é, í, ó, ú o el carácter especial ñ aparezcan correctamente en los despliegues del navegador web.

  b) `<meta name=»description» content=»Ejemplo HTML 5»>`: Con este metadato se pretende crear una pequeña descripción de la página web que se pretende desplegar. Será importante para los directorios de páginas y los buscadores web.

c) `<meta name=»keywords» content=»HTML 5, CSS3, JavaScript»>`: Es un conjunto de palabras que muestran la relevancia de la página web. Son palabras claves que los buscadores e indexadores web pueden utilizar para realizar las búsquedas.

d) `<meta name=»author» content=»Enrique Gómez | Jonathan Moreno»>`: Con esta etiqueta se pretende determinar la autoría del o los diseñadores de la página web.

e) `<meta name=»robots» content=»noindex, nofollow»>`: Este metadato permite definir el comportamiento que los robots tendrán en la página. Los valores que permiten el content se pueden separar por comas y tendrán diferentes significados. En este caso, noindex significa que el robot no indexe la página, nofollow que el robot no siga los enlaces de la página.

f) `<title>Nuestro City Mall</title>`: Indica el título que aparecerá en la cabecera del navegador, identificando la página actual.

g) `<meta name=»viewport»…>`: Permite establecer el modo responsive de una página web (es decir, que indique la disponibilidad para visualización en cualquier tipo de pantalla). Los valores incluidos en este caso son width que define el ancho en pixeles del viewport; initial-scale, define el ratio entre el ancho del dispositivo y el tamaño del viewport (sus valores están entre 0.0 y 10.0); minimum-scale, permite definir el valor máximo del zoom, el cual debe ser mayor o igual a maximum-scale.

- De la línea 14 a la 30 se tiene el elemento body el cual esencialmente es el cuerpo del documento HTML.

- En las líneas 15 a 23 se tiene la sección nav (propia de HTML5) que permite establecer la zona de navegación de la página web.

- Las líneas 16 a 22 permiten la creación de una lista desordenada HTML (Unordered List, por sus siglas en inglés). Posteriormente, en las líneas 17 a 21 se crean los ítems de la lista (UL) mediante li (list items).

- En la línea 28 se tiene el footer que es el pie de la página web.

10. Ahora se procede a ejecutar lo que se tiene hasta ahora. Pulsar sobre el archivo `index.html` (**figura 7.5**). Se mostrará algo similar a la **figura 7.6**:

Se debe continuar con la estructura de la página web `index.html`. Esta vez se agrega la imagen a utilizar en la página.

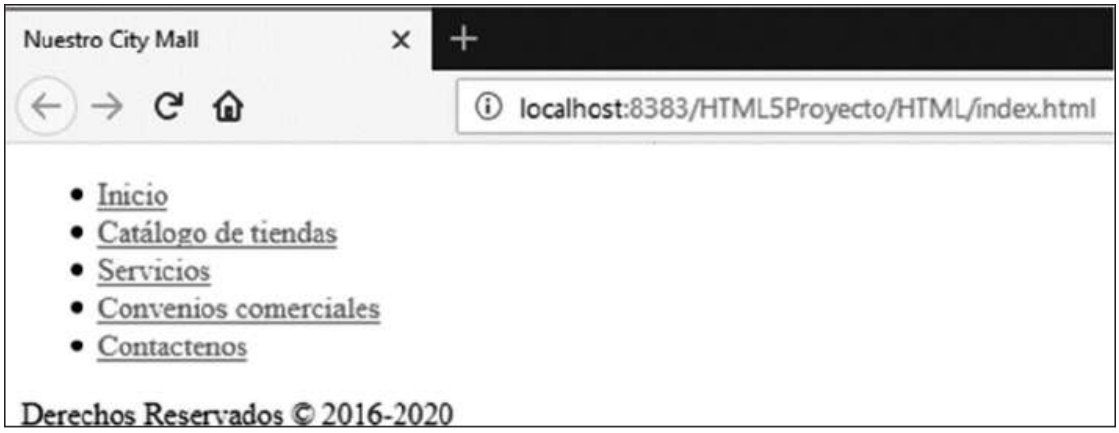


Figura 7.6 Ejecución del archivo `index.html` del proyecto *HTML5Proyecto*

Figura 7.7 Colocar la imagen *Mall*.jpg en el archivo `index.html`

Como es posible notar en la **figura 7.6**, la página web no tiene formato. Constituye solamente una representación semántica de la página. El siguiente paso es crear algunos archivos css (Cascade Sheet Style) para darle el diseño requerido por la página.

7.3 CSS3

Se ha indicado que HTML permite crear la estructura semántica de una página web. Sin embargo, para darle el diseño adecuado: disposición, color, tipo de letra, tamaños de letra, entre otros, se requiere el uso de CSS.

Se puede afirmar que CSS es un lenguaje que describe el estilo que tendrá un documento HTML. Esto servirá para determinar cómo los elementos HTML serán desplegados en un navegador.

Este capítulo no pretende ser un texto completo de HTML, CSS o JavaScript, más bien es una introducción corta a estos temas dado que serán base para la creación de un sitio web en Java. Lo que se genere se documentará resumida y adecuadamente a este nivel. Al final del libro se dan referencias web recomendadas, donde se podrán estudiar con más detalle estos temas. Como una referencia rápida se aporta la **tabla 7.2** sobre algunas propiedades elementales de los selectores de CSS3 que permitirán formarse una idea sobre su funcionalidad.

Tabla 7.2 Algunas propiedades de selectores en CSS3

PROPIEDAD	DESCRIPCIÓN
display	Permite establecer el tipo de caja que el navegador empleará para mostrar un elemento, los tipos más comunes son inline y block, aunque existen otros.
margin	Establece los márgenes de un elemento especificado entre uno y cuatro valores, donde cada valor puede ser una longitud, un porcentaje o auto. Los cuatro valores se refieren respectivamente a los márgenes superior, derecho, inferior e izquierdo.
border	Es la forma rápida de establecer el ancho, estilo y color de todos los bordes de un elemento al mismo tiempo.
padding	Establece las anchuras (márgenes) de algunas o de todas las zonas de relleno de los elementos.
box-sizing	Ayuda a definir mejor el ancho y el alto de la caja, dando la orden al navegador si se pretende que el ancho y el alto incluyan o no al padding, al border, ya que modifica los valores por defecto del CSS box-model que calculan el ancho y alto.
background	Permite establecer de forma abreviada el valor de una o más propiedades individuales. En concreto, background permite establecer simultáneamente las cinco propiedades relacionadas con el color e imagen de fondo de cada elemento, entre ellas su posición; si la imagen es fija o no y si se repite o no.
height	Establece la altura de los elementos de un bloque, puede ser representado por sólo los siguientes valores: medida, porcentaje, auto e inherit.
width	Establece la anchura de los elementos de un bloque, puede ser representado por sólo los siguientes valores: medida, porcentaje, auto e inherit.
color	Establece el color del texto de un elemento.
text-align	Establece la alineación del contenido de un elemento de bloque, se pueden usar los siguientes valores, left, right, center, justify.
font-size	Establece el tamaño de la fuente.

Para mayor información acerca de CSS consultar:

- https://developer.mozilla.org/es/docs/Learn/Getting_started_with_the_web/CSS_basics
- https://developer.mozilla.org/es/docs/Web/CSS/Referencia_CSS
- https://developer.mozilla.org/es/docs/Learn/CSS/Introduction_to_CSS/Sintaxis.

Ahora es necesario dar el diseño por medio de CSS al documento HTML creado anteriormente.

1. Crear una hoja de estilo denominada *estilos.css* en una carpeta llamada CSS, como se muestra en la **figura 7.8**:

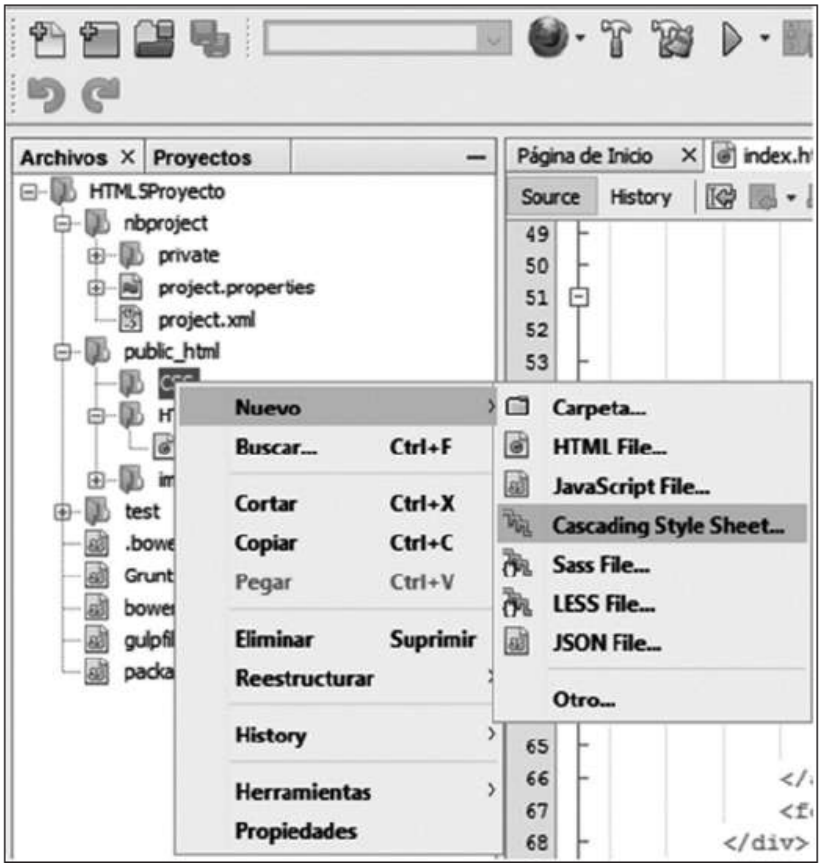


Figura 7.8 Creación de la hoja de estilo *estilos.css*

El código en el archivo *estilo.css* será como se muestra en el listado 7.2 siguiente:

Listado No. 7.2 Código de la hoja de estilo: *estilos.css*

```
1.  *{
2.      margin:0;
3.      padding:0;
4.      box-sizing: border-box;
5.  }
6.  body{
7.      background: #FFB833;
8.  }
9.      .contenido-menu{
10.     margin-top: 0px;
11.     margin:auto;
12.     max-width: 1000px;
13.     width: 100%;
14. }
15. .cabeceraImagen
16. {
17.     width: 100%;
18.     display: flex;
19.     justify-content: space-between;
20.     margin-bottom: 5px;
21. }
22. .contenedor-imagenPie
23. {
24.     display: flex;
25.     flex-flow: row wrap;
26.     margin:auto;
27.     max-width: 1000px;
28.     background: #BDBDBD;
29. }
30. .menu
```

```
{
    padding-left: 0;
    margin-top: 0;
    list-style: none;
    background-color: #453F3F;
}
.menu li{
    width: 100%;
    flex: auto;
    border-bottom: 1px solid #777474;
}
    .menu li:last-child{
    border: none;
}

.menu li a{
    color: #eee;
    text-decoration: none;
    line-height: 3;
    display: block;
    padding-left: 1em;
    font-weight: bold;
    font-size: .9em;
}
.menu li a:hover
{
    background-color: #e74c3c;
}
Footer
{
    width: 100%;
    height: 70%;
```

```
63.      background: #453F3F;
64.      color: #FFF;
65.      text-align: center;
66.      padding: 20px;
67.      display: flex;
68.      flex-wrap: wrap;
69.      justify-content: space-between;
70.      font: bold 12px verdana, sans-serif;
71.  }
72.  @media screen and (max-width:800px){
73.      .contenedor{
74.      flex-direction: column;
75.  }
76.    header {
77.      flex-direction: column;
78.      padding: 0;
79.  }
80.  }
81.  @media screen and (min-width:700px){
82.    .menu{
83.      display: flex;
84.  }
85.      .menu li{
86.      text-align: center;
87.      border-right: 1px solid white;
88.  }
89.    .menu li a{
90.      padding-left: 0;
91.     }
92.  }
```

Comentarios sobre el código del listado 7.2:

- Las líneas 1 a 5 permiten la configuración del despliegue del documento HTML en el navegador web. En el caso de tener margin:0; significa que sus márgenes iniciales serán de ese valor iniciando su despliegue desde la fila 0 y columna 0 del navegador. Posteriormente, la instrucción `padding:0`; establece el espacio de relleno requerido en todos los lados de cualquier objeto en un documento HTML.

  Finalmente, `box-sizing: border-box`; se utiliza para modificar las propiedades que tiene por defecto el CSS box model, mismo que calcula el alto y el ancho de los elementos de un documento HTML. Para la configuración de objetos en la visualización se utilizan las propiedades `margin`, `border` y `padding`. La **figura 7.9** muestra la estructura de estas propiedades.

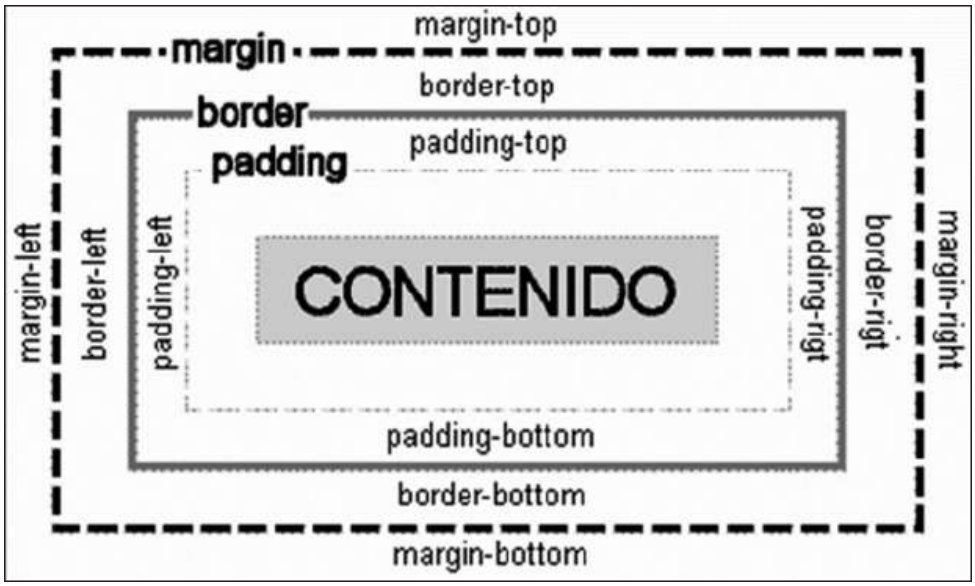


Figura 7.9 Propiedades *margin*, *border* y *padding*

- Las filas 6 a 8 definen el color (anaranjado en este caso) del cuerpo del documento HTML.

- Las filas 9 a 14 permiten determinar la configuración del contenido (las opciones) que tendrá el menú de la página `index.html`. Primero, tendrá margen superior igual a 0 en el navegador, previniendo que un elemento de bloque (block level) se estire hasta los bordes del contenedor, sea a la izquierda o a la derecha.

  Dentro de ese contexto se tendrá el máximo en pixeles para su despliegue, que será de 1000 (*max-width:1000px*). La propiedad `width:100%`; significa que según el

contenedor será siempre 100% de la anchura del mismo, según se despliegue en un monitor de PC o en la pantalla de un dispositivo móvil.

Se debe hacer una pausa en este punto para explicar someramente dos conceptos CSS:

a) En CSS existen reglas que son declarativas y permiten ejecutar propiedades sobre elementos HTML para configurar su forma, color, posición, entre otras. Las reglas a su vez están conformadas por un selector y dentro de éste las propiedades y sus respectivos valores.

Figura 7.10 Sintaxis de las reglas, selectores, propiedades y valores en CSS

b) En CSS existen clases e identificadores para la declarativa de estos selectores. Las clases contienen propiedades diferentes con valores asignados que pueden ser utilizadas por muchos elementos contenidos en un documento HTML. Se identifican con un punto al inicio del nombre del selector. Los identificadores, por su parte, sólo pueden ser utilizados por un único elemento dentro del documento HTML. Se identifican con un signo # al inicio. La **figura 7.11** muestra estos dos tipos de reglas.

```
#formato{
    text-align:center;
    color:navy;
    font-size:18px;
    font-style:italic;
    font-family:"Arial Black", Gadget, sans-serif;
    text-transform:uppercase;
}

.texto{
    text-align:justify;
    color:#009933;
    font-family:"Times New Roman", Times, serif;
    font-size:14px;
    }
```

Identificador

Clase

Figura 7.11 Clases e identificadores en CSS

- En las filas 15 a 21 se da forma a la imagen principal que tendrá el archivo *index.html*. En este caso el `width` (ancho) y el `margin-bottom` (margen inferior) que tendrá el `archivo html`. Lo nuevo aquí es el uso de `display:flex y justify-content: space-between`. Para ello, se explicará brevemente la estrategia de diseño FlexBox.

  Mozilla Developer define este modelado como: "**La propiedad Flexible Box**, *o* **flexbox**, *de CSS3 es un modo de diseño que permite colocar los elementos de una página para que se comporten de forma predecible cuando el diseño de la página debe acomodarse a diferentes tamaños de pantalla y diferentes dispositivos*." .

  Lo anterior significa que el modelado Flexible Box habilita la capacidad de alterar el ancho y alto de los elementos de un documento HTML para que logre ajustarse lo mejor posible en el espacio disponible de cualquier dispositivo.

  No es propósito de este libro detallar tecnologías para diseño CSS en sitios web. La tecnología Flexible Box o Bootstrap permitirían fácilmente ayudar con el diseño responsivo de una página web. Sin embargo, se utilizará la primera para darle un diseño básico a la página principal creada (index.html).

Para mayor más información acerca de Flex Box consultar:

https://developer.mozilla.org/es/docs/Web/CSS/CSS_Flexible_Box_Layout/Usando_las_cajas_flexibles_CSS

- Las filas 22 a 29 contienen reglas propias para la configuración del contenedor de la imagen principal de index.html. Si se observan las reglas se ve claramente el uso de Flex Box y algunas propiedades ya explicadas anteriormente.

- Las filas 37 a la 92 establecen la configuración del menú de index.html. En este caso se desglosa su contenido:

  a) De la línea 30 a 36 las propiedades `padding`, `margin` y `background-color` tienen sus respectivos valores. Ya se sabe cómo funcionan estas propiedades. En la fila 34 aparece la propiedad `list-style:none`; que permite que no aparezcan las viñetas o los elementos de la lista.

  b) De las líneas 37 a 41 se configura la visualización de la lista en el menú de index.html, permitiendo alinear los elementos contenidos (items) dentro de un área rectangular.

c) En las líneas 46 a 54 se establece para `<a>` (`anchor`, que en HTML5 permite crear un enlace a otras páginas web, archivos o ubicaciones dentro del mismo documento HTML) algunas propiedades como color, font, display, entre otras. En este sentido, las instrucciones:

```
<ul class=»menu»>
<li><a href=»index.html»>Inicio</a></li>
```

Establece para la clase menú que configura la lista un enlace a index.html (en este caso un llamado a sí misma desde la página), algunas propiedades como color, font, display, font-weight, entre otras.

d) En las líneas 55 a 58 se crea un hover (hoves es una seudoclase CSS que se presenta cuando el usuario coloca el puntero sobre alguno de los elementos del documento HTML y que no necesariamente está activo) para dotarlo de un color diferente. Así, el usuario se da cuenta que está posicionado sobre esa opción del menú cuando coloca el puntero sobre él.

e) En las líneas 59 a 71 se establecen los valores de algunas propiedades, tales como anchura (`width`), altura (`height`), color (`color`), alineamiento de texto (`text-align`), entre otras para el pie de la página principal del sitio.

f) En las filas 71 a 92 se incluye el concepto `media queries`, representado por @media. Este concepto permite adaptar la representación del contenido de un documento HTML a las características del dispositivo. En otras palabras, la representación de la página puede adaptarse a un rango específico de dispositivos de salida, según su tamaño, tales como una pantalla ancha (dispositivo móvil). En esta sección es posible visualizar las siguientes reglas:

- Cuando la pantalla tenga una anchura máxima de 800 pixeles el contenedor presenta una visualización de pantalla columnar (*flex-direction: column*) que permite que dicho contenedor se muestre de inicio a fin en el explorador (la columna definida ocupa todo el espacio horizontal disponible en el contenedor). Esto se expresa en las líneas 72 a la 80.

- Cuando la pantalla tenga un ancho mínimo de 700 pixeles, el despliegue del menú será tipo flex (*display:flex*). En este caso, es similar al *display:block* de CSS estándar, es decir, cada objeto ocupará su determinado espacio, según su ancho. De este modo, podrían coexistir varios objetos en forma de caja que se despliegan en la misma fila del contenedor.

