



*Si cualquier hombre
prepara su caso y coloca su
nombre al pie de la primera
página, yo le daré una
respuesta inmediata.
Si me obliga a dar vuelta a
la hoja, deberá esperar a mi
conveniencia.*

—Lord Sandwich

*Regla uno: nuestro cliente
siempre tiene la razón.
Regla dos: si piensas que
nuestro cliente está mal,
consulta la Regla uno.*

—Anónimo

*Una pregunta justa debe
ir seguida de un acto en
silencio.*

—Dante Alighieri

*Vendrás aquí y obtendrás
libros que abrirán tus
ojos, oídos y tu
curiosidad; y sacarán tu
interior, o meterán
tu exterior.*

—Ralph Waldo Emerson

Aplicaciones Web: parte I

OBJETIVOS

En este capítulo aprenderá a:

- Desarrollar aplicaciones Web mediante el uso de las tecnologías de Java y Java Studio Creator 2.0.
- Crear JavaServer Pages con componentes JavaServer Faces.
- Crear aplicaciones Web que consistan de varias páginas.
- Validar la entrada del usuario en una página Web.
- Mantener la información de estado acerca de un usuario, con rastreo de sesión y cookies.

26.1	Introducción
26.2	Transacciones HTTP simples
26.3	Arquitectura de aplicaciones multinivel
26.4	Tecnologías Web de Java
26.4.1	Servlets
26.4.2	JavaServer Pages
26.4.3	JavaServer Faces
26.4.4	Tecnologías Web en Java Studio Creator 2
26.5	Creación y ejecución de una aplicación simple en Java Studio Creator 2
26.5.1	Análisis de un archivo JSP
26.5.2	Análisis de un archivo de bean de página
26.5.3	Ciclo de vida del procesamiento de eventos
26.5.4	Relación entre la JSP y los archivos de bean de página
26.5.5	Análisis del XHTML generado por una aplicación Web de Java
26.5.6	Creación de una aplicación Web en Java Studio Creator 2
26.6	Componentes JSF
26.6.1	Componentes de texto y gráficos
26.6.2	Validación mediante los componentes de validación y los validadores personalizados
26.7	Rastreo de sesiones
26.7.1	Cookies
26.7.2	Rastreo de sesiones con el objeto <code>SessionBean</code>
26.8	Conclusión
26.9	Recursos Web
Resumen Terminología Ejercicios de autoevaluación Respuestas a los ejercicios de autoevaluación Ejercicios	

26.1 Introducción

En este capítulo, presentaremos el **desarrollo de aplicaciones Web** con tecnología basada en Java. Las aplicaciones basadas en Web crean contenido Web para los clientes navegadores Web. Este contenido Web incluye el Lenguaje de marcado de hipertexto extensible (XHTML), las secuencias de comandos del lado servidor, imágenes y datos binarios. Para aquellos que no están familiarizados con XHTML, en el CD que se incluye con este libro hay tres capítulos de nuestro libro *Internet & World Wide Web How to Program, 3/e* [Introduction to XHTML: Part 1, Introduction to XHTML: Part 2 y Cascading Style Sheets (CSS)]. En los capítulos 26 a 28, vamos a suponer que usted ya sabe utilizar XHTML.

Este capítulo empieza con las generalidades de la arquitectura de aplicaciones multinivel, y las tecnologías Web de Java para implementar aplicaciones multinivel. Después presentaremos varios ejemplos que demuestran el desarrollo de aplicaciones Web. El primer ejemplo lo introducirá al desarrollo Web de Java. En el segundo ejemplo, crearemos una aplicación Web que simplemente muestra la apariencia visual de varios componentes de GUI de aplicaciones Web. Después, le demostraremos cómo utilizar los componentes de validación y los métodos de validación personalizados para asegurar que la entrada del usuario sea válida antes de enviarla para que el servidor la procese. El capítulo termina con dos ejemplos acerca de cómo personalizar la experiencia de un usuario mediante el rastreo de sesiones.

En el capítulo 27 continuaremos nuestra discusión acerca del desarrollo de aplicaciones Web con conceptos más avanzados, incluyendo los componentes habilitados para AJAX del modelo de programación Java BluePrints de Sun. AJAX ayuda a las aplicaciones basadas en Web a proporcionar la interactividad y capacidad de respuesta que los usuarios esperan comúnmente de las aplicaciones de escritorio.

A lo largo de este capítulo y del capítulo 27 utilizaremos **Sun Java Studio Creator 2.0**: un IDE que ayuda al programador a crear aplicaciones Web mediante el uso de tecnologías de Java, como JavaServer Pages y JavaServer Faces. Para implementar los ejemplos que se presentan en este capítulo, debe instalar Java Studio Creator 2.0, el

cual está disponible para su descarga en developers.sun.com/prodtech/javatools/jscreator/downloads/index.jsp. Las características de Java Studio Creator 2.0 se están incorporando en Netbeans 5.5, mediante un complemento llamado Netbeans Visual Web Pack 5.5 (www.netbeans.org/products/visualweb/).

26.2 Transacciones HTTP simples

El desarrollo de aplicaciones Web requiere una comprensión básica de las redes y de World Wide Web. En esta sección, hablaremos sobre el Protocolo de transferencia de hipertexto (HTTP) y lo que ocurre “tras bambalinas”, cuando un usuario solicita una página Web en un navegador. HTTP especifica un conjunto de métodos y encabezados que permiten a los clientes y servidores interactuar e intercambiar información de una manera uniforme y confiable.

En su forma más simple, una página Web no es más que un documento XHTML: un archivo de texto simple que contiene **marcado** (es decir, **etiquetas**) para describir a un navegador Web cómo mostrar y dar formato a la información del documento. Por ejemplo, el siguiente marcado de XHTML:

```
<title>Mi pagina Web</title>
```

indica que el navegador debe mostrar el texto entre la **etiqueta inicial** `<title>` y la **etiqueta final** `</title>` en la barra de título del navegador. Los documentos XHTML también pueden contener datos de **hipertexto** (lo que se conoce comúnmente como **hipervínculos**) que vinculan a distintas páginas, o a otras partes de la misma página. Cuando el usuario activa un hipervínculo (por lo general, haciendo clic sobre él con el ratón), la página Web solicitada se carga en el navegador Web del usuario.

HTTP utiliza **URIs (Identificadores uniformes de recursos)** para identificar datos en Internet. Los URIs que especifican las ubicaciones de los documentos se llaman **URLs (Localizadores uniformes de recursos)**. Los URLs comunes se refieren a archivos, directorios u objetos que realizan tareas complejas, como búsquedas en bases de datos y en Internet. Si usted conoce el URL de HTTP de un documento XHTML disponible públicamente en cualquier parte en la Web, puede acceder a este documento a través de HTTP.

Un URL contiene la información que dirige a un navegador al recurso que el usuario desea utilizar. Las computadoras que ejecutan software de **servidor Web** hacen disponibles esos recursos. Vamos a examinar los componentes del URL:

```
http://www.deitel.com/libros/descargas.html
```

El `http://` indica que el recurso se debe obtener mediante el protocolo HTTP. La porción intermedia, `www.deitel.com`, es el **nombre del host** completamente calificado del servidor: el nombre del servidor en el que reside el recurso. Esta computadora se conoce comúnmente como **host**, debido a que aloja y da mantenimiento a los recursos. El nombre de host `www.deitel.com` se traduce en una **dirección IP** (68.236.123.125), la cual identifica al servidor así como un número telefónico identifica de forma única a una línea telefónica específica. Esta traducción se lleva a cabo mediante un **servidor del sistema de nombres de dominio (DNS)**: una computadora que mantiene una base de datos de nombres de host y sus correspondientes direcciones IP; a este proceso se le conoce como **búsqueda DNS (DNS lookup)**.

El resto del URL (es decir, `/libros/descargas.html`) especifica tanto el nombre del recurso solicitado (el documento XHTML llamado `descargas.html`) como su ruta o ubicación (`/libros`), en el servidor Web. La ruta podría especificar la ubicación de un directorio actual en el sistema de archivos del servidor Web. Sin embargo, por cuestiones de seguridad, la ruta generalmente especifica la ubicación de un **directorio virtual**. El servidor traduce el directorio virtual en una ubicación real en el servidor (o en otra computadora en la red del servidor), con lo cual se oculta la verdadera ubicación del recurso. Algunos recursos se crean en forma dinámica, por lo que no residen en ninguna parte del servidor. El nombre de host en el URL para dicho recurso especifica el servidor correcto; la ruta y la información sobre el recurso identifican la ubicación del recurso con el que se va a responder a la petición del cliente.

Cuando el navegador Web recibe un URL, realiza una transacción HTTP simple para obtener y mostrar la página Web que se encuentra en esa dirección. En la figura 26.1 se ilustra la transacción en forma detallada, mostrando la interacción entre el navegador Web (el lado cliente) y la aplicación servidor Web (el lado servidor).

En la figura 26.1, el navegador Web envía una petición HTTP al servidor. La petición (en su forma más simple) es

```
GET /libros/descargas.html HTTP/1.1
```

La palabra **GET** es un **método HTTP**, el cual indica que el cliente desea obtener un recurso del servidor. El resto de la petición proporciona el nombre de la ruta del recurso (un documento XHTML), junto con el nombre del protocolo y el número de versión (HTTP/1.1).

Cualquier servidor que entienda HTTP (versión 1.1) puede traducir esta petición y responder en forma apropiada. En la figura 26.2 se muestran los resultados de una petición exitosa. Primero, el servidor responde enviando una línea de texto que indica la versión de HTTP, seguida de un código numérico y una frase que describe el estado de la transacción. Por ejemplo,

```
HTTP/1.1 200 OK
```

indica que se tuvo éxito, mientras que

```
HTTP/1.1 404 Not found
```

informa al cliente que el servidor Web no pudo localizar el recurso solicitado. En la página www.w3.org/Protocols/HTTP/HTRESP.html encontrará una lista completa de códigos numéricos que indican el estado de una transacción HTTP.

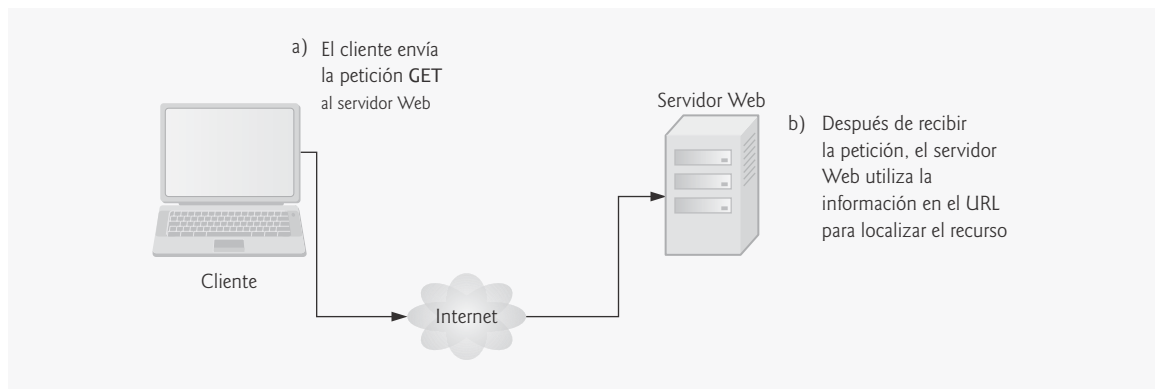


Figura 26.1 | Interacción entre el cliente y el servidor Web. Paso 1: la petición GET.

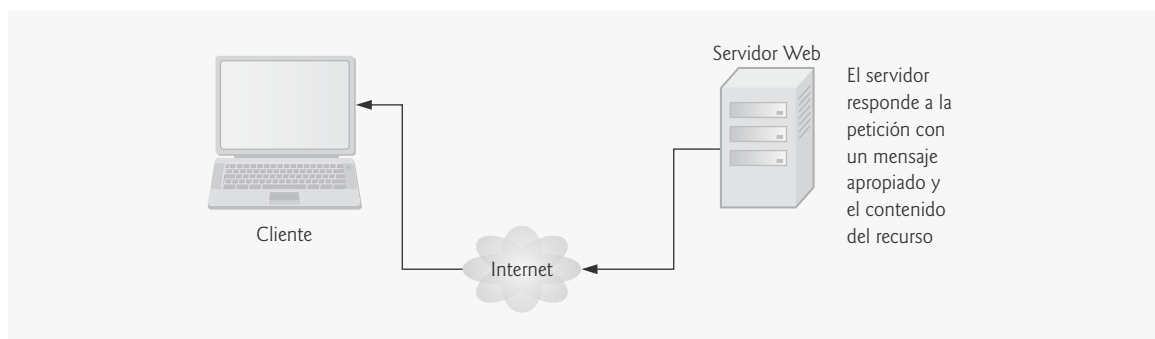


Figura 26.2 | Interacción entre el cliente y el servidor Web. Paso 2: la respuesta HTTP.

Después, el servidor envía uno o más **encabezados HTTP**, los cuales proporcionan información adicional sobre los datos que se van a enviar. En este caso, el servidor está enviando un documento de texto XHTML, por lo que el encabezado HTTP para este ejemplo sería:

```
Content-type: text/html
```

La información que se proporciona en este encabezado especifica el tipo de **Extensiones de correo Internet multipropósito (MIME)** del contenido que el servidor va a transmitir al navegador. MIME es un estándar de

Internet que especifica formatos de datos, para que los programas puedan interpretar los datos en forma correcta. Por ejemplo, el tipo MIME `text/plain` indica que la información enviada es texto que puede mostrarse directamente, sin interpretar el contenido como marcado de XHTML. De manera similar, el tipo MIME `image/jpeg` indica que el contenido es una imagen JPEG. Cuando el navegador recibe este tipo MIME, trata de mostrar la imagen.

El encabezado, o conjunto de encabezados, va seguido por una línea en blanco, la cual indica al cliente que el servidor terminó de enviar encabezados HTTP. Después, el servidor envía el contenido del documento XHTML solicitado (`descargas.html`). El servidor termina la conexión cuando se completa la transferencia del recurso. El navegador del lado cliente analiza el marcado de XHTML que recibe y **despliega** (o visualiza) los resultados.

26.3 Arquitectura de aplicaciones multinivel

Las aplicaciones basadas en Web son **aplicaciones multinivel** (comúnmente conocidas como **aplicaciones de *n* niveles**), que dividen la funcionalidad en **niveles** separados (es decir, agrupaciones lógicas de funcionalidad). Aunque los niveles pueden localizarse en la misma computadora, por lo general, los niveles de las aplicaciones basadas en Web residen en computadoras separadas. En la figura 26.3 se presenta la estructura básica de una **aplicación basada en Web de tres niveles**.

El **nivel inferior** (también conocido como Nivel de datos o nivel de información) mantiene los datos de la aplicación. Por lo general, este nivel almacena los datos en un sistema de administración de bases de datos relacionales (RDBMS). En el capítulo 25 hablamos sobre los sistemas RDBMS. Por ejemplo, una tienda podría tener una base de datos de información sobre el inventario, que contenga descripciones de productos, precios y cantidades en almacén. La misma base de datos podría también contener información sobre los clientes, como los nombres de usuarios, direcciones de facturación y números de tarjetas de crédito. Podría haber varias bases de datos residiendo en una o más computadoras, que en conjunto forman los datos de la aplicación.

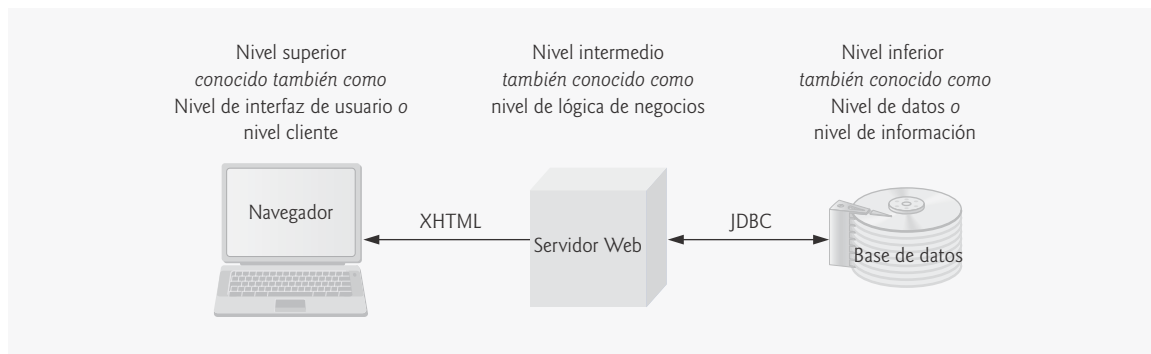


Figura 26.3 | Arquitectura de tres niveles.

El **nivel intermedio** implementa la lógica de negocios, de controlador y de presentación para controlar las interacciones entre los clientes de la aplicación y sus datos. El nivel intermedio actúa como intermediario entre los datos en el nivel de información y los clientes de la aplicación. La **lógica de control** del nivel intermedio procesa las peticiones de los clientes (como las peticiones para ver un catálogo de productos) y obtiene datos de la base de datos. Después, la **lógica de presentación** del nivel intermedio procesa los datos del nivel de información y presenta el contenido al cliente. Por lo general, las aplicaciones Web presentan datos a los clientes en forma de documentos XHTML.

La **lógica comercial** en el nivel intermedio hace valer las **reglas comerciales** y asegura que los datos sean confiables antes de que la aplicación servidor actualice la base de datos, o presente los datos a los usuarios. Las reglas comerciales dictan la forma en que los clientes pueden o no acceder a los datos de la aplicación, y la forma en que ésta procesa los datos. Por ejemplo, una regla comercial en el nivel intermedio de una aplicación basada en Web para una tienda podría asegurar que todas las cantidades de los productos sean siempre positivas. La petición de un cliente de establecer una cantidad negativa en la base de datos de información de productos del nivel inferior sería rechazada por la lógica comercial del nivel intermedio.

El **nivel superior** (nivel cliente) es la interfaz de usuario de la aplicación, la cual recopila los datos de entrada y de salida. Los usuarios interactúan en forma directa con la aplicación a través de la interfaz de usuario, que por lo general es el navegador Web, el teclado y el ratón. En respuesta a las acciones del usuario (por ejemplo, hacer clic en un hipervínculo), el nivel cliente interactúa con el nivel intermedio para hacer peticiones y obtener datos del nivel de información. Después, el nivel cliente muestra los datos obtenidos para el usuario. El nivel cliente nunca interactúa directamente con el nivel de información.

Las aplicaciones multinivel de Java se implementan comúnmente mediante el uso de las características de Java Enterprise Edition (Java EE). Las tecnologías que usaremos para desarrollar aplicaciones Web en los capítulos 26 a 28 son parte de Java EE 5 (java.sun.com/javaee).

26.4 Tecnologías Web de Java

Las tecnologías Web de Java evolucionan en forma continua, para ofrecer a los desarrolladores niveles mayores de abstracción, y una mayor separación de los niveles de la aplicación. Esta separación facilita el mantenimiento y la extensibilidad de las aplicaciones Web. Un diseñador gráfico puede crear la interfaz de usuario de la aplicación sin tener que preocuparse por la lógica de páginas subyacente, la cual estará a cargo de un programador. Mientras tanto, el programador está libre para enfocarse en la lógica comercial de la aplicación, dejando al diseñador los detalles sobre la construcción de una aplicación atractiva y fácil de usar. Java Studio Creator 2 es el paso más reciente en esta evolución, ya que nos permite desarrollar la GUI de una aplicación Web mediante una herramienta de diseño tipo “arrastrar y soltar”, mientras que podemos manejar la lógica comercial en clases de Java separadas.

26.4.1 Servlets

Los servlets son la vista de nivel más bajo de las tecnologías de desarrollo en Java que veremos en este capítulo. Utilizan el modelo petición-respuesta HTTP de comunicación entre cliente y servidor.

Los **servlets** extienden la funcionalidad de un servidor, al permitir que éste genere contenido dinámico. Por ejemplo, los servlets pueden generar en forma dinámica documentos XHTML personalizados, ayudar a proporcionar un acceso seguro a un sitio Web, interactuar con bases de datos a beneficio de un cliente y mantener la información de sesión única para cada cliente. Un componente del servidor Web, conocido como **contenedor de servlets**, ejecuta los servlets e interactúa con ellos. Los paquetes `javax.servlet` y `javax.servlet.http` proporcionan las clases e interfaces para definir servlets. El contenedor de servlets recibe peticiones HTTP de un cliente y dirige cada petición al servlet apropiado. El servlet procesa la petición y devuelve una respuesta apropiada al cliente; por lo general en forma de un documento XHTML o XML (**Lenguaje de marcado extensible**) para mostrarlo en el navegador. XML es un lenguaje que se utiliza para intercambiar datos estructurados en la Web.

Desde el punto de vista arquitectónico, todos los servlets deben implementar a la interfaz **Servlet** del paquete `javax.servlet`, la cual asegura que cada servlet se pueda ejecutar en el marco de trabajo proporcionado por el contenedor de servlets. La interfaz **Servlet** declara métodos que el contenedor de servlets utiliza para administrar el ciclo de vida del servlet. Este ciclo de vida empieza cuando el contenedor de servlets lo carga en memoria; por lo general, en respuesta a la primera petición del servlet. Antes de que el servlet pueda manejar esa petición, el contenedor invoca al método `init` del servlet, el cual se llama sólo una vez durante el ciclo de vida de un servlet para inicializarlo. Una vez que `init` termina su ejecución, el servlet está listo para responder a su primera petición. Todas las peticiones se manejan mediante el **método service** de un servlet, el cual es el método clave para definir la funcionalidad de un servlet. El método `service` recibe la petición, la procesa y envía una respuesta al cliente. Durante el ciclo de vida de un servlet, se hace una llamada al método `service` por cada petición. Cada nueva petición se maneja comúnmente en un subproceso de ejecución separado (administrado por el contenedor de servlets), por lo que cada servlet debe ser seguro para los subprocesos. Cuando el contenedor de servlets termina el servlet (por ejemplo, cuando el contenedor de servlets necesita más memoria o cuando se cierra), se hace una llamada al método `destroy` del servlet para liberar los recursos que éste ocupa.

26.4.2 JavaServer Pages

La tecnología **JavaServer Pages (JSP)** es una extensión de la tecnología de los servlets. El contenedor de JSPs traduce cada JSP y la convierte en un servlet. A diferencia de los servlets, las JSPs nos ayudan a separar la presentación del contenido. Las JavaServer Pages permiten a los programadores de aplicaciones Web crear contenido dinámico mediante la reutilización de componentes predefinidos, y mediante la interacción con componentes que utilizan secuencias de comandos del lado servidor. Los programadores de JSPs pueden utilizar componen-

tes especiales de software llamados JavaBeans, y bibliotecas de etiquetas personalizadas que encapsulan una funcionalidad dinámica y compleja. Un **JavaBean** es un componente reutilizable que sigue ciertas convenciones para el diseño de clases. Por ejemplo, las clases de JavaBeans que permiten operaciones de lectura y escritura de variables de instancias deben proporcionar métodos *obtener (get)* y *establecer (set)* apropiados. El conjunto completo de convenciones de diseño de clases se describe en la especificación de los JavaBeans (java.sun.com/products/javabeans/glasgow/index.html).

Bibliotecas de etiquetas personalizadas

Las **bibliotecas de etiquetas personalizadas** son una poderosa característica de la tecnología JSP, que permite a los desarrolladores de Java ocultar el código para acceder a una base de datos y otras operaciones complejas mediante **etiquetas personalizadas**. Para usar dichas herramientas, sólo tenemos que agregar las etiquetas personalizadas a la página. Esta simpleza permite a los diseñadores de páginas Web, que no estén familiarizados con Java, mejorar las páginas Web con poderoso contenido dinámico y capacidades de procesamiento. Las clases e interfaces de JSP se encuentran en los paquetes `javax.servlet.jsp` y `javax.servlet.jsp.tagext`.

Componentes de JSP

Hay cuatro componentes clave para las JSPs: directivas, acciones, elementos de secuencia de comandos y bibliotecas de etiquetas. Las **directivas** son mensajes para el **contenedor de JSPs**: el componente del servidor Web que ejecuta las JSPs. Las directivas nos permiten especificar configuraciones de páginas, para incluir contenido de otros recursos y especificar bibliotecas de etiquetas personalizadas para usarlas en las JSPs. Las **acciones** encapsulan la funcionalidad en etiquetas predefinidas que los programadores pueden incrustar en JSPs. A menudo, las acciones se realizan con base en la información que se envía al servidor como parte de una petición específica de un cliente. También pueden crear objetos de Java para usarlos en las JSPs. Los **elementos de secuencia de comandos** permiten al programador insertar código que interactúe con los componentes en una JSP (y posiblemente con otros componentes de aplicaciones Web) para realizar el procesamiento de peticiones. Las **bibliotecas de etiquetas** forman parte del **mecanismo de extensión de etiquetas** que permite a los programadores crear etiquetas personalizadas. Dichas etiquetas permiten a los diseñadores de páginas Web manipular el contenido de las JSPs sin necesidad de tener un conocimiento previo sobre Java. La **Biblioteca de etiquetas estándar de JavaServer Pages (JSTL)** proporciona la funcionalidad para muchas tareas de aplicaciones Web comunes, como iterar a través de una colección de objetos y ejecutar instrucciones de SQL.

Contenido estático

Las JSPs pueden contener otro tipo de contenido estático. Por ejemplo, las JSPs comúnmente incluyen marcado XHTML o XML. A dicho marcado se le conoce como **datos de plantilla fija** o **texto de plantilla fija**. Cualquier texto literal en una JSP se traduce en una literal `String` en la representación de la JSP en forma de servlet.

Procesamiento de una petición de JSP

Cuando un servidor habilitado para JSP recibe la primera petición para una JSP, el contenedor de JSPs traduce esa JSP en un servlet, el cual maneja la petición actual y las futuras peticiones a esa JSP. Por lo tanto, las JSPs se basan en el mismo mecanismo de petición-respuesta que los servlets para procesar las peticiones de los clientes, y enviar las respuestas.



Tip de rendimiento 26.1

Algunos contenedores de JSPs traducen las JSPs en servlets al momento de desplegar las JSPs (es decir, cuando la aplicación se coloca en un servidor Web). Esto elimina la sobrecarga de la traducción para el primer cliente que solicita cada JSP, ya que la JSP se traducirá antes de que un cliente la haya solicitado.

26.4.3 JavaServer Faces

JavaServer Faces (JSF) es un marco de trabajo para aplicaciones Web que simplifica el diseño de la interfaz de usuario de una aplicación, y separa aún más la presentación de una aplicación Web de su lógica comercial. Un **marco de trabajo (framework)** simplifica el desarrollo de aplicaciones, al proporcionar bibliotecas y (algunas veces) herramientas de software para ayudar al programador a organizar y crear sus aplicaciones. Aunque el marco

de trabajo JSF puede usar muchas tecnologías para definir las páginas en las aplicaciones Web, este capítulo se enfoca en las aplicaciones JSF que utilizan JavaServer Pages. JSF proporciona un conjunto de componentes de interfaz de usuario, o **componentes de JSF** que simplifican el diseño de páginas Web. Estos componentes son similares a los componentes de Swing que se utilizan para crear aplicaciones con GUI. JSF proporciona dos bibliotecas de etiquetas personalizadas de JSP para agregar estos componentes a una página JSP. JSF también incluye APIs para manejar eventos de componentes (como el procesamiento de los cambios de estado de los componentes y la validación de la entrada del usuario), navegar entre las páginas de una aplicación Web y mucho más. El programador diseña la apariencia visual de una página con JSF, agregando etiquetas a un archivo JSP y manipulando sus atributos. El programador define el comportamiento de la página por separado, en un archivo de código fuente de Java relacionado.

Aunque los componentes estándar de JSF son suficientes para la mayoría de las aplicaciones Web básicas, también podemos escribir bibliotecas de componentes personalizados. Hay bibliotecas de componentes adicionales, disponibles en el proyecto **Java BluePrints**, el cual muestra las mejores prácticas para desarrollar aplicaciones en Java. Muchos otros distribuidores ofrecen bibliotecas de componentes de JSF. Por ejemplo, Oracle proporciona alrededor de 100 componentes en su biblioteca ADF Faces. Aquí hablaremos sobre una de esas bibliotecas de componentes, conocida como BluePrints AJAX (blueprints.dev.java.net/ajaxcomponents.html). En el siguiente capítulo hablaremos sobre los componentes de Java BluePrints para crear aplicaciones de JSF habilitadas para AJAX.

26.4.4 Tecnologías Web en Java Studio Creator 2

Las aplicaciones Web de Java Studio Creator 2 consisten en una o más páginas Web JSP, integradas en el marco de trabajo JavaServer Faces. Estos archivos JSP tienen la extensión de archivo `.jsp` y contienen los elementos de la GUI de la página Web. Las JSPs también pueden contener JavaScript para agregar funcionalidad a la página. Las JSPs se pueden personalizar en Java Studio Creator 2 al agregar componentes de JSF, incluyendo etiquetas, campos de texto, imágenes, botones y otros componentes de GUI. El IDE nos permite diseñar las páginas en forma visual, al arrastrar y soltar estos componentes en una página; también podemos personalizar una página Web al editar el archivo `.jsp` en forma manual.

Cada archivo JSP que se crea en Java Studio Creator 2 representa una página Web, y tiene su correspondiente clase `JavaBean`, denominada **bean de página**. Una clase `JavaBean` debe tener un constructor predeterminado (o sin argumentos), junto con métodos *obtener* (*get*) y *establecer* (*set*) para todas las propiedades del bean (es decir, las variables de instancia). El bean de página define las propiedades para cada uno de los elementos de la página. El bean de página también contiene los manejadores de eventos y los métodos de ciclo de vida de las páginas para administrar tareas, como la inicialización y despliegue de las páginas, y demás código de soporte para la aplicación Web.

Toda aplicación Web creada con Java Studio Creator 2 tiene otros tres `JavaBeans`. El objeto **RequestBean** se mantiene en **ámbito de petición**; este objeto existe sólo mientras dure una petición HTTP. Un objeto **SessionBean** tiene **ámbito de sesión**; el objeto existe durante una sesión de navegación del usuario, o hasta que se agota el tiempo de la sesión. Hay un único objeto `SessionBean` para cada usuario. Por último, el objeto **ApplicationBean** tiene **ámbito de aplicación**; este objeto es compartido por todas las instancias de una aplicación y existe mientras que la aplicación esté desplegada en un servidor Web. Este objeto se utiliza para almacenar datos a nivel de aplicación o para procesamiento; sólo existe una instancia para la aplicación, sin importar el número de sesiones abiertas.

26.5 Creación y ejecución de una aplicación simple en Java Studio Creator 2

Nuestro primer ejemplo muestra la hora del día del servidor Web en una ventana del navegador. Al ejecutarse, este programa muestra el texto "Hora actual en el servidor Web", seguido de la hora del servidor Web. La aplicación contiene una sola página Web y, como mencionamos antes, consiste de dos archivos relacionados: un archivo JSP (figura 26.4) y un archivo de bean de página de soporte (figura 26.6). La aplicación tiene también los tres beans de datos con ámbito para los ámbitos de petición, sesión y aplicación. Como esta aplicación no almacena datos, estos beans no se utilizan en este ejemplo. Primero hablaremos sobre el marcado en el archivo JSP, el código en el archivo de bean de página y la salida de la aplicación; después proporcionaremos instrucciones

detalladas para crear el programa. [Nota: el marcado en la figura 26.4 y en los demás listados de archivos JSP en este capítulo es el mismo que el marcado que aparece en Java Studio Creator 2, pero hemos cambiado el formato de estos listados para fines de presentación, para que el código sea más legible].

Java Studio Creator 2 genera todo el marcado que se muestra en la figura 26.4 cuando establecemos el título de la página Web, arrastramos dos componentes **Texto estático** en la página y establecemos las propiedades de estos componentes. Los componentes **Texto estático** muestran texto que el usuario no puede editar. En breve le mostraremos estos pasos.

```

1  <?xml version = "1.0" encoding = "UTF-8"?>
2
3  <!-- Fig. 26.4: Hora.jsp -->
4  <!-- Archivo JSP generado por Java Studio Creator 2, que muestra -->
5  <!-- la hora actual en el servidor Web -->
6  <jsp:root version = "1.2"
7      xmlns:f = "http://java.sun.com/jsp/core"
8      xmlns:h = "http://java.sun.com/jsp/html"
9      xmlns:jsp = "http://java.sun.com/JSP/Page"
10     xmlns:ui = "http://www.sun.com/web/ui">
11     <jsp:directive.page contentType = "text/html; charset = UTF-8"
12         pageEncoding = "UTF-8"/>
13     <f:view>
14         <ui:page binding = "#{Hora.page1}" id = "page1">
15             <ui:html binding = "#{Hora.html1}" id = "html1">
16                 <ui:head binding = "#{Hora.head1}" id = "head1"
17                     title = "Hora Web: un ejemplo simple">
18                     <ui:link binding = "#{Hora.link1}" id = "link1"
19                         url = "/resources/styleSheet.css"/>
20                     </ui:head>
21                     <ui:meta content = "60" httpEquiv = "refresh" />
22                     <ui:body binding = "#{Hora.body1}" id = "body1"
23                         style = "-rave-layout: grid">
24                         <ui:form binding = "#{Hora.form1}" id = "form1">
25                             <ui:staticText binding = "#{Hora.encabezadoHora}" id =
26                                 "encabezadoHora" style = "font-size: 18px; left: 24px;
27                                 top: 24px; position: absolute" text = "Hora actual
28                                 en el servidor Web :"/>
29                             <ui:staticText binding = "#{Hora.textoReloj}" id =
30                                 "textoReloj" style = "background-color: black;
31                                 color: yellow; font-size: 18px; left: 24px; top:
32                                 48px; position: absolute"/>
33                         </ui:form>
34                     </ui:body>
35                 </ui:html>
36             </ui:page>
37         </f:view>
38     </jsp:root>

```

Figura 26.4 | Archivo JSP generado por Java Studio Creator 2, que muestra la hora actual en el servidor Web.

26.5.1 Análisis de un archivo JSP

Los archivos JSP que se utilizan en este ejemplo (y los siguientes) se generan casi completamente mediante Java Studio Creator 2, el cual proporciona un Editor visual que nos permite crear la GUI de una página al arrastrar y soltar componentes en un área de diseño. El IDE genera un archivo JSP en respuesta a las interacciones del programador. En la línea 1 de la figura 26.4 está la declaración XML, la cual indica que la JSP está expresada en sintaxis XML, junto con la versión de XML que se utiliza. En las líneas 3 a 5 hay comentarios que agregamos a la JSP, para indicar su número de figura, nombre de archivo y propósito.

En la línea 6 empieza el elemento raíz para la JSP. Todas las JSPs deben tener este elemento **jsp:root**, el cual tiene un atributo **version** para indicar la versión de JSP que se está utilizando (línea 6), y uno o más atributos **xmlns** (líneas 7 a 10). Cada **atributo xmlns** especifica un prefijo y un URL para una biblioteca de etiquetas, lo cual permite a la página usar las etiquetas especificadas en esa biblioteca. Por ejemplo, la línea 9 permite a la página usar los elementos estándar de las JSPs. Para usar estos elementos, hay que colocar el prefijo **jsp** antes de la etiqueta de cada elemento. Todas las JSPs generadas por Java Studio Creator 2 incluyen las bibliotecas de etiquetas especificadas en las líneas 7 a 10 (la biblioteca de componentes JSF básicos, la biblioteca de componentes JSF de HTML, la biblioteca de componentes JSP estándar y la biblioteca de componentes JSF de interfaz de usuario).

En las líneas 11 y 12 se encuentra el elemento **jsp:directive.page**. Su atributo **contentType** especifica el tipo MIME (**text/html**) y el conjunto de caracteres (UTF-8) que utiliza la página. El atributo **pageEncoding** especifica la codificación de caracteres que utiliza el origen de la página. Estos atributos ayudan al cliente (por lo general, un navegador Web) a determinar cómo desplegar el contenido.

Todas las páginas que contienen componentes JSF se representan en un **árbol de componentes** (figura 26.5) con el elemento JSF raíz **f:view**, que es de tipo **UIViewRoot**. Para representar la estructura de este árbol de componentes en una JSP, se encierran todas las etiquetas de los componentes JSF dentro del elemento **f:view** (líneas 13 a 37).

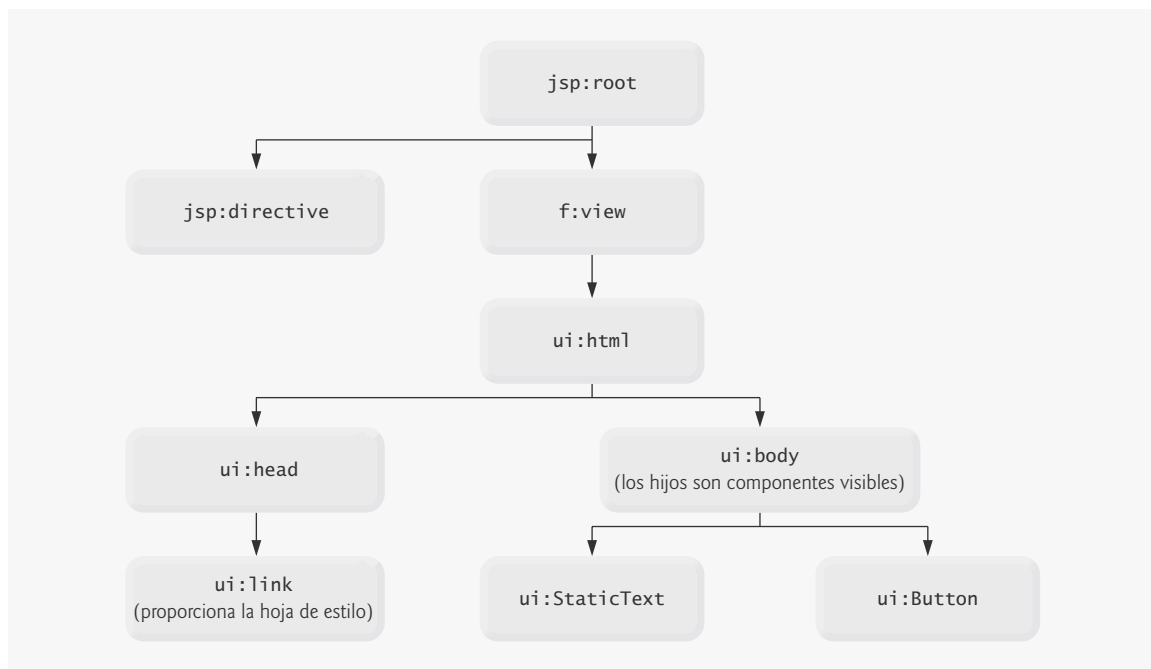


Figura 26.5 | Árbol de componentes JSF de ejemplo.

En las líneas 14 a 20 empieza la definición de la JSP con las etiquetas **ui:page**, **ui:html** y **ui:head**, todas de la biblioteca de etiquetas **ui** (componentes JSF de interfaz de usuario). Éstos y muchos otros elementos de página tienen un atributo **binding**. Por ejemplo, el elemento **ui:head** (línea 16) tiene el atributo **binding** = `"#{Hora.head}"`. Este atributo utiliza la notación del **Lenguaje de expresiones JSF** (es decir, `#{Hora.head}`) para hacer referencia a la propiedad **head** en la clase **Hora** que representa al bean de página (en la figura 26.6 podrá ver esta clase). Es posible enlazar un solo atributo de un elemento JSP a una propiedad en cualquiera de los JavaBeans de la aplicación Web. Por ejemplo, el atributo **text** de un componente **ui:label** se puede enlazar a una propiedad **String** en el objeto **SessionBean** de la aplicación. En la sección 26.7.2 veremos un ejemplo de esto.

El elemento **ui:head** (líneas 16 a 20) tiene un atributo **title** que especifica el título de la página. Este elemento también contiene un elemento **ui:link** (líneas 18 y 19), el cual especifica la hoja de estilo CSS que utiliza la página. El elemento **ui:body** (líneas 22 a 34) contiene un elemento **ui:form** (líneas 24 a 33), el cual contiene

dos componentes `ui:staticText` (líneas 25 a 28 y 29 a 32). Estos componentes muestran el texto de la página. El componente `encabezadoHora` (líneas 25 a 28) tiene un atributo `text` (líneas 27 y 28) que especifica el texto a mostrar (es decir, "Hora actual en el servidor Web:"). El componente `textoReloj` (líneas 29 a 32) no especifica un atributo de texto, ya que el texto de este componente se establecerá mediante programación.

Para que el marcado en este archivo se muestre en un navegador Web, todos los elementos de la JSP se asignan automáticamente a elementos de XHTML que el navegador reconoce. El mismo componente Web se puede asignar a varios elementos de XHTML distintos, dependiendo del navegador Web cliente y de las configuraciones de las propiedades del componente. En este ejemplo, los componentes `ui:staticText` (líneas 25 a 28, 29 a 32) se asignan a elementos `span` de XHTML. Un elemento `span` contiene texto que se muestra en una página Web, y que comúnmente se utiliza para controlar el formato del texto. Los atributos `style` de un elemento `ui:staticText` de una JSP se representan como parte del correspondiente atributo `style` del elemento `span` cuando el navegador despliega la página. En un momento le mostraremos el documento XHTML que se produce cuando un navegador solicita la página `Hora.jsp`.

26.5.2 Análisis de un archivo de bean de página

En la figura 26.6 se presenta el archivo de bean de página. En la línea 3 se indica que esta clase pertenece al paquete `horaweb`. Esta línea se genera automáticamente y especifica el nombre del proyecto como el nombre del paquete. En la línea 17 empieza la declaración de la clase `Hora` e indica que hereda de la clase `AbstractPageBean` (del paquete `com.sun.rave.web.ui.appbase`). Todas las clases de bean de página que soportan archivos JSP con componentes JSF deben heredar de la clase abstracta `AbstractPageBean`, la cual proporciona métodos para el ciclo de vida de las páginas. Observe que el IDE hace que el nombre de la clase coincida con el nombre de la página. El paquete `com.sun.rave.web.ui.component` incluye clases para muchos de los componentes JSF básicos (vea las instrucciones `import` en las líneas 6 a 11 y 13).

```

1  // Fig. 26.6: Hora.java
2  // Archivo de bean de página que establece textoReloj a la hora en el servidor Web.
3  package horaweb;
4
5  import com.sun.rave.web.ui.appbase.AbstractPageBean;
6  import com.sun.rave.web.ui.component.Body;
7  import com.sun.rave.web.ui.component.Form;
8  import com.sun.rave.web.ui.component.Head;
9  import com.sun.rave.web.ui.component.Html;
10 import com.sun.rave.web.ui.component.Link;
11 import com.sun.rave.web.ui.component.Page;
12 import javax.faces.FacesException;
13 import com.sun.rave.web.ui.component.StaticText;
14 import java.text.DateFormat;
15 import java.util.Date;
16
17 public class Hora extends AbstractPageBean
18 {
19     private int __placeholder;
20
21     // método de inicialización de componentes, generado automáticamente.
22     private void _init() throws Exception
23     {
24         // cuerpo vacío
25     } // fin del método _init
26
27     private Page page1 = new Page();
28
29     public Page getPage1()
30     {

```

Figura 26.6 | Archivo de bean de página que establece `textoReloj` a la hora en el servidor Web. (Parte I de 4).

```

31     return page1;
32 } // fin del método getPage1
33
34 public void setPage1(Page p)
35 {
36     this.page1 = p;
37 } // fin del método setPage1
38
39 private Html html1 = new Html();
40
41 public Html getHtml1()
42 {
43     return html1;
44 } // fin del método getHtml1
45
46 public void setHtml1(Html h)
47 {
48     this.html1 = h;
49 } // fin del método setHtml1
50
51 private Head head1 = new Head();
52
53 public Head getHead1()
54 {
55     return head1;
56 } // fin del método getHead1
57
58 public void setHead1(Head h)
59 {
60     this.head1 = h;
61 } // fin del método setHead1
62
63 private Link link1 = new Link();
64
65 public Link getLink1()
66 {
67     return link1;
68 } // fin del método getLink1
69
70 public void setLink1(Link l)
71 {
72     this.link1 = l;
73 } // fin del método setLink1
74
75 private Body body1 = new Body();
76
77 public Body getBody1()
78 {
79     return body1;
80 } // fin del método getBody1
81
82 public void setBody1(Body b)
83 {
84     this.body1 = b;
85 } // fin del método setBody1
86
87 private Form form1 = new Form();
88
89 public Form getForm1()

```

Figura 26.6 | Archivo de bean de página que establece textoRe1oj a la hora en el servidor Web. (Parte 2 de 4).

```

90     {
91         return form1;
92     } // fin del método getForm1
93
94     public void setForm1(Form f)
95     {
96         this.form1 = f;
97     } // fin del método setForm1
98
99     private StaticText encabezadoHora = new StaticText();
100
101     public StaticText getEncabezadoHora()
102     {
103         return encabezadoHora;
104     } // fin del método getEncabezadoHora
105
106     public void setEncabezadoHora(StaticText st)
107     {
108         this.encabezadoHora = st;
109     } // fin del método setEncabezadoHora
110
111     private StaticText textoReloj = new StaticText();
112
113     public StaticText getTextoReloj()
114     {
115         return textoReloj;
116     } // fin del método getTextoReloj
117
118     public void setTextoReloj(StaticText st)
119     {
120         this.textoReloj = st;
121     } // fin del método setTextoReloj
122
123     // Construye una nueva instancia de bean de página
124     public Hora()
125     {
126         // constructor vacío
127     } // fin del constructor
128
129     // Devuelve una referencia al bean de datos con ámbito
130     protected RequestBean1 getRequestBean1()
131     {
132         return (RequestBean1)getBean("RequestBean1");
133     } // fin del método getRequestBean1
134
135     // Devuelve una referencia al bean de datos con ámbito
136     protected ApplicationBean1 getApplicationBean1()
137     {
138         return (ApplicationBean1)getBean("ApplicationBean1");
139     } // fin del método getApplicationBean1
140
141     // Devuelve una referencia al bean de datos con ámbito
142     protected SessionBean1 getSessionBean1()
143     {
144         return (SessionBean1)getBean("SessionBean1");
145     } // fin del método getSessionBean1
146
147     // inicializa el contenido de la página
148     public void init()

```

Figura 26.6 | Archivo de bean de página que establece textoReloj a la hora en el servidor Web. (Parte 3 de 4).

```

149     {
150         super.init();
151         try
152         {
153             _init();
154         } // fin de try
155         catch ( Exception e )
156         {
157             log( "Error al inicializar Hora", e );
158             throw e instanceof FacesException ? ( FacesException ) e :
159                 new FacesException( e );
160         } // fin de catch
161     } // fin del método init
162
163     // método que se llama cuando ocurre una petición de devolución de envío
164     public void preprocess()
165     {
166         // cuerpo vacío
167     } // fin del método preprocess
168
169     // método al que se llama antes de desplegar la página
170     public void prerender()
171     {
172         textoReloj.setValue( DateFormat.getTimeInstance(
173             DateFormat.LONG ).format( new Date() ) );
174     } // fin del método prerender
175
176     // método al que se llama una vez que se completa el despliegue, si se llamó a init
177     public void destroy()
178     {
179         // cuerpo vacío
180     } // fin del método destroy
181 } // fin de la clase Hora

```

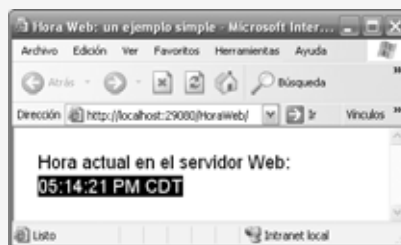


Figura 26.6 | Archivo de bean de página que establece `textoReloj` a la hora en el servidor Web. (Parte 4 de 4).

Este archivo de bean de página proporciona métodos *obtener* (*get*) y *establecer* (*set*) para cada elemento del archivo JSP de la figura 26.4. El IDE genera estos métodos de manera automática. Incluimos el archivo de bean de página completo en este primer ejemplo, pero en los siguientes ejemplos omitiremos estas propiedades y sus métodos *obtener* y *establecer* para ahorrar espacio. En las líneas 99 a 109 y 111 a 121 del archivo de bean de página se definen los dos componentes **Static Text** que soltamos en la página, junto con sus métodos *obtener* y *establecer*. Estos componentes son objetos de la clase `StaticText` en el paquete `com.sun.rave.web.ui.component`.

La única lógica requerida en esta página es establecer el texto del componente `textoReloj` para que lea la hora actual en el servidor. Esto lo hacemos en el método `prerender` (líneas 170 a 174). Más adelante hablaremos sobre el significado de éste y otros métodos de bean de página. En las líneas 172 y 173 se obtiene y da formato a la hora en el servidor, y se establece el valor de `textoReloj` con esa hora.

26.5.3 Ciclo de vida del procesamiento de eventos

El modelo de aplicación de Java Studio Creator 2 coloca varios métodos en el bean de página, los cuales se enlazan en el **ciclo de vida del procesamiento de eventos**. Estos métodos representan cuatro etapas principales: inicialización, pre-procesamiento, pre-despliegue y destrucción. Cada uno de ellos corresponde a un método en la clase de bean de página: `init`, `preprocess`, `prerender` y `destroy`, respectivamente. Java Studio Creator 2 crea estos métodos de manera automática, pero podemos personalizarlos para manejar las tareas de procesamiento del ciclo de vida, como desplegar un elemento en una página sólo si un usuario hace clic en un botón.

El **método `init`** (figura 26.6, líneas 148 a 161) es llamado por el contenedor de JSPs la primera vez que se solicita la página, y en las peticiones de devolución de envío. Una **petición de devolución de envío (postback)** ocurre cuando se envían los datos de un formulario, y la página junto con su contenido se envían al servidor para ser procesados. El método `init` invoca la versión de su superclase (línea 150) y después trata de llamar al método `_init` (declarado en las líneas 22 a 25). El método `_init` también se genera en forma automática, y maneja las tareas de inicialización de componentes (si los hay), como establecer las opciones para un grupo de botones de opción.

El **método `preprocess`** (líneas 164 a 167) se llama después de `init`, pero sólo si la página está procesando una petición de devolución de envío. El **método `prerender`** (líneas 170 a 174) se llama justo antes de que el navegador despliegue (muestre) una página. Este método se debe utilizar para establecer las propiedades de los componentes; las propiedades que se establecen antes (como en el método `init`) pueden sobrescribirse antes de que el navegador despliegue la página. Por esta razón, establecemos el valor de `textoReloj` en el método `prerender`.

Por último, el **método `destroy`** (líneas 177 a 180) se llama una vez que la página se ha desplegado, pero sólo si se hizo la llamada al método `init`. Este método maneja tareas tales como liberar los recursos que se utilizan para desplegar la página.

26.5.4 Relación entre la JSP y los archivos de bean de página

El bean de página tiene una propiedad para cada elemento que aparece en el archivo JSP de la figura 26.4, desde el elemento `html` hasta los dos componentes `Texto` estático. Recuerde que los elementos en el archivo JSP se enlazaron explícitamente a estas propiedades mediante el atributo `binding` de cada elemento, usando una instrucción en Lenguaje de expresiones JSP. Como ésta es una clase `JavaBean`, también se incluyen métodos *obtener (get)* y *establecer (set)* para cada una de estas propiedades (líneas 27 a 121). El IDE genera este código automáticamente para cada proyecto de aplicación Web.

26.5.5 Análisis del XHTML generado por una aplicación Web de Java

En la figura 26.7 se muestra el XHTML que se genera cuando un navegador Web cliente solicita la página `Hora.jsp` (figura 26.4). Para ver este XHTML, seleccione **Ver > Código fuente** en Internet Explorer. [Nota: agregamos los comentarios de XHTML en las líneas 3 y 4, y cambiamos el formato del XHTML para que se conforme a nuestras convenciones de codificación].

El documento XHTML en la figura 26.7 es similar en estructura al archivo JSP de la figura 26.4. En las líneas 5 y 6 está la declaración del tipo de documento, la cual lo declara como documento XHTML 1.0 Transicional. Las etiquetas `ui:meta` en las líneas 9 a 13 son equivalentes a los encabezados HTTP, y se utilizan para controlar el comportamiento del navegador Web.

```

1  <?xml version = "1.0"?>
2
3  <!-- Fig. 26.7: Hora.html -->
4  <!--La respuesta XHTML generada cuando el navegador solicita el archivo Hora.jsp. -->
5  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
6    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
7  <html xmlns = "http://www.w3.org/1999/xhtml">
8    <head>
9      <meta content = "no-cache" http-equiv = "Pragma" />

```

Figura 26.7 | Respuesta XHTML generada cuando el navegador solicita el archivo `Hora.jsp`. (Parte I de 2).

```

10 <meta content = "no-cache" http-equiv = "Cache-Control" />
11 <meta content = "no-store" http-equiv = "Cache-Control" />
12 <meta content = "max-age=0" http-equiv = "Cache-Control" />
13 <meta content = "1" http-equiv = "Expires" />
14 <title>Hora Web: un ejemplo simple</title>
15 <script type = "text/javascript"
16     src = "/HoraWeb/theme/com/sun/rave/web/ui/defaulttheme/
17     javascript/formElements.js"></script>
18 <link rel = "stylesheet" type = "text/css" href = "/HoraWeb/theme/
19     com/sun/rave/web/ui/defaulttheme/css/css_master.css" />
20 <link rel = "stylesheet" type = "text/css" href = "/HoraWeb/theme/
21     com/sun/rave/web/ui/defaulttheme/css/css_ie55up.css" />
22 <script type = "text/javascript">
23     var sjwuic_ScrollCookie = new sjwuic_ScrollCookie(
24         '/Hora.jsp', '/HoraWeb/faces/Hora.jsp' );
25 </script>
26 <link id = "link1" rel = "stylesheet" type = "text/css"
27     href = "/HoraWeb/resources/stylesheet.css" />
28 </head>
29 <meta id = "_id0" http-equiv = "refresh" content = "5" />
30 <body id = "body1" style = "-rave-layout: grid">
31 <form id = "form1" class = "form" method = "post"
32     action = "/HoraWeb/faces/Hora.jsp"
33     enctype = "application/x-www-form-urlencoded">
34 <span id = "form1:encabezadoHora" style = "font-size: 18px; left: 24px;
35     top: 24px; position: absolute">Hora actual en el servidor Web:
36 </span>
37 <span id = "form1:textoReloj" style = "background-color: black;
38     color: yellow; font-size: 18px; left: 24px; top: 48px; position:
39     absolute">10:28:47 PM CDT</span>
40 <input id = "form1_hidden" name = "form1_hidden"
41     value = "form1_hidden" type = "hidden" />
42 </form>
43 </body>
44 </html>

```

Figura 26.7 | Respuesta XHTML generada cuando el navegador solicita el archivo Hora.jsp. (Parte 2 de 2).

En las líneas 30 a 43 se define el cuerpo (body) del documento. En la línea 31 empieza el formulario (form), un mecanismo para recolectar información del usuario y enviarla de vuelta al servidor Web. En este programa específico, el usuario no envía datos al servidor Web para procesarlos; sin embargo, el procesamiento de los datos del usuario es una parte imprescindible de muchas aplicaciones Web, la cual se facilita mediante el uso de los formularios. En ejemplos posteriores demostraremos cómo enviar datos al servidor.

Los formularios XHTML pueden contener componentes visuales y no visuales. Los componentes visuales incluyen botones y demás componentes de GUI con los que interactúan los usuarios. Los componentes no visuales, llamados **elementos de formulario hidden**, almacenan datos tales como direcciones de e-mail, que el autor del documento especifica. Una de estas entradas ocultas se define en las líneas 40 y 41. Más adelante en este capítulo hablaremos sobre el significado preciso de esta entrada oculta. El atributo **method** del elemento form (línea 31) especifica el método mediante el cual el navegador Web envía el formulario al servidor. De manera predeterminada, las JSPs utilizan el método post. Los dos **tipos de peticiones HTTP** más comunes (también conocidas como **métodos de petición**) son **get** y **post**. Una petición **get** obtiene (o recupera) la información de un servidor. Dichas peticiones comúnmente recuperan un documento HTML o una imagen. Una petición **post** envía datos a un servidor, como la información de autenticación o los datos de un formulario que recopilan la entrada del usuario. Por lo general, las peticiones **post** se utilizan para enviar un mensaje a un grupo de noticias o a un foro de discusión, pasar la entrada del usuario a un proceso manejador de datos en el servidor, y almacenar o actualizar los datos en un servidor. El atributo **action** de form (línea 32) identifica el recurso que se pedirá cuando se envíe este formulario; en este caso, /HoraWeb/faces/Hora.jsp.

Observe que los dos componentes **Texto estático** (es decir, `encabezadoHora` y `textoReloj`) se representan mediante dos elementos `span` en el documento XHTML (líneas 34 a 36, 37 a 39) como vimos anteriormente. Las opciones de formato que se especificaron como propiedades de `encabezadoHora` y `textoReloj`, como el tamaño de la fuente y el color del texto en los componentes, ahora se especifican en el atributo `style` de cada elemento `span`.

26.5.6 Creación de una aplicación Web en Java Studio Creator 2

Ahora que hemos presentado el archivo JSP, el archivo de bean de página y la página Web de XHTML resultante que se envía al navegador Web, vamos a describir los pasos para crear esta aplicación. Para crear la aplicación `HoraWeb`, realice los siguientes pasos en Java Studio Creator 2:

Paso 1: Creación del proyecto de aplicación Web

Seleccione **Archivo > Nuevo proyecto...** para mostrar el cuadro de diálogo **Nuevo proyecto**. En este cuadro de diálogo, seleccione **Web** en el panel **Categorías**, **Aplicación Web JSF** en el panel **Proyectos** y haga clic en **Siguiente**. Cambie el nombre del proyecto a `HoraWeb` y use la ubicación predeterminada del proyecto y el paquete Java predeterminado. Estas opciones crearán un directorio `HoraWeb` en su directorio `Mis documentos\Creator\Projects` para almacenar los archivos del proyecto. Haga clic en **Terminar** para crear el proyecto de aplicación Web.

Paso 2: Análisis de la ventana del Editor visual del nuevo proyecto

Las siguientes figuras describen características importantes del IDE, empezando con la ventana **Editor visual** (figura 26.8). Java Studio Creator 2 crea una sola página Web llamada `Page1` cuando se crea un nuevo proyecto. Esta página se abre de manera predeterminada en el Editor visual en **modo Diseño**, cuando el proyecto se carga por primera vez. A medida que arrastre y suelte nuevos componentes en la página, el modo **Diseño** le permitirá ver cómo se desplegará su página en el navegador. El archivo JSP para esta página, llamado `Page1.jsp`, se puede ver haciendo clic en el botón **JSP** que se encuentra en la parte superior del Editor visual, o haciendo clic con el botón derecho del ratón en cualquier parte dentro del Editor visual y seleccionando la opción **Editar origen JSP**. Como dijimos antes, cada página Web está soportada por un archivo de bean de página. Java Studio Creator 2 crea un archivo llamado `Page1.java` cuando se crea un nuevo proyecto. Para abrir este archivo, haga clic en el botón **Java** que se encuentra en la parte superior del Editor visual, o haga clic con el botón derecho del ratón en cualquier parte dentro del Editor visual y seleccione la opción **Editar origen Java Page1**.

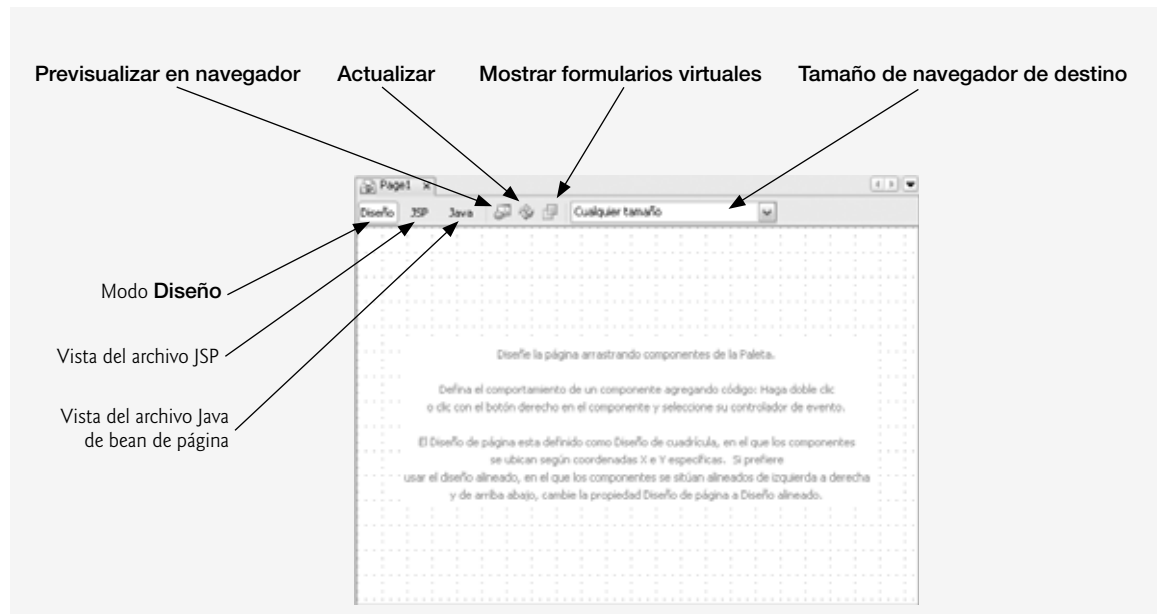


Figura 26.8 | Ventana Editor visual en modo **Diseño**.

El botón **Previsualizar en navegador** en la parte superior de la ventana Editor visual le permite ver sus páginas en un navegador sin tener que crear y ejecutar la aplicación. El botón **Actualizar** vuelve a dibujar la página en el Editor visual. El botón **Mostrar formularios virtuales** nos permite ver qué elementos de formulario están participando en los formularios virtuales (hablaremos sobre este concepto en el capítulo 27). La lista desplegable **Tamaño de navegador de destino** nos permite especificar la resolución óptima del navegador para ver la página, y nos permite ver cuál será la apariencia de la página en distintas resoluciones de pantalla.

Paso 3: Análisis de la Paleta en Java Studio Creator 2

En la figura 26.9 se muestra la **Paleta** que aparece en el IDE cuando se carga el proyecto. La parte a) muestra el inicio de la lista **Básicos** de componentes Web, y la parte b) muestra el resto de los componentes **Básicos**, junto con la lista de componentes **Diseño**. Hablaremos sobre componentes específicos de la figura 26.9 a medida que los utilicemos en el capítulo.

Paso 4: Análisis de la ventana Proyectos

En la figura 26.10 se muestra la ventana **Proyectos**, la cual aparece en la esquina inferior derecha del IDE. Esta ventana muestra la jerarquía de todos los archivos incluidos en el proyecto. Los archivos JSP para cada página se enlistan en el nodo **Páginas Web**. Este nodo también incluye la carpeta **resources**, la cual contiene la hoja de estilo CSS para el proyecto, y cualquier otro archivo que puedan necesitar las páginas para mostrarse en forma apropiada, como los archivos de imagen. Todo el código fuente de Java, incluyendo el archivo de bean de página para cada página Web y los beans de aplicación, sesión y petición, se pueden encontrar bajo el nodo **Paquetes de origen**. Otro archivo útil que se muestra en la ventana del proyecto es el archivo **Navegación de página**, el cual define las reglas para navegar por las páginas del proyecto, con base en el resultado de algún evento iniciado por el usuario, como hacer clic en un botón o en un vínculo. También podemos acceder al archivo **Navegación de página** si hacemos clic con el botón derecho del ratón en el Editor visual, estando en modo **Diseño**; para ello, seleccionamos la opción **Navegación de página**....

Paso 5: Análisis de los archivos JSP y Java en el IDE

En la figura 26.11 se muestra Page1.jsp; el archivo JSP generado por Java Studio Creator 2 para Page1. [Nota: cambiamos el formato del código para adaptarlo a nuestras convenciones de codificación]. Haga clic en el botón **JSP** que está en la parte superior del Editor visual para abrir el archivo JSP. Cuando se crea por primera vez, este

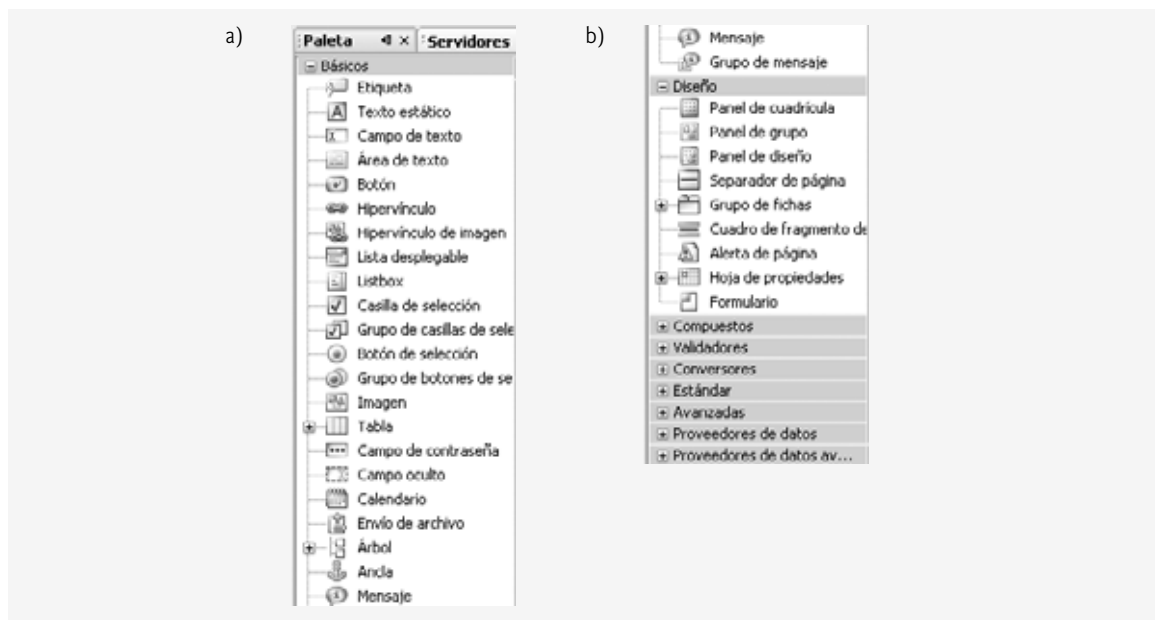


Figura 26.9 | La **Paleta** en Java Studio Creator 2.

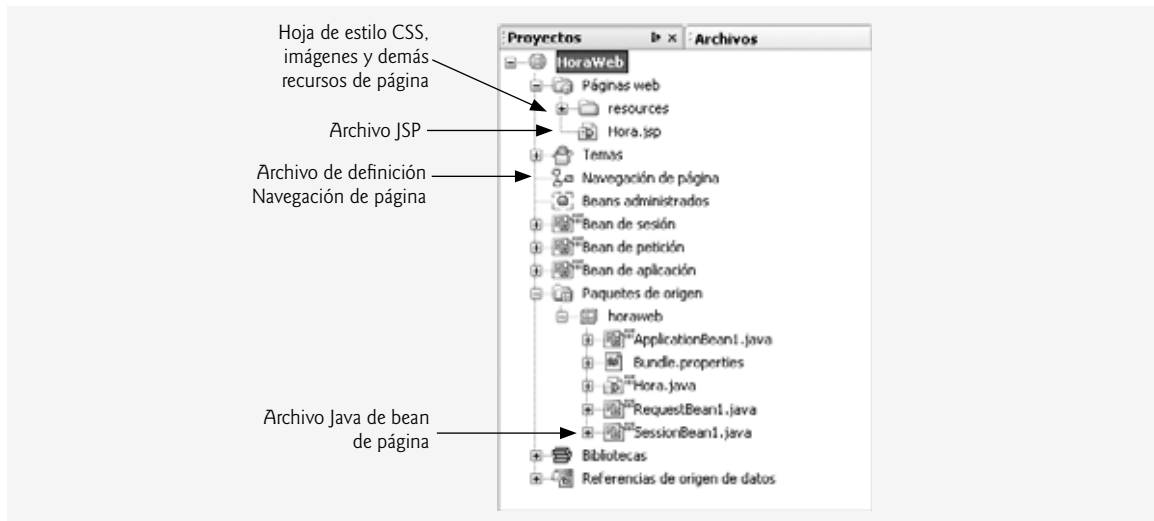


Figura 26.10 | Ventana **Proyectos** para el proyecto HoraWeb.

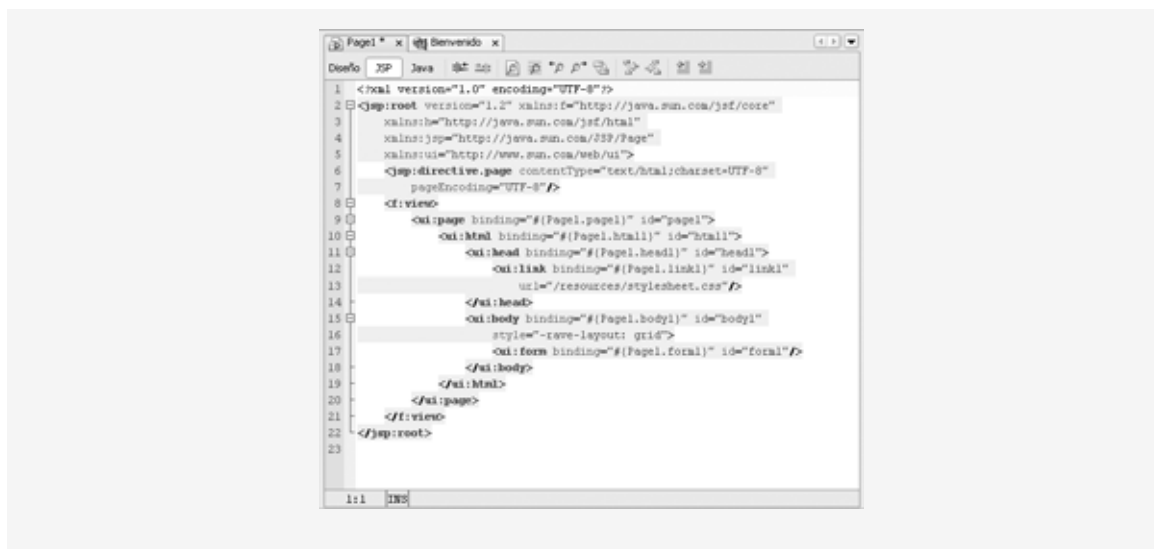


Figura 26.11 | Archivo JSP generado para la página 1 (Page1) por Java Studio Creator 2.

archivo contiene varias etiquetas para configurar la página, incluyendo creación de vínculos a la hoja de estilo de la página y definir las bibliotecas JSF necesarias. En cualquier otro caso, las etiquetas del archivo JSP están vacías, ya que no se han agregado todavía componentes a la página.

En la figura 26.12 se muestra parte de `Page1.java`; el archivo de bean de página generado por Java Studio Creator 2 para Page1. Haga clic en el botón **Java** que está en la parte superior del Editor visual para abrir el archivo de bean de página. Este archivo contiene una clase de Java con el mismo nombre que la página (es decir, Page1), la cual extiende a la clase `AbstractPageBean`. Como dijimos antes, `AbstractPageBean` tiene varios métodos para manejar el ciclo de vida de la página. Cuatro de estos métodos (`init`, `preprocess`, `render` y `destroy`) son sobrescritos por `Page1.java`. Excepto por el método `init`, estos métodos están vacíos al principio. Sirven como receptáculos para que el programador pueda personalizar el comportamiento de su aplicación Web. El archivo de bean de página también incluye métodos *establecer* (*set*) y *obtener* (*get*) para todos los elementos de la página: `page`, `html`, `head` y `link` para empezar. Para ver estos métodos *obtener* y *establecer*, haga clic en el signo más (+) en la línea que dice **Creator-managed Component Definition**.

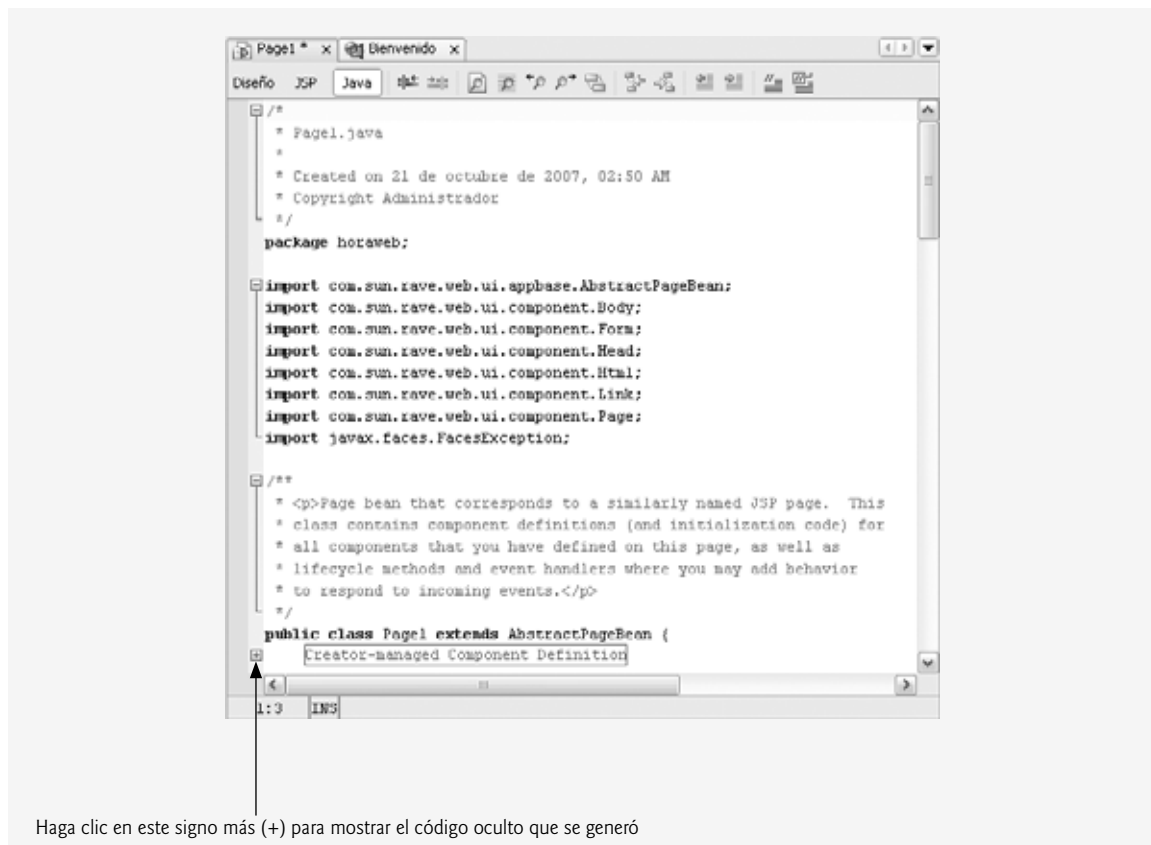


Figura 26.12 | Archivo de bean de página para Page1.jsp, generado por Java Studio Creator 2.

Paso 6: Cambiar el nombre de los archivos JSP y JSF

Por lo general, es conveniente cambiar el nombre de los archivos JSP y Java en un proyecto, de manera que sus nombres sean relevantes para nuestra aplicación. Haga clic con el botón derecho del ratón en el archivo Page1.jsp en la **Ventana Proyectos** y seleccione **Cambiar nombre**, para que aparezca el cuadro de diálogo **Cambiar nombre**. Escriba el nuevo nombre Hora para el archivo. Si está activada la opción **Previsualizar todos los cambios**, aparecerá la **Ventana Refactorización** en la parte inferior del IDE cuando haga clic en **Siguiente >**. La **Refactorización** es el proceso de modificar el código fuente para mejorar su legibilidad y reutilización, sin modificar su comportamiento; por ejemplo, al cambiar los nombres a los métodos o variables, o al dividir métodos extensos en varios métodos más cortos, Java Studio Creator 2 tiene herramientas de refactorización integradas, las cuales automatizan ciertas tareas de refactorización. Al usar estas herramientas para cambiar el nombre a los archivos del proyecto, se actualizan los nombres tanto del archivo JSP como del archivo de bean de página. La herramienta de refactorización también modifica el nombre de la clase en el archivo de bean de página y todos los enlaces de los atributos en el archivo JSP, para reflejar el nuevo nombre de la clase. Observe que no se hará ninguno de estos cambios, sino hasta que haga clic en el botón **Refactorizar** de la **Ventana Refactorización**. Si no previsualiza los cambios, la refactorización ocurre al momento en que haga clic en el botón **Siguiente >** del cuadro de diálogo **Cambiar nombre**.

Paso 7: Cambiar el título de la página

Antes de diseñar el contenido de la página Web, vamos a darle el título "Hora Web: un ejemplo simple". De manera predeterminada, la página no tiene un título cuando el IDE la genera. Para agregar un título, abra el archivo JSP en modo **Diseño**. En la ventana **Propiedades**, escriba el nuevo título enseguida de la propiedad **Title** y oprima **Intro**. Vea la JSP para cerciorarse que el atributo title = "Hora Web: un ejemplo simple" se haya agregado automáticamente a la etiqueta ui:head.

Paso 8: Diseñar la página

Diseñar una página Web es más sencillo en Java Studio Creator 2. Para agregar componentes a la página, puede arrastarlos y soltarlos desde la **Paleta** hacia la página en modo **Diseño**. Al igual que la misma página Web, cada componente es un objeto que tiene propiedades, métodos y eventos. Puede establecer estas propiedades y eventos en forma visual, mediante la ventana **Propiedades**, o mediante programación en el archivo de bean de página. Los métodos *obtener (get)* y *establecer (set)* se agregan automáticamente al archivo de bean de página para cada componente que se agregue a la página.

El IDE genera las etiquetas JSP para los componentes que el programador arrastra y suelta mediante el uso de un diseño de cuadrícula, como se especifica en la etiqueta `ui:body`. Esto significa que los componentes se desplegarán en el navegador usando **posicionamiento absoluto**, de manera que aparezcan exactamente en donde se sueltan en la página. A medida que el programador agregue componentes a la página, el atributo `style` en el elemento JSP de cada componente incluirá el número de píxeles desde los márgenes superior e izquierdo de la página en la que se posicione el componente.

En este ejemplo, usamos dos componentes **Texto estático**. Para agregar el primero a la página Web, arrástrelo y suéltelo desde la lista de componentes **Básicos** de la **Paleta**, hasta la página en modo **Diseño**. Edite el texto del componente, escribiendo "Hora actual en el servidor Web:" directamente en el componente. También puede editar el texto modificando la propiedad `text` del componente en la ventana **Propiedades**. Java Studio Creator 2 es un editor WYSIWYG (**What You See Is What You Get** —Lo que ve es lo que obtiene); cada vez que realice un cambio a una página Web en modo **Diseño**, el IDE creará el marcado (visible en modo **JSP**) necesario para lograr los efectos visuales deseados que aparecen en modo **Diseño**. Después de agregar el texto a la página Web, cambie al modo **JSP**. Ahí podrá ver que el IDE agregó un elemento `ui:staticText` al cuerpo de la página, el cual está enlazado al objeto `staticText1` en el archivo de bean de página, y cuyo atributo `text` coincide con el texto que acaba de escribir. De vuelta al modo **Diseño**, haga clic en el componente **Texto estático** para seleccionarlo. En la ventana **Propiedades**, haga clic en el botón de elipsis enseguida de la propiedad `style` para abrir un cuadro de diálogo y editar el estilo del texto. Seleccione 18 px para el tamaño de la fuente y haga clic en **Aceptar**. De nuevo en la ventana **Propiedades**, cambie la propiedad `id` a `encabezadoHora`. Al establecer la propiedad `id` también se modifica el nombre de la propiedad correspondiente del componente en el bean de página, y se actualiza su atributo `binding` en la JSP de manera acorde. Observe que se ha agregado `font-size: 18 px` al atributo `style` y que el atributo `id` ha cambiado a `encabezadoHora` en la etiqueta del componente en el archivo JSP. El IDE debe aparecer ahora como se muestra en la figura 26.13.

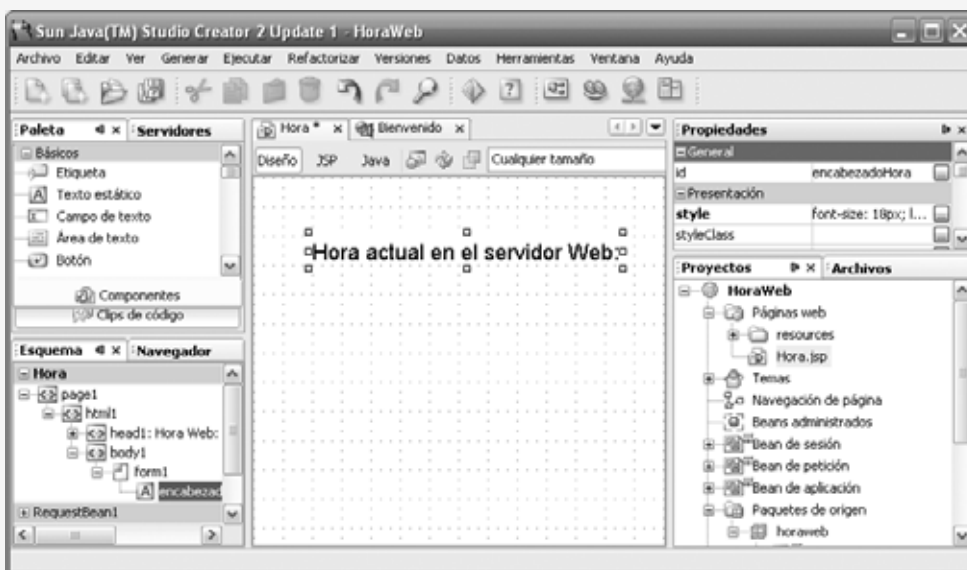


Figura 26.13 | Hora.jsp después de insertar el primer componente **Texto estático**.

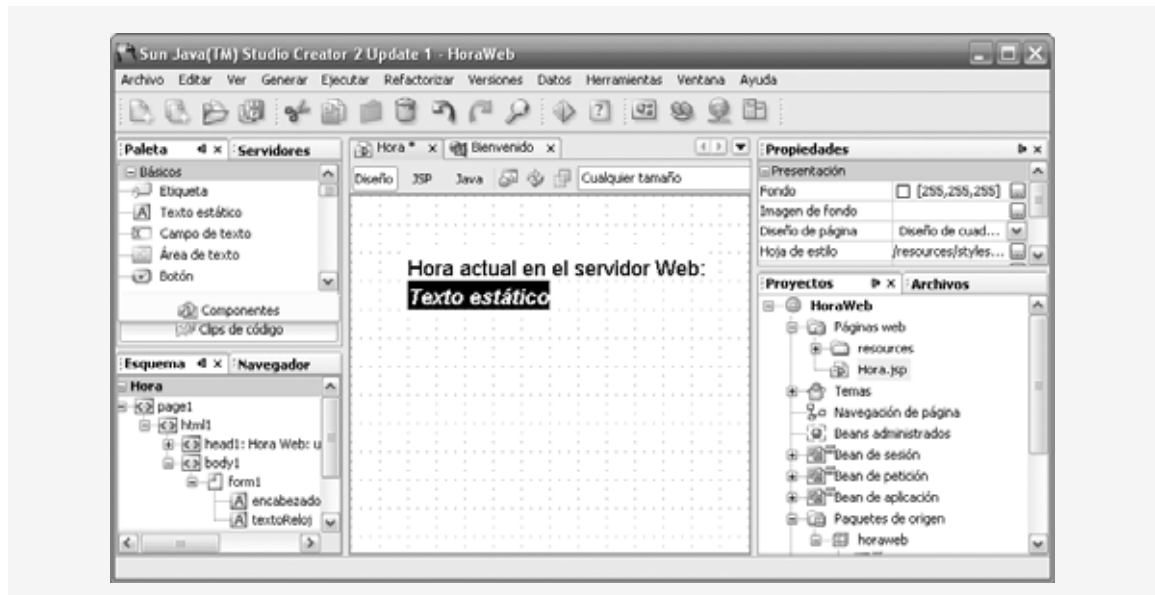


Figura 26.14 | Hora.jsp después de agregar el segundo componente Texto estático.