Para mayor más información acerca de Media Queries, consultar:

https://developer.mozilla.org/es/docs/CSS/Media_queries

Al finalizar de escribir este archivo CSS se tiene que modificar el listado 7.1 para que incluya el llamado a dicho CSS y así establecer las clases que invocará cada sección del documento HTML. El listado 7.3 muestra esta modificación (se muestran los cambios en letra negrita).

Listado No. 7.3 Agregar CSS al documento `index.html`

```
<!DOCTYPE html>
  <html lang="es">
    <head>
        <meta charset="UTF-8">
        <meta name="description" content="Ejemplo HTML 5">
        <meta name="keywords" content="HTML 5, CSS3, JavaScript">
        <meta name="author" content="Enrique Gómez | Jonathan Moreno">
        <meta name="robots" content="noindex, nofollow">
        <title>Nuestro City Mall</title>
        <meta name="viewport" content="width=device-width, user-scalable=no,
              initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
        <link href="../CSS/estilo.css" rel="stylesheet" type="text/css"/>
    </head>
<body>
    <nav class="contenido-menu">
              <ul class="menu">
                    <li><a href="index.html">Inicio</a></li>
        <li><a href="#">Catálogo de tiendas</a></li>
        <li><a href="">Servicios</a></li>
       <li><a href="">Convenios comerciales</a></li>
       <li><a href="contactenos.html">Contáctenos</a></li>
       </ul>
    </nav>
```

```
55.     <div class="contenedor-imagenPie">
56.             <div class="cabeceraImagen">
57.                 <img src="../imagenes/Mall.jpg" width="100%" height="90%"
                    alt=""/>
58.             </div>
59.       <footer>Derechos Reservados &copy; 2016-2020</footer>
60.          </div>
61. </body>
62. </html>
```

Si se ejecuta ahora el archivo index.html, se mostraría algo similar a la **figura 7.12**.

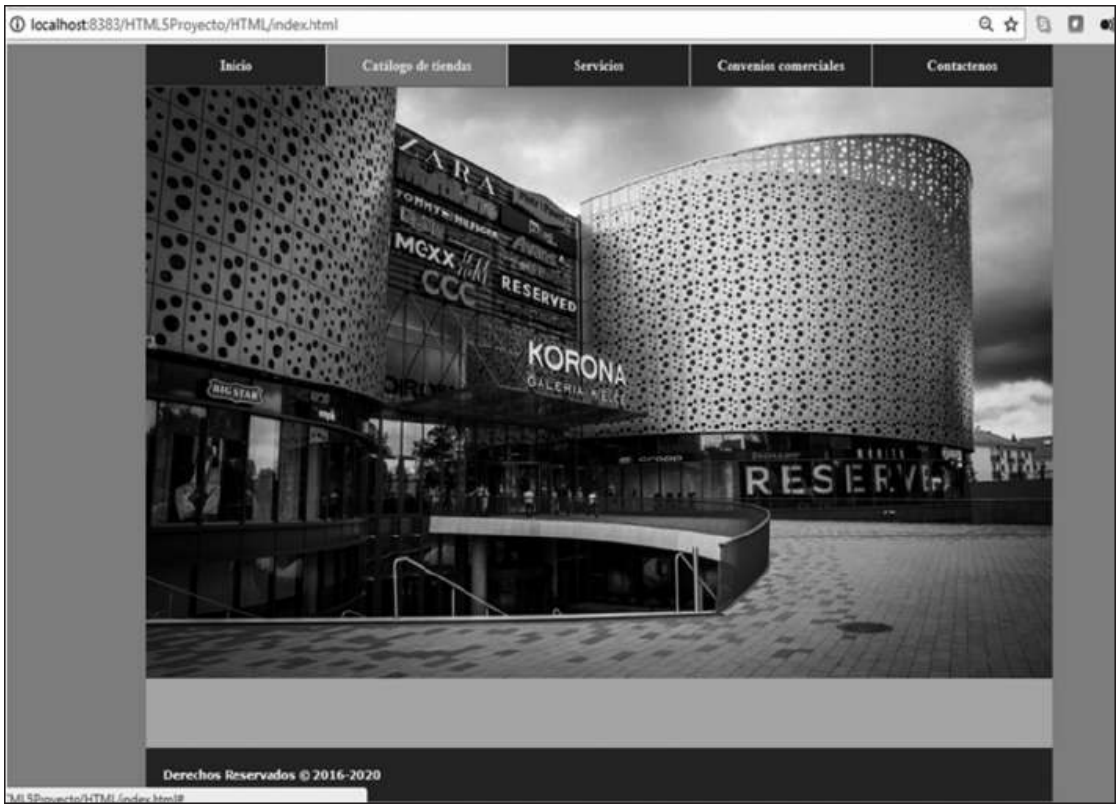


Figura 7.12 Ejecución de *index.html* en vista navegador de PC

Si se requiere observar la versión para móviles se mostraría similar a la **figura 7.13**.

Figura 7.13 Ejecución *index.html* en vista para móviles

Resumen

En este capítulo se desarrollaron temas relacionados con los fundamentos de la programación web; no se trataron en profundidad, ya que la intención es introducir al lector en ellos. Tal es el caso de las tecnologías HTML y CSS, que sin llegar a ser muy detallados se logró crear un documento HTML 5 sencillo con su respectivo CSS. En resumen, en este capítulo se trataron los siguientes temas:

- Fundamentación de la programación web a nivel front-end, considerando HTML5 y CSS3.

- Maquetación semántica de un documento HTML, con el uso de HTML5.
- Diseño gráfico de la maquetación semántica HTML mediante el uso de CSS3.

☑ Autoevaluación

1. Según su comprensión ¿A qué se denomina front-end?

2. En sus propias palabras ¿Cuál es la definición de HTML?

3. Mencionar algunas de las funcionalidades de CSS.

4. Describir la funcionalidad del componente <article> en un documento HTML. Puede brindar un ejemplo.

5. Citar un ejemplo donde se utilice el componente <aside>

🗀 Evidencia

☐ Recrear el diseño de la página *index.html*.

☐ Escribir el código para el archivo *estilo.css*.

☐ Analizar y comprender la sintaxis de los ejemplos HTML5 aportados en el sitio http://public.dhe.ibm.com/software/dw/web/wa-html5/html5.examples.zip

☐ Crear un mapa conceptual de las tecnologías HTML5 y CSS3 explicadas en este capítulo.

☐ Investigar las herramientas de front-end que se citaron en este capítulo.

☑ Respuestas a las preguntas de autoevaluación

1. Son todas aquellas herramientas que permiten la ejecución de interfaces y programación web en el lado del cliente (navegador web o máquina del usuario).

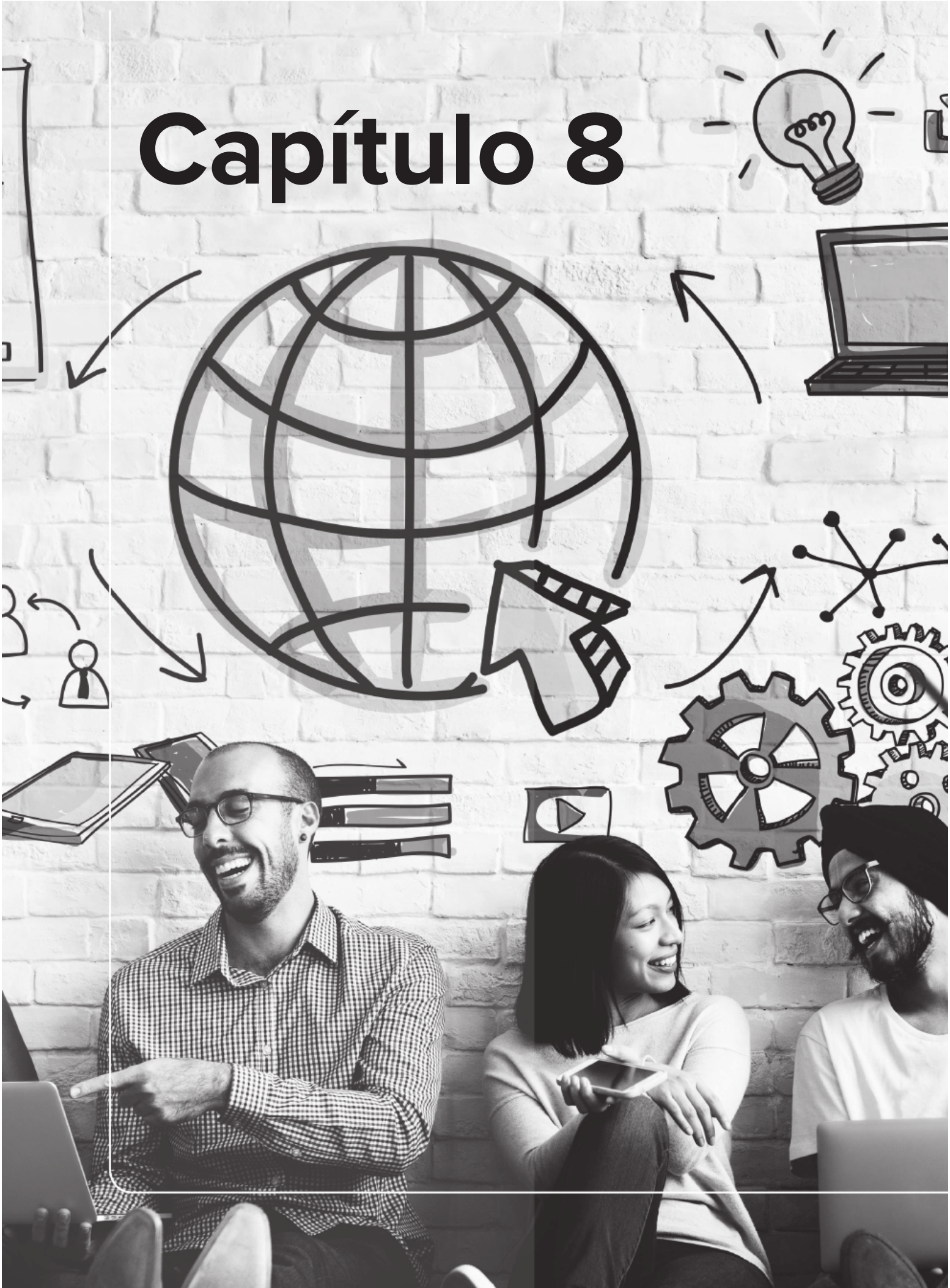
2. Es un lenguaje de marcas que permite crear la estructura semántica de un documento HTML que se despliega en un navegador web.

3. Permite establecer el estilo y diseño de presentación de la estructura semántica de un documento HTML (colores, fuentes tipográficas, formato de objetos, imágenes, animaciones, entre otros). Por otro lado, hace posible la responsividad de los sitios web.

4. Es un componente semántico HTML5 que permite describir un elemento específico dentro de un conjunto de elementos contenidos en un componente section. Por ejemplo, el *article* futbol podría ser un *article* dentro de otros denominados baloncesto, beisbol, tenis, entre otros, que pertenecen a un componente *section* llamado deportes.

5. Este componente podría ser utilizado en un sitio web de un hotel para colocar enlaces a sitios externos pero que de una u otra forma se relacionan con el negocio. En este caso, podrían ser enlaces a rent-a-car o líneas aéreas.

Capítulo 8



Servlets Java

Reflexione y responda
las siguientes preguntas

¿Conoce la diferencia entre código fuente que se ejecuta en el cliente como el que lo hace en el servidor web?

¿Es posible ser expertos en front-end y back-end, o solamente se puede especializar en uno de ellos?

¿Cómo incorpora Java código servidor en un archivo HTML?

Contenido

8.1 Introducción
- **8.1.1** Solicitudes y respuestas en sitios y aplicaciones web

8.2 Conceptos básicos de Servlets
- **8.2.1** Uso de cookies con Servlet
- **8.2.2** Manejo de Http Session con Servlet

Expectativa

Desarrollar un aplicativo web representa un alto grado de compromiso con los usuarios que lo utilizarán, así como una disposición completa con el cumplimiento de las mejores prácticas en la creación de la solución. En este sentido, la funcionalidad del aplicativo web va de la mano con la arquitectura técnica y la calidad del software que se produzca. La expectativa en este caso será entonces crear software de calidad con un alto cumplimiento de las funcionalidades que exige el usuario.

En un mundo donde la información es cada vez más importante para los negocios y las personas, el software constituye la fuente que permite obtenerla. Las aplicaciones web han superado poco a poco a las soluciones de escritorio, lo que hace necesario que conozcamos diferentes tecnologías informáticas para proveer al usuario de mayor funcionalidad en el menor tiempo posible y en diferentes formatos.

En este contexto, la expectativa circunda el conocimiento hacia tecnologías front-end, back-end, bases de datos, entre otras.

Después de estudiar este capítulo, el lector será capaz de:

- Entender cómo se dan las solicitudes y respuestas en una aplicación web.
- Comprender conceptos básicos de los servlets en Java.
- Usar cookies cuando se trabaje en proyectos con servlet.
- Manejar sesiones con servlets Java.
- Realizar proyectos que integren el uso de páginas JSP formateadas mediante CSS y con el uso de Servlet para procesar peticiones de los clientes y brindar respuestas a las mismas, con la utilización de sesiones y validación de éstas.

.

8.1 Introducción

Cuando se ingresa al mundo de la programación Java se puede comprender que existe una importante variedad de aplicaciones: escritorio, web, dispositivos móviles, juegos, software empotrado, entre otros. Entonces, es posible dimensionar la poderosa herramienta que constituye Java y el enorme abanico de posibilidades que brinda para desarrollar soluciones informáticas. Este capítulo y los siguientes se enfocarán en la tecnología orientada a la creación de aplicaciones web considerando varias herramientas y estrategias de desarrollo, tales como Servlets, JavaBeans, Java Server Pages, entre otras.

Cuando se habla de un sitio web éste hace referencia a una colección de páginas estáticas HTML formateadas con CSS y con algunas funcionalidades dinámicas a nivel de cliente mediante JavaScript. Sin embargo, no se puede establecer dinamismo como el manejo de bases de datos o transacciones. Esto significa que a este nivel se tiene sólo el front-end del aplicativo web. Por su parte, una aplicación web sí permite establecer esa transaccionalidad de bases de datos, aunado al front-end.

Este capítulo se dedicará a desarrollar los fundamentos de la programación web orientados hacia el back-end. Es decir, se explicarán los principios que rigen el desarrollo de las capas web, las de negocio y las de datos, así como las herramientas que permiten esa implementación.

8.1.1 Solicitudes y respuestas en sitios y aplicaciones web

Las comunicaciones de una aplicación web en Internet se rigen por peticiones y respuestas. El concepto básico de la comunicación entre el software que se encuentra en un servidor y el acceso a través de la computadora de un usuario se realiza generalmente a través del protocolo HyperText Transfer Protocol (HTTP). El mecanismo es el siguiente: el navegador del usuario envía una petición de una página web al servidor a través de una petición HTTP (*request*) la cual incluye un archivo *.html y el servidor responde mediante una respuesta HTTP (*response*).

Cuando las páginas corresponden a un sitio web (web estática) el servidor responde con un documento *.html solicitado por el usuario, es decir, las respuestas del servidor corresponden a páginas *.html requeridas por el usuario.

En el caso de que sea un aplicativo web (web dinámico) la respuesta al usuario dependerá de la solicitud que realice y los datos que proporcione: por ejemplo, puede solicitar el acceso a un sistema mediante una interfaz de menú y la respuesta dependerá de sus datos personales (si accede o se le brinda una pantalla de error).

8.2 Conceptos básicos de Servlets

Antes de iniciar con Servlets es importante conceptualizar la arquitectura de un aplicativo web con Java. En .NET se construye una aplicación tras considerar las capas de datos, negocios y presentación. En Java también se contempla esa arquitectura para construir aplicaciones. Generalmente, las capas de una aplicación Java son Cliente, Web, Negocios y Datos, las cuales se definen como sigue:

a) Capa de cliente: Están presentes herramientas como HTML 5, CSS 3, JavaScript, entre otras. Su funcionamiento principal es mostrar contenido al usuario que interviene a través de un navegador web. Es el front-end del aplicativo web; tiene como propósito realizar las peticiones del cliente al servidor Java como también mostrar información al usuario.

En este contexto se pueden utilizar herramientas que ayuden al desarrollador en el proceso de diseño. Por ejemplo, podría utilizar Bootstrap, Pure, Foundation, Metro UI u otro framework para el maquetado del aplicativo web, despreocupándose del aspecto responsive (se adapta al tamaño de la pantalla del dispositivo) del mismo. En la parte dinámica se puede utilizar JavaScript, JQuery, Node.js, Angular.js, entre otros.

b) Capa web: En esta capa se incluyen los Servlets que se utilizan para una importante cantidad de funciones, como: realizar consultas de datos en un servidor web, insertar, modificar y eliminar. También se incluyen los JSP (Java Server Pages) que implementan además de lógica Java la estructuración semántica de HTML.

c) Capa de negocio: En ésta se pueden incluir los JavaBeans que permiten encapsular objetos en uno solo (Bean o vaina). Los JavaBeans son clases Java con funcionalidades determinadas; por ejemplo, las clases de Swing y AWT son JavaBeans. La importancia de los JavaBeans, aparte de su encapsulamiento en un objeto genérico, radica en su reutilización pues proveen la facilidad de reconstrucción de éste en la etapa de diseño mediante sus propiedades y la redefinición de los sucesos que soporta.

d) Capa de datos: En esta capa se incluyen herramientas como Hibernate o JPA que son framework y que permiten la interacción con una base de datos a través de JDBC; éste contiene todos los componentes requeridos para la gestión de bases de datos no importando cuál sea el proveedor, siempre que aporte el driver respectivo. Entonces, se puede brindar soporte a bases de datos Oracle, SQL Server, MySQL, entre otros.

También en esta capa se incluye la facilidad de construcción de un contexto a través de un ORM (Object Relational Mapping); en Java o en .NET es una técnica que permite crear un mapa conceptual del modelo físico de una base de datos y es contra este mapa conceptual que se trabaja la gestión de datos. En .NET Framework de Microsoft® por

ejemplo, se cuenta con Entity Framework o nHibernate que permite realizar estos mapeos. En Java se puede diseñar mediante Hibernate, iBatis, Ebean, entre otros.

La **figura 8.1** muestra la arquitectura de una aplicación web en Java.

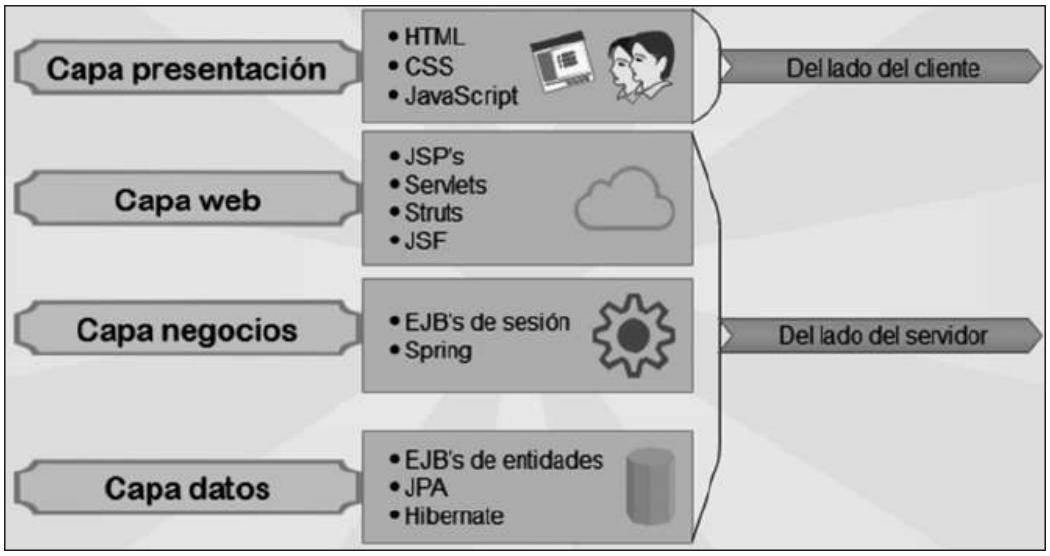


Figura 8.1 Arquitectura de una aplicación web Java

Los Servlets constituyen módulos escritos en Java que se utilizan en un servidor, el cual puede o no ser un servidor web, para extender sus capacidades de respuesta a peticiones de los usuarios. Se podría crear una analogía: los Servlets son a los servidores lo que los applets a los navegadores, aunque los primeros no poseen interfaz gráfica.

Los Servlets permiten la generación de documentos dinámicos, conexiones a bases de datos, manejo de peticiones concurrentes, programación distribuida, entre otras funcionalidades, utilizan para ello la programación Java. Por ejemplo, se podría tener un Servlet que se encargue de procesar los datos enviados desde un documento HTML: registrar la transacción, actualizar la base de datos, devolver un resultado, contactar a otro Servlet, entre otras funciones.