Coloque un segundo componente **Texto estático** en la página, y establezca su id en `textoReLoj`. Edite su propiedad `style` de manera que el tamaño de la fuente sea 18 px, el color de texto sea amarillo (`yellow`) y el color de fondo sea negro (`black`). No edite el texto del componente, ya que éste se establecerá mediante programación en el archivo de bean de página. El componente se mostrará con el texto **Texto estático** en el IDE, pero no mostrará ningún texto en tiempo de ejecución, a menos que éste se establezca mediante programación. La figura 26.14 muestra el IDE después de agregar el segundo componente.

Paso 9: Agregar la lógica de la página

Después de diseñar la interfaz de usuario, puede modificar el archivo de bean de página para establecer el texto del elemento `textoReLoj`. En este ejemplo, agregamos una instrucción al método `preRender` (líneas 170 a 174 de la figura 26.6). Recuerde que utilizamos el método `preRender` para asegurar que `textoReLoj` se actualice cada vez que se actualice la página. En las líneas 172 y 173 de la figura 26.6 se establece mediante programación el texto de `textoReLoj` con la hora actual en el servidor.

Nos gustaría que esta página se actualizara automáticamente para mostrar una hora actualizada. Para lograr esto, agregue la etiqueta vacía `<ui:meta content = "60" httpEquiv = "refresh" />` al archivo JSP, entre el final de la etiqueta `ui:head` y el inicio de la etiqueta `ui:body`. Esta etiqueta indica al navegador que debe volver a cargar la página automáticamente cada 60 segundos. También puede agregar esta etiqueta arrastrando un componente **Meta** de la sección **Avanzados** de la **Paleta** a su página, y después estableciendo el atributo `content` del componente a 60 y su atributo `httpEquiv` a `refresh`.

Paso 10: Análisis de la ventana Esquema

En la figura 26.15 se muestra la **ventana Esquema** en Java Studio Creator 2. Los cuatro archivos Java del proyecto se muestran como nodos de color gris. El nodo **Hora** que representa el archivo de bean de página está expandido y muestra el contenido del árbol de componentes. Los beans de ámbito de petición, sesión y aplicación están contraídos de manera predeterminada, ya que no hemos agregado propiedades a estos beans en este ejemplo. Al hacer clic en el árbol de componentes de la página, se selecciona el elemento en el Editor visual.

Paso 11: Ejecutar la aplicación

Después de crear la página Web, podemos verla de varias formas. Primero seleccione **Generar > Generar proyecto principal**, y después que se complete la generación, seleccione **Ejecutar > Ejecutar proyecto principal** para ejecu-

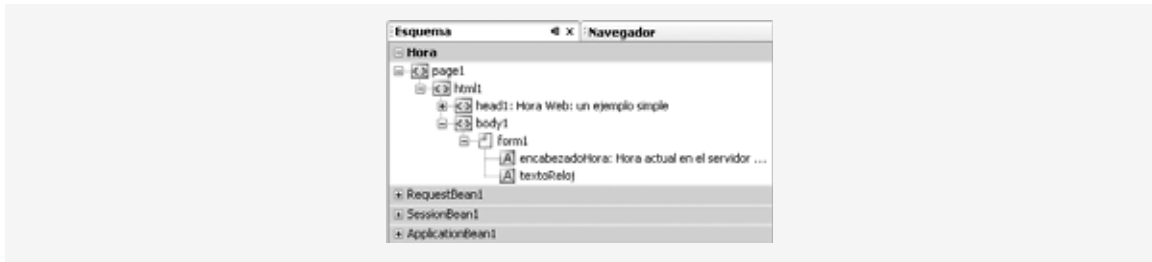


Figura 26.15 | Ventana **Esquema** en Java Studio Creator 2.

tar la aplicación en una ventana del navegador. Para ejecutar un proyecto que ya haya sido generado, oprima el icono **Ejecutar proyecto principal** (▶) en la barra de herramientas que se encuentra en la parte superior del IDE. Observe que, si se hacen cambios a un proyecto, éste debe volver a generarse para que puedan reflejarse cuando se vea la aplicación en un navegador Web. Como esta aplicación se generó en el sistema de archivos local, el URL que se muestre en la barra de dirección del navegador cuando se ejecute la aplicación será `http://localhost:29080/HoraWeb/` (figura 26.6), en donde 29080 es el número de puerto en el que se ejecuta el servidor de prueba integrado de Java Studio Creator 2 (**Sun Application Server 8**) de manera predeterminada. Al ejecutar un programa en el servidor de prueba, aparece un icono cerca de la parte inferior derecha de la pantalla, para demostrar que Sun Application Server se está ejecutando. Para cerrar el servidor después de salir de Java Studio Creator 2, haga clic con el botón derecho en el icono de la bandeja y seleccione **Stop Domain creator**.

De manera alternativa, puede oprimir *F5* para generar la aplicación y después ejecutarla en modo de depuración; el depurador integrado de Java Studio Creator 2 puede ayudarle a diagnosticar fallas en las aplicaciones. Si escribe *<Ctrl> F5*, el programa se ejecuta sin habilitar la depuración.



Tip para prevenir errores 26.1

*Si tiene problemas al generar su proyecto debido a errores en los archivos XML generados por Java Studio Creator, que se utilizan para la generación, pruebe a limpiar el proyecto y volver a generar. Para ello, seleccione **Generar > Limpiar y generar proyecto principal**, u oprima *<Alt> B*.*

Por último, para ejecutar su aplicación generada, abra una ventana del navegador y escriba el URL de la página Web en el campo **Dirección**. Como su aplicación reside en el sistema de archivos local, primero debe iniciar Sun Application Server. Si ejecutó antes la aplicación utilizando uno de los métodos anteriores, el servidor ya se estará ejecutando. De no ser así, puede iniciar el servidor desde el IDE; para ello abra la ficha **Servidores** (que se encuentra en el mismo panel que la **Paleta**), haga clic con el botón derecho del ratón en el **Servidor de ejecución**, seleccione **Iniciar/Detener servidor** y haga clic en el botón **Iniciar**, en el cuadro de diálogo que aparezca. Después, puede escribir el URL (incluyendo el número de puerto para el servidor de aplicación, 29080) en el navegador para ejecutar la aplicación. Para este ejemplo no es necesario escribir el URL completo, `http://localhost:29080/HoraWeb/faces/Hora.jsp`. La ruta para el archivo `Hora.jsp` (es decir, `faces/Hora.jsp`) se puede omitir, ya que este archivo se estableció de manera predeterminada como la página inicial del proyecto. Para los proyectos con varias páginas, puede modificar la página inicial haciendo clic en la página deseada en la ventana **Proyectos**, y seleccionando **Definir como página de inicio**. La página de inicio se indica mediante una flecha verde enseguida del nombre de la página en la ventana **Proyectos**. [Nota: si utiliza Netbeans Visual Web Pack 5.5, el número de puerto dependerá del servidor en el que despliegue su aplicación Web. Además, la ficha **Servidores** se llama **Tiempo de ejecución (Runtime)** en Netbeans].

26.6 Componentes JSF

En esta sección presentaremos algunos de los componentes JSF que se incluyen en la **Paleta** (figura 26.9). En la figura 26.16 se sintetizan algunos de los componentes JSF que se utilizan en los ejemplos del capítulo.

26.6.1 Componentes de texto y gráficos

En la figura 26.17 se muestra un formulario simple para recopilar la entrada del usuario. Este ejemplo utiliza todos los componentes enlistados en la figura 26.16, con la excepción de **Etiqueta**, que veremos en ejemplos

Componentes JSF	Descripción
Etiqueta	Muestra texto que se puede asociar con un elemento de entrada.
Texto estático	Muestra texto que el usuario no puede editar.
Campo de texto	Recopila la entrada del usuario y muestra texto.
Botón	Desencadena un evento cuando se oprime.
Hipervínculo	Muestra un hipervínculo.
Lista desplegable	Muestra una lista desplegable de opciones.
Grupo de botones de selección	Muestra botones de opción.
Imagen	Muestra imágenes (como GIF y JPG).

Figura 26.16 | Componentes JSF de uso común.

posteriores. Todo el código en la figura 26.17 se generó mediante Java Studio Creator 2, en respuesta a las acciones realizadas en modo **Diseño**. Este ejemplo no realiza ninguna tarea cuando el usuario hace clic en **Registrar**. Le pediremos que agregue funcionalidad a este ejemplo como un ejercicio. En los siguientes ejemplos, demostraremos cómo agregar funcionalidad a muchos de estos componentes JSF.

Antes de hablar sobre los componentes JSF que se utilizan en este archivo JSP, explicaremos el XHTML que crea el esquema de la figura 26.17. Como dijimos antes, Java Studio Creator 2 utiliza el posicionamiento absoluto, por lo que los componentes se despliegan en donde se hayan soltado en el Editor visual. En este ejemplo, además del posicionamiento absoluto utilizamos un componente **Panel de cuadrícula** (líneas 31 a 52) del grupo de componentes **Diseño** de la **Paleta**. El prefijo **h:** indica que se encuentra en la biblioteca de etiquetas HTML de JSF. Este componente, un objeto de la clase `HtmlPanelGrid` en el paquete `javax.faces.component.html`, controla el posicionamiento de los componentes que contiene. El componente **Panel de cuadrícula** permite al diseñador especificar el número de columnas que debe contener la cuadrícula. Después se pueden soltar los componentes en cualquier parte dentro del panel, y éstos se reposicionarán automáticamente en columnas espaciadas de manera uniforme, en el orden en el que se suelten. Cuando el número de componentes excede al número de columnas, el panel desplaza los componentes adicionales hacia una nueva fila. De esta forma, el **Panel de cuadrícula** se comporta como una tabla de XHTML, y de hecho se despliega en el navegador como una tabla XHTML. En este ejemplo, usamos el **Panel de cuadrícula** para controlar las posiciones de los componentes **Imagen** y **Campo de texto** en la sección de la página acerca de la información del usuario.

```

1  <?xml version = "1.0" encoding = "UTF-8"?>
2
3  <!-- Fig. 26.17: ComponentesWeb.jsp -->
4  <!-- Formulario de registro que demuestra el uso de los componentes JSF. -->
5  <jsp:root version = "1.2" xmlns:f = "http://java.sun.com/jsf/core"
6      xmlns:h = "http://java.sun.com/jsf/html" xmlns:jsp =
7      "http://java.sun.com/JSP/Page" xmlns:ui = "http://www.sun.com/web/ui">
8      <jsp:directive.page contentType = "text/html; charset = UTF-8"
9          pageEncoding = "UTF-8"/>
10     <f:view>
11         <ui:page binding = "#{ComponentesWeb.page1}" id = "page1">
12             <ui:html binding = "#{ComponentesWeb.html1}" id = "html1">
13                 <ui:head binding = "#{ComponentesWeb.head1}" id = "head1">
14                     <ui:link binding = "#{ComponentesWeb.link1}" id = "link1"
15                         url = "/resources/stylesheet.css"/>
16                 </ui:head>

```

Figura 26.17 | Formulario de registro que demuestra el uso de los componentes JSF. (Parte I de 3).


```

17 <ui:body binding = "#{ComponentesWeb.body1}" id = "body1"
18 style = "-rave-layout: grid">
19 <ui:form binding = "#{ComponentesWeb.form1}" id = "form1">
20 <ui:staticText binding = "#{ComponentesWeb.encabezado}"
21 id = "encabezado" style = "font-size: 18px; left: 48px;
22 top: 24px; position: absolute" text = "Este es un
23 formulario de registro de ejemplo."/>
24 <ui:staticText binding = "#{ComponentesWeb.instrucciones}"
25 id = "instrucciones" style = "font-style: italic;
26 left: 48px; top: 60px; position: absolute" text =
27 "Por favor complete todos los campos y haga clic en Registro."/>
28 <ui:image binding = "#{ComponentesWeb.imagenUsuario}" id =
29 "imagenUsuario" style = "left: 48px; top: 96px;
30 position: absolute" url = "/resources/usuario.JPG"/>
31 <h:panelGrid binding = "#{ComponentesWeb.gridPanel1}"
32 columns = "4" id = "gridPanel1" style = "height: 120px;
33 left: 48px; top: 120px; position: absolute"
34 width = "576">
35 <ui:image binding = "#{ComponentesWeb.imagenNombre}"
36 id = "imagenNombre" url = "/resources/nombre.JPG"/>
37 <ui:textField binding = "#{ComponentesWeb.cuadroTextoNombre}"
38 id = "cuadroTextoNombre"/>
39 <ui:image binding = "#{ComponentesWeb.imagenApellido}"
40 id = "imagenApellido" url = "resources/apellido.JPG"/>
41 <ui:textField binding = "#{ComponentesWeb.cuadroTextoApellido}"
42 id = "cuadroTextoApellido"/>
43 <ui:image binding = "#{ComponentesWeb.imagenEmail}"
44 id = "imagenEmail" url = "/resources/email.JPG"/>
45 <ui:textField binding = "#{ComponentesWeb.cuadroTextoEmail}"
46 id = "cuadroTextoEmail"/>
47 <ui:image binding = "#{ComponentesWeb.imagenTelefono}"
48 id = "imagenTelefono" url = "/resources/telefono.JPG"/>
49 <ui:textField binding = "#{ComponentesWeb.cuadroTextoTelefono}"
50 id = "cuadroTextoTelefono" label = "Debe tener la forma (555)
51 555-5555"/>
52 </h:panelGrid>
53 <ui:image binding = "#{ComponentesWeb.imagenPublicaciones}"
54 id = "imagenPublicaciones" style = "left: 48px; top: 264px;
55 position: absolute" url =
56 "/resources/publicaciones.JPG"/>
57 <ui:staticText binding =
58 "#{ComponentesWeb.etiquetaPublicacion}" id =
59 "etiquetaPublicacion" style = "position: absolute;
60 left: 300px; top: 264px" text = "De que libro desea obtener
61 informacion?"/>
62 <ui:dropDown binding = "#{ComponentesWeb.librosDesplegable}"
63 id = "librosDesplegable" items = "#{ComponentesWeb.
64 librosDesplegableDefaultOptions.options}" style = "left:
65 48px; top: 300px; position: absolute" width: 240px />
66 <ui:hyperlink binding = "#{ComponentesWeb.librosVinculo}"
67 id = "librosVinculo" style = "left: 48px; top: 348px;
68 position: absolute" target = "_blank" text = "Haga clic
69 aqui para ver mas informacion acerca de nuestros libros."
70 url = "http://www.deitel.com"/>
71 <ui:image binding = "#{ComponentesWeb.image1}" id =
72 "image1" style = "left: 48px; top: 396px;
73 position: absolute" url = "/resources/so.JPG"/>
74 <ui:staticText binding = "#{ComponentesWeb.etiquetaS0}" id =

```

Figura 26.17 | Formulario de registro que demuestra el uso de los componentes JSF. (Parte 2 de 3).

```

75     "etiquetaS0" style = "position: absolute; left: 252px;
76     top: 396px" text = "Que sistema operativo
77     utiliza?"/>
78     <ui:radioButtonGroup binding=
79     "#{ComponentesWeb.botonSeleccionS0}" id =
80     "botonesSeleccionS0" items = "#{ComponentesWeb.
81     botonesSeleccionS0DefaultOptions.options}" style =
82     "left: 48px; top: 432px; position: absolute"/>
83     <ui:button binding = "#{ComponentesWeb.botonRegistro}"
84     id = "botonRegistro" style = "left: 47px; top: 564px;
85     position: absolute" text = "Registro"/>
86     </ui:form>
87     </ui:body>
88     </ui:html>
89     </ui:page>
90     </f:view>
91     </jsp:root>

```

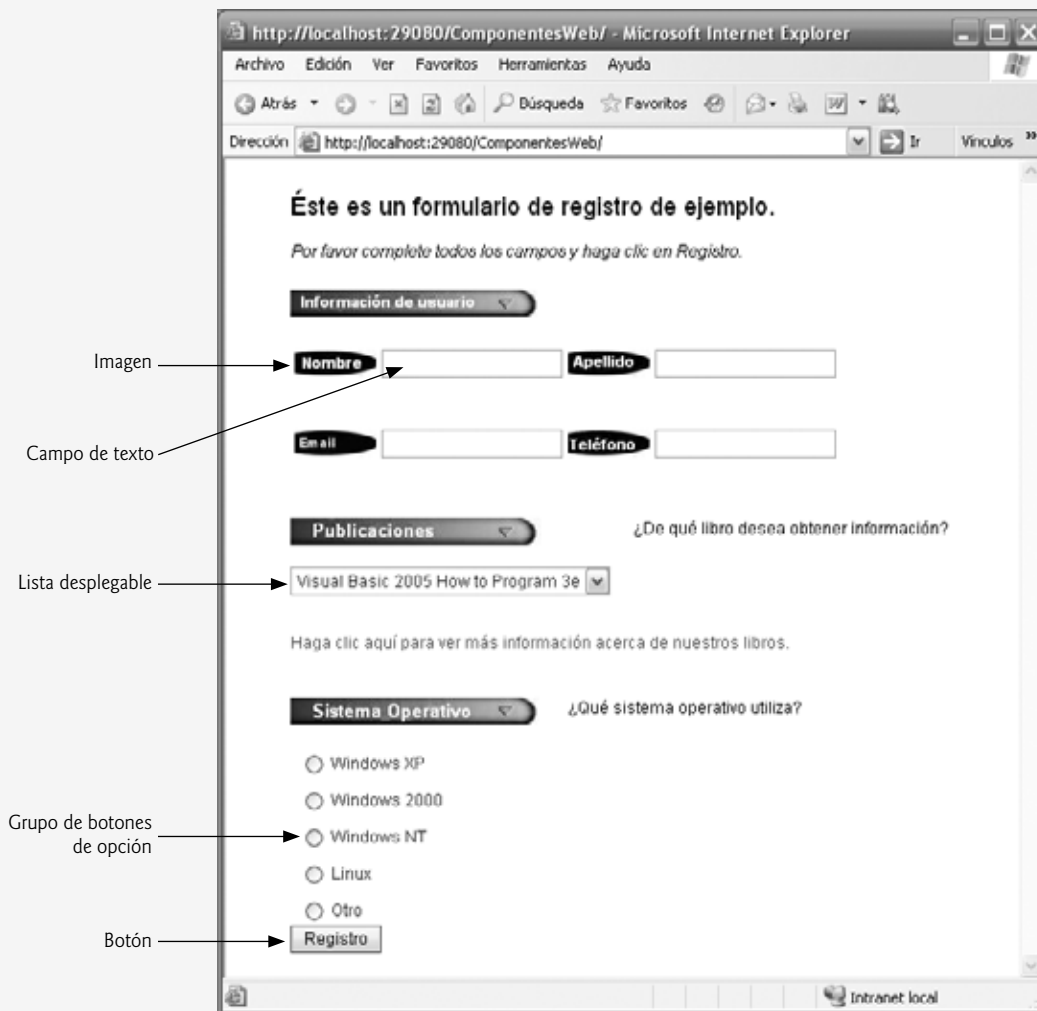


Figura 26.17 | Formulario de registro que demuestra el uso de los componentes JSF. (Parte 3 de 3).

Cómo agregar un componente de formato a una página Web

Para crear el esquema para la sección **Información del usuario** del formulario que se muestra en la figura 26.17, arrastre un componente **Panel de cuadrícula** en la página. En la ventana **Propiedades**, establezca la propiedad `columns` del componente en 4. El componente también tiene propiedades para controlar el relleno de las celdas, el espaciado y otros elementos relacionados con la apariencia del componente. En este caso, acepte los valores predeterminados para estas propiedades. Ahora, simplemente puede arrastrar los componentes **Imagen** y **Campo de texto** para la información del usuario en el **Panel de cuadrícula**. Este componente administrará su espaciado y su organización en filas y columnas.

Cómo analizar los componentes Web en un formulario de registro de ejemplo

En las líneas 28 a 30 de la figura 26.17 se define un componente **Imagen**, un objeto de la clase `Image` que inserta una imagen en una página Web. Las imágenes que se utilizan en este ejemplo se encuentran en el directorio de ejemplos de este capítulo. Las imágenes que se van a mostrar en una página Web se deben colocar en la carpeta `resources` del proyecto. Para agregar imágenes al proyecto, suelte un componente **Imagen** en la página y haga clic en el botón de elipse que está a un lado de la propiedad `url` en la ventana **Propiedades**. A continuación se abrirá un cuadro de diálogo, en el que puede seleccionar la imagen a mostrar. Como aún no se han agregado imágenes a la carpeta `resources`, haga clic en el botón **Agregar archivo**, localice la imagen en el sistema de archivos de su computadora y haga clic en **Agregar archivo**. A continuación se copiará el archivo que usted seleccionó en el directorio `resources` del proyecto. Ahora puede seleccionar la imagen de la lista de archivos en la carpeta `resources` y hacer clic en **Aceptar** para insertar la imagen en la página.

Las líneas 31 a 52 contienen un elemento `h:panelGrid`, el cual representa al componente **Panel de cuadrícula**. Dentro de este elemento hay ocho componentes **Imagen** y **Campo de texto**. Los componentes **Campo de texto** nos permiten obtener la entrada de texto del usuario. Por ejemplo, en las líneas 37 y 38 se define un control **Campo de texto** que se utiliza para recolectar el nombre de pila del usuario. En las líneas 49 a 51 se define un **Campo de texto** con la propiedad `label` establecida en "Debe tener la forma (555) 555-5555". Al establecer la propiedad `label` de un **Campo de texto**, se coloca el texto directamente encima de este componente. De manera alternativa, para etiquetar un **Campo de texto** puede arrastrar y soltar un componente **Etiqueta** en la página, lo cual le permitirá personalizar la posición y el estilo del componente **Etiqueta**.

El orden en el que se arrastran los componentes **Campo de texto** a la página es importante, ya que sus etiquetas JSP se agregan al archivo JSP en ese orden. Cuando un usuario oprime la tecla *Tab* para navegar de un campo de entrada a otro, navegarán por los campos en el orden en el que se hayan agregado las etiquetas JSP al archivo JSP. Para especificar el orden de navegación, debe arrastrar los componentes a la página en ese orden. De manera alternativa, puede establecer la propiedad `Tab Index` de cada campo de texto en la ventana **Propiedades**, para controlar el orden en el que el usuario avanzará mediante la tecla *Tab* por los campos de texto. Un componente con un índice de tabulación de 1 será el primero en la secuencia de tabulaciones.

En las líneas 62 a 65 se define una **Lista desplegable**. Cuando un usuario hace clic en la lista desplegable, expande y muestra una lista, en la que el usuario puede seleccionar un elemento. Este componente es un objeto de la clase `DropDownList` y está enlazado al objeto `librosDesplegable`, un objeto `SingleSelectOptionsList` que controla la lista de opciones. Este objeto se puede configurar de manera automática, haciendo clic con el botón derecho en la lista desplegable en modo **Diseño** y seleccionando **Configurar opciones predeterminadas**, con lo cual se abrirá el cuadro de diálogo **Personalizador de opciones** para agregar opciones a la lista. Cada opción consiste en un objeto `String` para mostrar, el cual representará la opción en el navegador, y de un objeto `String` de valor, el cual se devolverá cuando se obtenga la selección del usuario de la lista desplegable, por medio de programación. Java Studio Creator 2 construye el objeto `SingleSelectOptionsList` en el archivo de bean de página, con base en los pares mostrar-valor introducidos en el cuadro de diálogo **Personalizador de opciones**. Para ver el código que construye al objeto, cierre el cuadro de diálogo haciendo clic en **Aceptar**, abra el archivo de bean de página y expanda el nodo **Creator-managed Component Definition** cerca de la parte superior del archivo. El objeto se construye en el método `_init`, el cual se llama desde el método `init` la primera vez que se carga la página.

El componente **Hipervínculo** (líneas 66 a 70) de la clase `HyperLink` agrega un vínculo a una página Web. La propiedad `url` de este componente especifica el recurso (<http://www.deitel.com> en este caso) que se solicita cuando un usuario hace clic en el hipervínculo. Al establecer la propiedad `target` en `_blank`, especificamos que la página Web solicitada debe abrirse en una nueva ventana del navegador. De manera predeterminada, los componentes **Hipervínculo** hacen que las páginas se abran en la misma ventana del navegador.

En las líneas 78 a 82 se define un componente **Grupo de botones de selección** de la clase `RadioButtonGroup`, el cual proporciona una serie de botones de opción, de los cuales el usuario sólo puede seleccionar uno. Al igual que **Lista desplegable**, un **Grupo de botones de selección** está enlazado a un objeto `SingleSelectOptionsList`. Para editar las opciones, haga clic con el botón derecho del ratón en el componente y seleccione **Configurar opciones predeterminadas**. Al igual que la lista desplegable, el IDE genera el constructor del objeto `SingleSelectOptionsList` automáticamente y lo coloca en el método `_init` de la clase de bean de página.

El último control Web en la figura 26.17 es un **Botón** (líneas 83 a 85), un componente JSF de la clase `Button` que desencadena una acción cuando es oprimido. Por lo general, un componente **Botón** se asigna a un elemento `input` de XHTML, en donde su atributo `type` se establece en `submit`. Como dijimos antes, al hacer clic en el botón **Registro** en este ejemplo no se produce ninguna acción.

26.6.2 Validación mediante los componentes de validación y los validadores personalizados

En esta sección presentamos la **validación** de formularios. La validación de la entrada del usuario es un importante paso para recolectar la información de los usuarios. La validación ayuda a evitar los errores de procesamiento debido a que los datos de entrada del usuario estén incompletos, o tengan un formato inapropiado. Por ejemplo, puede realizar la validación para asegurar que se hayan completado todos los campos requeridos, o que un campo de código postal contenga exactamente cinco dígitos. Java Studio Creator 2 proporciona tres componentes de validación. Un **Validador de longitud** determina si un campo contiene un número aceptable de caracteres. El **Validador de intervalo doble** y el **Validador de intervalo largo** determinan si la entrada numérica se encuentra dentro de intervalos aceptables. El paquete `javax.faces.validators` contiene las clases para estos validadores. Studio Creator 2 también permite la validación personalizada con métodos de validación en el archivo de bean de página. El siguiente ejemplo demuestra la validación mediante el uso de un componente de validación y de la validación personalizada.

Cómo validar los datos de un formulario en una aplicación Web

El ejemplo en esta sección pide al usuario que introduzca su nombre, dirección de e-mail y número telefónico. Después de que el usuario introduce los datos, pero antes de que éstos se envíen al servidor Web, la validación nos asegura que el usuario haya introducido un valor en cada campo, que el nombre introducido no exceda a 30 caracteres y que la dirección de e-mail y el número telefónico se encuentren en un formato aceptable. En este ejemplo, (555) 123-4567, 555-123-4567 y 123-4567 se consideran números telefónicos válidos. Una vez que se envían los datos, el servidor Web responde mostrando un mensaje apropiado y un componente **Panel de cuadrícula** que repite la información enviada. Observe que una aplicación comercial real, por lo general, almacena los datos enviados en una base de datos o en el servidor. Nosotros simplemente enviamos de vuelta los datos a la página, para demostrar que el servidor recibió los datos.

Creación de la página Web

Esta aplicación web introduce dos componentes JSF adicionales: **Etiqueta** y **Mensaje**, de la sección de componentes **Básicos** de la **Paleta**. Cada uno de los tres campos de texto debe tener su propia etiqueta y su propio mensaje. Los componentes **Etiqueta** describen a otros componentes, y se pueden asociar con los campos de entrada del usuario si se establece su propiedad `for`. Los componentes **Mensaje** muestran mensajes de error cuando falla la validación. Esta página requiere tres componentes **Campo de texto**, tres componentes **Etiqueta** y tres componentes **Mensaje**, así como un componente **Botón** para enviar los datos. Para asociar los componentes **Etiqueta** y **Mensaje** con sus correspondientes componentes **Campo de texto**, mantenga oprimidas las teclas *Ctrl* y *Mayús*, y después arrastre la etiqueta o mensaje hacia el **Campo de texto** apropiado. En la ventana **Propiedades**, observe que la propiedad `for` de cada **Etiqueta** y **Mensaje** se encuentra establecida con el componente **Campo de texto** apropiado.

También es conveniente agregar un componente **Texto estático** para mostrar un mensaje de éxito en la validación al final de la página. Establezca el texto en "Gracias por enviar sus datos.
 Recibimos la siguiente información:" y cambie el id del componente a `textoResultado`. En la ventana **Propiedades**, desactive las propiedades `rendered` y `escape` del componente. La propiedad `rendered` controla si el componente se mostrará la primera vez que se cargue la página. Al establecer `escaped` en `false`, el navegador podrá reconocer la etiqueta `
`, de manera que pueda empezar una nueva línea de texto, en vez de mostrar los caracteres "
" en la página web.

Por último, agregue un componente **Panel de cuadrícula** debajo del componente `textoResultado`. El panel debe tener dos columnas, una para mostrar componentes **Texto estático** que etiqueten los datos validados del usuario, y uno para mostrar componentes **Texto estático** que vuelvan a imprimir esos datos.

El archivo JSP para esta página se muestra en la figura 26.18. En las líneas 30 a 34, 35 a 39 y 40 a 44 se definen elementos `ui:textField` para obtener el nombre del usuario, la dirección de e-mail y el número telefónico, respectivamente. En las líneas 45 a 48, 49 a 53 y 54 a 58 se definen elementos `ui:label` para cada uno de estos campos de texto. En las líneas 63 a 74 se definen los elementos `ui:message` de los campos de texto. En las líneas 59 a 62 se define un elemento `ui:button` llamado **Enviar**. En las líneas 75 a 80 se crea un elemento `ui:staticText` llamado `textoResultado`, el cual muestra la respuesta del servidor cuando el usuario envía el formulario con éxito, y en las líneas 81 a 101 se define un elemento `ui:panel` de `cuadrícula` que contiene componentes para repetir la entrada validada del usuario y mostrarla de nuevo en el navegador.

```

1  <?xml version = "1.0" encoding = "UTF-8"?>
2
3  <!-- Fig. 26.18: Validacion.jsp -->
4  <!-- JSP que demuestra la validación de la entrada del usuario. -->
5  <jsp:root version = "1.2" xmlns:f = "http://java.sun.com/jsf/core"
6  xmlns:h = "http://java.sun.com/jsf/html" xmlns:jsf =
7  "http://java.sun.com/JSP/Page" xmlns:ui = "http://www.sun.com/web/ui">
8  <jsp:directive.page contentType = "text/html; charset = UTF-8"
9  pageEncoding = "UTF-8"/>
10 <f:view>
11   <ui:page binding = "#{Validacion.page1}" id = "page1">
12     <ui:html binding = "#{Validacion.html1}" id = "html1">
13       <ui:head binding = "#{Validacion.head1}" id = "head1"
14         title = "Validación">
15           <ui:link binding = "#{Validacion.link1}" id = "link1"
16             url = "/resources/stylesheet.css"/>
17         </ui:head>
18       <ui:body binding = "#{Validacion.body1}" focus = "form1.ctextoNombre"
19         id = "body1" style = "-rave-layout: grid">
20         <ui:form binding = "#{Validacion.form1}" id = "form1">
21           <ui:staticText binding = "#{Validacion.encabezado}" id =
22             "encabezado" style = "font-size: 18px; height: 22px;
23             left: 24px; top: 24px; position: absolute; width: 456px"
24             text = "Por favor complete el siguiente formulario."/>
25           <ui:staticText binding = "#{Validacion.instrucciones}"
26             id = "instrucciones" style = "font-size: 14px;
27             font-style: italic; left: 24px; top: 60px; position:
28             absolute; width: 406 px" text = "Todos los campos son requeridos
29             y deben contener información válida."/>
30           <ui:textField binding = "#{Validacion.ctextoNombre}" columns =
31             "30" id = "ctextoNombre" required = "true" style = "left:
32             180px; top: 96px; position: absolute; width: 216px"
33             validator =
34             "#{Validacion.longitudNombreValidador.validate}"/>
35           <ui:textField binding = "#{Validacion.ctextoEmail}"
36             columns = "28" id = "ctextoEmail" required = "true"
37             style = "left: 180px; top: 144px; position: absolute;
38             width: 216px" validator =
39             "#{Validacion.ctextoEmail_validate}"/>
40           <ui:textField binding = "#{Validacion.ctextoTelefono}"
41             columns = "30" id = "ctextoTelefono" required = "true"
42             style = "left: 180px; top: 192px; position: absolute;
43             width: 216px" validator =
44             "#{Validacion.ctextoTelefono_validate}"/>

```

Figura 26.18 | JSP que demuestra la validación de la entrada del usuario. (Parte I de 4).

```

45      <ui:label binding = "#{Validacion.etiquetaNombre}" for =
46          "ctextoNombre" id = "etiquetaNombre" style = "font-weight:
47          normal; height: 24px; left: 24px; top: 96px;
48          position: absolute; width: 94px" text = "Nombre:"/>
49      <ui:label binding = "#{Validacion.etiquetaEmail}" for =
50          "ctextoEmail" id = "etiquetaEmail" style = "font-weight:
51          normal; height: 24px; left: 24px; top: 144px;
52          position: absolute; width: 142px" text =
53          "Dirección de e-mail: "/>
54      <ui:label binding = "#{Validacion.etiquetaTelefono}" for=
55          "ctextoTelefono" id = "etiquetaTelefono" style = "font-weight:
56          normal; height: 24px; left: 24px; top: 192px
57          position: absolute; width: 142px" text =
58          "Teléfono:"/>
59      <ui:button action = "#{Validacion.botonEnviar_action}"
60          binding = "#{Validacion.botonEnviar}" id =
61          "botonEnviar" style = "position: absolute; left: 24px;
62          top: 240px" text = "Enviar"/>
63      <ui:message binding = "#{Validacion.mensajeEmail}" for =
64          "ctextoEmail" id = "mensajeEmail" showDetail = "false"
65          showSummary = "true" style = "left: 504px; top:
66          144px; position: absolute"/>
67      <ui:message binding = "#{Validacion.mensajeTelefono}" for =
68          "ctextoTelefono" id = "mensajeTelefono" showDetail= "false"
69          showSummary = "true" style = "left: 504px; top:
70          192px; position: absolute"/>
71      <ui:message binding = "#{Validacion.mensajeNombre}" for =
72          "ctextoNombre" id = "mensajeNombre" showDetail = "false"
73          showSummary = "true" style = "left: 504px; top: 96px;
74          position: absolute"/>
75      <ui:staticText binding = "#{Validacion.textoResultado}"
76          escape = "false" id = "textoResultado" rendered = "false"
77          style = "height: 46px; left: 24px; top: 312px;
78          position: absolute; width: 312px" text = "Gracias por
79          enviar sus datos. &lt;br/&gt;Recibimos
80          la siguiente información:"/>
81      <h:panelGrid binding = "#{Validacion.panelCuadrícula}"
82          columns = "2" id = "panelCuadrícula" rendered = "false"
83          style = "background-color: seashell; height: 120px;
84          left: 24px; top: 360px; position: absolute"
85          width = "360">
86          <ui:staticText binding =
87              "#{Validacion.etiquetaResultadoNombre}" id=
88              "etiquetaResultadoNombre" text= "Nombre:"/>
89          <ui:staticText binding = "#{Validacion.resultadoNombre}"
90              id = "resultadoNombre"/>
91          <ui:staticText binding =
92              "#{Validacion.etiquetaResultadoEmail}" id =
93              "etiquetaResultadoEmail" text = "E-mail: "/>
94          <ui:staticText binding = "#{Validacion.resultadoEmail}"
95              id= "resultadoEmail"/>
96          <ui:staticText binding =
97              "#{Validacion.etiquetaResultadoTelefono}" id =
98              "etiquetaResultadoTelefono" text = "Teléfono: "/>
99          <ui:staticText binding = "#{Validacion.resultadoTelefono}"
100              id = "resultadoTelefono"/>
101      </h:panelGrid>
102  </ui:form>
103 </ui:body>

```

Figura 26.18 | JSP que demuestra la validación de la entrada del usuario. (Parte 2 de 4).

```

104     </ui:html>
105     </ui:page>
106     </f:view>
107 </jsp:root>

```

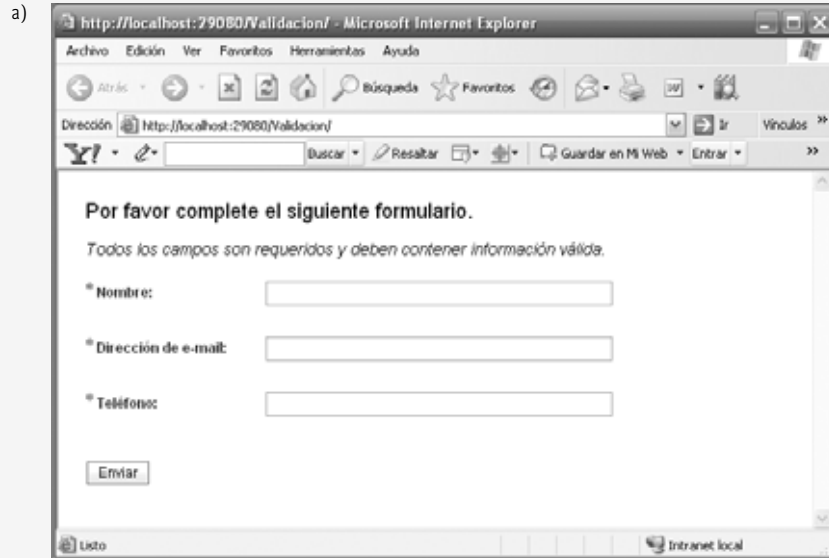


Figura 26.18 | JSP que demuestra la validación de la entrada del usuario. (Parte 3 de 4).

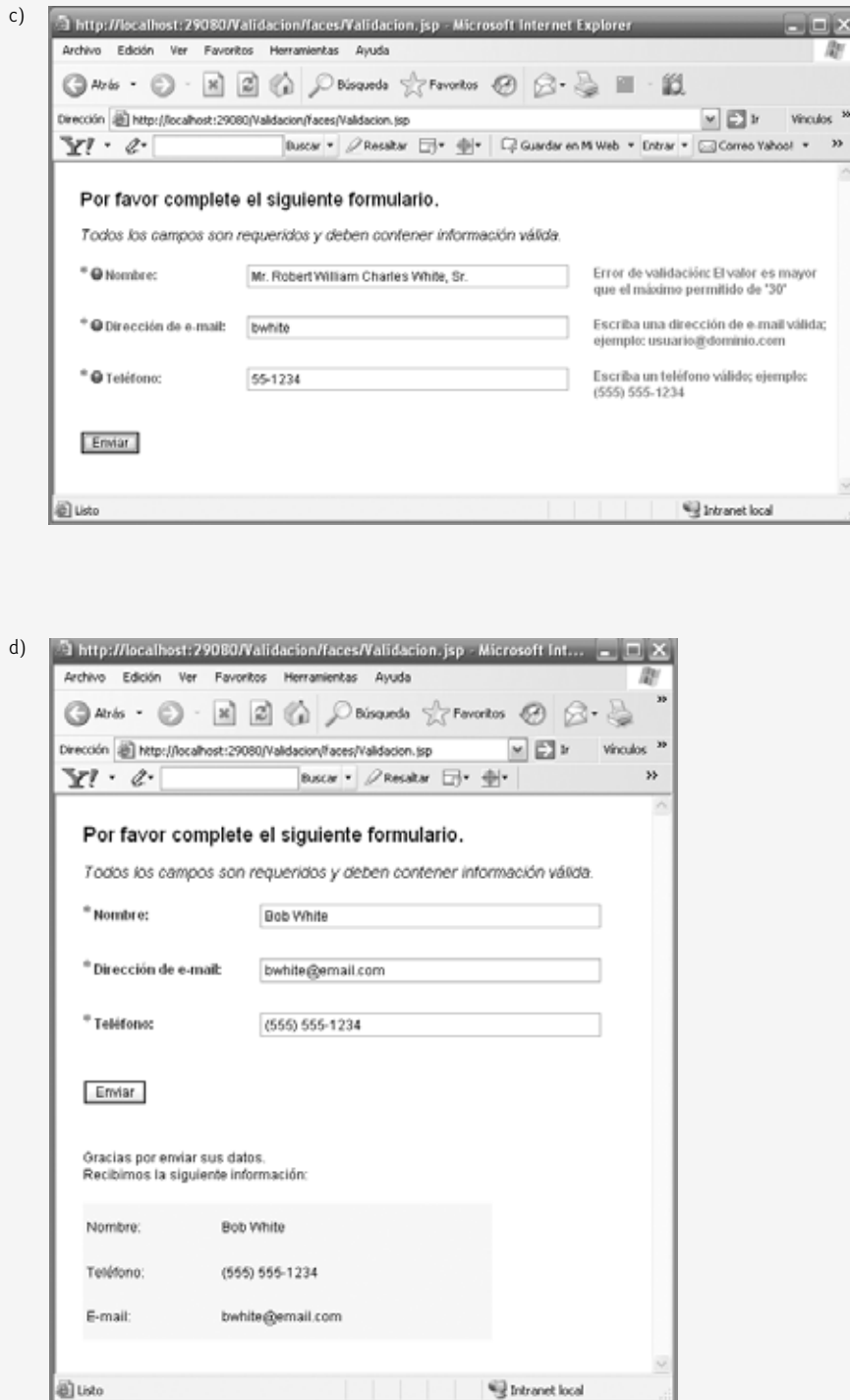


Figura 26.18 | JSP que demuestra la validación de la entrada del usuario. (Parte 4 de 4).

*Cómo establecer la propiedad **Required** de un componente de entrada*

Asegurarse que el usuario haya realizado una selección, o que haya escrito texto en un elemento de entrada requerido, es un tipo básico de validación. Para ello, hay que activar la casilla **required** en la ventana **Properties** del elemento. Si agrega un componente de validación o un método de validación personalizado a un campo de entrada, la propiedad **required** del campo debe establecerse en **true** para que se lleve a cabo la validación. Observe que los tres elementos `ui:textField` de entrada en este ejemplo (figura 26.18, líneas 30 a 44) tienen su propiedad **required** establecida en **true**. Observe además en el Editor visual que la etiqueta para un campo requerido se marca automáticamente mediante un asterisco color rojo. Si un usuario envía este formulario con campos de texto vacíos, se mostrará el mensaje de error predeterminado para un campo requerido en el componente `ui:message` asociado del campo vacío.

*Uso del componente **LengthValidator***

En este ejemplo, utilizamos el componente **Validador de longitud** (el cual se encuentra en la sección **Validadores** de la **Paleta**) para asegurar que la longitud del nombre del usuario no exceda a 30 caracteres. Esto podría ser útil para asegurar que un valor pueda guardarse en un campo específico de la base de datos.

Para agregar un **Validador de longitud** a un componente, simplemente arrastre el validador de la **Paleta** y suéltelo en el campo a validar. A continuación, aparecerá un nodo **lengthValidator** en la sección **Validación** de la ventana **Esquema**. Para editar las propiedades del componente de validación, haga clic en este nodo y establezca las propiedades **maximum** y **minimum** en el número deseado de caracteres en la ventana **Propiedades**. Aquí sólo estableceremos la propiedad **maximum** en 30. También modificamos el **id** del componente a **longitudNombreValidador**. Observe que el campo de entrada **textoNombre** en el archivo JSP se ha enlazado al método **validate** de la propiedad **longitudNombreValidador** en el archivo de bean de página (líneas 33 y 34).

Este validador permite a los usuarios escribir todo el texto que deseen en el campo y, si exceden el límite, se mostrará el mensaje de error de validación de longitud predeterminado en el componente `ui:message` del campo, después de que el usuario haga clic en el botón **Enviar**. Es posible limitar la longitud de la entrada del usuario sin validación. Al establecer la propiedad **maxLength** de un **Campo de texto**, el cursor de este componente no avanzará más allá del máximo número permisible de caracteres, por lo que el usuario no podrá enviar datos que excedan al límite de longitud.

Uso de expresiones regulares para realizar la validación personalizada

Algunas de las tareas de validación más comunes incluyen comprobar la entrada del usuario para el formato apropiado. Por ejemplo, tal vez sea necesario comprobar las direcciones de e-mail y los números telefónicos que se hayan introducido, para asegurar que se conformen al formato estándar para direcciones de e-mail y números telefónicos válidos. Comparar la entrada del usuario con una expresión regular es un método efectivo para asegurar que la entrada tenga un formato apropiado (en la sección 30.7 hablaremos sobre las expresiones regulares). Java Studio Creator 2 no proporciona componentes para validar mediante el uso de expresiones regulares, por lo que nosotros agregaremos nuestros propios métodos de validación al archivo de bean de página. Para agregar un validador personalizado a un componente de entrada, haga clic con el botón derecho del ratón sobre el componente y seleccione **Editar manejador de eventos > validate**. Esto crea un método de validación para el componente, con un cuerpo vacío en el archivo de bean de página. En breve agregaremos código a este método. Observe que los atributos **validate** de **textoEmail** y **textoTelefono** están enlazados con sus respectivos métodos de validación en el archivo de bean de página (líneas 38 a 39 y 43 a 44).

Análisis del archivo de bean de página para un formulario que reciba la entrada del usuario

La figura 26.19 contiene el archivo de bean de página para el archivo JSP de la figura 26.18. En la línea 33 se establece la longitud máxima para **longitudNombreValidador**, que es una propiedad de este bean de página. Recuerde que el campo de texto del nombre se enlazó a esta propiedad en el archivo JSP. Los métodos **textoEmail_validate** (líneas 398 a 410) y **textoTelefono_validate** (líneas 414 a 426) son los métodos validadores personalizados que verifican que el usuario haya introducido la dirección de e-mail y el número telefónico, respectivamente. El método **botonEnviar_action** (líneas 429 a 440) vuelve a imprimir de vuelta al usuario los datos introducidos, si la validación fue exitosa. Los métodos de validación se llaman antes del manejador de eventos, por lo que si la validación falla, no se hará la llamada a **botonEnviar_action** y la entrada del usuario no se repetirá.

Los dos métodos de validación en este archivo de bean de página validan el contenido de un campo de texto, comparándolo con una expresión regular mediante el método `match` de `String`, el cual recibe una expresión regular como argumento y devuelve `true` si ese objeto `String` se conforma con el formato especificado.

```

1  // Fig. 26.19: Validacion.java
2  // Bean de página para validar la entrada del usuario y volver a mostrar esa
3  // entrada si es válida.
4  package validacion;
5
6  import com.sun.rave.web.ui.appbase.AbstractPageBean;
7  import com.sun.rave.web.ui.component.Body;
8  import com.sun.rave.web.ui.component.Form;
9  import com.sun.rave.web.ui.component.Head;
10 import com.sun.rave.web.ui.component.Html;
11 import com.sun.rave.web.ui.component.Link;
12 import com.sun.rave.web.ui.component.Page;
13 import javax.faces.FacesException;
14 import com.sun.rave.web.ui.component.StaticText;
15 import com.sun.rave.web.ui.component.TextField;
16 import com.sun.rave.web.ui.component.TextArea;
17 import com.sun.rave.web.ui.component.Label;
18 import com.sun.rave.web.ui.component.Button;
19 import com.sun.rave.web.ui.component.Message;
20 import javax.faces.component.UIComponent;
21 import javax.faces.context.FacesContext;
22 import javax.faces.validator.ValidatorException;
23 import javax.faces.application.FacesMessage;
24 import javax.faces.component.html.HtmlPanelGrid;
25 import javax.faces.validator.LengthValidator;
26
27 public class Validacion extends AbstractPageBean
28 {
29     private int __placeholder;
30
31     private void _init() throws Exception
32     {
33         longitudNombreValidador.setMaximum( 30 );
34     } // fin del método _init
35
36     // Para ahorrar espacio, omitimos el código de las líneas 36 a 345. El código
37     // fuente completo se proporciona con los ejemplos de este capítulo.
38
39     public Validacion()
40     {
41         // constructor vacío
42     } // fin del constructor
43
44     protected ApplicationBean1 getApplicationBean1()
45     {
46         return (ApplicationBean1)getBean("ApplicationBean1");
47     } // fin del método getApplicationBean1
48
49     protected RequestBean1 getRequestBean1()
50     {
51         return (RequestBean1)getBean("RequestBean1");
52     } // fin del método getRequestBean1

```

Figura 26.19 | Bean de página para validar la entrada del usuario y volver a mostrar esa entrada, si es válida. (Parte I de 3).

```

360
361 protected SessionBean1 getSessionBean1()
362 {
363     return (SessionBean1)getBean("SessionBean1");
364 } // fin del método getSessionBean1
365
366 public void init()
367 {
368     super.init();
369     try
370     {
371         _init();
372     } // fin de try
373     catch (Exception e)
374     {
375         log("Error de inicializacion de Validacion", e);
376         throw e instanceof FacesException ? (FacesException) e :
377             new FacesException(e);
378     } // fin de catch
379 } // fin del método init
380
381 public void preprocess()
382 {
383     // cuerpo vacío
384 } // fin del método preprocess
385
386 public void prerender()
387 {
388     // cuerpo vacío
389 } // fin del método prerender
390
391 public void destroy()
392 {
393     // cuerpo vacío
394 } // fin del método destroy
395
396 // valida la dirección de email introducida, comparándola con la expresión
397 // regular que representa la forma de una dirección de email válida.
398 public void ctextoEmail_validate(FacesContext context,
399     UIComponent component, Object value)
400 {
401     String email = String.valueOf( value );
402
403     // si la dirección de e-mail introducida no está en un formato válido
404     if ( !email.matches(
405         "\\w+([-+.']\\w+)*@\\w+([-.]\\w+)*\\.\\w+([-.]\\w+)*" ) )
406     {
407         throw new ValidatorException( new FacesMessage(
408             "Escriba una direccion de e-mail valida; ejemplo: usuario@dominio.com" ) );
409     } // fin de if
410 } // fin del método ctextoEmail_validate
411
412 // valida el número telefónico introducido, comparándolo con la expresión
413 // regular que representa la forma de un número telefónico válido.
414 public void ctextoTelefono_validate(FacesContext context,
415     UIComponent component, Object value)
416 {
417     String telefono = String.valueOf( value );

```

Figura 26.19 | Bean de página para validar la entrada del usuario y volver a mostrar esa entrada, si es válida. (Parte 2 de 3).

```

418
419 // si el número telefónico introducido no está en un formato válido
420 if ( !telefono.matches(
421     "(\\(\\d{3}\\) ?)|(\\d{3}-)?\\d{3}-\\d{4}" ) )
422 {
423     throw new ValidatorException( new FacesMessage(
424         "Escriba un telefono valido; ejemplo: (555) 555-1234" ) );
425 } // fin de if
426 } // fin del método ctextoTelefono_validate
427
428 // muestra las entradas del formulario validadas en un Panel de cuadrícula
429 public String botonEnviar_accion()
430 {
431     String nombre = String.valueOf( ctextoNombre.getValue() );
432     String email = String.valueOf( ctextoEmail.getValue() );
433     String telefono = String.valueOf( ctextoTelefono.getValue() );
434     resultadoNombre.setValue( nombre );
435     resultadoEmail.setValue( email );
436     resultadoTelefono.setValue( telefono );
437     panelCuadrícula.setRendered( true );
438     textoResultado.setRendered( true );
439     return null;
440 } // fin del método botonEnviar_accion
441 } // fin de la clase Validacion

```

Figura 26.19 | Bean de página para validar la entrada del usuario y volver a mostrar esa entrada, si es válida. (Parte 3 de 3).

Para el método `ctextoEmail_validate`, usamos la siguiente expresión de validación:

```
\w+([-+.'\\w+)*@\\w+([-.'\\w+)*\\.\\w+([-.'\\w+)*
```

Observe que cada barra diagonal inversa en la expresión regular `String` (línea 405) se debe escapar con otra barra diagonal inversa (como en `\\`), ya que el carácter de barra diagonal inversa normalmente representa el inicio de una secuencia de escape. Esta expresión regular indica que una dirección de e-mail es válida si la parte antes del símbolo `@` contiene uno o más caracteres de palabra (es decir, caracteres alfanuméricos o de guión bajo), seguidos de uno o más objetos `String` compuestos de un guión corto, signo más, punto o apóstrofo (`'`) y de más caracteres de palabra. Después del símbolo `@`, una dirección de e-mail válida debe contener uno o más grupos de caracteres de palabras, que pueden estar separados por guiones cortos o puntos, seguidos de un punto requerido y de otro grupo de uno o más caracteres, que pueden estar separados por guiones cortos o puntos. Por ejemplo, las direcciones de e-mail `bob's-personal.email@white.email.com`, `bob-white@my-email.com` y `bob.white@email.com` son todas válidas. Si el usuario escribe texto en `ctextoEmail` que no tenga el formato correcto y trata de enviar el formulario, en las líneas 407 y 408 se lanza una excepción `ValidatorException`. El componente `mensajeEmail` atraparé esta excepción y mostrará el mensaje en color rojo.

La expresión regular en `ctextoTelefono_validate` asegura que el componente `ctextoTelefono` contenga un número telefónico válido antes de enviar el formulario. La entrada del usuario se compara con la expresión regular

```
(\\(\\d{3}\\) ?)|(\\d{3}-)?\\d{3}-\\d{4}
```

(De nuevo, cada barra diagonal inversa se escapa en la expresión regular `String` de la línea 421). Esta expresión indica que un número telefónico puede contener un código de área de tres dígitos, ya sea entre paréntesis o no, y debe ir seguido de un espacio opcional, o sin paréntesis y seguido por un guión corto obligatorio. Después de un código de área opcional, un número telefónico debe contener tres dígitos, un guión corto y otros cuatro dígitos. Por ejemplo, `(555) 123-4567`, `555-123-4567` y `123-4567` son todos números telefónicos válidos. Si un usuario escribe un número telefónico inválido, en las líneas 423 y 424 se lanza una excepción `ValidatorException`. El componente `mensajeTelefono` atrapa esta excepción y muestra el mensaje de error en color rojo.

Si todos los seis validadores tienen éxito (es decir, que cada componente `TextField` contenga datos, que el nombre tenga menos de 30 caracteres y que la dirección de e-mail y el número telefónico sean válidos), al hacer clic en el botón **Enviar** se enviarán los datos del formulario al servidor. Como se muestra en la figura 26.18(d), el método `botonEnviar_action` muestra los datos enviados en un `panelCuadrícula` (líneas 434 a 437) y un mensaje de éxito en `textoResultado` (línea 438).

26.7 Rastreo de sesiones

En los primeros días de Internet, los comercios electrónicos no podían proveer el tipo de servicio personalizado que comúnmente se experimenta en las tiendas reales. Para lidiar con este problema, los comercios electrónicos empezaron a establecer mecanismos mediante los cuales pudieran personalizar las experiencias de navegación de los usuarios, preparando contenido a la medida de los usuarios individuales, permitiéndoles ignorar al mismo tiempo la información irrelevante. Para lograr este nivel de servicio, los comercios rastrean el movimiento de cada cliente a través de sus sitios Web y combinan los datos recolectados con la información que proporciona el consumidor, incluyendo la información de facturación y las preferencias personales, intereses y pasatiempos.

Personalización

La **personalización** hace posible que los comercios electrónicos se comuniquen con eficiencia con sus clientes, y también mejora la habilidad del usuario para localizar los productos y servicios deseados. Las compañías que proporcionan contenido de interés especial para los usuarios pueden establecer relaciones con los clientes, y fomentar esas relaciones con el paso del tiempo. Además, al enviar a los clientes ofertas personales, recomendaciones, anuncios, promociones y servicios, los comercios electrónicos crean una lealtad en los clientes. Los sitios Web pueden utilizar tecnología sofisticada para permitir a los visitantes personalizar las páginas de inicio para satisfacer sus necesidades y preferencias personales. De manera similar, los sitios de compras en línea comúnmente almacenan la información personal para los clientes, notificaciones personalizadas y ofertas especiales de acuerdo con sus intereses. Dichos servicios alientan a los clientes a visitar los sitios y realizar compras con más frecuencia.

Privacidad

Sin embargo, existe una concesión entre el servicio de comercio electrónico personalizado y la protección de la privacidad. Algunos consumidores adoptan la idea del contenido personalizado, pero otros temen a las posibles consecuencias adversas, si la información que proporcionan a los comercios electrónicos es liberada o recolectada por tecnologías de rastreo. Los consumidores y los defensores de la privacidad preguntan: ¿Qué pasa si el comercio electrónico al que proporcionamos nuestros datos personales vende o proporciona esa información a otra organización, sin nuestro consentimiento? ¿Qué pasa si no queremos que nuestras acciones en Internet (un medio supuestamente anónimo) sean rastreadas y registradas por terceros desconocidos? ¿Qué pasa si personas no autorizadas obtienen acceso a los datos privados delicados, como los números de tarjetas de crédito o el historial médico? Todas estas son preguntas con las que los programadores, consumidores, comercios electrónicos y legisladores deben debatir y lidiar.

Cómo reconocer a los clientes

Para proporcionar servicios personalizados a los consumidores, los comercios electrónicos deben tener la capacidad de reconocer a los clientes cuando solicitan información de un sitio. Como hemos visto antes, el sistema de petición/respuesta en el que opera la Web se lleva a cabo mediante HTTP. Por desgracia, HTTP es un protocolo sin estado; no soporta conexiones persistentes que permitan a los servidores Web mantener información de estado, en relación con clientes específicos. Por lo tanto, los servidores Web no pueden determinar si una petición proviene de un cliente específico, o si una serie de peticiones provienen de uno o varios clientes. Para sortear este problema, los sitios pueden proporcionar mecanismos para identificar a los clientes individuales. Una sesión representa a un cliente único en un sitio Web. Si el cliente sale de un sitio y regresa después, aún será reconocido como el mismo usuario. Para ayudar al servidor a diferenciar un cliente de otro, cada cliente debe identificarse a sí mismo con el servidor. El rastreo de clientes individuales, conocido como **rastreo de sesiones**, puede lograrse de varias formas. Una técnica popular utiliza cookies (sección 26.7.1); otra utiliza el objeto `SessionBean` (sección 26.7.2). Otras técnicas adicionales de rastreo de sesiones incluyen el uso de elementos `input form` de tipo "hidden" y la reescritura de URLs. Con los elementos "hidden", un formulario Web puede escribir los datos de rastreo de sesión en un componente `form` en la página Web que devuelve al cliente, en respuesta a una petición

previa. Cuando el usuario envía el formulario en la nueva página Web, todos los datos del formulario (incluyendo los campos "hidden") se envían al manejador del formulario en el servidor Web. Con la reescritura de URLs, el servidor Web incrusta la información de rastreo de sesión directamente en los URLs de los hipervínculos en los que el usuario hace clic para enviar las subsiguientes peticiones al servidor Web.

26.7.1 Cookies

Las **cookies** proporcionan a los desarrolladores Web una herramienta para personalizar las páginas Web. Una cookie es una pieza de datos que, por lo general, se almacena en un archivo de texto en la computadora del usuario. Una cookie mantiene información acerca del cliente, durante y entre las sesiones del navegador. La primera vez que un usuario visita el sitio Web, su computadora podría recibir una cookie; después, esta cookie se reactiva cada vez que el usuario vuelve a visitar ese sitio. La información recolectada tiene el propósito de ser un registro anónimo que contiene datos, los cuales se utilizan para personalizar las visitas futuras del usuario al sitio Web. Por ejemplo, las cookies en una aplicación de compras podría almacenar identificadores únicos para los usuarios. Cuando un usuario agregue elementos a un carrito de compras en línea, o cuando realice alguna otra tarea que origine una petición al servidor Web, éste recibe una cookie del cliente, la cual contiene el identificador único del usuario. Después, el servidor utiliza el identificador único para localizar el carrito de compras y realizar cualquier procesamiento requerido.

Además de identificar a los usuarios, las cookies también pueden indicar las preferencias de compra del usuario. Cuando un servidor Web recibe una petición de un cliente, el servidor puede analizar la(s) cookie(s) que envió al cliente durante las sesiones previas de comunicación, con lo cual puede identificar las preferencias del cliente y mostrar de inmediato productos que sean de su interés.

Cada interacción basada en HTTP entre un cliente y un servidor incluye un encabezado, el cual contiene información sobre la petición (cuando la comunicación es del cliente al servidor) o sobre la respuesta (cuando la comunicación es del servidor al cliente). Cuando una página recibe una petición, el encabezado incluye información como el tipo de petición (por ejemplo, GET o POST) y cualquier cookie que se haya enviado anteriormente del servidor, para almacenarse en el equipo cliente. Cuando el servidor formula su respuesta, la información del encabezado contiene cualquier cookie que el servidor desee almacenar en la computadora cliente, junto con más información, como el tipo MIME de la respuesta.

La **fecha de expiración** de una cookie determina la forma en que ésta permanecerá en la computadora del cliente. Si no establecemos una fecha de expiración para la cookie, el navegador Web mantendrá la cookie mientras dure la sesión de navegación. En caso contrario, el navegador Web mantendrá la cookie hasta que llegue la fecha de expiración. Cuando el navegador solicita un recurso de un servidor Web, las cookies que el servidor Web envió previamente al cliente se devuelven al servidor como parte de la petición formulada por el navegador. Las cookies se eliminan cuando **expiran**.



Tip de portabilidad 26.1

Los clientes pueden deshabilitar las cookies en sus navegadores Web para tener más privacidad. Cuando esos clientes utilicen aplicaciones Web que dependan de las cookies para mantener la información de estado, las aplicaciones no se ejecutarán correctamente.

Uso de cookies para proporcionar recomendaciones de libros

La siguiente aplicación Web muestra cómo utilizar cookies. El ejemplo contiene dos páginas. En la primera página (figuras 26.20 y 26.22), los usuarios seleccionan un lenguaje de programación favorito de un grupo de botones de opción y envían el formulario al servidor Web, para que éste lo procese. El servidor Web responde creando una cookie que almacena el lenguaje seleccionado y el número ISBN para un libro recomendado sobre ese tema. Después, el servidor despliega nuevos componentes en el navegador, que permiten al usuario seleccionar otro lenguaje de programación favorito o ver la segunda página en nuestra aplicación (figuras 26.23 y 26.24), la cual enlista los libros recomendados que pertenezcan al (los) lenguaje(s) de programación que el usuario haya seleccionado. Cuando el usuario hace clic en el hipervínculo, las cookies previamente almacenadas en el cliente se leen y se utilizan para formar la lista de recomendaciones de libros.

El archivo JSP de la figura 26.20 contiene un **Grupo de botones de selección** (líneas 26 a 39) con las opciones **Java**, **C**, **C++**, **Visual Basic 2005** y **Visual C# 2005**. Recuerde que puede establecer los objetos **String** **Mostrar**

y Valor de los botones de opción haciendo clic derecho en **Grupo de botones de selección** y seleccionando **Configurar opciones predeterminadas**. Para seleccionar un lenguaje de programación, el usuario debe hacer clic en uno de los botones de opción. Cuando el usuario oprime el botón **Enviar**, la aplicación Web crea una cookie que contiene el lenguaje seleccionado. Esta cookie se agrega al encabezado de respuesta HTTP y se envía al cliente como parte de la respuesta.

Al hacer clic en **Enviar**, se ocultan los elementos `ui:label`, `ui:radioButtonGroup` y `ui:button` que se utilizan para seleccionar un lenguaje, y se muestran un elemento `ui:staticText` y dos elementos `ui:hyperlink`. Al principio, cada elemento `ui:staticText` y `ui:hyperlink` tiene establecida su propiedad `rendered` en `false` (líneas 31, 37 y 43). Esto indica que estos componentes no son visibles la primera vez que se carga la página, ya que queremos que la primera vez que el usuario vea la página sólo se incluyan los componentes para seleccionar un lenguaje de programación y enviar la selección.

```

1  <?xml version = "1.0" encoding = "UTF-8"?>
2
3  <!-- Fig. 26.20: Opciones.jsp -->
4  <!-- Archivo JSP que permite al usuario seleccionar un lenguaje de programación -->
5  <jsp:root version = "1.2" xmlns:f = "http://java.sun.com/jsp/core"
6  xmlns:h = "http://java.sun.com/jsp/html" xmlns:jsp =
7  "http://java.sun.com/JSP/Page" xmlns:ui = "http://www.sun.com/web/ui">
8  <jsp:directive.page contentType = "text/html; charset = UTF-8"
9  pageEncoding = "UTF-8"/>
10 <f:view>
11   <ui:page binding = "#{Opciones.page}" id = "page">
12     <ui:html binding = "#{Opciones.html}" id = "html">
13       <ui:head binding = "#{Opciones.head}" id = "head" title=
14       "Opciones">
15         <ui:link binding = "#{Opciones.link}" id = "link"
16         url = "/resources/styleSheet.css"/>
17       </ui:head>
18       <ui:body binding = "#{Opciones.body}" id = "body"
19       style = "-rave-layout: grid">
20         <ui:form binding = "#{Opciones.form}" id = "form">
21           <ui:label binding = "#{Opciones.etiquetaLenguaje}" for =
22           "listaLenguajes" id = "etiquetaLenguaje" style =
23           "font-size: 16px; font-weight: bold; left: 24px; top:
24           24px; position: absolute" text = "Seleccione un
25           lenguaje de programación:"/>
26           <ui:radioButtonGroup binding = "#{Opciones.listaLenguajes}"
27           id = "listaLenguajes" items =
28           "#{Opciones.listaLenguajesDefaultOptions.options}" style =
29           "left: 24px; top: 48px; position: absolute"/>
30           <ui:staticText binding = "#{Opciones.etiquetaRespuesta}" id =
31           "etiquetaRespuesta" rendered = "false" style =
32           "font-size: 16px; font-weight: bold; height: 24px;
33           left: 24px; top: 24px; position: absolute;
34           width: 216px"/>
35           <ui:hyperlink action = "#{Opciones.vinculoLenguajes_action}"
36           binding = "#{Opciones.vinculoLenguajes}" id =
37           "vinculoLenguajes" rendered = "false" style = "left:
38           24px; top: 96px; position: absolute" text = "Haga clic aqui
39           para elegir otro lenguaje."/>
40           <ui:hyperlink action =
41           "#{Opciones.vinculoRecomendaciones_action}" binding =
42           "#{Opciones.vinculoRecomendaciones}" id =

```

Figura 26.20 | Archivo JSP que permite al usuario seleccionar un lenguaje de programación. (Parte I de 3).

```

43 "vinculoRecomendaciones" rendered = "false" style =
44 "left: 24px; top: 120px; position: absolute" text =
45 "Haga clic aquí para obtener recomendaciones de libros."
46 url = "/faces/Recomendaciones.jsp"/>
47 <ui:button action = "#{Opciones.enviar_action}" binding =
48 "#{Opciones.enviar}" id = "enviar" style = "left:
49 23px; top: 192px; position: absolute" text =
50 "Enviar"/>
51 </ui:form>
52 </ui:body>
53 </ui:html>
54 </ui:page>
55 </f:view>
56 </jsp:root>

```

a)



b)

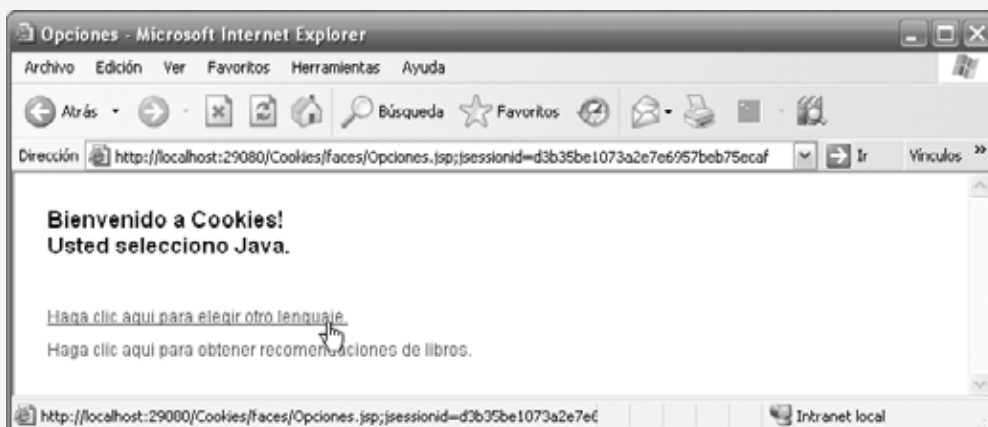


Figura 26.20 | Archivo JSP que permite al usuario seleccionar un lenguaje de programación. (Parte 2 de 3).

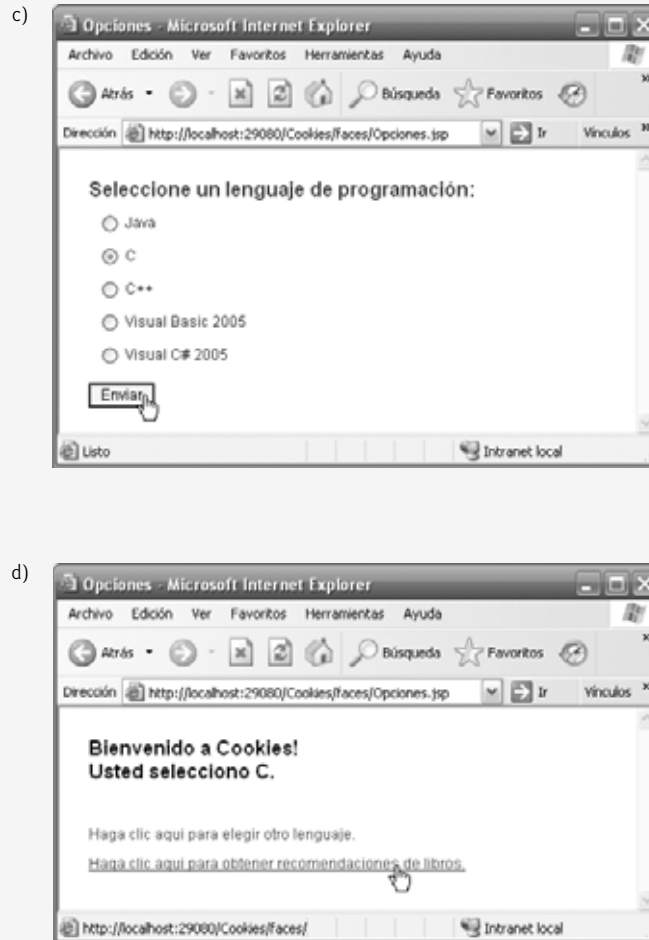


Figura 26.20 | Archivo JSP que permite al usuario seleccionar un lenguaje de programación. (Parte 3 de 3).

El primer hipervínculo (líneas 35 a 39) solicita esta página, y el segundo (líneas 40 a 46) solicita Recomendaciones.jsp. La propiedad url del segundo vínculo se establece en /faces/Recomendaciones.jsp. Recuerde que en un ejemplo anterior en este capítulo, establecimos una propiedad url a un sitio Web remoto (<http://www.deitel.com>). Para establecer esta propiedad a una página dentro de la aplicación actual, haga clic en el botón de elipsis que está enseguida de la propiedad url en la ventana **Propiedades**, para abrir un cuadro de diálogo. Use este cuadro de diálogo para seleccionar una página dentro de su proyecto como destino para el vínculo.

Cómo agregar una nueva página y crear un vínculo

Para establecer la propiedad url a una página en la aplicación actual, la página de destino debe existir de antemano. Para establecer la propiedad url de un vínculo a Recomendaciones.jsp, primero debemos crear esta página. Haga clic en el nodo **Páginas Web** en la ventana **Propiedades** y seleccione **Nuevo > Página** en el menú que aparezca. En el cuadro de diálogo **Nueva Página**, cambie el nombre de la página a Recomendaciones y haga clic en **Terminar** para crear los archivos Recomendaciones.jsp y Recomendaciones.java. (En breve hablaremos sobre el contenido de estos archivos). Una vez que exista el archivo Recomendaciones.jsp, puede seleccionarlo como el valor de url para vínculoRecomendaciones.

Para Opciones.jsp, en vez de establecer la propiedad url de vínculoLenguajes, vamos a agregar al bean de página un manejador de acciones para este componente. El manejador de acciones nos permitirá mostrar y

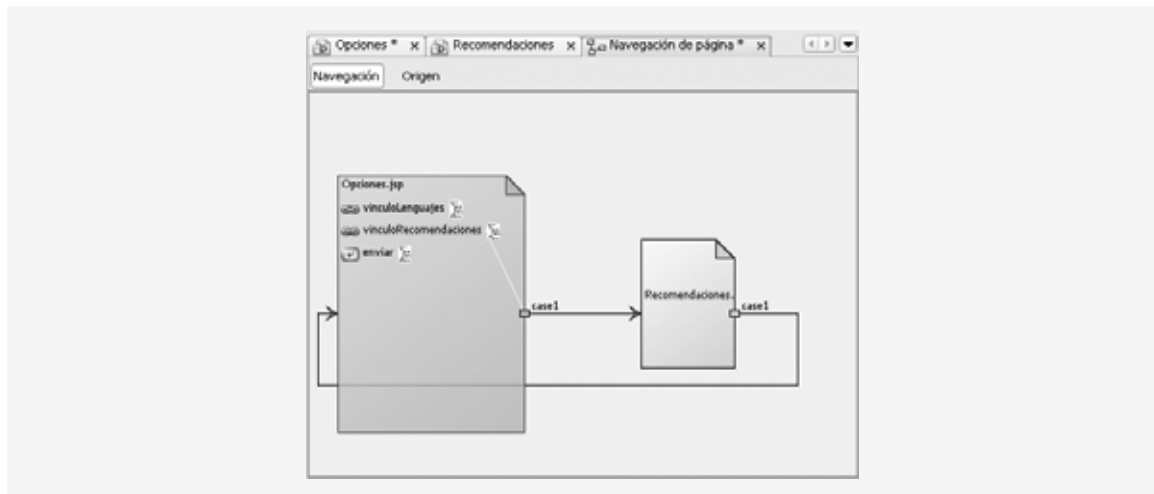


Figura 26.21 | Edición del archivo **Navegación de página**.

ocultar componentes de la página, sin necesidad de redirigir al usuario a otra página. Si especificamos un `url` de destino, se sobrescribirá el manejador de acciones del componente y el usuario será redirigido a la página especificada, por lo cual es importante que no establezcamos la propiedad `url` en este caso. Como vamos a utilizar este vínculo para volver a cargar la página actual, el manejador de acciones simplemente debe devolver `null`, lo cual hará que `Opciones.jsp` se vuelva a cargar.

Para agregar un manejador de acciones a un hipervínculo que también debe dirigir al usuario a otra página, debemos agregar una regla al archivo **Navegación de página** (figura 26.21). Para editar este archivo, haga clic con el botón derecho del ratón en cualquier parte del Diseñador visual y seleccione **Navegación de página....** Localice el vínculo cuya regla de navegación desea establecer y arrástrelo a la página de destino. Ahora el vínculo puede dirigir al usuario a una nueva página sin sobrescribir su manejador de acciones. También es útil editar el archivo **Navegación de página** cuando es conveniente tener elementos de acción que no puedan especificar una propiedad `url`, como los botones, para dirigir a los usuarios a otra página.

La figura 26.22 contiene el código que escribe una cookie al equipo cliente, cuando el usuario selecciona un lenguaje de programación. El archivo también determina cuáles componentes deben aparecer en la página, mostrando ya sea los componentes para elegir un lenguaje, o los vínculos para navegar por la aplicación, dependiendo de las acciones del usuario.

```

1 // Fig. 26.22: Opciones.java
2 // Bean de página que almacena la selección de lenguaje del usuario como
3 // una cookie en el cliente.
4 package cookies;
5
6 import com.sun.rave.web.ui.appbase.AbstractPageBean;
7 import com.sun.rave.web.ui.component.Body;
8 import com.sun.rave.web.ui.component.Form;
9 import com.sun.rave.web.ui.component.Head;
10 import com.sun.rave.web.ui.component.Html;
11 import com.sun.rave.web.ui.component.Link;
12 import com.sun.rave.web.ui.component.Page;
13 import javax.faces.FacesException;
14 import com.sun.rave.web.ui.component.StaticText;

```

Figura 26.22 | Bean de página que almacena la selección de lenguaje del usuario como una cookie en el cliente. (Parte I de 4).

```

15 import com.sun.rave.web.ui.component.Label;
16 import com.sun.rave.web.ui.component.RadioButtonGroup;
17 import com.sun.rave.web.ui.model.SingleSelectOptionsList;
18 import com.sun.rave.web.ui.component.HyperLink;
19 import com.sun.rave.web.ui.component.Button;
20 import javax.servlet.http.HttpServletResponse;
21 import javax.servlet.http.Cookie;
22 import java.util.Properties;
23
24 public class Opciones extends AbstractPageBean
25 {
26     private int __placeholder;
27
28     // el método _init inicializa los componentes y establece
29     // las opciones para el grupo de botones de selección.
30     private void _init() throws Exception
31     {
32         listaLenguajesDefaultOptions.setOptions(
33             new com.sun.rave.web.ui.model.Option[]
34             {
35                 new com.sun.rave.web.ui.model.Option("Java", "Java"),
36                 new com.sun.rave.web.ui.model.Option("C", "C"),
37                 new com.sun.rave.web.ui.model.Option("C++", "C++"),
38                 new com.sun.rave.web.ui.model.Option("Visual/Basic/2005",
39                     "Visual Basic 2005"),
40                 new com.sun.rave.web.ui.model.Option("Visual/C#/2005",
41                     "Visual C# 2005")
42             }
43         );
44     } // fin del método _init
45
46     // Para ahorrar espacio, omitimos el código en las líneas 46 a 203. El código
47     // fuente completo se proporciona con los ejemplos de este capítulo.
48
204     private Properties libros = new Properties();
205
206     // Construye una nueva instancia del bean de página e inicializa las propiedades
207     // que asocian los lenguajes con los números ISBN de los libros recomendados.
208     public Opciones()
209     {
210         // inicializa el objeto Properties de los valores que se van
211         // a almacenar como cookies.
212         libros.setProperty( "Java", "0-13-222220-5" );
213         libros.setProperty( "C", "0-13-142644-3" );
214         libros.setProperty( "C++", "0-13-185757-6" );
215         libros.setProperty( "Visual/Basic/2005", "0-13-186900-0" );
216         libros.setProperty( "Visual/C#/2005", "0-13-152523-9" );
217     } // fin del constructor de Opciones
218
219     protected ApplicationBean getApplicationBean()
220     {
221         return (ApplicationBean) getBean( "ApplicationBean" );
222     } // fin del método getApplicationBean
223
224     protected RequestBean getRequestBean()
225     {
226         return (RequestBean) getBean( "RequestBean" );
227     } // fin del método getRequestBean

```

Figura 26.22 | Bean de página que almacena la selección de lenguaje del usuario como una cookie en el cliente. (Parte 2 de 4).

```

228
229     protected SessionBean getSessionBean()
230     {
231         return (SessionBean) getBean( "SessionBean" );
232     } // fin del método getSessionBean
233
234     public void init()
235     {
236         super.init();
237         try
238         {
239             _init();
240         } // fin de try
241         catch ( Exception e )
242         {
243             log( "Error al inicializar Opciones", e );
244             throw e instanceof FacesException ? ( FacesException ) e :
245                 new FacesException( e );
246         } // fin de catch
247     } // fin del método init
248
249     public void preprocess()
250     {
251         // cuerpo vacío
252     } // fin del método preprocess
253
254     public void prerender()
255     {
256         // cuerpo vacío
257     } // fin del método prerender
258
259     public void destroy()
260     {
261         // cuerpo vacío
262     } // fin del método destroy
263
264     // Manejador de acciones para el botón Enviar. Verifica si se seleccionó un
265     // lenguaje y, de ser así, registra una cookie para ese lenguaje, y
266     // establece la etiquetaRespuesta para indicar el lenguaje seleccionado.
267     public String enviar_action()
268     {
269         String msj = "Bienvenido a Cookies!   Usted ";
270
271         // si el usuario hizo una selección
272         if ( listaLenguajes.getSelected() != null )
273         {
274             String lenguaje = listaLenguajes.getSelected().toString();
275             String mostrarLenguaje = lenguaje.replace( '/', ' ' );
276             msj += "selecciono " + mostrarLenguaje + ".";
277
278             // obtiene el número ISBN del libro para el lenguaje dado.
279             String ISBN = libros.getProperty( lenguaje );
280
281             // crea cookie usando un par nombre-valor de lenguaje-ISBN
282             Cookie cookie = new Cookie( lenguaje, ISBN );
283
284             // agrega la cookie al encabezado de respuesta para colocarla en
285             // el equipo del usuario

```

Figura 26.22 | Bean de página que almacena la selección de lenguaje del usuario como una cookie en el cliente. (Parte 3 de 4).

```

286     HttpServletResponse respuesta =
287         (HttpServletResponse) getServletContext().getResponse();
288     respuesta.addCookie( cookie );
289 } // fin de if
290 else
291     msj += "no selecciono un lenguaje.";
292
293     etiquetaRespuesta.setValue(msj);
294     listaLenguajes.setRendered(false);
295     etiquetaLenguaje.setRendered(false);
296     enviar.setRendered(false);
297     etiquetaRespuesta.setRendered(true);
298     vinculoLenguajes.setRendered(true);
299     vinculoRecomendaciones.setRendered(true);
300     return null; // vuelve a cargar la página
301 } // fin del método enviar_action
302
303 // vuelve a mostrar los componentes utilizados para permitir al usuario
304 // seleccionar un lenguaje.
305 public String vinculoLenguajes_action()
306 {
307     etiquetaRespuesta.setRendered(false);
308     vinculoLenguajes.setRendered(false);
309     vinculoRecomendaciones.setRendered(false);
310     listaLenguajes.setRendered(true);
311     etiquetaLenguaje.setRendered(true);
312     enviar.setRendered(true);
313     return null;
314 } // fin del método vinculoLenguajes_action
315 } // fin de class Opciones

```

Figura 26.22 | Bean de página que almacena la selección de lenguaje del usuario como una cookie en el cliente. (Parte 4 de 4).

Como dijimos antes, el método `_init` maneja la inicialización de componentes. Como esta página contiene un objeto `RadioButtonGroup` que requiere inicialización, el método `_init` (líneas 30 a 44) construye un arreglo de objeto `Options` que van a mostrar los botones. En las líneas 38 y 40, los nombres de las opciones contienen barras diagonales en vez de espacios, ya que posteriormente los utilizamos como nombres de cookies y Java no permite que los nombres de cookies tengan espacios.

En las líneas 212 a 216 del constructor se inicializa un objeto `Properties`: una estructura de datos que almacena pares clave-valor tipo `String`. La aplicación utiliza la clave para almacenar y obtener el valor asociado en el objeto `Properties`. En este ejemplo, las claves son objetos `String` que contienen los nombres de los lenguajes de programación, y los valores son objetos `String` que contienen los números ISBN para los libros recomendados. La clase `Properties` proporciona el método `setProperty`, el cual recibe como argumentos una clave y un valor. Un valor que se agrega a través del método `setProperty` se coloca en el objeto `Properties`, en una ubicación determinada por la clave. El valor para una entrada específica en el objeto `Properties` se puede determinar invocando al método `getProperty` en el objeto `Properties`, con la clave del valor como argumento.



Observación de ingeniería de software 26.1

*Java Studio Creator2 puede importar automáticamente cualquier paquete que necesite su archivo de Java y que no esté incluido. Por ejemplo, después de agregar el objeto `Properties` a `Opciones.java`, puede hacer clic con el botón derecho en la ventana del editor de Java y seleccionar **Corregir importaciones** para importar automáticamente el paquete `java.util.Properties`.*

Al hacer clic en **Enviar**, se invoca el manejador de eventos `enviar_action` (líneas 267 a 301), el cual muestra un mensaje indicando el lenguaje seleccionado en el elemento `etiquetaRespuesta`, y agrega una nueva cookie a

la respuesta. Si se seleccionó un lenguaje (línea 272) se obtiene el valor seleccionado (línea 274). En la línea 275 se convierte la selección en un objeto `String` que se puede mostrar en la `etiquetaRespuesta`, sustituyendo las barras diagonales con espacios. En la línea 276 se agrega el lenguaje seleccionado al mensaje de resultados.

En la línea 279 se obtiene el ISBN para el lenguaje seleccionado de las propiedades (`Properties`) de los libros. Después, en la línea 282 se crea un nuevo objeto `cookie` (de la clase `Cookie` en el paquete `javax.servlet.http`), usando el lenguaje seleccionado como el nombre de la `cookie` y su correspondiente número ISBN como el valor de la `cookie`. Esta `cookie` se agrega al encabezado de respuesta HTTP en las líneas 286 a 288. Un objeto de la clase `HttpServletResponse` (del paquete `javax.servlet.http`) representa la respuesta. Para acceder a este objeto, se invoca el método `getExternalContext` en el bean de página y después se invoca a `getResponse` en el objeto resultante. Si no se seleccionó un lenguaje, en la línea 291 se establece el mensaje de resultados para indicar que no hubo selección.

Las líneas 293 a 299 controlan la apariencia de la página, una vez que el usuario hace clic en **Enviar**. En la línea 293 se establece la `etiquetaRespuesta` para mostrar el objeto `String` llamado `msj`. Como el usuario acaba de enviar una selección de lenguaje, se ocultan los componentes utilizados para recolectar la selección (líneas 294 a 296), y se muestran `etiquetaRespuesta` y los vínculos utilizados para navegar por la aplicación (líneas 297 a 299). El manejador de acciones devuelve `null` en la línea 300, la cual vuelve a cargar `Opciones.jsp`.

Las líneas 305 a 311 contienen el manejador de eventos de `vinculoLenguajes`. Cuando el usuario hace clic en este vínculo, se ocultan `etiquetaRespuesta` y los dos vínculos (líneas 307 y 309), y se vuelven a mostrar los componentes que permiten al usuario seleccionar un lenguaje (líneas 310 a 312). El método devuelve `null` en la línea 313, lo cual hace que `Opciones.jsp` se vuelva a cargar.

Cómo mostrar las recomendaciones de libros con base en los valores de cookies

Después de hacer clic en **Enviar**, el usuario puede solicitar la recomendación de un libro. El hipervínculo de recomendaciones de libros lleva al usuario a `Recomendaciones.jsp` (figura 26.23) para mostrar las recomendaciones con base en los lenguajes seleccionados por el usuario.

```

1  <?xml version = "1.0" encoding = "UTF-8"?>
2
3  <!-- Fig. 26.23: Recomendaciones.jsp -->
4  <!-- Muestra las recomendaciones de libros mediante el uso de cookies -->
5  <jsp:root version = "1.2" xmlns:f = "http://java.sun.com/jsp/core"
6      xmlns:h = "http://java.sun.com/jsp/html" xmlns:jsp =
7      "http://java.sun.com/JSP/Page" xmlns:ui = "http://www.sun.com/web/ui">
8      <jsp:directive.page contentType = "text/html; charset = UTF-8"
9          pageEncoding = "UTF-8"/>
10     <f:view>
11         <ui:page binding = "#{Recomendaciones.page}" id = "page">
12             <ui:html binding = "#{Recomendaciones.html}" id = "html">
13                 <ui:head binding = "#{Recomendaciones.head}" id = "head">
14                     title = "Recomendaciones">
15                         <ui:link binding = "#{Recomendaciones.link}" id = "link"
16                             url= "/resources/styleSheet.css"/>
17                 </ui:head>
18                 <ui:body binding = "#{Recomendaciones.body}" id = "body">
19                     style = "-rave-layout: grid">
20                         <ui:form binding = "#{Recomendaciones.form}" id = "form">
21                             <ui:label binding = "#{Recomendaciones.etiquetaLenguaje}"
22                                 for = "cuadroListaLibrosOpciones" id = "etiquetaLenguaje" style =
23                                 "font-size: 20px; font-weight: bold; left: 24px; top:
24                                 24px; position: absolute" text = "Recomendaciones"/>
25                             <ui:listbox binding = "#{Recomendaciones.cuadroListaLibros}"
26                                 id = "cuadroListaLibros" items = "#{Recomendaciones.
27                                 cuadroListaLibrosOpciones.options}" rows = "6" style =
28                                 "left: 24px; top: 72px; position: absolute;

```

Figura 26.23 | Archivo de JSP que muestra las recomendaciones de libros, con base en cookies. (Parte I de 2).

```

29         width: 360px"/>
30     <ui:hyperlink action = "case1" binding =
31         "#{Recomendaciones.vinculoOpciones}" id = "vinculoOpciones"
32         style = "left: 24px; top: 192px; position: absolute"
33         text = "Haga clic aquí para elegir otro lenguaje."/>
34     </ui:form>
35 </ui:body>
36 </ui:html>
37 </ui:page>
38 </f:view>
39 </jsp:root>

```



Figura 26.23 | Archivo de JSP que muestra las recomendaciones de libros, con base en cookies. (Parte 2 de 2).

`Recomendaciones.jsp` contiene un componente **Etiqueta** (líneas 21 a 24), un **List box** (líneas 25 a 29) y un **Hipervínculo** (líneas 30 a 33). La **Etiqueta** muestra el texto **Recomendaciones** en la parte superior de la página. Un componente **List box** muestra una lista de opciones, de las que el usuario puede seleccionar varias. El **List box** en este ejemplo muestra las recomendaciones creadas por el bean de página `Recomendaciones.java` (figura 26.24), o el texto "No hay recomendaciones. Por favor seleccione un lenguaje". El **Hipervínculo** permite al usuario regresar a `Opciones.jsp` para seleccionar más lenguajes.

Bean de página que crea las recomendaciones de libros a partir de cookies

En `Recomendaciones.java` (figura 26.24), el método `prerender` (líneas 192 a 223) obtiene las cookies del cliente, usando el método `getCookies` del objeto petición (líneas 195 a 197). Un objeto de la clase `HttpServletRequest` (del paquete `javax.servlet.http`) representa la petición. Este objeto puede obtenerse invocando al método `getExternalContext` en el bean de página, y después invocando a `getRequest` en el objeto resultante. La llamada a `getCookies` devuelve un arreglo de las cookies que se habían escrito antes en el cliente. Una aplicación puede leer las cookies sólo si un servidor las creó en el dominio en el que se ejecuta la aplicación; un servidor Web no puede acceder a las cookies creadas por los servidores en otros dominios. Por ejemplo, una cookie creada por un servidor Web en el dominio `deitel.com` no puede ser leída por un servidor Web de cualquier otro dominio.

En la línea 203 se determina si por lo menos existe una cookie. (Ignoramos la primera cookie en el arreglo, ya que contienen información que no es específica para nuestra aplicación). En las líneas 205 a 213 se agrega la información en la(s) cookie(s) a un arreglo de tipo `Option`. Los arreglos de objetos `Option` pueden mostrarse como una lista de elementos en un componente **List box**. El ciclo obtiene el nombre y el valor de cada cookie, usando la variable de control para determinar el valor actual en el arreglo de cookies. Si no se seleccionó un lenguaje, en las líneas 215 a 220 se agrega un mensaje a un arreglo de tipo `Option`, el cual pide al usuario que seleccione un lenguaje. En la línea 222 se establece `cuadroListaLibros` para mostrar el arreglo de tipo `Option` resultante. En la figura 26.25 sintetizamos los métodos de `Cookie` de uso común.

```

1 // Fig. 26.24: Recomendaciones.java
2 // Bean de pagina que muestra las recomendaciones de libros con base en cookies
3 // que almacenan los lenguajes de programación seleccionados por el usuario.
4 package cookies;
5
6 import com.sun.rave.web.ui.appbase.AbstractPageBean;
7 import com.sun.rave.web.ui.component.Body;
8 import com.sun.rave.web.ui.component.Form;
9 import com.sun.rave.web.ui.component.Head;
10 import com.sun.rave.web.ui.component.Html;
11 import com.sun.rave.web.ui.component.Link;
12 import com.sun.rave.web.ui.component.Page;
13 import javax.faces.FacesException;
14 import com.sun.rave.web.ui.component.Listbox;
15 import com.sun.rave.web.ui.model.DefaultOptionsList;
16 import com.sun.rave.web.ui.component.Label;
17 import com.sun.rave.web.ui.component.Hyperlink;
18 import com.sun.rave.web.ui.model.Option;
19 import javax.servlet.http.HttpServletRequest;
20 import javax.servlet.http.Cookie;
21 import com.sun.rave.web.ui.component.HiddenField;
22
23 public class Recomendaciones extends AbstractPageBean
24 {
25     private int __placeholder;
26
27     private void _init() throws Exception
28     {
29         // cuerpo vacío
30     } // fin del método _init()
31
32     // Para ahorrar espacio, omitimos el código en las líneas 32 a 151. El código
33     // fuente completo se proporciona con los ejemplos de este capítulo.
34
35     public Recomendaciones()
36     {
37         // constructor vacío
38     } // fin del constructor
39
40     protected ApplicationBean getApplicationBean()
41     {
42         return (ApplicationBean) getBean( "ApplicationBean" );
43     } // fin del método getApplicationBean
44
45     protected RequestBean getRequestBean()
46     {
47         return (RequestBean) getBean( "RequestBean" );
48     } // fin del método getRequestBean
49
50     protected SessionBean getSessionBean()
51     {
52         return (SessionBean) getBean( "SessionBean" );
53     } // fin del método getSessionBean
54
55     public void init()
56     {
57         super.init();
58         try

```

Figura 26.24 | Bean de página que muestra las recomendaciones de libros con base en cookies que almacenan los lenguajes de programación seleccionados por el usuario. (Parte I de 2).


```

176     {
177         _init();
178     } // fin de try
179     catch ( Exception e )
180     {
181         log( "Error al inicializar Recomendaciones", e );
182         throw e instanceof FacesException ? (FacesException) e :
183             new FacesException( e );
184     } // fin de catch
185 } // fin del método init
186
187 public void preprocess()
188 {
189     // cuerpo vacío
190 } // fin del método preprocess
191
192 public void prerender()
193 {
194     // obtiene las cookies del cliente
195     HttpServletRequest peticion =
196         (HttpServletRequest)getExternalContext().getRequest();
197     Cookie [] cookies = peticion.getCookies();
198
199     // si hay cookies, almacena los correspondientes libros y
200     // números ISBN en un arreglo de Opciones
201     Option [] recomendaciones;
202
203     if ( cookies.length > 1 )
204     {
205         recomendaciones = new Option[ cookies.length - 1 ];
206         for ( int i = 0; i < cookies.length - 1; i++ )
207         {
208             String lenguaje =
209                 cookies[i].getName().replace( '/', ' ' );
210             recomendaciones[ i ] = new Option( lenguaje +
211                 "How to Program. ISBN#: " + cookies[i].getValue() );
212         } // fin de for
213     } // fin de if
214
215     // en caso contrario, almacena un mensaje indicando que no se seleccionó un
216     lenguaje
217     else
218     {
219         recomendaciones = new Option[ 1 ];
220         recomendaciones[ 0 ] = new Option(
221             "No hay recomendaciones. " + "Seleccione un lenguaje." );
222     } // fin de else
223
224     cuadroListaLibros.setItems(recomendaciones);
225 } // fin del método prerender
226
227 public void destroy()
228 {
229     // cuerpo vacío
230 } // fin del método destroy
231 } // fin de la clase Recomendaciones

```

Figura 26.24 | Bean de página que muestra las recomendaciones de libros con base en cookies que almacenan los lenguajes de programación seleccionados por el usuario. (Parte 2 de 2).

Métodos	Descripción
<code>getDomain</code>	Devuelve un objeto <code>String</code> que contiene el dominio de la cookie (es decir, el dominio desde el que se escribió la cookie). Esto determina cuáles servidores Web pueden recibir la cookie. De manera predeterminada, las cookies se envían al servidor Web que envió originalmente la cookie al cliente. Si se modifica la propiedad <code>Domain</code> , la cookie se devolverá a un servidor Web distinto del que la escribió originalmente.
<code>getMaxAge</code>	Devuelve un valor <code>int</code> , el cual indica cuántos segundos persistirá la cookie en el navegador. El valor predeterminado es <code>-1</code> , lo cual significa que la cookie persistirá hasta que el navegador se cierre.
<code>getName</code>	Devuelve un objeto <code>String</code> que contiene el nombre de la cookie.
<code>getPath</code>	Devuelve un objeto <code>String</code> que contiene la ruta a un directorio en el servidor, en el cual se aplica la cookie. Las cookies pueden “dirigirse” a directorios específicos en el servidor Web. De manera predeterminada, una cookie se devuelve sólo a las aplicaciones que operan en el mismo directorio que la aplicación que envió la cookie, o en un subdirectorio de ese directorio. Si se modifica la propiedad <code>Path</code> , la cookie se devolverá a un directorio distinto al directorio en el que se escribió originalmente.
<code>getSecure</code>	Devuelve un valor <code>boolean</code> que indica si la cookie debe transmitirse a través de un protocolo seguro. El valor <code>true</code> hace que se utilice un protocolo seguro.
<code>getValue</code>	Devuelve un objeto <code>String</code> que contiene el valor de la cookie.

Figura 26.25 | Métodos de `javax.servlet.http.Cookie`.

26.7.2 Rastreo de sesiones con el objeto `SessionBean`

También podemos realizar el rastreo de sesiones mediante la clase `SessionBean` que se proporciona en cada una de las aplicaciones Web creadas con Java Studio Creator 2. Cuando se solicita una página Web que está dentro del proyecto, se crea un objeto `SessionBean`. Se puede acceder a las propiedades de este objeto durante una sesión con el navegador, mediante la invocación del método `getSessionBean` en el bean de página. Para demostrar las técnicas de rastreo de sesiones usando un objeto `SessionBean`, modificamos los archivos de bean de página de las figuras 26.22 y 26.24, de manera que utilicen el objeto `SessionBean` para almacenar los lenguajes seleccionados por el usuario. Empezamos con el archivo `Opciones.jsp` actualizado (figura 26.26). La figura 26.29 presenta el archivo `SessionBean.java` y la figura 26.30 presenta el archivo de bean de página modificado para `Opciones.jsp`.

El archivo `Opciones.jsp` de la figura 26.26 es similar al que se presenta en la figura 26.20 para el ejemplo de las cookies. En las líneas 38 a 45 se definen dos elementos `ui:staticText` que no se presentaron en el ejemplo de las cookies. El primer elemento muestra el texto “Número de selecciones hasta ahora:”. El atributo `text` del segundo elemento está enlazado a la propiedad `numSelecciones` en el objeto `SessionBean` (líneas 44 y 45). En un momento hablaremos acerca de cómo enlazar el atributo `text` a una propiedad de `SessionBean`.

```

1  <?xml version = "1.0" encoding = "UTF-8"?>
2
3  <!-- Fig. 26.26: Opciones.jsp -->
4  <!-- Archivo JSP que permite al usuario seleccionar un lenguaje de programación. -->
5  <jsp:root version = "1.2" xmlns:f = "http://java.sun.com/jsp/core"
6      xmlns:h = "http://java.sun.com/jsp/html" xmlns:jsp =

```

Figura 26.26 | Archivo JSP que permite al usuario seleccionar un lenguaje de programación. (Parte I de 4).

```

7      "http://java.sun.com/JSP/Page" xmlns:ui = "http://www.sun.com/web/ui">
8      <jsp:directive.page contentType = "text/html; charset = UTF-8"
9          pageEncoding = "UTF-8"/>
10     <f:view>
11         <ui:page binding = "#{Opciones.page}" id = "page">
12             <ui:html binding = "#{Opciones.html}" id = "html">
13                 <ui:head binding = "#{Opciones.head}" id = "head">
14                     <ui:link binding = "#{Opciones.link}" id = "link"
15                         url = "/resources/stylesheet.css"/>
16                 </ui:head>
17                 <ui:body binding = "#{Opciones.body}" id = "body"
18                     style = "-rave-layout: grid">
19                     <ui:form binding = "#{Opciones.form}" id = "form">
20                         <ui:label binding = "#{Opciones.etiquetaLenguaje}" for =
21                             "listaLenguajes" id = "etiquetaLenguaje" style =
22                             "font-size: 16px; font-weight: bold; left: 24px; top:
23                             24px; position: absolute" text = "Seleccione un
24                             lenguaje de programación:"/>
25                         <ui:radioButtonGroup binding = "#{Opciones.listaLenguajes}"
26                             id = "listaLenguajes" items =
27                             "#{Opciones.listaLenguajesDefaultOptions.options}" style =
28                             "left: 24px; top: 48px; position: absolute"/>
29                         <ui:button action = "#{Opciones.enviar_action}" binding =
30                             "#{Opciones.enviar}" id = "enviar" style = "left:
31                             23px; top: 192px; position: absolute" text =
32                             "Enviar"/>
33                         <ui:staticText binding = "#{Opciones.etiquetaRespuesta}" id =
34                             "etiquetaRespuesta" rendered = "false" style =
35                             "font-size: 16px; font-weight: bold; height: 24px;
36                             left: 24px; top: 24px; position: absolute;
37                             width: 216px"/>
38                         <ui:staticText binding = "#{Opciones.etiquetaNumSelec}"
39                             id = "etiquetaNumSelec" rendered = "false" style =
40                             "left: 24px; top: 96px; position: absolute" text =
41                             "Número de selecciones hasta ahora:"/>
42                         <ui:staticText binding = "#{Opciones.numSelec}" id =
43                             "numSelec" rendered = "false" style = "left:
44                             240px; top: 96px; position: absolute" text =
45                             "#{SessionBean1.numSelecciones}"/>
46                         <ui:hyperlink action = "case1" binding = "#{Opciones.
47                             vinculoRecomendaciones}"
48                             id = "vinculoRecomendaciones" rendered = "false" style = "left:
49                             24px; top: 168px; position: absolute" text =
50                             "Haga clic aquí para obtener recomendaciones de libros." url =
51                             "/faces/Recomendaciones.jsp"/>
52                         <ui:hyperlink action = "#{Opciones.vinculoLenguajes1_action}"
53                             binding = "#{Opciones.vinculoLenguajes1}" id =
54                             "vinculoLenguajes1" rendered = "false" style = "left:
55                             24px; top: 144px; position: absolute" text = "Haga clic
56                             aquí para seleccionar otro lenguaje."/>
57                     </ui:form>
58                 </ui:body>
59             </ui:html>
60         </ui:page>
61     </f:view>
62 </jsp:root>

```

Figura 26.26 | Archivo JSP que permite al usuario seleccionar un lenguaje de programación. (Parte 2 de 4).

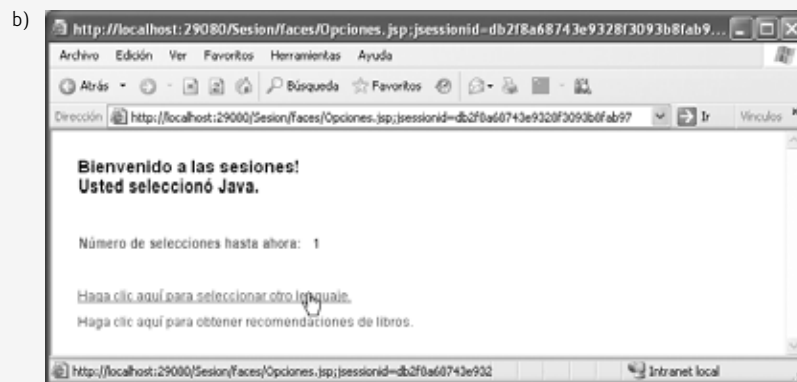


Figura 26.26 | Archivo JSP que permite al usuario seleccionar un lenguaje de programación. (Parte 3 de 4).



Figura 26.26 | Archivo JSP que permite al usuario seleccionar un lenguaje de programación. (Parte 4 de 4).

Cómo agregar propiedades al objeto `SessionBean`

En este ejemplo, utilizamos el rastreo de sesiones para almacenar no sólo los lenguajes seleccionados por el usuario, sino también el número de selecciones realizadas. Para almacenar esta información en el objeto `SessionBean`, agregamos propiedades a la clase `SessionBean`.

Para agregar una propiedad que almacene el número de selecciones hasta ahora, haga clic con el botón derecho del ratón en el nodo `SessionBean1` de la ventana **Esquema** y seleccione **Agregar | Propiedad** para que aparezca el cuadro de diálogo **Nuevo patrón de propiedad** (figura 26.27). Este cuadro de diálogo nos permite agregar propiedades primitivas, `String` o de envoltura de tipo primitivo al objeto `SessionBean1`. Agregue una propiedad `int` llamada `numSelecciones` y haga clic en **Aceptar** para aceptar las opciones predeterminadas para esta propiedad. Abra el archivo `SessionBean1` y verá una nueva definición de propiedad, un método `get` y un método `set` para `numSelecciones`.

La propiedad `numSelecciones` se manipulará en el archivo de bean de página para almacenar el número de lenguajes que seleccionó el usuario. Para mostrar el valor de esta propiedad en el elemento **Texto estático** llamado `numSelec` en el archivo JSP, haga clic con el botón derecho en el componente **Texto estático** en la ventana **Esquema** en modo **Diseño**, y seleccione **Enlazar con datos....** En el cuadro de diálogo **Enlazar con datos** (figura 26.28), seleccione la ficha **Enlazar con un objeto**, localice la propiedad `numSelecciones` bajo el nodo `SessionBean1` y haga clic en **Aceptar**. Ahora, el elemento **Texto estático** mostrará el valor de la propiedad `numSelecciones` de `SessionBean1`. Si cambia el valor de la propiedad el texto también cambia, de manera que no es necesario establecer el texto en el bean de página mediante programación.



Figura 26.27 | Cuadro de diálogo **Nuevo patrón de propiedad** para agregar una propiedad al objeto `SessionBean`.

Ahora que hemos agregado una propiedad para almacenar el número de selecciones en el objeto `SessionBean`, debemos agregar una segunda propiedad para almacenar las selecciones en sí. Nos gustaría almacenar las selecciones con pares clave-valor del lenguaje seleccionado y el número ISBN de un libro relacionado, algo similar a la forma en que se almacenaron las selecciones mediante el uso de cookies. Para hacer esto, agregamos un objeto `Properties` llamado `lenguajesSeleccionados` al objeto `SessionBean`. Agregamos en forma manual esta propiedad al archivo `SessionBean`, pero podemos agregarla usando el cuadro de diálogo **Nuevo patrón de propiedad** en la figura 26.27. Simplemente escriba `java.util.Properties` en el campo del cuadro de lista desplegable **Tipo**, configure la propiedad y haga clic en **Aceptar**. El archivo `SessionBean` final, después de haber agregado las dos propiedades, se muestra en la figura 26.29.

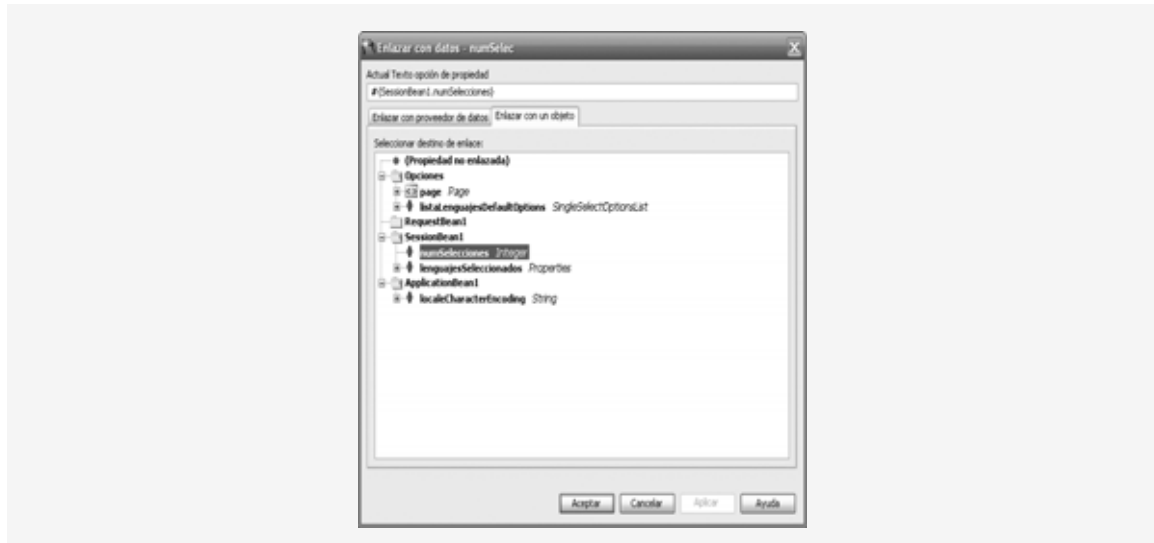


Figura 26.28 | Cuadro de diálogo **Enlazar con datos**.

```

1 // Fig. 26.29: SessionBean1.java
2 // Archivo SessionBean1 para almacenar las selecciones de lenguajes.
3 package session;
4
5 import com.sun.rave.web.ui.appbase.AbstractSessionBean;
6 import java.util.Properties;
7 import javax.faces.FacesException;
8
9 public class SessionBean1 extends AbstractSessionBean
10 {
11     private int __placeholder;
12
13     private void _init() throws Exception
14     {
15         // cuerpo vacío
16     } // fin del método _init
17
18     public SessionBean1()
19     {
20         // constructor vacío
21     } // fin del constructor
22
23     protected ApplicationBean1 getApplicationBean1()

```

Figura 26.29 | Archivo `SessionBean1` para almacenar las selecciones de lenguajes. (Parte I de 2).

```

24     {
25         return (ApplicationBean1) getBean( "ApplicationBean1" );
26     } // fin del método getApplicationBean1
27
28     public void init()
29     {
30         super.init();
31         try
32         {
33             _init();
34         } // fin de try
35         catch ( Exception e )
36         {
37             log( "Error al inicializar SessionBean1", e);
38             throw e instanceof FacesException ? (FacesException) e:
39                 new FacesException( e );
40         } // fin de catch
41     } // fin del método init
42
43     public void passivate()
44     {
45         // cuerpo vacío
46     } // fin del método passivate
47
48     public void activate()
49     {
50         // cuerpo vacío
51     } // fin del método activate
52
53     public void destroy()
54     {
55         // cuerpo vacío
56     } // fin del método destroy
57
58     private int numSelecciones = 0; // almacena el número de selecciones únicas
59
60     public int getNumSelecciones()
61     {
62         return this.numSelecciones;
63     } // fin del método getNumSelecciones
64
65     public void setNumSelecciones( int numSelecciones )
66     {
67         this.numSelecciones = numSelecciones;
68     } // fin del método setNumSelecciones
69
70     // Almacena pares clave-valor de lenguajes seleccionados
71     private Properties lenguajesSeleccionados = new Properties();
72
73     public Properties getLenguajesSeleccionados()
74     {
75         return this.lenguajesSeleccionados;
76     } // fin del método getLenguajesSeleccionados
77
78     public void setLenguajesSeleccionados( Properties lenguajesSeleccionados )
79     {
80         this.lenguajesSeleccionados = lenguajesSeleccionados;
81     } // fin del método setLenguajesSeleccionados
82 } // fin de la clase SessionBean1

```

Figura 26.29 | Archivo SessionBean1 para almacenar los lenguajes seleccionados. (Parte 2 de 2).

En la línea 58 se declara la propiedad `numSelecciones`, y en las líneas 60 a 63 y 65 a 68 se declaran sus métodos *obtener* (*get*) y *establecer* (*set*), respectivamente. Esta parte del código se generó de manera automática cuando utilizamos el cuadro de diálogo **Nuevo patrón de propiedad**. En la línea 71 se define el objeto `Properties` llamado `lenguajesSeleccionados` que almacenará las selecciones del usuario. En las líneas 73 a 76 y 78 a 81 están los métodos *get* y *set* para esta propiedad.

Manipulación de las propiedades de SessionBean en un archivo de bean de página

El archivo de bean de página para la página `Opciones.jsp` se muestra en la figura 26.30. Debido a que gran parte de este ejemplo es idéntica al anterior, nos concentraremos en las nuevas características.

```

1 // Fig. 26.30: Opciones.java
2 // Bean de página que almacena las selecciones de lenguajes en una propiedad SessionBean.
3 package session;
4
5 import com.sun.rave.web.ui.appbase.AbstractPageBean;
6 import com.sun.rave.web.ui.component.Body;
7 import com.sun.rave.web.ui.component.Form;
8 import com.sun.rave.web.ui.component.Head;
9 import com.sun.rave.web.ui.component.Html;
10 import com.sun.rave.web.ui.component.Link;
11 import com.sun.rave.web.ui.component.Page;
12 import javax.faces.FacesException;
13 import com.sun.rave.web.ui.component.RadioButtonGroup;
14 import com.sun.rave.web.ui.component.Hyperlink;
15 import com.sun.rave.web.ui.component.Button;
16 import com.sun.rave.web.ui.component.Label;
17 import com.sun.rave.web.ui.component.StaticText;
18 import com.sun.rave.web.ui.model.SingleSelectOptionsList;
19 import java.util.Properties;
20 import javax.servlet.http.Cookie;
21 import javax.servlet.http.HttpServletRequest;
22 import javax.servlet.http.HttpSession;
23
24 public class Opciones extends AbstractPageBean
25 {
26     private int __placeholder;
27
28     private void _init() throws Exception
29     {
30         listaLenguajesDefaultOptions.setOptions(
31             new com.sun.rave.web.ui.model.Option[]
32             {
33                 new com.sun.rave.web.ui.model.Option( "Java", "Java" ),
34                 new com.sun.rave.web.ui.model.Option( "C", "C" ),
35                 new com.sun.rave.web.ui.model.Option( "C++", "C++" ),
36                 new com.sun.rave.web.ui.model.Option( "Visual Basic 2005",
37                     "Visual Basic 2005" ),
38                 new com.sun.rave.web.ui.model.Option( "Visual C# 2005",
39                     "Visual C# 2005" )
40             }
41         );
42     } // fin del método init
43
44     // para ahorrar espacio, omitimos el código de las líneas 44 a 219. El código
45     // fuente completo se proporciona con los ejemplos de este capítulo.

```

Figura 26.30 | Bean de página que almacena las selecciones de lenguajes en una propiedad `SessionBean`. (Parte I de 3).


```

46
220 private Properties libros = new Properties();
221
222 public Opciones()
223 {
224     // inicializa el objeto Properties de valores que se van a almacenar en
225     // el bean de sesión.
226     libros.setProperty( "Java", "0-13-222220-5" );
227     libros.setProperty( "C", "0-13-142644-3" );
228     libros.setProperty( "C++", "0-13-185757-6" );
229     libros.setProperty( "Visual Basic 2005", "0-13-186900-0" );
230     libros.setProperty( "Visual C# 2005", "0-13-152523-9" );
231 } // fin del constructor
232
233 protected ApplicationBean1 getApplicationBean1()
234 {
235     return (ApplicationBean1) getBean( "ApplicationBean1" );
236 } // fin del método getApplicationBean1
237
238 protected RequestBean1 getRequestBean1()
239 {
240     return (RequestBean1) getBean( "RequestBean1" );
241 } // fin del método getRequestBean1
242
243 protected SessionBean1 getSessionBean1()
244 {
245     return (SessionBean1) getBean( "SessionBean1" );
246 } // fin del método getSessionBean1
247
248 public void init()
249 {
250     super.init();
251     try
252     {
253         _init();
254     } // fin de try
255     catch ( Exception e )
256     {
257         log( "Error al inicializar Opciones", e );
258         throw e instanceof FacesException ? (FacesException) e :
259             new FacesException( e );
260     } // fin de catch
261 } // fin del método init
262
263 public void preprocess()
264 {
265     // cuerpo vacío
266 } // fin del método preprocess
267
268 public void prerender()
269 {
270     // cuerpo vacío
271 } // fin del método prerender
272
273 public void destroy()
274 {
275     // cuerpo vacío
276 } // fin del método destroy

```

Figura 26.30 | Bean de página que almacena las selecciones de lenguajes en una propiedad SessionBean. (Parte 2 de 3).

```

277
278 // manejador de acciones para el botón enviar, almacena los lenguajes seleccionados
279 // en ámbito de sesión para obtenerlos al hacer recomendaciones de libros.
280 public String enviar_action()
281 {
282     String msg = "Bienvenido a las sesiones! Usted ";
283
284     // si el usuario hizo una selección
285     if ( getListaLenguajes().getSelected() != null )
286     {
287         String lenguaje = listaLenguajes.getSelected().toString();
288         msg += "selecciono " + lenguaje + ".";
289
290         // obtiene el número ISBN del libro para el lenguaje dado.
291         String ISBN = libros.getProperty( lenguaje );
292
293         // agrega la selección al objeto Properties de SessionBean1
294         Properties selections = getSessionBean1().getLenguajesSeleccionados();
295         Object resultado = selections.setProperty( lenguaje, ISBN );
296
297         // incrementa numSelecciones en el objeto SessionBean1 y actualiza
298         // lenguajesSeleccionados si el usuario no ha realizado esta selección
299         // antes
300         if ( resultado == null )
301         {
302             int numSelec = getSessionBean1().getNumSelecciones();
303             getSessionBean1().setNumSelecciones(++numSelec);
304         } // fin de if
305     } // fin de if
306     else
307         msg += "no selecciono un lenguaje.";
308
309     etiquetaRespuesta.setValue( msg );
310     listaLenguajes.setRendered( false );
311     etiquetaLenguaje.setRendered( false );
312     enviar.setRendered( false );
313     etiquetaRespuesta.setRendered( true );
314     etiquetaNumSelec.setRendered( true );
315     numSelec.setRendered( true );
316     vinculoLenguajes.setRendered( true );
317     vinculoRecomendaciones.setRendered( true );
318     return null;
319 } // fin del método enviar_action
320
321 // vuelve a mostrar los componentes usados para permitir al usuario
322 // seleccionar un lenguaje.
323 public String vinculoLenguajes1_action() {
324     etiquetaRespuesta.setRendered( false );
325     etiquetaNumSelec.setRendered( false );
326     numSelec.setRendered( false );
327     vinculoLenguajes.setRendered( false );
328     vinculoRecomendaciones.setRendered( false );
329     listaLenguajes.setRendered( true );
330     etiquetaLenguaje.setRendered( true );
331     enviar.setRendered( true );
332     return null;
333 } // fin del método vinculoLenguajes1_action
334 } // fin de la clase Opciones

```

Figura 26.30 | Bean de página que almacena las selecciones de lenguajes en una propiedad SessionBean. (Parte 3 de 3).

El manejador de acciones del componente Botón llamado `enviar` (líneas 280 a 319) almacena las selecciones del usuario en el objeto `SessionBean1` e incrementa el número de selecciones realizadas, si es necesario. En la línea 294 se obtiene el objeto `Properties` del objeto `SessionBean1` que contiene las selecciones del usuario. En la línea 295 se agrega la selección actual al objeto `Properties`. El método `setProperty` devuelve el valor previamente asociado con la nueva clave, o `null` si esta clave no se había almacenado ya en el objeto `Properties`. Si al agregar la nueva propiedad se devuelve `null`, entonces el usuario ha realizado una nueva selección. En este caso, en las líneas 302 y 303 se incrementa la propiedad `numSelecciones` en el objeto `SessionBean1`. En las líneas 309 a 317 y en el manejador de acciones `vinculoLenguajes1` (líneas 323 a 334) se controlan los componentes que se mostrarán en la página, igual que en los ejemplos de cookies.



Observación de ingeniería de software 26.2

Un beneficio de usar las propiedades de `SessionBean` (en vez de cookies) es que pueden almacenar cualquier tipo de objeto (no sólo objetos `String`) como valores de atributos. Esto nos proporciona más flexibilidad para mantener la información de estado del cliente.

Cómo mostrar las recomendaciones con base en los valores de sesiones

Al igual que en el ejemplo de las cookies, esta aplicación proporciona un vínculo a `Recomendaciones.jsp` (figura 26.31), el cual muestra una lista de recomendaciones de libros con base en los lenguajes seleccionados por el usuario. Es idéntico al archivo `Recomendaciones.jsp` del ejemplo de las cookies (figura 26.23).

```

1  <?xml version = "1.0" encoding = "UTF-8"?>
2
3  <!-- Fig. 26.31: Recomendaciones.jsp -->
4  <!-- Archivo JSP que muestra las recomendaciones de libros con base en los -->
5  <!-- lenguajes seleccionados que se almacenan en el ámbito de sesión. -->
6  <jsp:root version = "1.2" xmlns:f = "http://java.sun.com/jsf/core"
7  xmlns:h = "http://java.sun.com/jsf/html" xmlns:jsp =
8  "http://java.sun.com/JSP/Page" xmlns:ui = "http://www.sun.com/web/ui">
9  <jsp:directive.page contentType = "text/html; charset = UTF-8"
10  pageEncoding = "UTF-8"/>
11  <f:view>
12  <ui:page binding = "#{Recomendaciones.page}" id = "page">
13  <ui:html binding = "#{Recomendaciones.html}" id = "html">
14  <ui:head binding = "#{Recomendaciones.head}" id = "head">
15  <ui:link binding = "#{Recomendaciones.link}" id = "link"
16  url = "/resources/stylesheet.css"/>
17  </ui:head>
18  <ui:body binding = "#{Recomendaciones.body}" id = "body"
19  style = "-rave-layout: grid">
20  <ui:form binding = "#{Recomendaciones.form}" id = "form">
21  <ui:label binding = "#{Recomendaciones.etiquetaLenguaje}"
22  for = "cuadroListaLibros" id = "etiquetaLenguaje" style =
23  "font-size: 20px; font-weight: bold; left: 24px; top:
24  24px; position: absolute" text = "Recomendaciones"/>
25  <ui:listbox binding = "#{Recomendaciones.cuadroListaLibros}"
26  id = "cuadroListaLibros" items = "#{Recomendaciones.
27  cuadroListaLibrosDefaultOptions.options}" rows = "6"
28  style= "left: 24px; top: 72px; position: absolute;
29  width: 360px"/>
30  <ui:hyperlink action = "case1" binding =
31  "#{Recomendaciones.vinculoOpciones}" id = "vinculoOpciones"
32  style = "left: 24px; top: 192px; position: absolute"
33  text = "Haga clic aquí para seleccionar otro lenguaje."/>

```

Figura 26.31 | Archivo JSP que muestra las recomendaciones de libros con base en los lenguajes seleccionados que se almacenan en el ámbito de sesión. (Parte I de 2).

```

34         </ui:form>
35     </ui:body>
36 </ui:html>
37 </ui:page>
38 </f:view>
39 </jsp:root>

```

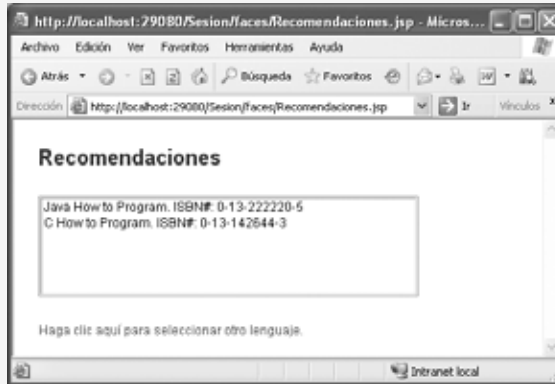


Figura 26.31 | Archivo JSP que muestra las recomendaciones de libros con base en los lenguajes seleccionados que se almacenan en el ámbito de sesión. (Parte 2 de 2).

*Bean de página que crea recomendaciones de libros a partir de una propiedad de **SessionBean***

La figura 26.32 presenta el bean de página para Recomendaciones.jsp. De nuevo, gran parte de este código es similar al bean de página utilizado en el ejemplo de las cookies. Sólo hablaremos de las nuevas características.

```

1  // Fig. 26.32: Recomendaciones.java
2  // Bean de página que muestra las recomendaciones de libros con base en
3  // una propiedad de un objeto SessionBean.
4  package sesion;
5
6  import com.sun.rave.web.ui.appbase.AbstractPageBean;
7  import com.sun.rave.web.ui.component.Body;
8  import com.sun.rave.web.ui.component.Form;
9  import com.sun.rave.web.ui.component.Head;
10 import com.sun.rave.web.ui.component.Html;
11 import com.sun.rave.web.ui.component.Link;
12 import com.sun.rave.web.ui.component.Page;
13 import javax.faces.FacesException;
14 import com.sun.rave.web.ui.component.Listbox;
15 import com.sun.rave.web.ui.component.Label;
16 import com.sun.rave.web.ui.component.Hyperlink;
17 import com.sun.rave.web.ui.model.DefaultOptionsList;
18 import java.util.Enumeration;
19 import com.sun.rave.web.ui.model.Option;
20 import java.util.Properties;
21
22 public class Recomendaciones extends AbstractPageBean
23 {
24     private int __placeholder;

```

Figura 26.32 | Bean de página que muestra las recomendaciones de libros con base en una propiedad de un objeto SessionBean. (Parte I de 3).

```

25
26 private void _init() throws Exception
27 {
28     // cuerpo vacío
29 } // fin del método _init
30
31 // para ahorrar espacio, omitimos el código de las líneas 31 a 150. El código
32 // fuente completo se proporciona con los ejemplos de este capítulo.
33
151 public Recomendaciones()
152 {
153     // constructor vacío
154 } // fin del constructor
155
156 protected RequestBean1 getRequestBean1()
157 {
158     return (RequestBean1) getBean( "RequestBean1" );
159 } // fin del método getRequestBean1
160
161 protected ApplicationBean1 getApplicationBean1()
162 {
163     return (ApplicationBean1) getBean( "ApplicationBean1" );
164 } // fin del método getApplicationBean1
165
166 protected SessionBean1 getSessionBean1()
167 {
168     return (SessionBean1) getBean( "SessionBean1" );
169 } // fin del método getSessionBean1
170
171 public void init()
172 {
173     super.init();
174     try
175     {
176         _init();
177     } // fin de try
178     catch ( Exception e )
179     {
180         log( "Error al inicializar Recomendaciones", e );
181         throw e instanceof FacesException ? (FacesException) e :
182             new FacesException( e );
183     } // fin de catch
184 } // fin del método init
185
186 public void preprocess()
187 {
188     // cuerpo vacío
189 } // fin del método preprocess
190
191 public void prerender()
192 {
193     // obtiene las selecciones del usuario y el número de selecciones realizadas
194     Properties lenguajes = getSessionBean1().getLenguajesSeleccionados();
195     Enumeration enumSelecciones = lenguajes.propertyNames();
196     int numSeleccionados = getSessionBean1().getNumSelecciones();
197
198     Option [] recomendaciones;
199

```

Figura 26.32 | Bean de página que muestra las recomendaciones de libros con base en una propiedad de un objeto SessionBean. (Parte 2 de 3).

```

200 // si por lo menos se hizo una selección
201 if ( numSeleccionados > 0 )
202 {
203     recomendaciones = new Option[ numSeleccionados ];
204
205     for( int i = 0; i < numSeleccionados; i++ )
206     {
207         String lenguaje = (String) enumSelecciones.nextElement() ;
208         recomendaciones[ i ] = new Option( lenguaje +
209             " How to Program.      ISBN#: " +
210             lenguajes.getProperty( lenguaje ) );
211     } // fin de for
212 } // fin de if
213 else
214 {
215     recomendaciones = new Option[ 1 ];
216     recomendaciones[ 0 ] = new Option( "No hay recomendaciones.
217         Seleccione un lenguaje." );
218 } // fin de else
219
220 cuadroListaLibros.setItems(recomendaciones);
221 } // fin del método prerender
222
223 public void destroy()
224 {
225     // cuerpo vacío
226 } // fin del método destroy
227 } // fin de la clase Recomendaciones

```

Figura 26.32 | Bean de página que muestra las recomendaciones de libros con base en una propiedad de un objeto SessionBean. (Parte 3 de 3).

En la línea 194 se obtiene el objeto `Properties` que contiene las selecciones que hizo el usuario del objeto `SessionBean1`, y en la línea 195 se obtiene una enumeración de todas las claves en ese objeto `Properties`. En la línea 196 se obtiene el número de selecciones que se realizaron a partir del objeto `SessionBean1`. Si se hizo alguna selección, en la línea 208 se construye un arreglo `Option` de tamaño apropiado para mostrar las selecciones en el elemento `ui:listBox` de `Recomendaciones.jsp`. En las líneas 205 a 211 se agrega cada una de las selecciones del usuario a este arreglo `Option`. En la línea 207 se obtiene la siguiente clave de la enumeración de teclas, y en las líneas 208 a 210 se agrega una recomendación al arreglo `Option`.

26.8 Conclusión

En este capítulo presentamos el desarrollo de aplicaciones Web mediante el uso de JavaServer Pages y JavaServer Faces en Java Studio Creator 2. Empezamos hablando sobre las transacciones HTTP simples que se llevan a cabo al solicitar y recibir una página Web a través de un navegador Web. Después hablamos sobre los tres niveles (es decir, el nivel cliente o superior, el nivel de lógica comercial o intermedio y el nivel de información o inferior) que conforman a la mayoría de las aplicaciones Web.

Aprendió acerca de la función que desempeñan los archivos JSP y los archivos de bean de página, y la relación entre ellos. Aprendió a utilizar Java Studio Creator 2 para compilar y ejecutar aplicaciones Web. También aprendió a crear aplicaciones Web en forma visual, mediante el uso de las herramientas de arrastrar y soltar de Java Studio Creator 2.

Demostramos varios componentes JSF comunes para mostrar texto e imágenes en las páginas Web. También hablamos sobre los componentes de validación y los métodos de validación personalizados, los cuales nos permiten asegurar que la entrada del usuario cumpla con ciertos requerimientos.

Hablamos sobre los beneficios de mantener la información del usuario a través de varias páginas de un sitio Web. Después demostramos cómo se pueden incluir dichas funcionalidades en una aplicación Web, mediante el

uso de cookies o propiedades en la clase `SessionBean`. En el siguiente capítulo continuaremos nuestra discusión acerca del desarrollo de aplicaciones Web. Aprenderá a utilizar una base de datos desde una aplicación Web, a utilizar varios de los componentes JSF habilitados para AJAX de la biblioteca Java Blueprints de Sun, y a utilizar los formularios virtuales.

26.9 Recursos Web

developers.sun.com/prodtech/javatools/jscreator

Presenta las generalidades acerca de Java Studio Creator 2 e incluye artículos, foros, demostraciones de productos y vínculos a recursos útiles, relevantes para la construcción de aplicaciones Web en Java Studio Creator 2.

developers.sun.com/prodtech/javatools/jscreator/index.jsp

El centro de Java Studio Creator de Sun, tiene todo lo que el programador necesita para empezar a trabajar. Descargue el IDE sin costo y dé un vistazo a la ficha de aprendizaje (Learning) para los tutoriales de Java Studio Creator.

developers.sun.com/prodtech/javatools/jscreator/learning/tutorials/index.jsp

Proporciona docenas de tutoriales, desde tips acerca de cómo empezar a trabajar con Java Studio Creator 2, hasta instrucciones de características específicas acerca de cómo utilizar muchas facetas del IDE.

developers.sun.com/prodtech/javatools/jscreator/reference/docs/apis/

La documentación para Java Studio Creator 2.

java.sun.com/javaee/jaserverfaces/

Este sitio oficial de Sun proporciona la documentación para JavaServer Faces, junto con vínculos hacia artículos y tutoriales relacionados.

www.netbeans.org/products/visualweb/

Aquí puede obtener el Netbeans Visual Web Pack 5.5 para Netbeans 5.5.

java.sun.com/javaee/5/docs/tutorial/doc/bnaph.html

El tutorial de JavaServer Faces de Java EE 5.

jsftutorials.net/

Vínculos a tutoriales y artículos generales sobre JavaServer Faces.

jaserverfaces.dev.java.net

Descargue la versión más reciente de la implementación de JavaServer Faces de Sun.

java.sun.com/javaee/jaserverfaces/reference/api/

Documentación sobre la biblioteca de etiquetas, la API y el Standard RenderKit para todas las versiones de JSF.

java.sun.com/javaee/5/docs/tutorial/doc/JSFCustom.html

Un tutorial acerca de cómo crear componentes JSF personalizados.

bpcatalog.dev.java.net/nonav/webtier/index.html

El catálogo de soluciones de Java BluePrints contiene ejemplos de código reutilizable para diseñar aplicaciones Web mediante el uso de JavaServer Faces y AJAX.

Resumen

Sección 26.1 Introducción

- Las aplicaciones basadas en Web crean contenido para los clientes navegadores Web.
- AJAX ayuda a las aplicaciones basadas en Web a proporcionar la interactividad y capacidad de respuesta que los usuarios esperan generalmente de las aplicaciones de escritorio.

Sección 26.2 Transacciones HTTP simples

- El Protocolo de transferencia de hipertexto (HTTP) especifica un conjunto de métodos y encabezados que permiten a los clientes y servidores interactuar e intercambiar información de una manera uniforme y confiable.
- En su forma más simple, una página Web no es nada más que un documento XHTML que contiene marcado para describir a un navegador Web cómo debe mostrar y dar formato a la información del documento.
- Los documentos XHTML pueden contener datos de hipertexto (hipervínculos) que vinculan a distintas páginas, o a otras partes de la misma página cuando el usuario hace clic en el vínculo.

- HTTP utiliza URIs (Identificadores uniformes de recursos) para identificar los datos en Internet.
- Los URIs que especifican las ubicaciones de los documentos se llaman URLs (Localizadores uniformes de recursos). Los URLs comunes hacen referencia a archivos o directorios, y pueden hacer referencia a objetos que realicen tareas complejas.
- Un URL contiene información que lleva a un navegador al recurso que el usuario desea utilizar. Las computadoras que ejecutan software de servidor Web hacen que dichos recursos estén disponibles.
- Cuando un navegador Web recibe un URL, realiza una transacción HTTP simple para obtener y mostrar la página Web que se encuentra en esa dirección.
- El método GET de HTTP indica que el cliente desea obtener un recurso del servidor.
- Los encabezados HTTP proporcionan información acerca de los datos que se envían a un servidor, como el tipo MIME.
- MIME (Extensiones de correo Internet multipropósito) es un estándar de Internet que especifica formatos de datos, para que los programas puedan interpretar los datos en forma correcta.

Sección 26.3 *Arquitectura de aplicaciones multinivel*

- Las aplicaciones basadas en Web son aplicaciones multinivel (o de n niveles), que dividen la funcionalidad en niveles separados que, por lo general, residen en computadoras separadas.
- El nivel inferior (también conocido como el nivel de datos o de información) mantiene los datos de la aplicación. Por lo general, este nivel almacena los datos en un sistema de administración de bases de datos relacionales (RDBMS).
- El nivel intermedio implementa la lógica de negocios, de controlador y de presentación para controlar las interacciones entre los clientes de la aplicación y sus datos. El nivel intermedio actúa como intermediario entre los datos en el nivel de información y los clientes de la aplicación.
- La lógica de control del nivel intermedio procesa las peticiones de los clientes y obtiene datos de la base de datos.
- La lógica de presentación del nivel intermedio procesa los datos del nivel de información y presenta el contenido al cliente.
- Por lo general, las aplicaciones Web presentan datos a los clientes en forma de documentos XHTML.
- La lógica comercial en el nivel intermedio hace valer las reglas comerciales y asegura que los datos sean confiables antes de que la aplicación servidor actualice la base de datos, o presente los datos a los usuarios.
- Las reglas comerciales dictan la forma en que los clientes pueden o no acceder a los datos de la aplicación, y la forma en que ésta procesa los datos.
- El nivel superior (nivel cliente) es la interfaz de usuario de la aplicación, la cual recopila los datos de entrada y de salida. Los usuarios interactúan en forma directa con la aplicación a través de la interfaz de usuario que, por lo general, es el navegador Web.
- En respuesta a las acciones del usuario, el nivel cliente interactúa con el nivel intermedio para hacer peticiones y obtener datos del nivel de información. Después, el nivel cliente muestra los datos obtenidos para el usuario. El nivel cliente nunca interactúa directamente con el nivel de información.

Sección 26.4 *Tecnologías Web de Java*

- Las tecnologías Web de Java evolucionan en forma continua, para ofrecer a los desarrolladores niveles mayores de abstracción, y una mayor separación de los niveles de la aplicación. Esta separación facilita el mantenimiento y la extensibilidad de las aplicaciones Web.
- Java Studio Creator 2 nos permite desarrollar la GUI de una aplicación Web mediante una herramienta de diseño tipo “arrastrar y soltar”, mientras que podemos manejar la lógica comercial en clases de Java separadas.

Sección 26.4.1 *Servlets*

- Los servlets utilizan el modelo petición-respuesta HTTP de comunicación entre cliente y servidor.
- Los servlets extienden la funcionalidad de un servidor, al permitir que éste genere contenido dinámico. Un contenedor de servlets, ejecuta los servlets e interactúa con ellos.
- Los paquetes `javax.servlet` y `javax.servlet.http` contienen las clases e interfaces para definir servlets.
- El contenedor de servlets recibe peticiones HTTP de un cliente y dirige cada petición al servlet apropiado. El servlet procesa la petición y devuelve una respuesta apropiada al cliente; por lo general en forma de un documento XHTML o XML.
- Todos los servlets implementan a la interfaz `Servlet` del paquete `javax.servlet`, la cual asegura que cada servlet se pueda ejecutar en el marco de trabajo proporcionado por el contenedor de servlets. La interfaz `Servlet` declara métodos que el contenedor de servlets utiliza para administrar el ciclo de vida del servlet.
- El ciclo de vida de un servlet empieza cuando el contenedor de servlets lo carga en memoria; por lo general, en respuesta a la primera petición del servlet. El contenedor invoca al método `init` del servlet, el cual se llama sólo

una vez durante el ciclo de vida de un servlet para inicializarlo. Una vez que `init` termina su ejecución, el servlet está listo para responder a su primera petición. Todas las peticiones se manejan mediante el método `service` de un servlet, el cual recibe la petición, la procesa y envía una respuesta al cliente. Se hace una llamada al método `service` por cada petición. Cuando el contenedor de servlets termina el servlet, se hace una llamada al método `destroy` del servlet para liberar los recursos que éste ocupa.

Sección 26.4.2 *JavaServer Pages*

- La tecnología JavaServer Pages (JSP) es una extensión de la tecnología de los servlets. El contenedor de JSPs traduce cada JSP y la convierte en un servlet.
- A diferencia de los servlets, las JSPs nos ayudan a separar la presentación del contenido.
- Las JavaServer Pages permiten a los programadores de aplicaciones Web crear contenido dinámico mediante la reutilización de componentes predefinidos, y mediante la interacción con componentes que utilizan secuencias de comandos del lado servidor.
- Los programadores de JSPs pueden utilizar componentes especiales de software llamados JavaBeans, y bibliotecas de etiquetas personalizadas que encapsulan una funcionalidad dinámica y compleja.
- Las bibliotecas de etiquetas personalizadas permiten a los desarrolladores de Java ocultar el código para acceder a una base de datos y otras operaciones complejas mediante etiquetas personalizadas. Para usar dichas herramientas, sólo tenemos que agregar las etiquetas personalizadas a la página. Esta simpleza permite a los diseñadores de páginas Web, que no estén familiarizados con Java, mejorar las páginas Web con poderoso contenido dinámico y capacidades de procesamiento.
- Las clases e interfaces de JSP se encuentran en los paquetes `javax.servlet.jsp` y `javax.servlet.jsp.tagext`.
- Hay cuatro componentes clave para las JSPs: directivas, acciones, elementos de secuencia de comandos y bibliotecas de etiquetas.
- Las directivas son mensajes para el contenedor de JSPs que nos permiten especificar configuraciones de páginas, incluir contenido de otros recursos y especificar bibliotecas de etiquetas personalizadas para usarlas en las JSPs.
- Las acciones encapsulan la funcionalidad en etiquetas predefinidas que los programadores pueden incrustar en JSPs. A menudo, las acciones se realizan con base en la información que se envía al servidor como parte de una petición específica de un cliente. También pueden crear objetos de Java para usarlos en las JSPs.
- Los elementos de secuencia de comandos permiten al programador insertar código de Java que interactúe con los componentes en una JSP para realizar el procesamiento de peticiones.
- Las bibliotecas de etiquetas permiten a los programadores crear etiquetas personalizadas, y a los diseñadores de páginas Web manipular el contenido de las JSPs sin necesidad de tener un conocimiento previo sobre Java.
- La Biblioteca de etiquetas estándar de JavaServer Pages (JSTL) proporciona la funcionalidad para muchas tareas de aplicaciones Web comunes.
- Las JSPs pueden contener otro tipo de contenido estático, como marcado XHTML o XML, el cual se conoce como datos de plantilla fija o texto de plantilla fija. Cualquier texto literal en una JSP se traduce en una literal `String` en la representación de la JSP en forma de servlet.
- Cuando un servidor habilitado para JSP recibe la primera petición para una JSP, el contenedor de JSPs traduce esa JSP en un servlet, el cual maneja la petición actual y las futuras peticiones a esa JSP.
- Las JSPs se basan en el mismo mecanismo de petición-respuesta que los servlets para procesar las peticiones de los clientes, y enviar las respuestas.

Sección 26.4.3 *JavaServer Faces*

- JavaServer Faces (JSF) es un marco de trabajo para aplicaciones Web que simplifica el diseño de la interfaz de usuario de una aplicación, y separa aún más la presentación de una aplicación Web de su lógica comercial.
- Un marco de trabajo simplifica el desarrollo de aplicaciones, al proporcionar bibliotecas y (algunas veces) herramientas de software para ayudar al programador a organizar y crear sus aplicaciones.
- JSF proporciona bibliotecas de etiquetas personalizadas que contienen componentes de interfaz de usuario, los cuales simplifican el diseño de páginas Web. JSF también incluye APIs para manejar eventos de componentes.
- El programador diseña la apariencia visual de una página con JSF, agregando etiquetas a un archivo JSP y manipulando sus atributos. El programador define el comportamiento de la página por separado, en un archivo de código fuente de Java relacionado.

Sección 26.4.4 *Tecnologías Web en Java Studio Creator 2*

- Las aplicaciones Web de Java Studio Creator 2 consisten en una o más páginas Web JSP, integradas en el marco de trabajo JavaServer Faces. Cada archivo JSP tiene la extensión `.jsp` y contiene los elementos de la GUI de la página Web.

- Java Studio Creator 2 nos permite diseñar páginas en forma visual, al arrastrar y soltar componentes JSF en una página; también podemos personalizar una página Web al editar su archivo `.jsp` en forma manual.
- Cada archivo JSP que se crea en Java Studio Creator 2 representa una página Web, y tiene su correspondiente clase `JavaBean`, denominada `bean` de página.
- Una clase `JavaBean` debe tener un constructor predeterminado (o sin argumentos), junto con métodos *obtener* (*get*) y *establecer* (*set*) para todas sus propiedades.
- El `bean` de página define las propiedades para cada uno de los elementos de la página, y contiene manejadores de eventos, métodos de ciclo de vida de las páginas y demás código de soporte para la aplicación Web.
- Toda aplicación Web creada con Java Studio Creator 2 tiene un `bean` de página, un objeto `RequestBean`, un objeto `SessionBean` y un objeto `ApplicationBean`.
- El objeto `RequestBean` se mantiene en ámbito de petición; este objeto existe sólo mientras dure una petición HTTP.
- Un objeto `SessionBean` tiene ámbito de sesión; el objeto existe durante una sesión de navegación del usuario, o hasta que se agota el tiempo de la sesión. Hay un único objeto `SessionBean` para cada usuario.
- El objeto `ApplicationBean` tiene ámbito de aplicación; este objeto es compartido por todas las instancias de una aplicación y existe mientras que la aplicación esté desplegada en un servidor Web. Este objeto se utiliza para almacenar datos a nivel de aplicación o para procesamiento; sólo existe una instancia para la aplicación, sin importar el número de sesiones abiertas.

Sección 26.5.1 Análisis de un archivo JSP

- Java Studio Creator 2 genera un archivo JSP en respuesta a las interacciones del programador con el Editor visual.
- Todas las JSPs tienen un elemento `jsp:root` con un atributo `version` para indicar la versión de JSP que se utiliza, y uno o más atributos `xmlns`. Cada atributo `xmlns` especifica un prefijo y un URL para una biblioteca de etiquetas, lo cual permite a la página usar las etiquetas especificadas en esa biblioteca.
- Todas las JSPs generadas por Java Studio Creator 2 incluyen las bibliotecas de etiquetas para la biblioteca de componentes JSF básicos, la biblioteca de componentes JSF de HTML, la biblioteca de componentes JSP estándar y la biblioteca de componentes JSP de interfaz de usuario.
- El atributo `contentType` del elemento `jsp:directive.page` especifica el tipo MIME y el conjunto de caracteres utilizado por la página. El atributo `pageEncoding` especifica la codificación de caracteres utilizada por la página de origen. Estos atributos ayudan al cliente a determinar cómo desplegar el contenido.
- Todas las páginas que contienen componentes JSF se representan en un árbol de componentes, con el elemento JSF raíz `f:view` (de tipo `UIViewRoot`). Todos los elementos de los componentes JSF se colocan en este elemento.
- Muchos elementos de página `ui` tienen un atributo `binding` para enlazar sus valores a las propiedades en los `JavaBeans` de la aplicación Web. Para realizar estos enlaces, se utiliza el Lenguaje de expresiones de JSF.
- El elemento `ui:head` tiene un atributo `title` que especifica el título de la página.
- Un elemento `ui:link` se puede utilizar para especificar la hoja de estilos CSS utilizada por una página.
- Un elemento `ui:body` define el cuerpo de la página.
- Un elemento `ui:form` define a un formulario en una página.
- Un componente `ui:staticText` muestra texto que no cambia.
- Los elementos JSP se asignan a elementos XHTML para desplegarlos en un navegador. Los mismos elementos JSP se pueden asignar a distintos elementos XHTML, dependiendo del navegador cliente y de las configuraciones de las propiedades de los componentes.
- Por lo general, un componente `ui:staticText` se asigna a un elemento `span` de XHTML. Un elemento `span` contiene texto que se muestra en una página Web, y se utiliza para controlar el formato del texto. El atributo `style` de un elemento `ui:staticText` se representará como parte del atributo `style` del elemento `span` correspondiente cuando el navegador despliegue la página.

Sección 26.5.2 Análisis de un archivo de bean de página

- Las clases de `bean` de página heredan de la clase `AbstractPageBean` (paquete `com.sun.rave.web.ui.appbase`), la cual proporciona métodos para el ciclo de vida de la página.
- El paquete `com.sun.rave.web.ui.component` incluye clases para muchos componentes JSF básicos.
- Un componente `ui:staticText` es un objeto `StaticText` (paquete `com.sun.rave.web.ui.component`).

Sección 26.5.3 Ciclo de vida del procesamiento de eventos

- El modelo de aplicación de Java Studio Creator 2 coloca varios métodos (`init`, `preprocess`, `render` y `destroy`) en el `bean` de página que se enlaza con el ciclo de vida de procesamiento de eventos JSF. Estos métodos representan cuatro etapas principales: inicialización, pre-procesamiento, pre-despliegue y destrucción.


- El contenedor de JSPs llama al método `init` la primera vez que se solicita la página, y en las peticiones de devolución de envío (postback). Una petición de devolución de envío ocurre cuando se envían formularios, y la página (junto con su contenido) se envía al servidor para su procesamiento.
- El método `init` invoca a la versión de su superclase, y después trata de llamar al método `_init`, el cual maneja las tareas de inicialización de los componentes.
- El método `preprocess` se llama después de `init`, pero sólo si la página está procesando una petición de devolución de envío. El método `render` se llama justo antes de que el navegador despliegue una página. Este método se debe utilizar para establecer las propiedades de los componentes; las propiedades que se establezcan antes de tiempo (como en el método `init`) pueden sobrescribirse antes de que el navegador llegue a desplegar la página.
- El método `destroy` se llama después de haber desplegado la página, pero sólo si se llamó al método `init`. Este método maneja las tareas tales como liberar los recursos utilizados para desplegar la página.

Sección 26.5.4 Relación entre la JSP y los archivos de bean de página

- El bean de página tiene una propiedad para cada elemento que aparece en el archivo JSP.

Sección 26.5.5 Análisis del XHTML generado por una aplicación Web de Java

- Para crear una nueva aplicación Web, seleccione **Archivo > Nuevo proyecto...** para mostrar el cuadro de diálogo **Nuevo proyecto**. En este cuadro de diálogo, seleccione **Web** en el panel **Categorías**, **Aplicación web JSF** en el panel **Proyectos** y haga clic en **Siguiente**. Especifique el nombre del proyecto y la ubicación. Haga clic en **Terminar** para crear el proyecto de aplicación Web.
- Al crear un nuevo proyecto, Java Studio Creator 2 crea una sola página Web llamada **Page1**. Esta página se abre de manera predeterminada en el Editor visual en modo **Diseño**, cuando el proyecto se carga por primera vez. A medida que el programador arrastra y suelta nuevos componentes en la página, el modo **Diseño** le permite ver cómo se desplegará la página en el navegador. El archivo JSP para esta página, llamado **Page1.jsp**, se puede ver haciendo clic en el botón **JSP** en la parte superior del Editor visual, o haciendo clic con el botón derecho del ratón en cualquier parte del Editor visual y seleccionando la opción **Editar origen JSP**.
- Para abrir el archivo de bean de página correspondiente, haga clic en el botón **Java** en la parte superior del Editor visual, o haga clic con el botón derecho del ratón en cualquier parte del Editor visual y seleccione **Editar origen Java Page1**.
- El botón **Previsualizar en navegador** en la parte superior de la ventana Editor visual le permite ver sus páginas en un navegador sin tener que crear y ejecutar la aplicación.
- El botón **Actualizar** vuelve a dibujar la página en el Editor visual.
- La lista desplegable **Tamaño de navegador de destino** nos permite especificar la resolución óptima del navegador para ver la página, y nos permite ver cuál será la apariencia de la página en distintas resoluciones de pantalla.
- La ventana **Proyectos** en la esquina inferior derecha del IDE muestra la jerarquía de todos los archivos del proyecto. El nodo **Páginas Web** contiene los archivos JSP e incluye la carpeta **resources**, la cual contiene la hoja de estilo CSS para el proyecto, y cualquier otro archivo que puedan necesitar las páginas para mostrarse en forma apropiada (como los archivos de imagen). El código fuente de Java, incluyendo el archivo de bean de página para cada página Web y los beans de aplicación, sesión y petición, se pueden encontrar bajo el nodo **Paquetes de origen**.
- El archivo **Navegación de página** define reglas para navegar por las páginas del proyecto, con base en el resultado de algún evento iniciado por el usuario, como hacer clic en un botón o en un vínculo. También podemos acceder al archivo **Navegación de página** si hacemos clic con el botón derecho del ratón en el Editor visual, estando en modo **Diseño**; para ello, seleccionamos la opción **Navegación de página...**
- Los métodos `init`, `preprocess`, `render` y `destroy` se sobrescriben en cada bean de página. Aparte del método `init`, estos métodos están vacíos al principio. Sirven como receptáculos para que usted pueda personalizar el comportamiento de su aplicación Web.
- Por lo general, es conveniente cambiar el nombre de los archivos JSP y Java en el proyecto, de manera que los nombres sean relevantes para nuestra aplicación. Para ello, haga clic en el archivo JSP en la **Ventana Proyectos** y seleccione **Cambiar nombre** para mostrar el cuadro de diálogo **Cambiar nombre**. Escriba el nuevo nombre para el archivo. Si está activada la opción **Previsualizar todos los cambios**, aparecerá la **Ventana Refactorización** en la parte inferior del IDE cuando haga clic en **Siguiente >**. No se realizarán cambios hasta que haga clic en el botón **Refactorizar** de la **Ventana Refactorización**. Si no previsualiza los cambios, la refactorización se llevará a cabo cuando haga clic en el botón **Siguiente >** del cuadro de diálogo **Cambiar nombre**.
- La refactorización es el proceso de modificar el código fuente para mejorar su legibilidad y reutilización, sin cambiar su comportamiento; por ejemplo, al cambiar el nombre a los métodos o variables, o al dividir métodos extensos en métodos más cortos. Java Studio Creator 2 tiene herramientas de refactorización integradas, que automatizan ciertas tareas de refactorización.

- Para agregar un título, abra la página JSP en modo **Diseño**. En la ventana **Propiedades**, escriba el nuevo título en la propiedad **Título** y oprima **Intro**.
- Para agregar componentes a una página, arrástrelos y suéltelos desde la **Paleta** hacia la página en modo **Diseño**. Cada componente es un objeto que tiene propiedades, métodos y eventos. Puede establecer estas propiedades y eventos en la ventana **Propiedades**, o mediante programación en el archivo de bean de página. Los métodos *obtener (get)* y *establecer (set)* se agregan al archivo de bean de página para cada componente que agregamos a la página.
- Los componentes se despliegan usando el posicionamiento absoluto, de manera que aparezcan exactamente en donde se sueltan en la página.
- Java Studio Creator 2 es un editor WYSIWYG (What You See Is What You Get —Lo que ve es lo que obtiene); cada vez que realice un cambio a una página Web en modo **Diseño**, el IDE creará el marcado (visible en modo **JSP**) necesario para lograr los efectos visuales deseados que aparecen en modo **Diseño**.
- Después de diseñar la interfaz, puede modificar el bean de página para agregar su lógica comercial.
- La ventana **Esquema** muestra el bean de página y los beans de ámbito de petición, sesión y aplicación. Al hacer clic en un elemento en el árbol de componentes del bean de página, se selecciona el elemento en el Editor visual.
- Seleccione **Generar > Generar proyecto principal**, y después **Ejecutar > Ejecutar proyecto principal** para ejecutar la aplicación.
- Para agregar un proyecto que ya haya sido creado, oprima el ícono **Ejecutar proyecto principal** () en la barra de herramientas que se encuentra en la parte superior del IDE.
- Si se hacen cambios a un proyecto, éste debe volver a generarse para que puedan reflejarse los cambios cuando se vea la aplicación en un navegador Web.
- Oprima **F5** para generar la aplicación y después ejecutarla en modo de depuración. Si escribe **<Ctrl> F5**, el programa se ejecutará sin la depuración habilitada.

Sección 26.5.6 Creación de una aplicación Web en Java Studio Creator 2

- El componente **Panel de cuadrícula** permite al diseñador especificar el número de columnas que debe contener la cuadrícula. Después se pueden soltar los componentes en cualquier parte dentro del panel, y éstos se reposicionarán automáticamente en columnas espaciadas de manera uniforme, en el orden en el que se suelten. Cuando el número de componentes excede al número de columnas, el panel desplaza los componentes adicionales hacia una nueva fila.
- Un componente **Imagen** (de la clase `Image`) inserta una imagen en una página Web. Las imágenes que se van a mostrar en una página Web se deben colocar en la carpeta `resources` del proyecto. Para agregar imágenes al proyecto, suelte un componente **Imagen** en la página y haga clic en el botón de elipsis que está a un lado de la propiedad `url` en la ventana **Propiedades**. A continuación se abrirá un cuadro de diálogo, en el que puede seleccionar la imagen a mostrar.
- Los componentes **Campo de texto** nos permiten obtener entrada de texto del usuario.
- Observe que el orden en el que se arrastran los componentes a la página es importante, ya que sus etiquetas JSP se agregan al archivo JSP en ese orden. El uso de tabuladores entre los componentes permite navegar por los componentes en el orden en el que se hayan agregado las etiquetas JSP al archivo JSP. Si desea que el usuario navegue por los componentes en cierto orden, debe arrastrarlos a la página en ese orden. De manera alternativa, puede establecer la propiedad **Tab Index** de cada campo de entrada en la ventana **Propiedades**. Un componente con un índice de tabulación de 1 será el primero en la secuencia de tabulaciones.
- Una **Lista desplegable** muestra una lista, de la cual el usuario puede seleccionar un elemento. Este objeto se puede configurar haciendo clic con el botón derecho en la lista desplegable en modo **Diseño** y seleccionando **Configurar opciones predeterminadas**, con lo cual se abrirá el cuadro de diálogo **Personalizador de opciones** para agregar opciones a la lista.
- Un componente **Hipervínculo** de la clase `HyperLink` agrega un vínculo a una página Web. La propiedad `url` de este componente especifica el recurso que se solicita cuando un usuario hace clic en el hipervínculo.
- Un componente **Grupo de botones de selección** de la clase `RadioButtonGroup` proporciona una serie de botones de opción, de los cuales el usuario puede seleccionar sólo uno. Para editar las opciones, haga clic con el botón derecho del ratón en el componente y seleccione **Configurar opciones predeterminadas**.
- Un **Botón** es un componente JSF de la clase `Button` que desencadena una acción cuando es oprimido. Por lo general, un componente **Button** se asigna a un elemento `input` de XHTML con el atributo `type` establecido en `submit`.

Sección 26.6.2 Validación mediante los componentes de validación y validadores personalizados

- La validación ayuda a prevenir los errores de procesamiento, debido a una entrada del usuario incompleta o con un formato inapropiado.
- Un **Validador de longitud** determina si un campo contiene un número aceptable de caracteres.

- El **Validador de intervalo doble** y el **Validador de intervalo largo** determinan si la entrada numérica se encuentra dentro de intervalos aceptables.
- El paquete `javax.faces.validators` contiene las clases para estos validadores.
- Los componentes **Etiqueta** describen a otros componentes, y se pueden asociar con los campos de entrada del usuario si se establece su propiedad `for`.
- Los componentes **Mensaje** muestran mensajes de error cuando falla la validación.
- Para asociar un componente **Etiqueta** o **Mensaje** con otro componente, mantenga oprimidas las teclas *Ctrl* y *Mayús*, y después arrastre la etiqueta o mensaje hacia el componente apropiado.
- Establezca la propiedad `required` de un componente para asegurar que el usuario escriba datos para éste.
- Si agrega un componente de validación o un método de validación personalizado a un campo de entrada, la propiedad `required` de ese campo se debe establecer en `true` para que ocurra la validación.
- En el Editor visual, la etiqueta para un campo requerido se marca automáticamente con un asterisco color rojo.
- Si un usuario envía un formulario con un campo de texto vacío para el cual se requiere un valor, se mostrará el mensaje de error predeterminado para ese campo en su componente `ui:message` asociado.
- Para editar las propiedades de un **Validador de intervalo doble** o de un **Validador de intervalo largo**, haga clic en su nodo en la ventana **Esquema** en modo **Diseño** y establezca las propiedades `minimum` y `maximum` en la ventana **Propiedades**.
- Es posible limitar la longitud de la entrada del usuario sin validación, para lo cual hay que establecer la propiedad `maxLength` de un componente **Campo de texto**.
- Comparar la entrada del usuario con una expresión regular es una forma efectiva de asegurar que la entrada tenga un formato apropiado.
- Java Studio Creator 2 no proporciona componentes para validación mediante el uso de expresiones regulares, pero podemos agregar nuestros propios métodos de validación al archivo de bean de página.
- Para agregar un método de validación personalizado a un componente de entrada, haga clic con el botón derecho en el componente y seleccione **Editar manejador de eventos > validate** para crear un método de validación para el componente en el archivo de bean de página.

Sección 26.7 Rastreo de sesiones

- La personalización hace posible que los comercios electrónicos se comuniquen con eficiencia con sus clientes, y también mejora la habilidad del usuario para localizar los productos y servicios deseados.
- Existe una concesión entre el servicio de comercio electrónico personalizado y la protección de la privacidad. Algunos consumidores adoptan la idea del contenido personalizado, pero otros temen a las posibles consecuencias adversas, si la información que proporcionan a los comercios electrónicos es liberada o recolectada por tecnologías de rastreo.
- Para proporcionar servicios personalizados a los consumidores, los comercios electrónicos deben tener la capacidad de reconocer a los clientes cuando solicitan información de un sitio. Por desgracia, HTTP es un protocolo sin estado; no soporta conexiones persistentes que permitan a los servidores Web mantener información de estado, en relación con clientes específicos. Por lo tanto, los servidores Web no pueden determinar si una petición proviene de un cliente específico, o si una serie de peticiones provienen de uno o varios clientes.
- Para ayudar al servidor a diferenciar un cliente de otro, cada cliente debe identificarse a sí mismo con el servidor. El rastreo de clientes individuales, conocido como rastreo de sesiones, puede lograrse de varias formas. Una técnica popular utiliza cookies; otra utiliza el objeto `SessionBean`.
- Con los elementos "hidden", un formulario Web puede escribir los datos de rastreo de sesión en un componente form en la página Web que devuelve al cliente, en respuesta a una petición previa. Cuando el usuario envía el formulario en la nueva página Web, todos los datos del formulario (incluyendo los campos "hidden") se envían al manejador del formulario en el servidor Web. Con la reescritura de URLs, el servidor Web incrusta la información de rastreo de sesión directamente en los URLs de los hipervínculos en los que el usuario hace clic para enviar las subsiguientes peticiones al servidor Web.

Sección 26.7.1 Cookies

- Una cookie es una pieza de datos que, por lo general, se almacena en un archivo de texto en la computadora del usuario. Una cookie mantiene información acerca del cliente, durante y entre las sesiones del navegador.
- La primera vez que un usuario visita el sitio Web, su computadora podría recibir una cookie; después, esta cookie se reactiva cada vez que el usuario vuelve a visitar ese sitio. La información recolectada tiene el propósito de ser un registro anónimo que contiene datos, los cuales se utilizan para personalizar las visitas futuras del usuario al sitio Web.
- Cada interacción basada en HTTP entre un cliente y un servidor incluye un encabezado, el cual contiene información sobre la petición (cuando la comunicación es del cliente al servidor) o sobre la respuesta (cuando la comunicación es del servidor al cliente).

- Cuando una página recibe una petición, el encabezado incluye información como el tipo de petición y cualquier cookie que se haya enviado anteriormente del servidor, para almacenarse en el equipo cliente. Cuando el servidor formula su respuesta, la información del encabezado contiene cualquier cookie que el servidor desee almacenar en la computadora cliente, junto con más información, como el tipo MIME de la respuesta.
- La fecha de expiración de una cookie determina la forma en que ésta permanecerá en la computadora del cliente. Si no establecemos una fecha de expiración para la cookie, el navegador Web mantendrá la cookie mientras dure la sesión de navegación. En caso contrario, el navegador Web mantendrá la cookie hasta que llegue la fecha de expiración.
- Al establecer el manejador de acciones para un **Hipervínculo** podemos responder a un clic sin necesidad de redirigir al usuario hacia otra página.
- Para agregar un manejador de acciones a un **Hipervínculo** que también debe dirigir al usuario a otra página, debemos agregar una regla al archivo **Navegación de página**. Para editar este archivo, haga clic con el botón derecho del ratón en cualquier parte del Diseñador visual y seleccione **Navegación de página...**; después arrastre el **Hipervínculo** apropiado a la página de destino.
- Un objeto cookie es una instancia de la clase `Cookie` en el paquete `javax.servlet.http`.
- Un objeto de la clase `HttpServletResponse` (del paquete `javax.servlet.http`) representa la respuesta. Para acceder a este objeto, hay que invocar el método `getExternalContext` en el bean de página, y después invocar a `getResponse` en el objeto resultante.
- Un objeto de la clase `HttpServletRequest` (del paquete `javax.servlet.http`) representa la petición. Para obtener este objeto, hay que invocar al método `getExternalContext` en el bean de página, y después invocar a `getRequest` en el objeto resultante.
- El método `getCookies` de `HttpServletRequest` devuelve un arreglo de las cookies que se escribieron previamente en el cliente.
- Un servidor Web no puede acceder a las cookies creadas por los servidores en otros dominios.

Sección 26.7.2 Rastreo de sesiones con el objeto *SessionBean*

- Podemos llevar a cabo el rastreo de sesiones con la clase `SessionBean` que se proporciona en cada aplicación Web creada con Java Studio Creator 2. Cuando un nuevo cliente solicita una página Web en el proyecto, se crea un objeto `SessionBean`.
- Podemos acceder al objeto `SessionBean` a través de una sesión, invocando al método `getSessionBean` en el bean de página. Podemos usar el objeto `SessionBean` para acceder a las propiedades de sesión almacenadas.
- Para almacenar información en el objeto `SessionBean`, agregue propiedades a la clase `SessionBean`. Para agregar una propiedad, haga clic con el botón derecho del ratón en el nodo `SessionBean` en la ventana **Esquema** y seleccione **Agregar | Propiedad** para mostrar el cuadro de diálogo **Nuevo patrón de propiedad**. Configure la propiedad y haga clic en **Aceptar** para crearla.

Terminología

`AbstractPageBean`

acción en una JSP

`action`, atributo del elemento `form` de XHTML

aplicación basada en Web de tres niveles

aplicación de *n* niveles

aplicación multinivel

`ApplicationBean`

árbol de componentes

bean de página

biblioteca de etiquetas

biblioteca de etiquetas personalizadas

búsqueda DNS

Button, componente JSF

Campo de texto, componente JSF

ciclo de vida del procesamiento de eventos

`com.sun.rave.web.ui.component`

contenedor de JSPs

contenedor de servlets

cookie

Cuadro de lista, componente JSF

datos de plantilla fija

desarrollo de aplicación Web

desplegar XHTML en un navegador Web

`destroy`, método del ciclo de vida de procesamiento de eventos

dirección IP

directiva en una JSP

directorio virtual

Diseño, modo

Editor visual

editor WYSIWYG (Lo que ve es lo que obtiene)

elemento de secuencia en una JSP

encabezado HTTP

entrada oculta en un formulario de XHTML

`escaped`, propiedad

Esquema, ventana

Etiqueta, componente JSF

etiqueta de XHTML

etiqueta final
 etiqueta inicial
 etiqueta personalizada
 fecha de expiración de una cookie
 GET, petición http
Grupo de botones de selección, componente JSF
 hipertexto
 hipervínculo
Hiperlink, componente JSF
 host
Image, componente JSF
`init`, método del ciclo de vida de procesamiento de eventos
 Java BluePrints
 Java Studio Creator 2
 JavaBeans
`javax.servlet`, paquete
`javax.servlet.http`, paquete
 JSF (JavaServer Faces)
 JSF, componentes
 JSF, Lenguaje de expresiones
 JSP (JavaServer Pages)
`.jsp`, extensión de archivo
 JSTL (Biblioteca de etiquetas estándar de JSP)
Lista desplegable, componente JSF
 localhost
 lógica comercial
 lógica de control
 lógica de presentación
 marcado de XHTML
 marco de trabajo
 mecanismo de extensión de etiquetas
Mensaje, componente JSF
`method`, atributo del elemento `form` de XHTML
 método HTTP
 MIME (Extensiones de correo Internet multipropósito)
 nivel cliente
 nivel de datos
 nivel de información
 nivel en una aplicación multinivel
 nivel inferior
 nivel intermedio
 nivel superior
 nombre de host
Paleta
Panel de cuadrícula, componente JSF
 personalización
 petición de devolución de envío (postback)
 posicionamiento absoluto
preprocess, método del ciclo de vida de procesamiento de eventos
prerender, método del ciclo de vida del procesamiento de eventos
 rastreo de sesiones
 refactorización
 regla comercial
rendered, propiedad
RequestBean
required, propiedad
service, método de la interfaz `Servlet`
 servidor DNS (sistema de nombres de dominio)
 servidor Web
`servlet`
Servlet, interfaz
SessionBean
`span`, elemento
 Sun Application Server 8
 texto de plantilla fija
Texto estático, componente JSF
title, elemento de XHTML
 validación
Validador de intervalo doble, componente JSF
Validador de intervalo largo, componente JSF
Validador de longitud, componente JSF
 XML
`xmlns`, atributos

Ejercicios de autoevaluación

- 26.1** Conteste con *verdadero* o *falso* a cada una de las siguientes proposiciones; en caso de ser *falso*, explique por qué.
- Toda página Web JSP creada en Java Studio Creator 2 tiene sus propios archivos `ApplicationBean`, `SessionBean` y `RequestBean`.
 - El método `init` del ciclo de vida de procesamiento de eventos se invoca cada vez que se carga una página.
 - Cada componente en una página Web JSP está enlazado a una propiedad en el archivo Java de bean de página.
 - Un solo componente JSF puede tener varios componentes de validación colocados sobre él.
 - Si no se establece una fecha de expiración para una cookie, esa cookie se destruirá al final de la sesión del navegador.
 - Cada componente JSF se asigna sólo a un elemento XHTML correspondiente.
 - Las expresiones en la sintaxis del Lenguaje de expresiones JSF se delimitan por los signos `<!--` y `-->`.
 - El objeto `SessionBean` puede almacenar sólo propiedades primitivas y propiedades de tipo `String`.