Los paquetes *Java javax.servlet* y *java.servlet.http* proporcionan interfaces y clases que permiten escribir nuestros propios Servlet. Mediante javax.servlet.Servlet es posible definir los métodos del ciclo de vida de un Servlet; tal ciclo establece las siguientes fases:

- **Inicialización:** Se produce cuando el servidor carga el Servlet y ejecuta el método init del mismo. Cabe mencionar que el proceso de inicialización del Servlet debe terminarse antes de poder manejar las peticiones que realizan los clientes y, lógicamente, antes de que el Servlet sea destruido.

- **Interacción con las peticiones de los clientes:** Una vez inicializado el Servlet se puede interactuar con las peticiones de los clientes, se debe tener sumo cuidado con el uso de variables compartidas entre diversas peticiones, pues se podrían producir problemas de sincronización entre diversos requerimientos simultáneos.

- **Destrucción del Servlet:** Los Servlets son destruidos por el servidor, ya sea porque éste sea cerrado o por que el administrador del sistema así lo disponga. Esta destrucción se lleva a cabo mediante el método *destroy* del propio Servlet. Para usarlo de nuevo hay que iniciar de nuevo el Servlet.

El funcionamiento de un Servlet se puede observar en la **figura 8.2**: el primer paso es que el usuario realice una solicitud a un servidor web, éste a su vez invoca a un Servlet el cual determina la funcionalidad, recoge datos o cualquier otro requerimiento según la solicitud, y se comunica con un documento HTML que es devuelto al usuario a través del servidor de aplicaciones.

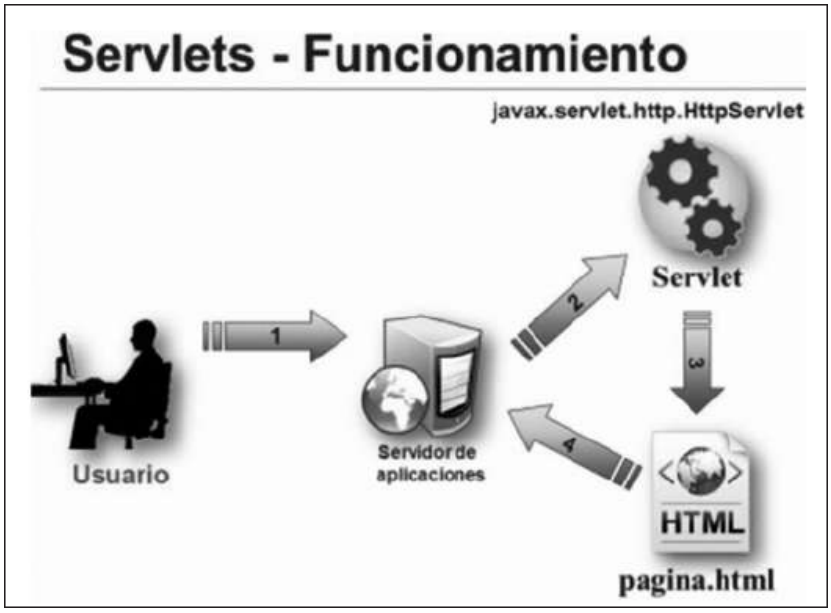


Figura 8.2 Funcionamiento de un Servlets Java

Ahora se presenta cómo funciona realmente un Servlet en Java mediante un ejemplo de aplicación.

Ejemplo No. 1 Primer Servlet: Hola mundo

1. Ingresar a NetBeans.
2. Crear un proyecto de tipo web application, como se observa en la **figura 8.3**.

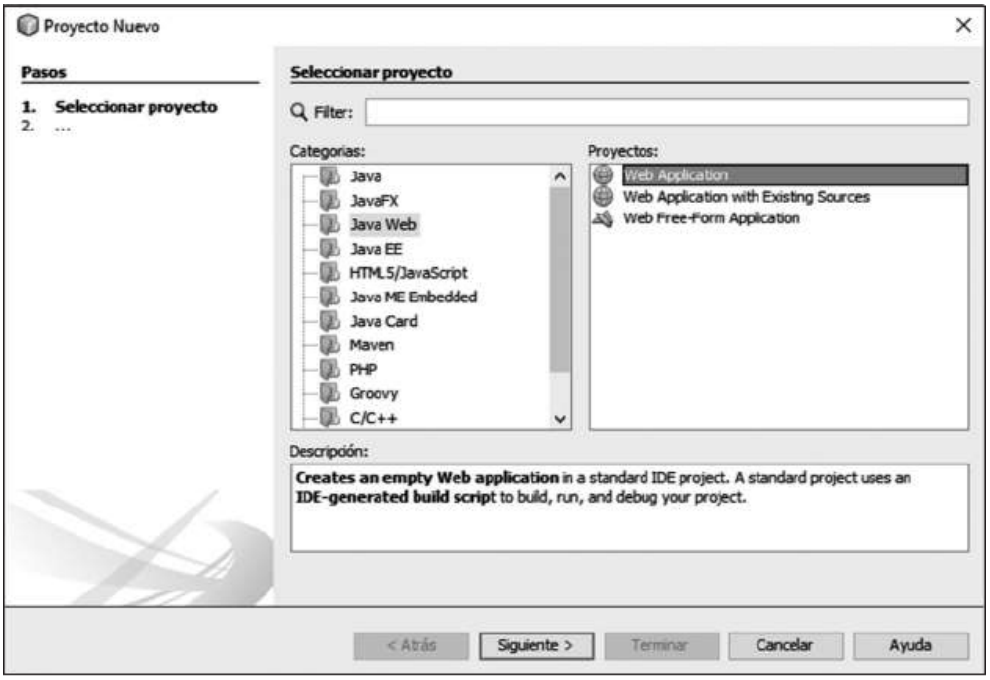


Figura 8.3 Creación de un proyecto tipo web application

3. Pulsar el botón Siguiente y seleccionar el nombre ServLetHolaMundo para el nombre del proyecto. Dar clic en Siguiente. En la pantalla que aparece se configura el servidor como *GlassFish 4.1.1*. como se muestra en la **figura 8.4**.

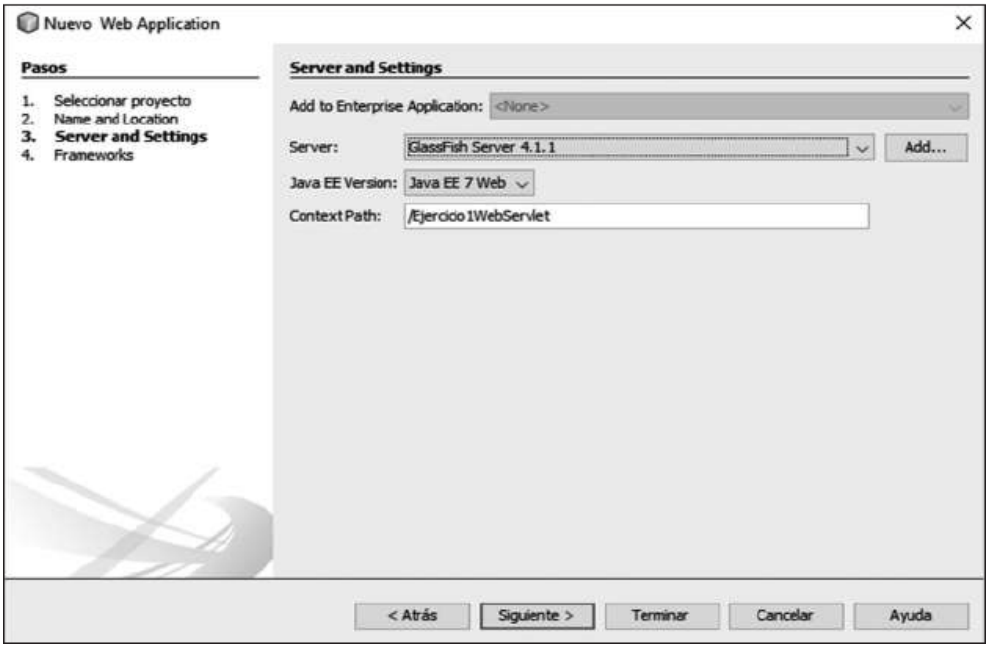


Figura 8.4 Configuración del servidor y la versión de JavaEE

4. Pulsar el botón Siguiente y el botón Terminar para crear el proyecto.

5. Una vez creado el proyecto, dar clic sobre el nombre de éste y seleccionar Nuevo y Servlet, como se muestra en la **figura 8.5**.

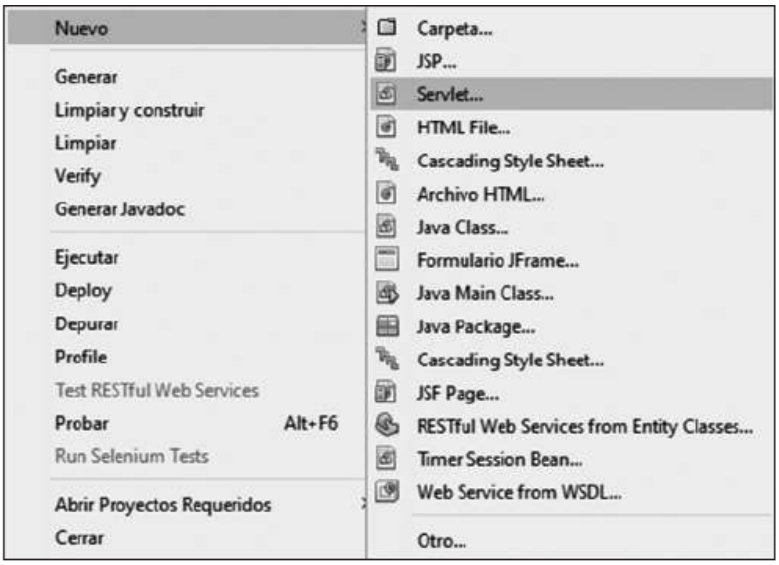


Figura 8.5 Creación del primer Servlet

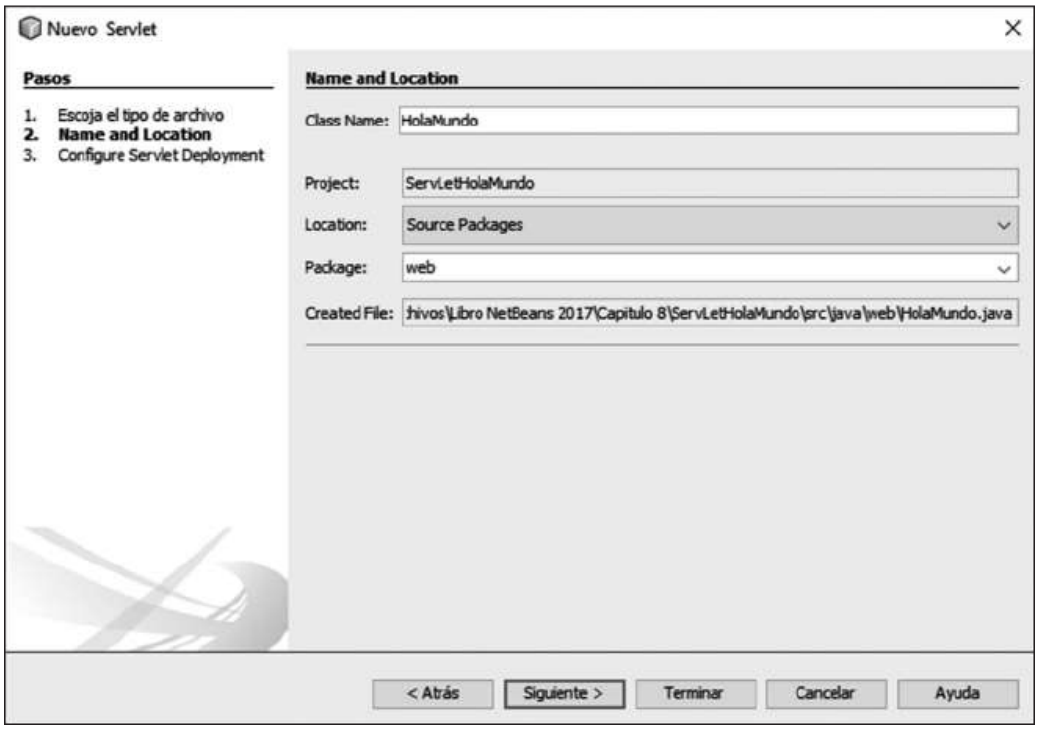


Figura 8.6 Nombre y ubicación del primer Servlet

6. El primer Servlet será una clase llamada HolaMundo y se debe escribir el nombre del paquete (Package) donde se almacenará. La **figura 8.6** muestra esta configuración.

7. Pulsar a continuación el botón Terminar para finalizar la creación del primer Servlet. La pantalla mostrada al dar clic en Siguiente (**figura 8.6**) no es importante por ahora.

8. Este proyecto se verá como se muestra en la **figura 8.7**.

Figura 8.7 Conformación del primer proyecto de Servlets

Como se puede observar en la **figura 8.7** el proyecto consiste en un directorio Web Pages donde radican los archivos html. Al contenido de este directorio sólo se podrá acceder internamente por la aplicación web. En el directorio Source Packages se tendrán los paquetes del primer proyecto, tal como el nombrado web donde radica el Servlet *HolaMundo.java*, así como también todas las clases java que se necesiten para los nuevos proyectos.

9. Ahora se modificará el archivo index.html para que sea similar al mostrado en el listado 8.1.

Listado No. 8.1 Código del archivo index.html

```
1.  <!DOCTYPE html>
2.  <html>
3.     <head>
4.          <title>TODO supply a title</title>
5.          <meta charset="UTF-8">
6.        <meta name="viewport" content="width=device-width, initial-scale=1.0">
7.      </head>
8.      <body>
9.          <h1>Invocando al Servlet HolaMundo :
```

```
10.             <a href="/ServLetHolaMundo/HolaMundo">Pulsa aquí</a>
11.         </h1>
12.     </body>
13. </html>
```

Sobre el código del listado 8.1:

- La línea 1 indica con el <!DOCTYPE> que es un documento HTML.
- De las líneas 3 a la 7 se construye el header del documento HTML que ya se estudió en el capítulo 7.
- De las líneas 8 a 12 se encuentra el body (cuerpo) del documento HTML. En esta sección se puede señalar lo siguiente:

  a) En la línea 8 se construye un título de tipo _h1 (header 1) de HTML.

  b) En la línea 9 se crea una referencia a la dirección dada por /ServLetHolaMundo/HolaMundo que en la primera parte (/ServLetHolaMundo) se refiere al nombre del proyecto web y la segunda (/HolaMundo) se refiere al Servlet que se creó anteriormente.

- En la línea 11 se cierra el encabezado h1.
- En la línea 12 se cierra el body del documento HTML.
- En la línea 13 se cierra el documento HTML.

10. Ahora se modifica el Servlet *HolaMundo.java*. El listado 8.2 muestra cómo se vería una vez modificado.

Listado No. 8.2 Código del Servlet HolaMundo.java

```
1.  package web;
2.  import java.io.IOException;
3.  import java.io.PrintWriter;
4.  import javax.servlet.ServletException;
5.  import javax.servlet.annotation.WebServlet;
6.  import javax.servlet.http.HttpServlet;
7.  import javax.servlet.http.HttpServletRequest;
8.  import javax.servlet.http.HttpServletResponse;
9.  /**
10.  *
11.  * @author
12.  */
```

```
13.
14. @WebServlet(name = “HolaMundo”, urlPatterns = {“/HolaMundo”})
15. public class HolaMundo extends HttpServlet {
16.     /**
17. * Processes requests for both HTTP <code>GET</code> and        <code>POST</
    code>
18.      * methods.
19.      *
20.      * @param request servlet request
21.      * @param response servlet response
22.      * @throws ServletException if a servlet-specific error occurs
23.      * @throws IOException if an I/O error occurs
24.      */
25.
26.
27.  @Override
28.   protected void doGet(HttpServletRequest request,     HttpServletResponse
    response) throws ServletException, IOException {
29.         response.setContentType(“text/html;charset=UTF-8”);
30.         try (PrintWriter out = response.getWriter( )) {
31.             /* TODO output your page here. You may use following sample
    code. */
32.             out.println(“<!DOCTYPE html>”);
33.             out.println(“<html>”);
34.             out.println(“<head>”);
35.             out.println(“<title>Servlet HolaMundo</title>”);
36.             out.println(“</head>”);
37.             out.println(“<body>”);
38.   out.println(“<h1>El Servlet HolaMundo está en el proyecto “ +
                                        request.getContextPath( ) + “</h1>”);
39.             out.println(“</body>”);
40.             out.println(“</html>”);
41.         }
42.     }
43. // <editor-fold defaultstate=“collapsed” desc=“HttpServlet methods. Click
    on the + sign on the left to edit the code.”>  //Borre el código generado
44. }    //Fin de la clase HolaMundo
```

Sobre el código del listado 8.2:

- La línea 1 hace referencia al paquete web donde se colocó el Servlet `HolaMundo.java`.
- En las líneas 2 a 8 se importan todas las librerías que se requerirán para el funcionamiento del Servlet HolaMundo.java.
- En la línea 14 se escribe la configuración del Servlet (la cual puede también hacerse en el archivo web.xml). La línea sería `@WebServlet(name = "HolaMundo", urlPatterns = {"/HolaMundo"})`, donde `@WebServlet(name = "HolaMundo"` significa el nombre del Servlet y `urlPatterns = {"/HolaMundo"}` indica la ruta relativa desde donde se localizará el Servlet. Aquí se pueden establecer varias rutas de ejecución, sin embargo, por la simplicidad de este Servlet basta con que se indique esta ruta.
- En la línea 15 se declara la clase `HolaMundo`, la cual extiende o hereda de la clase `HttpServlet` para que el servidor la reconozca como un Servlet.
- En la línea 27 se escribe la palabra reservada `@Override` dado que se necesita reescribir el método doGet. Esto amerita un detenimiento para hacer un pequeño resumen del tema.

Cuando se manejan las peticiones HTTP en un Servlet es necesario extender la clase `HttpServlet` y sobrescribir los métodos que posee y que se encargan de manejar dichas peticiones. En este caso, las peticiones a manejar son GET y POST. Los métodos que manejan estas peticiones son `doGet` y `doPost`. En el ejemplo anterior se trabaja el doGet. Más adelante, se retomará este tema con mayor detalle.

En este método también se puede observar que como parámetros se incluyen el request (solicitud) que es un parámetro entrante, como también se tiene el response (respuesta) que es el parámetro de salida del Servlet.

- En la línea 29 se establece que la respuesta será de tipo "txt/html" bajo una caracterización UTF-8.
- En las líneas 32 a 40 se crean las salidas del objeto `out`, invocando a su método println (que imprime contenido más un salto de línea). Este objeto fue creado con `PrintWriter` y servirá para imprimir contenido html. Cabe señalar que en la línea 38 se especifica el llamado a `request.getContextPath` lo cual permite obtener el nombre del aplicativo web.
- El código que exista entre la línea 43 y 44, y que se ha generado automáticamente (puede estar oculto, presionar +) al crear el Servlet se debe eliminar, por tanto queda igual que el listado 8.2.

11. Una vez explicado someramente el código del Servlet se procede a ejecutarlo. Para ello se pulsa sobre el botón verde que está debajo del menú principal de NetBeans (o la tecla F6). Se observa en el navegador algo similar a lo mostrado en la **figura 8.8**.

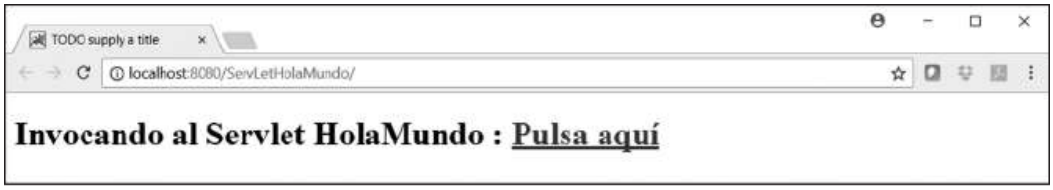


Figura 8.8 Ejecución del documento index.html en el proyecto de Servlet

12. Si se pulsa sobre el vínculo que indica Pulsa aquí, se puede revisar el código que se encuentra en el archivo *index.html*, fila 10 del listado 8.1, en éste se invoca al Servlet `HolaMundo` del proyecto. Una imagen similar a la mostrada en la figura 8.9 se desplegará.

Figura 8.9 Ejecución del Servlet *HolaMundo.*java en el proyecto de Servlet

Ejemplo No. 2 Creación del segundo Servlet: autenticación de usuario.