26.2 Complete las siguientes oraciones:

- a) Las aplicaciones Web contienen tres niveles básicos: _____, _____ y _____.
- b) El componente JSF _____ se utiliza para mostrar mensajes de error, en caso de que falle la validación.
- c) Un componente que comprueba la entrada en otro componente antes de enviar esa entrada al servidor se llama _____.
- d) Cada clase de bean de página hereda de la clase _____.
- e) Cuando una página se carga la primera vez, el evento _____ ocurre primero, seguido del evento _____.
- f) El archivo _____ contiene la funcionalidad para una JSP.
- g) Un _____ se puede utilizar en un método de validación personalizado para validar el formato de la entrada del usuario.
- h) El arreglo de objetos `Cookie` almacenados en el cliente se puede obtener llamando al método `getCookies` en el objeto _____.
- i) En una aplicación multinivel, el nivel _____ controla las interacciones entre los clientes de la aplicación y los datos de la misma.

Respuestas a los ejercicios de autoevaluación

26.1 a) Falso. Si una aplicación contiene varias JSPs, esas JSPs compartirán los beans de datos con ámbito. b) Falso. `init` se invoca la primera vez que se solicita la página, pero no en las actualizaciones de páginas. c) Verdadero. d) Verdadero. e) Verdadero. f) Falso. Un componente Web se puede asignar a un grupo de elementos de XHTML; las JSPs pueden generar marcado de XHTML complejo, a partir de componentes simples. g) Falso. `#{}` delimitan las instrucciones del Lenguaje de expresiones JSF. h) Falso. Los beans de datos con ámbito pueden almacenar cualquier tipo de propiedad.

26.2 a) inferior (información), intermedio (lógica comercial), superior (cliente). b) **Mensaje**. c) validador. d) `AbstractPageBean`. e) `init`, `render`. f) bean de página. g) expresión regular. h) Petición (`HttpServletRequest`). i) intermedio.

Ejercicios

26.3 (*Modificación de HoraWeb*) Modifique el ejemplo HoraWeb para que contenga listas desplegables que permitan al usuario modificar las propiedades de los componentes **Texto estático** tales como `background-color`, `color` y `font-size`. Cuando se vuelva a cargar la página, deberá reflejar los cambios especificados en las propiedades del componente **Texto estático** que muestra la hora.

26.4 (*Modificación del formulario de registro*) Modifique la aplicación ComponentesWeb para agregar funcionalidad al botón **Registro**. Cuando el usuario haga clic en **Enviar**, valide todos los campos de entrada para asegurar que el usuario haya llenado el formulario por completo y haya introducido una dirección de e-mail válida, junto con un número telefónico válido. Después, lleve al usuario a otra página que muestre un mensaje indicando un registro exitoso, y que vuelva a imprimir la información de registro para el usuario.

26.5 (*Contador de visitas a las páginas con Cookies*) Cree un archivo JSP que utilice una cookie persistente (es decir, una cookie con una fecha de expiración en el futuro) para llevar la cuenta de cuántas veces el equipo cliente ha visitado la página. Use el método `setMaxAge` para hacer que la cookie permanezca en el equipo cliente durante un mes. Muestre el número de visitas a la página (es decir, el valor de la cookie) cada vez que se cargue la página.

26.6 (*Contador de visitas de páginas con ApplicationBean*) Cree un archivo JSP que utilice el objeto `ApplicationBean` para llevar la cuenta de cuántas veces se ha visitado una página. [Nota: si desplegara esta página en la Web, contaría el número de veces que un equipo haya solicitado la página, a diferencia del ejercicio anterior]. Muestre el número de visitas a la página (es decir, el valor de una propiedad `int` en el objeto `ApplicationBean`) cada vez que se cargue la página.



Lo que de cualquier forma sea bello, tiene su fuente de belleza en sí mismo, y es completo por sí solo; el elogio no forma parte de éste.

—Marcus Aurelius Antoninus

Hay algo en un rostro, un aire, y una gracia peculiar, que los pintores más audaces no pueden trazar.

—William Somerville

Cato dijo que la mejor forma de mantener los buenos actos en la memoria era refrescarlos con nuevos actos.

—Francis Bacon

Nunca olvido un rostro, pero en su caso haré una excepción.

—Groucho Marx

La pintura es sólo un puente que enlaza la mente del pintor con la del observador.

—Eugène Delacroix

Aplicaciones Web: parte 2

OBJETIVOS

En este capítulo aprenderá a:

- Utilizar los proveedores de datos para acceder a las bases de datos desde las aplicaciones Web integradas en Java Studio Creator 2.
- Conocer los principios básicos y ventajas de la tecnología Ajax.
- Incluir componentes JSF habilitados para Ajax en un proyecto de aplicación Web de Java Studio Creator 2.
- Configurar formas virtuales que permiten a los subconjuntos de los componentes de entrada de un formulario ser enviados al servidor.

- 27.1 Introducción
 - 27.2 Acceso a bases de datos en las aplicaciones Web
 - 27.2.1 Creación de una aplicación Web que muestra datos de una base de datos
 - 27.2.2 Modificación del archivo de bean de página para la aplicación **LibretaDirecciones**
 - 27.3 Componentes JSF habilitados para Ajax
 - 27.3.1 Biblioteca de componentes Java BluePrints
 - 27.4 **AutoComplete Text Field** y formularios virtuales
 - 27.4.1 Configuración de los formularios virtuales
 - 27.4.2 Archivo JSP con formularios virtuales y un **AutoComplete Text Field**
 - 27.4.3 Cómo proporcionar sugerencias para un **AutoComplete Text Field**
 - 27.5 Componente **Map Viewer** de Google Maps
 - 27.5.1 Cómo obtener una clave de la API Google Maps
 - 27.5.2 Cómo agregar un componente **Map Viewer** a una página
 - 27.5.3 Archivo JSP con un componente **Map Viewer**
 - 27.5.4 Bean de página que muestra un mapa en el componente **Map Viewer**
 - 27.6 Conclusión
 - 27.7 Recursos Web
- Resumen | Terminología | Ejercicios de autoevaluación | Respuestas a los ejercicios de autoevaluación | Ejercicios

27.1 Introducción

En este capítulo continuaremos nuestra discusión acerca del desarrollo de aplicaciones Web con varios conceptos avanzados. Hablaremos sobre cómo acceder, actualizar y realizar búsquedas en bases de datos en una aplicación Web, cómo agregar formularios virtuales a páginas Web para permitir que se envíen subconjuntos de los componentes de entrada de un formulario al servidor, y cómo usar las bibliotecas de componentes habilitados para Ajax, para mejorar el rendimiento de la aplicación y la capacidad de respuesta de los componentes.

Presentaremos una aplicación de libreta de direcciones desarrollada en tres etapas, para ilustrar estos conceptos. La aplicación está respaldada por una base de datos Java DB para almacenar los nombres de los contactos y sus direcciones.

La aplicación de libreta de direcciones presenta un formulario que permite al usuario introducir un nuevo nombre y dirección para almacenarlos en la libreta de direcciones, y muestra el contenido de esta libreta en formato tabular. También proporciona un formulario de búsqueda que permite al usuario buscar un contacto y, si lo encuentra, muestra la dirección de ese contacto en un mapa. La primera versión de esta aplicación demuestra cómo agregar contactos a la base de datos, y cómo mostrar la lista de contactos en un componente JSF **Tabla**. En la segunda versión, agregamos un componente **Auto Complete Text Field** habilitado para Ajax y lo habilitamos para sugerir una lista de nombres de contactos, a medida que el usuario escribe información. La última versión nos permite buscar un contacto en la libreta de direcciones, y mostrar la dirección correspondiente en un mapa mediante el uso del componente **Map Viewer** habilitado para Ajax, controlado mediante Google Maps (maps.google.com).

Al igual que en el capítulo 26, desarrollamos los ejemplos de este capítulo en Java Studio Creator 2.0. Instalamos una biblioteca de componentes suplementaria (la **biblioteca de componentes Ajax de Java BluePrints**), la cual proporciona los componentes habilitados para Ajax que utilizamos en la aplicación de libreta de direcciones. En la sección 27.3.1 se incluyen instrucciones para instalar esta biblioteca.

27.2 Acceso a bases de datos en las aplicaciones Web

Muchas aplicaciones Web acceden a bases de datos para almacenar y obtener datos persistentes. En esta sección vamos a crear una aplicación Web que utiliza una base de datos Java DB para almacenar contactos en la libreta de direcciones y mostrar contactos de esta libreta en una página Web.

La página Web permite al usuario introducir nuevos contactos en un formulario. Este formulario consiste en componentes **Campo de texto** para el primer nombre del contacto, su dirección física, ciudad, estado y código-

go postal. El formulario también tiene un botón **Enviar** para enviar los datos al servidor, y un botón **Clear** para restablecer los campos del formulario. La aplicación almacena la información de la libreta de direcciones en una base de datos llamada `LibretaDirecciones`, la cual tiene una sola tabla llamada `Addresses`. (En el directorio de ejemplos para este capítulo proporcionamos esta base de datos. Puede descargar los ejemplos de www.deitel.com/books/jhttp7). Este ejemplo también introduce el componente JSF **Tabla**, el cual muestra las direcciones de la base de datos en formato tabular. En breve le mostraremos cómo configurar el componente **Tabla**.

27.2.1 Creación de una aplicación Web que muestra datos de una base de datos

Ahora le explicaremos cómo crear la GUI de la aplicación `LibretaDirecciones` y establecer un enlace de datos que permita al componente **Tabla** mostrar información de la base de datos. Más adelante en esta sección presentaremos el archivo JSP generado, y hablaremos sobre el archivo de bean de página relacionado en la sección 27.2.2. Para crear la aplicación `LibretaDirecciones`, realice los siguientes pasos:

Paso 1: Crear el proyecto

En Java Studio Creator 2, cree un proyecto **Aplicación web JSF** llamado `LibretaDirecciones`. Cambie el nombre a los archivos JSP y de bean de página a `LibretaDirecciones`, usando las herramientas de refactorización.

Paso 2: Crear el formulario para la entrada del usuario

En modo **Diseño**, agregue un componente **Texto estático** a la parte superior de la página que contenga el texto "Agregar un contacto a la libreta de direcciones:" y utilice la propiedad `style` del componente para establecer el tamaño de la fuente en 18px. Agregue seis componentes **Campo de texto** a la página y cambie su nombre a `nombreCampoTexto`, `apaternoCampoTexto`, `calleCampoTexto`, `ciudadCampoTexto`, `estadoCampoTexto` y `cpCampoTexto`. Establezca la propiedad `required` de cada **Campo de texto** en `true`; para ello seleccione el componente **Campo de texto** y después haga clic sobre la casilla de verificación de la propiedad `required`. Etiquete cada **Campo de texto** con un componente **Etiqueta** y asocie ese componente con su correspondiente **Campo de texto**. Por último, agregue los botones **Enviar** y **Borrar**. Establezca la propiedad `primary` del botón **Enviar** a `true`, para hacer que resalte más en la página que el botón **Borrar**, y para permitir que el usuario envíe un nuevo contacto, al oprimir **Intro** en vez de hacer clic en el botón **Enviar**. Establezca la propiedad `reset` del botón a `true`, para evitar la validación cuando el usuario haga clic en el botón **Borrar**. Como vamos a borrar los campos, no deseamos asegurarnos que contengan información. Hablaremos sobre el manejador de acciones para el botón **Enviar** después de presentar el archivo de bean de página. El botón **Borrar** no necesita un método manejador de acciones, ya que al establecer la propiedad `reset` a `true` el botón se configura de manera automática para restablecer todos los campos de entrada de la página. Cuando haya terminado estos pasos, su formulario deberá verse como el de la figura 27.1.

Paso 3: Agregar un componente Tabla a la página

Arrastre un componente **Tabla** de la sección **Básicos** de la **Paleta** a la página, y colóquelo justo debajo de los dos componentes **Botón**. Cambie su nombre a `direccionesTabla`. El componente **Tabla** da formato y muestra

Agregar un contacto a la libreta de direcciones:

*Primer nombre: *Apellido Paterno:

*Calle:

*Ciudad: *Estado: *Código postal:

Figura 27.1 | Formulario de la aplicación `LibretaDirecciones` para agregar un contacto.

datos de las tablas de una base de datos. En la ventana **Propiedades**, cambie la propiedad **title** de **Tabla** a **Contactos**. En breve le mostraremos cómo configurar la **Tabla** para que interactúe con la base de datos **LibretaDirecciones**.

Paso 4: Cómo agregar una base de datos a una aplicación Web de Java Studio Creator 2

Para este ejemplo, utilizaremos una base de datos Java DB llamada **LibretaDirecciones** con una sola tabla llamada **Addresses**. Para que esta base de datos esté disponible en sus proyectos, copie la carpeta **LibretaDirecciones** de la carpeta de ejemplos del capítulo, a la carpeta **SunAppServer8\derby\databases** de la carpeta de instalación de Java Studio Creator 2.

Para utilizar una base de datos en una aplicación Web de Java Studio Creator 2, primero debemos iniciar el **servidor de bases de datos integrado** del IDE, el cual permite utilizar conexiones a bases de datos en los proyectos de Java Studio Creator 2. El servidor incluye controladores para muchas bases de datos, incluyendo Java DB. Haga clic en la ficha **Servidores** debajo del menú **Archivo**, haga clic con el botón derecho en **Servidor Bundled Database** en la parte inferior de la ventana **Servidores** y seleccione **Iniciar Bundled Database**. Ahora podrá utilizar bases de datos que se ejecuten en este servidor en sus aplicaciones.

Para agregar la base de datos **LibretaDirecciones** a este proyecto, haga clic con el botón derecho en el nodo **Orígenes de datos** en la parte superior de la ventana **Servidores** y seleccione **Agregar origen de datos....** En el cuadro de diálogo **Agregar origen de datos** (figura 27.2), escriba **LibretaDirecciones** para el nombre del origen de datos y seleccione **Derby** en el tipo de servidor. (En el capítulo 25 vimos que Java DB es la versión producida por Sun de Apache Derby). El ID de usuario y la contraseña para esta base de datos son **jhttp7**. Para el URL de la base de datos, escriba **jdbc:derby://localhost:21527/LibretaDirecciones**. Este URL indica que la base de datos reside en el equipo local y acepta conexiones en el puerto 21527. Haga clic en el botón **Seleccionar** para elegir una tabla que se utilizará para validar la base de datos. En el cuadro de diálogo que aparezca, seleccione la tabla **JHTTP7.ADDRESSES**, ya que es la única tabla en la base de datos. Haga clic en **Seleccionar** para cerrar este cuadro de diálogo, y después haga clic en **Agregar** para agregar la base de datos como origen de datos para el proyecto y cierre el cuadro de diálogo. [Nota: Java Studio Creator 2 muestra los nombres de las bases de datos y las tablas en mayúsculas].

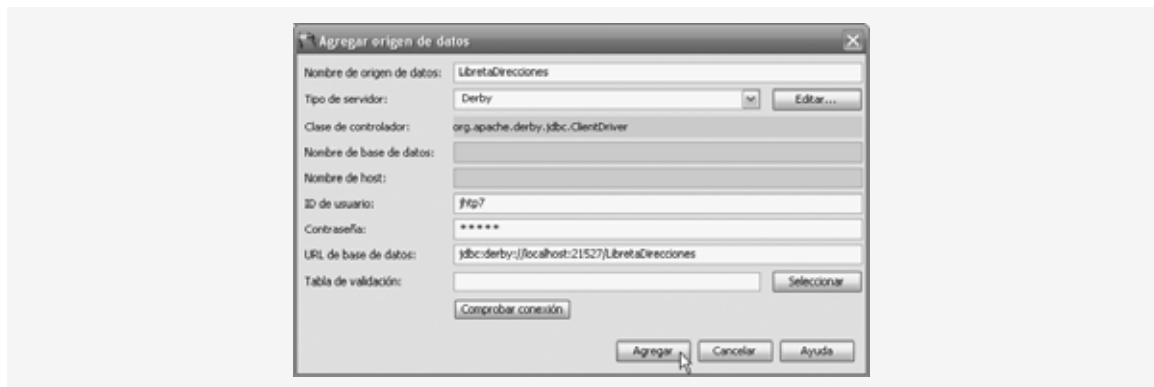


Figura 27.2 | Cuadro de diálogo para agregar un origen de datos.

Paso 5: Enlazar el componente Tabla a la tabla Addresses de la base de datos LibretaDirecciones

Ahora que hemos configurado un origen de datos para la tabla **Addresses** de la base de datos, podemos configurar el componente **Tabla** para mostrar los datos de **LibretaDirecciones**. Simplemente arrastre la tabla de la base de datos de la ficha **Servidores**, y suéltela en el componente **Tabla** para crear el enlace.

Si necesita un control más preciso sobre las columnas a mostrar, puede enlazar con una tabla de la base de datos de la siguiente manera: haga clic con el botón derecho del ratón en el componente **Tabla** y seleccione **Enlazar con datos** para mostrar el cuadro de diálogo **Diseño de tabla**. Haga clic en el botón **Agregar proveedor de datos...** para mostrar el cuadro de diálogo **Agregar proveedor de datos**, el cual contiene una lista de los orígenes de datos disponibles. Expanda el nodo **LibretaDirecciones**, expanda el nodo **Tablas**, seleccione **ADDRESSES** y haga clic

en **Agregar**. Ahora el cuadro de diálogo **Diseño de tabla** mostrará una lista de las columnas en la tabla *Addresses* de la base de datos (figura 27.3). Todos los elementos bajo el encabezado **Seleccionado** se mostrarán en la **Tabla**. Para eliminar una columna de la **Tabla**, puede seleccionarla y hacer clic en el botón <. Como deseamos mostrar todas estas columnas en nuestra tabla, simplemente haga clic en **Aceptar** para salir del cuadro de diálogo.

De manera predeterminada, la **Tabla** utiliza los nombres de las columnas de la tabla de la base de datos en mayúsculas como encabezados. Para modificar estos encabezados, seleccione una columna y edite su propiedad `headerText` en la ventana **Propiedades**. Para seleccionar una columna, expanda el nodo `direccionesTabla` en la ventana **Esquema** (estando en modo **Diseño**) y después seleccione el objeto columna apropiado. También modificamos la propiedad `id` de cada columna, para hacer más legibles los nombres de las variables en el código. En modo **Diseño**, los encabezados de las columnas de su **Tabla** deberán aparecer como en la figura 27.4.

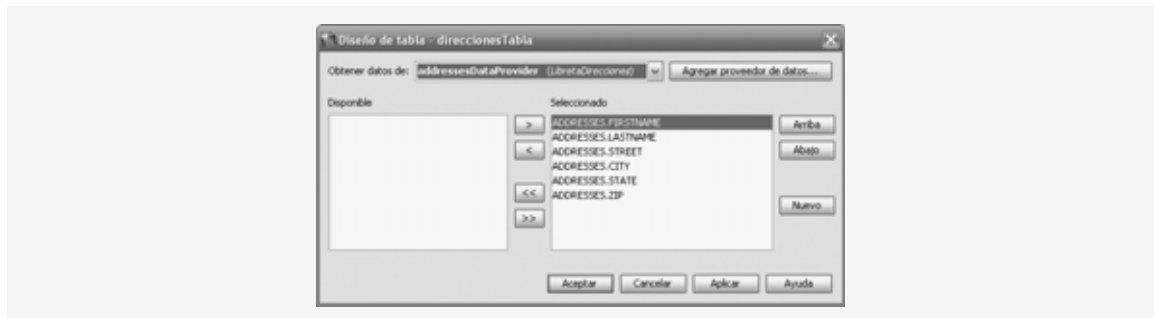


Figura 27.3 | Cuadro de diálogo para enlazar con la tabla *Addresses*.

Contactos						
Primer nombre	Apellido paterno	Calle	Ciudad	Estado	CP	
abc	abc	abc	abc	abc	abc	
abc	abc	abc	abc	abc	abc	
abc	abc	abc	abc	abc	abc	

Figura 27.4 | El componente **Tabla** después de enlazarlo con una tabla de la base de datos y editar los nombres de su columna, para fines de visualización.

Una libreta de direcciones podría contener muchos contactos, por lo que sería conveniente mostrar sólo unos cuantos a la vez. Al hacer clic en la casilla de verificación a un lado de la propiedad `paginationControl` de la tabla en la ventana **Propiedades**, se configura esta **Tabla** para paginación automática. Se mostrarán cinco filas a la vez, y se agregarán botones para avanzar hacia delante y hacia atrás, entre grupos de cinco contactos, al final de la **Tabla**. (También puede usar la ficha **Opciones** del cuadro de diálogo **Diseño de tabla** para seleccionar la paginación y el número de filas. Para ver esta ficha, haga clic con el botón derecho en la **Tabla**, seleccione **Diseño de página...**, y después haga clic en la ficha **Opciones**). A continuación, establezca la propiedad `internalVirtualForm` de `direccionesTabla`. Los formularios virtuales permiten enviar subconjuntos de los componentes de entrada de un formulario al servidor. Al establecer esta propiedad se evita que los botones de control de paginación en la **Tabla** envíen los componentes **Campo de texto** en el formulario cada vez que el usuario desea ver el siguiente grupo de contactos. En la sección 27.4.1 hablaremos sobre los formularios virtuales.

Observe que al enlazar la **Tabla** con un proveedor de datos, se agregó un nuevo objeto `addressesDataProvider` (una instancia de la clase `CachedRowSetDataProvider`) al nodo `LibretaDirecciones` en la ventana **Esquema**. Un objeto `CachedRowSetDataProvider` proporciona un objeto `RowSet` desplazable que puede enlazarse con un componente **Tabla** para mostrar los datos del objeto `RowSet`. Este proveedor de datos es una envoltura para

un objeto `CachedRowSet`. Si hace clic en el elemento `addressesDataProvider` en la ventana **Esquema**, podrá ver en la ventana **Propiedades** que su propiedad `cachedRowSet` se estableció en `addressesRowSet`, un objeto que implementa a la interfaz `CachedRowSet`.

Paso 6: Modificar la instrucción SQL de `addressesRowSet`

El objeto `CachedRowSet` envuelto por nuestro objeto `addressesDataProvider` está configurado de manera predeterminada para ejecutar una consulta SQL que seleccione todos los datos en la tabla `Direcciones` de la base de datos `LibretaDirecciones`. Para editar esta consulta SQL, puede expandir el nodo `SessionBean` en la ventana **Esquema** y hacer doble clic en el elemento `addressesRowSet` para abrir la ventana del editor de consultas (figura 27.5). Nos gustaría editar la instrucción SQL de manera que los registros con apellidos duplicados se ordenen por apellido, y después por primer nombre. Para ello, haga clic en la columna **Tipo de orden** enseguida de la fila `LASTNAME` y seleccione **Ascendente**. Después, repita esto para la fila `FIRSTNAME`. Observe que la expresión

```
ORDER BY JHTP7.ADDRESSES.LASTNAME ASC,
        JHTP7.ADDRESSES.FIRSTNAME ASC
```

se agregó a la instrucción SQL al final del editor.

Paso 7: Agregar validación

Es importante validar los datos del formulario en esta página, para asegurar que los datos puedan insertarse correctamente en la base de datos `LibretaDirecciones`. Todas las columnas de la base de datos son de tipo `varchar` y tienen restricciones de longitud. Por esta razón, debemos agregar un **Validador de longitud** a cada componente **Campo de texto**, o establecer la propiedad `maxLength` de cada componente **Campo de texto**. Optamos por establecer la propiedad `maxLength` de cada uno. Los componentes **Campo de texto** del primer nombre, apellido paterno, calle, ciudad, estado y código postal no pueden exceder a 20, 30, 100, 30, 2 y 5 caracteres, respectivamente.

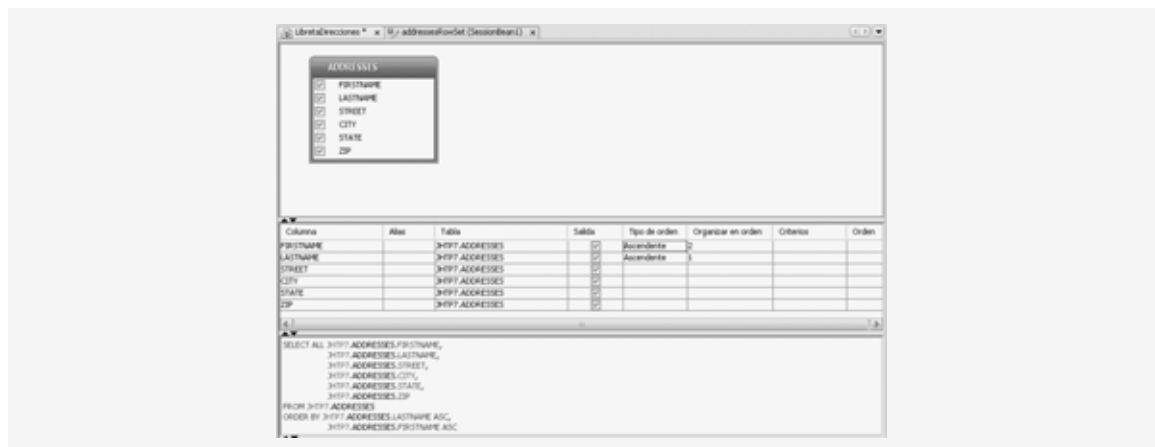


Figura 27.5 | Edición de la instrucción SQL de `addressesRowSet`.

Por último, arrastre un componente **Grupo de mensaje** a su página, a la derecha de la **Tabla**. Un componente **Grupo de mensaje** muestra mensajes del sistema. Utilizamos este componente para mostrar un mensaje de error cuando falla un intento de contactarse con la base de datos. Establezca la propiedad `showGlobalOnly` del **Grupo de mensaje** a `true`, para evitar que se muestren aquí mensajes de error de validación a nivel de componente.

Archivo JSP para una página Web que interactúa con una base de datos

El archivo JSP para la aplicación se muestra en la figura 27.6. Este archivo contiene una gran cantidad de marcado generado para los componentes que vimos en el capítulo 26. En este ejemplo sólo hablaremos sobre el marcado para los componentes que son nuevos.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <!-- Fig. 27.6: LibretaDirecciones.jsp -->
4  <!-- JSP de LibretaDirecciones con un formulario para agregar y un componente JSF Tabla
   -->
5
6  <jsp:root version="1.2" xmlns:f="http://java.sun.com/jsf/core"
7      xmlns:h="http://java.sun.com/jsf/html" xmlns:jsp=
8      "http://java.sun.com/JSP/Page" xmlns:ui="http://www.sun.com/web/ui">
9      <jsp:directive.page contentType="text/html; charset=UTF-8"
10         pageEncoding="UTF-8"/>
11      <f:view>
12          <ui:page binding="#{LibretaDirecciones.page1}" id="page1">
13              <ui:html binding="#{LibretaDirecciones.html1}" id="html1">
14                  <ui:head binding="#{LibretaDirecciones.head1}" id="head1">
15                      <ui:link binding="#{LibretaDirecciones.link1}" id="link1"
16                          url="/resources/stylesheet.css"/>
17                  </ui:head>
18                  <ui:body binding="#{LibretaDirecciones.body1}" id="body1"
19                      style="-rave-layout: grid">
20                      <ui:form binding="#{LibretaDirecciones.form1}" id="form1">
21                          <ui:staticText binding="#{LibretaDirecciones.staticText1}" id=
22                              "staticText1" style="font-size: 18px; left: 12px;
23                              top: 24px; position: absolute"
24                              text="Agregar un contacto a la libreta de direcciones:"/>
25                          <ui:textField binding="#{LibretaDirecciones.pnombreCampoTexto}"
26                              id="pnombreCampoTexto" maxLength="20" required="true"
27                              style="left: 132px; top: 72px;
28                              position: absolute"/>
29                          <ui:textField binding="#{LibretaDirecciones.apaternoCampoTexto}"
30                              id="apaternoCampoTexto" maxLength="30" required="true"
31                              style="left: 504px; top: 72px; position: absolute;
32                              width: 228px"/>
33                          <ui:textField binding="#{LibretaDirecciones.calleCampoTexto}"
34                              id="calleCampoTexto" maxLength="100" required="true"
35                              style="left: 132px; top: 96px; position: absolute;
36                              width: 600px"/>
37                          <ui:textField binding="#{LibretaDirecciones.ciudadCampoTexto}"
38                              id="ciudadCampoTexto" maxLength="30" required="true"
39                              style="left: 132px; top: 120px; position: absolute; width: 264px"/>
40                          <ui:textField binding="#{LibretaDirecciones.estadoCampoTexto}"
41                              id="estadoCampoTexto" maxLength="2" required="true"
42                              style="left: 480px; top: 120px; position: absolute;
43                              width: 60px"/>
44                          <ui:textField binding="#{LibretaDirecciones.cpCampoTexto}"
45                              id="cpCampoTexto" maxLength="5" required="true"
46                              style="left: 672px; top: 120px; position: absolute;
47                              width: 60px"/>
48                          <ui:label binding="#{LibretaDirecciones.pnombreEtiqueta}" for=
49                              "pnombreCampoTexto" id="pnombreEtiqueta" style="position: absolute"
50                              "left: 12px; top: 72px; text="Primer nombre:"/>
51                          <ui:label binding="#{LibretaDirecciones.apaternoEtiqueta}" for=
52                              "apaternoCampoTexto" id="apaternoEtiqueta" style="position: absolute;
53                              left: 384px; top: 72px" text="Apellido Paterno:"/>
54                          <ui:label binding="#{LibretaDirecciones.calleEtiqueta}" for=
55                              "calleCampoTexto" id="calleEtiqueta" style="position: absolute"
56                              "left: 12px; top: 96px; text="Calle:"/>
57                          <ui:label binding="#{LibretaDirecciones.ciudadEtiqueta}" for=
58                              "ciudadCampoTexto" id="ciudadEtiqueta" style="left: 12px;

```

Figura 27.6 | JSP de LibretaDirecciones con un formulario para agregar y un componente JSF Tabla. (Parte I de 4).

```

59         top: 120px; position: absolute" text="Ciudad:"/>
60     <ui:label binding="#{LibretaDirecciones.estadoEtiqueta}" for=
61         "estadoCampoTexto" id="estadoEtiqueta" style= position: absolute"
62         "left: 408px; top: 120px; text="Estado:"/>
63     <ui:label binding="#{LibretaDirecciones.cpEtiqueta}" for=
64         "cpCampoTexto" id="cpEtiqueta" style="height: 22px; left: 552px;
65         top: 120px; position: absolute; width: 94px" text="Código postal:"/>
66     <ui:button action="#{LibretaDirecciones.enviarBoton_action}"
67         binding="#{LibretaDirecciones.enviarBoton}" id=
68         "enviarBoton" primary="true" style= position: absolute"
69         "left: 131px; top: 168px; text="Enviar"/>
70     <ui:button binding="#{LibretaDirecciones.borrarBoton}" id=
71         "borrarBoton" reset="true" style="left: 251px; top:
72         168px; position: absolute" text="Borrar"/>
73     <ui:table augmentTitle="false" binding=
74         "#{LibretaDirecciones.direccionesTabla}" id="direccionesTabla"
75         paginationControls="rue" style="left: 12px; top: 204px;
76         position: absolute; width: 720px"
77         title="Contactos" width="720">
78     <script><![CDATA[
79     <!--Las líneas 79 a 140 contienen código de JavaScript que se eliminó para ahorrar espacio.
80     El código fuente completo se proporciona en la carpeta de este ejemplo. -->
141     ]]]></script>
142     <ui:tableRowGroup binding=
143         "#{LibretaDirecciones.tableRowGroup1}" id=
144         "tableRowGroup1" rows="5" sourceData=
145         "#{LibretaDirecciones.addressesDataProvider}"
146         sourceVar="currentRow">
147         <ui:tableColumn binding=
148             "#{LibretaDirecciones.pnombreColumna}" headerText=
149             "Primer nombre" id="pnombreColumna" sort=
150             "ADDRESSES.FIRSTNAME">
151             <ui:staticText binding=
152                 "#{LibretaDirecciones.staticText2}" id=
153                 "staticText2" text="#{currentRow.value[
154                 'ADDRESSES.FIRSTNAME']}" />
155             </ui:tableColumn>
156             <ui:tableColumn binding=
157                 "#{LibretaDirecciones.apaternoColumna}" headerText=
158                 "Apellido paterno" id="apaternoColumna"
159                 sort="ADDRESSES.LASTNAME">
160                 <ui:staticText binding=
161                     "#{LibretaDirecciones.staticText3}" id=
162                     "staticText3" text="#{currentRow.value[
163                     'ADDRESSES.LASTNAME']}" />
164                 </ui:tableColumn>
165                 <ui:tableColumn binding=
166                     "#{LibretaDirecciones.calleColumna}" headerText=
167                     "Calle" id="calleColumna"
168                     sort="ADDRESSES.STREET">
169                     <ui:staticText binding=
170                         "#{LibretaDirecciones.staticText4}" id=
171                         "staticText4" text="#{currentRow.value[
172                         'ADDRESSES.STREET']}" />
173                     </ui:tableColumn>
174                     <ui:tableColumn binding=
175                         "#{LibretaDirecciones.ciudadColumna}" headerText="Ciudad"
176                         id="ciudadColumna" sort="ADDRESSES.CITY">
177                     <ui:staticText binding=

```

Figura 27.6 | JSP de LibretaDirecciones con un formulario para agregar y un componente JSF **Tabla**. (Parte 2 de 4).


```

178         "#{LibretaDirecciones.staticText5}" id="staticText5"
179         text="#{currentRow.value[
180         'ADDRESSES.CITY']}" />
181     </ui:tableColumn>
182     <ui:tableColumn binding=
183         "#{LibretaDirecciones.estadoColumna}" headerText="Estado"
184         id="estadoColumna" sort="ADDRESSES.STATE">
185         <ui:staticText binding=
186             "#{LibretaDirecciones.staticText6}" id=
187             "staticText6" text="#{currentRow.value[
188             'ADDRESSES.STATE']}" />
189         </ui:tableColumn>
190     <ui:tableColumn binding=
191         "#{LibretaDirecciones.cpColumna}" headerText="CP"
192         id="cpColumna" sort="ADDRESSES.ZIP">
193         <ui:staticText binding=
194             "#{LibretaDirecciones.staticText7}" id="staticText7"
195             text="#{currentRow.value[
196             'ADDRESSES.ZIP']}" />
197         </ui:tableColumn>
198     </ui:tableRowGroup>
199 </ui:table>
200 <ui:messageGroup binding="#{LibretaDirecciones.messageGroup1}"
201     id="messageGroup1" showGlobalOnly="true" style=
202     "left: 744px; top: 204px; position: absolute" />
203 </ui:form>
204 </ui:body>
205 </ui:html>
206 </ui:page>
207 </f:view>
208 </jsp:root>

```

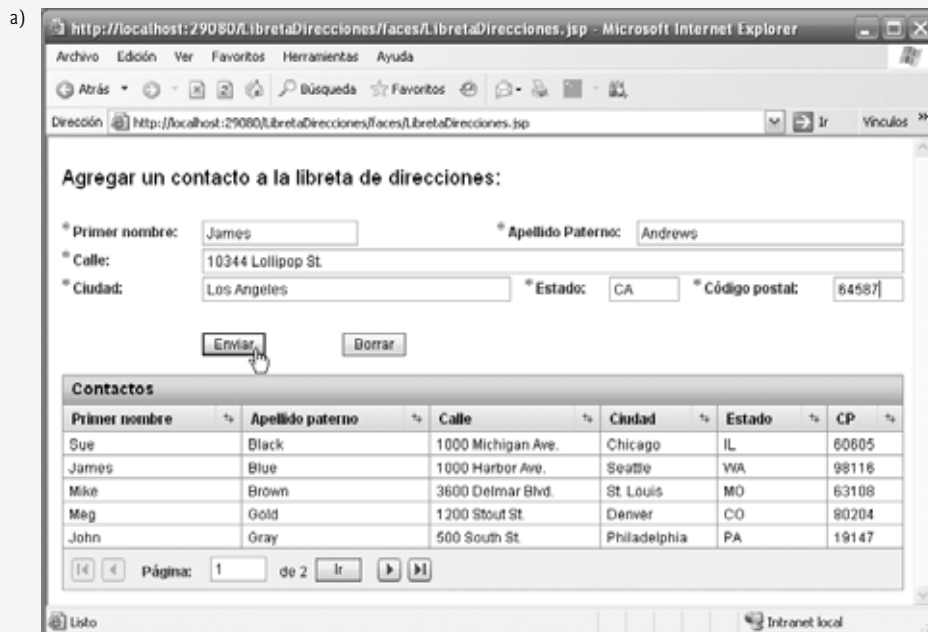


Figura 27.6 | JSP de LibretaDirecciones con un formulario para agregar y un componente JSF **Tabla**. (Parte 3 de 4).

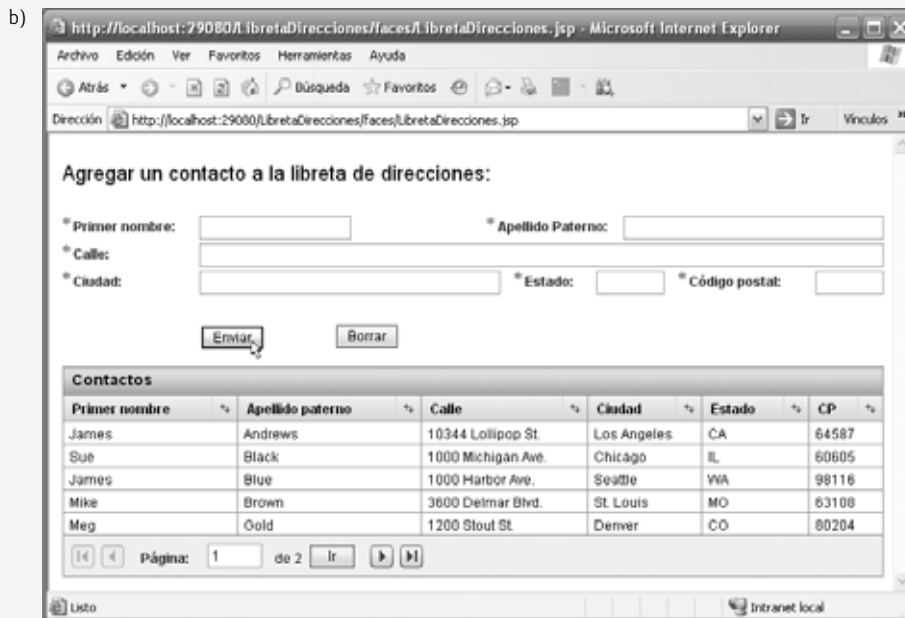


Figura 27.6 | JSP de LibretaDirecciones con un formulario para agregar y un componente JSF **Tabla**. (Parte 4 de 4).

Las líneas 21 a 72 contienen los componentes JSF que conforman el formulario que recopila la entrada del usuario. En las líneas 73 a 199 se define el elemento **Tabla** (ui:table) que muestra la información de las direcciones de la base de datos. Las líneas 79 a 140 (que no se muestran aquí) contienen funciones de JavaScript generadas por el IDE para manejar las acciones de **Tabla**, como un cambio en el estado de la fila actual. Los componentes JSF **Tabla** pueden tener varios grupos de filas que muestren distintos datos. Esta **Tabla** tiene un solo elemento ui:tableRowGroup con una marca inicial en las líneas 142 a 146. El atributo sourceData del grupo de filas está enlazado a nuestro objeto addressesDataProvider y recibe el nombre de variable currentRow. El grupo de filas también define las columnas de la **Tabla**. Cada elemento ui:tableColumn contiene un elemento ui:staticText con su atributo text enlazado con una columna en la fila actual (currentRow) del proveedor de datos. Estos elementos ui:staticText permiten a la **Tabla** mostrar los datos de cada fila.

Bean de sesión para la aplicación LibretaDirecciones

En la figura 27.7 muestra el archivo SessionBean1.java generado por Java Studio Creator 2 para la aplicación LibretaDirecciones. El objeto CachedRowSet que utiliza el proveedor de datos del componente **Tabla** para acceder a la base de datos LibretaDirecciones es una propiedad de esta clase (líneas 31 a 41).

```

1 // Fig. 27.7: SessionBean1.java
2 // Bean de sesión que inicializa el origen de datos para la
3 // base de datos LibretaDirecciones.
4 package libretadirecciones;
5
6 import com.sun.rave.web.ui.appbase.AbstractSessionBean;
7 import javax.faces.FacesException;
8 import com.sun.sql.rowset.CachedRowSetXImpl;
9
10 public class SessionBean1 extends AbstractSessionBean
11 {

```

Figura 27.7 | Bean de sesión que inicializa el origen de datos para la base de datos LibretaDirecciones. (Parte 1 de 2).

```

12     private int __placeholder;
13
14     private void _init() throws Exception
15     {
16         addressesRowSet.setDataSourceName(
17             "java:comp/env/jdbc/LibretaDirecciones" );
18         addressesRowSet.setCommand(
19             "SELECT ALL JHTP7.ADDRESSES.FIRSTNAME," +
20             "\nJHTP7.ADDRESSES.LASTNAME," +
21             "\nJHTP7.ADDRESSES.STREET," +
22             "\nJHTP7.ADDRESSES.CITY," +
23             "\nJHTP7.ADDRESSES.STATE," +
24             "\nJHTP7.ADDRESSES.ZIP" +
25             "\nFROM JHTP7.ADDRESSES" +
26             "\nORDER BY JHTP7.ADDRESSES.LASTNAME ASC," +
27             "\nJHTP7.ADDRESSES.FIRSTNAME ASC " );
28         addressesRowSet.setTableName( "ADDRESSES" );
29     } // fin del método _init
30
31     private CachedRowSetXImpl addressesRowSet = new CachedRowSetXImpl();
32
33     public CachedRowSetXImpl getAddressesRowSet()
34     {
35         return addressesRowSet;
36     }
37
38     public void setAddressesRowSet(CachedRowSetXImpl crsxi)
39     {
40         this.addressesRowSet = crsxi;
41     }
42
43     // Las líneas 43 a 76 del código generado en forma automática se eliminaron para
44     // ahorrar espacio.
45     // El código fuente completo se proporciona en la carpeta de este ejemplo.
46 } // fin de la clase SessionBean1

```

Figura 27.7 | Bean de sesión que inicializa el origen de datos para la base de datos *LibretaDirecciones*. (Parte 2 de 2).

El método `_init` (líneas 14 a 29) configura a `addressesRowSet` para que interactúe con la base de datos *LibretaDirecciones* (líneas 16 a 27). En las líneas 16 y 17 se conecta el conjunto de filas con la base de datos. En las líneas 18 a 27 se establece el comando SQL de `addressesRowSet` con la consulta configurada en la figura 27.5.

27.2.2 Modificación del archivo de bean de página para la aplicación *LibretaDirecciones*

Después de crear la página Web y configurar los componentes utilizados en este ejemplo, haga doble clic en el botón **Enviar** para crear un manejador de eventos de acción para este botón en el archivo de bean de página. El código para insertar un contacto en la base de datos se colocará en este método. El bean de página con el manejador de eventos completo se muestra en la figura 27.8 a continuación.

Las líneas 534 a 573 contienen el código para manejar los eventos del botón **Enviar**. En la línea 536 se determina si se puede anexar una nueva fila al proveedor de datos. De ser así, se anexa una nueva fila en la línea 540. Cada fila en un objeto `CachedRowSetDataProvider` tiene su propia clave; el método `appendRow` devuelve la clave para la nueva fila. En la línea 541 se establece el cursor del proveedor de datos a la nueva fila, de manera que cualquier modificación que realicemos al proveedor de datos afecte a esa fila. En las líneas 543 a 554 se establece cada una de las columnas de la fila a los valores introducidos por el usuario en los correspondientes componentes **Campo de texto**. En la línea 555 se almacena el nuevo contacto, llamando al método `commitChanges` de la clase `CachedRowSetDataProvider` para insertar la nueva fila en la base de datos *LibretaDirecciones*.

```

1 // Fig. 27.8: LibretaDirecciones.java
2 // Bean de página para agregar un contacto a la libreta de direcciones.
3 package libretadirecciones;
4
5 import com.sun.data.provider.RowKey;
6 import com.sun.rave.web.ui.appbase.AbstractPageBean;
7 import com.sun.rave.web.ui.component.Body;
8 import com.sun.rave.web.ui.component.Form;
9 import com.sun.rave.web.ui.component.Head;
10 import com.sun.rave.web.ui.component.Html;
11 import com.sun.rave.web.ui.component.Link;
12 import com.sun.rave.web.ui.component.Page;
13 import javax.faces.FacesException;
14 import com.sun.rave.web.ui.component.StaticText;
15 import com.sun.rave.web.ui.component.TextField;
16 import com.sun.rave.web.ui.component.Label;
17 import com.sun.rave.web.ui.component.Button;
18 import com.sun.rave.web.ui.component.Table;
19 import com.sun.rave.web.ui.component.TableRowGroup;
20 import com.sun.rave.web.ui.component.TableColumn;
21 import com.sun.data.provider.impl.CachedRowSetDataProvider;
22 import com.sun.rave.web.ui.component.MessageGroup;
23
24 public class LibretaDirecciones extends AbstractPageBean
25 {
26     private int __placeholder;
27
28     private void _init() throws Exception
29     {
30         addressesDataProvider.setCachedRowSet(
31             (javax.sql.rowset.CachedRowSet)
32             getValue("#{SessionBean1.addressesRowSet}"));
33         direccionesTabla.setInternalVirtualForm(true);
34     }
35
36     // Las líneas 36 a 521 del código generado en forma automática se eliminaron para
37     // ahorrar espacio.
38     // El código fuente completo se proporciona en la carpeta de este ejemplo.
39
522 public void prerender()
523 {
524     addressesDataProvider.refresh();
525 } // fin del método prerender
526
527 public void destroy()
528 {
529     addressesDataProvider.close();
530 } // fin del método destroy
531
532 // manejador de acciones que agrega un contacto a la base de datos LibretaDirecciones
533 // cuando el usuario hace clic en el botón Enviar
534 public String enviarBoton_action()
535 {
536     if ( addressesDataProvider.canAppendRow() )
537     {
538         try
539         {
540             RowKey rk = addressesDataProvider.appendRow();
541             addressesDataProvider.setCursorRow(rk);

```

Figura 27.8 | Bean de página para agregar un contacto a la libreta de direcciones. (Parte 1 de 2).

```

542         addressesDataProvider.setValue( "ADDRESSES.FIRSTNAME",
543             pnombreCampoTexto.getValue() );
544         addressesDataProvider.setValue( "ADDRESSES.LASTNAME",
545             apaternoCampoTexto.getValue() );
546         addressesDataProvider.setValue( "ADDRESSES.STREET",
547             calleCampoTexto.getValue() );
548         addressesDataProvider.setValue( "ADDRESSES.CITY",
549             ciudadCampoTexto.getValue() );
550         addressesDataProvider.setValue( "ADDRESSES.STATE",
551             estadoCampoTexto.getValue() );
552         addressesDataProvider.setValue( "ADDRESSES.ZIP",
553             cpCampoTexto.getValue());
554         addressesDataProvider.commitChanges();
555
556         // restablece los campos de texto
557         apaternoCampoTexto.setValue( "" );
558         pnombreCampoTexto.setValue( "" );
559         calleCampoTexto.setValue( "" );
560         ciudadCampoTexto.setValue( "" );
561         estadoCampoTexto.setValue( "" );
562         cpCampoTexto.setValue( "" );
563     } // fin de try
564     catch ( Exception ex )
565     {
566         error( "No se actualizo la libreta de direcciones. " +
567             ex.getMessage() );
568     } // fin de catch
569 } // fin de if
570
571 return null;
572 } // fin del método enviarBoton_action
573 } // fin de la clase LibretaDirecciones
574

```

Figura 27.8 | Bean de página para agregar un contacto a la libreta de direcciones. (Parte 2 de 2).

En las líneas 558 a 563 se borran todos los componentes **Campo de texto** del formulario. Si se omiten estas líneas, los campos retendrán sus valores actuales una vez que se actualice la base de datos y se vuelva a cargar la página. Además, el botón **Borrar** no trabajará en forma apropiada si no se borran los componentes **Campo de texto**. En vez de vaciar los componentes **Campo de texto**, los restablecerá a los valores que contenían la última vez que se envió el formulario.

En las líneas 565 a 569 se atrapan las excepciones que podrían ocurrir mientras se realiza la actualización de la base de datos `LibretaDirecciones`. En las líneas 567 y 568 se muestra un mensaje indicando que la base de datos no se actualizó, así como el mensaje de error de la excepción, en el componente `MessageGroup` de la página.

En el método `prerender`, en la línea 524 se hace una llamada al método `refresh` de `CachedRowSetDataProvider`. Esto vuelve a ejecutar la instrucción SQL del objeto `CachedRowSet` envuelto y se vuelven a ordenar las filas de la **Tabla**, de manera que la nueva fila se muestre en el orden apropiado. Si no se hace la llamada a `refresh`, la nueva dirección se mostrará al final de la **Tabla** (ya que anexamos la nueva fila al final del proveedor de datos). El IDE generó código automáticamente para liberar los recursos utilizados por el proveedor de datos (línea 529) en el método `destroy`.

27.3 Componentes JSF habilitados para Ajax

El término **Ajax** (**JavaScript y XML asíncronos**) fue ideado por Jesse James Garrett de Adaptive Path, Inc. en febrero de 2005, para describir un rango de tecnologías para desarrollar aplicaciones Web dinámicas y con gran capacidad de respuesta. Las aplicaciones Ajax incluyen Google Maps, Flickr de Yahoo y muchas más. Ajax separa la parte correspondiente a la interacción con el usuario de una aplicación, de la interacción con su servidor, permitiendo que ambas procedan en forma asíncrona y en paralelo. Esto permite a las aplicaciones Ajax basadas en

Web ejecutarse a velocidades que se asemejan a las de las aplicaciones de escritorio, con lo cual se reduce (o incluso se elimina) la tradicional ventaja de rendimiento que han tenido las aplicaciones de escritorio en comparación con las aplicaciones Web. Esto implica enormes ramificaciones para la industria de las aplicaciones de escritorio; la plataforma preferida para las aplicaciones está empezando a cambiar, del escritorio a la Web. Muchas personas creen que la Web (en especial, dentro del contexto del abundante software de código fuente abierto, las computadoras económicas y el explosivo incremento en el ancho de banda de Internet) creará la siguiente fase principal de crecimiento para las compañías de Internet.

Ajax realiza llamadas asíncronas al servidor para intercambiar pequeñas cantidades de datos con cada llamada. En donde, por lo general, la página completa se enviaría y se volvería a cargar con cada interacción del usuario en una página Web, Ajax permite que se vuelvan a cargar sólo las porciones necesarias de la página, lo cual ahorra tiempo y recursos.

Las aplicaciones Ajax contienen marcado de XHTML y CSS al igual que cualquier otra página Web, y hacen uso de las tecnologías de secuencias de comandos del lado cliente (como JavaScript) para interactuar con los elementos de las páginas. El objeto `XMLHttpRequestObject` permite los intercambios asíncronos con el servidor Web, lo cual hace que las aplicaciones Ajax tengan una gran capacidad de respuesta. Este objeto se puede utilizar en la mayoría de los lenguajes de secuencias de comandos para pasar datos XML del cliente al servidor, y para procesar los datos XML que el servidor envía de vuelta al cliente.

Aunque el uso de las tecnologías Ajax en las aplicaciones Web puede mejorar en forma considerable el rendimiento, la programación en Ajax es compleja y propensa a errores. Los diseñadores de páginas requieren conocer tanto los lenguajes de secuencias de comandos como los lenguajes de marcado. Las **bibliotecas Ajax** facilitan el proceso de aprovechar los beneficios de Ajax en las aplicaciones Web, sin tener que escribir Ajax “puro”. Estas bibliotecas proporcionan elementos de página habilitados para Ajax, que pueden incluirse en las páginas Web con sólo agregar al marcado de la página las etiquetas definidas en la biblioteca. Limitaremos nuestra discusión acerca de cómo crear aplicaciones Ajax al uso de una de esas bibliotecas en Java Studio Creator 2.

27.3.1 Biblioteca de componentes Java BluePrints

La biblioteca de componentes **Java BluePrints** de Ajax proporciona **componentes JSF habilitados para Ajax**. Estos componentes dependen de la tecnología Ajax para brindar la sensación y capacidad de respuesta de una aplicación de escritorio a través de la Web. En la figura 27.9 se muestra un resumen del conjunto actual de componentes que podemos descargar y usar con Java Studio Creator 2. En las siguientes dos secciones demostraremos los componentes **AutoComplete Text Field** y **Map Viewer**.

Componente	Descripción
AutoComplete Text Field	Hace peticiones Ajax para mostrar una lista de sugerencias, a medida que el usuario escribe en el campo de texto.
Buy Now Button	Inicia una transacción a través del sitio Web PayPal.
Map Viewer	Usa la API de Google Maps para mostrar un mapa que permite inclinaciones, acercamientos y puede mostrar marcadores para las ubicaciones de interés.
Popup Calendar	Proporciona un calendario que permite a un usuario desplazarse entre los meses y años. Llena un Campo de texto con una fecha con formato cuando el usuario selecciona un día.
Progress Bar	Muestra en forma visual el progreso de una operación que tarda cierto tiempo en ejecutarse. Usa un cálculo suministrado por el programador para determinar el porcentaje de progreso.
Rating	Proporciona una barra de calificación personalizable de cinco estrellas, que puede mostrar mensajes a medida que el usuario mueve el ratón sobre las calificaciones.
Rich Textarea Editor	Proporciona un área de texto editable, que permite al usuario aplicar formato al texto con fuentes, colores, hipervínculos y fondos.
Select Value Text Field	Muestra una lista de sugerencias en una lista desplegable a medida que el usuario escribe, de manera similar al componente AutoComplete Text Field .

Figura 27.9 | Componentes habilitados para Ajax, proporcionados por la biblioteca de componentes BluePrints de Ajax.

Para utilizar los componentes Java BluePrints habilitados para Ajax en Java Studio Creator 2, debemos descargarlos e importarlos. El IDE proporciona un asistente para instalar este grupo de componentes. Para usarlo, seleccione **Herramientas > Centro de actualización** para mostrar el cuadro de diálogo **Asistente del centro de actualización**. Haga clic en **Siguiente >** para buscar actualizaciones disponibles. En el área **Nuevos módulos y actualizaciones disponibles** del cuadro de diálogo, seleccione **BluePrints AJAX Components** y haga clic con el botón derecho del ratón en el botón de flecha (>) para agregarlo a la lista de elementos que desea instalar. Haga clic en **Siguiente >** y siga los indicadores para aceptar las condiciones de uso y descargar los componentes. Cuando se complete la descarga, haga clic en **Siguiente >** y luego en **Terminar**. Haga clic en **Aceptar** para reiniciar el IDE.

A continuación, debe importar los componentes en la **Paleta**. Seleccione **Herramientas > Administrador de bibliotecas de componentes** y después haga clic en **Importar...** Haga clic en el botón **Examinar...** en el cuadro de diálogo **Importar biblioteca de componentes** que aparezca. Seleccione el archivo `ui.compLib` y haga clic en **Abrir**. Haga clic en **Aceptar** para importar los componentes **BluePrints AJAX Components** y **BluePrints AJAX SupportBeans**. Cierre el **Administrador de bibliotecas de componentes** para regresar al IDE.

Ahora deberá ver dos nuevos nodos en la parte inferior de la **Paleta**. El primero, **BluePrints AJAX Components**, proporciona los ocho componentes que se enlistan en la figura 27.9. El segundo, **BluePrints AJAX Support Beans**, incluye componentes que ofrecen soporte a los componentes Ajax. Ahora puede crear aplicaciones Web Ajax de alto rendimiento con sólo arrastrar, soltar y configurar las propiedades de los componentes, de igual forma que con los demás componentes en la **Paleta**.

27.4 AutoComplete Text Field y formularios virtuales

Vamos a demostrar el componente **AutoComplete Text Field** del catálogo BluePrints; para ello, hay que agregar un nuevo formulario a nuestra aplicación **LibretaDirecciones**. El componente **AutoComplete Text Field** proporciona una lista de sugerencias a medida que el usuario escribe. Obtiene las sugerencias de un origen de datos, que puede ser una base de datos o un servicio Web. En un momento dado, el nuevo formulario permitirá a los usuarios buscar en la libreta de direcciones por apellido paterno, y después por primer nombre. Si el usuario selecciona un contacto, la aplicación mostrará el nombre y la dirección de ese contacto en un mapa del vecindario. Vamos a crear este formulario en dos etapas. Primero, agregaremos el componente **AutoComplete Text Field** que mostrará las sugerencias a medida que el usuario escriba el apellido paterno de un contacto. Después agregaremos la funcionalidad de búsqueda y, en el tercer paso, la visualización de un mapa.

*Agregar componentes de búsqueda a la página **LibretaDirecciones.jsp***

Utilice la aplicación **LibretaDirecciones** de la sección 27.2; suelte un componente **Texto estático** llamado **encabezadoBusqueda** debajo de **direccionesTabla**. Cambie su texto a "Buscar en la libreta de direcciones por apellido:" y cambie el tamaño de su fuente a 18 px. Ahora arrastre un componente **AutoComplete Text Field** a la página y nómbrelo **nombreAutoComplete**. Establezca la propiedad **required** de este campo en **true**. Agregue una **Etiqueta** llamada **buscarNombreEtiqueta** que contenga el texto "ApellidoPaterno:" a la izquierda del componente **AutoComplete Text Field**. Por último, agregue un botón llamado **buscarBoton** con el texto **Buscar** a la derecha del componente **AutoComplete Text Field**.

27.4.1 Configuración de los formularios virtuales

Los formularios virtuales se utilizan cuando deseamos que un botón envíe un subconjunto de los campos de entrada de la página al servidor. Recuerde que los formularios virtuales internos de **Tabla** estaban habilitados, para que al hacer clic en los botones de paginación no se enviaran los datos de los componentes **Campo de texto** utilizados para agregar un contacto a la base de datos **LibretaDirecciones**. Los formularios virtuales son especialmente útiles para mostrar varios formularios en la misma página. Nos permiten especificar un **emisor** y uno o más **participantes** para un formulario. Cuando se hace clic en el componente emisor del formulario virtual, sólo se enviarán al servidor los valores de sus componentes participantes. Utilizamos formularios virtuales en nuestra aplicación **LibretaDirecciones** para separar el formulario para agregar un contacto a la base de datos **LibretaDirecciones** del formulario para buscar en la base de datos.

Para agregar formularios virtuales a la página, haga clic con el botón derecho en el botón **Enviar** que se encuentra en el formulario superior, y seleccione **Configurar formularios virtuales...** en el menú contextual para que aparezca el cuadro de diálogo **Configurar formularios virtuales**. Haga clic en **Nuevo** para agregar un formulario virtual; después haga clic en la columna **Nombre** y cambie el nombre del nuevo formulario a **agregarForm**. Haga doble clic en la columna **Enviar** y cambie la opción a **Sí** para indicar que este botón se debe utilizar para

enviar el formulario virtual agregarForm. Haga clic en **Aceptar** para salir del cuadro de diálogo. Después, seleccione todos los componentes **Campo de texto** utilizados para introducir la información de un contacto en el formulario superior. Para ello, mantenga oprimida la tecla *ctrl*. Mientras hace clic en cada **Campo de texto**. Haga clic con el botón derecho del ratón en uno de los componentes **Campo de texto** seleccionados y elija la opción **Configurar formularios virtuales....** En la columna **Función** del formulario agregarForm, cambie la opción a **Sí** para indicar que los valores en estos componentes **Campo de texto** deben enviarse al servidor cuando se envíe el formulario. En la figura 27.10 se muestra el cuadro de diálogo **Configurar formularios virtuales**, después de haber agregado ambos formularios virtuales. Haga clic en **Aceptar** para salir.

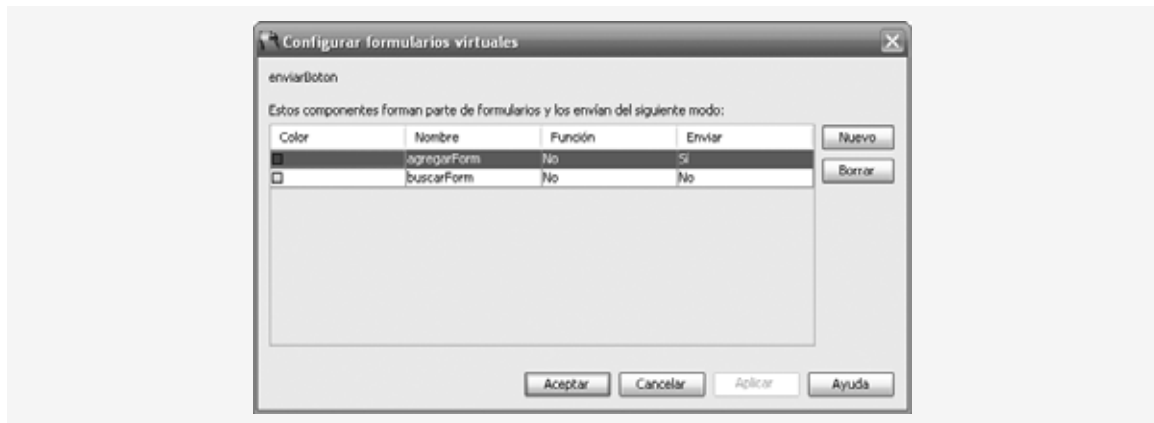


Figura 27.10 | Cuadro de diálogo **Configurar formularios virtuales**.

Repita el proceso antes descrito para crear un segundo formulario virtual llamado buscarForm para el formulario inferior. El botón **Buscar** deberá enviar el formulario buscarForm, y nombreAutoComplete deberá participar en este formulario. Después, regrese al modo **Diseño** y haga clic en el botón **Mostrar formularios virtuales** () en la parte superior del panel Diseñador visual para mostrar una leyenda de los formularios virtuales en la página. Sus formularios virtuales deberán estar configurados como en la figura 27.11. Los componentes **Campo**



Figura 27.11 | Leyenda para los formularios virtuales.

de texto con el contorno de color azul participan en el formulario virtual `agregarForm`. Los componentes con el contorno de color verde participan en el formulario virtual `buscarForm`. Los componentes con el contorno de línea punteada envían sus respectivos formularios. Se proporciona una clave de colores en la parte inferior derecha del área de **Diseño**, para que usted sepa cuáles componentes pertenecen a cada formulario virtual.

27.4.2 Archivos JSP con formularios virtuales y un AutoComplete Text Field

La figura 27.12 presenta el archivo JSP generado por Java Studio Creator 2 para esta etapa de la aplicación `LibretaDirecciones`. Observe que se especifica una nueva biblioteca de etiquetas en el elemento raíz (`xmlns:bp="http://java.sun.com/blueprints/ui/14"`; línea 6). Ésta es la biblioteca del catálogo de BluePrints que proporciona los componentes habilitados para Ajax, como el componente **AutoComplete Text Field**. Sólo nos enfocaremos en las nuevas características de esta JSP.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <!-- Fig. 27.12: LibretaDirecciones.jsp -->
4  <!-- JSP de LibretaDirecciones con un componente AutoComplete Text Field -->
5
6  <jsp:root version="1.2" xmlns:bp="http://java.sun.com/blueprints/ui/14"
7    xmlns:f="http://java.sun.com/jsf/core" xmlns:h=
8    "http://java.sun.com/jsf/html" xmlns:jsp="http://java.sun.com/JSP/Page"
9    xmlns:ui="http://www.sun.com/web/ui">
10   <jsp:directive.page contentType="text/html; charset=UTF-8"
11     pageEncoding="UTF-8"/>
12   <f:view>
13     <ui:page binding="#{LibretaDirecciones.page1}" id="page1">
14       <ui:html binding="#{LibretaDirecciones.html1}" id="html1">
15         <ui:head binding="#{LibretaDirecciones.head1}" id="head1">
16           <ui:link binding="#{LibretaDirecciones.link1}" id="link1"
17             url="/resources/styleSheet.css"/>
18         </ui:head>
19         <ui:body binding="#{LibretaDirecciones.body1}" id="body1"
20           style="-rave-layout: grid">
21           <ui:form binding="#{LibretaDirecciones.form1}" id="form1"
22             virtualFormsConfig="agregarForm | apaternoCampoTexto
23             pnombreCampoTexto calleCampoTexto estadoCampoTexto
24             cpCampoTexto ciudadCampoTexto | enviarBoton , buscarForm
25             | nombreAutoComplete | buscarBoton">
26             <ui:staticText binding="#{LibretaDirecciones.staticText1}" id=
27               "staticText1" style="font-size: 18px; left: 12px;
28               top: 24px; position: absolute" text=
29               "Agregar un contacto a la libreta de direcciones:"/>
30             <ui:textField binding="#{LibretaDirecciones.pnombreCampoTexto}"
31               id="pnombreCampoTexto" maxLength="20" required="true"
32               style="left: 132px; top: 72px;
33               position: absolute"/>
34             <ui:textField binding="#{LibretaDirecciones.apaternoCampoTexto}"
35               id="apaternoCampoTexto" maxLength="30" required="true"
36               style="left: 504px; top: 72px; position: absolute;
37               width: 228px"/>
38             <ui:textField binding="#{LibretaDirecciones.calleCampoTexto}"
39               id="calleCampoTexto" maxLength="100" required="true"
40               style="left: 132px; top: 96px; position: absolute;
41               width: 600px"/>
42             <ui:textField binding="#{LibretaDirecciones.ciudadCampoTexto}"
43               id="ciudadCampoTexto" maxLength="30" required="true"
44               style="left: 132px; top: 120px; position: absolute; width: 264px"/>

```

Figura 27.12 | JSP de `LibretaDirecciones` con un componente `AutoComplete Text Field`. (Parte I de 4).

```

45      <ui:textField binding="#{LibretaDirecciones.estadoCampoTexto}"
46      id="estadoCampoTexto" maxLength="2" required="true"
47      style="left: 480px; top: 120px; position: absolute;
48      width: 60px"/>
49      <ui:textField binding="#{LibretaDirecciones.cpCampoTexto}"
50      id="cpCampoTexto" maxLength="5" required="true"
51      style="left: 672px; top: 120px; position: absolute;
52      width: 60px"/>
53      <ui:label binding="#{LibretaDirecciones.pnombreEtiqueta}" for=
54      "pnombreCampoTexto" id="pnombreEtiqueta" style="left: 12px;
55      top: 72px; position: absolute" text="Primer nombre:"/>
56      <ui:label binding="#{LibretaDirecciones.apaternoEtiqueta}" for=
57      "apaternoCampoTexto" id="apaternoEtiqueta" style="position:
58      absolute; left: 384px; top: 72px" text="Apellido Paterno:"/>
59      <ui:label binding="#{LibretaDirecciones.calleEtiqueta}" for=
60      "calleCampoTexto" id="calleEtiqueta" style="left: 12px;
61      top: 96px; position: absolute" text="Calle:"/>
62      <ui:label binding="#{LibretaDirecciones.ciudadEtiqueta}" for=
63      "ciudadCampoTexto" id="ciudadEtiqueta" style="left: 12px;
64      top: 120px; position: absolute" text="Ciudad:"/>
65      <ui:label binding="#{LibretaDirecciones.estadoEtiqueta}" for=
66      "estadoCampoTexto" id="estadoEtiqueta" style="left: 408px;
67      top: 120px; position: absolute" text="Estado:"/>
68      <ui:label binding="#{LibretaDirecciones.cpEtiqueta}" for=
69      "cpCampoTexto" id="cpEtiqueta" style="height: 22px; left: 552px;
70      top: 120px; position: absolute; width: 94px" text="Código postal:"/>
71      <ui:button action="#{LibretaDirecciones.enviarBoton_action}"
72      binding="#{LibretaDirecciones.enviarBoton}" id=
73      "enviarBoton" primary="true" style="left: 131px;
74      top: 168px; position: absolute" text="Enviar"/>
75      <ui:button binding="#{LibretaDirecciones.borrarBoton}" id=
76      "borrarBoton" reset="true" style="left: 251px; top: 168px;
77      position: absolute" text="Borrar"/>
78      <ui:table augmentTitle="false" binding=
79      "#{LibretaDirecciones.direccionesTabla}" id="direccionesTabla"
80      paginationControls="true" style="height: 56px;
81      left: 12px; top: 204px; position: absolute; width: 720px"
82      title="Contactos" width="720">
83      <script><![CDATA[
84      <!--Las líneas 84 a 145 contienen código de JavaScript que se eliminó para ahorrar espacio.
85      El código fuente completo se proporciona en la carpeta de este ejemplo. -->
146      ]]]></script>
147      <ui:tableRowGroup binding=
148      "#{LibretaDirecciones.tableRowGroup1}"
149      id="tableRowGroup1" rows="5" sourceData=
150      "#{LibretaDirecciones.addressesDataProvider}"
151      sourceVar="currentRow">
152      <ui:tableColumn binding=
153      "#{LibretaDirecciones.pnombreColumna}" headerText=
154      "Primer nombre" id="pnombreColumna"
155      sort="ADDRESSES.FIRSTNAME">
156      <ui:staticText binding=
157      "#{LibretaDirecciones.staticText2}" id=
158      "staticText2" text="#{currentRow.value[
159      'ADDRESSES.FIRSTNAME']}" />
160      </ui:tableColumn>
161      <ui:tableColumn binding=
162      "#{LibretaDirecciones.apaternoColumna}" headerText=
163      "Apellido paterno" id="apaternoColumna"

```

Figura 27.12 | JSP de LibretaDirecciones con un componente autoComplete TextField. (Parte 2 de 4).

```

164         sort="ADDRESSES.LASTNAME">
165         <ui:staticText binding=
166             "#{LibretaDirecciones.staticText3}" id=
167             "staticText3" text="#{currentRow.value[
168                 'ADDRESSES.LASTNAME']}" />
169     </ui:tableColumn>
170     <ui:tableColumn binding=
171         "#{LibretaDirecciones.calleColumna}" headerText=
172         "Calle" id="calleColumna"
173         sort="ADDRESSES.STREET">
174         <ui:staticText binding=
175             "#{LibretaDirecciones.staticText4}" id=
176             "staticText4" text="#{currentRow.value[
177                 'ADDRESSES.STREET']}" />
178     </ui:tableColumn>
179     <ui:tableColumn binding=
180         "#{LibretaDirecciones.ciudadColumna}" headerText="Ciudad"
181         id="ciudadColumna" sort="ADDRESSES.CITY">
182         <ui:staticText binding=
183             "#{LibretaDirecciones.staticText5}" id="staticText5"
184             text="#{currentRow.value[
185                 'ADDRESSES.CITY']}" />
186     </ui:tableColumn>
187     <ui:tableColumn binding=
188         "#{LibretaDirecciones.estadoColumna}" headerText="Estado"
189         id="estadoColumna" sort="ADDRESSES.STATE">
190         <ui:staticText binding=
191             "#{LibretaDirecciones.staticText6}" id=
192             "staticText6" text="#{currentRow.value[
193                 'ADDRESSES.STATE']}" />
194     </ui:tableColumn>
195     <ui:tableColumn binding="#{LibretaDirecciones.cpColumna}"
196         headerText="CP" id="cpColumna"
197         sort="ADDRESSES.ZIP">
198         <ui:staticText binding=
199             "#{LibretaDirecciones.staticText7}" id="staticText7"
200             text="#{currentRow.value[
201                 'ADDRESSES.ZIP']}" />
202     </ui:tableColumn>
203 </ui:tableRowGroup>
204 </ui:table>
205 <ui:messageGroup binding="#{LibretaDirecciones.messageGroup1}"
206     id="messageGroup1" showGlobalOnly="true" style="height: 60px;
207     left: 456px; top: 432px; position: absolute; width: 190px" />
208 <ui:staticText binding="#{LibretaDirecciones.encabezadoBusqueda}"
209     id="encabezadoBusqueda" style="font-size: 18px; left: 24px;
210     top: 432px; position: absolute"
211     text="Buscar en la libreta de direcciones por apellido:" />
212 <ui:label binding="#{LibretaDirecciones.buscarNombreEtiqueta}"
213     id="buscarNombreEtiqueta"
214     style="left: 24px; top: 480px;
215     position: absolute"
216     text="Apellido paterno:" />
217 <ui:label binding="#{LibretaDirecciones.buscarNombreEtiqueta}"
218     for="nombreAutoComplete" id="buscarNombreEtiqueta"
219     requiredIndicator="true"
220     style="left: 24px; top: 480px; position: absolute"
221     text="Apellido paterno:" />

```

Figura 27.12 | JSP de LibretaDirecciones con un componente AutoComplete TextField. (Parte 3 de 4).

```

222         <ui:button binding="#{LibretaDirecciones.buscarBoton}"
223                 id="buscarBoton" style="left: 359px; top: 480px;
224                 position: absolute" text="Buscar"/>
225     </ui:form>
226 </ui:body>
227 </ui:html>
228 </ui:page>
229 </f:view>
230 </jsp:root>

```

http://localhost:29080/LibretaDirecciones/ - Microsoft Internet Explorer

Archivo Edición Ver Favoritos Herramientas Ayuda

Atrás Búsqueda Favoritos

Dirección http://localhost:29080/LibretaDirecciones/ Ir Vínculos

Agregar un contacto a la libreta de direcciones:

Primer nombre: Apellido Paterno:

Calle:

Ciudad: Estado: Código postal:

Primer nombre	Apellido paterno	Calle	Ciudad	Estado	CP
Sue	Black	1000 Michigan Ave.	Chicago	IL	60605
James	Blue	1000 Harbor Ave.	Seattle	WA	98116
Mike	Brown	3600 Delmar Blvd.	St Louis	MO	63108
Harrison	Ford	10567 Long Rd	El Segundo	CA	56687
Meg	Gold	1200 Stout St	Denver	CO	80204

Página: 1 de 2 Ir

Buscar en la libreta de direcciones por apellido:

Apellido paterno:

Black, Sue
Blue, James
Brown, Mike

Intranet local

Figura 27.12 | JSP de LibretaDirecciones con un componente AutoComplete TextField. (Parte 4 de 4).

En las líneas 21 a 25 se configuran los formularios virtuales para esta página. En las líneas 217 a 221 se define el componente **AutoComplete Text Field**. El atributo `completionMethod` de este componente está enlazado al método `nombreAutoComplete_complete` del bean de página (que veremos en la sección 27.4.3), el cual proporciona la lista de opciones que debe sugerir el componente **AutoComplete Text Field**. Para crear este método, haga clic con el botón derecho en el componente `nombreAutoComplete` en vista de **Diseño** y seleccione **Editar controlador de eventos > complete**. Observe que el botón **Buscar** (líneas 222 a 224) no especifica un enlace con el método manejador de acciones; agregaremos esto en la sección 27.5.

27.4.3 Cómo proporcionar sugerencias para un AutoComplete Text Field

En la figura 27.13 se muestra el archivo de bean de página para la JSP de la figura 27.12. Incluye el método `nombreAutoComplete_complete`, el cual proporciona la funcionalidad para el componente **AutoComplete Text Field**. Aparte de este método, este bean de página es idéntico al de la figura 27.8.

```

1  // Fig. 27.13: LibretaDirecciones.java
2  // Bean de página para sugerir nombres en el componente AutoComplete Text Field.
3  package libretadirecciones;
4
5  import com.sun.data.provider.RowKey;
6  import com.sun.rave.web.ui.appbase.AbstractPageBean;
7  import com.sun.rave.web.ui.component.Body;
8  import com.sun.rave.web.ui.component.Form;
9  import com.sun.rave.web.ui.component.Head;
10 import com.sun.rave.web.ui.component.Html;
11 import com.sun.rave.web.ui.component.Link;
12 import com.sun.rave.web.ui.component.Page;
13 import javax.faces.FacesException;
14 import com.sun.rave.web.ui.component.StaticText;
15 import com.sun.rave.web.ui.component.TextField;
16 import com.sun.rave.web.ui.component.Label;
17 import com.sun.rave.web.ui.component.Button;
18 import com.sun.rave.web.ui.component.Table;
19 import com.sun.rave.web.ui.component.TableRowGroup;
20 import com.sun.rave.web.ui.component.TableColumn;
21 import com.sun.data.provider.impl.CachedRowSetDataProvider;
22 import com.sun.rave.web.ui.component.MessageGroup;
23 import com.sun.j2ee.blueprints.ui.autocomplete.AutoCompleteComponent;
24 import com.sun.j2ee.blueprints.ui.autocomplete.CompletionResult;
25 import javax.faces.context.FacesContext;
26
27 public class LibretaDirecciones extends AbstractPageBean
28 {
29     private int __placeholder;
30
31     private void _init() throws Exception
32     {
33         addressesDataProvider.setCachedRowSet(
34             (javax.sql.rowset.CachedRowSet)
35             getValue("#{SessionBean1.addressesRowSet}" ));
36         direccionesTabla.setInternalVirtualForm(true);
37     }
38
39     // Las líneas 39 a 572 del código generado en forma automática se eliminaron para
40     // ahorrar espacio.
41     // El código fuente completo se proporciona en la carpeta de este ejemplo.
42
573 public void prerender()
574 {
575     addressesDataProvider.refresh();
576 } // fin del método prerender
577
578 public void destroy()
579 {
580     addressesDataProvider.close();
581 } // fin del método destroy
582
583 // manejador de acciones que agrega un contacto a la base de datos LibretaDirecciones
584 // cuando el usuario hace clic en el botón Enviar
585 public String enviarBoton_action()
586 {
587     if ( addressesDataProvider.canAppendRow() )
588     {
589         try

```

Figura 27.13 | Bean de página para sugerir nombres en el componente **AutoComplete Text Field**. (Parte I de 3).

```

590     {
591         RowKey rk = addressesDataProvider.appendRow();
592         addressesDataProvider.setCursorRow(rk);
593
594         addressesDataProvider.setValue( "ADDRESSES.FIRSTNAME",
595             pnombreCampoTexto.getValue() );
596         addressesDataProvider.setValue( "ADDRESSES.LASTNAME",
597             apaternoCampoTexto.getValue() );
598         addressesDataProvider.setValue( "ADDRESSES.STREET",
599             calleCampoTexto.getValue() );
600         addressesDataProvider.setValue( "ADDRESSES.CITY",
601             ciudadCampoTexto.getValue() );
602         addressesDataProvider.setValue( "ADDRESSES.STATE",
603             estadoCampoTexto.getValue() );
604         addressesDataProvider.setValue( "ADDRESSES.ZIP",
605             cpCampoTexto.getValue() );
606         addressesDataProvider.commitChanges();
607
608         // restablece los campos de texto
609         apaternoCampoTexto.setValue( "" );
610         pnombreCampoTexto.setValue( "" );
611         calleCampoTexto.setValue( "" );
612         ciudadCampoTexto.setValue( "" );
613         estadoCampoTexto.setValue( "" );
614         cpCampoTexto.setValue( "" );
615     } // fin de try
616     catch ( Exception ex )
617     {
618         error( "No se actualizo la libreta de direcciones." +
619             ex.getMessage() );
620     } // fin de catch
621 } // fin de if
622
623     return null;
624 } // fin del método enviarBoton_action
625
626
627 // manejador de acciones para el cuadro autocompletar que obtiene los nombres
628 // de la libreta de direcciones, cuyos prefijos coincidan con las letras escritas
629 // hasta un momento dado, y los muestra en una lista de sugerencias.
630 public void nombreAutoComplete_complete( FacesContext context, String
631     prefix, CompletionResult result )
632 {
633     try
634     {
635         boolean tieneElSiguiente = addressesDataProvider.cursorFirst();
636
637         while ( tieneElSiguiente )
638         {
639             // obtiene un nombre de la base de datos
640             String nombre =
641                 (String) addressesDataProvider.getValue(
642                     "ADDRESSES.LASTNAME" ) + ", " +
643                 (String) addressesDataProvider.getValue(
644                     "ADDRESSES.FIRSTNAME" ) ;
645
646             // si el nombre en la base de datos empieza con el prefijo, se
647             // agrega a la lista de sugerencias
648             if ( nombre.toLowerCase().startsWith( prefix.toLowerCase() ) )

```

Figura 27.13 | Bean de página para sugerir nombres en el componente **AutoComplete Text Field**. (Parte 2 de 3).

```

649         {
650             result.addItem( nombre );
651         } // fin de if
652     else
653     {
654         // termina el ciclo si el resto de los nombres son
655         // alfabéticamente menores que el prefijo
656         if ( prefix.compareTo( nombre ) < 0 )
657         {
658             break;
659         } // fin de if
660     } // fin de else
661
662     // desplaza el cursor a la siguiente fila de la base de datos
663     tieneElSiguiente = addressesDataProvider.cursorNext();
664 } // fin de while
665 } // fin de try
666 catch ( Exception ex )
667 {
668     result.addItem( "Excepcion al obtener nombres que coincidan." );
669 } // fin de catch
670 } // fin del método nombreAutoComplete_complete
671 } // fin de la clase LibretaDirecciones

```

Figura 27.13 | Bean de página para sugerir nombres en el componente **AutoComplete Text Field**. (Parte 3 de 3).

El método `nombreAutoComplete_complete` (líneas 630 a 670) se invoca después de cada pulsación de tecla en el componente **AutoComplete Text Field**, para actualizar la lista de sugerencias con base en el texto que el usuario ha escrito hasta cierto punto. El método recibe una cadena (`prefix`) que contiene el texto que introdujo el usuario, y un objeto `CompletionResult` (`result`) que se utiliza para mostrar sugerencias al usuario. El método itera a través de las filas del objeto `addressesDataProvider`, obtiene el nombre de cada fila, comprueba si el nombre empieza con las letras escritas hasta cierto punto y, de ser así, agrega el nombre a `result`. En la línea 635 se establece el cursor a la primera fila en el proveedor de datos. En la línea 637 se determina si hay más filas en el proveedor de datos. De ser así, en las líneas 640 a 644 se obtienen el apellido paterno y el primer nombre de la fila actual, y se crea un objeto `String` en el formato *apellido paterno, primer nombre*. En la línea 648 se comparan las versiones en minúscula de `nombre` y `prefix` para determinar si el nombre empieza con los caracteres escritos hasta ahora. De ser así, el nombre es una coincidencia y en la línea 650 se agrega a `result`.

Recuerde que el proveedor de datos envuelve un objeto `CachedRowSet` que contiene una consulta SQL que devuelve las filas en la base de datos ordenada por apellido paterno, y después por primer nombre. Esto nos permite iterar a través del proveedor de datos, una vez que llegamos a una fila cuyo nombre va alfabéticamente después que el texto introducido por el usuario; los nombres en las filas más allá de esto serán alfabéticamente mayores y, por ende, no son coincidencias potenciales. Si el nombre no coincide con el texto introducido hasta un momento dado, en la línea 656 se evalúa si el nombre actual es alfabéticamente mayor que el prefijo (`prefix`). De ser así, en la línea 658 se termina el ciclo.



Tip de rendimiento 27.1

*Al usar columnas de la base de datos para proporcionar sugerencias en un componente **AutoComplete Text Field**, si ordenamos las columnas eliminamos la necesidad de comprobar cada fila en la base de datos, en búsqueda de coincidencias potenciales. Esto mejora considerablemente el rendimiento cuando se maneja una base de datos extensa.*

Si el nombre no es una coincidencia, ni es alfabéticamente mayor que `prefix`, entonces en la línea 663 se desplaza el cursor a la siguiente fila en el proveedor de datos. Si hay otra fila, el ciclo vuelve a iterar, comprobando si el nombre en la siguiente fila coincide con el valor de `prefix` y debe agregarse a `results`.

En las líneas 666 a 669 se atrapan las excepciones que se generen mientras se realiza la búsqueda en la base de datos. En la línea 668 se agrega texto al cuadro de sugerencias, indicando el error al usuario.

27.5 Componente Map Viewer de Google Maps

Ahora completaremos la aplicación `LibretaDirecciones`, para lo cual agregaremos funcionalidad al **Botón Buscar**. Cuando el usuario hace clic en este **Botón**, el nombre en el componente **AutoComplete Text Field** se utiliza para buscar en la base de datos `LibretaDirecciones`. También agregamos a la página un componente JSF **Map Viewer** habilitado para Ajax, para mostrar un mapa del área para esa dirección. Un componente **Map Viewer** utiliza el servicio Web de la **API de Google Maps** para buscar y mostrar mapas. (En el capítulo 28 hablaremos sobre los detalles de los servicios Web). En este ejemplo, utilizar la API de Google Maps es un proceso análogo a crear llamadas a métodos ordinarios en un objeto **Map Viewer** y su bean de soporte en el archivo de bean de página. Al encontrar un contacto, mostramos un mapa del vecindario con un componente **Map Viewer** que apunta a la ubicación e indica el nombre del contacto y su dirección.