1. Ingresar a Netbeans.
2. Crear un proyecto de tipo web application, como se observa en la **figura 8.3**. El nombre que tendrá este proyecto es *ServletLogin*.
3. El archivo index.html tendrá el código que se muestra en el listado 8.3.

Listado No. 8.3 Código del archivo *index.html* del proyecto *ServletLogin*

```
1.  <!DOCTYPE html>
2.  <html>
3.      <head>
4.          <title>Ejemplo de Login con Servlet Java</title>
5.          <meta charset="UTF-8">
6.          <link rel="stylesheet" type="text/css" href="estilos.css">
7.          <meta name="viewport" content="width=device-width, initial-
            scale=1.0">
8.       </head>
9.      <body>
```

```
10.        <div id="contenedor">
11.           <form action= "Login" method="GET">
12.              <label>Usuario: </label>
13.              <input type="text" placeholder="Digite usuario" name="user"/><br>
14.              <label>Contraseña: </label>
15.                      <input type="password" placeholder="Digite contraseña"
                         name="password"/><br><br>
16.              <input type="submit" value="Enviar">
17.           </form>
18.     </div>
19.   </body>
20. </html>
```

Sobre el código del listado 8.3:

- La línea 1 indica con el <!DOCTYPE> que es un documento HTML.
- Las líneas 3 a 8 configuran los valores del head del documento HTML. Esta sección es similar a lo estudiado en el primer ejercicio.
- En las líneas 9 a 19 se encuentra el cuerpo (body) del documento HTML. A continuación se describen las instrucciones que contiene:

  a) En la línea 10 se crea un div HTML que estará configurado mediante el identificador contenedor el cual se encuentra en el archivo estilos.css que se creará posteriormente (en la fila 6 de este listado se hace la referencia a este archivo CSS denominado estilos.css). Este identificador permite crear el "layout" general del documento HTML, es decir, será el contenedor global de todos los elementos del HTML.

  b) En la línea 11 se crea un formulario HTML cuya acción es Login, el cual representa al documento HTML o algún archivo de ejecución en el servidor web. Además, contiene el método Get que se utilizará para la comunicación HTTP.

  c) En la línea 12 se crea una etiqueta (label) con el texto Usuario.

  d) En la línea 13 se crea un objeto input (de entrada) que desplegará el texto "Digite usuario". Contiene además un campo placeholder que es el texto que aparece temporalmente en la caja de texto cuando se encuentre vacía.

e) En las filas 14 y 15 se crea la etiqueta para desplegar un texto con la leyenda Contraseña: y su campo de entrada (input tipo password) respectivo.

f) En la línea 16 se crea un input de tipo submit que no es más que un objeto botón de comando con la leyenda Enviar.

- Finalmente, a partir de la línea 17 se cierran las diferentes secciones del documento HTML.

4. Ahora se crea el Servlet para el proyecto ServletLogin. En este caso el Servlet se llamará Login. En la **figura 8.5** y **8.6** se muestra cómo crearlo; en este caso, respecto de la figura 8.6 lo único que cambia será el nombre del Servlet.

5. En la pantalla siguiente se encontrará la opción de crear el archivo web.xml, donde se establecerán algunas configuraciones del nuevo Servlet. Seleccionar el "check" de `Add information to deployment descriptor (web.xml)`. como se observa en la **figura 8.10**.

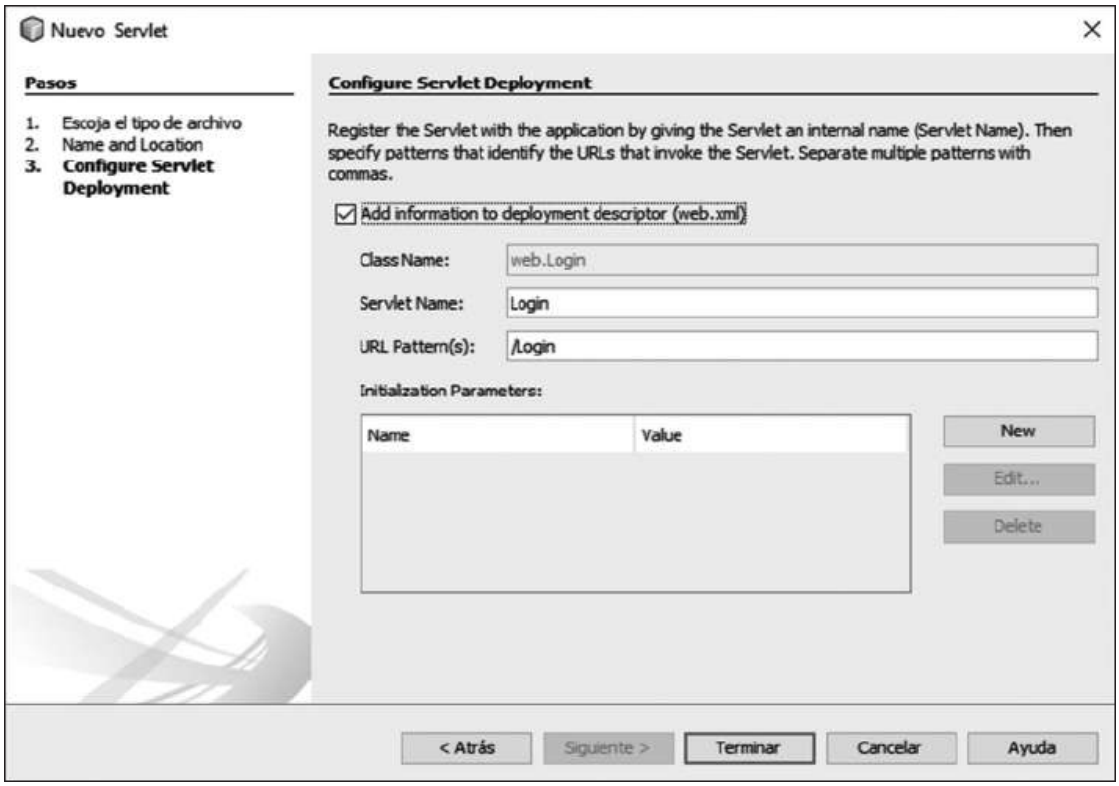


Figura 8.10 Agregar el archivo de configuración web.xml al proyecto

6. Ahora se crea el archivo estilos.css para dar formato al documento HTML. Dar clic derecho sobre el paquete Web Pages, elegir la opción Nuevo y luego Cascading Style Sheet, poner el nombre estilos y pulsar el botón Terminar. El listado 8.4 muestra esa codificación.

Listado No. 8.4 Código del archivo *estilos.css* del proyecto *ServletLogin*

```
1.  /*Configuración del contenedor general*/
2.  #contenedor {
3.      background-color: #F4AF7C;
4.      border-radius: 10px;
5.      box-shadow: 3px 3px 10px 0px rgba(50, 50, 50, 0.75);
6.      -moz-box-shadow: 3px 3px 10px 0px rgba(50, 50, 50, 0.75);
7.     -webkit-box-shadow: 3px 3px 10px 0px rgba(50, 50, 50, 0.75);
8.      margin-left: auto;
9.      margin-right: auto;
10.     margin-top: 10%;
11.     max-width: 25em;
12.     padding-bottom: 10px;
13.     padding-top: 5px;
14. }
15. /* Etiquetas del formulario */
16. label {
17.     color: # 0B0B0B;
18.     display: block;
19.     margin-bottom: 6px;
20.     margin-left: 1.2em;
21. }
22. /* Campos del formulario */
23. input[type=“text”],
24. input[type=“password”] {
25.     background-color: #F2FBEF;
26.     border: none;
27.     border-radius: 10px;
```

```
28.     display: block;
29.     font-size: 1em;
30.     height: 2em;
31.     text-align: center;
32.     width: 90%;
33.     margin-left: auto;
34.     margin-right: auto;
35. }
36. /* Configuración del Botón Enviar */
37. input[type=“submit”] {
38.     background-color: # 929191;
39.     border: none;
40.     border-radius: 10px;
41.     color: white;
42.     display: block;
43.     font-size: 1em;
44.     height: 3em;
45.     margin-left: auto;
46.     margin-right: auto;
47.     margin-top: -10px;
48.     text-align: center;
49.     width: 40%;
50. }
51. input[type=“submit”]:hover {
52.     cursor: pointer;
53.     background-color: #FA6232;
54.     opacity: 0.8;
55. }
```

Sobre el código del listado 8.4:

- De la línea 2 a la 14 se crea un selector por identificador (cabe recordar que una regla se compone del nombre de un selector, sus propiedades y los valores de éstas. Asimismo, un selector por id se identifica porque se le antepone el símbolo #, y a una clase, un

punto). En este caso, el selector #contenedor crea una serie de propiedades con sus valores, tales como el color de fondo o background-color, las sombras en los objetos (box-shadow), los márgenes (margin), entre otras propiedades que por el cometido de este libro no se detallarán.

- Las líneas 16 a 21 establecen los valores para las propiedades del selector label que es un objeto propiamente de HTML.

- Las líneas 23 a 55 configuran las propiedades de los objetos de entrada (`input`) del documento index.html.

7. El **listado 8.5** muestra la codificación del Servlet *Login.java*.

Listado No. 8.5 Código del Servlet *Login.java*

```
1.  package web;
2.  import java.io.IOException;
3.  import java.io.PrintWriter;
4.  import javax.servlet.ServletException;
5.  import javax.servlet.http.HttpServlet;
6.  import javax.servlet.http.HttpServletRequest;
7.  import javax.servlet.http.HttpServletResponse;
8.  /**
9.   *
10.  * @author
11.  */
12. public class Login extends HttpServlet {
13. @Override
14.    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
15.                      throws ServletException, IOException {
16.            String user = req.getParameter(“user”);
17.            String pass = req.getParameter(“password”);
18.            if (“EGJM”.equals(user) && “12345”.equals(pass)) {
19.                    response(resp, “Usuario y contraseña válidos!”);
20.            } else {
21.                    response(resp, “Usuario o contraseña no válidos”);
22.            }
```

```
23.     }
24.     private void response(HttpServletResponse resp, String msg)
25.                     throws IOException {
26.             PrintWriter out = resp.getWriter( );
27.             out.println("<html>");
28.             out.println("<body>");
29.             out.println("<h1>" + msg + "</h1>");
30.             out.println("</body>");
31.             out.println("</html>");
32.     }
33. }
```

Sobre el código del listado 8.5:

- Las líneas 16 y 17 crean código para utilizar el método `getParameter` del parámetro req que es de tipo `HttpServletRequest` para obtener los valores de `user` y `password`. Ambos se almacenan en las variables locales user y pass. Esto es parte de la funcionalidad del método `doGet` del Servlet.

- En las líneas 18 a 22 se realizan las comparaciones de los valores pasados por parámetro (`user` y `pass`) con valores predeterminados. De resultar verdadera o falsa la comparación se invoca al método `response`, remitiéndole las variables resp, de tipo `HttpServletResponse` y una literal alusiva.

- En las líneas 24 a 32 se programa el método `response`, encargado de brindar respuesta al usuario. Recibe los parámetros `resp` y `msg`. Posteriormente, en las líneas 27 a 31 se crea la respuesta HTML, destacando la línea 29 donde se genera el mensaje a desplegar en el navegador.

8. Ahora se procede a revisar el código generado automáticamente en el archivo web.xml el cual contendrá la configuración del Servlet *Login.java*. Este código se observa en el listado 8.6.

Listado No. 8.6 Código de web.xml del Servlet *Login.java*

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/
    javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.
    org/xml/ns/javaee/web-app_3_1.xsd">
```

```
3.      <servlet>
4.          <servlet-name>Login</servlet-name>
5.          <servlet-class>web.Login</servlet-class>
6.       </servlet>
7.       <servlet-mapping>
8.          <servlet-name>Login</servlet-name>
9.          <url-pattern>/Login</url-pattern>
10.      </servlet-mapping>
11.      <session-config>
12.         <session-timeout>
13.             30
14.         </session-timeout>
15.      </session-config>
16. </web-app>
```

Sobre el código del listado 8.6:

- Las líneas 4 y 5 permiten definir el nombre del Servlet (en este caso `Login`) y la clase donde está ubicado el mismo (`web.Login`).

- En las líneas 8 y 9 se especifican dos etiquetas: el nombre del Servlet sobre el cual se desea establecer un mapeo (`<servlet-name>`) y la URL sobre la que se realizará el mapeo (`<mappin>`). El mapping publica un Servlet para que sea accesible desde el exterior.

Figura 8.11 Ejecución de index.html del proyecto *ServletLogin*

9. Una vez terminado esto, se puede probar el Servlet Login utilizando `doGet`. Se debe observar la **figura 8.11** ejecutando *index.html* y **8.12** ejecutando el Servlet.

Figura 8.12 Ejecución del Servlet *Login.java*

Se puede observar que en el recuadro de la **figura 8.12** aparece la leyenda Login?user=E-GJM&password=12345. Esto significa que por ser un método GET la información pasada a través del navegador incluye los parámetros recibidos por el Servlet. En este caso user=E-GJM&password=12345, lo cual sería peligroso por cuanto se está publicando información sensible. Por ende, en este caso utilizar el método GET no sería lo adecuado; en caso de ser un formulario con datos muy sensibles es mejor usar el método POST. Para probarlo se debe realizar lo siguiente:

Cambiar la línea 11 del listado 8.3 (index.html) que indica que el llamado es a un método GET:

```
<form action= “Login” method=”GET”>
```

Por la expresión (que indicará que es un método POST):

```
<form action= “Login” method=”POST”>
```

Y también se debe cambiar la línea 14 del listado 8.5 que dice:

```
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
```

por lo siguiente (que indica que es un método POST):

```
protected void doPost(HttpServletRequest req, HttpServletResponse resp)
```

10. Se procede a guardar y ejecutar de nuevo el proyecto y se mostrará el resultado del Servlet, como se aprecia en la **figura 8.13**. Se debe observar que ya no se muestran los parámetros de consulta en la URL del navegador.

Figura 8.13 Ejecución del Servlet *Login.java* con el método POST

8.2.1 Uso de cookies con Servlet

Una cookie es un archivo que alberga información en forma de nombre y valor; se almacenan en el navegador web y su propósito es guardar información del usuario cada vez que acceda a un mismo sitio, como el caso de sus preferencias e inclusive usuarios y contraseñas. No deben ser utilizadas para almacenar datos sensibles del usuario por cuanto podrían ser accedidos por un tercero no autorizado; por tanto, este último uso constituye un riesgo de seguridad.

Las cookies se utilizan para almacenar información del usuario en el navegador web para que esté disponible cuando él la requiera. El protocolo HTTP no posee memoria y por ende requiere de algún tipo de mecanismo que le permita recordar algunos datos. Es ahí donde las cookies toman importancia, pues almacenan estos valores en archivos de texto. En Java, los Servlets pueden crear objetos cookies, almacenar valores y remitirlos a un servidor web. En este sentido, una cookie se asocia con una dirección IP o con el dominio del sitio web desde el cual proviene la solicitud HTTP.

Las siguientes instrucciones permiten crear un objeto cookie:

```
Cookie c = new Cookie(“user”, “Enrique”);   // se crea la cookie
c.getName( );  // se obtiene el nombre de la cookie, en este caso nombre c
c.getValue( );  //se obtiene el valor asociado a la cookie de nombre c
```

Las cookies pueden dividirse en dos tipos: no persistente y persistente. Las no persistentes son válidas para una sola sesión del usuario, una vez que se cierra el navegador automáticamente las elimina. Por su parte, las cookies persistentes se utilizan para múltiples sesiones del usuario y no se eliminan cuando alguna se cierra en el navegador, sólo si el usuario o una condición cierran la sesión.

Se pueden citar dos ventajas de las cookies: La primera es que es la técnica más simple de mantener el estado de una sesión y segunda porque se encuentran en el lado del cliente, es decir, en la máquina del usuario.

Aunque también existen desventajas, la primera es que las cookies no funcionan si están deshabilitadas desde el navegador (restricción en el navegador a utilizar cookies) y la segunda es que sólo la información textual se puede configurar en el objeto cookie. A esto hay que agregar la inseguridad de utilizarlas en datos muy sensibles, como la contraseña de un aplicativo web.

Cabe mencionar que el API de los Servlets no permite que se recupere una cookie directamente, sino que es necesaria la recuperación de todo el arreglo además de manipularse para obtener la que se necesita. Estas cookies pueden ser agregadas a la respuesta del servidor mediante el objeto response, utilizando el método addCookie(c) como en el caso del ejemplo anterior.

La **tabla 8.1** muestra los métodos más comunes de un objeto cookie:

Tabla 8.1 Métodos comunes en un objeto cookie:

MÉTODO	DESCRIPCIÓN
`getDomain/setDomain`	Se utiliza para especificar el dominio de proveniencia de una cookie o el dominio donde se almacenará.
`getMaxAge/setMaxAge`	Determina el tiempo de expiración de una cookie (en segundos).
`GetName`	Permite obtener el nombre de una cookie. Se debe utilizar el constructor de la clase para poner el nombre de la misma.
`getValue/setValue`	Se utiliza para especificar el valor asociado de la cookie.

Ejemplo No. 3 Contador de visitas mediante cookies y Servlets

1. Ingresar a NetBeans.

2. Crear un proyecto de tipo web application, como se observa en la **figura 8.3**.

3. El nombre del proyecto será CuentaVisitas, el cual almacenará datos del usuario en cookies para saber cuántas veces se ingresa al Servlet.

4. Crear ahora un Servlet de nombre Contador. Su package será Servlets.

5. Se debe habilitar la opción Add information to deployment descriptor (web.xml) en el paso 3 (Configure Servlet Deployment) de la creación del Servlet (pantalla asistente de la creación del Servlet). Pulsar el botón Terminar.

6. El nuevo proyecto CuentaVisitas tendrá la conformación de directorios y archivos que se muestran en la **figura 8.14**.

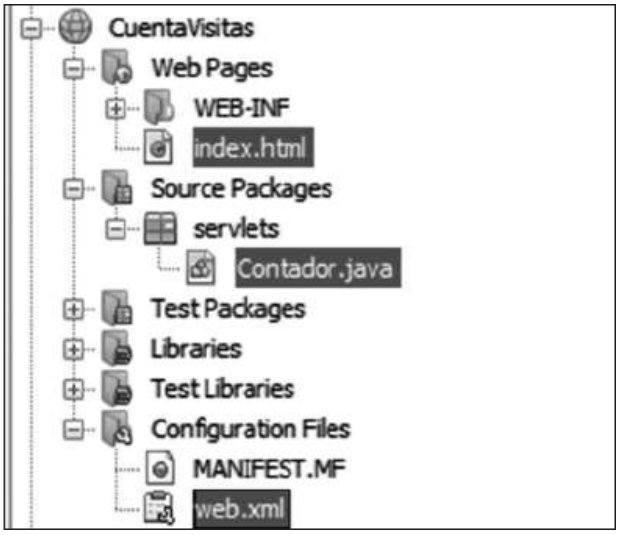


Figura 8.14 Conformación de directorios y archivos de proyecto *CuentaVisitas*

7. A continuación se modifica el archivo index.html, como se muestra en el listado 8.7.

Listado No. 8.7 Código de *index.html* del proyecto *CuentaVisitas*

```
1.  <!DOCTYPE html>
2.  <html>
3.      <head>
4.          <title>Contador de visitas mediante cookies y Servlet</title>
5.          <meta charset="UTF-8">
6.             <meta name="viewport" content="width=device-width, initial-sca-
    le=1.0">
7.      </head>
8.      <body>
9.          <h1> Visitas al Servlet Contador</h1>
10.          <br>
11.          <a href="/CuentaVisitas/Contador">
12.              Enlace al contador de visitas
13.          </a>
14.      </body>
15. </html>
```

8. Lo único que se señala del listado 8.7 es la línea 11 (<a href="/CuentaVisitas/Contador">), la cual hace el llamado al Servlet Contador que se creó con anterioridad y que

a continuación (en el listado 8.8) se modificará en el código. En el siguiente listado se muestra el código del Servlet Contador del proyecto CuentaVisitas.

Listado No. 8.8 Código del Servlet Contador del proyecto CuentaVisitas

```
1.  package servlets;
2.  import java.io.IOException;
3.  import java.io.PrintWriter;
4.  import javax.servlet.ServletException;
5.  import javax.servlet.http.Cookie;
6.  import javax.servlet.http.HttpServlet;
7.  import javax.servlet.http.HttpServletRequest;
8.  import javax.servlet.http.HttpServletResponse;
9.  /**
10.  * @author
11.  */
12. public class Contador extends HttpServlet {
13.     @Override
14.      protected void doGet(HttpServletRequest request, HttpServletResponse
        response)
15.             throws ServletException, IOException {
16.      //Inicializamos la variable cVisitas que nos servirá para contar
17.      //las visitas que se realicen al Servlet
18.          int cVisitas = 0;
19.      //Obtenemos las cookies existentes en el sitio y las almacenamos
20.      //en un vector de Cookies
21.       Cookie[ ] cookies = request.getCookies( );
22.      //Determinamos si el vector de Cookies no está vacio
23.       if (cookies != null) {
24.      //Si existe al menos una Cookie creamos un iterador de objetos
25.      //basados en el vector cookies y cada elemento e de tipo Cookie
26.          for (Cookie c : cookies) {
27.      //buscamos una cookie de nombre eVisitas que es la lleva el contador
28.              if (c.getName( ).equals("cVisitas")) {
```

```
29.                 try {
30.                 //Se obtiene el valor buscado y se convierte a entero
31.                             cVisitas = Integer.parseInt(c.getValue( ));
32.                } catch (NumberFormatException e) {
33.               //En caso de que no exista la cookie reiniciamos el contador en 0
34.                 cVisitas = 0;
35.                        }
36.                    }
37.                }
38.            }
39.        //incrementamos el contador eVisitas
40.         cVisitas++;
41.        //agregamos el contador cVisitas en la cookie del response
42.         Cookie c = new Cookie("cVisitas", Integer.toString(cVisitas));
43.      //se almacenará la cookie en la máquina del cliente por media hora
44.      //(1800 segundos)
45.            c.setMaxAge(1800);
46.            response.addCookie(c);
47.            //Se envia la respuesta del servidor
48.            response.setContentType("text/html");
49.            PrintWriter out = response.getWriter( );
50.            out.println("Este Servlet se ha visitado : " + cVisitas);
51.       }
52.  }
```

9. Como se observa en el listado 8.8 se ha documentado cada línea importante del código, por lo que no es necesario crear una sección de explicación del mismo. A continuación se ejecuta el proyecto; el resultado se mostrará similar a las **figuras 8.15** y **8.16**:

Resumen del código: Como se observa en la **figura 8.16**, cada vez que se invoque al Servlet Contador se incrementará la cookie que se tiene almacenada en el navegador. Basta con pulsar la flecha de retroceso en el navegador e invocar nuevamente al Servlet para que se incremente automáticamente el valor de la cookie. También se puede refrescar en el navegador con la ejecución del Servlet para que el servidor lo ejecute y así se incrementen los valores de la cookie.

Figura 8.15 Ejecución del proyecto *CuentaVisitas*

Figura 8.16 Ejecución del servlet Contador

8.2.2 Manejo de Http Session con Servlet

El API de los Servlet permite administrar las sesiones de los clientes en un navegador mediante HttpSession. Cada vez que un usuario realiza una petición al servidor web se crea, de forma automática, una sesión; inclusive desde la misma máquina pero con diferente navegador. Si se utiliza un mismo navegador pero diferentes ventanas la sesión creada será la misma y, por ende, no aplica la creación de sesiones cuando el navegador es el mismo. La sesión establecida por el usuario administrará las peticiones que realice. Cabe recordar que el protocolo HTTP

no posee estado, por lo tanto no podrá recordar información alguna que el usuario haya enviado de forma previa.

El objeto `HttpSession` se obtiene de `HttpServletRequest`. Una sesión posee un tiempo de vida más largo que una petición o request de usuario. Las sesiones sólo podrán destruirse cuando se termine su tiempo asignado de vida o manualmente a través del método invalidate. Existen diversos métodos que hacen posible el uso de HttpSession y el manejo de las peticiones que realiza un usuario en un aplicativo web. La tabla 8.2 muestra algunos de estos métodos.

Tabla 8.2 Métodos comunes en un objeto `HttpSession`

MÉTODO	DESCRIPCIÓN
`request.getSession( )`	Se utiliza para obtener la sesión creada a partir de la solicitud del usuario.
`session.getAttribute( )`	Permite la obtención de un atributo que se agregó previamente a la sesión del usuario.
`sesion.setAttribute( )`	Permite agregar un atributo a la sesión actual del usuario.
`sesion.removeAttribute( )`	Elimina un atributo a la sesión actual del usuario.
`sesion.invalidate( )`	Invalida la sesión actual del usuario.
`session.isNew( )`	Permite saber si la sesión ha sido recién creada.
`sesion.getCreationTime( )`	Determina la fecha y hora de creación de la sesión.
`sesion.getLastAccesedTime( )`	Determina la última fecha y hora en que el usuario accedió a la sesión.
`sesion.getMaxInactiveInterval( )`	Determina los segundos de inactividad requeridos para que la sesión sea destruida si no recibe alguna petición.
`sesion.setMaxInactiveInterval( )`	Permite modificar el valor de tiempo requerido para destruir una sesión que no reciba alguna petición.

Ejemplo No. 4 Manejo de sesiones mediante Servlets

1. Ingresar a NetBeans

2. Crear un proyecto de tipo web application, como se observa en la **figura 8.3**.

3. Enseguida se crea un proyecto de nombre ProyectoSesion, el cual manejará el estado de sesiones que realiza el usuario.

4. Se crea ahora un Servlet de nombre Sesion. Su package será Servlets.

5. Es preciso asegurarse de habilitar la opción Add information to deployment descriptor (web.xml), en el paso 3 (Configure Servlet Deployment) de la creación del Servlet (pantalla asistente de la creación del Servlet). Dar clic en Terminar.

6. El código del archivo *index.html* será similar al que se muestra en el listado 8.9. No se explicará este código dado que ya se ha hecho en los ejemplos anteriores.

Listado No. 8.9 Código de `index.html` del proyecto ProyectoSesion

```
1.  <!DOCTYPE html>
2.  <html>
3.      <head>
4.          <title>Ejemplo de uso de HttpSession con Servlet</title>
5.          <meta charset="UTF-8">
6.             <meta name="viewport" content="width=device-width, initial-sca-
    le=1.0">
7.      </head>
8.      <body>
9.      <h1>Ejemplo de uso de httpSession</h1>
10.        <br>
11.          <a href="/ProyectoSesion/Sesion">
12.         Enlace al Servlet de manejo de sesiones
13.      </a>
14.    </body>
15. </html>
```

7. Posteriormente se debe agregar el código al Servlet de nombre Sesion.java, el mismo se muestra en el listado 8.10. En este archivo es preciso eliminar todo el código que se muestra por defecto y escribir el que se facilita en el listado 8.10. Las líneas más importantes se han documentado para claridad del código.

Listado No. 8.10 Código del Servlet *Sesion.java* del proyecto ProyectoSesion

```
1.  package servlets;
2.  import java.io.IOException;
```

```
3.  import java.io.PrintWriter;
4.  import javax.servlet.ServletException;
5.  import javax.servlet.http.HttpServlet;
6.  import javax.servlet.http.HttpServletRequest;
7.  import javax.servlet.http.HttpServletResponse;
8.  import javax.servlet.http.HttpSession;
9.  public class Sesion extends HttpServlet {
10.    //sobreescribimos el metodo doGet
11.     @Override
12.      protected void doGet(HttpServletRequest request, HttpServletResponse
       response)
13.              throws ServletException, IOException {
14.         //contenido del método de tipo text/html
15.         response.setContentType(“text/html;charset=UTF-8”);
16.          String mensaje = null;
17.          //se obtiene la sesion actual, puede ser nula de no existir
18.          HttpSession sesion = request.getSession( );
19.          //si no existe el atributo cVisitas se agrega a la sesion creada
           (sesion)
20.          Integer cVisitas = (Integer) sesion.getAttribute(“cVisitas”);
21.          //si es null es que no existe el atributo y se inicializa con el
           valor 1
22.          if (cVisitas == null)
23.          {
24.            cVisitas = 1;
25.            mensaje = “ vez!”; //es primera vez que accede al Servlet
26.          }
27.          Else
28.          {
29.            //si existe se aumenta su contabilidad
30.             mensaje = “ veces!”; //ya es más de una vez que accede al Servlet
31.              cVisitas+=1;  //Se incrementa en número de visitas al Servlet
32.          }
33.          //se actualiza el atributo cVisitas con el nuevo contador cVisitas
```

```
34.         sesion.setAttribute("cVisitas",cVisitas);
            //cVisitas" es el nombre del atributo y cVisitas es el valor del mismo
35.         //se imprimen los valores en el navegador
36.         try (PrintWriter out = response.getWriter( )) {
37.             out.println("<!DOCTYPE html>");
38.             out.println("<html>");
39.             out.println("<head>");
40.             out.println("<title>Servlet Sesion</title>");
41.             out.println("</head>");
42.             out.println("<body>");
43.             out.println("<h1>La sesión " + sesion.getId( ) + "</h1>");
44.             out.println("<h1>Ha ingresado al Servlet " + cVisitas +
45.                         " " + mensaje + "</h1>");
46.             out.println("</body>");
47.             out.println("</html>");
48.         }
49.         }    }
```

8. Por último, se ejecuta el proyecto de sesiones. Las **figuras 8.17** y **8.18** muestran cómo se vería la ejecución de index.html y el llamado a Sesion.java, respectivamente.

Figura 8.17 Ejecución del archivo *index.html* del ProyectoSesion

Resumen del código: En el listado 8.9 del archivo *index.html* se destaca la línea donde se hace referencia al Servlet *sesion*, el cual se encarga de manejar las sesiones que se crean por parte del cliente.

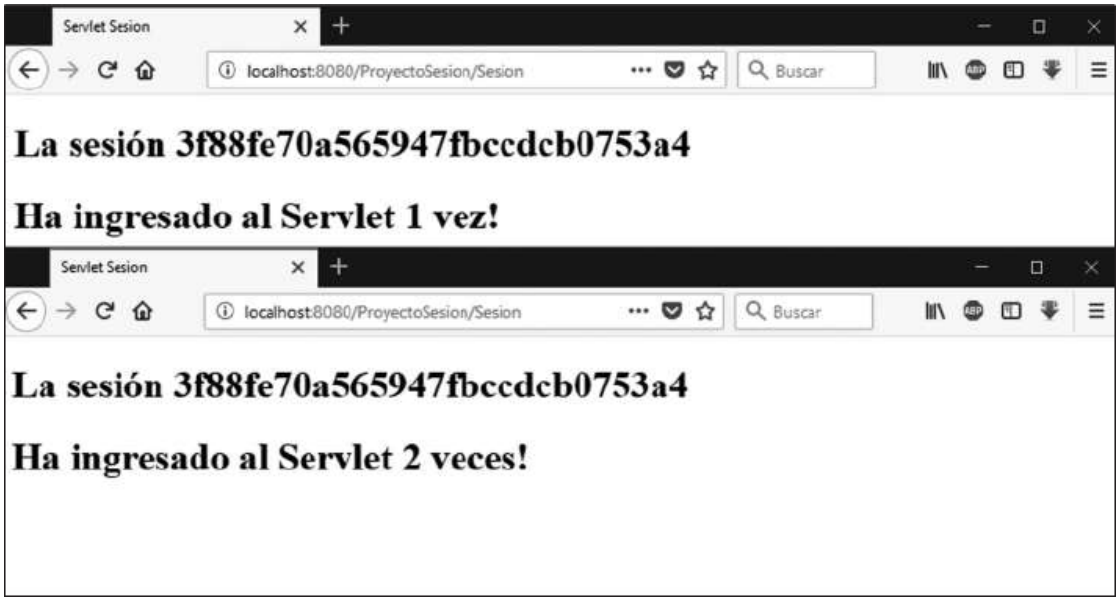


Figura 8.18 Ejecución del llamado al Servlet Sesion.java

Ejemplo No. 5 Utilización de funciones mediante Servlets

1. En este ejemplo se crea una calculadora sencilla en Java web mediante el uso de Servlets. El ejemplo no trata de ser un proyecto completo sino que tiene como finalidad aclarar y desarrollar el tema principal de este capítulo.

2. Crear un proyecto de tipo Web application, como se observa en la **figura 8.3**.

3. Luego crear un proyecto de nombre CalculadoraServlet, el cual devolverá cálculos sencillos de suma, resta, multiplicación y división.

4. Renombrar el archivo index.html como Calculadora.html. Esto se logra posicionando el puntero sobre el archivo index.html, luego dar clic derecho y seleccionar la opción Reestructurar; enseguida Cambiar nombre o simplemente pulsar las teclas CTRL+R, al mismo tiempo.

5. El código que se deberá incluir en el archivo Calculadora.html se muestra en el listado 8.11.

Listado No. 8.11 Código del archivo Calculadora.html

```
1.  <!DOCTYPE html>
2.  <!--
3.  To change this license header, choose License Headers in Project Properties.
4.  To change this template file, choose Tools | Templates
```

```
5.  and open the template in the editor.
6.  -->
7.  <html>
8.  <head>
9.  <title>calculadora con servlet</title>
10. </head>
11. <form action="CalculadorServlet" method="post">
12.     <div>
13.     <input name="operando1" type="text">
14.     </div>
15.     <div>
16.     <input name="operando2" type= "text">
17.     </div>
18.     <div>
19.         <select name = "operacion">
20.           <option value="1" selected>Sumar</option>
21.           <option value="2">Restar</option>
22.           <option value="3">Multiplicar</option>
23.           <option value="4">Dividir</option>
24.      </select>
25.     </div>
26.     <div>
27.     </div>
28. <input type="button" style="margin-top: 100px" name="calcular"
    value="calcular"
29. </form>
30. </body>
31. </html>
```

Sobre el código del listado 8.6:

- La línea 11 invoca a un Servlet llamado `CalculadorServlet` a través del método POST.
- En las líneas 12 a 18 se crean objetos `html` de tipo «`input`» para capturar los datos de entrada para la calculadora. Sus nombres son: operando1 y operando2.

- En las líneas 18 a 25 se crea un «combobox» denominado `operacion` el cual permitirá contener las opciones aritméticas a realizar por la calculadora (sumar, restar, multiplicar y dividir). Se observa que para la opción sumar (línea 20) ésta se encuentra seleccionada por default (selected).

- Finalmente, en la línea 28 se crea el botón `Calcular`, el cual permite invocar al Servlet.

6. Ahora se crea el Servlet `CalculadorServlet.java`, el cual contendrá las operaciones aritméticas requeridas. El listado 8.12 muestra el código fuente de este Servlet.

Listado No. 8.12 Código del Servlet CalculadorServlet.java

```
1.  package servlet;
2.  import java.io.IOException;
3.  import javax.servlet.ServletException;
4.  import javax.servlet.ServletOutputStream;
5.  import javax.servlet.http.HttpServlet;
6.  import javax.servlet.http.HttpServletRequest;
7.  import javax.servlet.http.HttpServletResponse;
8.  /**
9.   *
10.  * @author
11.  */
12. public class CalculadorServlet extends HttpServlet {
13. @Override
14. public void doPost (HttpServletRequest req, HttpServletResponse res)
15.              throws   ServletException,   IOException {
16.    double   op1,   op2,   result; int   operacion;
17.    String   operadores[ ]   =   {"+",   "-",   "*","/"};
18.    ServletOutputStream   out   =   res.getOutputStream( );
19.      op1   =   Double.parseDouble(req.getParameter("operando1"));
20.      op2   =   Double.parseDouble(req.getParameter("operando2"));
21.      operacion   =   Integer.parseInt(req.getParameter("operacion"));
22.      result   =   calcula(op1,   op2,   operacion);
23.      out.println("<html>");
24.      out.println("<head><title>Resultado de calcular con Servlet</title></
         head>");
```