27.5.1 Cómo obtener una clave de la API Google Maps

Para utilizar el componente **Map Viewer**, debe tener una cuenta con Google. Visite el sitio <https://www.google.com/accounts/ManageAccount> para registrarse y obtener una cuenta gratuita, si no tiene una ya. Una vez que haya iniciado sesión en su cuenta, debe obtener una clave para usar la API de Google Maps en www.google.com/apis/maps. La clave que reciba será específica para esta aplicación Web y limitará el número de mapas que puede mostrar la aplicación por día. Cuando se registre para la clave, tendrá que escribir el URL para la aplicación que utilizará la API de Google Maps. Si va a desplegar la aplicación sólo en el servidor de prueba Sun Application Server 8 de Java Studio Creator 2, escriba `http://localhost:29080` como el URL.

Una vez que acepte los términos y condiciones de Google, será redirigido a una página que contendrá su nueva clave para la API de Google Maps. Guarde esta clave en un archivo de texto, en una ubicación conveniente para una futura referencia.

27.5.2 Cómo agregar un componente y un Map Viewer a una página

Ahora que tiene una clave para usar la API de Google Maps, está listo para completar la aplicación `LibretaDirecciones`. Con el archivo `LibretaDirecciones.jsp` abierto en modo **Diseño**, agregue un componente **Map Viewer** llamado `mapViewer` debajo del componente `nombreAutoComplete`. En la ventana **Propiedades**, establezca la propiedad `clave` del componente **Map Viewer** con la clave que obtuvo para acceder a la API de Google Maps. Establezca la propiedad `rendered` en `false`, de manera que el mapa no se muestre cuando el usuario todavía no haya buscado una dirección. Establezca la propiedad `zoomLevel` en 1 (In), de manera que el usuario pueda ver los nombres de las calles en el mapa.

Sulte un componente **Map Marker** (llamado `mapMarker`) de la sección **AJAX Support Beans** de la **Paleta** en cualquier parte de la página. Este componente (que no está visible en modo **Diseño**) marca la ubicación del contacto en el mapa. Debe enlazar el marcador con el mapa, de manera que se muestre el marcador en el mapa. Para ello, haga clic con el botón derecho en el componente **Map Viewer** y seleccione **Enlaces de propiedades...** para mostrar el cuadro de diálogo **Enlaces de propiedades**. Seleccione `info` de la columna **Seleccionar propiedad enlazable** del cuadro de diálogo, y después seleccione `mapMarker` de la columna **Seleccionar destino de enlace**. Haga clic en **Aplicar** y después en **Cerrar**.

Por último, suelte un componente **Geocoding Service Object** (llamado `geoCoder`) de la sección **AJAX Support Beans** de la **Paleta**, en cualquier parte de la página. Este objeto (que no está visible en modo **Diseño**) convierte las direcciones de las calles en latitudes y longitudes que el componente **Map Viewer** utiliza para mostrar un mapa apropiado.

Cómo agregar un proveedor de datos a la página

Para completar esta aplicación, necesita un segundo proveedor de datos para buscar en la base de datos `LibretaDirecciones`, con base en el primer nombre y apellido paterno introducidos en el componente **AutoComplete Text Field**. Abra la ventana **Servidores** y expanda el nodo `LibretaDirecciones` junto con su nodo **Tablas** para revelar la tabla **Addresses**. Haga clic con el botón derecho del ratón en el nodo de la tabla y seleccione **Agregar a página** para mostrar el cuadro de diálogo **Agregar proveedor de datos con RowSet** (figura 27.14). Queremos crear un nuevo origen de datos, en vez de utilizar el existente, ya que la consulta para buscar contactos es distinta de la consulta para mostrar todos los contactos. Seleccione la opción **Crear** para el objeto `SessionBean1` y escriba el nombre `busquedaDirecciones` para el proveedor de datos. Haga clic en **Aceptar** para crear el nuevo proveedor de datos. En la ventana **Esquema**, se ha agregado un nuevo nodo llamado `busquedaDireccionesDataProvider`

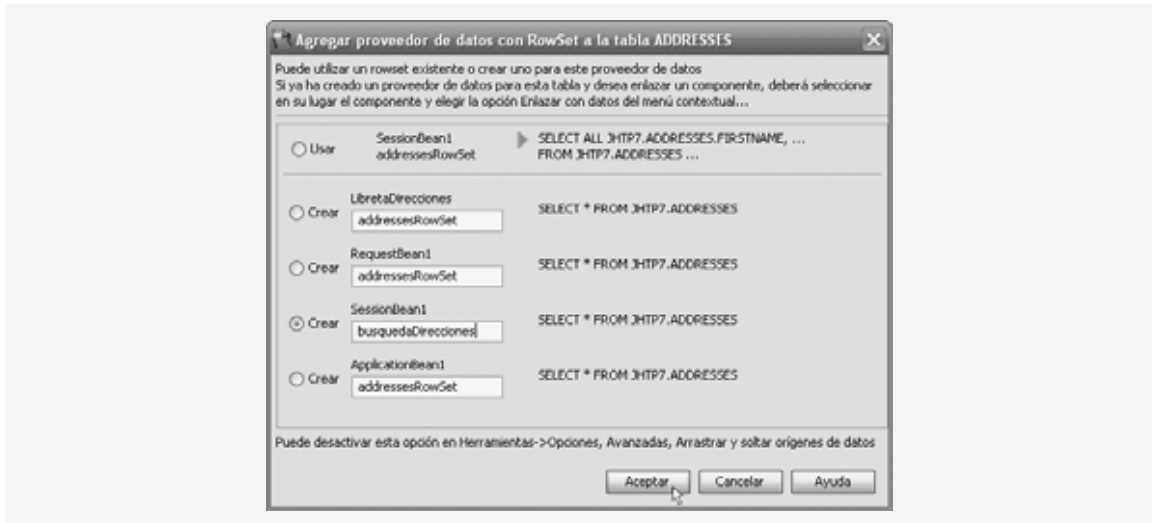


Figura 27.14 | Cuadro de diálogo para crear un nuevo proveedor de datos.

al nodo `LibretaDirecciones`, y se ha agregado un nodo llamado `busquedaDirecciones` al nodo `SessionBean`.

Haga doble clic en el nodo `busquedaDirecciones` para editar la instrucción SQL para este objeto RowSet. Como vamos a usar este conjunto de filas para buscar en la base de datos un apellido paterno y un primer nombre, necesitamos agregar parámetros de búsqueda a la instrucción `SELECT` que ejecutará el objeto RowSet. Para ello, escriba el texto `"= ?"` en la columna **Criterios** de las filas del primer nombre y apellido paterno en la tabla del editor de instrucciones SQL. El número 1 deberá aparecer en la columna **Orden** para el primer nombre, y el número 2 para el apellido paterno. Observe que se han agregado las líneas

```
WHERE JHTP7.ADDRESSES.FIRSTNAME = ?
AND JHTP7.ADDRESSES.LASTNAME = ?
```

a la instrucción SQL. Esto indica que el objeto RowSet ahora ejecuta una instrucción SQL con parámetros. Estos parámetros se pueden establecer mediante programación, en donde el primer nombre es el primer parámetro y el apellido paterno es el segundo.

27.5.3 Archivo JSP con un componente Map Viewer

La figura 27.15 presenta el archivo JSP para la aplicación de libreta de direcciones completa. Es casi idéntico al archivo JSP de las dos versiones anteriores de esta aplicación. La nueva característica es el componente **Map Viewer** (y sus componentes de soporte) que se utiliza para mostrar un mapa con la ubicación del contacto. Sólo hablaremos de los nuevos elementos de este archivo. [Nota: este código no se ejecutará sino hasta que haya especificado su propia clave Google Maps en las líneas 227 a 229. Puede pegar su clave en la propiedad **key** del componente **Map Viewer** en la ventana **Propiedades**].

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <!-- Fig. 27.15: LibretaDirecciones.jsp -->
4 <!-- Página JSP de LibretaDirecciones con un componente Map Viewer. -->
5
6 <jsp:root version="1.2" xmlns:bp="http://java.sun.com/blueprints/ui/14"
```

Figura 27.15 | JSP de `LibretaDirecciones` con un componente **Map Viewer**. (Parte I de 5).

```

7  xmlns:f="http://java.sun.com/jsf/core" xmlns:h=
8  "http://java.sun.com/jsf/html" xmlns:jspx="http://java.sun.com/JSP/Page"
9  xmlns:ui="http://www.sun.com/web/ui">
10 <jsp:directive.page contentType="text/html; charset=UTF-8"
11     pageEncoding="UTF-8"/>
12 <f:view>
13     <ui:page binding="#{LibretaDirecciones.page1}" id="page1">
14         <ui:html binding="#{LibretaDirecciones.html1}" id="html1">
15             <ui:head binding="#{LibretaDirecciones.head1}" id="head1">
16                 <ui:link binding="#{LibretaDirecciones.link1}" id="link1"
17                     url="/resources/stylesheet.css"/>
18             </ui:head>
19             <ui:body binding="#{LibretaDirecciones.body1}" id="body1"
20                 style="-rave-layout: grid">
21                 <ui:form binding="#{LibretaDirecciones.form1}" id="form1"
22                     virtualFormsConfig="agregarForm | apaternoCampoTexto
23                     pnombreCampoTexto calleCampoTexto estadoCampoTexto
24                     cpCampoTexto ciudadCampoTexto | enviarBoton , buscarForm
25                     | nombreAutoComplete | buscarBoton">
26                     <ui:staticText binding="#{LibretaDirecciones.staticText1}" id=
27                         "staticText1" style="font-size: 18px; left: 12px;
28                         top: 24px; position: absolute"
29                         text="Agregar un contacto a la libreta de direcciones:"/>
30                     <ui:textField binding="#{LibretaDirecciones.pnombreCampoTexto}"
31                         id="pnombreCampoTexto" maxLength="20" required="true"
32                         style="left: 132px; top: 72px;
33                         position: absolute"/>
34                     <ui:textField binding="#{LibretaDirecciones.apaternoCampoTexto}"
35                         id="apaternoCampoTexto" maxLength="30" required="true"
36                         style="left: 504px; top: 72px; position: absolute;
37                         width: 228px"/>
38                     <ui:textField binding="#{LibretaDirecciones.calleCampoTexto}"
39                         id="calleCampoTexto" maxLength="100" required="true"
40                         style="left: 132px; top: 96px; position: absolute;
41                         width: 600px"/>
42                     <ui:textField binding="#{LibretaDirecciones.ciudadCampoTexto}"
43                         id="ciudadCampoTexto" maxLength="30" required="true"
44                         style="left: 132px; top: 120px; position: absolute; width: 264px"/>
45                     <ui:textField binding="#{LibretaDirecciones.estadoCampoTexto}"
46                         id="estadoCampoTexto" maxLength="2" required="true"
47                         style="left: 480px; top: 120px; position: absolute;
48                         width: 60px"/>
49                     <ui:textField binding="#{LibretaDirecciones.cpCampoTexto}"
50                         id="cpCampoTexto" maxLength="5" required="true"
51                         style="left: 672px; top: 120px; position: absolute;
52                         width: 60px"/>
53                     <ui:label binding="#{LibretaDirecciones.pnombreEtiqueta}" for=
54                         "pnombreCampoTexto" id="pnombreEtiqueta" style="left: 12px;
55                         top: 72px; position: absolute" text="Primer nombre:"/>
56                     <ui:label binding="#{LibretaDirecciones.apaternoEtiqueta}" for=
57                         "apaternoCampoTexto" id="apaternoEtiqueta" style="position:
58                         absolute; left: 384px; top: 72px" text="Apellido Paterno:"/>
59                     <ui:label binding="#{LibretaDirecciones.calleEtiqueta}" for=
60                         "calleCampoTexto" id="calleEtiqueta" style="left: 12px;
61                         top: 96px; position: absolute" text="Calle:"/>
62                     <ui:label binding="#{LibretaDirecciones.ciudadEtiqueta}" for=
63                         "ciudadCampoTexto" id="ciudadEtiqueta" style="left: 12px;
64                         top: 120px; position: absolute" text="Ciudad:"/>
65                     <ui:label binding="#{LibretaDirecciones.estadoEtiqueta}"

```

Figura 27.15 | JSP de LibretaDirecciones con un componente Map Viewer. (Parte 2 de 5).

```

66         for="estadoCampoTexto" id="estadoEtiqueta" style="left: 408px;
67         top: 120px; position: absolute" text="Estado:"/>
68     <ui:label binding="#{LibretaDirecciones.cpEtiqueta}" for=
69     "cpCampoTexto" id="cpEtiqueta" style="height: 22px; left: 552px;
70     top: 120px; position: absolute; width: 94px" text="Código
        postal:"/>
71     <ui:button action="#{LibretaDirecciones.enviarBoton_action}"
72     binding="#{LibretaDirecciones.enviarBoton}" id=
73     "enviarBoton" primary="true" style="left: 131px;
74     top: 168px; position: absolute" text="Enviar"/>
75     <ui:button binding="#{LibretaDirecciones.borrarBoton}" id=
76     "borrarBoton" reset="true" style="left: 251px; top: 168px;
77     position: absolute" text="Borrar"/>
78     <ui:table augmentTitle="false" binding=
79     "#{LibretaDirecciones.direccionesTabla}" id="direccionesTabla"
80     paginationControls="true" style="height: 56px; left: 12px;
81     top: 204px; position: absolute; width: 720px"
82     title="Contactos" width="720">
83         <script><![CDATA[
84     <!--Las líneas 84 a 145 contienen código de JavaScript que se eliminó para ahorrar
        espacio.
85     El código fuente completo se proporciona en la carpeta de este ejemplo. -->
146     ]]></script>
147     <ui:tableRowGroup binding=
148     "#{LibretaDirecciones.tableRowGroup1}" id=
149     "tableRowGroup1" rows="5" sourceData=
150     "#{LibretaDirecciones.addressesDataProvider}"
151     sourceVar="currentRow">
152     <ui:tableColumn binding=
153     "#{LibretaDirecciones.pnombreColumna}" headerText=
154     "Primer nombre" id="pnombreColumna"
155     sort="ADDRESSES.FIRSTNAME">
156     <ui:staticText binding=
157     "#{LibretaDirecciones.staticText2}"
158     id="staticText2" text="#{currentRow.value[
159     'ADDRESSES.FIRSTNAME']}">
160     </ui:tableColumn>
161     <ui:tableColumn binding=
162     "#{LibretaDirecciones.apaternoColumna}" headerText=
163     "Apellido paterno" id="apaternoColumna"
164     sort="ADDRESSES.LASTNAME">
165     <ui:staticText binding=
166     "#{LibretaDirecciones.staticText3}" id=
167     "staticText3" text="#{currentRow.value[
168     'ADDRESSES.LASTNAME']}">
169     </ui:tableColumn>
170     <ui:tableColumn binding=
171     "#{LibretaDirecciones.calleColumna}" headerText=
172     "Calle" id="calleColumna"
173     sort="ADDRESSES.STREET">
174     <ui:staticText binding=
175     "#{LibretaDirecciones.staticText4}" id=
176     "staticText4" text="#{currentRow.value[
177     'ADDRESSES.STREET']}">
178     </ui:tableColumn>
179     <ui:tableColumn binding=
180     "#{LibretaDirecciones.ciudadColumna}" headerText=

```

Figura 27.15 | JSP de LibretaDirecciones con un componente **Map Viewer**. (Parte 3 de 5).

```

181         "Ciudad" id="ciudadColumna" sort="ADDRESSES.CITY">
182         <ui:staticText binding=
183             "#{LibretaDirecciones.staticText5}" id="staticText5"
184             text="#{currentRow.value[
185                 'ADDRESSES.CITY']}" />
186         </ui:tableColumn>
187         <ui:tableColumn binding=
188             "#{LibretaDirecciones.estadoColumna}" headerText="Estado"
189             id="estadoColumna" sort="ADDRESSES.STATE">
190             <ui:staticText binding=
191                 "#{LibretaDirecciones.staticText6}" id=
192                 "staticText6" text="#{currentRow.value[
193                     'ADDRESSES.STATE']}" />
194             </ui:tableColumn>
195             <ui:tableColumn binding="#{LibretaDirecciones.cpColumna}"
196                 headerText="CP" id="cpColumna"
197                 sort="ADDRESSES.ZIP">
198                 <ui:staticText binding=
199                     "#{LibretaDirecciones.staticText7}" id="staticText7"
200                     text="#{currentRow.value[
201                         'ADDRESSES.ZIP']}" />
202                 </ui:tableColumn>
203             </ui:tableRowGroup>
204         </ui:table>
205         <ui:messageGroup binding="#{LibretaDirecciones.messageGroup1}"
206             id="messageGroup1" showGlobalOnly="true" style="height: 60px;
207             left: 456px; top: 408px; position: absolute; width: 190px"/>
208         <ui:staticText binding="#{LibretaDirecciones.encabezadoBusqueda}" id=
209             "encabezadoBusqueda" style="font-size: 18px; left: 24px;
210             top: 432px; position: absolute"
211             text="Buscar en la libreta de direcciones por apellido:" />
212         <ui:label binding="#{LibretaDirecciones.buscarNombreEtiqueta}" for=
213             "nombreAutoComplete" id="buscarNombreEtiqueta"
214             requiredIndicator="true" style="left: 24px; top: 480px;
215             position: absolute" text="Apellido paterno:" />
216         <bp:autoComplete binding=
217             "#{LibretaDirecciones.nombreAutoComplete}" completionMethod=
218             "#{LibretaDirecciones.nombreAutoComplete_complete}"
219             id="nombreAutoComplete" required="true" style="left:
220             144px; top: 480px; position: absolute" />
221         <ui:button action="#{LibretaDirecciones.buscarBoton_action}"
222             binding="#{LibretaDirecciones.buscarBoton}" id=
223             "buscarBoton" style="left: 359px; top: 480px; position:
224             absolute" text="Buscar" />
225         <bp:mapViewer binding="#{LibretaDirecciones.mapViewer}" center=
226             "#{LibretaDirecciones.mapViewer_center}" id="mapViewer"
227             info="#{LibretaDirecciones.mapMarker}" key=
228             "ABQIAAAAXDzuwvNqoM33k508Fm0I9BRv3hXvfLy8rd_zkEeAYi6qFBadthS
229             as7kIyZ1EERRCUWTUqIrqGp8ybg" mapControls="false"
230             style="height: 550px; left: 24px; top: 528px; position:
231             absolute; width: 718px" zoomLevel="1" />
232     </ui:form>
233 </ui:body>
234 </ui:html>
235 </ui:page>
236 </f:view>
237 </jsp:root>

```

Figura 27.15 | JSP de LibretaDirecciones con un componente **Map Viewer**. (Parte 4 de 5).

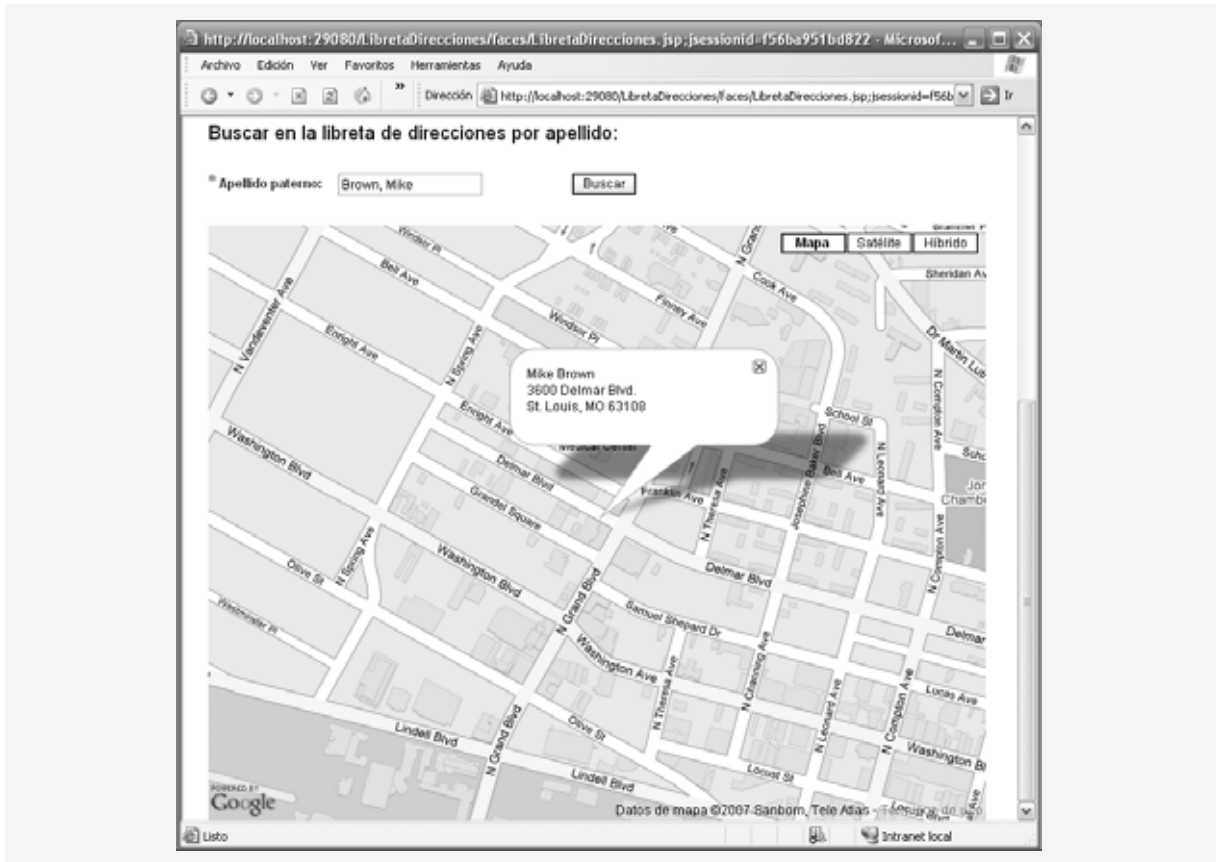


Figura 27.15 | JSP de LibretaDirecciones con un componente **Map Viewer**. (Parte 5 de 5).

En las líneas 242 a 247 se define el componente `mapViewer` que muestra un mapa del área que rodea a la dirección. El atributo `center` del componente está enlazado a la propiedad `mapViewer_center` del bean de página. Esta propiedad se manipula en el archivo de bean de página, para centrar el mapa en la dirección deseada.

El atributo `action` del botón **Buscar** ahora está enlazado al método `buscarBoton_action` en el bean de página (línea 226). Este manejador de acciones busca en la base de datos `LibretaDirecciones` el nombre introducido en el componente **AutoComplete Text Field** y muestra el nombre del contacto, junto con la dirección, en un mapa de la ubicación de ese contacto.

27.5.4 Bean de página que muestra un mapa en el componente Map Viewer

En la figura 27.16 se presenta el bean de página para la aplicación `LibretaDirecciones` completa. La mayor parte de este archivo es idéntico a los beans de página para las primeras dos versiones de esta aplicación. Hablaremos sólo del nuevo método manejador de acciones, `buscarBoton_action`.

El método `buscarBoton_action` (líneas 646 a 704) se invoca cuando el usuario hace clic en el botón **Buscar** en el formulario inferior de la página. Las líneas 649 a 652 recuperan el nombre del componente **AutoComplete Text Field** y lo separan en cadenas para el primer nombre y el apellido paterno. En cada una de las líneas 662 a 669 se obtiene el objeto `CachedRowSet` de `busquedaDireccionesDataProvider`, y después se utiliza su método `setObject` para establecer los parámetros de la consulta con el primer nombre y el apellido paterno. El método `setObject` sustituye un parámetro en la consulta SQL con una cadena especificada. En la línea 661 se actualiza el proveedor de datos, el cual ejecuta la consulta del objeto `RowSet` envuelto con los nuevos parámetros. El conjunto de resultados ahora contiene sólo filas que coinciden con el primer nombre y el apellido paterno del componente **AutoComplete Text Field**. En las líneas 662 a 669 se obtienen de la base de datos la dirección de

la calle, la ciudad, el estado y el código postal para este contacto. Observe que en este ejemplo, suponemos que no hay varias entradas en la libreta de direcciones para los mismos valores de primer nombre y apellido paterno, ya que sólo obtenemos la información de la dirección para la primera fila en el proveedor de datos. Cualquier fila adicional que coincida con el primer nombre y el apellido paterno se ignora.

```

1 // Fig. 27.16: LibretaDirecciones.java
2 // Bean de página para agregar un contacto a la libreta de direcciones.
3 package libretadirecciones;
4
5 import com.sun.data.provider.RowKey;
6 import com.sun.rave.web.ui.appbase.AbstractPageBean;
7 import com.sun.rave.web.ui.component.Body;
8 import com.sun.rave.web.ui.component.Form;
9 import com.sun.rave.web.ui.component.Head;
10 import com.sun.rave.web.ui.component.Html;
11 import com.sun.rave.web.ui.component.Link;
12 import com.sun.rave.web.ui.component.Page;
13 import javax.faces.FacesException;
14 import com.sun.rave.web.ui.component.StaticText;
15 import com.sun.rave.web.ui.component.TextField;
16 import com.sun.rave.web.ui.component.Label;
17 import com.sun.rave.web.ui.component.Button;
18 import com.sun.rave.web.ui.component.Table;
19 import com.sun.rave.web.ui.component.TableRowGroup;
20 import com.sun.rave.web.ui.component.TableColumn;
21 import com.sun.data.provider.impl.CachedRowSetDataProvider;
22 import com.sun.rave.web.ui.component.MessageGroup;
23 import com.sun.j2ee.blueprints.ui.autocomplete.AutoCompleteComponent;
24 import com.sun.j2ee.blueprints.ui.autocomplete.CompletionResult;
25 import javax.faces.context.FacesContext;
26 import com.sun.j2ee.blueprints.ui.mapviewer.MapComponent;
27 import com.sun.j2ee.blueprints.ui.mapviewer.MapPoint;
28 import com.sun.j2ee.blueprints.ui.geocoder.GeoCoder;
29 import com.sun.j2ee.blueprints.ui.geocoder.GeoPoint;
30 import com.sun.j2ee.blueprints.ui.mapviewer.MapMarker;
31
32 public class LibretaDirecciones extends AbstractPageBean
33 {
34     private int __placeholder;
35
36     private void _init() throws Exception
37     {
38         addressesDataProvider.setCachedRowSet(
39             (javax.sql.rowset.CachedRowSet)
40             getValue("#{SessionBean1.addressesRowSet}"));
41         direccionesTabla.setInternalVirtualForm(true);
42         busquedaDireccionesDataProvider.setCachedRowSet(
43             (javax.sql.rowset.CachedRowSet)
44             getValue("#{SessionBean1.busquedaDirecciones}"));
45         mapView.setRendered(false);
46     } // fin del método _init
47
48     // Las líneas 48 a 544 del código generado en forma automática se eliminaron para
49     // ahorrar espacio.
50     // El código fuente completo se proporciona en la carpeta de este ejemplo.
51
52     public void prerender()

```

Figura 27.16 | Bean de página que obtiene un mapa para mostrarlo en el componente MapViewer. (Parte 1 de 4).

```

546 {
547     addressesDataProvider.refresh();
548 } // fin del método prerender
549
550 public void destroy()
551 {
552     busquedaDireccionesDataProvider.close();
553     addressesDataProvider.close();
554 } // fin del método destroy
555
556 // manejador de acciones que agrega un contacto a la base de datos LibretaDirecciones
557 // cuando el usuario hace clic en el botón Enviar
558 public String enviarBoton_action()
559 {
560     if ( addressesDataProvider.canAppendRow() )
561     {
562         try
563         {
564             RowKey rk = addressesDataProvider.appendRow();
565             addressesDataProvider.setCursorRow(rk);
566
567             addressesDataProvider.setValue( "ADDRESSES.FIRSTNAME",
568                 pnombreCampoTexto.getValue() );
569             addressesDataProvider.setValue( "ADDRESSES.LASTNAME",
570                 apaternoCampoTexto.getValue() );
571             addressesDataProvider.setValue( "ADDRESSES.STREET",
572                 calleCampoTexto.getValue() );
573             addressesDataProvider.setValue( "ADDRESSES.CITY",
574                 ciudadCampoTexto.getValue() );
575             addressesDataProvider.setValue( "ADDRESSES.STATE",
576                 estadoCampoTexto.getValue() );
577             addressesDataProvider.setValue( "ADDRESSES.ZIP",
578                 cpCampoTexto.getValue() );
579             addressesDataProvider.commitChanges();
580
581             // restablece los campos de texto
582             apaternoCampoTexto.setValue( "" );
583             pnombreCampoTexto.setValue( "" );
584             calleCampoTexto.setValue( "" );
585             ciudadCampoTexto.setValue( "" );
586             estadoCampoTexto.setValue( "" );
587             cpCampoTexto.setValue( "" );
588         } // fin de try
589         catch ( Exception ex )
590         {
591             error( "No se actualizo la libreta de direcciones." +
592                 ex.getMessage() );
593         } // fin de catch
594     } // fin de if
595
596     return null;
597 } // fin del método enviarBoton_action
598
599 // manejador de acciones para el cuadro autocompletar que obtiene los nombres
600 // de la libreta de direcciones, cuyos prefijos coincidan con las letras escritas
601 // hasta un momento dado, y los muestra en una lista de sugerencias.
602 public void nombreAutoComplete_complete( FacesContext context, String
603     prefix, CompletionResult result )
604 {

```

Figura 27.16 | Bean de página que obtiene un mapa para mostrarlo en el componente MapViewer. (Parte 2 de 4).

```

605     try
606     {
607         boolean tieneElSiguiente = addressesDataProvider.cursorFirst();
608
609         while ( tieneElSiguiente )
610         {
611             // obtiene un nombre de la base de datos
612             String nombre =
613                 (String) addressesDataProvider.getValue(
614                     "ADDRESSES.LASTNAME" ) + ", " +
615                 (String) addressesDataProvider.getValue(
616                     "ADDRESSES.FIRSTNAME" ) ;
617
618             // si el nombre en la base de datos empieza con el prefijo, se
619             // agrega a la lista de sugerencias
620             if ( nombre.toLowerCase().startsWith( prefix.toLowerCase() ) )
621             {
622                 result.addItem( nombre );
623             } // fin de if
624             else
625             {
626                 // termina el ciclo si el resto de los nombres son
627                 // alfabéticamente menores que el prefijo
628                 if ( prefix.compareTo( nombre ) < 0 )
629                 {
630                     break;
631                 } // fin de if
632             } // fin de else
633
634             // desplaza el cursor a la siguiente fila de la base de datos
635             tieneElSiguiente = addressesDataProvider.cursorNext();
636         } // fin de while
637     } // fin de try
638     catch ( Exception ex )
639     {
640         result.addItem( "Excepcion al obtener nombres que coincidan." );
641     } // fin de catch
642 } // fin del método nombreAutoComplete_complete
643
644 // manejador de acciones para el buscarBoton que busca en la base de datos de
645 // la libreta de direcciones y muestra la dirección solicitada en un mapa correspondiente.
646 public String buscarBoton_action()
647 {
648     // divide el texto del campo AutoComplete en primer nombre y apellido
649     String nombre = String.valueOf( nombreAutoComplete.getValue() );
650     int splitIndex = nombre.indexOf( "," );
651     String apaterno = nombre.substring( 0, splitIndex );
652     String pnombre = nombre.substring( splitIndex + 2 );
653
654     try
655     {
656         // establece los parámetros para la consulta direccionesSeleccionadas
657         busquedaDireccionesDataProvider.getCachedRowSet().setObject(
658             1, pnombre );
659         busquedaDireccionesDataProvider.getCachedRowSet().setObject(
660             2, apaterno );
661         busquedaDireccionesDataProvider.refresh();
662         String calle = (String) busquedaDireccionesDataProvider.getValue(
663             "ADDRESSES.STREET" );

```

Figura 27.16 | Bean de página que obtiene un mapa para mostrarlo en el componente MapViewer. (Parte 3 de 4).


```

664 String ciudad = (String) busquedaDireccionesDataProvider.getValue(
665     "ADDRESSES.CITY" );
666 String estado = (String) busquedaDireccionesDataProvider.getValue(
667     "ADDRESSES.STATE" );
668 String cp = (String) busquedaDireccionesDataProvider.getValue(
669     "ADDRESSES.ZIP" );
670
671 // aplica formato a la dirección para Google Maps
672 String direccionGoogle = calle + ", " + ciudad + ", " + estado +
673     " " + cp;
674
675 // obtiene los puntos geográficos para la dirección
676 GeoPoint puntos[] = geoCoder.geoCode( direccionGoogle );
677
678 // si Google Maps no puede encontrar la dirección
679 if ( puntos == null )
680 {
681     error( "El mapa para " + direccionGoogle + " no se pudo encontrar");
682     mapView.setRendered( false ); // oculta el mapa
683     return null;
684 } // fin de if
685
686 // centra el mapa para la dirección dada
687 mapView_center.setLatitude( puntos[ 0 ].getLatitude() );
688 mapView_center.setLongitude( puntos[ 0 ].getLongitude() );
689
690 // crea un marcador para la dirección y establece su texto a mostrar
691 mapMarker.setLatitude( puntos[ 0 ].getLatitude() );
692 mapMarker.setLongitude( puntos[ 0 ].getLongitude() );
693 mapMarker.setMarkup( pnombre + " " + apaterno + "<br/>" + calle +
694     "<br/>" + ciudad + ", " + estado + " " + cp );
695
696 mapView.setRendered( true ); // muestra el mapa
697 } // fin de try
698 catch ( Exception e )
699 {
700     error( "Error al procesar búsqueda. " + e.getMessage() );
701 } // fin de catch
702
703 return null;
704 } // fin del método buscarBoton_action
705 } // fin de la clase LibretaDirecciones

```

Figura 27.16 | Bean de página que obtiene un mapa para mostrarlo en el componente MapViewer. (Parte 4 de 4).

En las líneas 672 y 673 se aplica formato a la dirección como un objeto String, para usarlo con la API de Google Maps. En la línea 219 se hace una llamada al método `geoCode` del componente **Geocoding Service Object** con la dirección como argumento. Este método devuelve un arreglo de objetos `GeoPoint` que representen ubicaciones que coincidan con el parámetro dirección. Los objetos `GeoPoint` proporcionan la latitud y longitud de una ubicación dada. Suministramos una dirección completa con la calle, ciudad, estado y código postal como argumento para `geoCode`, por lo que el arreglo devuelto contendrá sólo un objeto `GeoPoint`. En la línea 679 se determina si el arreglo de objetos `GeoPoint` es `null`. De ser así, la dirección no se podría encontrar, y en las líneas 681 a 683 se muestra un mensaje en el componente **Message Group**, informando al usuario sobre el error de búsqueda, se oculta el componente **Map Viewer** y se devuelve `null` para terminar el procesamiento.

En las líneas 687 a 688 se establecen la latitud y la longitud del centro del componente **Map Viewer**, en relación con la latitud y longitud del objeto `GeoPoint` que representa a la dirección seleccionada. En las líneas 691 a 694 se establecen la latitud y longitud del componente **Map Marker**, y se establece el texto a mostrar en el marcador. En la línea 696 se muestra el mapa vuelto a centrar, que contiene el componente **Map Marker** que indica la ubicación del contacto.

En las líneas 698 a 701 se atrapan las excepciones generadas en el cuerpo del método y se muestra un mensaje de error en el componente **Message Group**. Si el usuario sólo seleccionó un nombre de la lista de selecciones en el componente **AutoComplete Text Field**, no habrá errores al buscar en la base de datos, ya que se garantiza que el nombre estará en el formato *apellido paterno, primer nombre* apropiado, y estará incluido en la base de datos **LibretaDirecciones**. No incluimos código especial para manejar errores en los casos en que el usuario escribe un nombre que no se puede encontrar en la **LibretaDirecciones**, o para los nombres con formato inapropiado.

27.6 Conclusión

En este capítulo presentamos un ejemplo práctico en tres partes, acerca de cómo crear una aplicación Web que interactúe con una base de datos y proporcione una interacción intensiva con el usuario, mediante el uso de los componentes JSF habilitados para Ajax. Primero le mostramos cómo crear una aplicación **LibretaDirecciones** que permita a un usuario agregar direcciones a la **LibretaDirecciones** y explorar su contenido. Mediante este ejemplo, aprendió a insertar la entrada del usuario en una base de datos Java DB, y a mostrar el contenido de una base de datos en una página Web, mediante el uso de un componente JSF llamado **Tabla**.

Aprendió a descargar e importar la biblioteca de componentes Java BluePrints habilitada para Java. Después extendimos la aplicación **LibretaDirecciones** para incluir un componente **AutoComplete Text Field**. Le mostramos cómo usar una base de datos para mostrar sugerencias en el componente **AutoComplete Text Field**. También aprendió a utilizar formularios virtuales para enviar subconjuntos de los componentes de entrada de un formulario al servidor para procesarlos.

Por último, completamos la tercera parte de la aplicación **LibretaDirecciones** al agregar funcionalidad al formulario de búsqueda. Aprendió a utilizar los componentes **Map Viewer**, **Map Marker** y **Geocoding Service Object** de la biblioteca de componentes Java BluePrints habilitados para Ajax, para mostrar un mapa de Google que indique la ubicación de un contacto.

En el siguiente capítulo, aprenderá a crear y consumir servicios Web con Java. Utilizará el IDE Netbeans 5.5 para crear servicios Web y consumirlos desde aplicaciones de escritorio, y utilizará el IDE Java Studio Creator para consumir un servicio Web desde una aplicación Web. Si prefiere realizar todas estas tareas en un IDE, puede descargar el Netbeans Visual Web Pack 5.5 (www.netbeans.org/products/visualweb/) para Netbeans 5.5.

27.7 Recursos Web

www.deitel.com/ajax/Ajax_resourcecenter.html

Explore nuestro Centro de recursos Ajax, en donde encontrará vínculos a artículos sobre Ajax, tutoriales, aplicaciones, sitios Web comunitarios y mucho más.

developers.sun.com/prodtech/javatools/jscreator/learning/tutorials/index.jsp

Este sitio ofrece docenas de tutoriales acerca de Java Studio Creator 2. De especial interés para este capítulo son las secciones Access Databases (Acceso a bases de datos) y Work with Ajax Components (Trabajo con componentes Ajax).

developers.sun.com/prodtech/javadb/

El sitio oficial de Sun sobre Java DB; presenta las generalidades acerca de esta tecnología, y ofrece vínculos a artículos técnicos y un manual acerca del uso de bases de datos Apache Derby.

java.sun.com/reference/blueprints/

El sitio de referencia de Sun Developer Network para Java BluePrints.

blueprints.dev.java.net/

El sitio de java.net para el proyecto Java BluePrints.

blueprints.dev.java.net/ajaxcomponents.html

Información acerca de los componentes habilitados para Ajax que proporciona la biblioteca Java Blueprints.

developers.sun.com/prodtech/javatools/jscreator/reference/code/samplecomps/index.jsp

Demuestra los ocho componentes habilitados para Ajax que proporciona la biblioteca Java BluePrints.

google.com/apis/maps

Los poseedores de una cuenta de Google pueden registrarse aquí para obtener una clave y utilizar la API de Google Maps.

ajax.dev.java.net/

El marco de trabajo del proyecto jMaki Ajax para crear sus propios componentes habilitados para Java.

Resumen

Sección 27.2 Acceso a bases de datos en las aplicaciones Web

- Muchas aplicaciones Web acceden a bases de datos para almacenar y obtener datos persistentes. En esta sección creamos una aplicación Web que utiliza una base de datos Java DB para almacenar contactos en la libreta de direcciones y mostrar contactos de esta libreta en una página Web.
- La página Web permite al usuario introducir nuevos contactos en un formulario. Este formulario consiste en componentes **Campo de texto** para el primer nombre del contacto, su dirección física, ciudad, estado y código postal. El formulario también tiene un botón **Enviar** para enviar los datos al servidor, y un botón **Borrar** para restablecer los campos del formulario. La aplicación almacena la información de la libreta de direcciones en una base de datos llamada `LibretaDirecciones`, la cual tiene una sola tabla llamada `Addresses`. (En el directorio de ejemplos para este capítulo proporcionamos esta base de datos. Puede descargar los ejemplos de www.deitel.com/books/jhttp7). Este ejemplo también introduce el componente JSF **Tabla**, el cual muestra las direcciones de la base de datos en formato tabular. En breve le mostraremos cómo configurar el componente **Tabla**.
- El componente **Tabla** da formato y muestra datos de las tablas de una base de datos.
- Cambie la propiedad `title` del componente **Tabla** para especificar el texto que se va a mostrar en la parte superior de la **Tabla**.
- Para utilizar una base de datos en una aplicación Web de Java Studio Creator 2, primero debemos iniciar el servidor de bases de datos integrado del IDE, el cual incluye controladores para muchos tipos de bases de datos, incluyendo Java DB.
- Para iniciar el servidor, haga clic en la ficha **Servidores** debajo del menú **Archivo**, haga clic con el botón derecho en **Servidor Bundled Database** en la parte inferior de la ventana **Servidores** y seleccione **Iniciar Bundled Database**.
- Para agregar una base de datos Java DB a un proyecto, haga clic con el botón derecho en el nodo **Orígenes de datos** en la parte superior de la ventana **Servidores** y seleccione **Agregar origen de datos....** En el cuadro de diálogo **Agregar origen de datos**, escriba el nombre del origen de datos y seleccione **Derby** en el tipo de servidor. Especifique el ID de usuario y la contraseña para la base de datos. Para el URL de la base de datos, escriba `jdbc:derby://localhost:21527/NombreDeSuBaseDeDatos`. Este URL indica que la base de datos reside en el equipo local y acepta conexiones en el puerto 21527. Haga clic en el botón **Seleccionar** para elegir una tabla que se utilizará para validar la base de datos. Haga clic en **Seleccionar** para cerrar este cuadro de diálogo, y después haga clic en **Agregar** para agregar la base de datos como origen de datos para el proyecto y cierre el cuadro de diálogo.
- Para configurar un componente **Tabla** para mostrar los datos de una base de datos, haga clic con el botón derecho del ratón en el componente **Tabla** y seleccione **Enlazar con datos** para mostrar el cuadro de diálogo **Diseño de tabla**. Haga clic en el botón **Agregar proveedor de datos...** para mostrar el cuadro de diálogo **Agregar proveedor de datos**, el cual contiene una lista de los orígenes de datos disponibles. Expanda el nodo de la base de datos, expanda el nodo **Tablas**, seleccione una tabla y haga clic en **Agregar**. El cuadro de diálogo **Diseño de tabla** mostrará una lista de las columnas en la tabla de la base de datos. Todos los elementos bajo el encabezado **Seleccionado** se mostrarán en la **Tabla**. Para eliminar una columna de la **Tabla**, puede seleccionarla y hacer clic en el botón **<**.
- De manera predeterminada, la **Tabla** utiliza los nombres de las columnas de la tabla de la base de datos en mayúsculas como encabezados. Para modificar estos encabezados, seleccione una columna y edite su propiedad `headerText` en la ventana **Propiedades**. Para seleccionar una columna, expanda el nodo de la tabla en la ventana **Esquema** (estando en modo **Diseño**) y después seleccione el objeto columna apropiado.
- Al hacer clic en la casilla de verificación a un lado de la propiedad `paginationControl` de la tabla en la ventana **Propiedades**, se configura esta **Tabla** para paginación automática. Se mostrarán cinco filas a la vez, y se agregarán botones para avanzar hacia delante y hacia atrás, entre grupos de cinco contactos, al final de la **Tabla**.
- Al enlazar un componente **Tabla** con un proveedor de datos, se agrega un nuevo objeto `CachedRowSetDataProvider` al nodo de la aplicación en la ventana **Esquema**. Un objeto `CachedRowSetDataProvider` proporciona un objeto `RowSet` desplazable que puede enlazarse con un componente **Tabla** para mostrar los datos del objeto `RowSet`.
- El objeto `CachedRowSet` envuelto por nuestro objeto `addressesDataProvider` está configurado de manera predeterminada para ejecutar una consulta SQL que seleccione todos los datos en la tabla de la base de datos. Para editar esta consulta SQL, puede expandir el nodo `SessionBean` en la ventana **Esquema** y hacer doble clic en el elemento `RowSet` para abrir la ventana del editor de consultas.
- Cada fila en un objeto `CachedRowSetDataProvider` tiene su propia clave; el método `appendRow`, que agrega una nueva fila al objeto `CachedRowSet`, devuelve la clave para la nueva fila.
- El método `commitChanges` de la clase `CachedRowSetDataProvider` aplica los cambios en el objeto `CachedRowSet` a la base de datos.

- El método `refresh` de `CachedRowSetDataProvider` vuelve a ejecutar la instrucción SQL del objeto `CachedRowSet` envuelto.

Sección 27.3 Componentes JSF habilitados para Ajax

- El término Ajax (JavaScript y XML asíncronos) fue ideado por Jesse James Garrett de Adaptive Path, Inc. en febrero de 2005, para describir un rango de tecnologías para desarrollar aplicaciones Web dinámicas y con gran capacidad de respuesta.
- Ajax separa la parte correspondiente a la interacción con el usuario de una aplicación, de la interacción con su servidor, permitiendo que ambas procedan en forma asincrónica y en paralelo. Esto permite a las aplicaciones Ajax basadas en Web ejecutarse a velocidades que se asemejan a las de las aplicaciones de escritorio.
- Ajax realiza llamadas asíncronas al servidor para intercambiar pequeñas cantidades de datos con cada llamada.
- Ajax permite que se vuelvan a cargar sólo las porciones necesarias de la página, lo cual ahorra tiempo y recursos.
- Las aplicaciones Ajax contienen marcado de XHTML y CSS al igual que cualquier otra página Web, y hacen uso de las tecnologías de secuencias de comandos del lado cliente (como JavaScript) para interactuar con los elementos de las páginas.
- El objeto `XMLHttpRequestObject` permite los intercambios asíncronos con el servidor Web, lo cual hace que las aplicaciones Ajax tengan una gran capacidad de respuesta. Este objeto se puede utilizar en la mayoría de los lenguajes de secuencias de comandos para pasar datos XML del cliente al servidor, y para procesar los datos XML que el servidor envía de vuelta al cliente.
- Las bibliotecas Ajax facilitan el proceso de aprovechar los beneficios de Ajax en las aplicaciones Web, sin tener que escribir Ajax “puro”.
- La biblioteca de componentes Java BluePrints habilitados para Ajax proporciona componentes JSF habilitados para Ajax.
- Para utilizar los componentes Java BluePrints habilitados para Ajax en Java Studio Creator 2, debe descargarlos e importarlos. El IDE proporciona un asistente para instalar este grupo de componentes. Seleccione **Herramientas > Centro de actualización** para mostrar el cuadro de diálogo **Asistente del centro de actualización**. Haga clic en **Siguiente >** para buscar actualizaciones disponibles. En el área **Nuevos módulos y actualizaciones disponibles** del cuadro de diálogo, seleccione **BluePrints AJAX Components** y haga clic con el botón derecho del ratón en el botón de flecha (>) para agregarlo a la lista de elementos que desea instalar. Haga clic en **Siguiente >** y siga los indicadores para aceptar las condiciones de uso y descargar los componentes. Cuando se complete la descarga, haga clic en **Siguiente** y luego en **Terminar**. Haga clic en **Aceptar** para reiniciar el IDE.
- A continuación, debe importar los componentes en la **Paleta**. Seleccione **Herramientas > Administrador de bibliotecas de componentes** y después haga clic en **Importar...** Haga clic en el botón **Examinar...** en el cuadro de diálogo **Importar biblioteca de componentes** que aparezca. Seleccione el archivo `ui.comp1ib` y haga clic en **Abrir**. Haga clic en **Aceptar** para importar los componentes **BluePrints AJAX Components** y **BluePrints AJAX SupportBeans**.

Sección 27.4 AutoComplete Text Field y formularios virtuales

- El componente **AutoComplete Text Field** proporciona una lista de sugerencias de un origen de datos (como una base de datos o un servicio Web) a medida que el usuario escribe.
- Los formularios virtuales se utilizan cuando deseamos que un botón envíe un subconjunto de los campos de entrada de la página al servidor.
- Los formularios virtuales se utilizan cuando deseamos que un botón envíe un subconjunto de los campos de entrada de la página al servidor.
- Los formularios virtuales nos permiten mostrar varios formularios en la misma página. Nos permiten especificar un emisor y uno o más participantes para cada formulario. Al hacer clic en el componente emisor del formulario virtual, sólo se envían al servidor los valores de sus componentes participantes.
- Para agregar formularios virtuales a la página, haga clic con el botón derecho en el componente emisor que se encuentra en el formulario, y seleccione **Configurar formularios virtuales...** en el menú contextual para que aparezca el cuadro de diálogo **Configurar formularios virtuales**. Haga clic en **Nuevo** para agregar un formulario virtual; después haga clic en la columna **Nombre** y especifique el nombre del nuevo formulario. Haga doble clic en la columna **Enviar** y cambie la opción a **Sí** para indicar que este botón se debe utilizar para enviar el formulario virtual **agregar-Form**. Haga clic en **Aceptar** para salir del cuadro de diálogo. Después, seleccione todos los componentes de entrada que participarán en el formulario virtual. Haga clic con el botón derecho del ratón en uno de los componentes seleccionados y elija la opción **Configurar formularios virtuales...** En la columna **Función** del formulario virtual apropiado, cambie la opción a **Sí** para indicar que los valores en estos componentes deben enviarse al servidor cuando se envíe el formulario.

- Para ver los formularios virtuales en modo **Diseño**, haga clic en el botón **Mostrar formularios virtuales** (🔍) en la parte superior del panel Diseñador visual para mostrar una leyenda de los formularios virtuales en la página.
- El atributo `completionMethod` de un componente **AutoComplete Text Field** está enlazado al manejador de eventos `complete` de un bean de página. Para crear este método, haga clic con el botón derecho en el componente **AutoComplete Text Field** en vista **Diseño** y seleccione **Editar controlador de eventos > complete**.
- El manejador de eventos `complete` se invoca después de cada pulsación de tecla en un componente **AutoComplete Text Field** para actualizar la lista de sugerencias con base en el texto que el usuario ha escrito hasta cierto punto. El método recibe una cadena que contiene el texto introducido por el usuario, junto con un objeto `CompletionResult` que se utiliza para mostrar sugerencias al usuario.

Sección 27.5 Componente Map Viewer de Google Maps

- Un componente **Map Viewer** habilitado para Ajax utiliza el servicio Web de la API de Google Maps para buscar y mostrar mapas. Un componente **Map Marker** apunta a una ubicación en un mapa.
- Para utilizar el componente **Map Viewer**, debe tener una cuenta con Google. Visite el sitio <https://www.google.com/accounts/ManageAccount> para registrarse y obtener una cuenta gratuita. Debe obtener una clave para usar la API de Google Maps en www.google.com/apis/maps. La clave que reciba será específica para su aplicación Web y limitará el número de mapas que puede mostrar la aplicación por día. Cuando se registre para la clave, tendrá que escribir el URL para la aplicación que utilizará la API de Google Maps. Si va a desplegar la aplicación sólo en el servidor de prueba integrado de Java Studio Creator 2, escriba el URL `http://localhost:29080`.
- Para utilizar un componente **Map Viewer**, establezca su propiedad `key` con la clave de la API de Google Maps que obtuvo.
- Un componente **Map Marker** (de la sección **AJAX Support Beans** de la **Paleta**) marca una ubicación en un mapa. Debe enlazar el marcador con el mapa, de manera que el marcador se pueda mostrar en el mapa. Para ello, haga clic con el botón derecho en el componente **Map Viewer** en modo **Diseño** y seleccione **Enlaces de propiedades...** para mostrar el cuadro de diálogo **Enlaces de propiedades**. Seleccione **info** de la columna **Seleccionar propiedad enlazable** del cuadro de diálogo, y después seleccione `mapMarker` de la columna **Seleccionar destino de enlace**. Haga clic en **Aplicar** y después en **Cerrar**.
- Un componente **Geocoding Service Object** (de la sección **AJAX Support Beans** de la **Paleta**) convierte las direcciones de las calles en latitudes y longitudes que el componente **Map Viewer** utiliza para mostrar un mapa apropiado.
- El atributo `center` del componente **Map Viewer está enlazado a la propiedad `mapViewer_center` del bean de página. Esta propiedad se manipula en el archivo de bean de página, para centrar el mapa en la dirección deseada.**
- El método `geoCode` del componente **Geocoding Service Object** recibe una dirección como un argumento y devuelve un arreglo de objetos `GeoPoint` que representan ubicaciones, las cuales coinciden con el parámetro dirección. Los objetos `GeoPoint` proporcionan la latitud y la longitud de una ubicación dada.

Terminología

Ajax (JavaScript y XML asíncronos)

Apache Derby

API de Google Maps

AutoComplete Text Field, componente JSF

biblioteca de componentes Java BluePrints habilitados para Ajax

bibliotecas de componentes habilitadas para Ajax

Button, componente JSF

Buy Now Button, componente JSF

`CachedRowSet`, interfaz

`CachedRowSetDataProvider`, clase

ciclo de vida del procesamiento de eventos

`commitChanges`, método de la clase `CachedRowSetDataProvider`

componentes JSF habilitados para Ajax

elemento JSF

emisor en un formulario virtual

enlazar una **Tabla** JSF con la tabla de una base de datos

formulario virtual

`geoCode`, método de un componente **Geocoding Service Object**

Geocoding Service Object, componente

Google Maps

Java BluePrints

Java DB

JavaServer Faces (JSF)

Jesse James Garrett

Map Marker, componente JSF

Map Viewer, componente JSF

Message Group, componente JSF

participante en un formulario virtual

Popup Calendar, componente JSF

`primary`, propiedad de un **Botón** JSF

Progress Bar, componente JSF

proveedor de datos

Rating, componente JSF

`refresh`, método de la clase `CachedRowSetDataProvider`

reset, propiedad de un **Botón JSF**
Rich Textarea Editor, componente JSF
Select Value Text Field, componente JSF
 servidor de bases de datos integrado
Servidores, ficha en Java Studio Creator 2

Tabla, componente JSF
 ui:staticText, elemento JSF
 ui:table, elemento JSF
 ui:tableRowGroup, elemento JSF
 XMLHttpRequestObject

Ejercicios de autoevaluación

- 27.1** Conteste con *verdadero* o *falso* a cada una de las siguientes proposiciones; en caso de ser *falso*, explique por qué.
- El componente JSF **Tabla** nos permite desplegar otros componentes y texto en formato tabular.
 - Los formularios virtuales permiten mostrar varios formularios (cada uno con su botón **Enviar**) en la misma página Web.
 - Un componente `CachedRowSetDataProvider` se almacena en el objeto `SessionBean` y ejecuta consultas SQL para proporcionar componentes **Tabla** con datos a mostrar.
 - El objeto `XMLHttpRequestObject` proporciona acceso al objeto petición de una página.
 - El manejador de eventos `complete` para un componente **AutoComplete Text Field** se llama después de cada pulsación de tecla en el campo de texto, para proporcionar una lista de sugerencias con base en lo que ya se ha escrito.
 - Un proveedor de datos vuelve a ejecutar automáticamente su comando SQL para proporcionar información actualizada de la base de datos en cada actualización de página.
 - Para volver a centrar un componente **Map Viewer**, debe establecer la longitud y latitud del centro del mapa.
- 27.2** Complete los siguientes enunciados.
- Ajax es un acrónimo para _____.
 - El método _____ de la clase _____ actualiza una base de datos para reflejar los cambios realizados en el proveedor de datos de la base de datos.
 - Un _____ es un componente de soporte utilizado para traducir direcciones en latitudes y longitudes, para mostrarlas en un componente **Map Viewer**.
 - Un formulario virtual especifica que ciertos componentes JSF son _____ cuyos datos se enviarán cuando se haga clic en el componente emisor.
 - Los componentes Ajax para Java Studio Creator 2, como **AutoComplete Text Field** y **Map Viewer**, son proporcionados por _____.

Respuestas a los ejercicios de autoevaluación

- 27.1** a) Falso. Los componentes **Tabla** se utilizan para mostrar datos de las bases de datos. b) Verdadero. c) Falso. El componente `CachedRowSetDataProvider` es una propiedad del bean de página. Envuelve un objeto `CachedRowSet`, el cual se almacena en el objeto `SessionBean` y ejecuta consultas SQL. d) Falso. El objeto `XMLHttpRequestObject` es un objeto que permite intercambios asíncronos con un servidor Web. e) Verdadero. f) Falso. Debe llamar al método `refresh` en el proveedor de datos para volver a ejecutar el comando SQL. g) Verdadero.
- 27.2** a) JavaScript y XML asíncronos. b) `commitChanges`, `CachedRowSetDataProvider`. c) **Geocoding Service Object**. d) participantes. e) La biblioteca de componentes Java Blueprint habilitados para Ajax.

Ejercicios

27.3 (*Aplicación LibroVisitantes*) Cree una página Web JSF que permita a los usuarios registrarse en un libro de visitantes y verlo. Use la base de datos `LibroVisitantes` (que se proporciona en el directorio de ejemplos para este capítulo) para almacenar las entradas en el libro de visitantes. La base de datos `LibroVisitantes` tiene una sola tabla llamada `Messages`, la cual contiene cuatro columnas: `date`, `name`, `email` y `message`. La base de datos contiene unas cuantas entradas de ejemplo. En la página Web, proporcione componentes **Campo de texto** para el nombre del usuario y la dirección de correo electrónico, y un componente **Área de texto** para el mensaje. Agregue un **Botón Enviar** y un componente **Tabla**, y configure la **Tabla** para mostrar las entradas en el libro de visitantes. Use el método manejador de acciones del **Botón Enviar** para insertar una nueva fila que contenga la entrada del usuario y la fecha de hoy en la base de datos `LibroVisitantes`.

27.4 (*Modificación a la aplicación LibretaDirecciones*) Modifique la aplicación `LibretaDirecciones`, de manera que los usuarios introduzcan búsquedas en el componente **AutoComplete TextField**, en el formato *primer nombre, apellido paterno*. Necesitará agregar un nuevo proveedor de datos (o modificar el existente) para ordenar las filas en la base de datos `LibretaDirecciones` por primer nombre, y después por apellido paterno.

27.5 (*Aplicación de búsqueda en mapas*) Cree una página Web JSF que permita a los usuarios obtener un mapa de cualquier dirección. Recuerde que la búsqueda de una ubicación mediante la API de Google Maps devuelve un arreglo de objetos `GeoPoint`. Busque las ubicaciones que introduzca el usuario en un **Campo de texto**, y muestre un mapa en la primera ubicación del arreglo `GeoPoint` resultante. Para manejar varios resultados de búsqueda, muestre todos los resultados en un componente **Cuadro de lista**. Para obtener una representación de cadena de cada resultado, invoque el método `toString` en un objeto `GeoPoint`. Agregue un **Botón** que permita a los usuarios seleccionar un resultado del **Cuadro de lista** y muestre un mapa para ese resultado con un componente **Map Marker** que muestre la ubicación en el mapa. Por último, utilice un componente **Grupo de mensajes** para mostrar los mensajes relacionados con los errores de búsqueda. En caso de un error, y cuando la página se cargue por primera vez, vuelva a centrar el mapa en una ubicación predeterminada de su preferencia.

28.1	Introducción
28.1.1	Descarga, instalación y configuración de Netbeans 5.5 y Sun Java System Application Server
28.1.2	Centro de recursos de servicios Web y Centros de recursos sobre Java en www.deitel.com
28.2	Fundamentos de los servicios Web de Java
28.3	Creación, publicación, prueba y descripción de un servicio Web
28.3.1	Creación de un proyecto de aplicación Web y cómo agregar una clase de servicio Web en Netbeans
28.3.2	Definición del servicio Web EnteroEnorme en Netbeans
28.3.3	Publicación del servicio Web EnteroEnorme desde Netbeans
28.3.4	Prueba del servicio Web EnteroEnorme con la página Web Tester de Sun Java System Application Server
28.3.5	Descripción de un servicio Web con el Lenguaje de descripción de servicios Web (WSDL)
28.4	Cómo consumir un servicio Web
28.4.1	Creación de un cliente para consumir el servicio Web EnteroEnorme
28.4.2	Cómo consumir el servicio Web EnteroEnorme
28.5	SOAP
28.6	Rastreo de sesiones en los servicios Web
28.6.1	Creación de un servicio Web Blackjack
28.6.2	Cómo consumir el servicio Web Blackjack
28.7	Cómo consumir un servicio Web controlado por base de datos desde una aplicación Web
28.7.1	Configuración de Java DB en Netbeans y creación de la base de datos Reservacion
28.7.2	Creación de una aplicación Web para interactuar con el servicio Web Reservacion
28.8	Cómo pasar un objeto de un tipo definido por el usuario a un servicio Web
28.9	Conclusión
28.10	Recursos Web
Resumen Terminología Ejercicios de autoevaluación Respuestas a los ejercicios de autoevaluación Ejercicios	

28.1 Introducción

En este capítulo presentaremos los servicios Web, los cuales promueven la portabilidad y reutilización de software en aplicaciones que operan a través de Internet. Un **servicio Web** es un componente de software almacenado en una computadora, el cual se puede utilizar mediante llamadas a métodos desde una aplicación (u otro componente de software) en otra computadora, a través de una red. Los servicios Web se comunican mediante el uso de tecnologías como XML y HTTP. Varias APIs de Java facilitan los servicios Web. En este capítulo trataremos con APIs basadas en el **Protocolo simple de acceso a objetos (SOAP)**: un protocolo basado en XML que permite la comunicación entre los clientes y los servicios Web, aun cuando el cliente y el servicio Web estén escritos en distintos lenguajes. Hay otras tecnologías de servicios Web, como la Transferencia representativa de estado (REST), que no veremos en este capítulo. Para obtener información acerca de REST, consulte los recursos Web de la sección 28.10 y visite nuestro Centro de recursos sobre servicios Web en

www.deitel.com/WebServices

La Biblioteca Deitel de contenido gratuito (Deitel Free Content Library) incluye los siguientes tutoriales de introducción a XML:

www.deitel.com/articles/xml_tutorials/20060401/XMLBasics/
www.deitel.com/articles/xml_tutorials/20060401/XMLStructuringData/

Los servicios Web tienen grandes implicaciones para las **transacciones de negocio a negocio (B2B)**. Permiten a los negocios realizar transacciones a través de servicios Web estandarizados, con una amplia disponibilidad,

en vez de depender de aplicaciones propietarias. Los servicios Web y SOAP son independientes de la plataforma y del lenguaje, por lo que las compañías pueden colaborar a través de servicios Web sin tener que preocuparse por la compatibilidad de sus tecnologías de hardware, software y comunicaciones. Compañías como Amazon, Google, eBay, PayPal y muchas otras están usando servicios Web para su beneficio, al facilitar sus aplicaciones del lado servidor a sus socios, a través de los servicios Web.

Al comprar servicios Web y utilizar los diversos servicios Web gratuitos que son importantes para sus actividades comerciales, las compañías invierten menor tiempo en desarrollar nuevas aplicaciones y pueden crear nuevas aplicaciones innovadoras. Los comercios electrónicos pueden usar servicios Web para proporcionar a sus clientes experiencias de compra mejoradas. Consideremos el caso de una tienda de música en línea. El sitio Web de la tienda ofrece vínculos a información sobre varios CDs, lo cual permite a los usuarios comprar los CDs, saber acerca de los artistas, buscar más títulos de ellos, buscar música de otros artistas que los usuarios puedan disfrutar, y mucho más. Otra compañía que vende boletos para conciertos proporciona un servicio Web que muestra las fechas de los próximos conciertos de varios artistas, y permite a los usuarios comprar boletos. Al consumir el servicio Web para boletos de conciertos en su sitio, la tienda de música en línea puede proporcionar un servicio adicional a sus clientes e incrementar el tráfico de su sitio, y tal vez obtener una comisión por las ventas de los boletos de conciertos. La compañía que vende boletos de conciertos también se beneficia de la relación comercial al vender más boletos y quizá al recibir ingresos de la tienda de música en línea por el uso de su servicio Web.

Cualquier programador de Java con un conocimiento sobre los servicios Web puede escribir aplicaciones que puedan “consumir” servicios Web. Las aplicaciones resultantes llamarían a los métodos de los servicios Web de objetos que se ejecuten en servidores, que podrían estar a miles de kilómetros de distancia. Para saber más acerca de los servicios Web, lea las Generalidades sobre la tecnología Java y los servicios Web (Java Technology and Web Services Overview) en java.sun.com/webservices/overview.html.

Netbeans 5.5 y Sun Java Studio Creator 2

Netbeans 5.5 y **Sun Java Studio Creator 2** (ambos desarrollados por Sun) son dos de las muchas herramientas que permiten a los programadores “publicar” y “consumir” servicios Web. Vamos a demostrar cómo usar estas herramientas para implementar servicios Web e invocarlos desde aplicaciones cliente. Para cada ejemplo, proporcionaremos el código para el servicio Web, y después presentaremos una aplicación cliente que utiliza el servicio Web. En nuestros primeros ejemplos vamos a crear servicios Web y aplicaciones cliente en Netbeans. Después demostraremos servicios Web que utilizan características más sofisticadas, como la manipulación de bases de datos con JDBC (que presentamos en el capítulo 25, Acceso a bases de datos con JDBC) y la manipulación de objetos de clases.

Sun Java Studio Creator 2 facilita el desarrollo de aplicaciones Web y el consumo de servicios Web. Netbeans proporciona aún más herramientas, incluyendo la habilidad de crear, publicar y consumir servicios Web. Utilizamos Netbeans para crear y publicar servicios Web, y para crear aplicaciones de escritorio que los consuman. Utilizamos Sun Java Studio Creator 2 para crear aplicaciones Web que consuman servicios Web. Las herramientas de Sun Java Studio Creator 2 se pueden agregar a Netbeans mediante el Netbeans Visual Web Pack. Para obtener más información acerca de este complemento de Netbeans, visite www.netbeans.org/products/visualweb/.

28.1.1 Descarga, instalación y configuración de Netbeans 5.5 y Sun Java System Application Server

Para desarrollar los servicios Web en este capítulo, utilizamos Netbeans 5.5 y Sun Java System Application Server con las opciones de instalación predeterminadas. El sitio Web de Netbeans proporciona un instalador integrado para ambos productos. Para descargar el instalador, visite el sitio

www.netbeans.org/products/ide/

y después haga clic en el botón **Download Netbeans IDE**. En la siguiente página, seleccione su sistema operativo y lenguaje, y después siga las instrucciones que aparecen en pantalla. Es posible que para cuando usted entre, ya haya una versión más reciente.

Una vez que haya descargado e instalado estas herramientas, ejecute el IDE Netbeans. En Windows XP, el instalador colocará una entrada para Netbeans en **Inicio > Todos los programas**. Antes de proceder con el resto del capítulo, realice los siguientes pasos para configurar Netbeans y permitir realizar pruebas con Sun Java System Application Server:

1. Seleccione **Tools > Server Manager** para mostrar el cuadro de diálogo **Server Manager**. Si ya aparece Sun Java System Application Server en la lista de servidores, omita los *pasos del 2 al 6*.
2. Haga clic en el botón **Add Server...** en la esquina superior izquierda del cuadro de diálogo para que aparezca el cuadro de diálogo **Add Server Instance**.
3. Seleccione **Sun Java System Application Server** de la lista desplegable **Server**, y después haga clic en **Next >**.
4. En el campo **Platform Location**, especifique la ubicación de instalación de Sun Java System Application Server; el valor predeterminado es `C:\Sun\AppServer` en Windows. Haga clic en **Next >**.
5. Especifique el nombre de usuario y la contraseña para el servidor; los valores predeterminados son `admin` y `adminadmin`, respectivamente. Haga clic en **Finish**.
6. Haga clic en **Close** para cerrar el cuadro de diálogo **Server Manager**.

28.1.2 Centro de recursos de servicios Web y Centros de recursos sobre Java en www.deitel.com

Visite nuestro Centro de recursos de servicios Web en www.deitel.com/WebServices/ para obtener información acerca de cómo diseñar e implementar servicios Web en muchos lenguajes, y para obtener información acerca de los servicios Web que ofrecen compañías como Google, Amazon y eBay. También encontrará muchas herramientas adicionales de Java para publicar y consumir servicios Web.

Nuestros Centros de recursos sobre Java en

```
www.deitel.com/java/
http://www.deitel.com/ResourceCenters/Programming/JavaSE6/tabid/1056/Default.aspx
www.deitel.com/JavaEE5/
```

ofrecen información adicional específica sobre Java, como libros, informes, artículos, diarios, sitios Web y blogs que abarcan una gran variedad de temas sobre Java (incluyendo los servicios Web de Java).

28.2 Fundamentos de los servicios Web de Java

La computadora en la que reside un servicio Web se conoce como **equipo remoto** o **servidor**. La aplicación (es decir, el cliente) que accede al servicio Web envía la llamada a un método a través de una red al equipo remoto, el cual procesa la llamada y devuelve una respuesta a la aplicación, a través de la red. Este tipo de computación distribuida es benéfica en muchas aplicaciones. Por ejemplo, una aplicación cliente sin acceso directo a una base de datos en un servidor remoto podría obtener los datos a través de un servicio Web. De manera similar, una aplicación que carezca del poder para realizar ciertos cálculos podría usar un servicio Web para aprovechar los recursos superiores de otro sistema.

En Java, un servicio Web se implementa como una clase. En capítulos anteriores, todas las piezas de una aplicación residían en un equipo. La clase que representa el servicio Web reside en un servidor; no forma parte de la aplicación cliente.

Al proceso de hacer que un servicio Web esté disponible para recibir peticiones de los clientes se le conoce como **publicación de un servicio Web**; al proceso de utilizar un servicio Web desde una aplicación cliente se le conoce como **consumo de un servicio Web**. Una aplicación que consume un servicio Web consiste de dos partes: un objeto de una **clase proxy** para interactuar con el servicio Web y una aplicación cliente que consume el servicio Web, invocando a los métodos en el objeto de la clase proxy. El código cliente invoca los métodos en el objeto proxy, el cual se encarga de los detalles de la comunicación con el servicio Web (como el paso de los argumentos a los métodos del servicio Web y la recepción de los valores de retorno del servicio Web) por el cliente. Esta comunicación se puede llevar a cabo a través de una red local, de Internet o inclusive con un servicio Web en la misma computadora. El servicio Web realiza la tarea correspondiente y devuelve los resultados al objeto proxy, el cual devuelve entonces los resultados al código cliente. En la figura 28.1 se muestran las interacciones entre el código cliente, la clase proxy y el servicio Web. Como pronto veremos, Netbeans y Sun Java Studio Creator 2 crean estas clases proxy por usted en sus aplicaciones cliente.

Las peticiones a los servicios Web, y las respuestas de éste que se crean con **JAX-WS 2.0** (el marco de trabajo más reciente de servicios Web de Java) se transmiten, por lo regular, a través de SOAP. Cualquier cliente capaz de generar y procesar mensajes SOAP puede interactuar con un servicio Web, sin importar el lenguaje en el que esté escrito. En la sección 28.5 hablaremos sobre SOAP.

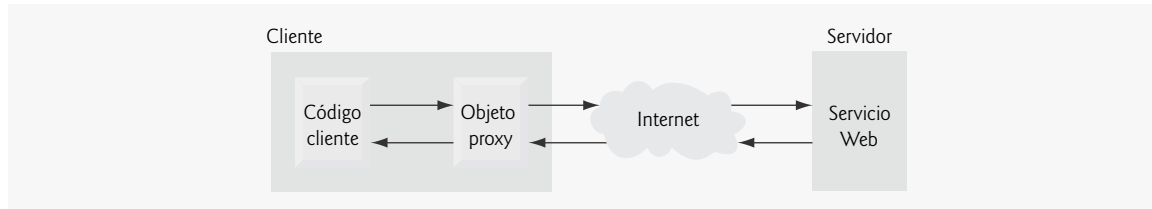


Figura 28.1 | Interacción entre un servicio Web y su cliente.

28.3 Creación, publicación, prueba y descripción de un servicio Web

Las siguientes subsecciones demuestran cómo crear, publicar y probar un servicio Web llamado **EnteroEnorme**, el cual realiza cálculos con enteros positivos de hasta 100 dígitos de longitud (que se mantienen como arreglos de dígitos). Dichos enteros son mucho más grandes de lo que los tipos primitivos integrales de Java pueden representar. El servicio Web **EnteroEnorme** proporciona métodos que reciben dos “enteros enormes” (representados como objetos `String`) y determinan su suma, su diferencia, cuál es mayor, cuál es menor o si los dos números son iguales. Estos métodos serán servicios disponibles para otras aplicaciones a través de la Web (de ahí que se les llame servicios Web).

28.3.1 Creación de un proyecto de aplicación Web y cómo agregar una clase de servicio Web en Netbeans

Al crear un servicio Web en Netbeans, nos enfocamos en la lógica del servicio Web y dejamos que el IDE se encargue de la infraestructura del servicio Web. Para crear un servicio Web en Netbeans, primero debemos crear un proyecto de tipo **aplicación Web (Web Application)**. Netbeans utiliza este tipo de proyecto para las aplicaciones Web que se ejecutan en clientes basados en navegador, y para los servicios Web invocados por otras aplicaciones.

Creación de un proyecto de aplicación Web en Netbeans 5.5

Para crear una aplicación Web, siga los siguientes pasos:

1. Seleccione **File > New Project** para abrir el cuadro de diálogo **New Project**.
2. Seleccione **Web** de la lista **Categories** del cuadro de diálogo, y después seleccione **Web Application** de la lista **Projects**. Haga clic en **Next >**.
3. Especifique el nombre de su proyecto (**EnteroEnorme**) en el campo **Project Name**, y en el campo **Project Location** especifique en dónde le gustaría guardar el proyecto. Puede hacer clic en el botón **Browse** para seleccionar la ubicación.
4. Seleccione **Sun Java System Application Server** de la lista desplegable **Server**.
5. Seleccione **Java EE 5** de la lista desplegable **J2EE Version**.
6. Haga clic en **Finish** para cerrar el cuadro de diálogo **New Project**.

Esto creará una aplicación Web que se ejecutará en un navegador Web. Al crear una **aplicación Web (Web Application)** en Netbeans, el IDE genera archivos adicionales que ofrecen soporte a la aplicación Web. En este capítulo, hablaremos sólo sobre los archivos específicos para los servicios Web.

Cómo agregar una clase de servicio Web a un proyecto de Aplicación Web

Para crear un servicio Web, realice los siguientes pasos para agregar una clase de servicio Web al proyecto:

1. En la ficha **Projects** en Netbeans (justo debajo del menú **File**), haga clic con el botón derecho del ratón en el nodo del proyecto **EnteroEnorme** y seleccione **New > Web Service...** para que aparezca el cuadro de diálogo **New Web Service**.
2. Especifique **EnteroEnorme** en el campo **Web Service Name**.
3. Especifique **com.deitel.jhttp7.cap28.enteroenorme** en el campo **Package**.
4. Haga clic en **Finish** para cerrar el cuadro de diálogo **New Web Service**.

El IDE genera una clase de servicio Web de ejemplo con el nombre que especificó en el *paso 2*. En esta clase, definirá los métodos que su servicio Web tiene disponibles para las aplicaciones cliente. Cuando en cierto momento genere su aplicación, el IDE generará otros archivos de soporte (que veremos en breve) para su servicio Web.

28.3.2 Definición del servicio Web EnteroEnorme en Netbeans

La figura 28.2 contiene el código del servicio Web EnteroEnorme. Puede implementar este código por su cuenta en el archivo EnteroEnorme.java que se creó en la sección 28.3.1, o puede simplemente sustituir el código en EnteroEnorme.java con una copia de nuestro código de la carpeta de este ejemplo. Encontrará este archivo en la carpeta del proyecto `src\java\com\deitel\jhttp7\cap28\enteroenorme`. Puede descargar los ejemplos del libro en www.deitel.com/books/jhttp7.

```

1 // Fig. 28.2: EnteroEnorme.java
2 // Servicio Web EnteroEnorme que realiza operaciones con enteros grandes.
3 package com.deitel.jhttp7.ch28.enteroenorme;
4
5 import javax.jws.WebService; // el programa usa la anotación @WebService
6 import javax.jws.WebMethod; // el programa usa la anotación @WebMethod
7 import javax.jws.WebParam; // el programa usa la anotación @WebParam
8
9 @WebService( // anota la clase como un servicio Web
10     name = "EnteroEnorme", // establece el nombre de la clase
11     serviceName = "ServicioEnteroEnorme" ) // establece el nombre del servicio
12 public class EnteroEnorme
13 {
14     private final static int MAXIMO = 100; // máximo número de dígitos
15     public int[] numero = new int[ MAXIMO ]; // almacena el entero enorme
16
17     // devuelve una representación String de un EnteroEnorme
18     public String toString()
19     {
20         String valor = "";
21
22         // convierte EnteroEnorme en un objeto String
23         for ( int digito : numero )
24             valor = digito + valor; // coloca el siguiente dígito al principio del valor
25
26         // localiza la posición del primer dígito distinto de cero
27         int longitud = valor.length();
28         int posicion = -1;
29
30         for ( int i = 0; i < longitud; i++ )
31         {
32             if ( valor.charAt( i ) != '0' )

```

Figura 28.2 | Servicio Web EnteroEnorme que realiza operaciones con enteros grandes. (Parte 1 de 3).

```

33         {
34             posicion = i; // primer dígito distinto de cero
35             break;
36         }
37     } // fin de for
38
39     return ( posicion != -1 ? valor.substring( posicion ) : "0" );
40 } // fin del método toString
41
42 // crea un objeto EnteroEnorme a partir de un objeto String
43 public static EnteroEnorme analizarEnteroEnorme( String s )
44 {
45     EnteroEnorme temp = new EnteroEnorme();
46     int tamano = s.length();
47
48     for ( int i = 0; i < tamano; i++ )
49         temp.numero[ i ] = s.charAt( tamano - i - 1 ) - '0';
50
51     return temp;
52 } // fin del método analizarEnteroEnorme
53
54 // Método Web que suma enteros enormes representados por argumentos String
55 @WebMethod( operationName = "sumar" )
56 public String sumar( @WebParam( name = "primero" ) String primero,
57                     @WebParam( name = "segundo" ) String segundo )
58 {
59     int acarreo = 0; // el valor que se va a acarrear
60     EnteroEnorme operando1 = EnteroEnorme.analizarEnteroEnorme( primero );
61     EnteroEnorme operando2 = EnteroEnorme.analizarEnteroEnorme( segundo );
62     EnteroEnorme resultado = new EnteroEnorme(); // almacena el resultado de la suma
63
64     // realiza la suma en cada dígito
65     for ( int i = 0; i < MAXIMO; i++ )
66     {
67         // suma los dígitos correspondientes en cada número y el valor acarreado;
68         // almacena el resultado en la columna correspondiente del resultado EnteroEnorme
69         resultado.numero[ i ] =
70             ( operando1.numero[ i ] + operando2.numero[ i ] + acarreo ) % 10;
71
72         // establece el acarreo para la siguiente columna
73         acarreo =
74             ( operando1.numero[ i ] + operando2.numero[ i ] + acarreo ) / 10;
75     } // fin de for
76
77     return resultado.toString();
78 } // fin del Método Web sumar
79
80 // Método Web que resta los enteros representados por argumentos String
81 @WebMethod( operationName = "restar" )
82 public String restar( @WebParam( name = "primero" ) String primero,
83                     @WebParam( name = "segundo" ) String segundo )
84 {
85     EnteroEnorme operando1 = EnteroEnorme.analizarEnteroEnorme( primero );
86     EnteroEnorme operando2 = EnteroEnorme.analizarEnteroEnorme( segundo );
87     EnteroEnorme resultado = new EnteroEnorme(); // almacena la diferencia
88
89     // resta el dígito inferior del dígito superior
90     for ( int i = 0; i < MAXIMO; i++ )
91     {

```

Figura 28.2 | Servicio Web EnteroEnorme que realiza operaciones con enteros grandes. (Parte 2 de 3).