```
25.       out.println("<h1>La   operacion   efectuada es</h1>");
26.       out.println ("<h2>"+ op1+" "+ operadores[operacion-1] +
27.                  " "+   op2   + " = "+ result + "</h2>");
28.       out.println("</body>");
29.       out.println("</html>");
30.       out.close( );
31.  }
32.   public double calcula(double op1, double op2 int operacion)
33.      {
34.       double   result   =   0;
35.       switch   (operacion)
36.           {
37.              case   1: return   op1   +   op2;
38.              case   2: return   op1   -   op2;
39.              case   3: return   op1   *   op2;
40.              case   4: return   op1   /   op2;
41.           }
42.              return   result;
43.     }
44.  }
```

Sobre el código del listado 8.12:

- En la línea 12 se declara el nombre de la clase `CalculadorServlet` la cual, por ser un Servlet, hereda de la clase `HttpServlet`.

- En la línea 13 se sobrescribe el método `doPost`.

- En las líneas 14 y 15 se declara el método doPost, donde se reciben parámetros req (de tipo `HttpServletRequest`, que es donde se recibirán los datos del cliente html invocados al Servlet) y res (de tipo `HttpServletResponse`, que remitirá la respuesta al cliente html que lo invoca).

- En las líneas 16 a 17 se declaran variables de tipo `double`, `int` y un vector de caracteres de nombre `operadores` que contendrá los símbolos de las operaciones aritméticas a procesar por la calculadora.

- La línea 18 declara una variable out de tipo `ServletOutputStream` que servirá para almacenar la información de salida del Servlet.

- En las líneas 19 y 20 se extraen los valores de `operando1` y `operando2` desde el parámetro `req`, convirtiéndolos en `double` y almacenándolos en las variables locales `op1` y `op2`.

- En la línea 22 se invoca a una función del Servlet denominada `calcula`, pasándole los parámetros `op1`, `op2` y `operacion` que es el vector de operandos. La función `calcula` devolverá el resultado de la operación a la variable `result`.

- En las líneas 23 a 30 se procesa la información de salida para mostrarla en el cliente `html`.

- Finalmente, en las líneas 32 a 44 se crea la función `calcula` que permite devolver el resultado de la operación.

7. La **figura 8.19** muestra la estructura que deberá observarse en el ejemplo de la calculadora.

Figura 8.19 Estructura del proyecto *CalculadoraServlet*

8. El resultado de ejecutar este ejemplo se muestra en las **figuras 8.20** y **8.21**.

Figura 8.20 Ejecución del archivo Calculadora.html

Figura 8.21 Ejecución del archivo *CalculadorServlet.java*

Así se llega al final de este capítulo de introducción a Servlets en Java. A pesar de no ser un tratamiento intensivo, se brindan las bases necesarias para incursionar en temas de mayor complejidad. Por ende, las referencias web aportadas al final de esta obra son importantes de revisar, analizar y recrear respecto de los ejercicios que incluyen. También es importante revisar los videos del canal de YouTube de uno de los autores de este texto (Enrique Gómez Jiménez) para retroalimentarse en el tema.

Resumen

En este capítulo se desarrollaron los temas relacionados con los fundamentos de la programación web mediante Servlets Java. No se tratan profundamente dado que la intención es introducir al lector en este tópico tan importante en la programación Java. En resumen, se trataron los siguientes temas:

- Cómo crear un Servlet en Java
- Manejo de parámetros con los métodos GET y POST en un Servlet
- Creación de cookies para el almacenamiento de datos del lado del cliente cuando de usa Servlet
- Manejo de sesiones cuando se trabaja con Servlet en Java

Autoevaluación

1. Según su comprensión ¿A qué se denomina Servlet?

2. ¿Cuál es la diferencia entre un documento .html y .jsp?

3. ¿Es posible utilizar CSS para dar formato a un documento JSP?

4. ¿Para qué se utilizan las cookies en la programación web?

5. ¿Mencione la importancia de utilizar sesiones cuando se programan aplicaciones web?

6. ¿En qué se diferencian las cookies de las sesiones?

Evidencia

- [ ] Escribo código CSS para mejorar la apariencia de los archivos HTML de los ejercicios desarrollados.
- [ ] Conozco y domino sin complicación las respuestas de las preguntas de autoevaluación.
- [ ] Creo un mapa conceptual de los métodos comunes explicados en este capítulo tanto para cookies como para HttpSesion.
- [ ] Poseo capacidad para dotar de más funcionalidades al proyecto No.5 (calculadora) Por ejemplo determinar el factorial de un número, la raíz cuadrada del mismo, entre otras funciones.

☑ Respuestas a las preguntas de autoevaluación:

1. Los Servlets son módulos escritos en Java que se utilizan en un servidor (éste puede ser o no ser un servidor web) para extender sus capacidades de respuesta a peticiones de los usuarios o clientes.

2. La diferencia es que un documento JSP es una extensión de HTML que permite incrustar código Java en el documento HTML estático, lo cual hace que la página pueda tener contenido dinámico, ganando mayor funcionalidad para los usuarios.

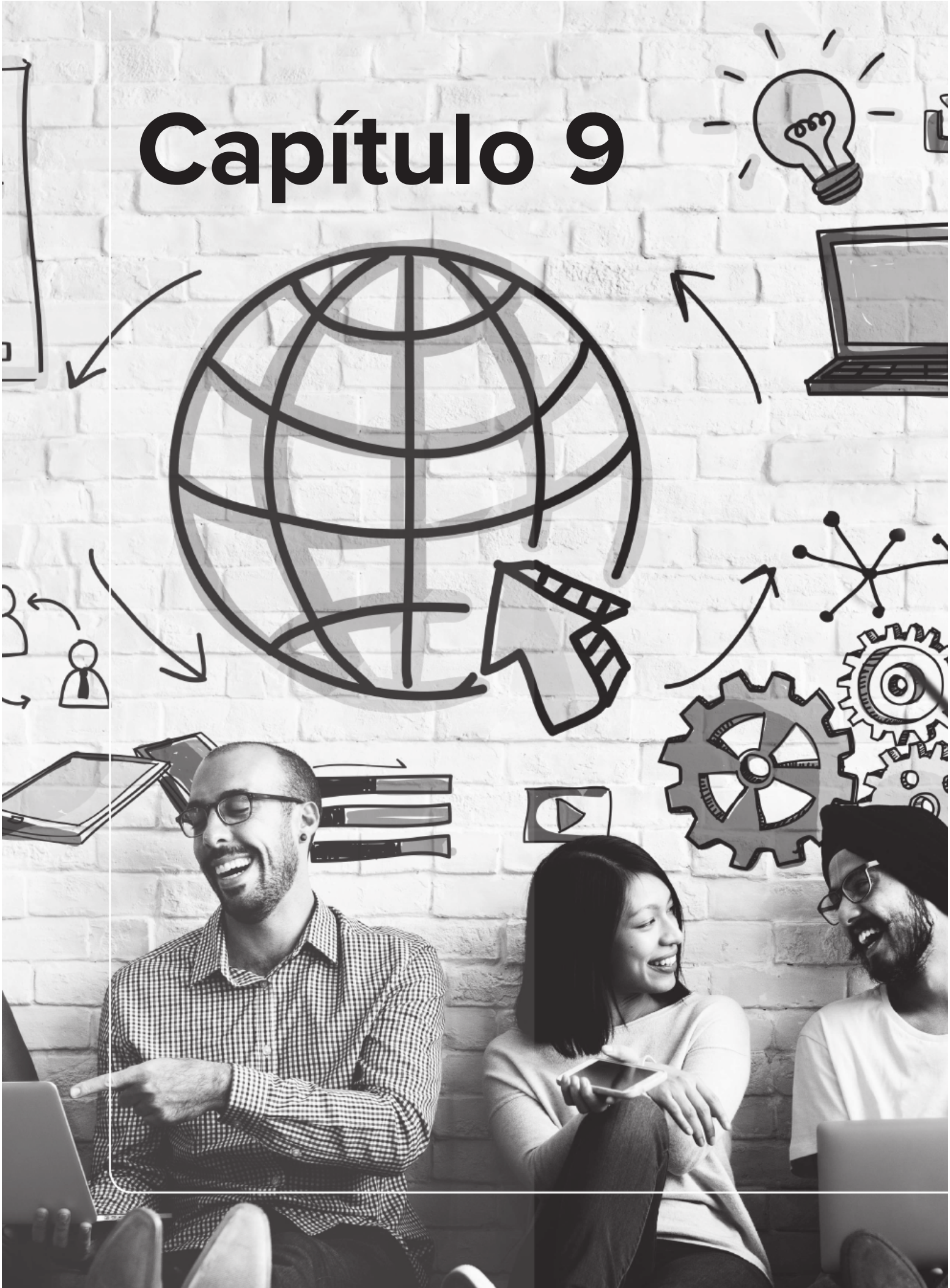
3. Sí, un documento JSP aumenta la funcionalidad de uno HTML pero en esencia sigue siendo HTML; por lo tanto, se le puede dar apariencia y formato mediante CSS.

4. Sirven para almacenar datos importantes para los usuarios en el lado del cliente, de manera que se garantiza que cuando un usuario vuelva a ingresar al sitio web sus datos persistan.

5. Las sesiones son importantes porque permiten almacenar de manera segura datos del lado del servidor, entre ellos mencionar algunos de acceso tipo login a los sitios web. Por otro lado, se puede validar el acceso directo a una URL, dependiendo si se posee la sesión abierta con los privilegios para ingresar o no.

6. Algunas de las diferencias existentes son: Las cookies se almacenan del lado del cliente, las sesiones del servidor, las cookies son inseguras y tienen la desventaja de depender de su activación en cada navegador, las sesiones poseen toda la seguridad con la que cuenta el servidor. Las cookies traen más sobrecarga a la red, las sesiones no; por último, las cookies sobreviven a las caídas del servidor ya que están almacenadas localmente en cada cliente.

Capítulo 9



Java Server Page (JSP)

Reflexione y responda
las siguientes preguntas

¿Qué son los componentes de software del lado del servidor?

¿Cuál es la ventaja de separar la funcionalidad que se ejecuta en el cliente de la que se ejecuta del lado del servidor?

¿Qué tan crítico es ejecutar procedimientos como cálculos o validaciones desde un Servlet en Java?

Contenido

9.1 Introducción
9.2 ¿Qué es un JSP?
- **9.2.1** Ventajas de JSP sobre Servlets
- **9.2.2** Beneficios de JSP
- **9.2.3** Ciclo de vida de una página JSP
- **9.2.4** Interfaces y clases de JSP API

9.3 Java Beans
9.4 Java Standard Tag Library (JSTL)
9.5 JavaBean
9.6 Manejo de EL (Expression Language) con JSP
- **9.6.1** Variables en Expression Language (EL)

Expectativa

El desarrollo de aplicaciones web agrupa una serie de tecnologías orientadas a facilitar el diseño de interfaces y el desarrollo de la lógica que soluciona requerimientos de negocios. Entonces, existen herramientas que permiten diseñar fácilmente formularios que pueden desplegarse en una computadora, dispositivo móvil, una "tablet" e inclusive en un televisor. Pero también está el código detrás (code behind) que permite implementar la lógica para la cual fue concebido un aplicativo.

En este capítulo se incursiona en lo básico que es necesario conocer sobre la parte que se esconde detrás de la fachada de una página web. Todo aquello que hace posible realizar capturas de datos, procesos de cálculo o validación para terminar en una visualización que el usuario espera. Existe variada tecnología implícita en este aspecto y es importante conocer las bases sobre las que se soportan para crear o replicar soluciones, en este caso, con el lenguaje Java.

Después de estudiar este capítulo, el lector será capaz de:

- Comprender la filosofía de programación implícita en el desarrollo de aplicaciones web con Java y, principalmente, con JSP.
- Reconocer las diferentes tecnologías sobre las que se apoya JSP para el desarrollo de páginas web.
- Aplicar tecnologías Java para el desarrollo de páginas web con JSP.

9.1 Introducción

En un escenario global donde las tecnologías informáticas convergen y dan pie a la creatividad y el desarrollo de aplicaciones web de alta complejidad, es necesario incursionar en herramientas que podrán dotar de la facilidad para crear. El mundo de los programadores ya conoce la potencia de .NET, la agresividad de PHP para generar soluciones fantásticas o el poder de Java para desarrollar algunas de alta complejidad. Por ende, familiarizarse con una u otra tecnología es una necesidad imperiosa para quienes se dedican al desarrollo de software.

Lejos de ser una cátedra en el desarrollo de aplicaciones web de alta complejidad, este capítulo incursiona en los aspectos básicos que todo programador que se inicia en Java debe conocer. El aprendizaje puede ser lento para algunos, para otros muy básico (aquí se pueden analizar las referencias web que se aportan), pero es real que existe la necesidad de iniciar de algo mediante el conocimiento de los fundamentos de JSP.

9.2 ¿Qué es un JSP?

JSP (Java Server Pages) es una tecnología que conjuga HTML y código Java dentro de un mismo documento. Para cerrar la brecha que existe entre la capa de presentación (que se ejecuta en el cliente) y las que lo hacen en el servidor, JSP proporciona etiquetas especializadas y un lenguaje de expresión. La tecnología JSP se utiliza para crear aplicaciones web Java que se ejecutan del lado del servidor. Comúnmente se piensa que se trata de una extensión de la tecnología Servlet, dado que proporciona funciones que generan vistas de usuario.

Se puede afirmar que la mayoría de veces se usa JSP con fines de visualización y se encomienda la lógica de negocios en los Servlets y las clases de modelo. Generalmente, cuando se recibe una solicitud de usuario mediante un Servlet, se procesa y se le agregan los atributos en el alcance de dicha solicitud en un documento JSP. También es posible utilizar en este contexto algunos parámetros en la solicitud, encabezados, cookies, sesiones, entre otras, para crear las vistas de respuesta.

9.2.1 Ventajas de JSP sobre Servlets

Anteriormente se dijo que los JSP podrían conceptualizarse como una extensión de los Servlets de Java, sin embargo, se puede entender su diferenciación mediante una lista de ventajas que poseen sobre los Servlets. Algunas son las siguientes:

a) Aunque se pueden generar respuestas HTML a partir de Servlets (como se vio en el capítulo 8), el proceso es engorroso y muchas veces propenso a errores cuando se trata de escribir una respuesta compleja. Escribir todo este código en un Servlet en ocasiones resulta ser tortuoso. JSP, por su parte, colabora en esta situación proporcionando flexibilidad al momento de escribir páginas HTML que generen una visualización en el navegador, aportando el dinamismo del código Java dentro del documento.

b) JSP proporciona funciones, bibliotecas de etiquetas, un lenguaje de expresiones, etiquetas personalizadas, entre otras características que ayudan a un desarrollo más rápido de las vistas estáticas y dinámicas de los usuarios.

c) Las páginas JSP son fáciles de implementar pues requieren solamente el remplazo de la página que se modifica en el servidor, siendo el contenedor el responsable de implementarla. En cambio, los Servlets necesitan volver a compilarse e implementarse.

Cabe entonces mencionar que tanto JSP como los Servlets se complementan entre sí. Una recomendación sana es que se debería utilizar Servlets como controladores que trabajan del lado del servidor y para establecer la comunicación con las clases del modelo, mientras que los JSP deberían utilizarse específicamente en la capa de presentación.

9.2.2 Beneficios de JSP

Al usar las páginas JSP se debe saber cuáles son sus beneficios sobre HTML puro con separación de código. Los siguientes son los beneficios más importantes para el uso de JSP.

a) Está enfocado a la escritura de código HTML, facilita el mantenimiento de la capa de presentación.

b) Brinda la facilidad de que se utilicen herramientas visuales para la creación de páginas HTML con incrustación de etiquetas dinámicas propias de JSP.

c) Permite la separación del código de presentación HTML del código Java propio de JSP.

d) Posibilita la separación de responsabilidades en el desarrollo de software, permite que el equipo de desarrollo pueda realizar distintas tareas en el mismo proyecto.

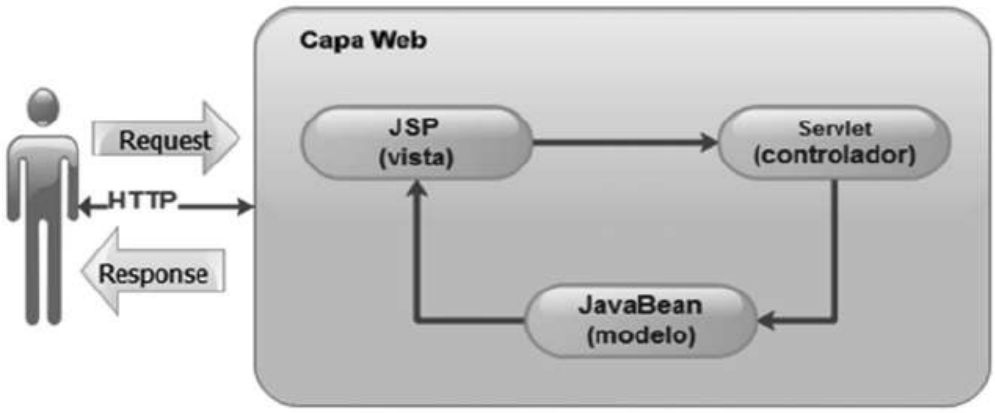


Figura 9.1 Función de una página JSP en un aplicativo web

La funcionalidad de JSP en un aplicativo web se puede analizar mediante un esquema (**figura 9.1**). En esta imagen se puede ver que el usuario se comunica con el aplicativo web mediante una solicitud (request) a través del protocolo HTTP, generándose una vista JSP (con el HTML). Luego, se invoca a un Servlet que eventualmente trabaja como un controlador y gestiona cualquier dato que esté contenido en el modelo (clases Java) y se devuelve la respuesta (response) al cliente nuevamente.

9.2.3 Ciclo de vida de una página JSP

El ciclo de vida de una página JSP se gestiona mediante un contenedor. Estos contenedores web para JSP también lo son para Servlets. La **figura 9.2** muestra las fases del ciclo de vida de una página JSP.

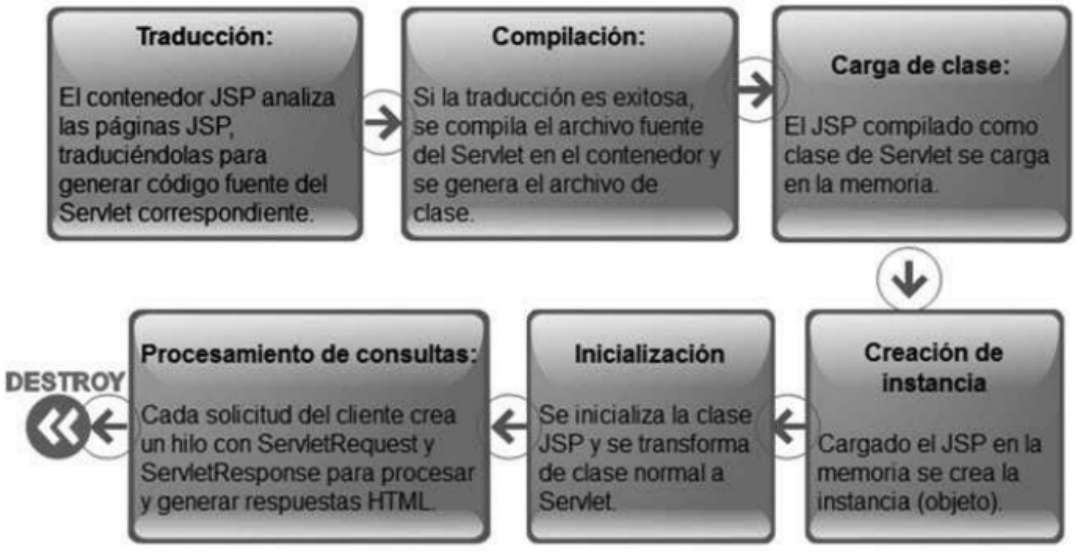


Figura 9.2 Ciclo de vida de una página JSP

En una forma resumida, los documentos JSP atraviesan varias etapas en su ciclo de vida. La inicialización comienza con el método `jspInit( )` que es declarado en la interfaz *JspPage*, llamándose sólo una vez en el ciclo de vida del JSP. La lógica principal del documento JSP se crea a través del método `_jspService( )`, el cual determina el ciclo de vida de la solicitud (request) hasta generar una respuesta (response). Finalmente, el JSP es destruido a través del método `jspDestroy( )`.

A continuación se resumirá una serie de tecnologías presentes en Java para el desarrollo de aplicaciones web. No pretende ser un análisis exhaustivo de cada una de ellas, sino un señalamiento de su existencia; cada una representa una cantidad importante de información que se debería desarrollar; por tanto, es necesario ampliar el conocimiento sobre las mismas

y su funcionamiento, con el estudio de las referencias web aportadas en este capítulo o por investigación propia.

9.2.4 Interfaces y clases de JSP API

Es importante conocer resumidamente las interfaces y clases que ofrece la API de JSP. Esto ayudará a reutilizarlas en nuevos desarrollos y a no desgastar energías en crear codificación existente y, por tanto, innecesaria. La **tabla 9.1** muestra algunas de estas interfaces y clases.

Tabla 9.1 Interfaces y clases de Core JSP API

NOMBRE	TIPO	DESCRIPCIÓN
JspPage	Interfaz	Amplía la interfaz del Servlet declarando métodos del ciclo de vida `jspInit( )` y `jspDestroy( )` en las páginas JSP.
HttpJspPage	Interfaz	Describe la interacción que debe cumplir la interacción de una clase implementada en JSP al utilizarse el protocolo HTTP.
JspWriter	Clase	Es similar a la funcionalidad de PrintWriter en los Servlets y es un soporte adicional a *buffering*.
JspContext	Clase	Se utiliza como clase base para `PageContext`, abstrayendo toda la información que no es específica de los Servlets.
PageContext	Clase	Proporcionan información de contexto útil cuando JSP es utilizado en aplicaciones web, accede en todos los espacios de nombre asociados con esas páginas.
JspFactory	Clase	Permite definir una cantidad de métodos que le permiten a una página JSP en ejecución crear instancias de varias interfaces y clases para respaldar la implementación de JSP.
JspEngineInfo	Clase	Proporciona información sobre el motor de JSP actual.
ErrorData	Clase	Proporciona información sobre errores en páginas de error.

Comentarios en JSP

Una página JSP está construida sobre HTML y, por tanto, se pueden escribir comentarios en el mismo documento. Un comentario en JSP puede ser escrito como:

```
<-- Este es un comentario  HTML en un documento JSP -->
```

El comentario anterior se envía al cliente como parte del documento HTML. Por tanto, el usuario del documento HTML podría visualizarse con la opción de Ver fuente de los navegadores. Si se quiere que el comentario se quede en el servidor y no viaje al cliente, se debe crear de la siguiente manera:

```
<%-- Este es un comentario JSP--%>
```

Declaraciones en JSP

En JSP se declaran los métodos miembros y las variables de la clase Servlet comenzando con <%! y terminando con %>. Por ejemplo, si queremos declarar una variable entera denominada contador; se haría de la siguiente manera:

```
<%! public static int contador = 0; %>
```

Scriptlets en JSP

Los scriplets son etiquetas que permiten facilitar la colocación de código Java en una página JSP. Código Java válido escrito dentro de una página JSP que sea de tipo scriplet se inserta en el método `service( )` del Servlet generado. Una etiqueta scriplet inicia con <% y termina con %>. El siguiente fragmento de código representa un scriplet Java.

```
<%
  float impuesto = 10.5;
  java.util.Date FechaHora = new java.util.Date( );
  out.print("Fecha y hora actual: "+ FechaHora);
%>
```

Expresiones en JSP

La mayoría de veces se usa el método `out.print( )` para imprimir datos dinámicos en páginas JSP, sin embargo, se puede utilizar un atajo mediante expresiones JSP. Una expresión equivale a una sentencia out.println. Estas expresiones inician con <%= y finalizan con %>. Por ejemplo;

```
<% out.print("Java Web"); %>
```

se puede escribir utilizando expresiones JSP como:

```
<%= "Java Web" %>
```

Variables explicitas en JSP

Existen variables que son de tipo explícito en JSP y muy importantes para la gestión de páginas web. Estas variables ya están instanciadas automáticamente en una página JSP y se pueden utilizar. Un resumen de estas variables es el siguiente:

a) Request: Son objetos de tipo `HttpServletRequest` que se utilizan para procesar la solicitud del usuario.

b) Response: Son objetos de tipo `HttpServletResponse` que se utilizan para procesar la respuesta a una solicitud del usuario.

c) Out: Son objetos tipo `JspWriter` que permiten escribir la salida, generalmente en el navegador del usuario.

d) Session: Son objetos de tipo `HttpSession` que se asocian a objetos de tipo request. Se puede deshabilitar el uso de variables de session en aplicaciones web JSP.

e) Application: Son los objetos `ServletContext` que se obtienen a partir del método `getServletContext( )` en un servlet.

Directivas en JSP

Las directivas JSP son utilizadas para dar instrucciones a un contenedor mientras la página JSP es traducida al código fuente del Servlet; permiten configurar información que puede ser utilizada en las páginas JSP. Funciones tales como importar clases, definir páginas de error o incluir páginas JSP dentro de otras, son posibles mediante directivas. Las directivas JSP inician con <%@ y finalizan con %>.

Tres directivas importantes en JSP que se pueden utilizar en las páginas JSP son:

a) Directiva **Page**: Se utiliza para establecer propiedades a una página JSP. Los atributos que contiene son: import, sesion, buffer, autoflush, entre otros.

b) Directiva **Include**: Permite insertar contenido de un archivo JSP en otro.

c) Directiva **TagLib**: Permite utilizar librerías de etiquetas creadas por el propio programador y por terceros; por ejemplo, se puede utilizar la librería de JSTL que se encuentra en http://java.sun.com/jsp/jstl/core.

Se pueden ver en la **tabla 9.2** ejemplos de algunos atributos en las directivas de un JSP.

Tabla 9.2 Atributos de directivas en un JSP

NOMBRE ATRIBUTO	EJEMPLO USO	DESCRIPCIÓN
import	`<%@page import="pack1.class1, ... packN.classN"%>`	El atributo import dentro de la directiva *page* especifica las clases que se van a utilizar dentro de la página JSP
contentType	`<%@ page contentType= "application/vnd.ms-excel" %>`	Especifica el tipo de respuesta que se va a enviar a un cliente. Por ejemplo, la respuesta puede ser la generación de un archivo Excel.
session	`<%@ page session="true" %>`	Permite que el JSP pueda acceder al objeto session que se haya creado anteriormente. Por defecto, el JSP puede acceder a los objetos session. Si se requiere lo contrario se debe poner *false* al valor en el atributo.
isELIgnored	`<%@ page isELIgnored= "false" %>`	Permite deshabilitar el uso de Expression Language. Por defecto, JSP utiliza EL.
buffer	`<%@ page buffer = "tamañoKb" %>`	Permite especificar el tamaño en Kb que puede manejar el buffer de la página JSP. Si llegara a agotar ese tamaño asignado, automáticamente se vaciará todo el contenido de los printwriter o a los out de los Servlets.
errorPage	`<%@ page errorPage = "url de la página JSP donde se desplegará el error" %>`	Permite establecer la página JSP que se redireccionará relativamente si ocurre un error en otra página JSP.
isErrorPage	`<%@ page isErrorPage= "true" %>`	Se maneja conjuntamente con errorPage y se coloca en la página JSP que contendrá el código de error.

9.3 Java Beans

Los Java Beans son clases Java que deben cumplir algunos objetivos: Contener propiedades, que el acceso a ellas sea a través de *get*, *set* e *is* y tener siempre un constructor vacío (aunque puede tener varios constructores con diferentes argumentos). Por tanto, los Java Beans son las clases que normalmente se conocen en varios lenguajes de programación, con los condicionantes citados anteriormente. Más adelante se retomará este tema con la aplicación de un ejemplo. El siguiente fragmento de código es un ejemplo de Java Beans.