```

92         // si el dígito en operando1 es menor que el correspondiente
93         // dígito en operando2, pide prestado al siguiente dígito
94         if ( operando1.numero[ i ] < operando2.numero[ i ] )
95             operando1.pedirprestado( i );
96
97         // resta los dígitos
98         resultado.numero[ i ] = operando1.numero[ i ] - operando2.numero[ i ];
99     } // fin de for
100
101     return resultado.toString();
102 } // fin del Método Web restar
103
104 // pide prestado 1 del siguiente dígito
105 private void pedirprestado( int lugar )
106 {
107     if ( lugar >= MAXIMO )
108         throw new IndexOutOfBoundsException();
109     else if ( numero[ lugar + 1 ] == 0 ) // si el siguiente dígito es cero
110         pedirprestado( lugar + 1 ); // pide prestado del siguiente dígito
111
112     numero[ lugar ] += 10; // suma 10 al dígito que prestó
113     --numero[ lugar + 1 ]; // resta uno al dígito de la izquierda
114 } // fin del método pedirprestado
115
116 // Método Web que devuelve true si el primer entero es mayor que el segundo
117 @WebMethod( operationName = "mayor" )
118 public boolean mayor( @WebParam( name = "primero" ) String primero,
119                     @WebParam( name = "segundo" ) String segundo )
120 {
121     try // trata de restar el primer número del segundo
122     {
123         String diferencia = restar( primero, segundo );
124         return !diferencia.matches( "^[0]+$" );
125     } // fin de try
126     catch ( IndexOutOfBoundsException e ) // primero es menor que segundo
127     {
128         return false;
129     } // fin de catch
130 } // fin del Método Web mayor
131
132 // Método Web que devuelve true si el primer entero es menor que el segundo
133 @WebMethod( operationName = "menor" )
134 public boolean menor( @WebParam( name = "primero" ) String primero,
135                     @WebParam( name = "segundo" ) String segundo )
136 {
137     return mayor( segundo, primero );
138 } // fin del Método Web menor
139
140 // Método Web que devuelve true si el primer entero es igual al segundo
141 @WebMethod( operationName = "igual" )
142 public boolean igual( @WebParam( name = "primero" ) String primero,
143                     @WebParam( name = "segundo" ) String segundo )
144 {
145     return !( mayor( primero, segundo ) || menor( primero, segundo ) );
146 } // fin del Método Web igual
147 } // fin de la clase EnteroEnorme

```

Figura 28.2 | Servicio Web EnteroEnorme que realiza operaciones con enteros grandes. (Parte 3 de 3).

En las líneas 5 a 7 se importan las anotaciones que se utilizan en este ejemplo. De manera predeterminada, cada nueva clase de servicio Web que se crea con las APIs de JAX-WS es un **POJO (Objeto Java simple)**, lo cual significa que, a diferencia de las APIs de servicios Web de Java anteriores, no necesita extender una clase o implementar una interfaz para crear un servicio Web. Al compilar una clase que utiliza estas anotaciones de JAX-WS 2.0, el compilador crea todos los **artefactos del lado servidor** que soportan el servicio Web; es decir, el marco de trabajo de código compilado que permite al servicio Web esperar las peticiones de los clientes y responder a esas peticiones, una vez que el servicio se despliega en un servidor de aplicaciones. Algunos servidores de aplicaciones populares que soportan los servicios Web de Java son: Sun Java System Application Server (www.sun.com/software/products/appsrvr/index.xml), GlassFish (glassfish.dev.java.net), Apache Tomcat (tomcat.apache.org), BEA Weblogic Server (www.bea.com), y JBoss Application Server (www.jboss.org/products/jbossas). En este capítulo utilizaremos Sun Java System Application Server.

Las líneas 9 a 11 contienen una anotación `@WebService` (importada en la línea 5) con las propiedades `name` y `serviceName`. La **anotación `@WebService`** indica que la clase `EnteroEnorme` representa a un servicio Web. La anotación va seguida por un conjunto de paréntesis que contienen elementos opcionales. El **elemento `name`** de la anotación (línea 10) especifica el nombre de la clase proxy que se generará para el cliente. El **elemento `serviceName`** (línea 11) especifica el nombre de la clase que el cliente debe utilizar para obtener un objeto de la clase proxy. [Nota: si el elemento `serviceName` no se especifica, se asume que el nombre del servicio Web será el nombre de la clase, seguido por la palabra `Service`]. Netbeans coloca la anotación `@WebService` al principio de cada nueva clase de servicio Web que el programador crea. Después se pueden agregar las propiedades `name` y `serviceName` en el paréntesis que sigue a la anotación.

En la línea 14 se declara la constante `MAXIMO`, la cual especifica el número máximo de dígitos para un `EnteroEnorme` (en este ejemplo, 100). En la línea 15 se crea el arreglo que almacena los dígitos en un entero enorme. En las líneas 18 a 40 se declara el método `toString`, el cual devuelve una representación `String` de un `EnteroEnorme`, sin ceros a la izquierda. En las líneas 43 a 52 se declara el método `static` llamado `analizarEnteroEnorme`, el cual convierte un objeto `String` en un objeto `EnteroEnorme`. Los métodos `sumar`, `restar`, `mayor`, `menor` e `igual` del servicio Web utilizan `analizarEnteroEnorme` para convertir sus argumentos `String` en objetos `EnteroEnorme` para procesarlos.

Los métodos `sumar`, `restar`, `mayor`, `menor` e `igual` de `EnteroEnorme` se marcan con la **anotación `@WebMethod`** (líneas 55, 81, 117, 133 y 141) para indicar que pueden llamarse en forma remota. Los métodos que no se marcan con `@WebMethod` no son accesibles para los clientes que consumen el servicio Web. Por lo general, dichos miembros son métodos utilitarios dentro de la clase del servicio Web. Observe que cada una de las anotaciones `@WebMethod` utiliza el elemento `operationName` para especificar el nombre del método que está expuesto al cliente del servicio Web.



Error común de programación 28.1

Si no se expone un método como método Web, declarándolo con la anotación `@WebMethod`, los clientes del servicio Web no podrán acceder a ese método.



Error común de programación 28.2

Los métodos con la anotación `@WebMethod` no pueden ser `static`. Debe existir un objeto de la clase de servicio Web para que un cliente pueda acceder a los métodos del servicio Web.

Cada uno de los métodos Web en la clase `EnteroEnorme` especifica parámetros que se anotan con la **anotación `@WebParam`** (por ejemplo, las líneas 56 y 57 del método `sumar`). El elemento `name` opcional de `@WebParam` indica el nombre del parámetro que está expuesto a los clientes del servicio Web.

En las líneas 55 a 78 y 81 a 102 se declaran los métodos Web `sumar` y `restar` de `EnteroEnorme`. Para simplificar, vamos a suponer que `sumar` no produce un desbordamiento (es decir, el resultado será de 100 dígitos o menor) y que el primer argumento para `restar` siempre será mayor que el segundo. El método `restar` llama al método `pedirprestado` (líneas 105 a 114) cuando es necesario pedir prestado 1 al siguiente dígito a la izquierda en el primer argumento; es decir, cuando un dígito específico en el operando izquierdo es menor que el dígito correspondiente en el operando derecho. El método `pedirprestado` suma 10 al dígito apropiado y resta 1 al siguiente dígito a la izquierda. Este método utilitario no está diseñado para ser llamado en forma remota, por lo que no se marca con `@WebMethod`.

En las líneas 117 a 130 se declara el método Web `mayor` de `EnteroEnorme`. En la línea 123 se invoca el método `restar` para calcular la diferencia entre los números. Si el primer número es menor que el segundo, se produce una excepción. En este caso, `mayor` devuelve `false`. Si `restar` no lanza una excepción, entonces en la línea 124 se devuelve el resultado de la expresión

```
!diferencia.matches( "^[0]+$" )
```

Esta expresión llama al método `String matches` para determinar si el objeto `String` `diferencia` coincide con la expresión regular `"^[0]+$"`, la cual determina si el objeto `String` consiste sólo de uno o más ceros. Los símbolos `^` y `$` indican que `matches` debe devolver `true` sólo si todo el objeto `String` `diferencia` coincide con la expresión regular. Después utilizamos el operador lógico de negación (`!`) para devolver el valor booleano opuesto. De esta forma, si los números son iguales (es decir, si su diferencia es 0), la expresión anterior devuelve `false`; el primer número no es mayor que el segundo. En cualquier otro caso, la expresión devuelve `true`. En la sección 30.7 hablaremos con detalle sobre las expresiones regulares.

En las líneas 133 a 146 se declaran los métodos `menor` e `igual`. El método `menor` devuelve el resultado de invocar al método `mayor` (línea 137) con los argumentos invertidos; si `primero` es menor que `segundo`, entonces `segundo` es mayor que `primero`. El método `igual` invoca a los métodos `mayor` y `menor` (línea 145). Si `mayor` o `menor` devuelven `true`, en la línea 145 se devuelve `false` debido a que los números no son iguales. Si ambos métodos devuelven `false`, los números son iguales y en la línea 145 se devuelve `true`.

28.3.3 Publicación del servicio Web `EnteroEnorme` desde Netbeans

Ahora que hemos creado la clase de servicio Web `EnteroEnorme`, utilizaremos Netbeans para generar y publicar (es decir, desplegar) el servicio Web, de manera que los clientes puedan consumir sus servicios. Netbeans se encarga de todos los detalles relacionados con la generación y despliegue de un servicio Web por nosotros. Esto incluye la creación del marco de trabajo requerido para dar soporte al servicio Web. Haga clic con el botón derecho del ratón en el nombre del proyecto (`EnteroEnorme`), en la ficha **Projects** de Netbeans para que aparezca el menú contextual que se muestra en la figura 28.3. Para determinar si hay errores de compilación en el proyecto, seleccione la opción **Build Project**. Cuando el proyecto se compile con éxito, puede seleccionar **Deploy Project** para desplegar el proyecto en el servidor que seleccionó cuando configuró la aplicación Web en la sección 28.3.1. Si el código en el proyecto ha cambiado desde la última vez que se generó, al seleccionar **Deploy Project** también se generará. Al seleccionar **Run Project** se ejecuta la aplicación Web. Si ésta no se había generado o desplegado antes, esta opción ejecuta primero estas tareas. Observe que las opciones **Deploy Project** y **Run Project** también inician

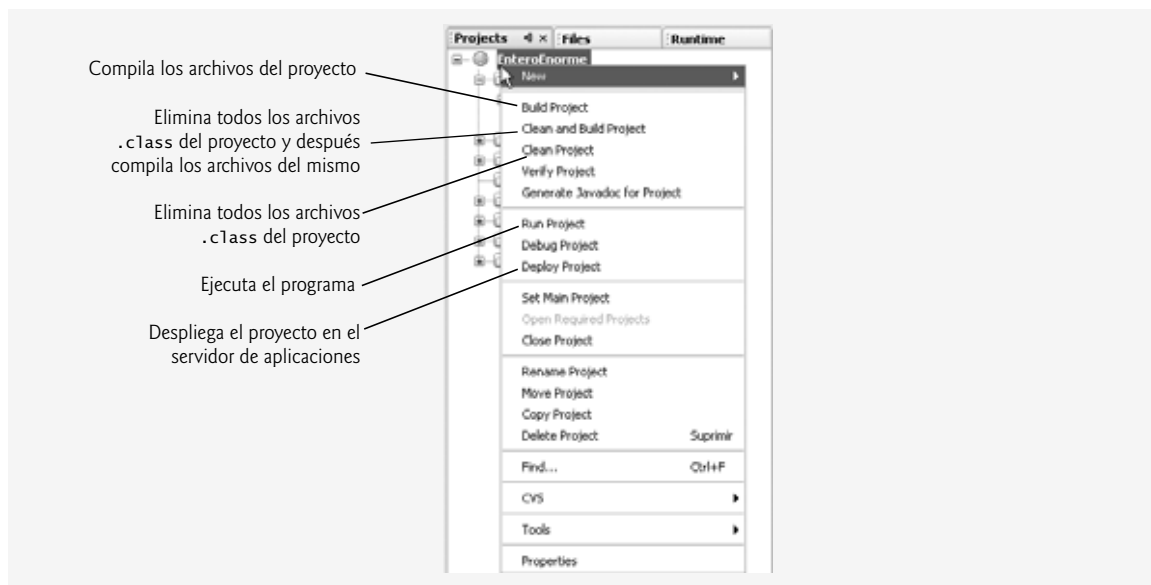


Figura 28.3 | Menú contextual que aparece al hacer clic con el botón derecho del ratón en el nombre de un proyecto, en la ficha **Projects** de Netbeans.

el servidor de aplicación (en nuestro caso, Sun Java System Application Server), en caso de que no se encuentre ya en ejecución. Para asegurar que todos los archivos de código fuente en un proyecto se vuelvan a compilar durante la siguiente operación de generación, podemos usar las opciones **Clean Project** y **Build Project**. Si no lo ha hecho todavía, seleccione **Deploy Project** ahora.

28.3.4 Prueba del servicio Web EnteroEnorme con la página Web Tester de Sun Java System Application Server

El siguiente paso es probar el servicio Web EnteroEnorme. Anteriormente seleccionamos el servidor Sun Java System Application Server para ejecutar la aplicación Web. Este servidor puede crear una página Web en forma dinámica, para probar los métodos de un servicio Web desde un navegador Web. Para habilitar esta capacidad:

1. Haga clic con el botón derecho del ratón en el nombre del proyecto (EnteroEnorme) en la ficha **Projects** de Netbeans, y seleccione **Properties** en el menú contextual para que aparezca el cuadro de diálogo **Project Properties**.
2. Haga clic en la opción **Run** de **Categories** para mostrar las opciones para ejecutar el proyecto.
3. En el campo **Relative URL**, escriba `/HugeIntegerService?Tester`.
4. Haga clic en **OK** para cerrar el cuadro de diálogo **Project Properties**.

El campo **Relative URL** especifica lo que debe ocurrir cuando se ejecute la aplicación Web. Si este campo está vacío, entonces se mostrará la JSP predeterminada de la aplicación Web al ejecutar el proyecto. Si especificamos `/EnteroEnormeService?Tester` en este campo y después ejecutamos el proyecto, Sun Java System Application Server genera la página Web Tester y la carga en el navegador Web. En la figura 28.4 se muestra la página Web Tester para el servicio Web EnteroEnorme. Una vez que haya desplegado el servicio Web, también puede escribir el URL

`http://localhost:8080/EnteroEnorme/EnteroEnormeService?Tester`



Figura 28.4 | Página Web Tester creada por Sun Java System Application Server para el servicio Web EnteroEnorme.

Nota para los usuarios de Windows XP Service Pack 2

Por cuestiones de seguridad, las computadoras que ejecutan Windows XP Service Pack 2 no permiten peticiones HTTP de otras computadoras de manera predeterminada. Si desea que otras computadoras se conecten a su computadora mediante HTTP, realice los siguientes pasos:

1. Seleccione **Inicio > Panel de control** para abrir la ventana **Panel de control** de su sistema, y después haga doble clic en **Firewall de Windows** para ver el cuadro de diálogo de configuración del **Firewall de Windows**.
2. En este cuadro de diálogo de configuración, haga clic en la ficha **Opciones avanzadas**, seleccione **Conexión de área local** (o el nombre de su conexión de red, si es distinto) en el cuadro de lista **Configuración de conexión de red**, y haga clic en el botón **Configuración...** para que aparezca el cuadro de diálogo **Configuración avanzada**.
3. En el cuadro de diálogo **Configuración avanzada**, asegúrese de que la casilla de verificación para **Servidor Web (HTTP)** esté seleccionada, para permitir que los clientes en otras computadoras puedan enviar peticiones al servidor Web de su computadora.
4. Haga clic en **Aceptar** en el cuadro de diálogo **Configuración avanzada** y después en **Aceptar** en el cuadro de diálogo de configuración de **Firewall de Windows**.

28.3.5 Descripción de un servicio Web con el Lenguaje de descripción de servicios Web (WSDL)

Una vez que se implementa un servicio Web, se compila y se despliega en un servidor de aplicaciones, una aplicación cliente puede consumirlo. Sin embargo, para ello el cliente debe saber en dónde encontrar el servicio Web, y se le debe proporcionar una descripción de cómo interactuar con el servicio Web; es decir, qué métodos están disponibles, qué parámetros esperan y qué devuelve cada método. Para este fin, JAX-WS utiliza el **Lenguaje de descripción de servicios Web (WSDL)**: un vocabulario XML estándar para describir los servicios Web en forma independiente de la plataforma.

No es necesario comprender los detalles de WSDL para aprovechar sus beneficios; el servidor genera un WSDL del servicio Web en forma dinámica por usted, y las herramientas cliente pueden analizar el WSDL para ayudar a crear la clase proxy del lado cliente que un cliente utiliza para acceder al servicio Web. Como el WSDL se crea en forma dinámica, los clientes siempre reciben la descripción más actualizada de un servicio Web desplegado. Para ver el WSDL del servicio Web **EnteroEnorme** (figura 28.6), escriba el siguiente URL en su navegador:

```
http://localhost:8080/EnteroEnorme/ServicioEnteroEnorme?WSDL
```

o haga clic en el vínculo **WSDL File** en la página Web Tester (que se muestra en la figura 28.4).

Cómo acceder al WSDL del servicio Web EnteroEnorme desde otra computadora

En algún momento dado, los clientes en otras computadoras querrán utilizar su servicio Web. Dichos clientes necesitan acceso al WSDL de su servicio Web, al cual pueden acceder mediante el siguiente URL:

```
http://host:8080/EnteroEnorme/ServicioEnteroEnorme?WSDL
```

en donde *host* es el nombre de host o dirección IP de la computadora en la que se va a desplegar el servicio Web. Como vimos en la sección 28.3.4, esto funcionará sólo si su computadora permite conexiones HTTP desde otras computadoras, como se da el caso para los servidores Web y de aplicaciones que están disponibles públicamente.

28.4 Cómo consumir un servicio Web

Ahora que hemos definido y desplegado nuestro servicio Web, podemos consumirlo desde una aplicación cliente. El cliente de un servicio Web puede ser cualquier tipo de aplicación, o incluso otro servicio Web. Para habilitar una aplicación cliente, de manera que pueda consumir un servicio Web, hay que **agregar una referencia del servicio Web** a la aplicación. Este proceso define la clase proxy que permite al cliente acceder al servicio Web.

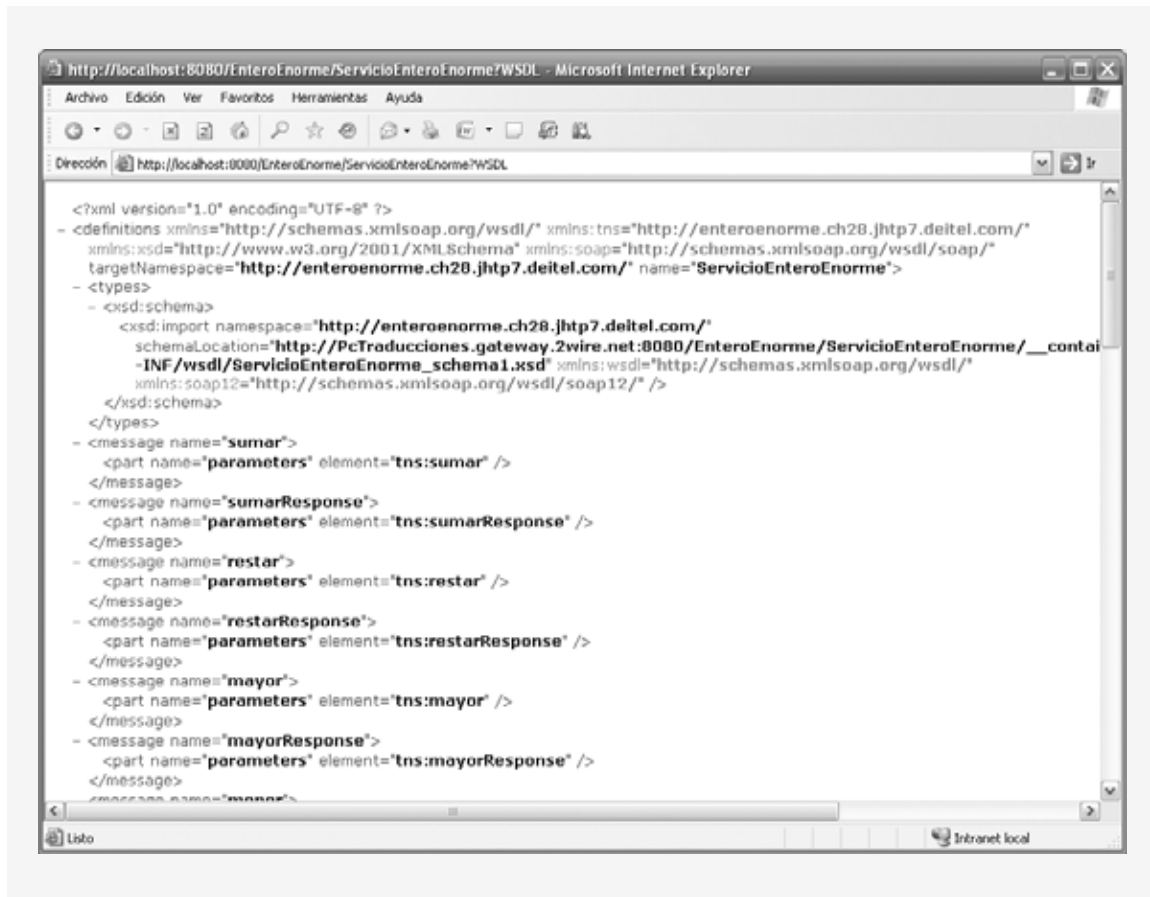


Figura 28.6 | Un fragmento del archivo .wsdl para el servicio Web EnteroEnorme.

28.4.1 Creación de un cliente para consumir el servicio Web EnteroEnorme

En esta sección, utilizará Netbeans para crear una aplicación Java de GUI de escritorio, y después agregará una referencia de servicio Web al proyecto para que el cliente pueda acceder al servicio Web. Al agregar la referencia de servicio Web, el IDE crea y compila los **artefactos del lado cliente**: el marco de trabajo de código de Java que da soporte a la clase proxy del lado cliente. Después, el cliente llama a los métodos en un objeto de la clase proxy, la cual utiliza el resto de los artefactos para interactuar con el servicio Web.

Creación de un proyecto de aplicación de escritorio en Netbeans 5.5

Antes de realizar los pasos en esta sección, asegúrese de que el servicio Web EnteroEnorme se haya desplegado, y de que Sun Java System Application Server se esté ejecutando (vea la sección 28.3.3). Realice los siguientes pasos para crear una aplicación de escritorio Java cliente en Netbeans:

1. Seleccione **File > New Project...** para abrir el cuadro de diálogo **New Project**.
2. Seleccione **General** de la lista **Categories** y **Java Application** de la lista **Projects**.
3. Especifique el nombre **UsoEnteroEnorme** en el campo **Project Name** y desactive la casilla de verificación **Create Main Class**. En un momento agregará una subclase de **JFrame** que contiene un método **main**.
4. Haga clic en **Finish** para crear el proyecto.

Cómo agregar una referencia de servicio Web a una aplicación

A continuación, agregará una referencia de servicio Web a su aplicación para que pueda interactuar con el servicio Web EnteroEnorme. Para agregar una referencia de servicio Web, realice los siguientes pasos.

1. Haga clic con el botón derecho del ratón en el nombre del proyecto (UsoEnteroEnorme) en la ficha **Projects** de Netbeans.
2. Seleccione **New > Web Service Client...** del menú contextual para mostrar el cuadro de diálogo **New Web Service Client** (figura 28.7).
3. En el campo **WSDL URL**, especifique el URL `http://localhost:8080/EnteroEnorme/ServicioEnteroEnorme?WSDL` (figura 28.7). Este URL indica al IDE en dónde puede encontrar la descripción WSDL del servicio Web. [Nota: si Sun Java System Application Server está ubicado en otro equipo, sustituya `localhost` con el nombre de host o dirección IP de esa computadora]. El IDE utiliza esta descripción WSDL para generar los artefactos del lado cliente que componen y dan soporte al proxy. Observe que el cuadro de diálogo **New Web Service Client** le permite buscar servicios Web en varias ubicaciones. Muchas compañías simplemente distribuyen los URL de WSDL exactos para sus servicios Web, que el programador puede colocar en el campo **WSDL URL**.

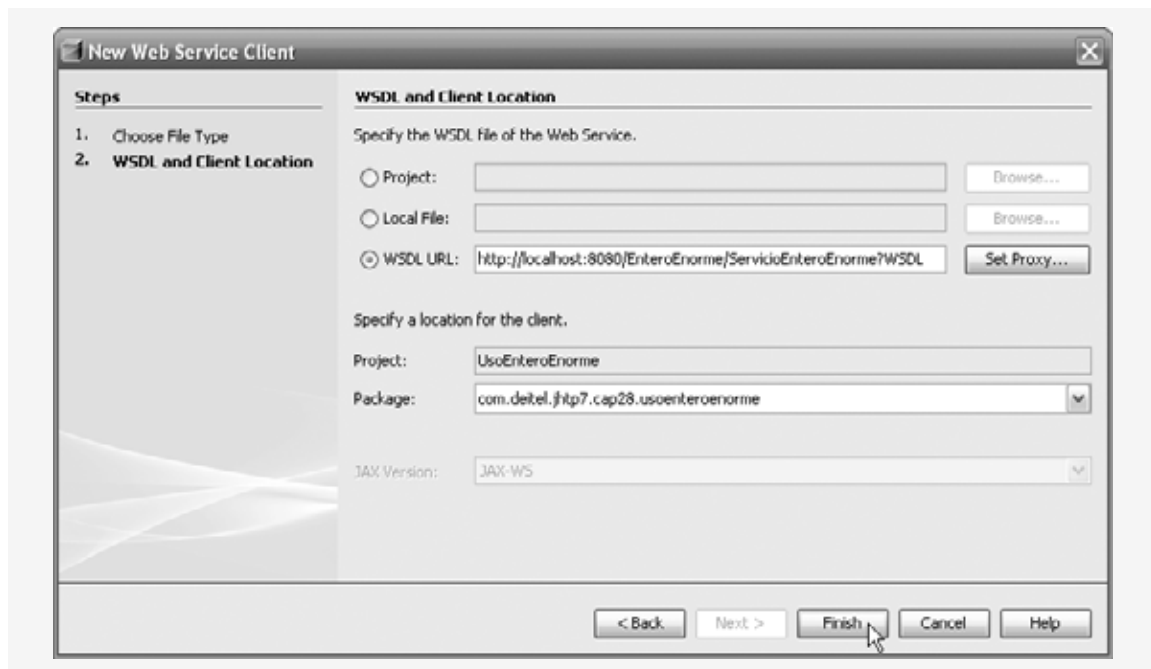


Figura 28.7 | Cuadro de diálogo **New Web Service Client**.

4. En el campo **Package**, especifique `com.deitel.jhtp7.cap28.usoenteroenorme` como el nombre del paquete.
5. Haga clic en **Finish** para cerrar el cuadro de diálogo **New Web Service Client**.

En la ficha **Projects** de Netbeans, el proyecto `UsoEnteroEnorme` ahora contiene una carpeta **Web Service References** con el proxy para el servicio Web EnteroEnorme (figura 28.8). Observe que el nombre del proxy se lista como `ServicioEnteroEnorme`, como especificamos en la línea 11 de la figura 28.2.

Al especificar el servicio Web que deseamos consumir, Netbeans accede a la información WSDL del servicio Web y la copia en un archivo en nuestro proyecto (llamado `ServicioEnteroEnorme.wsdl` en este ejemplo). Para

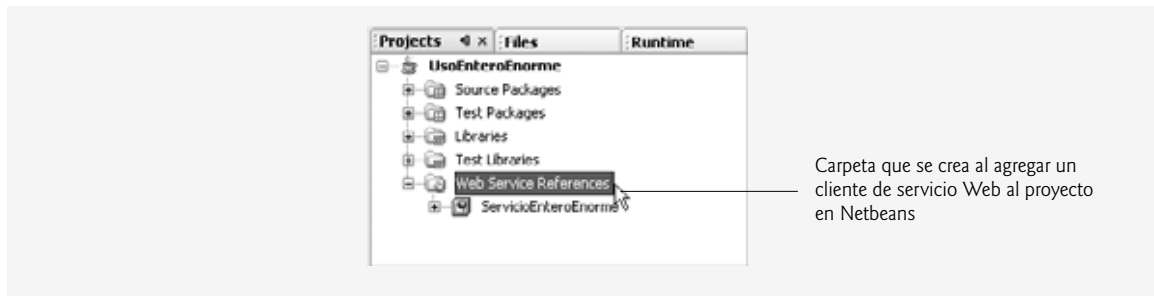


Figura 28.8 | Ficha **Project** de Netbeans después de agregar una referencia de servicio Web al proyecto.

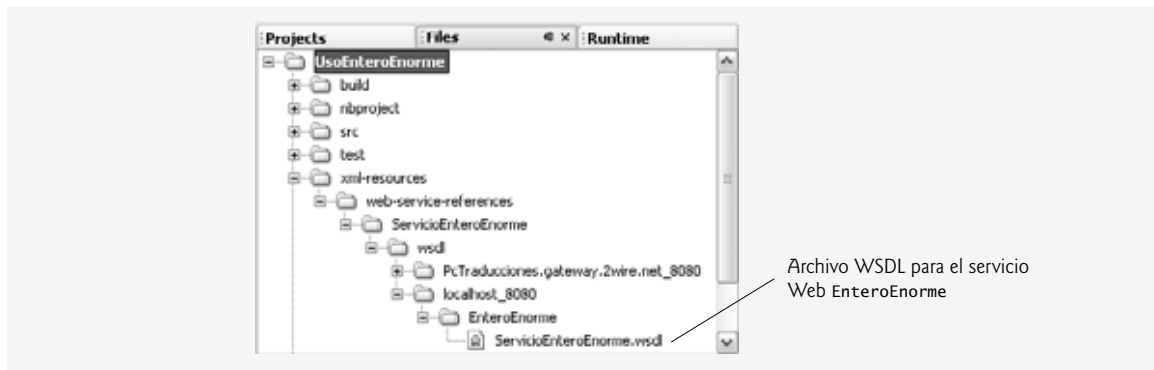


Figura 28.9 | Ubicación del archivo `ServicioEnteroEnorme.wsdl` en la ficha **Files** de Netbeans.

ver este archivo desde la ficha **Files** de Netbeans, expanda los nodos en la carpeta `xml-resources` del proyecto `UsoEnteroEnorme` como se muestra en la figura 28.9. Si el servicio Web cambia, los artefactos del lado cliente y la copia del archivo WSDL del cliente se pueden regenerar haciendo clic con el botón derecho del ratón en el nodo `ServicioEnteroEnorme` que se muestra en la figura 28.8, y seleccionando **Refresh Client**.

Para ver los artefactos del lado cliente generados por el IDE seleccione la ficha **Files** de Netbeans y expanda la carpeta `build` del proyecto `UsoEnteroEnorme`, como se muestra en la figura 28.10.

28.4.2 Cómo consumir el servicio Web EnteroEnorme

Para este ejemplo, utilizaremos una aplicación de GUI para interactuar con el servicio Web. Para crear la GUI de la aplicación cliente, primero hay que agregar una subclase de `JFrame` al proyecto. Para ello, realice los siguientes pasos:

1. Haga clic con el botón derecho del ratón en el nombre del proyecto en la ficha **Project** de Netbeans.
2. Seleccione **New > JFrame Form...** para que aparezca el cuadro de diálogo **New JFrame Form**.
3. Especifique `UsoEnteroEnormeJFrame` en el campo **Class Name**.
4. Especifique `com.deitel.jhtp7.cap28.clienteenteroenorme` en el campo **Package**.
5. Haga clic en **Finish** para cerrar el cuadro de diálogo **New JFrame Form**.

A continuación, cree la GUI que se muestra en las capturas de pantalla de ejemplo al final de la figura 28.11. Para obtener más información acerca de cómo usar Netbeans en la creación de una GUI y de manejadores de eventos, consulte el apéndice de `GroupLayout`.

La aplicación en la figura 28.11 utiliza el servicio Web `EnteroEnorme` para realizar cálculos con enteros positivos de hasta 100 dígitos. Para ahorrar espacio, no mostramos el método `initComponents` generado en forma automática por Netbeans, el cual contiene el código que crea los componentes de GUI, los posiciona y registra sus

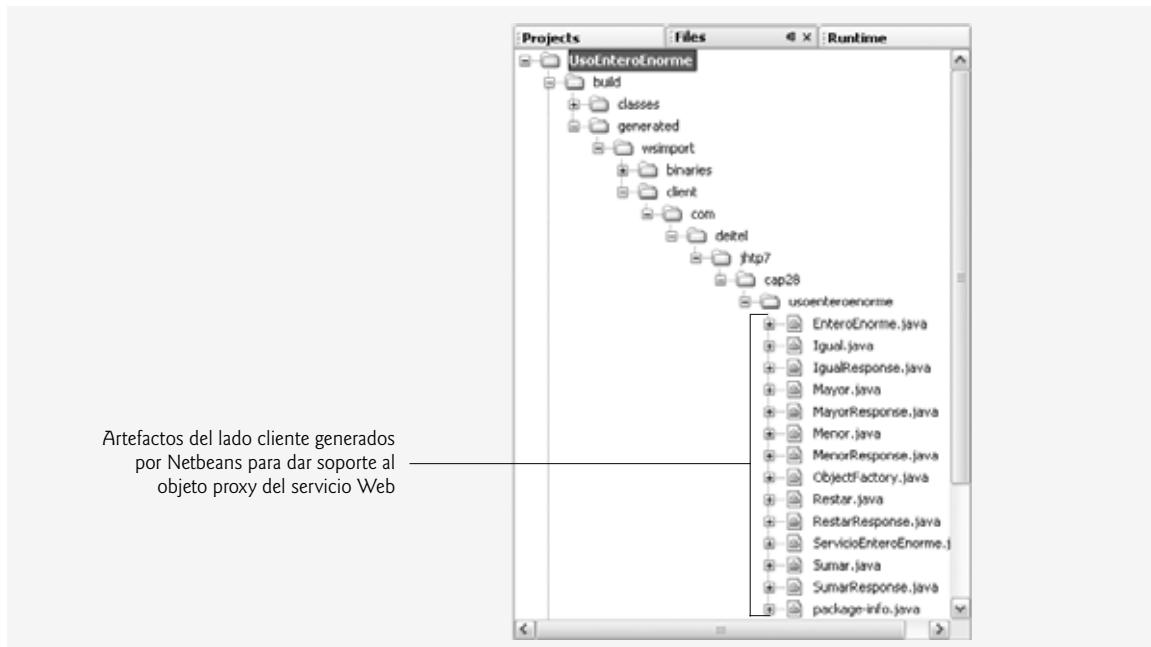


Figura 28.10 | Cómo ver los artefactos del lado cliente del servicio Web EnteroEnorme generados por Netbeans.

manejadores de eventos. Para ver el código fuente completo, abra el archivo `UsoEnteroEnormeJFrame.java` que se encuentra en la carpeta de este ejemplo, en `src\java\com\deitel\jhttp7\cap28\clienteenteroenorme`. Observe que Netbeans coloca las declaraciones de variables de instancia de los componentes de GUI al final de la clase (líneas 326 a 335). Java permite declarar variables de instancia en cualquier parte del cuerpo de una clase, siempre y cuando se coloquen fuera de los métodos de la clase. Seguiremos declarando nuestras propias variables de instancia en la parte superior de la clase.

```

1 // Fig. 28.11: UsoEnteroEnormeJFrame.java
2 // Aplicación de escritorio cliente para el servicio Web EnteroEnorme.
3 package com.deitel.jhttp7.cap28.clienteenteroenorme;
4
5 // importa las clases para acceder al proxy del servicio Web EnteroEnorme
6 import com.deitel.jhttp7.cap28.clienteenteroenorme.EnteroEnorme;
7 import com.deitel.jhttp7.cap28.clienteenteroenorme.ServicioEnteroEnorme;
8
9 import javax.swing.JOptionPane; // se utiliza para mostrar errores al usuario
10
11 public class UsoEnteroEnormeJFrame extends javax.swing.JFrame
12 {
13     private ServicioEnteroEnorme servicioEnteroEnorme; // se usa para obtener el proxy
14     private EnteroEnorme proxyEnteroEnorme; // se usa para acceder al servicio Web
15
16     // constructor sin argumentos
17     public UsoEnteroEnormeJFrame()
18     {
19         initComponents();
20     }

```

Figura 28.11 | Aplicación de escritorio cliente para el servicio Web EnteroEnorme. (Parte 1 de 6).


```

21     try
22     {
23         // crea los objetos para acceder al servicio Web EnteroEnorme
24         servicioEnteroEnorme = new ServicioEnteroEnorme();
25         proxyEnteroEnorme = servicioEnteroEnorme.getEnteroEnormePort();
26     }
27     catch ( Exception excepcion )
28     {
29         excepcion.printStackTrace();
30     }
31 } // fin del constructor de UsoEnteroEnormeJFrame
32
33 // El método initComponents se genera automáticamente por Netbeans y se llama
34 // desde el constructor para inicializar la GUI. No mostraremos aquí este método
35 // para ahorrar espacio. Abra UsoEnteroEnormeJFrame.java en la carpeta
36 // de este ejemplo para ver el código generado completo (líneas 37 a 153).
37
154 // invoca al método sumar del servicio Web EnteroEnorme para sumar objetos EnteroEnorme
155 private void sumarJButtonActionPerformed(
156     java.awt.event.ActionEvent evt )
157 {
158     String primerNumero = primeroJTextField.getText();
159     String segundoNumero = segundoJTextField.getText();
160
161     if ( esValido( primerNumero ) && esValido( segundoNumero ) )
162     {
163         try
164         {
165             resultadosJTextArea.setText(
166                 proxyEnteroEnorme.sumar( primerNumero, segundoNumero ) );
167         } // fin de try
168         catch ( Exception e )
169         {
170             JOptionPane.showMessageDialog( this, e.toString(),
171                 "Error en el metodo Sumar", JOptionPane.ERROR_MESSAGE );
172             e.printStackTrace();
173         } // fin de catch
174     } // fin de if
175 } // fin del método sumarJButtonActionPerformed
176
177 // invoca al método restar del servicio Web EnteroEnorme para restar el
178 // segundo EnteroEnorme del primero
179 private void restarJButtonActionPerformed(
180     java.awt.event.ActionEvent evt )
181 {
182     String primerNumero = primeroJTextField.getText();
183     String segundoNumero = segundoJTextField.getText();
184
185     if ( esValido( primerNumero ) && esValido( segundoNumero ) )
186     {
187         try
188         {
189             resultadosJTextArea.setText(
190                 proxyEnteroEnorme.restar( primerNumero, segundoNumero ) );
191         } // fin de try
192         catch ( Exception e )
193         {
194             JOptionPane.showMessageDialog( this, e.toString(),

```

Figura 28.11 | Aplicación de escritorio cliente para el servicio Web EnteroEnorme. (Parte 2 de 6).

```

195         "Error en el metodo Restar", JOptionPane.ERROR_MESSAGE );
196         e.printStackTrace();
197     } // fin de catch
198 } // fin de if
199 } // fin del método restarJButtonActionPerformed
200
201 // invoca al método mayor del servicio Web EnteroEnorme para determinar si
202 // el primer EnteroEnorme es mayor que el segundo
203 private void mayorJButtonActionPerformed(
204     java.awt.event.ActionEvent evt )
205 {
206     String primerNumero = primeroJTextField.getText();
207     String segundoNumero = segundoJTextField.getText();
208
209     if ( esValido( primerNumero ) && esValido( segundoNumero ) )
210     {
211         try
212         {
213             boolean result =
214                 proxyEnteroEnorme.mayor( primerNumero, segundoNumero );
215             resultadosJTextArea.setText( String.format( "%s %s %s %s",
216                 primerNumero, ( result ? "es" : "no es" ), "mayor que",
217                 segundoNumero ) );
218         } // fin de try
219         catch ( Exception e )
220         {
221             JOptionPane.showMessageDialog( this, e.toString(),
222                 "Error en metodo Mayor", JOptionPane.ERROR_MESSAGE );
223             e.printStackTrace();
224         } // fin de catch
225     } // fin de if
226 } // fin del método mayorJButtonActionPerformed
227
228 // invoca al método menor del servicio Web EnteroEnorme para determinar
229 // si el primer EnteroEnorme es menor que el segundo
230 private void menorJButtonActionPerformed(
231     java.awt.event.ActionEvent evt )
232 {
233     String primerNumero = primeroJTextField.getText();
234     String segundoNumero = segundoJTextField.getText();
235
236     if ( esValido( primerNumero ) && esValido( segundoNumero ) )
237     {
238         try
239         {
240             boolean resultado =
241                 proxyEnteroEnorme.menor( primerNumero, segundoNumero );
242             resultadosJTextArea.setText( String.format( "%s %s %s %s",
243                 primerNumero, ( resultado ? "es" : "no es" ), "menor que",
244                 segundoNumero ) );
245         } // fin de try
246         catch ( Exception e )
247         {
248             JOptionPane.showMessageDialog( this, e.toString(),
249                 "Error en metodo Menor", JOptionPane.ERROR_MESSAGE );
250             e.printStackTrace();
251         } // fin de catch
252     } // fin de if

```

Figura 28.11 | Aplicación de escritorio cliente para el servicio Web EnteroEnorme. (Parte 3 de 6).

```

253 } // fin del método menorJButtonActionPerformed
254
255 // invoca al método igual del servicio Web EnteroEnorme para determinar si
256 // el primer EnteroEnorme es igual al segundo
257 private void igualJButtonActionPerformed(
258     java.awt.event.ActionEvent evt )
259 {
260     String primerNumero = primeroJTextField.getText();
261     String segundoNumero = segundoJTextField.getText();
262
263     if ( esValido( primerNumero ) && esValido( segundoNumero ) )
264     {
265         try
266         {
267             boolean resultado =
268                 proxyEnteroEnorme.igual( primerNumero, segundoNumero );
269             resultadosJTextArea.setText( String.format( "%s %s %s %s",
270                 primerNumero, ( resultado ? "es" : "no es" ), "igual a",
271                 segundoNumero ) );
272         } // fin de try
273         catch ( Exception e )
274         {
275             JOptionPane.showMessageDialog( this, e.toString(),
276                 "Error en el metodo Igual", JOptionPane.ERROR_MESSAGE );
277             e.printStackTrace();
278         } // fin de catch
279     } // fin de if
280 } // fin del método igualJButtonActionPerformed
281
282 // comprueba el tamaño de un objeto String para asegurar que no sea demasiado grande
283 // como para usarlo como un EnteroEnorme; asegura que sólo haya dígitos en el String
284 private boolean esValido( String numero )
285 {
286     // comprueba la longitud del objeto String
287     if ( numero.length() > 100 )
288     {
289         JOptionPane.showMessageDialog( this,
290             "Los EnterosEnormes deben ser <= 100 digitos.", "Desbordamiento de
291             EnteroEnorme",
292             JOptionPane.ERROR_MESSAGE );
293         return false;
294     } // fin de if
295
296     // busca caracteres que no sean dígitos en el objeto String
297     for ( char c : numero.toCharArray() )
298     {
299         if ( !Character.isDigit( c ) )
300         {
301             JOptionPane.showMessageDialog( this,
302                 "Hay caracteres que no son digitos en el objeto String",
303                 "EnteroEnorme contiene caracteres que no son digitos",
304                 JOptionPane.ERROR_MESSAGE );
305             return false;
306         } // fin de if
307     } // fin de for
308
309     return true; // el número se puede usar como un EnteroEnorme
310 } // fin del método de validación

```

Figura 28.11 | Aplicación de escritorio cliente para el servicio Web EnteroEnorme. (Parte 4 de 6).

```

310
311 // el método main empieza la ejecución
312 public static void main( String args[] )
313 {
314     java.awt.EventQueue.invokeLater(
315         new Runnable()
316         {
317             public void run()
318             {
319                 new UsoEnteroEnormeJFrame().setVisible( true );
320             } // fin del método run
321         } // fin de la clase interna anónima
322     ); // fin de la llamada a java.awt.EventQueue.invokeLater
323 } // fin del método main
324
325 // Variables declaration - do not modify
326 private javax.swing.JButton igualJButton;
327 private javax.swing.JLabel indicacionesJLabel;
328 private javax.swing.JButton mayorJButton;
329 private javax.swing.JButton menorJButton;
330 private javax.swing.JTextField primeroJTextField;
331 private javax.swing.JButton restarJButton;
332 private javax.swing.JScrollPane resultadosJScrollPane;
333 private javax.swing.JTextArea resultadosJTextArea;
334 private javax.swing.JTextField segundoJTextField;
335 private javax.swing.JButton sumarJButton;
336 // fin de las variables declaration
337 } // fin de la clase UsoEnteroEnormeJFrame

```

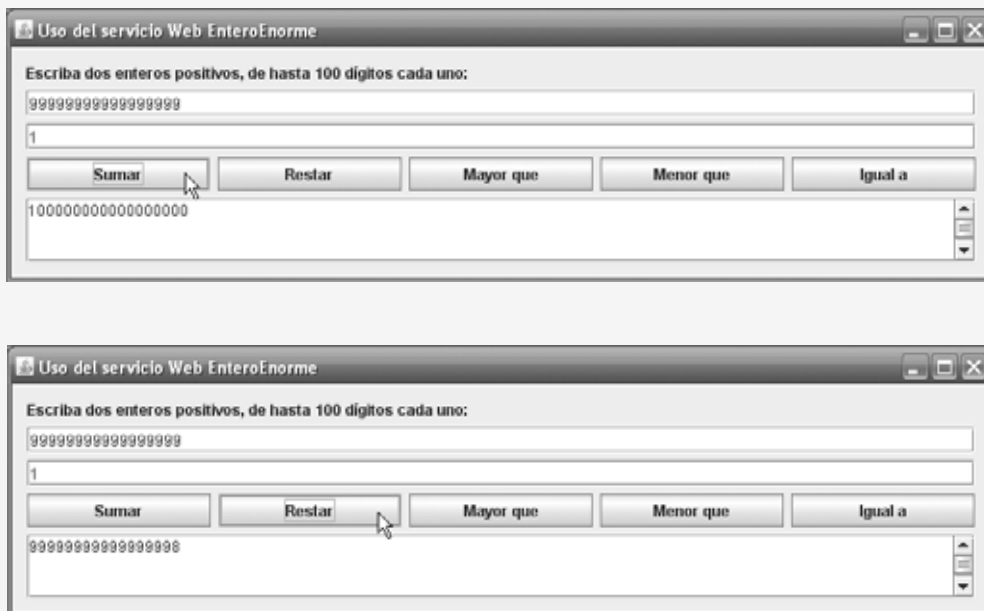


Figura 28.11 | Aplicación de escritorio cliente para el servicio Web EnteroEnorme. (Parte 5 de 6).

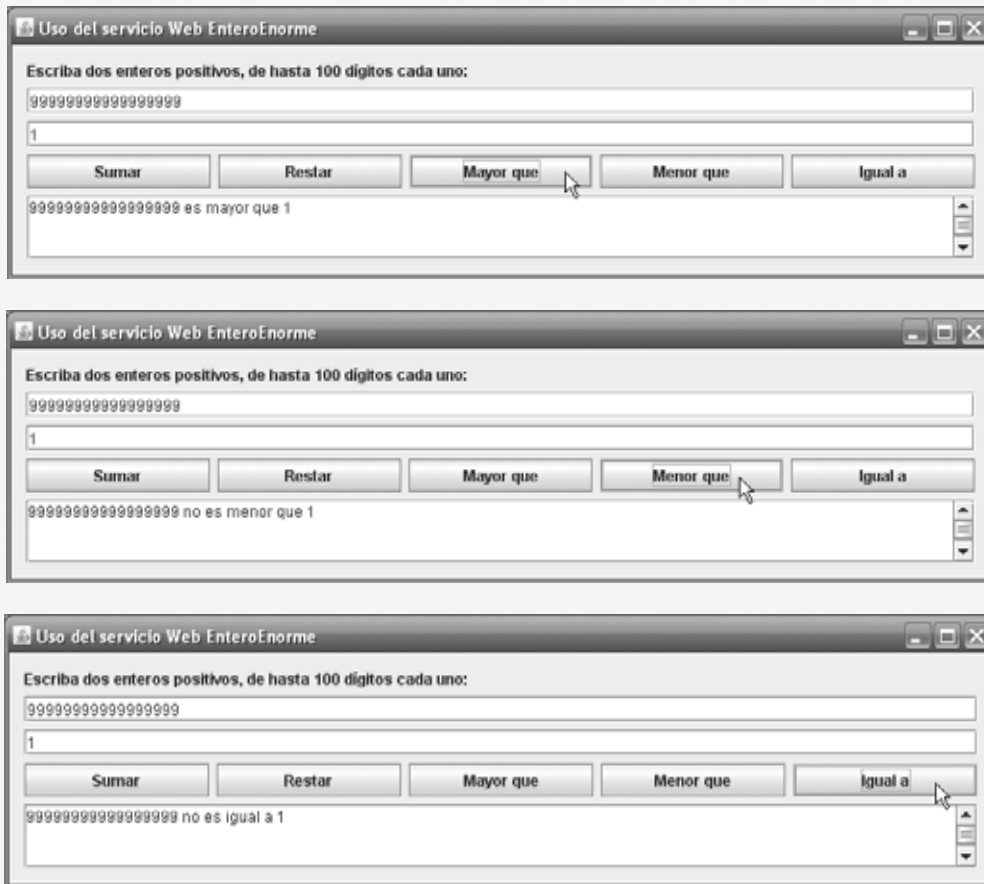


Figura 28.11 | Aplicación de escritorio cliente para el servicio Web EnteroEnorme. (Parte 6 de 6).

En las líneas 6 a 7 se importan las clases `EnteroEnorme` y `ServicioEnteroEnorme` que permiten a la aplicación cliente interactuar con el servicio Web. Aquí incluimos esas declaraciones `import` sólo para fines de documentación. Estas clases se encuentran en el mismo paquete que `UsoEnteroEnormeJFrame`, por lo que no son necesarias estas declaraciones `import`. Observe que no tenemos declaraciones `import` para la mayoría de los componentes de GUI que utilizamos en este ejemplo. Al crear una GUI en Netbeans, utiliza los nombres de clases completamente calificados (como `javax.swing.JFrame` en la línea 11), por lo que estas declaraciones no son necesarias.

En las líneas 13 y 14 se declaran las variables de tipo `ServicioEnteroEnorme` y `EnteroEnorme`, respectivamente. En la línea 24 en el constructor se crea un objeto `ServicioEnteroEnorme`. En la línea 25 se utiliza el método `getEnteroEnormePort` de este objeto para obtener el objeto `proxyEnteroEnorme` que utiliza la aplicación para invocar al método del servicio Web.

En las líneas 165 a 166, 189 a 190, 213 a 214, 240 a 241 y 267 a 268 en los diversos manejadores de `JButton`, se invocan los métodos Web del servicio `EnteroEnorme`. Observe que cada llamada se realiza en el objeto proxy local al que se hace referencia mediante `proxyEnteroEnorme`. Después, el objeto proxy se comunica con el servicio Web por el cliente.

El usuario introduce dos enteros, cada uno de hasta 100 dígitos máximo. Al hacer clic en cualquiera de los cinco objetos `JButton`, la aplicación invoca a un método Web para realizar la tarea correspondiente y devolver el resultado. Nuestra aplicación cliente no puede procesar números de 100 dígitos directamente. En vez de ello, el cliente pasa representaciones `String` de esos números a los métodos Web del servicio Web, los cuales realizan

tareas para el cliente. Así, la aplicación cliente utiliza entonces el valor de retorno de cada operación para mostrar un mensaje apropiado.

28.5 SOAP

SOAP (un acrónimo para Protocolo simple de acceso a objetos) es un protocolo independiente de la plataforma que usa XML para facilitar las llamadas a procedimientos remotos, por lo general, a través de HTTP. SOAP es un protocolo común para pasar información entre los clientes de servicios Web y los servicios Web. El protocolo que transmite mensajes de petición y respuesta se conoce también como el **formato de cable** o **protocolo de cable** del servicio Web, ya que define cómo se envía la información “a través del cable”.

Cada petición y respuesta se empaqueta en un **mensaje SOAP** (también conocido como **envoltura SOAP**): una “envoltura” XML que contiene la información que un servicio Web requiere para procesar el mensaje. Con unas cuantas excepciones, la mayoría de los **firewalls** (barreras de seguridad que restringen la comunicación entre redes) permiten que pase el tráfico http para que los clientes puedan navegar en sitios Web ubicados en servidores Web detrás de los firewalls. Por ende, XML y HTTP permiten que computadoras en distintas plataformas envíen y reciban mensajes SOAP, con unas cuantas limitaciones.

Los servicios Web también utilizan SOAP para el extenso conjunto de tipos que soporta. El formato de cable utilizado para transmitir peticiones y respuestas debe soportar todos los tipos que se pasan entre las aplicaciones. SOAP soporta tipos primitivos (como `int`) y sus tipos de envoltura (como `Integer`), así como `Date`, `Time` y otros. SOAP también puede transmitir arreglos y objetos de tipos definidos por el usuario (como veremos en la sección 28.8). Para obtener más información acerca de SOAP, visite www.w3.org/TR/soap/.

Cuando un programa invoca a un método Web, la petición y toda la información relevante se empaqueta en un mensaje SOAP, y se envía al servidor en el que reside el servicio Web. Este servicio procesa el contenido del mensaje SOAP (que se encuentra dentro de una envoltura SOAP), el cual especifica el método que el cliente desea invocar y los argumentos para este método. A este proceso de interpretar el contenido de un mensaje SOAP se le conoce como **análisis de un mensaje SOAP**. Una vez que el servicio Web recibe y analiza una petición, se hace una llamada al método apropiado con cualquier argumento especificado, y la respuesta se envía de vuelta al cliente en otro mensaje SOAP. El proxy del lado cliente analiza la respuesta, que contiene el resultado de la llamada al método, y devuelve el resultado a la aplicación cliente.

En la figura 28.5 se utilizó la página Web Tester del servicio Web EnteroEnorme para mostrar el resultado de invocar al método `sumar` de `EnteroEnorme` con los valores `9999999999999999` y `1`. La página Web Tester también muestra la petición SOAP y los mensajes de respuesta (que no se mostraron antes). En la figura 28.12 se muestra el mismo resultado con los mensajes SOAP que muestra la aplicación Tester. En el mensaje de petición de la figura 28.12, el texto

```
<ns1:sumar>
<primero>9999999999999999</primero>
<segundo>1</segundo>
</ns1:sumar>
```

especifica el método a llamar (`sumar`), los argumentos del método (`primero` y `segundo`), y los valores de los argumentos (`9999999999999999` y `1`). De manera similar, el texto

```
<ns1:sumarResponse>
<return>10000000000000000</return>
</ns1:sumarResponse>
```

del mensaje de respuesta en la figura 28.12 especifica el valor de retorno del método `sumar`.

Al igual que con el WSDL para un servicio Web, los mensajes SOAP se generan de manera automática para usted. Por lo tanto, no necesita comprender los detalles acerca de SOAP o XML para aprovecharlos a la hora de publicar y consumir los servicios Web.

28.6 Rastreo de sesiones en los servicios Web

En la sección 26.7 describimos las ventajas en cuanto al uso del rastreo de sesiones para mantener la información de estado de un cliente, de manera que podamos personalizar las experiencias de navegación del usuario. Ahora vamos a incorporar el rastreo de sesiones en un servicio Web. Suponga que una aplicación cliente necesita llamar a

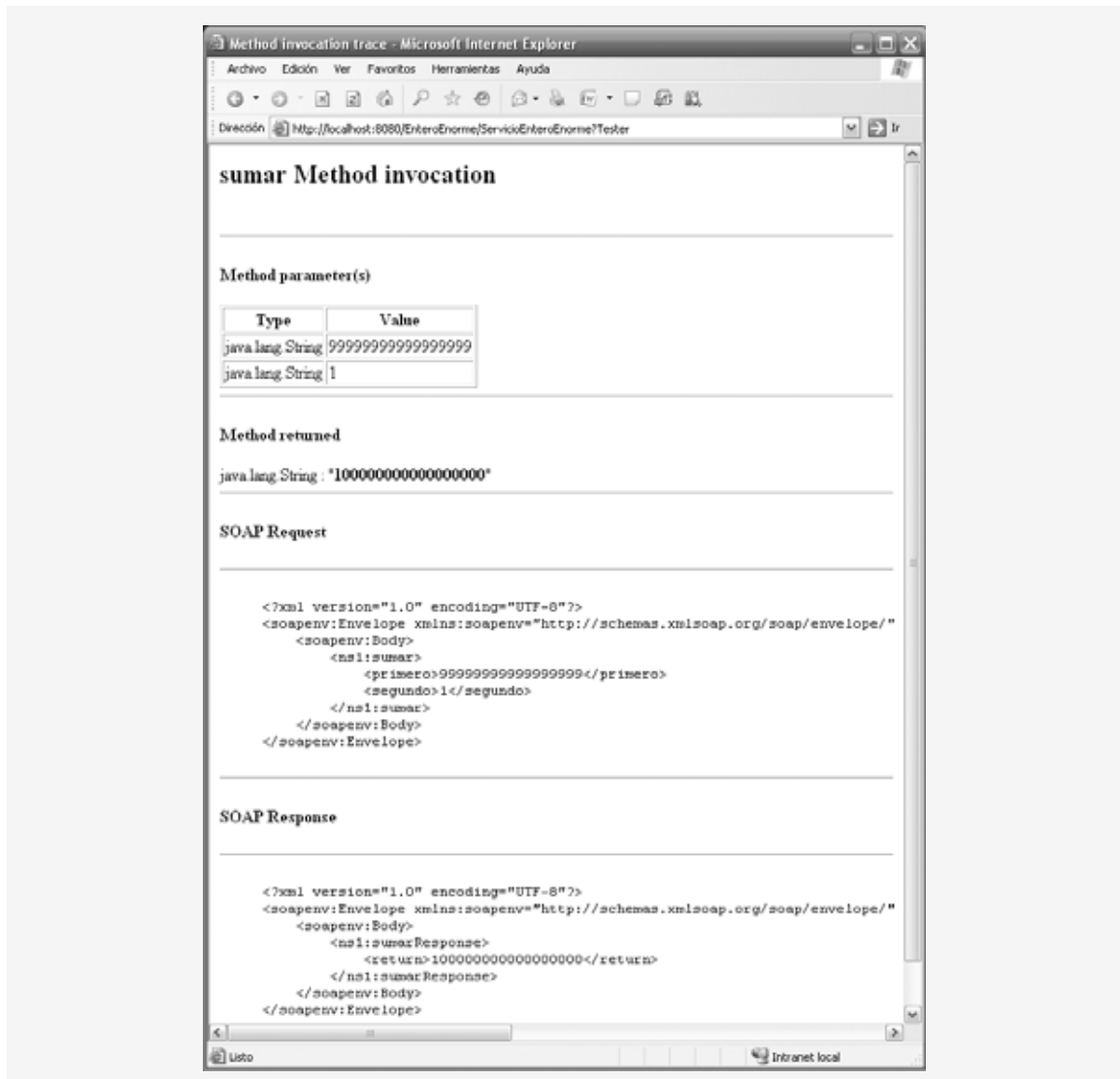


Figura 28.12 | Mensajes SOAP para el método sumar del servicio Web EnteroEnorme, como se muestra en la página Web Tester de Sun Java System Application Server.

varios métodos del mismo servicio Web, tal vez varias veces cada uno de ellos. En tal caso, puede ser benéfico para el servicio Web mantener la información de estado para el cliente, con lo cual se elimina la necesidad de pasar la información del cliente entre éste y el servicio Web varias veces. Por ejemplo, un servicio Web que proporciona reseñas de restaurantes locales podría almacenar el domicilio del usuario cliente durante la petición inicial, y después utilizarlo para devolver resultados personalizados y localizados en las peticiones subsiguientes. Al almacenar la información de la sesión, también permitimos que un servicio Web diferencie a un cliente de otro.

28.6.1 Creación de un servicio Web Blackjack

Nuestro siguiente ejemplo es un servicio Web que le ayudará a desarrollar un juego de cartas llamado Blackjack. El servicio Web Blackjack (figura 28.13) proporciona métodos Web para barajar un mazo de cartas, repartir una carta del mazo y evaluar una mano de cartas. Después de presentar el servicio Web, lo utilizaremos para que sirva como el repartidor en un juego de blackjack (figura 28.14). El servicio Web Blackjack usa un objeto

HttpSession para mantener un mazo único de cartas para cada aplicación cliente. Varios clientes pueden usar el servicio al mismo tiempo, pero las llamadas a los métodos Web que realiza un cliente específico sólo utilizan el mazo de cartas almacenadas en la sesión de ese cliente. Nuestro ejemplo utiliza las siguientes reglas de blackjack:

Se reparten dos cartas al repartidor y dos cartas al jugador. Las cartas del jugador se reparten con la cara hacia arriba. Sólo la primera de las cartas del repartidor se reparte con la cara hacia arriba. Cada carta tiene un valor. Una carta numerada del 2 al 10 vale lo que indique en su cara. Los jotos, reinas y reyes valen 10 cada uno. Los ases pueden valer 1 u 11, lo que sea más conveniente para el jugador (como pronto veremos). Si la suma de las dos cartas iniciales del jugador es 21 (por ejemplo, que se hayan repartido al jugador una carta con valor de 10 y un as, que cuenta como 11 en esta ocasión), el jugador tiene "blackjack" y gana el juego de inmediato (si el repartidor tampoco tienen blackjack, lo que resulta en un "empate"). En caso contrario, el jugador puede empezar a tomar cartas adicionales, una a la vez. Estas cartas se reparten con la cara hacia arriba, y el jugador decide cuándo dejar de tomar cartas. Si el jugador "se pasa" (es decir, si la suma de las cartas del jugador excede a 21), el juego termina y el jugador pierde. Cuando el jugador está satisfecho con su conjunto actual de cartas, "se planta" (es decir, deja de tomar cartas) y se revela la carta oculta del repartidor. Si el total del repartidor es 16 o menos, debe tomar otra carta; en caso contrario, el repartidor debe plantarse. El repartidor debe seguir tomando cartas hasta que la suma de todas sus cartas sea mayor o igual a 17. Si el repartidor se pasa de 21, el jugador gana. En caso contrario, la mano con el total de puntos que sea mayor gana. Si el repartidor y el jugador tienen el mismo total de puntos, el juego es un "empate" y nadie gana. Observe que el valor de un as para un repartidor depende de la(s) otra(s) carta(s) del repartidor y de las reglas de la casa del casino. Por lo general, un repartidor debe obtener totales de 16 o menos y debe "plantarse" al obtener totales de 17 o más. Sin embargo, para un "suave 17" (una mano con un total de 17, en donde un as cuenta como 11), algunos casinos requieren que el repartidor tome otra carta y otros requieren que se plante (nosotros vamos a requerir que el repartidor se plante). A dicha mano se le conoce como "suave 17", ya que al tomar otra carta la mano no puede "pasarse".

El servicio Web (figura 28.13) almacena cada carta como un objeto String que consiste en un número del 1 al 13, para representar la cara de la carta (del as al rey, respectivamente), seguido de un espacio y un dígito del 0 al 3, que representa el palo de la carta (corazones, diamantes, bastos o espadas, respectivamente). Por ejemplo, el joto de bastos se representa como "11 2" y el dos de corazones se representa como "2 0". Para crear y desplegar este servicio Web, siga los pasos que se presentan en las secciones 28.3.3 a 28.3.4 para el servicio Entero-Enorme.

```

1 // Fig. 28.13: Blackjack.java
2 // Servicio Web Blackjack que reparte cartas y evalúa manos
3 package com.deitel.jhttp7.ch28.blackjack;
4
5 import java.util.ArrayList;
6 import java.util.Random;
7 import javax.annotation.Resource;
8 import javax.jws.WebService;
9 import javax.jws.WebMethod;
10 import javax.jws.WebParam;
11 import javax.servlet.http.HttpSession;
12 import javax.servlet.http.HttpServletRequest;
13 import javax.xml.ws.WebServiceContext;
14 import javax.xml.ws.handler.MessageContext;
15
16 @WebService( name = "Blackjack", serviceName = "ServicioBlackjack" )
17 public class Blackjack

```

Figura 28.13 | Servicio Web Blackjack que reparte cartas y evalúa manos. (Parte I de 3).


```

18 {
19     // usa @Resource para crear un objeto WebServiceContext para rastrear sesiones
20     private @Resource WebServiceContext contextoServicioWeb;
21     private MessageContext contextoMensaje; // se utiliza en el rastreo de sesiones
22     private HttpSession sesion; // almacena los atributos de la sesión
23
24     // reparte una carta
25     @WebMethod( operationName = "repartirCarta" )
26     public String repartirCarta()
27     {
28         String carta = "";
29
30         ArrayList< String > mazo =
31             ( ArrayList< String > ) sesion.getAttribute( "mazo" );
32
33         carta = mazo.get( 0 ); // obtiene la carta superior del mazo
34         mazo.remove( 0 ); // elimina la carta superior del mazo
35
36         return carta;
37     } // fin del Método Web repartirCarta
38
39     // baraja el mazo
40     @WebMethod( operationName = "barajar" )
41     public void barajar()
42     {
43         // obtiene el objeto HttpSession para almacenar el mazo para el cliente actual
44         contextoMensaje = contextoServicioWeb.getMessageContext();
45         sesion = ( ( HttpServletRequest ) contextoMensaje.get(
46             MessageContext.SERVLET_REQUEST ) ).getSession();
47
48         // llena el mazo de cartas
49         ArrayList< String > mazo = new ArrayList< String >();
50
51         for ( int cara = 1; cara <= 13; cara++ ) // itera a través de las caras
52             for ( int palo = 0; palo <= 3; palo++ ) // itera a través de los palos
53                 mazo.add( cara + " " + palo ); // agrega cada carta al mazo
54
55         String cartaTemp; // guarda la carta temporalmente durante el intercambio
56         Random objetoAleatorio = new Random(); // genera números aleatorios
57         int indice; // índice la carta seleccionada al azar
58
59         for ( int i = 0; i < mazo.size() ; i++ ) // baraja
60         {
61             indice = objetoAleatorio.nextInt( mazo.size() - 1 );
62
63             // intercambia la carta en la posición i con la carta seleccionada al azar
64             cartaTemp = mazo.get( i );
65             mazo.set( i, mazo.get( indice ) );
66             mazo.set( indice, cartaTemp );
67         } // fin de for
68
69         // agrega este mazo a la sesión del usuario
70         sesion.setAttribute( "mazo", mazo );
71     } // fin del Método Web barajar
72
73     // determina el valor de una mano
74     @WebMethod( operationName = "obtenerValorMano" )
75     public int obtenerValorMano( @WebParam( name = "mano" ) String mano )
76     {

```

Figura 28.13 | Servicio Web Blackjack que reparte cartas y evalúa manos. (Parte 2 de 3).

```

77 // divide la mano en cartas
78 String[] cartas = mano.split( "\\t" );
79 int total = 0; // valor total de las cartas en la mano
80 int cara; // cara de la carta actual
81 int cuentaAses = 0; // número de ases en la mano
82
83 for ( int i = 0; i < cartas.length; i++ )
84 {
85     // analiza la cadena y obtiene el primer int en el objeto String
86     cara = Integer.parseInt(
87         cartas[ i ].substring( 0, cartas[ i ].indexOf( " " ) ) );
88
89     switch ( cara )
90     {
91         case 1: // en as, incrementa cuentaAses
92             ++cuentaAses;
93             break;
94         case 11: // joto
95         case 12: // reina
96         case 13: // rey
97             total += 10;
98             break;
99         default: // en cualquier otro caso, suma la cara
100             total += cara;
101             break;
102     } // fin de switch
103 } // fin de for
104
105 // calcula el uso óptimo de los ases
106 if ( cuentaAses > 0 )
107 {
108     // si es posible, cuenta un as como 11
109     if ( total + 11 + cuentaAses - 1 <= 21 )
110         total += 11 + cuentaAses - 1;
111     else // en cualquier otro caso, cuenta todos los ases como 1
112         total += cuentaAses;
113 } // fin de if
114
115 return total;
116 } // fin del Método Web obtenerValorMano
117 } // fin de la clase Blackjack

```

Figura 28.13 | Servicio Web Blackjack que reparte cartas y evalúa manos. (Parte 3 de 3).

Rastreo de sesiones en servicios Web

El cliente del servicio Web Blackjack llama primero al método `barajar` (líneas 40 a 71) para barajar el mazo de cartas. Este método también coloca el mazo de cartas en un objeto `HttpSession` que es específico para el cliente que llamó a `barajar`. Para usar el rastreo de sesiones en un servicio Web, debemos incluir código para los recursos que mantienen la información de estado de la sesión. Anteriormente, había que escribir el código, que algunas veces era tedioso, para crear estos recursos. Sin embargo, JAX-WS se encarga de esto por nosotros mediante la anotación `@Resource`. Esta anotación permite a herramientas como Netbeans “inyectar” el código complejo de soporte en nuestra clase, con lo cual nos podemos enfocar en la lógica de negocios, en vez de hacerlo en el código de soporte. El concepto de usar anotaciones para agregar código que de soporte a nuestras clases se conoce como **inyección de dependencias**. Las anotaciones como `@WebService`, `@WebMethod` y `@Webparam` también realizan la inyección de dependencias.

En la línea 20 se inyecta un objeto `WebServiceContext` en nuestra clase. Un objeto `WebServiceContext` permite a un servicio Web acceder a la información para una petición específica y darle mantenimiento, como el

estado de la sesión. A medida que analice el código de la figura 28.13 observará que nunca creamos el objeto `Web-ServiceContext`. Todo el código necesario para crearlo se inyecta en la clase mediante la anotación `@Resource`. En la línea 21 se declara una variable del tipo de interfaz `MessageContext`, que el servicio Web utilizará para obtener un objeto `HttpSession` para el cliente actual. En la línea 22 se declara la variable `HttpSession` que el servicio Web utilizará para manipular la información de estado de la sesión.

En la línea 44 del método `barajar` se utiliza el objeto `conextoServicioWeb` que se inyectó en la línea 20 para obtener un objeto `MessageContext`. Después, en las líneas 45 y 46 se utiliza el método `get` del objeto `MessageContext` para obtener el objeto `HttpSession` para el cliente actual. El método `get` recibe una constante que indica lo que se debe obtener del objeto `MessageContext`. En este caso, la constante `MessageContext.SER-VLET_REQUEST` indica que nos gustaría obtener el objeto `HttpServletRequest` para el cliente actual. Después llamamos al método `getSession` para obtener el objeto `HttpSession` del objeto `HttpServletRequest`.

En las líneas 49 a 70 se genera un objeto `ArrayList` que representa a un conjunto de cartas, se baraja el mazo y se almacena en el objeto `sesion` del cliente. En las líneas 51 a 53 se utilizan ciclos anidados para generar objetos `String` en la forma "*cara palo*" para representar a cada una de las posibles cartas en el mazo. En las líneas 59 a 67 se baraja el mazo, intercambiando cada carta con otra seleccionada al azar. En la línea 70 se inserta el objeto `ArrayList` en el objeto `sesion` para mantener el mazo entre las llamadas a los métodos desde un cliente específico.

En las líneas 25 a 37 se define el método `repartirCarta` como un método Web. En las líneas 30 y 31 se utiliza el objeto `sesion` para obtener el atributo de sesión "mazo" que se almacenó en la línea 70 del método `barajar`. El método `getAttribute` recibe como parámetro un objeto `String` que identifica el objeto `Object` a obtener del estado de la sesión. El objeto `HttpSession` puede almacenar muchos `Object`, siempre y cuando cada uno tenga un identificador único. Observe que se debe llamar al método `barajar` antes de llamar al método `repartirCarta` por primera vez para un cliente; en caso contrario, se producirá una excepción en las líneas 30 y 31, ya que `getAttribute` devolverá `null`. Después de obtener el mazo del usuario, `repartirCarta` obtiene la carta superior del mazo (línea 33), la quita del mazo (línea 34) y devuelve el valor de la carta como un objeto `String` (línea 36). Sin usar el rastreo de sesiones, el mazo de cartas tendría que pasarse entre cada una de las llamadas a los métodos. El rastreo de sesiones facilita el proceso de llamar al método `repartirCarta` (no requiere argumentos) y elimina la sobrecarga de enviar el mazo a través de la red varias veces.

El método `obtenerValorMano` (líneas 74 a 116) determina el valor total de las cartas en una mano, al tratar de obtener la mayor puntuación posible sin pasar de 21. Recuerde que un as se puede contar como 1 u 11, y todas las cartas con cara cuentan como 10. Este método no utiliza el objeto `sesion`, ya que el mazo de cartas no se utiliza en este método.

Como veremos pronto, la aplicación cliente mantiene una mano de cartas como un objeto `String`, en el cual cada carta va separada de un carácter de tabulación. En la línea 78 se divide en tokens la mano de cartas (representada por `repartir`) en cartas individuales, mediante una llamada al método `split` de `String`, y se pasa a un objeto `String` que contiene los caracteres delimitadores (en este caso, sólo un tabulador). El método `Split` usa los caracteres delimitadores para separar los tokens en el objeto `String`. En las líneas 83 a 103 se cuenta el valor de cada carta. En las líneas 86 y 87 se obtiene el primer entero (la cara) y se utiliza ese valor en la instrucción `switch` (líneas 89 a 102). Si la carta es un as, el método incrementa la variable `cuentaAses`. En breve hablaremos acerca de cómo se utiliza esta variable. Si la carta es un 11, 12 o 13 (joto, reina o rey), el método suma 10 al valor total de la mano (línea 97). Si la carta es cualquier otra cosa, el método incrementa el total en base a ese valor (línea 100).

Como un as puede tener uno de dos valores, se requiere lógica adicional para procesar los ases. En las líneas 106 a 113 del método `obtenerValorMano` se procesan los ases después de todas las demás cartas. Si una mano contiene varios ases, sólo un as puede contarse como 11. La condición en la línea 109 determina si el contar un as como 11 y el resto como 1 producirá un total que no exceda a 21. Si esto es posible, en la línea 110 se ajusta el total de manera acorde. En caso contrario, en la línea 112 se ajusta el total, y se cuenta cada as como 1.

El método `obtenerValorMano` incrementa al máximo el valor de las cartas actuales sin exceder a 21. Por ejemplo, imagine que el repartidor tiene un 7 y recibe un as. El nuevo total podría ser 8 o 18. Sin embargo, `obtenerValorMano` siempre incrementa al máximo el valor de las cartas sin pasar de 21, por lo que el nuevo total es 18.

28.6.2 Cómo consumir el servicio Web **Blackjack**

Ahora usaremos el servicio Web **Blackjack** en una aplicación de java (figura 28.14). La aplicación lleva la cuenta de las cartas del jugador y del repartidor, y el servicio Web lleva la cuenta de las cartas que se han repartido.

El constructor (líneas 34 a 83) establece la GUI (línea 36), modifica el color de fondo de la ventana (línea 40) y crea el objeto proxy del servicio Web Blackjack (líneas 46 y 47). En la GUI, cada jugador tiene 11 objetos JLabel: el máximo número de cartas que se pueden repartir sin exceder automáticamente a 21 (es decir, cuatro ases, cuatro de dos y tres de tres). Estos objetos JLabel se colocan en un objeto ArrayList de objetos JLabel (líneas 59 a 82), para que podamos indizar el objeto ArrayList durante el juego y determinar el objeto JLabel que mostrará una imagen de una carta específica.

En el marco de trabajo JAX-WS 2.0, el cliente debe indicar si desea permitir al servicio Web mantener la información de la sesión. En las líneas 50 y 51 del constructor se lleva a cabo esta tarea. Primero convertimos el objeto proxy al tipo de interfaz BindingProvider. Un objeto BindingProvider permite al cliente manipular la información de petición que se enviará al servidor. Esta información se almacena en un objeto que implementa a la interfaz RequestContext. Los objetos BindingProvider y RequestContext son parte del marco de trabajo que crea el IDE cuando agregamos un cliente de servicio Web a la aplicación. A continuación, en las líneas 50 y 51 se invoca el método getRequestContext de BindingProvider para obtener el objeto RequestContext. Luego se hace una llamada al método put de RequestContext para establecer la propiedad BindingProvider.SESSION_MAINTAIN_PROPERTY a true, lo cual permite el rastreo de sesiones desde el lado cliente, de manera que el servicio Web sepa qué cliente está invocando a sus métodos.

```

1 // Fig. 28.14: JuegoBlackjackJFrame.java
2 // Juego de Blackjack que utiliza el servicio Web Blackjack
3 package com.deitel.jhttp7.ch28.clienteb blackjack;
4
5 import java.awt.Color;
6 import java.util.ArrayList;
7 import javax.swing.ImageIcon;
8 import javax.swing.JLabel;
9 import javax.swing.JOptionPane;
10 import javax.xml.ws.BindingProvider;
11 import com.deitel.jhttp7.cap28.clienteb blackjack.Blackjack;
12 import com.deitel.jhttp7.cap28.clienteb blackjack.ServicioBlackjack;
13
14 public class JuegoBlackjackJFrame extends javax.swing.JFrame
15 {
16     private String cartasJugador;
17     private String cartasRepartidor;
18     private ArrayList< JLabel > naipes; // lista de objetos JLabel con imágenes de las
        cartas
19     private int cartaActualJugador; // número de carta actual del jugador
20     private int cartaActualRepartidor; // número de carta actual de proxyBlackjack
21     private ServicioBlackjack servicioBlackjack; // se utiliza para obtener el proxy
22     private Blackjack proxyBlackjack; // se utiliza para acceder al servicio Web
23
24     // enumeración de estados del juego
25     private enum EstadoJuego
26     {
27         EMPATE, // el juego termina en un empate
28         PIERDE, // el jugador pierde
29         GANA, // el jugador gana
30         BLACKJACK // el jugador tiene blackjack
31     } // fin de enum EstadoJuego
32
33     // constructor sin argumentos
34     public JuegoBlackjackJFrame()
35     {
36         initComponents();

```

Figura 28.14 | Juego de Blackjack que utiliza el servicio Web Blackjack. (Parte I de 9).

```

37
38 // debido a un error en Netbeans, debemos cambiar el color de fondo
39 // del objeto JFrame aquí, en vez de hacerlo en el diseñador
40 getContentPane().setBackground( new Color( 0, 180, 0 ) );
41
42 // inicializa el proxy blackjack
43 try
44 {
45     // crea los objetos para acceder al servicio Web Blackjack
46     servicioBlackjack = new ServicioBlackjack();
47     proxyBlackjack = servicioBlackjack.getBlackjackPort();
48
49     // habilita el rastreo de sesiones
50     ( ( BindingProvider ) proxyBlackjack ).getRequestContext().put(
51         BindingProvider.SESSION_MAINTAIN_PROPERTY, true );
52 } // fin de try
53 catch ( Exception e )
54 {
55     e.printStackTrace();
56 } // fin de catch
57
58 // agrega componentes JLabel al objeto ArrayList naipes para manipularlo mediante
59 // programación
60 naipes = new ArrayList< JLabel >();
61
62 naipes.add( 0, carta1RepartidorJLabel );
63 naipes.add( carta2RepartidorJLabel );
64 naipes.add( carta3RepartidorJLabel );
65 naipes.add( carta4RepartidorJLabel );
66 naipes.add( carta5RepartidorJLabel );
67 naipes.add( carta6RepartidorJLabel );
68 naipes.add( carta7RepartidorJLabel );
69 naipes.add( carta8RepartidorJLabel );
70 naipes.add( carta9RepartidorJLabel );
71 naipes.add( carta10RepartidorJLabel );
72 naipes.add( carta11RepartidorJLabel );
73 naipes.add( carta1JugadorJLabel );
74 naipes.add( carta2JugadorJLabel );
75 naipes.add( carta3JugadorJLabel );
76 naipes.add( carta4JugadorJLabel );
77 naipes.add( carta5JugadorJLabel );
78 naipes.add( carta6JugadorJLabel );
79 naipes.add( carta7JugadorJLabel );
80 naipes.add( carta8JugadorJLabel );
81 naipes.add( carta9JugadorJLabel );
82 naipes.add( carta10JugadorJLabel );
83 naipes.add( carta11JugadorJLabel );
84 } // fin del constructor sin argumentos
85
86 // juega la mano del repartidor
87 private void juegoRepartidor()
88 {
89     try
90     {
91         // mientras el valor de la mano del repartidor sea menor a 17
92         // el repartidor debe seguir tomando cartas
93         String[] cartas = cartasRepartidor.split( "\\t" );
94
95         // muestra las cartas del repartidor

```

Figura 28.14 | Juego de Blackjack que utiliza el servicio Web Blackjack. (Parte 2 de 9).

```

95     for ( int i = 0; i < cartas.length; i++ )
96         mostrarCarta( i, cartas[ i ] );
97
98     while ( proxyBlackjack.obtenerValorMano( cartasRepartidor ) < 17 )
99     {
100         String nuevaCarta = proxyBlackjack.repartirCarta(); // reparte una nueva
            carta
101         cartasRepartidor += "\t" + nuevaCarta; // reparte una nueva carta
102         mostrarCarta( cartaActualRepartidor, nuevaCarta );
103         ++cartaActualRepartidor;
104         JOptionPane.showMessageDialog( this, "El repartidor toma una carta",
105             "Turno del repartidor", JOptionPane.PLAIN_MESSAGE );
106     } // end while
107
108     int totalRepartidor = proxyBlackjack.obtenerValorMano( cartasRepartidor );
109     int totalJugador = proxyBlackjack.obtenerValorMano( cartasJugador );
110
111     // si el repartidor se pasó, el jugador gana
112     if ( totalRepartidor > 21 )
113     {
114         finDelJuego( EstadoJuego.GANA );
115         return;
116     } // fin de if
117
118     // si el repartidor y el jugador tienen menos de 21
119     // la mayor puntuación gana, si tienen igual puntuación es un empate
120     if ( totalRepartidor > totalJugador )
121         finDelJuego( EstadoJuego.PIERDE );
122     else if ( totalRepartidor < totalJugador )
123         finDelJuego( EstadoJuego.GANA );
124     else
125         finDelJuego( EstadoJuego.EMPATE );
126 } // fin de try
127 catch ( Exception e )
128 {
129     e.printStackTrace();
130 } // fin de catch
131 } // fin del método juegoRepartidor
132
133 // muestra la carta representada por valorCarta en el objeto JLabel especificado
134 public void mostrarCarta( int carta, String valorCarta )
135 {
136     try
137     {
138         // obtiene el objeto JLabel correcto de naipes
139         JLabel mostrarEtiqueta = naipes.get( carta );
140
141         // si la cadena que representa la carta está vacía, muestra la parte posterior
142         // de la carta
143         if ( valorCarta.equals( "" ) )
144         {
145             mostrarEtiqueta.setIcon( new ImageIcon( getClass().getResource(
146                 "/com/deitel/jhttp7/cap28/clienteblackjack/" +
147                 "blackjack_imagenes/cartpost.png" ) ) );
148             return;
149         } // fin de if
150
151         // obtiene el valor de la cara de la carta
152         String cara = valorCarta.substring( 0, valorCarta.indexOf( " " ) );

```

Figura 28.14 | Juego de Blackjack que utiliza el servicio Web Blackjack. (Parte 3 de 9).

```

152
153 // obtiene el palo de la carta
154 String palo =
155     valorCarta.substring( valorCarta. indexOf( " " ) + 1 );
156
157 char letraPalo; // letra del palo que se usa para formar el archivo de imagen
158
159 switch ( Integer.parseInt( palo ) )
160 {
161     case 0: // corazones
162         letraPalo = 'c';
163         break;
164     case 1: // diamantes
165         letraPalo = 'd';
166         break;
167     case 2: // bastos
168         letraPalo = 'b';
169         break;
170     default: // espadas
171         letraPalo = 'e';
172         break;
173 } // fin de switch
174
175 // establece la imagen para mostrarEtiqueta
176 mostrarEtiqueta.setIcon( new ImageIcon( getClass().getResource(
177     "/com/deitel/jhttp7/cap28/clienteblackjack/blackjack_imagenes/" +
178     cara + letraPalo + ".png" ) ) );
179 } // fin de try
180 catch ( Exception e )
181 {
182     e.printStackTrace();
183 } // fin de catch
184 } // fin del método mostrarCarta
185
186 // muestra todas las cartas del jugador y el mensaje apropiado
187 public void finDelJuego( EstadoJuego ganador )
188 {
189     String[] cartas = cartasRepartidor.split( "\\t" );
190
191     // muestra las cartas de proxyBlackjack
192     for ( int i = 0; i < cartas.length; i++ )
193         mostrarCarta( i, cartas[ i ] );
194
195     // muestra la imagen del estado apropiado
196     if ( ganador == EstadoJuego.GANA )
197         estadoJLabel.setText( "Usted gana!" );
198     else if ( ganador == EstadoJuego.PIERDE )
199         estadoJLabel.setText( "Usted pierde." );
200     else if ( ganador == EstadoJuego.EMPATE )
201         estadoJLabel.setText( "Es un empate." );
202     else // blackjack
203         estadoJLabel.setText( "Blackjack!" );
204
205     // muestra las puntuaciones finales
206     int totalRepartidor = proxyBlackjack.obtenerValorMano( cartasRepartidor );
207     int totalJugador = proxyBlackjack.obtenerValorMano( cartasJugador );
208     totalRepartidorJLabel.setText( "Repartidor: " + totalRepartidor );
209     totalJugadorJLabel.setText( "Jugador: " + totalJugador );
210

```

Figura 28.14 | Juego de Blackjack que utiliza el servicio Web Blackjack. (Parte 4 de 9).

```

211     // restablece para nuevo juego
212     pasarJButton.setEnabled( false );
213     pedirJButton.setEnabled( false );
214     repartirJButton.setEnabled( true );
215 } // fin del método finDeJuego
216
217 // El método initComponents es generado automáticamente por Netbeans y se llama
218 // desde el constructor para inicializar la GUI. No mostraremos aquí este método
219 // para ahorrar espacio. Abra JuegoBlackjackJFrame.java en la carpeta de
220 // este ejemplo para ver el código generado completo (líneas 221 a 531)
221
232 // maneja el clic del objeto pasarJButton
233 private void pasarJButtonActionPerformed(
234     java.awt.event.ActionEvent evt )
235 {
236     pasarJButton.setEnabled( false );
237     pedirJButton.setEnabled( false );
238     repartirJButton.setEnabled( true );
239     juegoRepartidor();
240 } // fin del método pasarJButtonActionPerformed
241
242 // maneja el clic del objeto pedirJButton
243 private void pedirJButtonActionPerformed(
244     java.awt.event.ActionEvent evt )
245 {
246     // obtiene otra carta para el jugador
247     String carta = proxyBlackjack.repartirCarta(); // reparte una nueva carta
248     cartasJugador += "\t" + carta; // agrega la carta a la mano
249
250     // actualiza la GUI para mostrar una nueva carta
251     mostrarCarta( cartaActualJugador, carta );
252     ++cartaActualJugador;
253
254     // determina el nuevo valor de la mano del jugador
255     int total = proxyBlackjack.obtenerValorMano( cartasJugador );
256
257     if ( total > 21 ) // el jugador se pasa
258         finDeJuego( EstadoJuego.PIERDE );
259     if ( total == 21 ) // el jugador no puede tomar más cartas
260     {
261         pedirJButton.setEnabled( false );
262         juegoRepartidor();
263     } // fin de if
264 } // fin del método pedirJButtonActionPerformed
265
266 // maneja el clic del objeto repartirJButton
267 private void repartirJButtonActionPerformed(
268     java.awt.event.ActionEvent evt )
269 {
270     String carta; // almacena una carta temporalmente hasta que se agrega a una mano
271
272     // borra las imágenes de las cartas
273     for ( int i = 0; i < naipes.size(); i++ )
274         naipes.get( i ).setIcon( null );
275
276     estadoJLabel.setText( "" );
277     totalRepartidorJLabel.setText( "" );
278     totalJugadorJLabel.setText( "" );
279

```

Figura 28.14 | Juego de Blackjack que utiliza el servicio Web Blackjack. (Parte 5 de 9).


```

580 // crea un nuevo mazo barajado en un equipo remoto
581 proxyBlackjack.barajar();
582
583 // reparte dos cartas al jugador
584 cartasJugador = proxyBlackjack.repartirCarta(); // agrega la primera carta a la mano
585 mostrarCarta( 11, cartasJugador ); // muestra la primera carta
586 carta = proxyBlackjack.repartirCarta(); // reparte la segunda carta
587 mostrarCarta( 12, carta ); // muestra la segunda carta
588 cartasJugador += "\t" + carta; // agrega la segunda carta a la mano
589
590 // reparte dos cartas a proxyBlackjack, pero sólo muestra la primera
591 cartasRepartidor = proxyBlackjack.repartirCarta(); // agrega la primera carta a
    la mano
592 mostrarCarta( 0, cartasRepartidor ); // muestra la primera carta
593 carta = proxyBlackjack.repartirCarta(); // reparte la segunda carta
594 mostrarCarta( 1, "" ); // muestra la parte posterior de la carta
595 cartasRepartidor += "\t" + carta; // agrega la segunda carta a la mano
596
597 pasarJButton.setEnabled( true );
598 pedirJButton.setEnabled( true );
599 repartirJButton.setEnabled( false );
600
601 // determina el valor de las dos manos
602 int totalRepartidor = proxyBlackjack.obtenerValorMano( cartasRepartidor );
603 int totalJugador = proxyBlackjack.obtenerValorMano( cartasJugador );
604
605 // si ambas manos son iguales a 21, es un empate
606 if ( totalJugador == totalRepartidor && totalJugador == 21 )
607     finDelJuego( EstadoJuego.EMPATE );
608 else if ( totalRepartidor == 21 ) // proxyBlackjack tiene blackjack
609     finDelJuego( EstadoJuego.PIERDE );
610 else if ( totalJugador == 21 ) // blackjack
611     finDelJuego( EstadoJuego.BLACKJACK );
612
613 // la siguiente carta para proxyBlackjack tiene el índice 2
614 cartaActualRepartidor = 2;
615
616 // la siguiente carta para el jugador tiene el índice 13
617 cartaActualJugador = 13;
618 } // fin del método repartirJButtonActionPerformed
619
620 // empieza la ejecución de la aplicación
621 public static void main( String args[] )
622 {
623     java.awt.EventQueue.invokeLater(
624         new Runnable()
625         {
626             public void run()
627             {
628                 new JuegoBlackjackJFrame().setVisible(true);
629             }
630         }
631     ); // fin de la llamada a java.awt.EventQueue.invokeLater
632 } // fin del método main
633
634 // Declaración de variables - no modificar
635 private javax.swing.JLabel carta10JugadorJLabel;
636 private javax.swing.JLabel carta10RepartidorJLabel;
637 private javax.swing.JLabel carta11JugadorJLabel;

```

Figura 28.14 | Juego de Blackjack que utiliza el servicio Web Blackjack. (Parte 6 de 9).

```

638 private javax.swing.JLabel carta1RepartidorJLabel;
639 private javax.swing.JLabel carta1JugadorJLabel;
640 private javax.swing.JLabel carta1RepartidorJLabel;
641 private javax.swing.JLabel carta2JugadorJLabel;
642 private javax.swing.JLabel carta2RepartidorJLabel;
643 private javax.swing.JLabel carta3JugadorJLabel;
644 private javax.swing.JLabel carta3RepartidorJLabel;
645 private javax.swing.JLabel carta4JugadorJLabel;
646 private javax.swing.JLabel carta4RepartidorJLabel;
647 private javax.swing.JLabel carta5JugadorJLabel;
648 private javax.swing.JLabel carta5RepartidorJLabel;
649 private javax.swing.JLabel carta6JugadorJLabel;
650 private javax.swing.JLabel carta6RepartidorJLabel;
651 private javax.swing.JLabel carta7JugadorJLabel;
652 private javax.swing.JLabel carta7RepartidorJLabel;
653 private javax.swing.JLabel carta8JugadorJLabel;
654 private javax.swing.JLabel carta8RepartidorJLabel;
655 private javax.swing.JLabel carta9JugadorJLabel;
656 private javax.swing.JLabel carta9RepartidorJLabel;
657 private javax.swing.JLabel estadoJLabel;
658 private javax.swing.JLabel jugadorJLabel;
659 private javax.swing.JButton pasarJButton;
660 private javax.swing.JButton pedirJButton;
661 private javax.swing.JLabel repartidorJLabel;
662 private javax.swing.JButton repartirJButton;
663 private javax.swing.JLabel totalJugadorJLabel;
664 private javax.swing.JLabel totalRepartidorJLabel;
665 // End of variables declaration
666 } // fin de la clase JuegoBlackjackJFrame

```

a) Manos del repartidor y del jugador, después de que el usuario hace clic en el botón **Repartir**.



Figura 28.14 | Juego de Blackjack que utiliza el servicio Web Blackjack. (Parte 7 de 9).

b) Manos del repartidor y del jugador, después de que el usuario hace clic en **Pedir** dos veces y luego en **Pasar**. En este caso, el jugador gana.



c) Manos del repartidor y del jugador, después de que el usuario hace clic en el botón **Pasar** con base en la mano inicial. En este caso, el jugador pierde.



Figura 28.14 | Juego de Blackjack que utiliza el servicio Web Blackjack. (Parte 8 de 9).

d) Manos del repartidor y del jugador, después de que al usuario se le reparte blackjack.



e) Manos del repartidor y del jugador, después de que al repartidor se le reparte blackjack.



Figura 28.14 | Juego de Blackjack que utiliza el servicio Web Blackjack. (Parte 9 de 9).

El método `finDeJuego` (líneas 178 a 215) muestra todas las cartas del repartidor, muestra el mensaje apropiado en `estadoJLabel` y muestra los totales finales en puntos del repartidor y del jugador. El método `finDeJuego` recibe como argumento un miembro de la enumeración `EstadoJuego` (definida en las líneas 25 a 31). La enumeración representa si el jugador empató, perdió o ganó el juego; sus cuatro miembros son `EMPATE`, `PIERDE`, `GANAR` y `BLACKJACK`.

Cuando el jugador hace clic en el botón **Repartir**, el método `repartirJButtonActionPerformed` (líneas 567 a 618) borra todos los objetos `JLabel` que muestran cartas o información sobre el estado del juego. A continuación, el mazo se baraja (línea 581), y el jugador y el repartidor reciben dos cartas cada uno (líneas 584 a 595). Después, en las líneas 602 y 603 se calcula el total de cada mano. Si el jugador y el repartidor obtienen puntuaciones de 21, el programa llama al método `finDeJuego`, y le pasa `EstadoJuego.EMPATE` (línea 607). Si sólo el repartidor tiene 21, el programa pasa `EstadoJuego.PIERDE` al método `finDeJuego` (línea 609). Si sólo el jugador tiene 21 después de repartir las primeras dos cartas, el programa pasa `EstadoJuego.BLACKJACK` al método `finDeJuego` (línea 611).

Si `repartirJButtonActionPerformed` no llama a `finDeJuego`, el jugador puede tomar más cartas haciendo clic en el botón **Pedir**, el cual llama a `pedirJButtonActionPerformed` en las líneas 543 a 564. Cada vez que un jugador hace clic en **Pedir**, el programa reparte una carta más al jugador y la muestra en la GUI. Si el jugador excede a 21, el juego termina y el jugador pierde. Si el jugador tiene exactamente 21, no puede tomar más cartas y se hace una llamada al método `juegoRepartidor` (líneas 86 a 131), lo cual provoca que el repartidor tome cartas hasta que su mano tenga un valor de 17 o más (líneas 98 a 106). Si el repartidor excede a 21, el jugador gana (línea 114); en caso contrario, se comparan los valores de las manos, y se hace una llamada a `finDeJuego` con el argumento apropiado (líneas 120 a 125).

Al hacer clic en el botón **Pasar**, estamos indicando que el jugador no desea recibir otra carta. El método `pasarJButtonActionPerformed` (líneas 533 a 540) deshabilita los botones **Pedir** y **Pasar**, habilita el botón **Repartir** y después llama al método `juegoRepartidor`.

El método `mostrarCarta` (líneas 134 a 184) actualiza la GUI para mostrar una carta recién repartida. El método recibe como argumentos un índice entero para el objeto `JLabel` en el objeto `ArrayList` que debe tener establecida su imagen, y un objeto `String` que representa a la carta. Un objeto `String` vacío indica que deseamos mostrar la cara de la carta hacia abajo. Si el método `mostrarCarta` recibe un objeto `String` que no esté vacío, el programa extrae la cara y el palo del objeto `String`, y utiliza esta información para mostrar la imagen correcta. La instrucción `switch` (líneas 159 a 173) convierte el número que representa el palo en un entero, y asigna el carácter apropiado a `cartaPalo` (c para corazones, d para diamantes, b para bastos y e para espadas). El carácter en `tetraPalo` se utiliza para completar el nombre del archivo de imagen (líneas 176 a 178).

En este ejemplo, aprendió a configurar un servicio Web para dar soporte al manejo de sesiones, de manera que pueda llevar la cuenta del estado de la sesión de cada cliente. También aprendió a indicar desde una aplicación de escritorio cliente que desea participar en el rastreo de sesiones. Ahora aprenderá a acceder a una base de datos desde un servicio Web, y cómo consumir un servicio Web desde una aplicación Web cliente.

28.7 Cómo consumir un servicio Web controlado por base de datos desde una aplicación Web

Nuestros ejemplos anteriores acceden a los servicios Web desde aplicaciones de escritorio creadas en Netbeans. Sin embargo, podemos utilizarlos con igual facilidad en aplicaciones Web creadas con Netbeans o Sun Java Studio Creator 2. De hecho, y como los comercios basados en Internet prevalecen cada vez más, es común que las aplicaciones Web consuman servicios Web. En esta sección le presentaremos un servicio Web de reservación de una aerolínea, que recibe información sobre el tipo de asiento que desea reservar un cliente, y hace una reservación si está disponible dicho asiento. Más adelante en esta sección, le presentaremos una aplicación Web que permite a un cliente especificar una petición de reservación, y después utiliza el servicio Web de reservación de una aerolínea para tratar de ejecutar la petición. Utilizaremos Sun Java Studio Creator 2 para crear la aplicación Web.

28.7.1 Configuración de Java DB en Netbeans y creación de la base de datos Reservacion

En este ejemplo, nuestro servicio Web utiliza una base de datos llamada `Reservacion`, la cual contiene una sola tabla llamada `Asientos` para localizar un asiento que coincida con la petición de un cliente. Usted creará la base

de datos *Reservacion* mediante el uso de las herramientas que se proporcionan en Netbeans para crear y manipular bases de datos Java DB.

Cómo agregar una base de datos Java DB

Para agregar un servidor de bases de datos Java DB en Netbeans, realice los siguientes pasos:

1. Seleccione **Tools > Options...** para que aparezca el cuadro de diálogo **Options** de Netbeans.
2. Haga clic en el botón **Advanced Options** para mostrar el cuadro de diálogo **Advanced Options**.
3. En **IDE Configuration**, expanda el nodo **Server and External Tool Settings** y seleccione **Java DB Database**.
4. Si las propiedades de Java DB no están ya configuradas, establezca la propiedad **Java DB Location** a la ubicación de Java DB en su sistema. JDK 6 incluye una versión de Java DB, la cual se ubica en Windows, en el directorio C:\Archivos de programa\Java\jdk1.6.0\db. Sun Java System Application Server también se incluye con Java DB en C:\Sun\AppServer\javadb. Además, establezca la propiedad **Database Location** a la ubicación en donde desea que se almacenen las bases de datos Java DB.



Creación de una base de datos Java DB

Ahora que está configurado el software de bases de datos, cree una nueva base de datos de la siguiente manera:

1. Seleccione **Tools > Java DB Database > Create Java DB Database...**
2. Escriba el nombre de la base de datos a crear (*Reservacion*), un nombre de usuario (*jhttp7*) y una contraseña (*jhttp7*), y después haga clic en **OK** para crear la base de datos.

Cómo agregar una tabla y datos a la base de datos Asientos

Puede usar la ficha **Runtime** de Netbeans (a la derecha de las fichas **Projects** y **Files**) para crear tablas y ejecutar instrucciones SQL que llenen la base de datos con datos:

1. Haga clic en la ficha **Runtime** de Netbeans y expanda el nodo **Databases**.
2. Netbeans debe estar conectado a la base de datos para ejecutar instrucciones SQL. Si Netbeans ya está conectado, proceda al *paso 3*. Si Netbeans no está conectado a la base de datos, aparecerá el ícono  enseguida del URL de la base de datos (`jdbc:derby://localhost:1527/Reservacion`). En este caso, haga clic con el botón derecho del ratón en el ícono y seleccione **Connect...** Una vez conectado, el ícono cambiará a .
3. Expanda el nodo para la base de datos *Reservacion*, haga clic con el botón derecho del ratón en el nodo **Tables** y seleccione **Create Table...** para que aparezca el cuadro de diálogo **Create Table**. Agregue una tabla llamada *Asientos* a la base de datos, y establezca las columnas *Numero*, *Ubicacion*, *Clase* y *Reservado*, como se muestra en la figura 28.15. Use el botón **Add column** para agregar una fila en el cuadro de diálogo por cada columna en la base de datos.

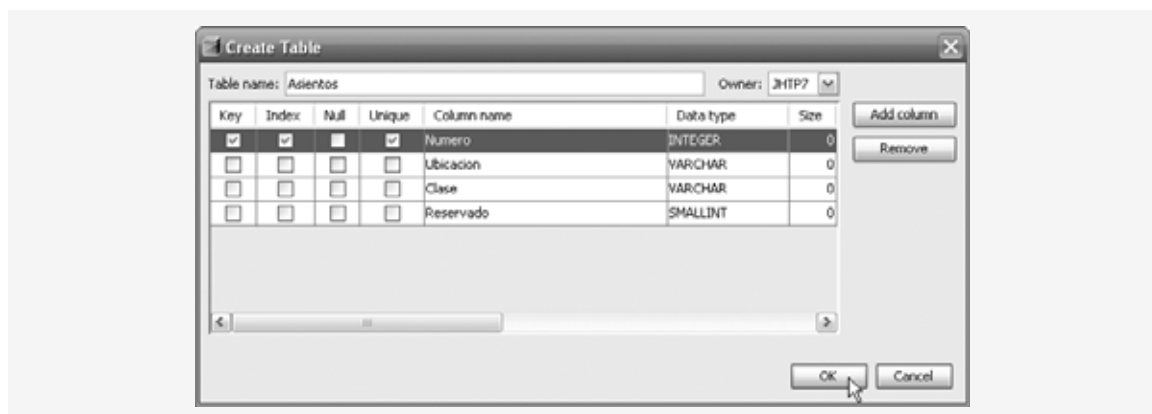


Figura 28.15 | Tabla de configuración de la base de datos *Asientos*.

Número	Ubicación	Clase	Reservado
1	Pasillo	Economica	0
2	Pasillo	Economica	0
3	Pasillo	Primera	0
4	Centro	Economica	0
5	Centro	Economica	0
6	Centro	Primera	0
7	Ventana	Economica	0
8	Ventana	Economica	0
9	Ventana	Primera	0
10	Ventana	Primera	0

Figura 28.16 | Datos de la tabla Asientos.

4. A continuación, use comandos `INSERT INTO` para llenar la base de datos con los datos que se muestran en la figura 28.16. Para ello, haga clic con el botón derecho del ratón en la tabla **Asientos** en la ficha **Runtime** y seleccione **Execute Command...** para que aparezca una ficha **SQL Command** en el editor de Netbeans. El archivo `InstruccionesSQLParaLaFig28_16.txt` que se proporciona con los ejemplos de este capítulo contiene los 10 comandos `INSERT INTO` que almacenan los datos que se muestran en la figura 28.16. Sólo copie el texto en ese archivo y péguelo en la ficha **SQL Command**, y después oprima el botón **Run SQL** (🏃) a la derecha de la lista desplegable **Connection** en la ficha **SQL Command** para ejecutar los comandos. Para confirmar que los datos se hayan insertado correctamente, haga clic con el botón derecho del ratón en la tabla **Asientos** en la ficha **Runtime**, y seleccione **View Data...**

Creación del servicio Web Reservacion

Ahora puede crear un servicio Web que utilice la base de datos **Reservacion** (figura 28.17). El servicio Web de reservación de la aerolínea tiene un solo método Web: **reservar** (líneas 25 a 73), el cual busca en una base de datos **Reservacion** que contenga una sola tabla llamada **Asientos** para localizar un asiento que coincida con la petición del usuario. El método recibe dos argumentos: un objeto `String` que representa el tipo de asiento deseado (es decir, "Ventana", "Centro" o "Pasillo") y un objeto `String` que representa el tipo de clase deseado (es decir, "Economica" o "Primera"). Si encuentra un asiento apropiado, el método **reservar** actualiza la base de datos para hacer la reservación y devuelve `true`; en caso contrario, no se realiza la reservación y el método devuelve `false`. Observe que las instrucciones en las líneas 34 a 37 y en las líneas 43 a 44, que consultan y actualizan la base de datos, usan objetos de los tipos `ResultSet` y `Statement` (que vimos en el capítulo 25).

Nuestra base de datos contiene cuatro columnas: el número de asiento (es decir, 1-10), el tipo de asiento (es decir, Ventana, Centro o Pasillo), el tipo de clase (Economica o Primera), y una columna que contiene 1 (verdadero) o 0 (falso) para indicar si el asiento está reservado o no. En las líneas 34 a 37 se obtienen los números de asiento de cualquier asiento disponible que coincida con el asiento y tipo de clase solicitados. Esta instrucción llena el objeto `conjuntoResultados` con los resultados de la consulta

```
SELECT "Numero"
FROM "Asientos"
WHERE ("Reservado" = 0) AND ("Tipo" = tipo) AND ("Clase" = clase)
```

Los parámetros *tipo* y *clase* en la consulta se sustituyen con los valores de los parámetros `tipoAsiento` y `tipoClase` del método **reservar**. Cuando usamos las herramientas de Netbeans para crear una tabla de una base de datos y sus columnas, las herramientas de Netbeans colocan automáticamente la tabla y los nombres de las columnas entre comillas dobles. Por esta razón, debemos colocar la tabla y los nombres de las columnas entre comillas dobles en las instrucciones SQL que interactúan con la base de datos **Reservacion**.

Si `conjuntoResultados` no está vacío (es decir, que por lo menos haya un asiento disponible que coincida con los criterios seleccionados), la condición en la línea 40 es `true` y el servicio Web reserva el primer número de asiento que coincida. Recuerde que el método `next` de `conjuntoResultados` devuelve `true` si existe una fila que no esté vacía, y posiciona el cursor en esa fila. Para obtener el número de asiento (línea 42), accedemos a la primera columna de `conjuntoResultado` (es decir, `conjuntoResultados.getInt(1)`; la primera columna en la fila). Después, en las líneas 43 y 44 se invoca el método `executeUpdate` de `instruccion` para ejecutar el SQL:

```
UPDATE "Asientos"
SET "Reservado" = 1
WHERE ("Numero" = número)
```

el cual marca el asiento como reservado en la base de datos. El parámetro *número* se sustituye con el valor de `numeroAsiento`. El método `reserve` devuelve `true` (línea 45) para indicar que la reservación se realizó con éxito. Si no hay asientos que coincidan, o si ocurrió una excepción, el método `reserve` devuelve `false` (líneas 48, 53, 58 y 70) para indicar que ningún asiento coincidió con la petición del usuario.

```
1 // Fig. 28.17: Reservacion.java
2 // Servicio Web de reservaciones de una aerolínea.
3 package com.deitel.jhttp7.cap28.reservacion;
4
5 import java.sql.Connection;
6 import java.sql.Statement;
7 import java.sql.DriverManager;
8 import java.sql.ResultSet;
9 import java.sql.SQLException;
10 import javax.jws.WebService;
11 import javax.jws.WebMethod;
12 import javax.jws.WebParam;
13
14 @WebService( name = "Reservacion", serviceName = "ServicioReservacion" )
15 public class Reservacion
16 {
17     private static final String URL_BASEDATOS =
18         "jdbc:derby://localhost:1527/Reservacion";
19     private static final String USUARIO = "jhttp7";
20     private static final String CONTRASENIA = "jhttp7";
21     private Connection conexion;
22     private Statement instruccion;
23
24     // un Método Web que puede reservar un asiento
25     @WebMethod( operationName = "reservar" )
26     public boolean reserve( @WebParam( name = "tipoAsiento" ) String tipoAsiento,
27         @WebParam( name = "tipoClase" ) String tipoClase )
28     {
29         try
30         {
31             conexion = DriverManager.getConnection(
32                 URL_BASEDATOS, USUARIO, CONTRASENIA );
33             instruccion = conexion.createStatement();
34             ResultSet conjuntoResultados = instruccion.executeQuery(
35                 "SELECT \"Numero\" FROM \"Asientos\" " +
36                 "WHERE (\"Reservado\" = 0) AND (\"Ubicacion\" = '" + tipoAsiento +
37                 "') AND (\"Clase\" = '" + tipoClase + "') );";
38
39             // si el asiento solicitado está disponible, lo reserva
40             if ( conjuntoResultados.next() )
```

Figura 28.17 | Servicio Web de reservaciones de una aerolínea. (Parte 1 de 2).


```

41      {
42          int asiento = conjuntoResultados.getInt( 1 );
43          instruccion.executeUpdate( "UPDATE \"Asientos\" " +
44              "SET \"Reservado\" = 1 WHERE \"Numero\" = " + asiento );
45          return true;
46      } // fin de if
47
48      return false;
49  } // fin de try
50  catch ( SQLException e )
51  {
52      e.printStackTrace();
53      return false;
54  } // fin de catch
55  catch ( Exception e )
56  {
57      e.printStackTrace();
58      return false;
59  } // fin de catch
60  finally
61  {
62      try
63      {
64          instruccion.close();
65          conexion.close();
66      } // fin de try
67      catch ( Exception e )
68      {
69          e.printStackTrace();
70          return false;
71      } // fin de catch
72  } // fin de finally
73  } // fin del Método Web reservar
74  } // fin de la clase Reservacion

```

Figura 28.17 | Servicio Web de reservaciones de una aerolínea. (Parte 2 de 2).

28.7.2 Creación de una aplicación Web para interactuar con el servicio Web Reservacion

En esta sección presentaremos una aplicación Web llamada *ClienteReservacion* para consumir el servicio Web *Reservacion*. La aplicación permite a los usuarios seleccionar asientos con base en la clase ("Económica" o "Primera") y la ubicación ("Pasillo", "Centro" o "Ventana"), y después enviar sus peticiones al servicio Web de reservaciones de la aerolínea. Si la petición de la base de datos no tiene éxito, la aplicación instruye al usuario para que modifique la petición e intente de nuevo. La aplicación que presentamos aquí se creó mediante Sun Java Studio Creator 2, JavaServer Faces (JSF) y las técnicas presentadas en los capítulos 26 y 27.

Cómo agregar una referencia de servicio Web a un proyecto en Sun Java Studio Creator 2

Para agregar un servicio Web a una aplicación Web en Java Studio Creator 2, realice los siguientes pasos:

1. Haga clic en el botón **Agregar servicio Web...** (🔗) para que aparezca el cuadro de diálogo **Agregar servicio Web**.
2. Haga clic en el botón **Obtener información sobre el servicio Web**.
3. Haga clic en **Agregar** para cerrar el cuadro de diálogo y agregar el proxy del servicio Web a la aplicación Web. Ahora el servicio Web aparecerá en la ficha **Servidores** de Java Studio Creator 2, bajo el nodo **Servicios Web**.

4. Haga clic con el botón derecho del ratón en el nodo **ServicioReservacion** bajo el nodo **Servicios Web**, y seleccione **Agregar a página** para crear una instancia de la clase proxy del servicio Web que puede usar en la clase **Reservar** que proporciona la lógica de la JSP.

Para los fines de este ejemplo, vamos a suponer que usted ya leyó los capítulos 26 y 27, y por ende sabe cómo crear la GUI de una aplicación Web, crear manejadores de eventos, y agregar propiedades al bean de sesión de una aplicación Web (que vimos en la sección 26.4.4).

Reservar.jsp

El archivo **Reservar.jsp** (figura 28.18) define dos objetos **ListaDesplegable** y un objeto **Botón**. El objeto **tipoAsientoListaDesplegable** (líneas 26 a 31) muestra todos los tipos de asientos que el usuario puede seleccionar. El objeto **claseListaDesplegable** (líneas 32 a 37) proporciona opciones para el tipo de clase. Los usuarios hacen clic en el objeto **Botón** llamado **reservarBoton** (líneas 38 a 41) para enviar peticiones después de realizar selecciones de los objetos **ListaDesplegable**. Esta página también define tres objetos **Etiqueta**: **instruccionesEtiqueta** (líneas 21 a 25) para mostrar instrucciones, **exitoEtiqueta** (líneas 42 a 45) para indicar una reservación exitosa, y **errorEtiqueta** (líneas 46 a 50) para mostrar un mensaje apropiado si no hay un asiento disponible que coincida con la selección del usuario. El archivo de bean de página (figura 28.19) adjunta manejadores de eventos a **tipoAsientoListaDesplegable**, **claseListaDesplegable** y **reservarBoton**.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <!-- Fig. 28.18 Reservar.jsp -->
3  <!-- JSP que permite a un usuario seleccionar un asiento -->
4
5  <jsp:root version="1.2" xmlns:f="http://java.sun.com/jsf/core"
6      xmlns:h="http://java.sun.com/jsf/html"
7      xmlns:jsp="http://java.sun.com/JSP/Page"
8      xmlns:ui="http://www.sun.com/web/ui">
9      <jsp:directive.page contentType="text/html; charset=UTF-8"
10         pageEncoding="UTF-8"/>
11      <f:view>
12          <ui:page binding="#{Reservar.page1}" id="page1">
13              <ui:html binding="#{Reservar.html1}" id="html1">
14                  <ui:head binding="#{Reservar.head1}" id="head1">
15                      <ui:link binding="#{Reservar.link1}" id="link1"
16                          url="/resources/stylesheet.css"/>
17                  </ui:head>
18                  <ui:body binding="#{Reservar.body1}" id="body1"
19                      style="-rave-layout: grid">
20                      <ui:form binding="#{Reservar.form1}" id="form1">
21                          <ui:label binding="#{Reservar.instruccionesEtiqueta}"
22                              id="instruccionesEtiqueta"
23                              style="position: absolute; left: 24px; top: 24px"
24                              text="Seleccione el tipo de asiento y la clase a
25                              reservar:"/>
26                          <ui:dropDown binding="#{Reservar.tipoAsientoListaDesplegable}"
27                              id="tipoAsientoListaDesplegable" items=
28                              "#{Reservar.tipoAsientoListaDesplegableDefaultOptions.options}"
29                              style="left: 24px; top: 48px; position: absolute;
30                              width: 96px" valueChangeListener=
31                              "#{Reservar.tipoAsientoListaDesplegable_processValueChange}"/>
32                          <ui:dropDown binding="#{Reservar.claseListaDesplegable}"
33                              id="claseListaDesplegable" items=
34                              "#{Reservar.claseListaDesplegableDefaultOptions.options}"
35                              style="left: 144px; top: 48px; position: absolute;
36                              width: 96px" valueChangeListener=
37                              "#{Reservar.claseListaDesplegable_processValueChange}"/>

```

Figura 28.18 | JSP que permite a un usuario seleccionar un asiento. (Parte 1 de 3).

d) Intento de reservar otro asiento de ventana en clase económica, cuando no hay dichos asientos disponibles:

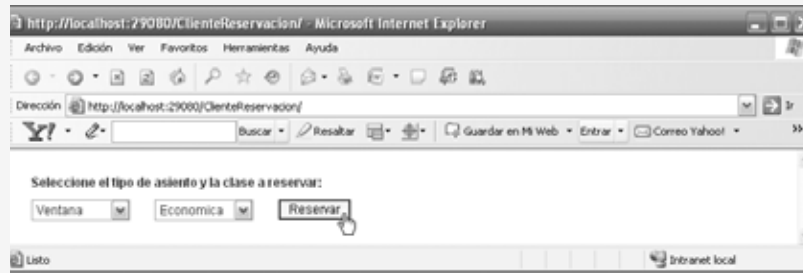


Figura 28.18 | JSP que permite a un usuario seleccionar un asiento. (Parte 3 de 3).

Reservar.java

La figura 28.19 contiene el código de bean de página que proporciona la lógica para `Reservar.jsp` (para ahorrar espacio, no mostramos el código que se genera automáticamente en las líneas 28 a 283). Como vimos en la sección 26.5.2, la clase que representa al bean de página extiende a `AbstractPageBean`. Cuando el usuario hace clic en **Reservar** en la JSP, se ejecuta el manejador de eventos `reservarBoton_action` (líneas 285 a 319). En la línea 289 se crea un objeto proxy `ServicioReservacion1Client`. En las líneas 290 a 292 se utiliza este objeto para invocar el método `reservar` del servicio Web, al cual se le pasa el tipo de asiento seleccionado y el tipo de clase como argumentos. Si `reservar` devuelve `true`, en las líneas 296 a 301 se ocultan los componentes de la GUI en la JSP y se muestra el componente `exitoEtiqueta` (línea 300) para agradecer al usuario por hacer una reservación; en caso contrario, en las líneas 305 a 310 se asegura que los componentes de la GUI se sigan mostrando en pantalla y se muestra el componente `errorEtiqueta` (línea 310) para notificar al usuario que el tipo de asiento solicitado no está disponible, y pide al usuario que intente de nuevo. Cuando el usuario selecciona un valor en uno de los componentes `ListaDesplegable`, se hace una llamada al manejador de eventos correspondiente (`tipoAsientoListaDesplegable_processValueChange` en las líneas 322 a 327, o `claseListaDesplegable_processValueChange` en las líneas 330 a 335) para establecer las propiedades `tipoAsiento` y `tipoClase` de la sesión, las cuales agregamos al bean de sesión de la aplicación. Los valores de estas propiedades se utilizan como argumentos en la llamada al método `reservar` del servicio Web.

```

1 // Fig. 28.19: Reservar.java
2 // Clase de bean de soporte con ámbito de página para el cliente de reservación de asientos
3 package com.deitel.jhttp7.cap28.clientereservacion;
4
5 import com.sun.rave.web.ui.appbase.AbstractPageBean;
6 import com.sun.rave.web.ui.component.Body;
7 import com.sun.rave.web.ui.component.Form;
8 import com.sun.rave.web.ui.component.Head;
9 import com.sun.rave.web.ui.component.Html;
10 import com.sun.rave.web.ui.component.Link;
11 import com.sun.rave.web.ui.component.Page;
12 import javax.faces.FacesException;
13 import com.sun.rave.web.ui.component.Label;
14 import com.sun.rave.web.ui.component.DropDown;
15 import com.sun.rave.web.ui.model.SingleSelectOptionsList;
16 import com.sun.rave.web.ui.component.Button;
17 import com.sun.rave.web.ui.component.StaticText;
18 import webservice.servicioreservacion.
19     servicioreservacion1.ServicioReservacion1Client;
20 import javax.faces.event.ValueChangeEvent;
21

```

Figura 28.19 | Clase de bean de soporte con ámbito de página para el cliente de reservación de asientos. (Parte 1 de 2).

```

22 public class Reservar extends AbstractPageBean
23 {
24     // Para ahorrar espacio, no mostramos aquí las líneas 24 a 283 del código generado
25     // por Java Studio Creator 2. En el archivo Reservar.java podrá ver el código completo
26     // junto con los ejemplos de este capítulo.
27
284 // método que invoca al servicio Web cuando el usuario hace clic en el botón Reservar
285 public String reservarBoton_action()
286 {
287     try
288     {
289         ServicioReservacion1Client cliente = getServicioReservacion1Client1();
290         boolean reservado =
291             cliente.reservar( getSessionBean1().getTipoAsiento(),
292                             getSessionBean1().getTipoClase() );
293
294         if ( reservado )
295         {
296             instruccionesEtiqueta.setVisible( false );
297             tipoAsientoListaDesplegable.setVisible( false );
298             claseListaDesplegable.setVisible( false );
299             reservarBoton.setVisible( false );
300             exitoEtiqueta.setVisible( true );
301             errorEtiqueta.setVisible( false );
302         } // fin de if
303         else
304         {
305             instruccionesEtiqueta.setVisible( true );
306             tipoAsientoListaDesplegable.setVisible( true );
307             claseListaDesplegable.setVisible( true );
308             reservarBoton.setVisible( true );
309             exitoEtiqueta.setVisible( false );
310             errorEtiqueta.setVisible( true );
311         } // fin de else
312     } // fin de try
313     catch ( Exception e )
314     {
315         e.printStackTrace();
316     } // fin de catch
317
318     return null;
319 } // fin del método reservarBoton_action
320
321 // almacena el tipo de asiento seleccionado en el bean de sesión
322 public void tipoAsientoListaDesplegable_processValueChange(
323     ValueChangeEvent event)
324 {
325     getSessionBean1().setTipoAsiento(
326         ( String ) tipoAsientoListaDesplegable.getSelected() );
327 } // fin del método tipoAsientoListaDesplegable_processValueChange
328
329 // almacena la clase seleccionada en el bean de sesión
330 public void claseListaDesplegable_processValueChange(
331     ValueChangeEvent event)
332 {
333     getSessionBean1().setTipoClase(
334         ( String ) claseListaDesplegable.getSelected() );
335 } //fin del método claseListaDesplegable_processValueChange
336 } // fin de la clase Reservar

```

Figura 28.19 | Clase de bean de soporte con ámbito de página para el cliente de reservación de asientos. (Parte 2 de 2).

28.8 Cómo pasar un objeto de un tipo definido por el usuario a un servicio Web

Los métodos Web que hemos demostrado hasta ahora reciben y devuelven sólo valores de tipos primitivos u objetos `String`. Los servicios Web también pueden recibir y devolver objetos de tipos definidos por el usuario; a éstos se les conoce como **tipos personalizados**. En esta sección presentaremos un servicio Web llamado `GeneradorEcuaciones`, el cual genera preguntas aritméticas aleatorias de tipo `Ecuacion`. El cliente es una aplicación de escritorio para enseñar matemáticas, en la cual el usuario selecciona el tipo de pregunta matemática que desea resolver (suma, resta o multiplicación) y el nivel de habilidad del usuario; el nivel 1 utiliza números de un dígito en cada pregunta, el nivel 2 utiliza números de dos dígitos y el nivel 3 utiliza números de tres dígitos. El cliente pasa esta información al servicio Web, el cual a su vez genera un objeto `Ecuacion` que consiste en números aleatorios con el número apropiado de dígitos. La aplicación cliente recibe el objeto `Ecuacion`, muestra la pregunta de ejemplo al usuario en una aplicación Java, permite al usuario proporcionar una respuesta y la verifica para determinar si es correcta o no.

Serialización de los tipos definidos por el usuario

Anteriormente mencionamos que todos los tipos que se pasan o reciben de servicios Web SOAP deben recibir soporte de SOAP. Entonces, ¿cómo puede SOAP dar soporte a un tipo que no se ha creado todavía? Los tipos personalizados que se envían o reciben de un servicio Web se serializan en formato XML. A este proceso se le conoce como **serialización XML**. El proceso de serializar objetos a XML y deserializar objetos de XML se maneja de manera automática, sin necesidad de que el programador intervenga.

Requerimientos para los tipos definidos por el usuario que se utilizan con métodos Web

Una clase que se utiliza para especificar tipos de parámetros o de valores de retorno en los métodos Web debe cumplir varios requerimientos:

1. Debe proporcionar un constructor predeterminado `public` o sin argumentos. Cuando un servicio Web, o el consumidor de un servicio Web, recibe un objeto serializado XML, el Marco de trabajo JAX-WS 2.0 debe tener la capacidad de llamar a este constructor al momento de deserializar el objeto (es decir, convertirlo de XML otra vez en un objeto Java).
2. Las variables de instancia que deben serializarse en formato XML deben tener métodos *set* (*establecer*) y *get* (*obtener*) `public` para acceder a las variables de instancia `private` (recomendado), o las variables de instancia deben declararse `public` (no se recomienda).
3. Las variables de instancia que no sean `public` y deban serializarse deben proporcionar métodos *set* y *get* (incluso si tienen cuerpos vacíos); en caso contrario, no se serializan.

Cualquier variable de instancia que no se serialice simplemente recibe su valor predeterminado (o el valor proporcionado por el constructor sin argumentos) cuando se deserializa un objeto de la clase.



Error común de programación 28.3

Si tratamos de deserializar un objeto de una clase que no tenga un constructor predeterminado o sin argumentos, se produce un error en tiempo de ejecución.

Definición de la clase `Ecuacion`

En la figura 28.20 definimos la clase `Ecuacion`. En las líneas 18 a 31 se define un constructor que recibe tres argumentos: dos valores `int` que representan a los operandos izquierdo y derecho, y un valor `String` que representa la operación aritmética a realizar. El constructor establece las variables de instancia `operandoIzq`, `operandoDer` y `tipoOperacion`, y después calcula el resultado apropiado. El constructor sin argumentos (líneas 13 a 16) llama al constructor de tres argumentos (líneas 18 a 31) y le pasa valores predeterminados. No utilizamos constructor sin argumentos de manera explícita, pero el mecanismo de serialización de XML lo utiliza al momento de deserializar objetos de esta clase. Como proporcionamos un constructor con parámetros, debemos definir de manera explícita el constructor sin argumentos en esta clase, de manera que puedan pasarse (o devolverse) objetos de la clase hacia o desde los métodos Web.

```

1 // Fig. 28.20: Ecuacion.java
2 // Clase Ecuacion que contiene información acerca de una ecuación
3 package com.deitel.jhttp7.cap28.generadorecuaciones;
4
5 public class Ecuacion
6 {
7     private int operandoIzq;
8     private int operandoDer;
9     private int valorResultado;
10    private String tipoOperacion;
11
12    // constructor sin argumentos requerido
13    public Ecuacion()
14    {
15        this( 0, 0, "+" );
16    } // fin del constructor sin argumentos
17
18    public Ecuacion( int valorIzq, int valorDer, String tipo )
19    {
20        operandoIzq = valorIzq;
21        operandoDer = valorDer;
22        tipoOperacion = tipo;
23
24        // determina valorResultado
25        if ( tipoOperacion.equals( "+" ) ) // suma
26            valorResultado = operandoIzq + operandoDer;
27        else if ( tipoOperacion.equals( "-" ) ) // resta
28            valorResultado = operandoIzq - operandoDer;
29        else // multiplicación
30            valorResultado = operandoIzq * operandoDer;
31    } // fin del constructor con tres argumentos
32
33    // devuelve una representación String de una Ecuacion
34    public String toString()
35    {
36        return operandoIzq + " " + tipoOperacion + " " +
37            operandoDer + " = " + valorResultado;
38    } // fin del método toString
39
40    // devuelve el lado izquierdo de la ecuación como un objeto String
41    public String getLadoIzq()
42    {
43        return operandoIzq + " " + tipoOperacion + " " + operandoDer;
44    } // fin del método getLadoIzq
45
46    // devuelve el lado derecho de la ecuación como un objeto String
47    public String getLadoDer()
48    {
49        return "" + valorResultado;
50    } // fin del método getLadoDer
51
52    // obtiene el operandoIzq
53    public int getOperandoIzq()
54    {
55        return operandoIzq;
56    } // fin del método getOperandoIzq
57
58    // obtiene el operandoDer
59    public int getOperandoDer()

```

Figura 28.20 | Clase Ecuacion que contiene información acerca de una ecuación. (Parte 1 de 2).

```

60     {
61         return operandoDer;
62     } // fin del método getOperandoDer
63
64     // obtiene el valorResultado
65     public int getValorRetorno()
66     {
67         return valorResultado;
68     } // fin del método getValorRetorno
69
70     // obtiene el tipoOperacion
71     public String getTipoOperacion()
72     {
73         return tipoOperacion;
74     } // fin del método getTipoOperacion
75
76     // método set requerido
77     public void setLadoIzq( String value )
78     {
79         // cuerpo vacío
80     } // fin del método setLadoIzq
81
82     // método set requerido
83     public void setLadoDer( String value )
84     {
85         // cuerpo vacío
86     } // fin del método setLadoDer
87
88     // método set requerido
89     public void setOperandoIzq( int value )
90     {
91         // cuerpo vacío
92     } // fin del método setOperandoIzq
93
94     // método set requerido
95     public void setOperandoDer( int value )
96     {
97         // cuerpo vacío
98     } // fin del método setOperandoDer
99
100    // método set requerido
101    public void setValorRetorno( int value )
102    {
103        // cuerpo vacío
104    } // fin del método setValorRetorno
105
106    // método set requerido
107    public void setTipoOperacion( String value )
108    {
109        // cuerpo vacío
110    } // fin del método setTipoOperacion
111 } // fin de la clase Ecuacion

```

Figura 28.20 | Clase Ecuacion que contiene información acerca de una ecuación. (Parte 2 de 2).

La clase Ecuacion define los métodos getLadoIzq y setLadoIzq (líneas 41 a 44 y 77 a 80); getLadoDer y setLadoDer (líneas 47 a 50 y 83 a 86); getOperandoIzq y setOperandoIzq (líneas 53 a 56 y 89 a 92); getOperandoDer y setOperandoDer (líneas 59 a 62 y 95 a 98); getValorRetorno y setValorRetorno (líneas 65 a 68 y 101 a 104); y getTipoOperacion y setTipoOperacion (líneas 71 a 74 y 107 a 110). El cliente del servicio Web no necesita modificar los valores de las variables de instancia. Sin embargo, recuerde que una propiedad se

puede serializar sólo si tiene los métodos de acceso *get* y *set*, o si es *public*. Por lo tanto, proporcionamos métodos *set* con cuerpos vacíos para cada una de las variables de instancia de la clase. El método *getLadoIzq* (líneas 41 a 44) devuelve un objeto *String* que representa todo lo que hay a la izquierda del signo igual (=) en la ecuación, y *getLadoDer* (líneas 47 a 50) devuelve un objeto *String* que representa todo lo que hay a la derecha del signo de igual (=). El método *getOperandoIzq* (líneas 53 a 56) devuelve el entero a la izquierda del operador, y *getOperandoDer* (líneas 59 a 62) obtiene el entero a la derecha del operador. El método *getValorRetorno* (líneas 65 a 68) devuelve la solución a la ecuación, y *getTipoOperacion* (líneas 71 a 74) devuelve el operador en la ecuación. El cliente en este ejemplo no utiliza la propiedad *ladoDer*, pero la incluimos para que los futuros clientes puedan usarla.

Creación del servicio Web *GeneradorEcuaciones*

La figura 28.21 presenta el servicio Web *GeneradorEcuaciones*, el cual crea objetos *Ecuacion* aleatorios y personalizados. Este servicio Web sólo contiene el método *generarEcuacion* (líneas 18 a 31), el cual recibe dos parámetros: la operación matemática (ya sea "+", "-" o "*") y un *int* que representa el nivel de dificultad (1 a 3).

```

1  // Fig. 28.21: GeneradorEcuaciones.java
2  // Servicio Web que genera ecuaciones aleatorias
3  package com.deitel.jhttp7.cap28.generadorecuaciones;
4
5  import java.util.Random;
6  import javax.jws.WebService;
7  import javax.jws.WebMethod;
8  import javax.jws.WebParam;
9
10 @WebService( name = "GeneradorEcuaciones",
11             serviceName = "ServicioGeneradorEcuaciones" )
12 public class GeneradorEcuaciones
13 {
14     private int minimo;
15     private int maximo;
16
17     // genera una ecuación matemática y la devuelve como un objeto Ecuacion
18     @WebMethod( operationName = "generarEcuacion" )
19     public Ecuacion generarEcuacion(
20         @WebParam( name = "operacion" ) String operacion,
21         @WebParam( name = "dificultad" ) int dificultad )
22     {
23         minimo = ( int ) Math.pow( 10, dificultad - 1 );
24         maximo = ( int ) Math.pow( 10, dificultad );
25
26         Random objetoAleatorio = new Random();
27
28         return new Ecuacion(
29             objetoAleatorio.nextInt( maximo - minimo ) + minimo,
30             objetoAleatorio.nextInt( maximo - minimo ) + minimo, operacion );
31     } // fin del método generarEcuacion
32 } // fin de la clase GeneradorEcuaciones

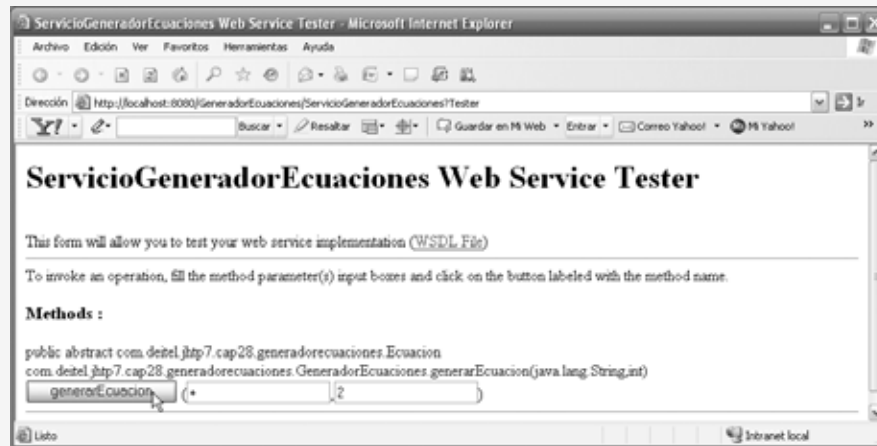
```

Figura 28.21 | Servicio Web que genera ecuaciones aleatorias.

Prueba del servicio Web *GeneradorEcuaciones*

En la figura 28.22 se muestra el resultado de probar el servicio Web *GeneradorEcuaciones* con la página Web Tester. En la *parte (b)* de la figura, observe que el valor de retorno de nuestro método Web está codificado en XML. Sin embargo, este ejemplo es distinto de los anteriores porque el XML especifica los valores para todos los datos del objeto serializado en XML que se devuelve. La clase proxy recibe este valor de retorno y lo deserializa en un objeto de la clase *Ecuacion*, y después lo pasa al cliente.

a) Uso de la página Web del servicio Web **GeneradorEcuaciones** para generar un objeto **Ecuacion**.



b) Resultado de generar un objeto **Ecuacion**.

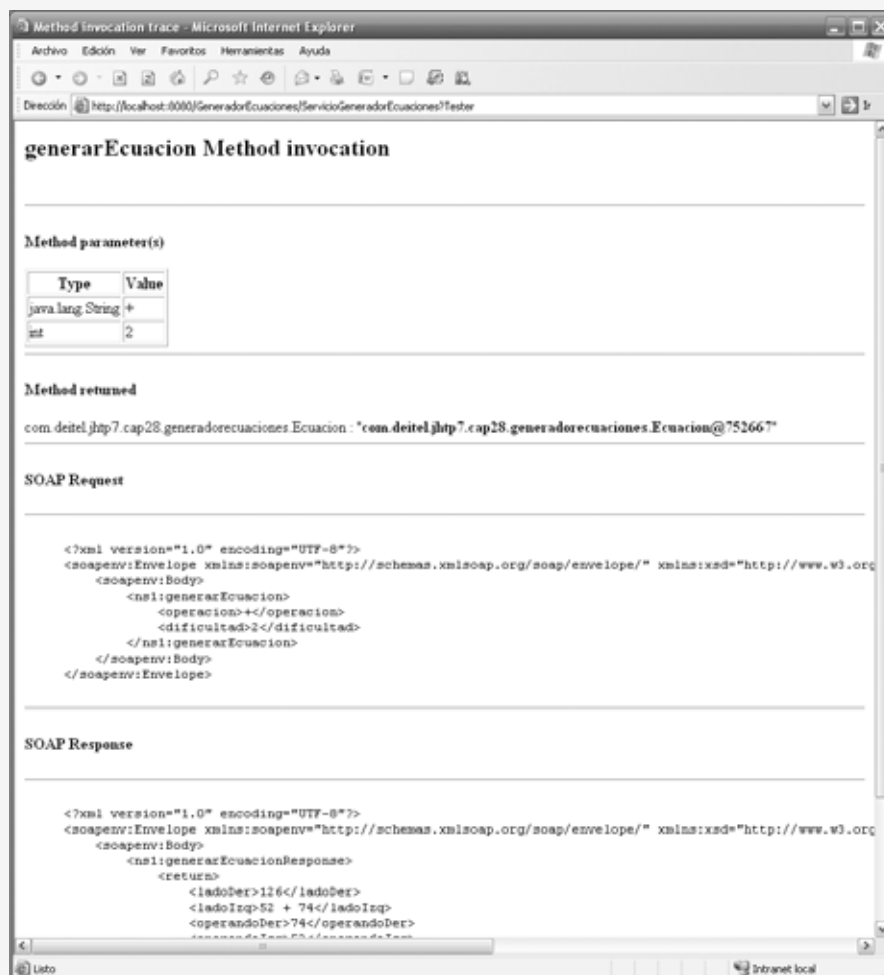


Figura 28.22 | Prueba de un método Web que devuelve un objeto **Ecuacion** serializado con XML.

Observe que *no* se está pasando un objeto `Ecuacion` entre el servicio Web y el cliente. En vez de ello, la información en el objeto se envía en forma de datos codificados en XML. Los clientes creados en Java tomarán la información y crearán un nuevo objeto `Ecuacion`. Sin embargo, los clientes creados en otras plataformas tal vez utilicen la información en forma distinta. Los lectores que creen clientes en otras plataformas deben revisar la documentación de los servicios Web para la plataforma específica que utilicen, para ver cómo pueden sus clientes procesar tipos personalizados.

Detalles del servicio Web `GeneradorEcuaciones`

Vamos a analizar el método Web `GenerarEcuacion` más de cerca. En las líneas 23 a 24 de la figura 28.21 se definen los límites superior e inferior de los números aleatorios que utiliza el método para generar un objeto `Ecuacion`. Para establecer estos límites, el programa llama primero al método `static pow` de la clase `Math`; este método eleva su primer argumento a la potencia de su segundo argumento. El valor de la variable `minimo` se determina elevando 10 a una potencia que sea uno menos que `nivel` (línea 23). Esto calcula el número más pequeño con `nivel` dígitos. Si `nivel` es 1, `minimo` es 1; si `nivel` es 2, `minimo` es 10; y si `nivel` es 3, `minimo` es 100. Para calcular el valor de `maximo` (el límite superior para cualquier número generado al azar que se utilice para formar un objeto `Ecuacion`), el programa eleva 10 a la potencia del argumento `nivel` especificado (línea 23). Si `nivel` es 1, `maximo` es 10; si `nivel` es 2, `maximo` es 100; y si `nivel` es 3, `maximo` es 1000.

En las líneas 28 a 30 se crea y devuelve un nuevo objeto `Ecuacion`, el cual consiste en dos números aleatorios y el objeto `String` llamado `operacion` que recibe `generarEcuacion`. El programa llama al método `Random.nextInt`, el cual devuelve un `int` que es menor que el límite superior especificado. Este método genera un valor de operando izquierdo que es mayor o igual a `minimo`, pero menor que `maximo` (es decir, un número con `nivel` dígitos). El operando derecho es otro número aleatorio con las mismas características.

Consumo del servicio Web `GeneradorEcuaciones`

La aplicación **Tutor de matemáticas** (figura 28.23) utiliza el servicio Web `GeneradorEcuaciones`. Esta aplicación llama al método `generarEcuacion` del servicio Web para crear un objeto `Ecuacion`. Después, el tutor muestra el lado izquierdo del objeto `Ecuacion` y espera a que el usuario introduzca datos. En la línea 9 también se declara una variable de instancia llamada `ServicioGeneradorEcuaciones`, la cual utilizamos para obtener un objeto proxy `GeneradorEcuaciones`. En las líneas 10 a 11 se declaran variables de instancia de los tipos `GeneradorEcuaciones` y `Ecuacion`.

Después de mostrar una ecuación, la aplicación espera a que el usuario escriba una respuesta. La opción predeterminada para el nivel de dificultad es **Números de un dígito**, pero el usuario puede cambiar esto si selecciona un nivel del objeto `JComboBox` llamado **Seleccione el nivel**. Al hacer clic en cualquiera de los niveles se invoca el método `nivelJComboBoxItemStateChanged` (líneas 158 a 163), el cual establece la variable `dificultad` con el nivel seleccionado por el usuario. Aunque la opción predeterminada para el tipo de pregunta es **Suma**, el usuario también puede cambiar esto si selecciona una operación del objeto `JComboBox` **Seleccione la operación**. Al hacer esto, se invoca al método `operacionJComboBoxItemStateChanged` (líneas 166 a 177), el cual establece la variable `String` llamada `operacion` con el símbolo matemático apropiado.

Cuando el usuario hace clic en el objeto `JButton` **Generar ecuación**, el método `generarJButtonActionPerformed` (líneas 207 a 221) invoca al método `generarEcuacion` (línea 212) del servicio Web `GeneradorEcuaciones`. Después de recibir un objeto `Ecuacion` del servicio Web, el manejador muestra el lado izquierdo de la ecuación en el componente `ecuacionJLabel` (línea 214) y habilita el componente `comprobarRespuestaJButton`, de manera que el usuario pueda enviar una respuesta. Cuando el usuario hace clic en el botón `JButton` **Comprobar respuesta**, el método `comprobarRespuestaJButtonActionPerformed` (líneas 180 a 204) determina si el usuario proporcionó la respuesta correcta.

```

1 // Fig. 28.23: ClienteGeneradorEcuacionesJFrame.java
2 // Programa tutor de matemáticas que usa servicios Web para generar ecuaciones
3 package com.deitel.jhttp7.cap28.clientegeneradorecuaciones;
4
5 import javax.swing.JOptionPane;
6

```

Figura 28.23 | Aplicación tutor de matemáticas. (Parte I de 4).

```

7 public class ClienteGeneradorEcuacionesJFrame extends javax.swing.JFrame
8 {
9     private ServicioGeneradorEcuaciones servicio; // se utiliza para obtener el proxy
10    private GeneradorEcuaciones proxy; // se utiliza para acceder al servicio Web
11    private Ecuacion ecuacion; // representa una ecuación
12    private int respuesta; // la respuesta del usuario a la pregunta
13    private String operacion = "+"; // operación matemática +, - o *
14    private int dificultad = 1; // 1, 2 o 3 dígitos en cada número
15
16    // constructor sin argumentos
17    public ClienteGeneradorEcuacionesJFrame()
18    {
19        initComponents();
20
21        try
22        {
23            // crea los objetos para acceder al servicio GeneradorEcuaciones
24            servicio = new ServicioGeneradorEcuaciones();
25            proxy = servicio.getGeneradorEcuacionesPort();
26        } // fin de try
27        catch ( Exception ex )
28        {
29            ex.printStackTrace();
30        } // fin de catch
31    } // fin de constructores sin argumentos
32
33    // El método initComponents se genera automáticamente por Netbeans y se llama
34    // desde el constructor para inicializar la GUI. Aquí no se muestra este
35    // método para ahorrar espacio. Abra ClienteGeneradorEcuacionesJFrame.java en la
36    // carpeta de este ejemplo para ver el código generado completo (líneas 37 a 156).
37
157    // obtiene el nivel de dificultad seleccionado por el usuario
158    private void nivelJComboBoxItemStateChanged(
159        java.awt.event.ItemEvent evt )
160    {
161        // los índices empiezan en 0, por lo que se suma 1 para obtener el nivel de
162        // dificultad
163        dificultad = nivelJComboBox.getSelectedIndex() + 1;
164    } // fin del método nivelJComboBoxItemStateChanged
165
166    // obtiene la operación matemática seleccionada por el usuario
167    private void operacionJComboBoxItemStateChanged(
168        java.awt.event.ItemEvent evt )
169    {
170        String elemento = ( String ) operacionJComboBox.getSelectedItem();
171
172        if ( elemento.equals( "Suma" ) )
173            operacion = "+"; // el usuario seleccionó suma
174        else if ( elemento.equals( "Resta" ) )
175            operacion = "-"; // el usuario seleccionó resta
176        else
177            operacion = "*"; // el usuario seleccionó multiplicación
178    } // fin del método operacionJComboBoxItemStateChanged
179
180    // comprueba la respuesta del usuario
181    private void comprobarRespuestaJButtonActionPerformed(
182        java.awt.event.ActionEvent evt )
183    {
184        if ( respuestaJTextField.getText().equals( "" ) )

```

Figura 28.23 | Aplicación tutor de matemáticas. (Parte 2 de 4).

```

184     {
185         JOptionPane.showMessageDialog(
186             this, "Escriba su respuesta." );
187     } // fin de if
188
189     int respuestaUsuario = Integer.parseInt( respuestaJTextField.getText() );
190
191     if ( respuestaUsuario == respuesta )
192     {
193         ecuacionJLabel.setText( "" );
194         respuestaJTextField.setText( "" );
195         comprobarRespuestaJButton.setEnabled( false );
196         JOptionPane.showMessageDialog( this, "Correcto! Bien hecho!",
197             "Correcto", JOptionPane.PLAIN_MESSAGE );
198     } // fin de if
199     else
200     {
201         JOptionPane.showMessageDialog( this, "Incorrecto. Intente de nuevo.",
202             "Incorrecto", JOptionPane.PLAIN_MESSAGE );
203     } // fin de else
204 } // fin del método checkAnswerJButtonActionPerformed
205
206 // genera un nuevo objeto Ecuacion con base en las selecciones del usuario
207 private void generarJButtonActionPerformed(
208     java.awt.event.ActionEvent evt )
209 {
210     try
211     {
212         ecuacion = proxy.generarEcuacion( operacion, dificultad );
213         respuesta = ecuacion.getValorRetorno();
214         ecuacionJLabel.setText( ecuacion.getLadoIzq() + " = " );
215         comprobarRespuestaJButton.setEnabled( true );
216     } // fin de try
217     catch ( Exception e )
218     {
219         e.printStackTrace();
220     } // fin de catch
221 } // fin del método generateJButtonActionPerformed
222
223 // empieza la ejecución del programa
224 public static void main( String args[] )
225 {
226     java.awt.EventQueue.invokeLater(
227         new Runnable()
228         {
229             public void run()
230             {
231                 new ClienteGeneradorEcuacionesJFrame().setVisible( true );
232             } // fin del método run
233         } // fin de la clase interna anónima
234     ); // fin de la llamada a java.awt.EventQueue.invokeLater
235 } // fin del método main
236
237 // Variables declaration - do not modify
238 private javax.swing.JButton comprobarRespuestaJButton;
239 private javax.swing.JLabel ecuacionJLabel;
240 private javax.swing.JButton generarJButton;
241 private javax.swing.JComboBox nivelJComboBox;
242 private javax.swing.JLabel nivelJLabel;

```

Figura 28.23 | Aplicación tutor de matemáticas. (Parte 3 de 4).

```

243 private javax.swing.JComboBox operacionJComboBox;
244 private javax.swing.JLabel operacionJLabel;
245 private javax.swing.JLabel preguntaJLabel;
246 private javax.swing.JLabel respuestaJLabel;
247 private javax.swing.JTextField respuestaJTextField;
248 // Fin de variables declaration
249 } // fin de la clase ClienteGeneradorEcuacionesJFrame

```



Figura 28.23 | Aplicación tutor de matemáticas. (Parte 4 de 4).

28.9 Conclusión

En este capítulo se introdujeron los servicios Web JAX-WS 2.0, los cuales promueven la portabilidad y reutilización de software en aplicaciones que operan a través de Internet. Aprendió que un servicio Web es un componente de software almacenado en una computadora a la que una aplicación (u otro componente de software) puede acceder en otra computadora a través de una red, y se comunican a través de tecnologías como XML, SOAP y

HTTP. Hablamos sobre los diversos beneficios de este tipo de computación distribuida; por ejemplo, los clientes pueden acceder a los datos en equipos remotos, los clientes que no tengan el poder de procesamiento para realizar cálculos específicos pueden aprovechar los recursos de los equipos remotos, y pueden desarrollarse por completo tipos nuevos de aplicaciones innovadoras.

Explicamos cómo Netbeans, Sun Java Studio Creator 2 y las APIs de JAX-WS 2.0 facilitan la creación y el consumo de servicios Web. Le mostramos cómo establecer proyectos y archivos en estas herramientas, y cómo las herramientas administran la infraestructura del servicio Web necesaria para dar soporte a los servicios creados por el programador. Aprendió a definir los servicios Web y los métodos Web, así como a consumirlos desde aplicaciones Java de escritorio creadas en Netbeans, y también desde aplicaciones Web creadas en Sun Java Studio Creator 2. Después de explicar la mecánica de los servicios Web con nuestro ejemplo EnteroEnorme, demostramos servicios Web más sofisticados que utilizan rastreo de sesiones tanto del lado servidor como del lado cliente, y servicios Web que acceden a bases de datos mediante el uso de JDBC. También explicamos la serialización con XML y le mostramos cómo pasar objetos de tipos definidos por el usuario a los servicios Web, y cómo devolverlos de los servicios Web.

En el siguiente capítulo hablaremos acerca de cómo dar formato a la salida con el método `System.out.printf` y la clase `Formatter`.

28.10 Recursos Web

Además de los recursos Web que se muestran a continuación, también puede consultar los recursos Web relacionados con JSP que se proporcionan al final del capítulo 26.

www.deitel.com/WebServices/

Visite nuestro Centro de recursos de servicios Web para obtener información acerca de cómo diseñar e implementar servicios Web en muchos lenguajes, e información acerca de los servicios Web ofrecidos por compañías como Google, Amazon y eBay. También encontrará muchas herramientas adicionales de Java para publicar y consumir servicios Web.

www.deitel.com/java/

www.deitel.com/JavaSE6Mustang/

www.deitel.com/JavaEE5/

www.deitel.com/JavaCertification/

www.deitel.com/JavaDesignPatterns/

Nuestros Centros de recursos sobre Java proporcionan información específica de este lenguaje, como libros, documentos, artículos, diarios, sitios Web y blogs que abarcan una gran variedad de temas relacionados con Java (incluyendo los servicios Web de java).

www.deitel.com/ResourceCenters.html

De un vistazo a nuestra lista cada vez más extensa de Centros de recursos sobre programación, Web 2.0, software y demás temas interesantes.

java.sun.com/webservices/jaxws/index.jsp

El sitio oficial para la API de Sun Java para los Servicios Web de XML (JAX-WS). Incluye la API, documentación, tutoriales y demás vínculos de utilidad.

www.webservices.org

Ofrece noticias relacionadas con la industria, artículos y recursos para los servicios Web.

www-130.ibm.com/developerworks/webservices

El sitio de IBM para la arquitectura orientada al servicio (SOA) y los servicios Web incluye artículos, descargas, demos y foros de discusión relacionados con la tecnología de los servicios Web.

www.w3.org/TR/wsd1

Ofrece gran cantidad de documentación acerca de WSDL, incluyendo una explicación detallada de los servicios Web y las tecnologías relacionadas, como XML, SOAP, HTTP los tipos MIME en el contexto de WSDL.

www.w3.org/TR/soap

Ofrece gran cantidad de documentación acerca de los mensajes SOAP, el uso de SOAP con HTTP y cuestiones de seguridad relacionadas con SOAP.

www.ws-i.org

El sitio Web de la Organización de Interoperabilidad de Servicios Web proporciona información detallada sobre la creación de servicios Web basados en estándares que promuevan la interoperabilidad y una verdadera independencia de la plataforma.

webservices.xml.com/security

Artículos acerca de la seguridad de los servicios Web y los protocolos de seguridad estándar.

Servicios Web basados en REST

en.wikipedia.org/wiki/REST

Recurso de Wikipedia que explica la Transferencia representativa de estado (REST).

www.xfront.com/REST-Web-Services.html

Artículo titulado “Building Web Services the REST Way” (Cómo crear servicios Web al estilo REST).

www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

La disertación que propuso originalmente el concepto de los servicios basados en REST.

rest.blueoxen.net/cgi-bin/wiki.pl?ShortSummaryOfRest

Una breve introducción a REST (en inglés).

www.prescod.net/rest

Vínculos a muchos recursos sobre REST (en inglés).

Resumen

Sección 28.1 Introducción

- Un servicio Web es un componente de software almacenado en una computadora a la que se puede acceder mediante llamadas a métodos desde una aplicación (u otro componente de software) en otra computadora, a través de una red.
- Los servicios Web se comunican mediante el uso de tecnologías como XML y HTTP.
- El Protocolo simple de acceso a objetos (SOAP) es un protocolo basado en XML que permite la comunicación entre los clientes y los servicios Web, en forma independiente de la plataforma.
- Los servicios Web permiten a los comercios realizar transacciones a través de servicios Web estandarizados y ampliamente disponibles, en vez de depender de aplicaciones propietarias.
- Las compañías como Amazon, Google, eBay, PayPal y muchas otras están usando servicios Web para su beneficio, al hacer que sus aplicaciones del lado cliente estén disponibles para sus socios a través de los servicios Web.
- Al comprar servicios Web y utilizar la gran diversidad de servicios Web gratuitos, las compañías pueden invertir menos tiempo en desarrollar nuevas aplicaciones y pueden crear nuevas e innovadoras aplicaciones.
- Netbeans 5.5 y Sun Java Studio Creator 2 (ambos desarrollados por Sun) son dos de las diversas herramientas que permiten a los programadores “pubicar” y “consumir” servicios Web.

Sección 28.2 Fundamentos de los servicios Web de Java

- La computadora en la que reside un servicio Web se conoce como equipo remoto o servidor. Una aplicación cliente que accede a un servicio Web envía una llamada a un método a través de la red a un equipo remoto, el cual procesa la llamada y devuelve una respuesta a la aplicación, a través de la red.
- En Java, un servicio Web se implementa como una clase. La clase que representa el servicio Web reside en un servidor; no forma parte de la aplicación cliente.
- Al proceso de hacer que un servicio Web esté disponible para recibir peticiones de los clientes se le conoce como publicación de un servicio Web; al proceso de usar un servicio Web desde una aplicación cliente se le conoce como consumo de un servicio Web.
- Una aplicación que consume un servicio Web consiste de dos partes: un objeto de una clase proxy para interactuar con el servicio Web y una aplicación cliente que consume el servicio Web, al invocar métodos en el objeto proxy. El objeto proxy maneja los detalles relacionados con la comunicación con el servicio Web por el cliente.
- Las peticiones a (y las respuestas de) los servicios Web creados con JAX-WS 2.0 se transmiten comúnmente mediante SOAP. Cualquier cliente capaz de generar y procesar mensajes SOAP puede interactuar con un servicio Web, sin importar el lenguaje en el que esté escrito.

Sección 28.3.1 Creación de un proyecto de aplicación Web y cómo agregar una clase de servidor Web en Netbeans

- Al crear un servicio Web en Netbeans, nos enfocamos en la lógica del servicio Web y dejamos que el IDE se encargue de la infraestructura del servicio Web.
- Para crear un servicio Web en Netbeans, primero debemos crear un proyecto de tipo **aplicación Web (Web Application)**. Netbeans utiliza este tipo de proyecto para las aplicaciones Web que se ejecutan en clientes basados en navegador, y para los servicios Web invocados por otras aplicaciones.
- Al crear una **aplicación Web** en Netbeans, el IDE genera archivos adicionales que dan soporte a la aplicación Web.

Sección 28.3.2 Definición del servicio Web EnteroEnorme en Netbeans

- De manera predeterminada, cada nueva clase de servicio Web que se crea con las APIs de JAX-WS es un POJO (Objeto Java simple); no necesita extender una clase o implementar una interfaz para crear un servicio Web.
- Al compilar una clase que utiliza estas anotaciones de JAX-WS 2.0, el compilador crea el marco de trabajo de código compilado que permite al servicio Web esperar las peticiones de los clientes y responder a esas peticiones.
- La anotación `@WebService` indica que una clase representa a un servicio Web. El elemento opcional `name` especifica el nombre de la clase proxy que se generará para el cliente. El elemento opcional `serviceName` especifica el nombre de la clase que el cliente debe utilizar para obtener un objeto de la clase proxy.
- Netbeans coloca la anotación `@WebService` al principio de cada nueva clase de servicio Web que crea el programador. Se pueden agregar los elementos opcionales `name` y `serviceName` en los paréntesis de la anotación.
- Los métodos que se etiquetan con la anotación `@WebMethod` se pueden llamar en forma remota.
- Los métodos que no están etiquetados con `@WebMethod` no están accesibles para los clientes que consumen el servicio Web. Por lo general, éstos son métodos utilitarios dentro de la clase de servicio Web.
- La anotación `@WebMethod` tiene un elemento `operationName` opcional para especificar el nombre del método que está expuesto al cliente del servicio Web.
- Los parámetros de los métodos Web se anotan con la anotación `@WebParam`. El elemento opcional `name` indica el nombre del parámetro que está expuesto a los clientes del servicio Web.

Sección 28.3.3 Publicación del servicio Web EnteroEnorme desde Netbeans

- Netbeans se encarga de todos los detalles relacionados con la generación y despliegue de un servicio Web por nosotros. Esto incluye crear el marco de trabajo requerido para dar soporte al servicio Web.
- Para determinar si hay errores de compilación en el proyecto, haga clic con el botón derecho del ratón en el nombre del proyecto en la ficha **Projects** de Netbeans, y después seleccione la opción **Build Project**.
- Seleccione **Deploy Project** para desplegar el proyecto en el servidor Web que seleccionó durante la configuración de la aplicación.
- Seleccione **Run Project** para ejecutar la aplicación Web.
- Las opciones **Deploy Project** y **Run Project** también generan el proyecto si ha cambiado, e inician el servidor de aplicaciones en caso de que no se encuentre ya en ejecución.
- Para asegurar que se vuelvan a compilar todos los archivos de código fuente en un proyecto durante la siguiente operación de generación, puede usar las opciones **Clean Project** o **Clean and Build Project**.

Sección 28.3.4 Prueba del servicio Web EnteroEnorme con la página Web Tester de Sun Java System Application Server

- Sun Java System Application Server puede crear en forma dinámica una página Web llamada **Tester** para probar los métodos de un servicio Web desde un navegador Web. Para habilitar esta característica, use las opciones de ejecución (**Run**) del proyecto.
- Para mostrar la página Web **Tester**, ejecute la aplicación desde Netbeans o escriba el URL del servicio Web en el campo dirección del navegador, seguido de `?Tester`.
- Un cliente puede acceder a un servicio Web sólo cuando el servidor de aplicaciones se encuentra en ejecución. Si Netbeans inicia el servidor de aplicaciones por usted, éste se cerrará cuando cierre Netbeans. Para mantener el servidor de aplicaciones en ejecución, puede iniciarlo en forma independiente de Netbeans.

Sección 28.3.5 Descripción de un servicio Web con el Lenguaje de descripción de servicios Web (WSDL)

- Para consumir un servicio Web, un cliente debe saber en dónde encontrar el servicio Web, y debe recibir la descripción del mismo.
- JAX-WS utiliza el Lenguaje de descripción de servicios Web (WSDL): un vocabulario XML estándar para describir servicios Web de manera independiente de la plataforma.

- No necesita comprender el WSDL para aprovechar sus beneficios; el servidor genera un WSDL del servicio Web en forma dinámica por usted, y las herramientas cliente pueden analizar el WSDL para ayudarnos a crear la clase proxy del lado cliente que utiliza un cliente para acceder al servicio Web.
- Como el WSDL se crea en forma dinámica, los clientes siempre reciben la descripción más actualizada de un servicio Web desplegado.
- Para ver el WSDL de un servicio Web, escriba su URL en el campo dirección del navegador seguido de ?WSDL, o haga clic en el vínculo **WSDL File** en la página Web Tester de Sun Java System Application Server.

Sección 28.4 Cómo consumir un servicio Web

- El cliente de un servicio Web puede ser cualquier tipo de aplicación, o incluso otro servicio Web.
- En Netbeans, para habilitar una aplicación cliente de manera que pueda consumir un servicio Web, hay que agregar una referencia del servicio Web a la aplicación, la cual define la clase proxy del lado cliente.

Sección 28.4.1 Creación de un cliente para consumir el servicio Web EnteroEnorme

- Al agregar una referencia de servicio Web, el IDE crea y compila los artefactos del lado cliente: el marco de trabajo de código de Java que da soporte a la clase proxy del lado cliente.
- El cliente llama a los métodos en un objeto proxy, el cual usa los artefactos del lado cliente para interactuar con el servicio Web.
- Para agregar una referencia de servicio Web, haga clic con el botón derecho del ratón en el nombre del proyecto del cliente en la ficha **Projects** de Netbeans, y después seleccione **New > Web Service Client...** En el campo **WSDL URL** del cuadro de diálogo, especifique el URL del WSDL del servicio Web.
- Netbeans utiliza la descripción WSDL para generar la clase proxy y los artefactos del lado cliente.
- Al especificar el servicio Web que el programador desea consumir, Netbeans copia el WSDL del servicio Web en un archivo en el proyecto. Podemos ver este archivo desde la ficha **Files** de Netbeans, expandiendo los nodos en la carpeta `xml-resources` del proyecto.
- Los artefactos del lado cliente y la copia del cliente del archivo WSDL se pueden regenerar, haciendo clic con el botón derecho del ratón en el nodo del servicio Web en la ficha **Projects** de Netbean, y seleccionando **Refresh Client**.
- Para ver los artefactos del lado cliente generados por el IDE, seleccione la ficha **Files** de Netbeans y expanda la carpeta `build` del proyecto.

Sección 28.5 SOAP

- SOAP es un protocolo basado en XML, de uso común e independiente de la plataforma, que facilita las llamadas a procedimientos remotos, comúnmente a través de HTTP.
- El protocolo que transmite mensajes de petición y respuesta también se conoce como el formato de cable o protocolo de cable del servicio Web, ya que define la forma en que se envía la información “a lo largo del cable”.
- Cada petición y respuesta se empaquetan en un mensaje SOAP (también conocido como envoltura SOAP), el cual contiene la información que un servicio Web requiere para procesar el mensaje.
- El formato de cable utilizado para transmitir peticiones y respuestas debe tener soporte para todos los tipos que se pasen de una aplicación a otra. SOAP soporta los tipos primitivos y sus tipos de envoltura, así como `Date`, `Time` y otros. SOAP también puede transmitir arreglos y objetos de tipos definidos por el usuario.
- Cuando un programa invoca a un método Web, la petición y toda la información relevante se empaquetan en un mensaje SOAP y se envían al servidor en el que reside el servicio Web. El servicio Web procesa el contenido del mensaje SOAP; este mensaje especifica el método a invocar y sus argumentos. Una vez que el servicio Web recibe y analiza una petición, se hace una llamada al método apropiado y la respuesta se envía de vuelta al cliente, en otro mensaje SOAP. El proxy del lado cliente analiza la respuesta, que contiene el resultado de la llamada al método, y después devuelve el resultado a la aplicación cliente.
- Los mensajes SOAP se generan de manera automática por usted. Por lo tanto, no necesita comprender los detalles acerca de SOAP o XML para aprovechar estas tecnologías al publicar y consumir servicios Web.

Sección 28.6 Rastreo de sesiones en los servicios Web

- Puede ser benéfico para un servicio Web mantener la información de estado del cliente, con lo cual se elimina la necesidad de pasar la información del cliente entre éste y el servicio Web varias veces. Al almacenar la información de la sesión, el servicio Web puede también diferenciar a un cliente de otro.

Sección 28.6.1 Creación de un servicio Web Blackjack

- Para usar el rastreo de sesiones en un servicio Web, debemos incluir código para los recursos que mantienen la información de estado de la sesión. Anteriormente, había que escribir el código, que algunas veces era tedioso, para



crear estos recursos. Sin embargo, JAX-WS se encarga de esto por nosotros mediante la anotación `@Resource`. Esta anotación permite a herramientas como Netbeans “inyectar” el código complejo de soporte en nuestra clase, con lo cual nos podemos enfocar en la lógica de negocios, en vez de hacerlo en el código de soporte.

- Al uso de anotaciones para agregar código que dé soporte a nuestras clases se le conoce como inyección de dependencias. Las anotaciones como `@WebService`, `@WebMethod` y `@WebParam` también realizan la inyección de dependencias.
- Un objeto `WebServiceContext` permite a un servicio Web acceder a la información para una petición específica y darle mantenimiento, como el estado de la sesión. El código necesario que crea a un objeto `WebServiceContext` se inyecta en la clase mediante la anotación `@Resource`.
- El objeto `WebServiceContext` se utiliza para obtener un objeto `MessageContext`. Un servicio Web utiliza a un objeto `MessageContext` para obtener un objeto `HttpSession` para el cliente actual.
- El método `get` del objeto `MessageContext` se utiliza para obtener el objeto `HttpSession` para el cliente actual. El método `get` recibe una constante que indica lo que se debe obtener del objeto `MessageContext`. La constante `MessageContext.SERVLET_REQUEST` indica que deseamos obtener el objeto `HttpServletRequest` para el cliente actual. Después llamamos al método `getSession` para obtener el objeto `HttpSession` del objeto `HttpServletRequest`.
- El método `getAttribute` de `HttpSession` recibe un objeto `String` que identifica el objeto `Object` a obtener del estado de la sesión.


Sección 28.6.2 *Cómo consumir el servicio Web Blackjack*

- En el marco de trabajo JAX-WS 2.0, el cliente debe indicar si desea permitir al servicio Web mantener la información de la sesión. Para ello, primero convertimos el objeto proxy al tipo de interfaz `BindingProvider`. Un objeto `BindingProvider` permite al cliente manipular la información de petición que se enviará al servidor. Esta información se almacena en un objeto que implementa a la interfaz `RequestContext`. Los objetos `BindingProvider` y `RequestContext` son parte del marco de trabajo que crea el IDE cuando agregamos un cliente de servicio Web a la aplicación.
- A continuación, se invoca el método `getRequestContext` de `BindingProvider` para obtener el objeto `RequestContext`. Luego se hace una llamada al método `put` de `RequestContext` para establecer la propiedad `BindingProvider.SESSION_MAINTAIN_PROPERTY` a `true`, lo cual permite el rastreo de sesiones desde el lado cliente, de manera que el servicio Web sepa qué cliente está invocando a sus métodos.

Sección 28.7.1 *Configuración de Java DB en Netbeans y creación de la base de datos Reservacion*

- Para agregar un servidor de bases de datos Java DB en Netbeans, realice los siguientes pasos: Seleccione **Tools > Options...** para que aparezca el cuadro de diálogo **Options** de Netbeans. Haga clic en el botón **Advanced Options** para mostrar el cuadro de diálogo **Advanced Options**. En **IDE Configuration**, expanda el nodo **Server and External Tool Settings** y seleccione **Java DB Database**. Si las propiedades de Java DB no están ya configuradas, establezca la propiedad **Java DB Location** con la ubicación de Java DB en su sistema. Además, establezca la propiedad **Database Location** con la ubicación en donde desea que se almacenen las bases de datos Java DB.
- Para crear una nueva base de datos: seleccione **Tools > Java DB Databases > Create Java DB Database...** Escriba el nombre de la base de datos a crear, un nombre de usuario y contraseña, y después haga clic en **OK** para crear la base de datos.
- Puede utilizar la ficha **Runtime** de Netbeans para crear tablas y ejecutar instrucciones SQL que llenen la base de datos con información. Haga clic en la ficha **Runtime** de Netbeans y expanda el nodo **Databases**.
- Netbeans debe estar conectado a la base de datos para ejecutar instrucciones SQL. Si Netbeans no está conectado a la base de datos, aparecerá el ícono  enseguida del URL de la base de datos. En este caso, haga clic con el botón derecho del ratón en el ícono y seleccione **Connect...** Una vez conectado, el ícono cambiará a .

Sección 28.7.2 *Creación de una aplicación Web para interactuar con el servicio Web Reservacion*

- Para agregar un servicio Web a una aplicación Web en Java Studio Creator 2, haga clic en el botón **Agregar servicio Web...** (). Después podrá especificar el WSDL del servicio Web en el cuadro de diálogo que aparezca.

Sección 28.8 *Cómo pasar un objeto de un tipo definido por el usuario a un servicio Web*

- Los servicios Web pueden recibir y devolver objetos de tipos definidos por el usuario; a éstos se les conoce como tipos personalizados.
- Los tipos personalizados que se envían o reciben de un servicio Web mediante el uso de SOAP se serializan en formato XML. A este proceso se le conoce como serialización XML y se maneja de manera automática, sin necesidad de que el programador intervenga.

- Una clase que se utiliza para especificar tipos de parámetros o de valores de retorno en los métodos Web debe proporcionar un constructor predeterminado `public` o sin argumentos. Además, las variables de instancia que deban deserializarse deben tener métodos `set` y `get` `public`, o las variables de instancia se deben declarar como `public`.
- Cualquier variable de instancia que no se serialice, simplemente recibe su valor predeterminado (o el valor proporcionado por el constructor sin argumentos) a la hora de deserializar un objeto de la clase.

Terminología

@Resource, anotación

@WebMethod, anotación

@WebParam, anotación

@WebService, anotación

AbstractPageBean, clase

Add Server Instance, cuadro de diálogo

agregar una referencia de servicio Web a un proyecto en Netbeans

analizar un mensaje SOAP

artefactos del lado cliente

artefactos del lado servidor

BEA Weblogic Server

BindingProvider, interfaz

Build Project, opción en Netbeans

clase proxy para un servicio Web

Clean and Build Project, opción en Netbeans

Clean Project, opción en Netbeans

consumir un servicio Web

Deploy Project, opción en Netbeans

desplegar un servicio Web

envoltura SOAP

equipo remoto

formato de cable

get, método de la interfaz MessageContext

getRequestContext, método de la interfaz Binding-Provider

IDE Netbeans 5.5

inyección de dependencias

JAX-WS 2.0

JBoss Application Server

Lenguaje de descripción de servicios Web (WSDL)

mensaje SOAP

MessageContext, interfaz

name, elemento de la anotación @WebParam

name, elemento de la anotación @WebService

New Project, cuadro de diálogo en Netbeans

New Web Service Client, cuadro de diálogo en Netbeans

New Web Service, cuadro de diálogo en Netbeans

objeto proxy maneja los detalles de la comunicación con el servicio Web

operationName, elemento de la anotación @WebMethod

Organización de Interoperabilidad de Servicios Web (WS-I)

POJO (Objeto Java simple)

probar un servicio Web

Project Properties, cuadro de diálogo en Netbeans

protocolo de cable

publicar un servicio Web

put, método de la interfaz RequestContext

rastreo de sesiones en servicios Web

referencia de servicio Web

RequestContext, interfaz

REST (Transferencia representativa de estado)

Run Project, opción en Netbeans

serialización en XML

Server Manager, cuadro de diálogo en Netbeans

serviceName, elemento de la anotación @WebService

servidor Apache Tomcat

servidor de aplicaciones

servidor GlassFish

SOAP (Protocolo simple de acceso a objetos)

Sun Java Studio Creator 2

Sun Java System Application Server

Tester, página Web de Sun Java System Application Server

tipo personalizado

transacciones B2B (negocio a negocio)

transacciones de negocio a negocio (B2B)

Transferencia representativa de estado (REST)

Web Application, proyecto en Netbeans

WebServiceContext, interfaz

WS-I Basic Profile 1.1 (BP 1.1)

Ejercicios de autoevaluación

- 28.1** Conteste con *verdadero* o *falso* a cada una de las siguientes proposiciones; en caso de ser *falso*, explique por qué.
- Todos los métodos de una clase de servicio Web pueden ser invocados por los clientes de ese servicio Web.
 - Al consumir un servicio Web en una aplicación cliente creada en Netbeans, debemos crear la clase proxy que permita al cliente comunicarse con el servicio Web.
 - Una clase proxy que se comunica con un servicio Web generalmente usa SOAP para enviar y recibir mensajes.
 - El rastreo de sesiones está habilitado de manera automática en un cliente de un servicio Web.
 - Los métodos Web no se pueden declarar como `static`.

- f) Un tipo definido por el usuario que se utilice en un servicio Web debe definir métodos *get* y *set* para cualquier propiedad que se vaya a serializar.

28.2 Complete los siguientes enunciados.

- Cuando se envían mensajes entre una aplicación y un servicio Web mediante SOAP, cada mensaje se coloca en un(a) _____.
- Un servicio Web en java es un _____; no necesita implementar ninguna interfaz o extender una clase.
- Las peticiones a los servicios Web se transportan generalmente a través de Internet, mediante el protocolo _____.
- Para establecer el nombre expuesto de un método Web, utilice el elemento _____ de la anotación `@WebMethod`.
- La _____ transforma a un objeto en un formato que se pueda enviar entre un servicio Web y un cliente.