```
public class UsuarioInfoBean implements java.io.Serializable
{
  private String nombre;
  private boolean registrado;
  public String getNombre( ) {
```

```
    return nombre;
  }
  public void setNombre(String nombre) {
    this.nombre = nombre;
  }
  public boolean estaRegistrado( ) {
    return registrado;
  }
  public void setRegistrado(boolean registrado) {
    this.registrado = registrado;
  }
}
```

9.4 Java Standard Tag Library (JSTL)

La librería JSTL es un componente que se ubica dentro de la especificación de Java 2 Enterprise Edition (J2EE). Este conjunto de librerías con etiquetas simples y estándares encapsulan la funcionalidad que se utiliza en la escritura de páginas JSP. Tales etiquetas se organizan en cuatro librerías:

a) core: Compuesta por funciones de script básicas como loops, condicionales y operaciones de entrada salida e/s.
b) xml: Establece funcionalidades para el manejo de archivos xml.
c) fmt: Establece funciones de internacionalización y formato de valores tales como monedas y fechas.
d) sql: Establece la funcionalidad para la gestión de bases de datos.

Se argumenta que el código embebido en Java, mediante scriplets, es desordenado. Si se piensa a profundidad quizás se deduzca que el código embebido a través de scriplets en un documento JSP no puede ser reutilizado por otros JSP. Sería necesario rescribir ese código nuevamente en cada JSP que lo requiera.

Entonces, la idea de que las etiquetas JSTL se integren de manera uniforme a las etiquetas HTML ha ganado muchos adeptos. Sin embargo, también existen algunas desventajas de los JSTL, dado que pueden sobrecargar el servidor. Cuando existen scriplets en páginas JSP éstos se copian en el Servlet que se crea cuando se ejecuta; en cambio, los JSTL crean un poco más de código en el Servlet cuando se utilizan. Además, se debe considerar que los scriplets son más potentes que las etiquetas JSTL. El siguiente fragmento de código muestra la impresión de una cadena de texto en una página JSP utilizando JSTL. Se utiliza out para la salida.

```
<c:out value=" Hola mundo con JSTL"/>
```

Ahora, en la práctica se creará una serie de ejemplos que ayuden a comprender la funcionalidad de muchos de estos conceptos.

Ejemplo No. 1 El primer JSP: Hola Mundo

1. Ingresar a NetBeans.

2. Crear un proyecto de tipo web application, como se observa en la figura 8.3 del capítulo anterior.

3. Pulsar el botón Siguiente y seleccionar el nombre `HolaMundoJSP` para el proyecto. Dar clic nuevamente en Terminar. No se necesitan más configuraciones en este proyecto.

4. Ahora pulsar con el botón derecho sobre la carpeta Web Pages del proyecto creado y seleccionar la opción Archivo Nuevo. Se mostrará una pantalla similar a la **figura 9.3**.

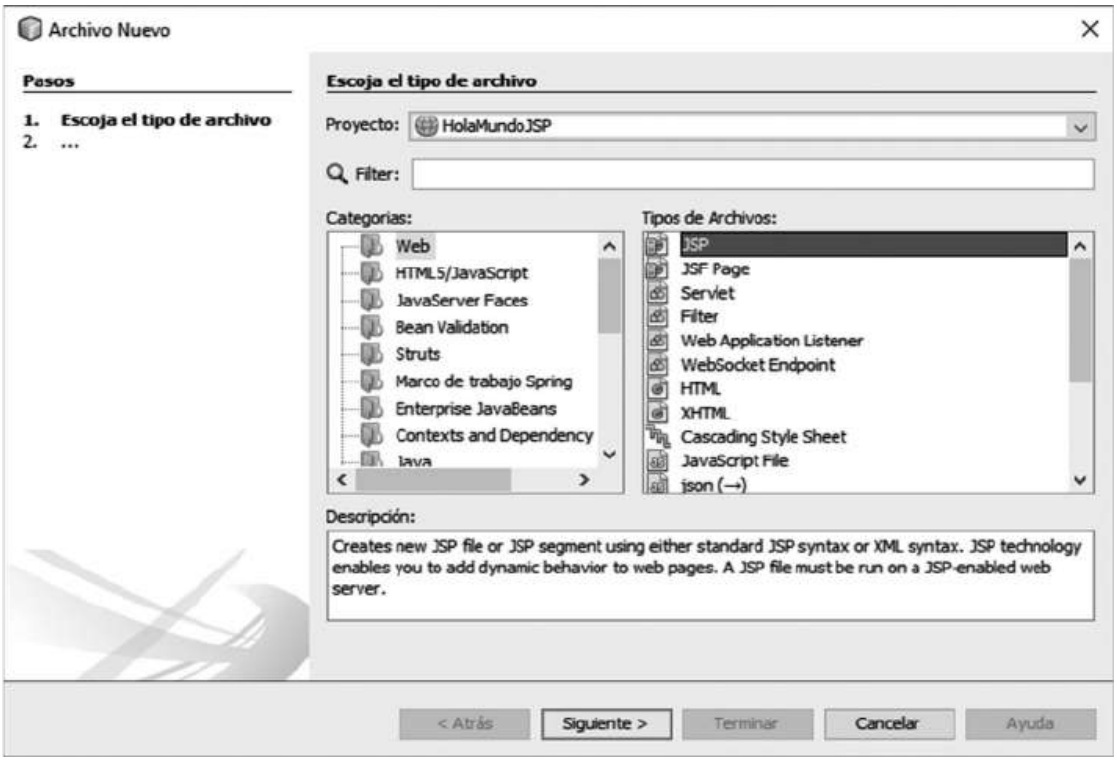


Figura 9.3 Creación de la primera página JSP: Hola Mundo

5. Pulsar la tecla Siguiente y poner el nombre index en File Name o nombre de archivo en la pantalla que aparece. Dar clic en Terminar para finalizar la creación de la página JSP. En este momento se puede observar que la página `index.jsp` se creó en el directorio

WEB-INF donde yacen los archivos HTML también, a diferencia de los Servlet que se crean en el directorio Source Packages.

6. Se elimina el archivo `index.jsp` que se muestra conjuntamente con `index.jsp`, ya que de aquí en adelante se desarrollará con el uso de los archivos JSP en lugar de html para evitar que se ejecute también la página index.html en lugar de `index.jsp`.

7. Ahora se procede a crear el archivo JSP que utiliza todas las formas de mostrar expresiones tanto a nivel de cliente como de servidor. El código de la página `index.jsp` se puede observar en el listado 9.1.

Listado No. 9.1 Código de nuesta página JSP `index.jsp`

```
1.  <%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
2.  <%@page contentType="text/html" pageEncoding="UTF-8"%>
3.  <!DOCTYPE html>
4.  <html>
5.      <head>
6.          <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
7.          <title>Hola Mundo con JSP</title>
8.          <link href="css/estilos.css" rel="stylesheet" type="text/css"/>
9.      </head>
10.     <body>
11.         <h1>Imprimiendo cadenas con diferentes tecnologías JAVA</h1>
12.         <span>Desde el lado del cliente</span>
13.          <ul>
14.             <li><% out.print("Hola mundo con Scriplets");%></li>
15.             <li>${"Hola Mundo con Expression Language (EL)"}</li>
16.             <li> <c:out value="Hola mundo con
                                                    Java Standard Tag
    Library (JSTL)"></c:out></li>
17.         </ul>
18.         <span>Desde el lado del servidor</span>
19.          <ul>
20.             <li>Desplegando la fecha actual con Java Expression.
21.                 Hoy es: <%=new java.util.Date( )%></li>
22.             <li>El valor del parámetro nombre :
23.                 <%=request.getParameter("nombre") %></li>
24.         </ul>
25.     </body>
26. </html>
```

Comentarios sobre el código del listado 9.1:

- La línea 1 hace referencia a una directiva de tipo `TagLib` que hace un llamado a la librería Java Estándar `Tag Library (JSTL)` para poder utilizar el c:out que se encuentra en la fila 16 de este listado.
- En la línea 8 se crea la referencia a la hoja de estilos *estilos.css.*
- La línea 14 realiza una salida por pantalla, utilizando para ello scriplets.
- La línea 15 realiza una salida por pantalla, utilizando para ello el lenguaje de expresiones de Java o Expression Language (EL).
- La línea 16 realiza una salida por pantalla, utilizando para ello el Java Standard Tag Library (JSTL).
- Las líneas 21 y 23 despliegan por pantalla la fecha de hoy y el valor de un parámetro, pero lo hace desde el servidor.

8. Ahora se crea la hoja de estilo *estilos.css* que permitirá dar formato a la salida de la página JSP. El listado 9.2 muestra el código de esta hoja de estilo.

Listado No. 9.2 Código de hoja de estilo *estilos.css*

```
1.  /*Establece reglas de estilo para cabeceras h1 y h2*/
2.  h1, h2
3.  {
4.    text-align: center;
5.    color:tomato;
6.  }
7.  /* Establece reglas de estilo para la etiqueta span*/
8.  Span
9.  {
10.  color: blue;
11.  font-size: 24px;
12.  margin-left: 140px;
13. }
14. /* Establece reglas de estilo para a las listas desordenadas ul*/
15. ul{
16.     font-size: 24px;
17.     margin-left: 100px;
18.     list-style: square;
19. }
```

9. A continuación se procede a ejecutar el archivo *index.jsp* y se obtendrá una pantalla similar a la **figura 9.4**.

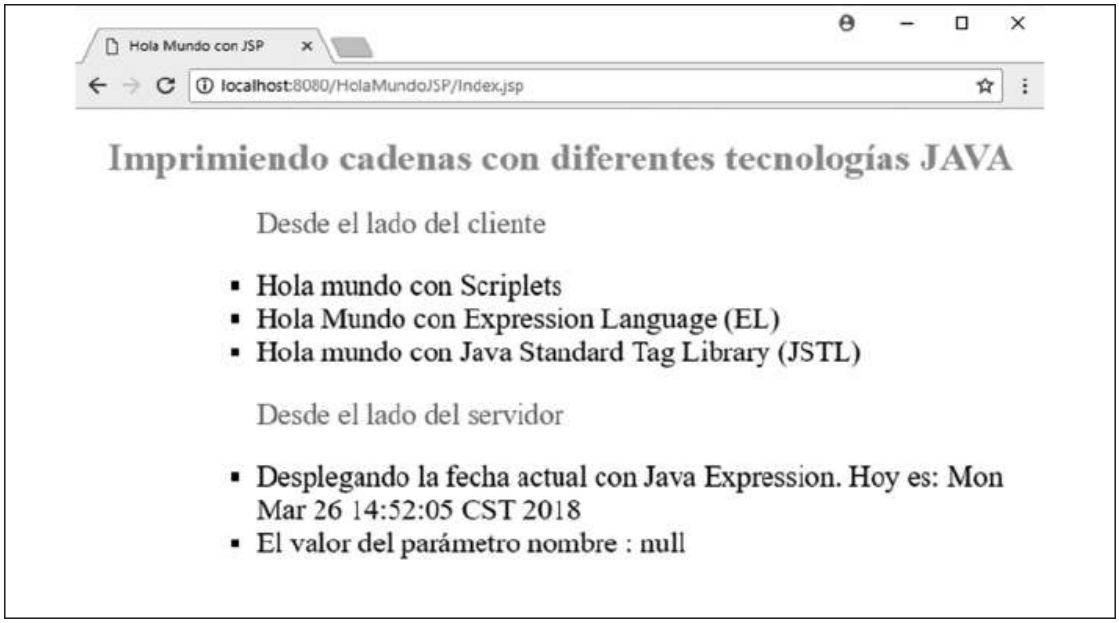


Figura 9.4 Ejecución de la página JSP: Hola Mundo

Ejemplo No. 2 Uso de directivas: Atributos isErrorPage y errorPage de la directiva Page

1. Ingresar a NetBeans.

2. Crear un proyecto de tipo web application, como se observa en la figura 8.3 del capítulo anterior.

3. El nombre del aplicativo web será `jspDirectivas`.

4. Eliminar el archivo *index.html* que se encuentra en el directorio *Web Pages*.

5. Se crea un archivo JSP de nombre *index.jsp*. El código que contendrá esta página *index.jsp* se muestra en el listado 9.3

Listado No. 9.3 Código de la página *index.jsp* del proyecto `jspDirectivas`

```
1.  <%@page contentType="text/html" pageEncoding="UTF-8"%>
2.  <%@page errorPage="paginaError.jsp"%>
3.  <!DOCTYPE html>
4.  <html>
5.      <head>
6.          <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
```

```
7.          <title>Prueba de la Directiva isErrorPage y errorPage</title>
8.      </head>
9.      <body>
10.    <form action="index.jsp" method="POST">
11.        <h1>Dividiendo</h1>
12.      <table>
13.          <tr>
14.              <td>Ingrese dividendo : </td>
15.              <td><input type="text" name="dividendo"></td>
16.          </tr>
17.          <tr>
18.              <td>Ingrese divisor : </td>
19.              <td><input type="text" name="divisor"></td>
20.          </tr>
21.          <tr>
22.              <td>
23.                  <input type="submit" name ="enviar" value="Dividir">
24.              </td>
25.          </tr>
26.      </table>
27.  </form>
28. </body>
29. </html>
30. <%
31. if(request.getParameter("enviar") != null)
32. {
33.     int nDividendo = Integer.parseInt(request.getParameter("dividendo"));
34.     int nDivisor = Integer.parseInt(request.getParameter("divisor"));
35.     int cociente = nDividendo / nDivisor;
36.     out.print("El resultado de la división es : " + cociente );
37. }
38. %>
```

Comentarios sobre el código del listado 9.3:

- En la línea 2 se referencia una directiva `errorPage` que permite redireccionar a la página JSP `paginaError.jsp` si se presenta algún tipo de error en `index.jsp`.

- De las líneas 12 a 26 se crea una tabla que contiene dos `input.text` (dividendo y divisor) que permitirá la captura de dos valores (que espera sean de tipo numérico)

para efectuar una división. Asimismo, contiene un botón de tipo `submit` que permite la ejecución de la división.

- En las filas 30 a 38 se crea un fragmento de código JSP que recibe un parámetro de nombre enviar, es decir, el nombre del botón de envío del formulario (línea 23). Dentro del código se obtienen los parámetros `nDividendo` y `nDivisor`, los cuales se dividen para generar el valor de `cociente`. Si se produce un error en la división se invocará a la página `paginaError.jsp`. En la línea 36 se imprime el resultado.

6. Ahora se procede a crear el código de la página `paginaError.jsp`, mostrado en el listado 9.4.

Listado No. 9.4 Código fuente de la página `paginaError.jsp`

```
1.  <%@page contentType="text/html" pageEncoding="UTF-8"%>
2.  <!DOCTYPE html>
3.  <%@page isErrorPage="true" %>
4.  <html>
5.      <head>
6.          <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Página de error!</title>
7.      </head>
8.      <body>
9.       <h1>Ocurrió un error en la página index.jsp</h1>
10.     </body>
11. </html>
```

Comentarios sobre el código del listado 9.4:

- En la línea 3 se establece la instrucción `<%@page isErrorPage="true" %>`. Una página JSP que contiene este atributo es una donde el contenedor web procederá al reenvío desde la página propensa a excepciones una vez que se produzca la excepción.

- Si en la página `index.jsp` se produce un error entonces se invocará a `paginaError.jsp`, la cual verifica si se presentó alguna anomalía según lo explicado anteriormente, se mostrará el mensaje que se indica en la línea 10.

7. Luego se procede a ejecutar `index.jsp`. La pantalla que se muestra es similar a la de la **figura 9.5**.

Figura 9.5 Ejecución de la página *index.jsp* del proyecto `jspDirectivas`

8. Se puede observar en la **figura 9.5** que la división se ejecuta correctamente. Sin embargo, si se ingresa el valor 0 (cero) en el divisor, se disparará una excepción y se mostrará la página `paginaError.jsp`, como se muestra en la **figura 9.6**.

Figura 9.6 Ejecución de la página *paginaError.jsp*

Para mayor más información acerca del uso de directiva se puede visitar:

http://www.javamexico.org/blogs/kaztle_8/directivas_jsp_jsp_parte_2

9.5 JavaBean

Un **JavaBean** se puede definir como un componente reutilizable independiente de la plataforma y que puede ser manipulado visualmente en una herramienta de diseño. Se trata de clases especiales Java que encapsulan muchos objetos en uno solo (bean). En palabras sencillas, un JavaBeans no es más que una clase de Java. Un JavaBean se diferencia de otra clase Java por lo siguiente:

a) Dispone de un constructor predeterminado, sin argumentos, es decir, vacío.
b) Debe ser serializable, implementando la interfaz *Serializable*.
c) Debe tener atributos que sean privados.
d) Debe contener varias propiedades que se pueden leer o escribir.
e) Debe tener métodos `getter` y `setter` para las propiedades.

Una **propiedad JavaBean** es un atributo al cual se puede tener acceso, puede ser de lectura, escritura, sólo lectura o sólo escritura. ¿Por qué es necesario utilizar los JavaBean? La respuesta es simple: porque encapsula muchos objetos en uno solo, por lo que es fácil acceder a él desde muchos lugares. Además, proporciona una gran facilidad de mantenimiento.

Un JSP puede acceder a los JavaBeans llamando indirectamente a los métodos `get` o `set` asociados a la propiedad indicada en el JSP. El manejo de este acceso a una propiedad se puede representar de la siguiente manera:

a) Un nombre de propiedad válido es `numeroCuenta`.

b) Para esa propiedad deben asociarse los métodos `getNumeroCuenta` (para obtener su valor) y `setNumeroCuenta` (para actualizar el valor de la propiedad).

c) JSP puede obtener el valor de la propiedad con la siguiente instrucción: `<jsp:getProperty … property="numeroCuenta" />` y actualizarla con la instrucción: `<jsp:setProperty … property="numeroCuenta" />`.

Los JavaBeans permiten separar la lógica de ejecución (en el JavaBean) de la presentación (en el Servlet que se genera). Se utilizan también para reducir al máximo el código Java que se inserta en una página JSP; en lugar de agregarlo directamente en el archivo JSP, se crea en un objeto aparte que se puede llamar desde un JSP.

Sin entrar en mucho detalle sobre los JavaBean, se creará un proyecto que demuestre su utilización.

Ejemplo No. 3 Implementación de JavaBean en un proyecto JSP

1. Ingresar a NetBeans.

2. Crear un proyecto de tipo web application, como se observa en la figura 8.3 del capítulo anterior.

3. El nombre del aplicativo web será `jspJavaBean`.

4. Una vez creado el proyecto, se procede a borrar el archivo index.html.

5. A continuación se genera una clase Java, como se muestra en la **figura 9.7**. La clase se denominará `UsuarioInfoBean`.

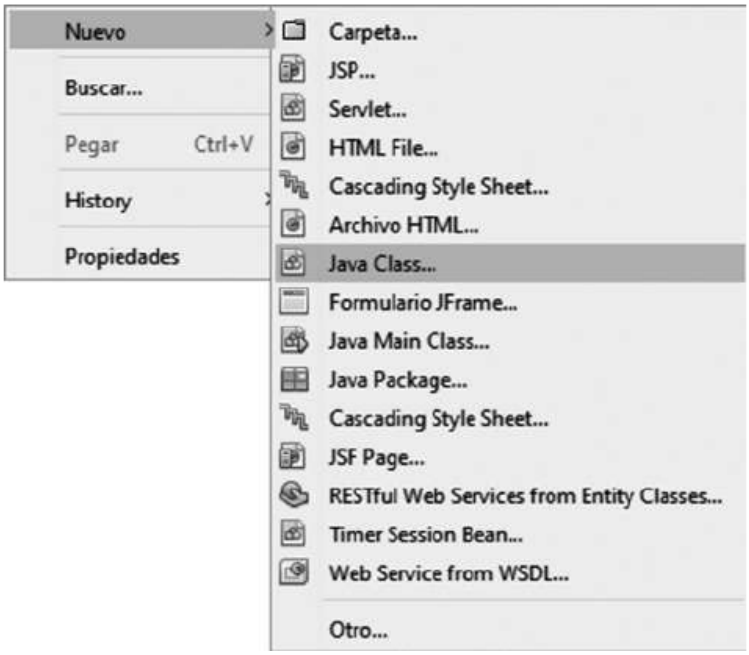


Figura 9.7 Creación de clase Java

6. Ahora se debe crear el código de la clase, el listado 9.5 lo muestra.

Listado No. 9.5 Código de la clase `UsuarioInfoBean`

```
1.  public class UsuarioInfoBean implements java.io.Serializable {
2.      private String nombre = null;
3.      private String apellidos = null;
4.      private String direccion = null;
```

```
5.      public UsuarioInfoBean( ) {
6.      }
7.      public String getNombre( ){
8.         return nombre;
9.      }
10.     public String getApellidos( ){
11.        return apellidos;
12.     }
13.     public String getDireccion( ){
14.        return direccion;
15.     }
16.     public void setNombre(String nombre){
17.        this.nombre = nombre;
18.     }
19.     public void setApellidos(String apellidos){
20.        this.apellidos = apellidos;
21.     }
22.     public void setDireccion(String direccion){
23.        this.direccion = direccion;
24.     }
25. }
```

No se documentará el código anterior porque ya se ha explicado en otros capítulos, principalmente en el 3 que trató el tema de orientación a objetos.

7. Para acceder a JavaBeans se declara el bean convirtiéndose en una variable de scripting a la que se puede acceder tanto en sus elementos como a otras etiquetas personalizadas en el JSP. La sintaxis en este caso es como sigue:

```
<jsp:useBean id = "nombre del bean" scope = "alcance del bean" typeSpec/>
```

Así, el valor del atributo id puede representar a cualquiera que deba ser único entre otras declaraciones del `useBean` en el mismo JSP. Los valores para el atributo de alcance pueden ser una página (page), una solicitud (request), una sesión o una aplicación, según los requisitos.

8. A continuación se crearán dos archivos JSP. El primero se llamará `index.jsp` y el segundo `DatosBean.jsp`.

9. El listado 9.6 muestra el código para la página index.jsp, el cual se encuentra debidamente documentado en sí mismo.

Listado No. 9.6 Código de la página `index.jsp`

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <title>Capturando los datos</title>
    </head>
    <body>
        <!–Se especifica una regla CSS para darle formato a la
            página index.jsp-->
        <div style="background-color:#CECEF6;border-radius: 10px;
             box-shadow: 3px 3px 10px 0px rgba(50, 50, 50, 0.75);
             padding-bottom: 10px;padding-top: 10px;
             padding-left: 20px;
             margin-left:20px;
             margin-right: 1000px;">
       <!--Indicamos que la acción a ejecutar es llamar a otra
           página JSP de nombre DatosBean.jsp, y el método será
           de tipo post-->
        <form action= "DatosBean.jsp" method="post">
        <!--Creamos una tabla para la captura de los datos-->
            <table>
                <tr>
                    <td>Ingresar Nombre: </td>
                    <td><input type="text" name="nombre"/></td>
                </tr>
                <tr>
                    <td>Ingresar Apellidos: </td>
                    <td><input type="text" name="apellidos"/></td>
                </tr>
                <tr>
                    <td>Ingresar Dirección: </td>
                    <td><input type="text" name="direccion"/></td>
                </tr>
            </table>
        <!–Se configura la visualización del botón Enviar datos-->
        <div style="margin-top: 20px;margin-left: 150px;">
```

```
37.             <input type=”submit” value=”Enviar datos”/>
38.             </div>
39.         </form>
40.     </div>
41.    </body>
42. </html>
```

10. Ahora se crea el listado 9.7 de la página DatosBean.jsp. El código de este listado también se encuentra debidamente documentado, por lo que no se explicará cada línea.

Listado No. 9.7 Código de la página DatosBean.jsp

```
1.  <%--     Document: DatosBean
2.       Created on : 27/01/2018, 03:31:11 PM
3.       Author:
4.  --%>
5.  <!--Se importa la clase JavaBean UsuarioInfoBean
6.       que se encuentra en el directorio clases-->
7.  <%@page import=”clases.UsuarioInfoBean”%>
8.  <%@page contentType=”text/html” pageEncoding=”UTF-8”%>
9.  <!DOCTYPE html>
10. <html>
11.     <head>
12.         <meta http-equiv=”Content-Type” content=”text/html; charset=UTF-8”>
13.         <title>Ejemplo de uso de JavaBean en páginas JSP</title>
14.     </head>
15.     <body>
16.         <h1>Mostrando los Datos de JavaBean!</h1>
17.        <!--Se inicia la codificación de la página JSP-->
18.         <%
19.           String Nombre =””, Apellidos=””, Direccion=””;
20.          if(request.getParameter(“nombre”)!=null)
21.           {
22.              /*Se obtienen los parámetros de index.jsp
23.                nombre, apellidos y direccion, los cuales
24.               fueron capturados mediante input de tipo text
25.               */
26.              Nombre=request.getParameter(“nombre”);
```

```
27.              Apellidos=request.getParameter(“apellidos”);
28.              Direccion = request.getParameter(“direccion”);
29.           }
30.           %>
31.           <!--Se invoca el uso del Bean, en este caso se le nombra
              datosSession,
32.               Pero se puede poner cualquier nombre válido. Luego se indica
33.               la clase con su respectivo directorio (clases.
                  UsuarioInfoBean) para
34.               indicar dónde y cuál es el Bean a utilizar. Finalmente, se
                  indica el
35.               scope o alcance que tendrá el manejo del Bean. En este caso, será
36.               solamente para esta sesión-->
37.           <jsp:useBean id=”datosSesion” class=”clases.UsuarioInfoBean”
              scope=”session”/>
38.            <!--Se obtienen los parámetros de Nombre, Apellidos y Direccion,
               para
39.                setearlos a las propiedades de UsuarioInfoBean.-->
40.            <jsp:setProperty name=”datosSesion”
               property=”nombre” value=”<%=Nombre%>”/>
41.            <jsp:setProperty name=”datosSesion”
               property=”apellidos” value=”<%=Apellidos%>”/>
42.            <jsp:setProperty name=”datosSesion” property=”direccion”
               value=”<%=Direccion%>”/>
43.          <table>
44.               <!--Se despliegan los valores de las propiedades mediante la
45.                   obtención en las variables contenidas en datosSesion-->
46.               <tr><td>Nombre    : </td><td><jsp:getProperty
                  name=”datosSesion”
47.                                  property=”nombre”/></td></tr>
48.               <tr><td>Apellidos : </td><td><jsp:getProperty
                  name=”datosSesion”
49.                                  property=”apellidos”/></td></tr>
50.               <tr><td>Dirección : </td><td><jsp:getProperty
                  name=”datosSesion”
51.                                  property=”direccion”/></td></tr>
52.           </table>
53.     </body>
54. </html>
```

11. Una vez codificados tanto la clase Bean como los dos documentos JSP (`index,jsp` y `DatosBean.jsp`), se tendrá una arquitectura del proyecto como se muestra en la **figura 9.8**.

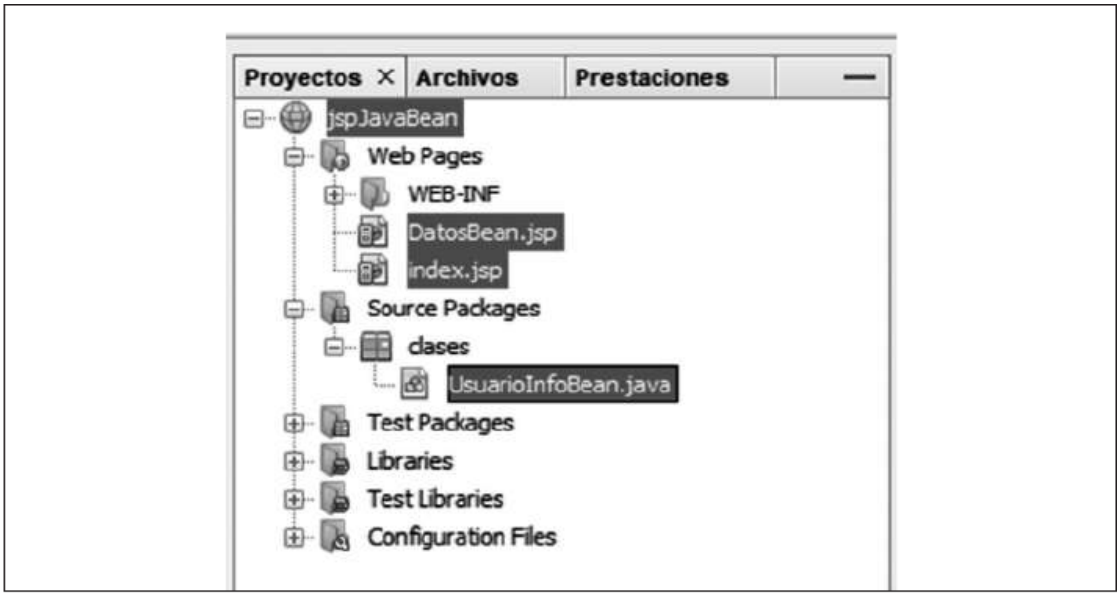


Figura 9.8 Arquitectura del proyecto *jspJavaBean*

12. Si se ejecuta el proyecto, lo primero que se visualizará es el documento `index.jsp`, como se muestra en la **figura 9.9**.

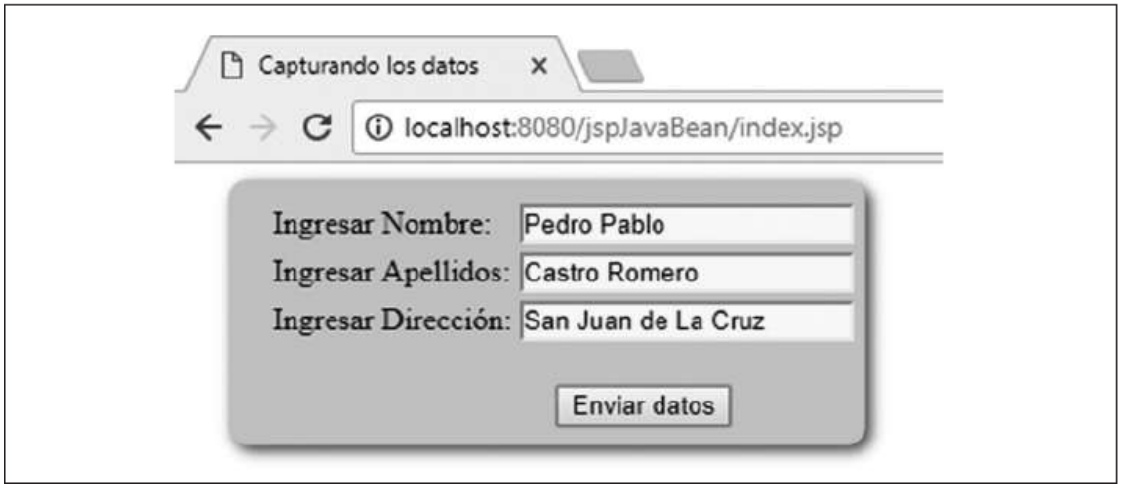


Figura 9.9 Ejecución de `index.jsp` del proyecto `jspJavaBean`

13. Finalmente, si se pulsa el botón Enviar datos se mostrará la página JSP `DatosBean`, como se muestra en la **figura 9.10**.

Figura 9.10 Ejecución de `DatosBean.jsp`

9.6 Manejo de EL (Expression Language) con JSP

EL (Expression Language) simplifica el acceso a los datos que se encuentran almacenados en componentes JavaBeans y en otros objetos, como una solicitud, una sesión, un aplicativo, entre otros. Existen muchos objetos implícitos, operadores, palabras reservadas en Expression Language, éste es utilizado en páginas JSP para interactuar tanto con JavaBeans como con datos de un Servlet, sin importar el alcance de los atributos (request, session, application). Combinándose con la librería JSTL Core (Java Server Tag Library) permite la construcción de toda la lógica de los documentos JSP de una manera mucho más versátil.

Las expresiones que se construyen con Expression Language y que se ejecutan en un documento JSP desde el servidor son evaluadas y reemplazadas, enviándose posteriormente al cliente mediante el HTML contenido en el JSP. Algunas características de EL son:

a) Para utilizarse no requiere ninguna declaración especial en los JSP, simplemente debe cumplir con que sea un archivo JSP válido.

b) Con EL no se permite la utilización de objetos implícitos (o sea que no requieren declaración) y operadores.

c) Mediante EL no es posible iterar con objetos como Array, List, entre otros, por lo que es necesario utilizar JSTL para ello.

d) Para utilizar objetos JavaBean éstos se deben agregar previamente en algún alcance (scope) a través del método `setAttribute` en un Servlet: en el page, request, session o application.

e) Con EL sólo se permite la lectura de información mediante los getters. Es posible obtener y mostrar la información, no así modificarla con los setters. De todos modos, no sería apropiado desde los JSP cambiar la información de esta forma.

9.6.1 Variables en Expression Language (EL)

Para obtener el valor de una variable del tipo que sea, la única condición en EL es escribir el nombre entre la simbología *${}*. Para acceder a la propiedad de un objeto se requiere escribir el nombre, un punto y luego el nombre de la propiedad. Por ejemplo:

```
<c:out value="${usuario.nombre}"/>
```

Si se desea realizar una comparativa entre la sintaxis JSP y la sintaxis EL, se podría resumir la simplicidad del código, la **tabla 9.3** la demuestra al hacer un llamado a una sesión de datos de una clase y la invocación a una propiedad específica.

Tabla 9.3 Comparativa sintaxis JSP y sintaxis EL

SINTAXIS JSP	SINTAXIS EL
`<jsp:useBean id="datosSesion" class="clases.UsuarioInfoBean" scope="session"/>` `<jsp:getProperty name="datosSesion" property="nombre"/>`	`${datosSesion.nombre}`

Con EL se puede tener acceso a variables implícitas tales como `$(pageContent.session.id)` o `${param.nombre}`. Este último se realiza mediante la invocación a valores de parámetros con la palabra reservada `param` o `paramValues`. También podría accederse a las cookies mediante `${cookie.nombreCookie.value}`.

A continuación se desarrollará un ejemplo demostrativo del uso de Expression Language (EL). En este caso, se programará una solución simple para determinar el área de un triángulo.

Según (OnlineMSchool, 2018) "***Triángulo*** *es una figura que se compone de tres puntos que están en una línea recta, y tres segmentos que conectan a pares estos puntos. Los puntos son vértices del triángulo, y los segmentos – sus lados.". En esta ocasión se utilizará el* método de longitud del lado y la altura para determinar y cálcular el área del triángulo. Se procede a crear el ejercicio.

Ejemplo No. 4 Implementación de Expression Language (EL) en un proyecto JSP

1. Debe ingresarse a NetBeans.

2. Se crea un proyecto de tipo web application, como se observa en la **figura 8.3** del capítulo anterior.

3. El nombre del aplicativo web será *ELEjemplo.*

4. Con el proyecto se crea automáticamente un archivo `index.html` el cual debe ser eliminado.

5. A continuación, se crea una clase Java de nombre `Triangulo.java`. El listado 9.8 muestra el código de esta clase.

Listado No. 9.8 Código de la clase `Triangulo.java`

```
1.  package beans;
2.  /**
3.   *
4.   * @author
5.   */
6.  public class Triangulo {
7.      private int a = 0;
8.      private int ha = 0;
9.      public Triangulo(int a, int ha) {
10.         this.a = a;
11.         this.ha = ha;
12.     }
13.     public Triangulo( )
14.     {}
15.     public int getA( ) {
16.         return a;
17.     }
18.     public void setA(int a) {
19.         this.a = a;
20.     }
21.     public int getHa( ) {
22.         return ha;
23.     }
24.     public void setHa(int ha) {
25.         this.ha = ha;
26.     }
27. }
```

6. Posteriormente, se creará un archivo JSP que se llamará `index.jsp`. El código de este archivo JSP se muestra en el listado 9.9.

Listado No. 9.9 Código de la página `index.jsp` del proyecto *ELEjemplo*

```
1.  <%--
2.      Document: index
3.      Created on: 29/01/2018, 08:29:56 PM
4.      Author:
5.  --%>
6.  <%@page contentType="text/html" pageEncoding="UTF-8"%>
7.  <!DOCTYPE html>
8.  <html>
9.      <head>
10.         <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
11.         <title>JSP Page</title>
12.         <style type="text/css">
13.           body {color: blue; }
14.           div {margin-left: 10px;
15.                margin-top: 50px;
16.                margin-right: 10px;
17.                }
18.         </style>
19.     </head>
20.     <body>
21.         <form name="form1" action="resultado.jsp">
22.             <div>
23.                Valor de a   : <input type="text" name="a"/>
24.             </div>
25.             <br>
26.               Valor de ha : <input type="text" name="ha"/>
27.             <br>
28.             <br>
29.               <input type="submit" name="Calcular"/>
30.         </form>
31.     </body>
32. </html>
```

Comentarios sobre el código del listado 9.9:

- En las líneas 12 a 17 se crea una regla CSS en el archivo `index.jsp` que permite establecer el color azul al cuerpo del documento JSP. También permite establecer los márgenes de los objetos que estén contenidos dentro de un `div` en el documento JSP.

- En las líneas 21 a 30 se crea un formulario con objetos de entrada tipo `input` para capturar los datos de base y altura por parte del usuario. En la línea 21 el formulario de nombre `form1` contiene una acción o llamado a otro documento JSP de nombre `resultado.jsp`.

7. Se crea ahora el archivo `resultado.jsp`, el cual tiene como objetivo presentar los datos capturados en la página `index.jsp` mediante Expression Language (EL). El listado 9.10 muestra el código de este nuevo documento JSP, el cual se encuentra debidamente documentado, por lo que no se incluye una sección de explicación del mismo. En el listado se puede observar el código Expression Language (EL) en forma sombreada.

Listado No. 9.10 Código de la página `resultado.jsp`

```
1.  <%--
2.      Document   : resultado
3.      Created on : 29/01/2018, 08:36:03 PM
4.      Author     :
5.  --%>
6.
7.  <%@page contentType="text/html" pageEncoding="UTF-8"%>
8.  <!DOCTYPE html>
9.  <html>
10.     <head>
11.         <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
12.         <title>Calculador área de triángulo con EL y JSP</title>
13.         <style type="text/css">
14.             body {color: blue; }
15.             div {margin-left: 150px;
16.                  margin-top: 10px;
17.                  margin-right: 10px;
18.             }
19.         </style>
20.     </head>
21.     <body>
22.         <!--Se define el uso del beans llamado "triangulo" que
23.          actúa a nivel de page (porque no se le da el scope) y se asocia
24.              con la clase "Triangulo" que se encuentra en el directorio beans-->
25.         <jsp:useBean id="triangulo" class="beans.Triangulo"/>
26.         <!-- Se asignan los valores del formulario al JavaBean para cada una
```

```
27.                 de las variables-->
28.         <jsp:setProperty name="triangulo" property="*" />
29.         <!--    Se obtienen las propiedades "base" y "altura" del
30.                 JavaBean "Triangulo.java"-->
31.         <div>
32.             Valor de a: ${triangulo.a} metros
33.         </div>
34.         <div>
35.             Valor ha: ${triangulo.ha} metros
36.         </div>
37.         <div>
38.             <!--   Se realiza el cálculo del área del triángulo-->
39.        Cálculo del área: ${1/2 * triangulo.a * triangulo.ha}
40.         </div>
41.         <div>
42.             <a href="index.jsp">Volver a calcular</a>
43.         </div>
44.     </body>
45. </html>
```

8. Ahora se prueba el aplicativo web. Primero se ejecuta el formulario `index.jsp`. La **figura 9.11** muestra el resultado de esta ejecución.

Figura 9.11 Ejecución de `index.jsp` del proyecto *ELEjemplo*