Respuestas a los ejercicios de autoevaluación

28.1 a) Falso. Sólo los métodos declarados con la anotación `@WebMethod` pueden ser invocados por los clientes del servicio Web. b) Falso. Netbeans crea la clase proxy cuando agregamos un cliente de servicio Web a la aplicación. c) Verdadero. d) Falso. En el marco de trabajo JAX-WS 2.0, el cliente debe indicar si desea permitir que el servicio Web mantenga información sobre la sesión. Primero hay que convertir el objeto proxy al tipo de la interfaz `BindingProvider`, y después hay que usar el método `getRequestContext` de `BindingProvider` para obtener el objeto `RequestContext`. Por último, hay que usar el método `put` del objeto `RequestContext` para establecer la propiedad `BindingProvider.SESSION_MAINTAIN_PROPERTY` a `true`. e) Verdadero. f) Verdadero.

- 28.2** a) mensaje SOAP o envoltura SOAP.
 b) POJO (Objeto Java simple).
 c) HTTP.
 d) `operationName`.
 e) Serialización en XML.

Ejercicios

28.3 (*Servicio Web Libreta Telefónica*) Cree un servicio Web que almacene las entradas en una libreta telefónica, en la base de datos `LibretaDireccionesBD`, y una aplicación cliente Web que consuma este servicio. Use los pasos de la sección 28.7.1 para crear la base de datos `LibretaTelefonica`. Esta base sólo debe contener una tabla (`LibretaDirecciones`) con tres columnas: `ApellidoPaterno`, `PrimerNombre` y `NumeroTelefonico`, cada una de tipo `VARCHAR`. Las columnas `ApellidoPaterno` y `PrimerNombre` deben almacenar hasta 30 caracteres. La columna `NumeroTelefonico` debe soportar números telefónicos de la forma (800) 555-1212, que contienen 14 caracteres.

Dé al usuario cliente la capacidad de escribir un nuevo contacto (método Web `agregarEntrada`) y buscar contactos por apellido paterno (método Web `getEntradas`). Pase sólo objetos `String` como argumentos para el servicio Web. El método Web `getEntradas` debe devolver un arreglo de objetos `String` que contenga las entradas que coinciden en la libreta de direcciones. Cada objeto `String` en el arreglo debe consistir en el apellido paterno, primer nombre y número telefónico para una entrada en la libreta de direcciones. Estos valores deben separarse mediante comas.

La consulta `SELECT` para buscar una entrada en `LibretaDirecciones` por apellido paterno debería ser:

```
SELECT ApellidoPaterno, PrimerNombre, NumeroTelefonico
FROM LibretaDirecciones
WHERE (ApellidoPaterno = ApellidoPaterno)
```

La instrucción `INSERT` para insertar una nueva entrada en la base de datos `LibretaDirecciones` debe ser:

```
INSERT INTO LibretaDirecciones (ApellidoPaterno, PrimerNombre, NumeroTelefonico)
VALUES (ApellidoPaterno, PrimerNombre, NumeroTelefonico)
```

28.4 (*Modificación al servicio Web Libreta telefónica*) Modifique el ejercicio 28.3, de manera que utilice una clase llamada `EntradaLibretaTelefonica` para representar a una fila en la base de datos. Al agregar contactos, la aplicación cliente debe proporcionar objetos de tipo `EntradaLibretaDirecciones` al servicio Web, y debe recibir objetos de tipo `EntradaLibretaDirecciones` a la hora de buscar contactos.

28.5 (*Modificación al servicio Web Blackjack*) Modifique el ejemplo del servicio Web Blackjack en la sección 28.6 para incluir la clase Carta. Modifique el método Web repartirCarta, de manera que devuelva un objeto de tipo Carta, y modifique el método Web obtenerValorMano, de manera que reciba un arreglo de objetos Carta del cliente. Modifique además la aplicación cliente para llevar la cuenta de qué cartas se han repartido, mediante el uso de objetos ArrayList de objetos Carta. La clase proxy creada por Netbean considerará el parámetro del arreglo de un método Web como un objeto List, de manera que podemos pasar estos objetos ArrayList de objetos Carta directamente al método obtenerValorMano. Su clase Carta deberá incluir métodos *set* y *get* para la cara y el palo de la carta.

—Adolph S. Ochs

—John Keats

—Amenehope

OBJETIVOS

En este capítulo aprenderá a:

- Trabajar con los flujos de entrada y salida.
- Usar el formato `printf`.
- Imprimir con anchuras de campo y precisiones.
- Usar banderas de formato en la cadena de formato de `printf`.
- Imprimir con un índice como argumento.
- Imprimir literales y secuencias de escape.
- Dar formato a la salida con la clase `Formatter`.

- 29.1 Introducción
- 29.2 Flujos
- 29.3 Aplicación de formato a la salida con `printf`
- 29.4 Impresión de enteros
- 29.5 Impresión de números de punto flotante
- 29.6 Impresión de cadenas y caracteres
- 29.7 Impresión de fechas y horas
- 29.8 Otros caracteres de conversión
- 29.9 Impresión con anchuras de campo y precisiones
- 29.10 Uso de banderas en la cadena de formato de `printf`
- 29.11 Impresión con índices como argumentos
- 29.12 Impresión de literales y secuencias de escape
- 29.13 Aplicación de formato a la salida con la clase `Formatter`
- 29.14 Conclusión

Resumen | Terminología | Ejercicios de autoevaluación | Respuestas a los ejercicios de autoevaluación | Ejercicios

29.1 Introducción

Una parte importante de la solución a cualquier problema es la presentación de los resultados. En este capítulo, hablaremos sobre las características de formato del método `printf` y la clase `Formatter` (paquete `java.util`). El método `printf` aplica formato a los datos y los imprime en el flujo de salida estándar (`System.out`). La clase `Formatter` aplica formato a los datos y los imprime en un destino especificado, como una cadena o un flujo de salida de archivo.

Anteriormente en este libro hablamos sobre muchas de las características de `printf`. En este capítulo se sintetizan esas características y se presentan otras, como mostrar datos de fecha y hora en varios formatos, reordenar la salida con base en el índice del argumento, y mostrar números y cadenas con varias banderas.

29.2 Flujos

Por lo general, las operaciones de entrada y salida se llevan a cabo con flujos, los cuales son secuencias de bytes. En las operaciones de entrada, los bytes fluyen de un dispositivo (como un teclado, una unidad de disco, una conexión de red) a la memoria principal. En las operaciones de salida, los bytes fluyen de la memoria principal a un dispositivo (como una pantalla, una impresora, una unidad de disco, una conexión de red).

Cuando empieza la ejecución de un programa, tres flujos se conectan a éste de manera automática. Por lo común, el flujo de entrada estándar se conecta al teclado, y el flujo de salida estándar se conecta a la pantalla. Un tercer flujo, el **flujo de error estándar** (`System.err`), se conecta generalmente a la pantalla y se utiliza para imprimir mensajes de error en ésta, de manera que puedan verse de inmediato; aún y cuando el flujo de salida estándar esté escribiendo en un archivo. Generalmente, los sistemas operativos permiten redirigir estos flujos a otros dispositivos. En el capítulo 14, Archivos y flujos, y en el capítulo 24, Redes, se describen los flujos con detalle.

29.3 Aplicación de formato a la salida con `printf`

Con `printf` podemos lograr un formato preciso en la salida. [*Nota:* Java SE 5 tomó prestada esta característica del lenguaje de programación C]. El método `printf` cuenta con las siguientes herramientas de formato, cada una de las cuales se verá en este capítulo:

1. Redondeo de valores de punto flotante a un número indicado de posiciones decimales.
2. Alineación de una columna de números con puntos decimales, que aparezcan uno encima de otro.
3. Justificación a la derecha y justificación a la izquierda de los resultados.

4. Inserción de caracteres literales en posiciones precisas de una línea de salida.
5. Representación de números de punto flotante en formato exponencial.
6. Representación de enteros en formato octal y hexadecimal (vea el apéndice E, Sistemas numéricos, para obtener más información acerca de los valores octales y hexadecimales).
7. Visualización de todo tipo de datos con anchuras de campo de tamaño fijo y precisiones.
8. Visualización de fechas y horas en diversos formatos.

Cada llamada a `printf` proporciona como primer argumento una **cadena de formato**, la cual describe el formato de salida. La cadena de formato puede consistir en **texto fijo** y **especificadores de formato**. El texto fijo se imprime mediante `printf` de igual forma que como se imprimiría mediante los métodos `print` o `println` de `System.out`. Cada especificador de formato es un receptáculo para un valor y especifica el tipo de datos a imprimir. Los especificadores de formato también pueden incluir información de formato opcional.

En su forma más simple, cada especificador de formato empieza con un signo de porcentaje (%) y va seguido de un **carácter de conversión** que representa el tipo de datos del valor a imprimir. Por ejemplo, el especificador de formato `%s` es un receptáculo para una cadena, y el especificador de formato `%d` es un receptáculo para un valor `int`. La información de formato opcional se especifica entre el signo de porcentaje y el carácter de conversión. La información de formato opcional incluye un índice como argumento, banderas, anchura de campo y precisión. A lo largo de este capítulo, definiremos cada uno de estos elementos, y mostraremos ejemplos.

29.4 Impresión de enteros

Un entero es un número completo, como 776, 0 o -52, que no contiene punto decimal. Los valores enteros se muestran en uno de varios formatos. En la figura 29.1 se describen los **caracteres de conversión integrales**.

En la figura 29.2 se imprime un entero usando cada una de las conversiones integrales. En las líneas 9 a 10, observe que el signo positivo no se muestra de manera predeterminada, pero el signo negativo sí. Más adelante en este capítulo (figura 29.14) veremos cómo forzar a que se impriman los signos positivos.

El método `printf` tiene la forma

```
printf( cadena-de-formato, lista-de-argumentos );
```

en donde *cadena-de-formato* describe el formato de salida, y *lista-de-argumentos* contiene los valores que corresponden a cada especificador de formato en *cadena-de-formato*. Puede haber muchos especificadores de formato en una cadena de formato.

Cada cadena de formato en las líneas 8 a 10 especifica que `printf` debe imprimir un entero decimal (`%d`) seguido de un carácter de nueva línea. En la posición del especificador de formato, `printf` sustituye el valor del primer argumento después de la cadena de formato. Si la cadena de formato contiene varios especificadores de formato, en cada posición del siguiente especificador de formato, `printf` sustituirá el valor del siguiente argumento en la lista de argumentos. El especificador de formato `%o` en la línea 11 imprime el entero en formato octal. El especificador de formato `%x` en la línea 12 imprime el entero en formato hexadecimal. El especificador de formato `%X` en la línea 13 imprime el entero en formato hexadecimal, con letras mayúsculas.

Carácter de conversión	Descripción
d	Muestra un entero decimal (base 10).
o	Muestra un entero octal (base 8).
x o X	Muestra un entero hexadecimal (base 16). X hace que se muestren los dígitos del 0 al 9 y la letras A a la F, y x hace que se muestren los dígitos del 0 al 9 y las letras de la a a la f.

Figura 29.1 | Caracteres de conversión de enteros.

```

1 // Fig. 29.2: PruebaConversionEnteros.java
2 // Uso de los caracteres de conversión integrales.
3
4 public class PruebaConversionEnteros
5 {
6     public static void main( String args[] )
7     {
8         System.out.printf( "%d\n", 26 );
9         System.out.printf( "%d\n", +26 );
10        System.out.printf( "%d\n", -26 );
11        System.out.printf( "%o\n", 26 );
12        System.out.printf( "%x\n", 26 );
13        System.out.printf( "%X\n", 26 );
14    } // fin de main
15 } // fin de la clase PruebaConversionEnteros

```

```

26
26
-26
32
1a
1A

```

Figura 29.2 | Uso de caracteres de conversión de enteros.

29.5 Impresión de números de punto flotante

Un valor de punto flotante contiene un punto decimal, como en 33.5, 0.0 o -657.983. Los valores de punto flotante se muestran en uno de varios formatos. En la figura 29.3 se describen las conversiones de punto flotante. Los **caracteres de conversión e y E** muestran valores de punto flotante en **notación científica computarizada** (también conocida como **notación exponencial**). La notación exponencial es el equivalente computacional de la notación científica que se utiliza en las matemáticas. Por ejemplo, el valor 150.4582 se representa en notación científica matemática de la siguiente manera:

$$1.504582 \times 10^2$$

y se representa en notación exponencial como

$$1.504582e+02$$

en Java. Esta notación indica que 1.504582 se multiplica por 10 elevado a la segunda potencia (e+02). La **e** representa al “exponente”.

Carácter de conversión	Descripción
e o E	Muestra un valor de punto flotante en notación exponencial. Cuando se utiliza el carácter de conversión E, la salida se muestra en letras mayúsculas.
f	Muestra un valor de punto flotante en formato decimal.
g o G	Muestra un valor de punto flotante en el formato de punto flotante f o en el formato exponencial e, con base en la magnitud del valor. Si la magnitud es menor que 10^{-3} , o si es mayor o igual que 10^7 , el valor de punto flotante se imprime con e (o E). En cualquier otro caso, el valor se imprime en el formato f. Cuando se utiliza el carácter de conversión G, la salida se muestra en letras mayúsculas.
a o A	Muestra un número de punto flotante en formato hexadecimal. Cuando se usa el carácter de conversión A, la salida se muestra en letras mayúsculas.

Figura 29.3 | Caracteres de conversión de punto flotante.

Los valores que se imprimen con los caracteres de conversión `e`, `E` y `f` se muestran con seis dígitos de precisión en el lado derecho del punto decimal de manera predeterminada (por ejemplo, 1.045921); otras precisiones se deben especificar de manera explícita. Para los valores impresos con el carácter de conversión `g`, la precisión representa el número total de dígitos mostrados, excluyendo el exponente. El valor predeterminado es de seis dígitos (por ejemplo, 12345678.9 se muestra como 1.23457e+07). El **carácter de conversión `f`** siempre imprime por lo menos un dígito a la izquierda del punto decimal. Los caracteres de conversión `e` y `E` imprimen una `e` minúscula y una `E` mayúscula antes del exponente, y siempre imprimen sólo un dígito a la izquierda del punto decimal. El redondeo ocurre si el valor al que se está dando formato tiene más dígitos significativos que la precisión.

El **carácter de conversión `g`** (o `G`) imprime en formato `e` (`E`) o `f`, dependiendo del valor de punto flotante. Por ejemplo, los valores 0.0000875, 87500000.0, 8.75, 87.50 y 875.0 se imprimen como 8.750000e-05, 8.750000e+07, 8.750000, 87.500000 y 875.000000 con el carácter de conversión `g`. El valor 0.0000875 utiliza la notación `e` ya que la magnitud es menor que 10^{-3} . El valor 87500000.0 utiliza la notación `e`, debido a que la magnitud es mayor que 10^7 . En la figura 29.4 se muestra cada uno de los caracteres de conversión de punto flotante.

```

1 // Fig. 29.4: PruebaPuntoFlotante.java
2 // Uso de los caracteres de conversión de punto flotante.
3
4 public class PruebaPuntoFlotante
5 {
6     public static void main( String args[] )
7     {
8         System.out.printf( "%e\n", 12345678.9 );
9         System.out.printf( "%e\n", +12345678.9 );
10        System.out.printf( "%e\n", -12345678.9 );
11        System.out.printf( "%E\n", 12345678.9 );
12        System.out.printf( "%f\n", 12345678.9 );
13        System.out.printf( "%g\n", 12345678.9 );
14        System.out.printf( "%G\n", 12345678.9 );
15    } // fin de main
16 } // fin de la clase PruebaPuntoFlotante

```

```

1.234568e+07
1.234568e+07
-1.234568e+07
1.234568E+07
12345678.900000
1.23457e+07
1.23457E+07

```

Figura 29.4 | Uso de los caracteres de conversión de punto flotante.

29.6 Impresión de cadenas y caracteres

Los caracteres de conversión `c` y `s` se utilizan para imprimir caracteres individuales y cadenas, respectivamente. El carácter de conversión `s` también puede imprimir objetos con los resultados de las llamadas implícitas al método `toString`. Los **caracteres de conversión `c`** y `C` requieren un argumento `char`. Los **caracteres de conversión `s`** y `S` pueden recibir un objeto `String` o cualquier objeto `Object` (se incluyen todas las subclases de `Object`) como argumento. Cuando se pasa un objeto al carácter de conversión `s`, el programa utiliza de manera implícita el método `toString` del objeto para obtener la representación `String` del objeto. Cuando se utilizan los caracteres de conversión `C` y `S`, la salida se muestra en letras mayúsculas. El programa que se muestra en la figura 29.5 imprime en pantalla caracteres, cadenas y objetos con los caracteres de conversión `c` y `s`. Observe que se realiza una conversión `autoboxing` en la línea 10, cuando se asigna una constante `int` a un objeto `Integer`. En la línea 15 se asocia un objeto `Integer` como argumento para el carácter de conversión `s`, el cual invoca de manera implícita al método `toString` para obtener el valor entero. Observe que también puede imprimir en pantalla un objeto `Integer` mediante el uso del especificador de formato `%d`. En este caso, se realizará una conversión `unboxing` con el valor `int` en el objeto `Integer` y se imprimirá en pantalla.

```

1 // Fig. 29.5: ConversionCadenasChar.java
2 // Uso de los caracteres de conversión de cadenas y caracteres.
3
4 public class ConversionCadenasChar
5 {
6     public static void main( String args[] )
7     {
8         char caracter= 'A'; // inicializa el char
9         String cadena = "Esta tambien es una cadena"; // objeto String
10        Integer entero = 1234; // inicializa el entero (autoboxing)
11
12        System.out.printf( "%c\n", caracter );
13        System.out.printf( "%s\n", "Esta es una cadena" );
14        System.out.printf( "%s\n", cadena );
15        System.out.printf( "%S\n", cadena );
16        System.out.printf( "%s\n", entero ); // llamada implícita a toString
17    } // fin de main
18 } // fin de la clase ConversionCadenasChar

```

```

A
Esta es una cadena
Esta tambien es una cadena
ESTA TAMBIEN ES UNA CADENA
1234

```

Figura 29.5 | Uso de los caracteres de conversión de cadenas y caracteres.



Error común de programación 29.1

El uso de %c para imprimir una cadena produce una excepción `IllegalFormatConversionException`; una cadena no se puede convertir en un carácter.

29.7 Impresión de fechas y horas

Con el **carácter de conversión** `t` o `T`, podemos imprimir fechas y horas en diversos formatos. El carácter de conversión `t` o `T` siempre va seguido de un **carácter de sufijo de conversión** que especifica el formato de fecha y/o de hora. Cuando se utiliza el carácter de conversión `T`, la salida se muestra en letras mayúsculas. En la figura 29.6 se listan los caracteres de sufijo de conversión comunes para aplicar formato a las **composiciones de fecha y hora** que muestran tanto la fecha como la hora. En la figura 29.7 se listan los caracteres de sufijo de conversión comunes para aplicar formato a las fechas. En la figura 29.8 se listan los caracteres de sufijo de conversión comunes para aplicar formato a las horas. Para ver la lista completa de caracteres de sufijo de conversión, visite el sitio Web.java.sun.com/javase/6/docs/api/java/util/Formatter.html.

Carácter de sufijo de conversión	Descripción
<code>c</code>	Muestra la fecha y hora con el formato <p>día mes fecha hora:minuto:segundo zona-horaria año</p> <p>con tres caracteres para día y mes, dos dígitos para fecha, hora, minuto y segundo, y cuatro dígitos para año; por ejemplo, Mié Mar 03 16:30:25 GMT -05:00 2004. Se utiliza el reloj de 24 horas. En este ejemplo, GMT -05:00 es la zona horaria.</p>
<code>F</code>	Muestra la fecha con el formato año-mes-día con cuatro dígitos para el año y dos dígitos para el mes y la fecha (por ejemplo, 2004-05-04).

Figura 29.6 | Caracteres de sufijo de conversión de composiciones de fecha y hora. (Parte I de 2).

Carácter de sufijo de conversión	Descripción
D	Muestra la fecha con el formato mes/día/año, con dos dígitos para el mes, día y año (por ejemplo, 03/03/04).
r	Muestra la hora con el formato hora:minuto:segundo AM PM, con dos dígitos para la hora, minuto y segundo (por ejemplo, 04:30:25 PM). Se utiliza el reloj de 12 horas.
R	Muestra la hora con el formato hora:minuto, con dos dígitos para la hora y el minuto (por ejemplo, 16:30). Se utiliza el reloj de 24 horas.
T	Muestra la hora con el formato hora:minuto:segundo, con dos dígitos para la hora, minuto y segundo (por ejemplo, 16:30:25). Se utiliza el reloj de 24 horas.

Figura 29.6 | Caracteres de sufijo de conversión de composiciones de fecha y hora. (Parte 2 de 2).

Carácter de sufijo de conversión	Descripción
A	Muestra el nombre completo del día de la semana (por ejemplo, Miércoles).
a	Muestra el nombre corto de tres caracteres del día de la semana (por ejemplo, Mié).
B	Muestra el nombre completo del mes (por ejemplo, Marzo).
b	Muestra el nombre corto de tres caracteres del mes (por ejemplo, Mar).
d	Muestra el día del mes con dos dígitos, rellenando con ceros a la izquierda si es necesario (por ejemplo, 03).
m	Muestra el mes con dos dígitos, rellenando con ceros a la izquierda si es necesario (por ejemplo, 07).
e	Muestra el día del mes sin ceros a la izquierda (por ejemplo, 3).
Y	Muestra el año con cuatro dígitos (por ejemplo, 2004).
y	Muestra los dos últimos dígitos del año con ceros a la izquierda, según sea necesario (por ejemplo, 04).
j	Muestra el día del año con tres dígitos, rellenando con ceros a la izquierda según sea necesario (por ejemplo, 016).

Figura 29.7 | Caracteres de sufijo de conversión para aplicar formato a las fechas.

Carácter de sufijo de conversión	Descripción
H	Muestra la hora en el reloj de 24 horas, con un cero a la izquierda si es necesario (por ejemplo, 16).
I	Muestra la hora en el reloj de 12 horas, con un cero a la izquierda si es necesario (por ejemplo, 04).
k	Muestra la hora en el reloj de 24 horas sin ceros a la izquierda (por ejemplo, 16).
l	Muestra la hora en el reloj de 12 horas sin ceros a la izquierda (por ejemplo, 4).

Figura 29.8 | Caracteres de sufijo de conversión para aplicar formato a las horas. (Parte 1 de 2).

Carácter de sufijo de conversión	Descripción
M	Muestra los minutos con un cero a la izquierda, si es necesario (por ejemplo, 06).
S	Muestra los segundos con un cero a la izquierda, si es necesario (por ejemplo, 05).
Z	Muestra la abreviación para la zona horaria (por ejemplo, GMT -05:00, que representa a la Hora estándar occidental, la cual se encuentra a 5 horas de retraso de la Hora del Meridiano de Greenwich).
p	Muestra las iniciales de mañana o tarde en minúsculas (por ejemplo, pm).
P	Muestra el marcador de mañana o tarde en mayúsculas (por ejemplo, PM).

Figura 29.8 | Caracteres de sufijo de conversión para aplicar formato a las horas. (Parte 2 de 2).

En la figura 29.9 se utiliza el carácter de conversión `t` con los caracteres de sufijo de conversión para mostrar fechas y horas en diversos formatos. El carácter de conversión `t` requiere que su correspondiente argumento sea de tipo `long`, `Long`, `Calendar` o `Date` (ambas clases se encuentran en el paquete `java.util`); los objetos de cada una de estas clases pueden representar fechas y horas. La clase `Calendar` es la preferida para este propósito, ya que ciertos constructores y métodos en la clase `Date` se sustituyen por los de la clase `Calendar`. En la línea 10 se invoca el método `static getInstance` de `Calendar` para obtener un calendario con la fecha y hora actuales. En las líneas 13 a 17, 20 a 22 y 25 a 26 se utiliza este objeto `Calendar` en instrucciones `printf` como el valor al que se aplicará formato con el carácter de conversión `t`. Observe que en las líneas 20 a 22 y 25 a 26 se utiliza el **índice como argumento** opcional ("`1$`") para indicar que todos los especificadores de formato en la cadena de formato utilizan el primer argumento después de la cadena de formato en la lista de argumentos. En la sección 29.11 aprenderá más acerca de los índices como argumentos. Al usar el índice como argumento, se elimina la necesidad de listar repetidas veces el mismo argumento.

```

1 // Fig. 29.9: PruebaFechaHora.java
2 // Aplicación de formato a fechas y horas con los caracteres de conversión t y T.
3 import java.util.Calendar;
4
5 public class PruebaFechaHora
6 {
7     public static void main( String args[] )
8     {
9         // obtiene la fecha y hora actuales
10        Calendar fechaHora = Calendar.getInstance();
11
12        // impresión con caracteres de conversión para composiciones de fecha/hora
13        System.out.printf( "%tc\n", fechaHora );
14        System.out.printf( "%tF\n", fechaHora );
15        System.out.printf( "%tD\n", fechaHora );
16        System.out.printf( "%tr\n", fechaHora );
17        System.out.printf( "%tT\n", fechaHora );
18
19        // impresión con caracteres de conversión para fechas
20        System.out.printf( "%1$tA, %1$tB %1$td, %1$tY\n", fechaHora );
21        System.out.printf( "%1$TA, %1$TB %1$Td, %1$TY\n", fechaHora );
22        System.out.printf( "%1$ta, %1$tb %1$te, %1$ty\n", fechaHora );
23
24        // impresión con caracteres de conversión para horas
25        System.out.printf( "%1$tH:%1$tM:%1$tS\n", fechaHora );

```

Figura 29.9 | Aplicación de formato a fechas y horas con los caracteres de conversión `t`. (Parte 1 de 2).

```
26      System.out.printf( "%1$tZ %1$tI:%1$tM:%1$tS %Tp", fechaHora );
27      } // fin de main
28  } // fin de la clase PruebaFechaHora
```

```
mié nov 07 11:54:30 CST 2007
2007-11-07
11/07/07
11:54:30 AM
11:54:30
miércoles, noviembre 07, 2007
MIÉRCOLES, NOVIEMBRE 07, 2007
mié, nov 7, 07
11:54:30
CST 11:54:30 AM
```

Figura 29.9 | Aplicación de formato a fechas y horas con los caracteres de conversión t. (Parte 2 de 2).

29.8 Otros caracteres de conversión

El resto de los caracteres de conversión son **b**, **B**, **h**, **H**, **%** y **n**. Éstos se describen en la figura 29.10.

En las líneas 9 y 10 de la figura 29.11 se utiliza **%b** para imprimir el valor de los valores booleanos `false` y `true`. En la línea 11 se asocia un objeto `String` a **%b**, el cual devuelve `true` debido a que no es `null`. En la línea 12 se asocia un objeto `null` a **%B**, el cual muestra `FALSE` ya que prueba es `null`. En las líneas 13 y 14 se utiliza **%h** para imprimir las representaciones de cadena de los valores de código hash para las cadenas "hola" y "Hola". Estos valores se podrían utilizar para almacenar o colocar las cadenas en un objeto `Hashtable` o `HashMap` (los cuales vimos en el capítulo 19, Colecciones). Observe que los valores de código hash para estas dos cadenas difieren, ya que una cadena empieza con letra minúscula y la otra con letra mayúscula. En la línea 15 se utiliza **%H** para imprimir `null` en letras mayúsculas. Las últimas dos instrucciones `printf` (líneas 16 y 17) utilizan **%%** para imprimir el carácter `%` en una cadena, y **%n** para imprimir un separador de línea específico de la plataforma.



Error común de programación 29.2

Tratar de imprimir un carácter de porcentaje literal mediante el uso de `%` en vez de `%%` en la cadena de formato podría provocar un error lógico difícil de detectar. Cuando aparece el `%` en una cadena de formato, debe ir seguido de un carácter de conversión en la cadena. El signo de por ciento individual podría ir seguido accidentalmente de un carácter de conversión legítimo, con lo cual se produciría un error lógico.

Carácter de conversión	Descripción
b o B	Imprime "true" o "false" para el valor de un boolean o Boolean. Estos caracteres de conversión también pueden aplicar formato al valor de cualquier referencia. Si la referencia no es null, se imprime "true"; en caso contrario, se imprime "false". Cuando se utiliza el carácter de conversión B, la salida se muestra en letras mayúsculas.
h o H	Imprime la representación de cadena del valor de código hash de un objeto en formato hexadecimal. Si el correspondiente argumento es null, se imprime "null". Cuando se utiliza el carácter de conversión H, la salida se muestra en letras mayúsculas.
%	Imprime el carácter de por ciento.
n	Imprime el separador de línea específico de la plataforma (por ejemplo, \r\n en Windows o \n en UNIX/LINUX).

Figura 29.10 | Otros especificadores de conversión.

```

1 // Fig. 29.11: OtrasConversiones.java
2 // Uso de los caracteres de conversión b, B, h, H, % y n.
3
4 public class OtrasConversiones
5 {
6     public static void main( String args[] )
7     {
8         Object prueba = null;
9         System.out.printf( "%b\n", false );
10        System.out.printf( "%b\n", true );
11        System.out.printf( "%b\n", "Prueba" );
12        System.out.printf( "%B\n", prueba );
13        System.out.printf( "El código hash de \"hola\" es %h\n", "hola" );
14        System.out.printf( "El código hash de \"Hola\" es %h\n", "Hola" );
15        System.out.printf( "El código hash de null es %H\n", prueba );
16        System.out.printf( "Impresión de un %% en una cadena de formato\n" );
17        System.out.printf( "Impresión de una nueva línea %nla siguiente línea empieza aquí" );
18    } // fin de main
19 } // fin de la clase OtrasConversiones

```

```

false
true
true
FALSE
El código hash de "hola" es 30f4bc
El código hash de "Hola" es 2268dc
El código hash de null es NULL
Impresión de un % en una cadena de formato
Impresión de una nueva línea
la siguiente línea empieza aquí

```

Figura 29.11 | Uso de los caracteres de conversión b, B, h, H, % y n.

29.9 Impresión con anchuras de campo y precisiones

El tamaño exacto de un campo en el que se imprimen datos se especifica mediante una **anchura de campo**. Si la anchura de campo es mayor que los datos que se van a imprimir, éstos se justifican a la derecha dentro de ese campo, de manera predeterminada. (En la sección 29.10 demostraremos la justificación a la izquierda). El programador inserta un entero que representa la anchura de campo entre el signo de porcentaje (%) y el carácter de conversión (por ejemplo, %4d) en el especificador de formato. En la figura 29.12 se imprimen dos grupos de cinco números cada uno, y se justifican a la derecha los números que contienen menos dígitos que la anchura de campo. Observe que la anchura de campo se incrementa para imprimir valores más anchos que el campo, y que el signo menos para un valor negativo utiliza una posición de carácter en el campo. Además, si no se especifica la anchura del campo, los datos se imprimen en todas las posiciones que sean necesarias. Las anchuras de campo pueden utilizarse con todos los especificadores de formato, excepto el separador de línea (%n).



Error común de programación 29.3

Si no se proporciona una anchura de campo suficientemente extensa como para manejar un valor a imprimir, se pueden desplazar los demás datos que se impriman, con lo que se producirán resultados confusos. ¡Debe conocer sus datos!

El método `printf` también proporciona la habilidad de especificar la precisión con la que se van a imprimir los datos. La precisión tiene distintos significados para los diferentes tipos. Cuando se utiliza con los caracteres de conversión de punto flotante e y f, la precisión es el número de dígitos que aparecen después del punto decimal.


```

1 // Fig. 29.12: PruebaAnchuraCampo.java
2 // Justificación a la derecha de enteros en campos.
3
4 public class PruebaAnchuraCampo
5 {
6     public static void main( String args[] )
7     {
8         System.out.printf( "%4d\n", 1 );
9         System.out.printf( "%4d\n", 12 );
10        System.out.printf( "%4d\n", 123 );
11        System.out.printf( "%4d\n", 1234 );
12        System.out.printf( "%4d\n\n", 12345 ); // datos demasiado extensos
13
14        System.out.printf( "%4d\n", -1 );
15        System.out.printf( "%4d\n", -12 );
16        System.out.printf( "%4d\n", -123 );
17        System.out.printf( "%4d\n", -1234 ); // datos demasiado extensos
18        System.out.printf( "%4d\n", -12345 ); // datos demasiado extensos
19    } // fin de main
20 } // fin de la clase PruebaAnchuraCampo

```

```

1
12
123
1234
12345

-1
-12
-123
-1234
-12345

```

Figura 29.12 | Justificación a la derecha de enteros en campos.

Cuando se utiliza con el carácter de conversión `g`, la precisión es el número máximo de dígitos significativos a imprimir. Cuando se utiliza con el carácter de conversión `s`, la precisión es el número máximo de caracteres a escribir de la cadena. Para utilizar la precisión, se debe colocar entre el signo de porcentaje y el especificador de conversión un punto decimal (`.`), seguido de un entero que representa la precisión. En la figura 29.13 se muestra el uso de la precisión en las cadenas de formato. Observe que, cuando se imprime un valor de punto flotante con una precisión menor que el número original de posiciones decimales en el valor, éste se redondea. Además, observe que el especificador de formato `%.3g` indica que el número total de dígitos utilizados para mostrar el valor de punto flotante es 3. Como el valor tiene tres dígitos a la izquierda del punto decimal, se redondea a la posición de las unidades.

La anchura de campo y la precisión pueden combinarse, para lo cual se coloca la anchura de campo, seguida de un punto decimal, seguido de una precisión entre el signo de porcentaje y el carácter de conversión, como en la siguiente instrucción:

```
printf( "%9.3f", 123.456789 );
```

la cual muestra 123.457 con tres dígitos a la derecha del punto decimal, y se justifica a la derecha en un campo de nueve dígitos; antes del número se colocarán dos espacios en blanco en su campo.

29.10 Uso de banderas en la cadena de formato de printf

Pueden usarse varias banderas con el método `printf` para suplementar sus herramientas de formato de salida. Hay siete banderas disponibles para usarlas en las cadenas de formato (figura 29.14).

```

1 // Fig 29.13: PruebaPrecision.java
2 // Uso de la precisión para números de punto flotante y cadenas.
3 public class PruebaPrecision
4 {
5     public static void main( String args[] )
6     {
7         double f = 123.94536;
8         String s = "Feliz Cumpleaños";
9
10        System.out.printf( "Uso de la precision para numeros de punto flotante\n" );
11        System.out.printf( "\t%.3f\n\t%.3e\n\t%.3g\n\n", f, f, f );
12
13        System.out.printf( "Uso de la precision para las cadenas\n" );
14        System.out.printf( "\t%.11s\n", s );
15    } // fin de main
16 } // fin de la clase PruebaPrecision

```

```

Uso de la precision para numeros de punto flotante
    123.945
    1.239e+02
    124

```

```

Uso de la precision para las cadenas
    Feliz Cump1

```

Figura 29.13 | Uso de la precisión para los números de punto flotante y las cadenas.

Bandera	Descripción
- (signo negativo)	Justifica a la izquierda la salida dentro del campo especificado.
+ (signo positivo)	Muestra un signo positivo antes de los valores positivos, y un signo negativo antes de los valores negativos.
<i>espacio</i>	Imprime un espacio antes de un valor positivo que no se imprime con la bandera +.
#	Antepone un 0 al valor de salida cuando se utiliza con el carácter de conversión octal o. Antepone 0x al valor de salida cuando se usa con el carácter de conversión hexadecimal x.
0 (cero)	Rellena un campo con ceros a la izquierda.
, (coma)	Usa el separador de miles específico para la configuración regional (es decir, ',' para los EUA), para mostrar números decimales y de punto flotante.
(Encierra los números negativos entre paréntesis.

Figura 29.14 | Banderas de la cadena de formato.

Para usar una bandera en una cadena de formato, coloque la bandera justo a la derecha del signo de porcentaje. Pueden usarse varias banderas en el mismo especificador de formato. En la figura 29.15 se muestra la justificación a la derecha y la justificación a la izquierda de una cadena, un entero, un carácter y un número de punto flotante. Observe que la línea 9 sirve como mecanismo de conteo para la salida en la pantalla.

En la figura 29.16 se imprime un número positivo y un número negativo, cada uno con y sin la **bandera +**. Observe que el signo negativo se muestra en ambos casos, pero el signo positivo se muestra sólo cuando se utiliza la bandera +.

```

1 // Fig. 29.15: PruebaBanderaMenos.java
2 // Justificación a la derecha y justificación a la izquierda de los valores
3
4 public class PruebaBanderaMenos
5 {
6     public static void main( String args[] )
7     {
8         System.out.println( "Columnas:" );
9         System.out.println( "0123456789012345678901234567890123456789\n" );
10        System.out.printf( "%10s%10d%10c%10f\n\n", "hello", 7, 'a', 1.23 );
11        System.out.printf(
12            "%-10s%-10d%-10c%-10f\n", "hola", 7, 'a', 1.23 );
13    } // fin de main
14 } // fin de la clase PruebaBanderaMenos

```

```

Columnas:
0123456789012345678901234567890123456789

      hola          7          a  1.230000
hola      7          a          1.230000

```

Figura 29.15 | Justificación a la derecha y justificación a la izquierda de los valores.

En la figura 29.17 se antepone un espacio al número positivo mediante la **bandera de espacio**. Esto es útil para alinear números positivos y negativos con el mismo número de dígitos. Observe que el valor -547 no va precedido por un espacio en la salida, debido a su signo negativo. En la figura 29.18 se utiliza la **bandera #** para anteponer un 0 al valor octal, y **0x** para el valor hexadecimal.

En la figura 29.19 se combinan la bandera +, la **bandera 0** y la bandera de espacio para imprimir 452 en un campo con una anchura de 9, con un signo + y ceros a la izquierda; después se imprime 452 en un campo de anchura 9, usando sólo la bandera 0, y después se imprime 452 en un campo de anchura 9, usando sólo la bandera de espacio.

```

1 // Fig. 29.16: PruebaBanderaMas.java
2 // Impresión de números con y sin la bandera +.
3
4 public class PruebaBanderaMas
5 {
6     public static void main( String args[] )
7     {
8         System.out.printf( "%d\t%d\n", 786, -786 );
9         System.out.printf( "%+d\t%d\n", 786, -786 );
10    } // fin de main
11 } // fin de la clase PruebaBanderaMas

```

```

786      -786
+786      -786

```

Figura 29.16 | Impresión de números con y sin la bandera +.

En la figura 29.20 se utiliza la bandera de coma (,) para mostrar un número decimal y un número de punto flotante con el separador de miles. En la figura 29.21 se encierran números negativos entre paréntesis, usando la bandera (. Observe que el valor 50 no se encierra entre paréntesis en la salida, ya que es un número positivo.

```

1 // Fig. 29.17: PruebaBanderaEspacio.java
2 // Impresión de un espacio antes de valores no negativos.
3
4 public class PruebaBanderaEspacio
5 {
6     public static void main( String args[] )
7     {
8         System.out.printf( "% d\n% d\n", 547, -547 );
9     } // fin de main
10 } // fin de la clase PruebaBanderaEspacio

```

```

547
-547

```

Figura 29.17 | Uso de la bandera de espacio para imprimir un espacio antes de los valores no negativos.

```

1 // Fig. 29.18: PruebaBanderaLibras.java
2 // Uso de la bandera # con los caracteres de conversión o y x.
3
4 public class PruebaBanderaLibras
5 {
6     public static void main( String args[] )
7     {
8         int c = 31;          // inicializa c
9
10        System.out.printf( "%#o\n", c );
11        System.out.printf( "%#x\n", c );
12    } // fin de main
13 } // fin de la clase PruebaBanderaLibras

```

```

037
0x1f

```

Figura 29.18 | Uso de la bandera # con los caracteres de conversión o y x.

```

1 // Fig. 29.19: PruebaBanderaCero.java
2 // Impresión con la bandera 0 (cero) para rellenar con ceros a la izquierda.
3
4 public class PruebaBanderaCero
5 {
6     public static void main( String args[] )
7     {
8         System.out.printf( "%+09d\n", 452 );
9         System.out.printf( "%09d\n", 452 );
10        System.out.printf( "% 9d\n", 452 );
11    } // fin de main
12 } // fin de la clase PruebaBanderaCero

```

```

+00000452
00000452
 452

```

Figura 29.19 | Impresión con la bandera 0 (cero) para rellenar con ceros a la izquierda.

```

1 // Fig. 29.20: PruebaBanderaComa.java
2 // Uso de la bandera de coma (,) para mostrar números con el separador de miles.
3
4 public class PruebaBanderaComa
5 {
6     public static void main( String args[] )
7     {
8         System.out.printf( "%,d\n", 58625 );
9         System.out.printf( "%,.2f\n", 58625.21 );
10        System.out.printf( "%,.2f", 12345678.9 );
11    } // fin de main
12 } // fin de la clase PruebaBanderaComa

```

```

58,625
58,625.21
12,345,678.90

```

Figura 29.20 | Uso de la bandera de coma (,) para mostrar números con el separador de miles.

```

1 // Fig. 29.21: PruebaBanderaParentesis.java
2 // Uso de la bandera ( para colocar paréntesis alrededor de números negativos.
3
4 public class PruebaBanderaParentesis
5 {
6     public static void main( String args[] )
7     {
8         System.out.printf( "%(d\n", 50 );
9         System.out.printf( "%(d\n", -50 );
10        System.out.printf( "%(.1e\n", -50.0 );
11    } // fin de main
12 } // fin de la clase PruebaBanderaParentesis

```

```

50
(50)
(5.0e+01)

```

Figura 29.21 | Uso de la bandera (para colocar paréntesis alrededor de números negativos.

29.11 Impresión con índices como argumentos

Un **índice como argumento** es un entero opcional, seguido de un signo de \$, el cual indica la posición del argumento en la lista de argumentos. Por ejemplo, en las líneas 20 a 21 y 24 a 25 de la figura 29.9 se utiliza el índice como argumento "1\$" para indicar que todos los especificadores de formato utilizan el primer argumento en la lista de argumentos. Los índices como argumentos permiten a los programadores reordenar la salida, de manera que los argumentos en la lista de argumentos no necesariamente se encuentren en el orden de sus especificadores de formato correspondientes. Los índices como argumentos también ayudan a evitar argumentos duplicados. En la figura 29.22 se muestra cómo imprimir argumentos en la lista de argumentos en orden inverso, mediante el uso del índice como argumento.

```

1 // Fig. 29.22: PruebaIndiceArgumento
2 // Reordenamiento de la salida con los índices como argumentos.
3
4 public class PruebaIndiceArgumento

```

Figura 29.22 | Reordenamiento de la salida con los índices como argumentos. (Parte I de 2).

```

5  {
6      public static void main( String args[] )
7      {
8          System.out.printf(
9              "Lista de parametros sin reordenar: %s %s %s %s\n",
10             "primero", "segundo", "tercero", "cuarto" );
11          System.out.printf(
12              "Lista de parametros despues de reordenar: %4$s %3$s %2$s %1$s\n",
13              "primero", "segundo", "tercero", "cuarto" );
14      } // fin de main
15 } // fin de la clase PruebaIndiceArgumento

```

```

Lista de parametros sin reordenar: primero segundo tercero cuarto
Lista de parametros despues de reordenar: cuarto tercero segundo primero

```

Figura 29.22 | Reordenamiento de la salida con los índices como argumentos. (Parte 2 de 2).

29.12 Impresión de literales y secuencias de escape

La mayoría de los caracteres literales que se imprimen en una instrucción `printf` sólo necesitan incluirse en la cadena de formato. Sin embargo, hay varios caracteres “problemáticos”, como el signo de comillas dobles (") que delimita a la cadena de formato en sí. Varios caracteres de control, como el carácter de nueva línea y el de tabulación, deben representarse mediante secuencias de escape. Una secuencia de escape se representa mediante una barra diagonal inversa (\), seguida de un carácter de escape. En la figura 29.23 se listan las secuencias de escape y las acciones que producen.



Error común de programación 29.4

Tratar de imprimir un carácter de comillas dobles o un carácter de barra diagonal inversa como datos literales en una instrucción `printf`, sin anteponer al carácter una barra diagonal inversa para formar una secuencia de escape apropiada, podría producir un error de sintaxis.

Secuencia de escape	Descripción
\' (comilla sencilla)	Imprime el carácter de comilla sencilla (').
\" (doble comilla)	Imprime el carácter de doble comilla (").
\\ (barra diagonal inversa)	Imprime el carácter de barra diagonal inversa (\).
\b (retroceso)	Desplaza el cursor una posición hacia atrás en la línea actual.
\f (nueva página o avance de página)	Desplaza el cursor al principio de la siguiente página lógica.
\n (nueva línea)	Desplaza el cursor al principio de la siguiente línea.
\r (retorno de carro)	Desplaza el cursor al principio de la línea actual.
\t (tabulador horizontal)	Desplaza el cursor hacia la siguiente posición del tabulador horizontal.

Figura 29.23 | Secuencias de escape.

29.13 Aplicación de formato a la salida con la clase `Formatter`

Hasta ahora, hemos visto cómo mostrar salida con formato en el flujo de salida estándar. ¿Qué deberíamos hacer si quisiéramos enviar salidas con formato a otros flujos de entrada o dispositivos, como un objeto `JTextArea` o un archivo? La solución recae en la clase `Formatter` (en el paquete `java.util`), la cual proporciona las mismas herramientas de formato que `printf`. `Formatter` es una clase utilitaria que permite a los programadores imprimi-

mir datos con formato hacia un destino especificado, como un archivo en el disco. De manera predeterminada, un objeto `Formatter` crea una cadena en la memoria. En la figura 29.24 se muestra cómo usar un objeto `Formatter` para crear una cadena con formato, la cual después se muestra en un cuadro de diálogo de mensaje.

En la línea 11 se crea un objeto `Formatter` mediante el uso del constructor predeterminado, por lo que este objeto creará una cadena en la memoria. Se incluyen otros constructores para que el programador pueda especificar el destino hacia el que se deben enviar los datos con formato. Para obtener más información, consulte la página java.sun.com/javase/6/docs/api/java/util/Formatter.html

En la línea 12 se invoca el método `format` para dar formato a la salida. Al igual que `printf`, el método `format` recibe una cadena de formato y una lista de argumentos. La diferencia es que `printf` envía la salida con formato directamente al flujo de salida estándar, mientras que `format` envía la salida con formato al destino especificado por su constructor (en este programa, una cadena en la memoria). En la línea 15 se invoca el método `toString` de `Formatter` para obtener los datos con formato, como una cadena que luego se muestra en un cuadro de diálogo de mensaje.

Observe que la clase `String` también proporciona un método de conveniencia `static` llamado `format`, el cual nos permite crear una cadena en la memoria, sin necesidad de crear primero un objeto `Formatter`. Podríamos haber sustituido las líneas 11 a 12 y la línea 15 de la figura 29.24 por:

```
String s = String.format( "%d = %#o = %#X", 10, 10, 10 );
JOptionPane.showMessageDialog( null, s );
```

```
1 // Fig. 29.24: PruebaFormatter.java
2 // Cadena de formato con la clase Formatter.
3 import java.util.Formatter;
4 import javax.swing.JOptionPane;
5
6 public class PruebaFormatter
7 {
8     public static void main( String args[] )
9     {
10         // crea un objeto Formatter y aplica formato a la salida
11         Formatter formatter = new Formatter();
12         formatter.format( "%d = %#o = %#X", 10, 10, 10 );
13
14         // muestra la salida en el componente JOptionPane
15         JOptionPane.showMessageDialog( null, formatter.toString() );
16     } // fin de main
17 } // fin de la clase PruebaFormatter
```



Figura 29.24 | Aplicar formato a la salida con la clase `Formatter`.

29.14 Conclusión

En este capítulo vimos un resumen acerca de cómo aplicar formato a la salida mediante los diversos caracteres y banderas de formato. Mostramos números decimales mediante el uso de los caracteres de formato `d`, `o`, `x` y `X`. Mostramos números de punto flotante usando los caracteres de formato `e`, `E`, `f`, `g` y `G`. Mostramos la fecha y la hora en diversos formatos, usando los caracteres de formato `t` y `T` junto con sus caracteres de sufijo de conversión. Aprendió a mostrar la salida con anchuras de campo y precisiones. Presentamos las banderas `+`, `-`, espacio, `#`, `0`, coma y `(` que se utilizan en conjunto con los caracteres de formato para producir la salida. También demostramos

cómo aplicar formato a la salida con la clase `Formatter`. En el siguiente capítulo hablaremos sobre los métodos de la clase `String` para manipular cadenas. También presentaremos las expresiones regulares y demostraremos cómo validar la entrada del usuario mediante éstas.

Resumen

Sección 29.2 Flujos

- Por lo general, las operaciones de entrada y salida se llevan a cabo con flujos, los cuales son secuencias de bytes. En las operaciones de entrada, los bytes fluyen de un dispositivo a la memoria principal. En las operaciones de salida, los bytes fluyen de la memoria principal a un dispositivo.
- Por lo común, el flujo de entrada estándar se conecta al teclado, y el flujo de salida estándar se conecta a la pantalla de la computadora.

Sección 29.3 Aplicación de formato a la salida con `printf`

- La cadena de formato de `printf` describe los formatos en los que aparecen los valores de salida. El especificador de formato consiste en un índice como argumento, banderas, anchuras de campo, precisiones y caracteres de conversión.

Sección 29.4 Impresión de enteros

- Los enteros se imprimen con los caracteres de conversión `d` para enteros decimales, `o` para enteros en formato octal y `x` (o `X`) para los enteros en formato hexadecimal. Cuando se utiliza el carácter de conversión `X`, la salida se muestra en letras mayúsculas.

Sección 29.5 Impresión de números de punto flotante

- Los valores de punto flotante se imprimen con los caracteres de conversión `e` (o `E`) para la notación exponencial, `f` para la notación de punto flotante regular, y `g` (o `G`) para la notación `e` (o `E`) o `f`. Cuando se indica el especificador de conversión `g`, se utiliza el carácter de conversión `e` si el valor es menor que 10^{-3} , o mayor o igual a 10^7 ; en caso contrario, se utiliza el carácter de conversión `f`. Cuando se utilizan los caracteres de conversión `E` y `G`, la salida se muestra en letras mayúsculas.

Sección 29.6 Impresión de cadenas y caracteres

- El carácter de conversión `c` imprime un carácter.
- El carácter de conversión `s` (o `S`) imprime una cadena de caracteres. Cuando se utiliza el carácter de conversión `S`, la salida se muestra en letras mayúsculas.

Sección 29.7 Impresión de fechas y horas

- El carácter de conversión `t` (o `T`) seguido de un carácter de sufijo de conversión imprime la fecha y la hora en diversos formatos. Cuando se utiliza el carácter de conversión `T`, la salida se muestra en letras mayúsculas.
- El carácter de conversión `t` (o `T`) requiere que el argumento sea de tipo `long`, `Long`, `Calendar` o `Date`.

Sección 29.8 Otros caracteres de conversión

- El carácter de conversión `b` (o `B`) imprime la representación de cadena de un valor `boolean` o `Boolean`. Estos caracteres de conversión también imprimen `"true"` para las referencias que no son `null`, y `"false"` para las referencias `null`. Cuando se utiliza el carácter de conversión `B`, la salida se muestra en letras mayúsculas.
- El carácter de conversión `h` (o `H`) devuelve `null` para una referencia `null`, y una representación `String` del valor de código hash (en base 16) del objeto. Los códigos de hash se utilizan para almacenar y obtener objetos que se encuentran en objetos `Hashtable` y `HashMap`. Cuando se utiliza el carácter de conversión `H`, la salida se muestra en letras mayúsculas.
- El carácter de conversión `n` imprime el separador de línea específico de la plataforma.
- El carácter de conversión `%` se utiliza para mostrar un `%` literal.

Sección 29.9 Impresión con anchuras de campo y precisiones

- Si la anchura de campo es mayor que el objeto a imprimir, éste se justifica a la derecha en el campo.

- Las anchuras de campo se pueden usar con todos los caracteres de conversión, excepto la conversión con el separador de línea.
- La precisión que se utiliza con los caracteres de conversión de punto flotante e y f indica el número de dígitos que aparecen después del punto decimal. La precisión que se utiliza con el carácter de conversión de punto flotante g indica el número de dígitos significativos que deben aparecer.
- La precisión que se utiliza con el carácter de conversión s indica el número de caracteres a imprimir.
- La anchura de campo y la precisión se pueden combinar, para lo cual se coloca la anchura de campo, seguida de un punto decimal, seguido de la precisión entre el signo de porcentaje y el carácter de conversión.

Sección 29.10 Uso de banderas en la cadena de formato de **printf**

- La bandera - justifica a la izquierda su argumento en un campo.
- La bandera + imprime un signo más para los valores positivos, y un signo menos para los valores negativos.
- La bandera de espacio imprime un espacio antes de un valor positivo. La bandera de espacio y la bandera + no se pueden utilizar juntas en un carácter de conversión integral.
- La bandera # antepone un 0 a los valores octales, y 0x a los valores hexadecimales.
- La bandera 0 imprime ceros a la izquierda para un valor que no ocupa todo su campo.
- La bandera de coma (,) utiliza el separador de miles específico para la configuración regional (por ejemplo, ',' para los EUA), para mostrar números enteros y de punto flotante.
- La bandera (encierra un número negativo entre paréntesis.

Sección 29.11 Impresión con índices como argumentos

- Un índice como argumento es un entero decimal opcional, seguido de un signo \$ que indica la posición del argumento en la lista de argumentos.
- Los índices como argumentos permiten a los programadores reordenar la salida, de manera que los argumentos en la lista de argumentos no estén necesariamente en el orden de sus correspondientes especificadores de formato. Los índices como argumentos también ayudan a evitar los argumentos duplicados.

Sección 29.13 Aplicación de formato a la salida con la clase **Formatter**

- La clase `Formatter` (en el paquete `java.util`) proporciona las mismas herramientas de formato que `printf`. `Formatter` es una clase utilitaria que permite a los programadores imprimir salida con formato hacia varios destinos, incluyendo componentes de GUI, archivos y otros flujos de salida.
- El método `format` de la clase `Formatter` imprime los datos con formato al destino especificado por el constructor de `Formatter`.
- El método `static format` de la clase `String` aplica formato a los datos y devuelve los datos con formato, como un objeto `String`.

Terminología

#, bandera	c, carácter de conversión
%, carácter de conversión	C, carácter de sufijo de conversión
(, bandera	cadena de formato
+ (más), bandera	carácter de conversión
- (menos), bandera	conversión de enteros
, (coma), bandera	conversión de números de punto flotante
0 (cero), bandera	d, carácter de conversión
A, carácter de sufijo de conversión	D, carácter de sufijo de conversión
a, carácter de sufijo de conversión	e, carácter de conversión
alineación	E, carácter de conversión
anchura de campo	e, carácter de sufijo de conversión
b, carácter de conversión	especificador de formato
B, carácter de conversión	f, carácter de conversión
B, carácter de sufijo de conversión	F, carácter de sufijo de conversión
b, carácter de sufijo de conversión	flujo
bandera	flujo de entrada estándar
bandera de espacio	flujo de error estándar

flujo de salida estándar	notación científica
format, método de Formatter	o, carácter de conversión
format, método de String	P, carácter de sufijo de conversión
formato de punto flotante exponencial	p, carácter de sufijo de conversión
formato hexadecimal	precisión
formato octal	printf, método
Formatter, clase	punto flotante
g, carácter de conversión	r, carácter de sufijo de conversión
G, carácter de conversión	redirigir un flujo
h, carácter de conversión	redondeo
H, carácter de conversión	s, carácter de conversión
H, carácter de sufijo de conversión	S, carácter de conversión
I, carácter de sufijo de conversión	S, carácter de sufijo de conversión
índice como argumento	t, carácter de conversión
j, carácter de sufijo de conversión	T, carácter de conversión
justificación a la derecha	T, carácter de sufijo de conversión
justificación a la izquierda	toString, método de Formatter
K, carácter de sufijo de conversión	x, carácter de conversión
m, carácter de sufijo de conversión	y, carácter de sufijo de conversión
M, carácter de sufijo de conversión	Y, carácter de sufijo de conversión
n, carácter de conversión	Z, carácter de sufijo de conversión

Ejercicios de autoevaluación

29.1 Complete los enunciados:

- Todas las operaciones de entrada y salida se manejan en forma de _____.
- El flujo _____ se conecta generalmente al teclado.
- El flujo _____ se conecta generalmente a la pantalla de la computadora.
- El método _____ de `System.out` se puede utilizar para aplicar formato al texto que se muestra en la salida estándar.
- El carácter de conversión _____ puede utilizarse para imprimir en pantalla un entero decimal.
- Los caracteres de conversión _____ y _____ se utilizan para mostrar enteros en formato octal y hexadecimal, respectivamente.
- El carácter de conversión _____ se utiliza para mostrar un valor de punto flotante en notación exponencial.
- Los caracteres de conversión `e` y `f` se muestran con _____ dígitos de precisión a la derecha del punto decimal, si no se especifica una precisión.
- Los caracteres de conversión _____ y _____ se utilizan para imprimir cadenas y caracteres, respectivamente.
- El carácter de conversión _____ y el carácter de sufijo de conversión _____ se utilizan para imprimir la hora para el reloj de 24 horas, como `hora:minuto:segundo`.
- La bandera _____ hace que la salida se justifique a la izquierda en un campo.
- La bandera _____ hace que los valores se muestren con un signo más o con un signo menos.
- El índice como argumento _____ corresponde al segundo argumento en la lista de argumentos.
- La clase _____ tiene la misma capacidad que `printf`, pero permite a los programadores imprimir salida con formato en varios destinos, además del flujo de salida estándar.

29.2 Encuentre el error en cada uno de los siguientes enunciados, y explique cómo se puede corregir.

- La siguiente instrucción debe imprimir el carácter `'c'`.
`System.out.printf("%c\n", "c");`
- La siguiente instrucción debe imprimir `9.375%`.
`System.out.printf("%.3f%", 9.375);`

- c) La siguiente instrucción debe imprimir el tercer argumento en la lista de argumentos:
`System.out.printf("%2$s\n", "Lun", "Mar", "Mie", "Jue", "Vie");`
- d) `System.out.printf(""Una cadena entre comillas"");`
- e) `System.out.printf(%d %d, 12, 20);`
- f) `System.out.printf("%s\n", 'Richard');`

29.3 Escriba una instrucción para cada uno de los siguientes casos:

- a) Imprimir 1234 justificado a la derecha, en un campo de 10 dígitos.
- b) Imprimir 123.456789 en notación exponencial con un signo (+ o -) y 3 dígitos de precisión.
- c) Imprimir 100 en formato octal, precedido por 0.
- d) Dado un objeto `Calendario` de la clase `Calendar`, imprima una fecha con formato de mes/día/año (cada uno con dos dígitos).
- e) Dado un objeto `Calendar` llamado `calendario`, imprimir una hora para el reloj de 24 horas como hora: minuto:segundo (cada uno con dos dígitos), usando un índice como argumento y caracteres de sufijo de conversión para aplicar formato a la hora.
- f) Imprimir 3.333333 con un signo (+ o -) en un campo de 20 caracteres, con una precisión de 3.

Respuestas a los ejercicios de autoevaluación

29.1 a) Flujos. b) de entrada estándar. c) de salida estándar. d) `printf`. e) d. f) o, x o X. g) e o E. h) 6. i) s o S, c o C. j) t, T. k) - (menos). l) + (más). m) 2\$. n) `Formatter`.

29.2 a) Error: el carácter de conversión `c` espera un argumento del tipo primitivo `char`.

Corrección: para imprimir el carácter `'c'`, cambie `"c"` a `'c'`.

b) Error: está tratando de imprimir el carácter literal `%` sin usar el especificador de formato `%%`.

Corrección: use `%%` para imprimir un carácter `%` literal.

c) Error: el índice como argumento no empieza con 0; por ejemplo, el primer argumento es `1$`.

Corrección: para imprimir el tercer argumento, use `3$`.

d) Error: está tratando de imprimir el carácter literal `"` sin usar la secuencia de escape `\`.

Corrección: sustituya cada comilla en el conjunto interno de comillas con `\`.

e) Error: la cadena de formato no va encerrada entre comillas dobles.

Corrección: encierre `%d %d` entre comillas dobles.

f) Error: la cadena a imprimir está encerrada entre comillas.

Corrección: use dobles comillas en vez de comillas sencillas para representar una cadena.

29.3 a) `System.out.printf("%10d\n", 1234);`

b) `System.out.printf("%+.3e\n", 123.456789);`

c) `System.out.printf("%#o\n", 100);`

d) `System.out.printf("%tD\n", calendario);`

e) `System.out.printf("%1$tH:%1$tM:%1$tS\n", calendario);`

f) `System.out.printf("%+20.3f\n", 3.333333);`

Ejercicios

29.4 Escriba una o más instrucciones para cada uno de los siguientes casos:

- a) Imprimir el entero 40000 justificado a la derecha en un campo de 15 dígitos.
- b) Imprimir 200 con y sin un signo.
- c) Imprimir 100 en formato hexadecimal, precedido por 0x.
- d) Imprimir 1.234 con tres dígitos de precisión en un campo de nueve dígitos con ceros a la izquierda.

29.5 Muestre lo que se imprime en cada una de las siguientes instrucciones. Si una instrucción es incorrecta, indique por qué.

a) `System.out.printf("-10d\n", 10000);`

b) `System.out.printf("c\n", "Esta es una cadena");`

c) `System.out.printf("%8.3f\n", 1024.987654);`

d) `System.out.printf("%#o\n%X\n", 17, 17);`

e) `System.out.printf("% d\n%+d\n", 1000000, 1000000);`

```
f) System.out.printf( "%10.2e\n", 444.93738 );
g) System.out.printf( "%d\n", 10.987 );
```

29.6 Encuentre el(los) error(es) en cada uno de los siguientes segmentos de programa. Muestre la instrucción corregida.

```
a) System.out.printf( "%s\n", 'Feliz cumpleaños' );
b) System.out.printf( "%c\n", 'Hola' );
c) System.out.printf( "%c\n", "Esta es una cadena" );
d) La siguiente instrucción debe imprimir "Buen viaje" con las dobles comillas:
   System.out.printf( ""$"" , "Buen viaje" );
e) La siguiente instrucción debe imprimir "Hoy es viernes":
   System.out.printf( "Hoy es %s\n", "Lunes", "Viernes" );
f) System.out.printf( 'Escriba su nombre: ' );
g) System.out.printf( %f, 123.456 );
h) La siguiente instrucción debe imprimir la hora actual en el formato "hh:mm:ss":
   Calendar fechaHora = Calendar.getInstance();
   System.out.printf( "%1$tK:1$tL:%1$tS\n", fechaHora );
```

29.7 (*Impresión de fechas y horas*) Escriba un programa que imprima fechas y horas en los siguientes formatos:

```
GMT-05:00 04/30/04 09:55:09 AM
GMT-05:00 Abril 30 2004 09:55:09
2004-04-30 dia-del-mes:30
2004-04-30 dia-del-ano: 121
Vie Abr 30 09:55:09 GMT-05:00 2004
```

[Nota: dependiendo de su ubicación, tal vez tenga una zona horaria distinta de GMT-05:00].

29.8 Escriba un programa para probar los resultados de imprimir el valor entero 12345 y el valor de punto flotante 1.2345 en campos de varios tamaños.

29.9 (*Redondeo de números*) Escriba un programa que imprima el valor 100.453267 redondeado al dígito, a la decena, centena, múltiplo de mil y de diez mil más cercanos.

29.10 Escriba un programa que reciba como entrada una palabra del teclado y determine su longitud. Imprima la palabra usando el doble de la longitud como anchura del campo.

29.11 (*Conversión de temperatura en grados Fahrenheit a Centígrados*) Escriba un programa que convierta temperaturas enteras en grados Fahrenheit de 0 a 212 grados, a temperaturas en grados Centígrados de punto flotante con tres dígitos de precisión. Use la siguiente fórmula:

```
centigrados = 5.0 / 9.0 * ( fahrenheit - 32 );
```

para realizar el cálculo. La salida debe imprimirse en dos columnas justificadas a la derecha de 10 caracteres cada una, y las temperaturas en grados Centígrados deben ir precedidas por un signo, tanto para los valores positivos como para los negativos.

29.12 Escriba un programa para probar todas las secuencias de escape en la figura 29.23. Para las secuencias de escape que desplazan el cursor, imprima un carácter antes y después de la secuencia de escape, de manera que se pueda ver con claridad a dónde se ha desplazado el cursor.

29.13 Escriba un programa que utilice el carácter de conversión g para imprimir el valor 9876.12345. Imprima el valor con precisiones que varíen de 1 a 9.

—Hilaire Belloc

—William Strunk, Jr.

—Blaise Pascal

- Crear y a manipular objetos de cadena de caracteres inmutables de la clase `String`.
- Crear y a manipular objetos de cadena de caracteres mutables de la clase `StringBuilder`.
- Crear y a manipular objetos de la clase `Character`.
- Usar un objeto `StringTokenizer` para dividir un objeto `String` en tokens.
- Usar expresiones regulares para validar los datos `String` introducidos en una aplicación.

- 30.1 Introducción
- 30.2 Fundamentos de los caracteres y las cadenas
- 30.3 La clase `String`
 - 30.3.1 Constructores de `String`
 - 30.3.2 Métodos `length`, `charAt` y `getChars` de `String`
 - 30.3.3 Comparación entre cadenas
 - 30.3.4 Localización de caracteres y subcadenas en las cadenas
 - 30.3.5 Extracción de subcadenas de las cadenas
 - 30.3.6 Concatenación de cadenas
 - 30.3.7 Métodos varios de `String`
 - 30.3.8 Método `valueOf` de `String`
- 30.4 La clase `StringBuilder`
 - 30.4.1 Constructores de `StringBuilder`
 - 30.4.2 Métodos `length`, `capacity`, `setLength` y `ensureCapacity` de `StringBuilder`
 - 30.4.3 Métodos `charAt`, `setCharAt`, `getChars` y `reverse` de `StringBuilder`
 - 30.4.4 Métodos `append` de `StringBuilder`
 - 30.4.5 Métodos de inserción y eliminación de `StringBuilder`
- 30.5 La clase `Character`
- 30.6 La clase `StringTokenizer`
- 30.7 Expresiones regulares, la clase `Pattern` y la clase `Matcher`
- 30.8 Conclusión

Resumen | Terminología | Ejercicios de autoevaluación | Respuestas a los ejercicios de autoevaluación | Ejercicios
 Sección especial: manipulación avanzada de cadenas | Sección especial: proyectos desafiantes de manipulación de cadenas

30.1 Introducción

En este capítulo presentamos las herramientas para el procesamiento de cadenas y caracteres en Java. Las técnicas que se describen aquí son apropiadas para validar la entrada en los programas, para mostrar información a los usuarios y otras manipulaciones basadas en texto. Estas técnicas son también apropiadas para desarrollar editores de texto, procesadores de palabras, software para el diseño de páginas, sistemas de composición computarizados y demás tipos de software para el procesamiento de texto. Ya hemos presentado varias herramientas para el procesamiento de texto en capítulos anteriores. En este capítulo describiremos detalladamente las herramientas de las clases `String`, `StringBuilder` y `Character` del paquete `java.lang`, y la clase `StringTokenizer` del paquete `java.util`. Estas clases proporcionan la base para la manipulación de cadenas y caracteres en Java.

En este capítulo también hablaremos sobre las expresiones regulares que proporcionan a las aplicaciones la capacidad de validar los datos de entrada. Esta funcionalidad se encuentra en la clase `String`, junto con las clases `Matcher` y `Pattern` ubicadas en el paquete `java.util.regex`.

30.2 Fundamentos de los caracteres y las cadenas

Los caracteres son los bloques de construcción básicos de los programas fuente de Java. Todo programa está compuesto de una secuencia de caracteres que, cuando se agrupan en forma significativa, son interpretados por la computadora como una serie de instrucciones utilizadas para realizar una tarea. Un programa puede contener **literales de carácter**. Una literal de carácter es un valor entero representado como un carácter entre comillas simples. Por ejemplo, `'z'` representa el valor entero de `z`, y `'\n'` representa el valor entero de una nueva línea. El valor de una literal de carácter es el valor entero del carácter en el **conjunto de caracteres Unicode**. En el apéndice B se presentan los equivalentes enteros de los caracteres en el conjunto de caracteres ASCII, el cual es un subconjunto de Unicode (que veremos en el apéndice I). Para obtener información detallada sobre Unicode, visite www.unicode.org.

Recuerde que en la sección 2.2 vimos que una cadena es una secuencia de caracteres que se trata como una sola unidad. Una cadena puede incluir letras, dígitos y varios **caracteres especiales**, tales como +, -, *, / y \$. Una cadena es un objeto de la clase `String`. Las **literales de cadena** (que se almacenan en memoria como objetos `String`) se escriben como una secuencia de caracteres entre comillas dobles, como en:

"Juan E. Pérez"	(un nombre)
"Calle Principal 9999"	(una dirección)
"Monterrey, Nuevo León"	(una ciudad y un estado)
"(201) 555-1212"	(un número telefónico)

Una cadena puede asignarse a una referencia `String`. La declaración

```
String color = "azul";
```

inicializa la variable `String` de nombre `color` para que haga referencia a un objeto `String` que contiene la cadena "azul".



Tip de rendimiento 30.1

Java trata a todas las literales de cadena con el mismo contenido como un solo objeto `String` que tiene muchas referencias. Esto ayuda a conservar memoria.

30.3 La clase String

La clase `String` se utiliza para representar cadenas en Java. En las siguientes subsecciones cubriremos muchas de las herramientas de la clase `String`.

30.3.1 Constructores de String

La clase `String` proporciona constructores para inicializar objetos `String` de varias formas. Cuatro de los constructores se muestran en el método `main` de la figura 30.1.

En la línea 12 se crea una instancia de un nuevo objeto `String`, utilizando el constructor sin argumentos de la clase `String`, y se le asigna su referencia a `s1`. El nuevo objeto `String` no contiene caracteres (la **cadena vacía**) y tiene una longitud de 0.

En la línea 13 se crea una instancia de un nuevo objeto `String`, utilizando el constructor de la clase `String` que toma un objeto `String` como argumento, y se le asigna su referencia a `s2`. El nuevo objeto `String` contiene la misma secuencia de caracteres que el objeto `String` de nombre `s`, el cual se pasa como argumento para el constructor.



Observación de ingeniería de software 30.1

*No es necesario copiar un objeto `String` existente. Los objetos `String` son **inmutables**: su contenido de caracteres no puede modificarse una vez que son creados, ya que la clase `String` no proporciona métodos que permitan modificar el contenido de un objeto `String`.*

En la línea 14 se crea la instancia de un nuevo objeto `String` y se le asigna su referencia a `s3`, utilizando el constructor de la clase `String` que toma un arreglo de caracteres como argumento. El nuevo objeto `String` contiene una copia de los caracteres en el arreglo.

En la línea 15 se crea la instancia de un nuevo objeto `String` y se le asigna su referencia a `s4`, utilizando el constructor de la clase `String` que toma un arreglo `char` y dos enteros como argumentos. El segundo argumento especifica la posición inicial (el desplazamiento) a partir del cual se accede a los caracteres en el arreglo. Recuerde que el primer carácter se encuentra en la posición 0. El tercer argumento especifica el número de caracteres (la cuenta) que se van a utilizar del arreglo. El nuevo objeto `String` contiene una cadena formada a partir de los caracteres utilizados. Si el desplazamiento o la cuenta especificados como argumentos ocasionan que se acceda a un elemento fuera de los límites del arreglo de caracteres, se lanza una excepción `StringIndexOutOfBoundsException`.

```

1 // Fig. 30.1: ConstructoresString.java
2 // Constructores de la clase String.
3
4 public class ConstructoresString
5 {
6     public static void main( String args[] )
7     {
8         char arregloChar[] = { 'c', 'u', 'm', 'p', 'l', 'e', ' ', 'a', 'n', 'i', 'o', 's' };
9         String s = new String( "hola" );
10
11         // usa los constructores de String
12         String s1 = new String();
13         String s2 = new String( s );
14         String s3 = new String( arregloChar );
15         String s4 = new String( arregloChar, 7, 5 );
16
17         System.out.printf(
18             "s1 = %s\ns2 = %s\ns3 = %s\ns4 = %s\n",
19             s1, s2, s3, s4 ); // muestra las cadenas
20     } // fin de main
21 } // fin de la clase ConstructoresString

```

```

s1 =
s2 = hola
s3 = cumple años
s4 = años

```

Figura 30.1 | Constructores de la clase String.



Error común de programación 30.1

Intentar acceder a un carácter que se encuentra fuera de los límites de una cadena (es decir, un índice menor que 0 o un índice mayor o igual a la longitud de la cadena), se produce una excepción `StringIndexOutOfBoundsException`.

30.3.2 Métodos `length`, `charAt` y `getChars` de String

Los métodos `length`, `charAt` y `getChars` de String determinan la longitud de una cadena, obtienen el carácter que se encuentra en una ubicación específica de una cadena y recuperan el conjunto completo de caracteres en una cadena, respectivamente. La aplicación de la figura 30.2 muestra cada uno de estos métodos.

```

1 // Fig. 30.2: VariosString.java
2 // Esta aplicación muestra los métodos length, charAt y getChars
3 // de la clase String.
4
5 public class VariosString
6 {
7     public static void main( String args[] )
8     {
9         String s1 = "hola a todos";
10         char arregloChar[] = new char[ 5 ];
11
12         System.out.printf( "s1: %s", s1 );
13
14         // prueba el método length
15         System.out.printf( "\nLongitud de s1: %d", s1.length() );

```

Figura 30.2 | Métodos de manipulación de caracteres de la clase String. (Parte I de 2).


```

16
17 // itera a través de los caracteres en s1 con charAt y muestra la cadena invertida
18 System.out.print( "\nLa cadena invertida es: " );
19
20 for ( int cuenta = s1.length() - 1; cuenta >= 0; cuenta-- )
21     System.out.printf( "%s ", s1.charAt( cuenta) );
22
23 // copia los caracteres de la cadena a arregloChar
24 s1.getChars( 0, 4, arregloChar, 0 );
25 System.out.print( "\nEl arreglo de caracteres es: " );
26
27 for ( char caracter : arregloChar )
28     System.out.print( caracter );
29
30 System.out.println();
31 } // fin de main
32 } // fin de la clase VariosString

```

```

s1: hola a todos
Longitud de s1: 12
La cadena invertida es: s o d o t   a   a l o h
El arreglo de caracteres es: hola

```

Figura 30.2 | Métodos de manipulación de caracteres de la clase String. (Parte 2 de 2).

En la línea 15 se utiliza el método `length` de `String` para determinar el número de caracteres en la cadena `s1`. Al igual que los arreglos, las cadenas conocen su propia longitud. Sin embargo, a diferencia de los arreglos, no podemos acceder a la longitud de una cadena mediante un campo `length`; en vez de ello, debemos llamar al método `length` del objeto `String`.

En las líneas 20 y 21 se imprimen los caracteres de la cadena `s1` en orden inverso (y separados por espacios). El método `charAt` de `String` (línea 21) devuelve el carácter ubicado en una posición específica en la cadena. El método `charAt` recibe un argumento entero que se utiliza como el índice, y devuelve el carácter en esa posición. Al igual que los arreglos, se considera que el primer elemento de una cadena está en la posición 0.

En la línea 24 se utiliza el método `getChars` de `String` para copiar los caracteres de una cadena en un arreglo de caracteres. El primer argumento es el índice inicial en la cadena, a partir del cual se van a copiar los caracteres. El segundo argumento es el índice que está una posición más adelante del último carácter que se va a copiar de la cadena. El tercer argumento es el arreglo de caracteres en el que se van a copiar los caracteres. El último argumento es el índice inicial en donde se van a colocar los caracteres copiados en el arreglo de caracteres de destino. A continuación, en la línea 28 se imprime el contenido del arreglo `char`, un carácter a la vez.

30.3.3 Comparación entre cadenas

En el capítulo 7 hablamos sobre el ordenamiento y la búsqueda en los arreglos. Con frecuencia, la información que se va a ordenar o buscar consiste en cadenas que deben compararse para determinar el orden o para determinar si una cadena aparece en un arreglo (u otra colección). La clase `String` proporciona varios métodos para comparar cadenas, los cuales mostraremos en los siguientes dos ejemplos.

Para comprender lo que significa que una cadena sea mayor o menor que otra, considere el proceso de alfabetizar una serie de apellidos. Sin duda usted colocaría a “Jones” antes que “Smith”, ya que en el alfabeto la primera letra de “Jones” viene antes que la primera letra de “Smith” en el alfabeto. Pero el alfabeto es algo más que una lista de 26 letras; es un conjunto ordenado de caracteres. Cada letra ocupa una posición específica dentro del conjunto. Z es más que una letra del alfabeto; es en específico la letra número veintiséis del alfabeto.

¿Cómo sabe la computadora que una letra va antes que otra? Todos los caracteres se representan en la computadora como códigos numéricos (vea el apéndice B). Cuando la computadora compara cadenas, en realidad compara los códigos numéricos de los caracteres en las cadenas.

En la figura 30.3 se muestran los métodos `equals`, `equalsIgnoreCase`, `compareTo` y `regionMatches` de `String`, y se muestra el uso del operador de igualdad `==` para comparar objetos `String`.

```

1 // Fig. 30.3: CompararCadenas.java
2 // Los métodos equals, equalsIgnoreCase, compareTo y regionMatches de String.
3
4 public class CompararCadenas
5 {
6     public static void main( String args[] )
7     {
8         String s1 = new String( "hola" ); // s1 es una copia de "hola"
9         String s2 = "adios";
10        String s3 = "Feliz cumpleaños";
11        String s4 = "feliz cumpleaños";
12
13        System.out.printf(
14            "s1 = %s\ns2 = %s\ns3 = %s\ns4 = %s\n\n", s1, s2, s3, s4 );
15
16        // prueba la igualdad
17        if ( s1.equals( "hola" ) ) // true
18            System.out.println( "s1 es igual a \"hola\"" );
19        else
20            System.out.println( "s1 no es igual a \"hola\"" );
21
22        // prueba la igualdad con ==
23        if ( s1 == "hola" ) // false; no son el mismo objeto
24            System.out.println( "s1 es el mismo objeto que \"hola\"" );
25        else
26            System.out.println( "s1 no es el mismo objeto que \"hola\"" );
27
28        // prueba la igualdad (ignora el uso de mayúsculas/minúsculas)
29        if ( s3.equalsIgnoreCase( s4 ) ) // true
30            System.out.printf( "%s es igual a %s si se ignora el uso de mayusculas/
31                               minusculas\n", s3, s4 );
32        else
33            System.out.println( "s3 no es igual a s4" );
34
35        // prueba con compareTo
36        System.out.printf(
37            "\ns1.compareTo( s2 ) es %d", s1.compareTo( s2 ) );
38        System.out.printf(
39            "\ns2.compareTo( s1 ) es %d", s2.compareTo( s1 ) );
40        System.out.printf(
41            "\ns1.compareTo( s1 ) es %d", s1.compareTo( s1 ) );
42        System.out.printf(
43            "\ns3.compareTo( s4 ) es %d", s3.compareTo( s4 ) );
44        System.out.printf(
45            "\ns4.compareTo( s3 ) es %d\n\n", s4.compareTo( s3 ) );
46
47        // prueba con regionMatches (sensible a mayúsculas/minúsculas)
48        if ( s3.regionMatches( 0, s4, 0, 5 ) )
49            System.out.println( "Los primeros 5 caracteres de s3 y s4 coinciden" );
50        else
51            System.out.println(
52                "Los primeros 5 caracteres de s3 y s4 no coinciden" );
53
54        // prueba con regionMatches (ignora el uso de mayúsculas/minúsculas)
55        if ( s3.regionMatches( true, 0, s4, 0, 5 ) )
56            System.out.println( "Los primeros 5 caracteres de s3 y s4 coinciden" );
57        else
58            System.out.println(
59                "Los primeros 5 caracteres de s3 y s4 no coinciden" );

```

Figura 30.3 | Comparaciones entre objetos String. (Parte I de 2).

```

59     } // fin de main
60 } // fin de la clase CompararCadenas

s1 = hola
s2 = adios
s3 = Feliz cumpleaños
s4 = feliz cumpleaños

s1 es igual a "hola"
s1 no es el mismo objeto que "hola"
Feliz cumpleaños es igual a feliz cumpleaños si se ignora el uso de mayusculas
/minusculas

s1.compareTo( s2 ) es 7
s2.compareTo( s1 ) es -7
s1.compareTo( s1 ) es 0
s3.compareTo( s4 ) es -32
s4.compareTo( s3 ) es 32

Los primeros 5 caracteres de s3 y s4 no coinciden
Los primeros 5 caracteres de s3 y s4 coinciden

```

Figura 30.3 | Comparaciones entre objetos String. (Parte 2 de 2).

La condición en la línea 17 utiliza el método `equals` para comparar la igualdad entre la cadena `s1` y la literal de cadena `"hola"`. El método `equals` (un método de la clase `Object`, sobrescrito en `String`) prueba la igualdad entre dos objetos (es decir, que las cadenas contenidas en los dos objetos sean idénticas). El método devuelve `true` si el contenido de los objetos es igual y `false` en caso contrario. La condición anterior es `true`, ya que la cadena `s1` fue inicializada con la literal de cadena `"hola"`. El método `equals` utiliza una **comparación lexicográfica**; se comparan los valores enteros Unicode (vea el apéndice I, Unicode®, en el sitio Web www.deitel.com/jhpt7) que representan a cada uno de los caracteres en cada cadena. Por lo tanto, si la cadena `"hola"` se compara con la cadena `"HOLA"`, el resultado es `false` ya que la representación entera de una letra minúscula es distinta de la letra mayúscula correspondiente.

La condición en la línea 23 utiliza el operador de igualdad `==` para comparar la igualdad entre la cadena `s1` y la literal de cadena `"hola"`. El operador `==` tiene distinta funcionalidad cuando se utiliza para comparar referencias, que cuando se utiliza para comparar valores de tipos primitivos. Cuando se comparan valores de tipos primitivos con `==`, el resultado es `true` si ambos valores son idénticos. Cuando se comparan referencias con `==`, el resultado es `true` si ambas referencias se refieren al mismo objeto en memoria. Para comparar la igualdad del contenido en sí (o información sobre el estado) de los objetos, hay que invocar un método. En el caso de los objetos `String`, ese método es `equals`. La condición anterior se evalúa como `false` en la línea 23, ya que la referencia `s1` se inició con la instrucción

```
s1 = new String( "hola" );
```

con lo cual se crea un nuevo objeto `String` con una copia de la literal de cadena `"hola"`, y se asigna el nuevo objeto a la variable `s1`. Si `s1` se hubiera inicializado con la instrucción

```
s1 = "hola";
```

con lo cual se asigna directamente la literal de cadena `"hola"` a la variable `s1`, la condición hubiera sido `true`. Recuerde que Java trata a todos los objetos de literal de cadena con el mismo contenido como un objeto `String`, al cual puede haber muchas referencias. Por lo tanto, en las líneas 8, 17 y 23 se hace referencia al mismo objeto `String "hola"` en memoria.



Error común de programación 30.2

Al comparar referencias con `==` se pueden producir errores lógicos, ya que `==` compara las referencias para determinar si se refieren al mismo objeto, no si dos objetos tienen el mismo contenido. Si se comparan dos objetos idénticos (pero separados) con `==`, el resultado será `false`. Si va a comparar objetos para determinar si tienen el mismo contenido, utilice el método `equals`.

Si va a ordenar objetos `String`, puede comparar si son iguales con el método `equalsIgnoreCase`, el cual ignora si las letras en cada cadena son mayúsculas o minúsculas al realizar la comparación. Por lo tanto, la cadena "hola" y la cadena "HOLA" se consideran iguales. En la línea 29 se utiliza el método `equalsIgnoreCase` de `String` para comparar si la cadena `s3` (Feliz Cumpleaños) es igual a la cadena `s4` (feliz cumpleaños). El resultado de esta comparación es `true`, ya que la comparación ignora el uso de mayúsculas y minúsculas.

En las líneas 35 a 44 se utiliza el método `compareTo` de `String` para comparar cadenas. El método `compareTo` se declara en la interfaz `Comparable` y se implementa en la clase `String`. En la línea 36 se compara la cadena `s1` con la cadena `s2`. El método `compareTo` devuelve 0 si las cadenas son iguales, un número negativo si la cadena que invoca a `compareTo` es menor que la cadena que se pasa como argumento, y un número positivo si la cadena que invoca a `compareTo` es mayor que la cadena que se pasa como argumento. El método `compareTo` utiliza una comparación lexicográfica; compara los valores numéricos de los caracteres correspondientes en cada cadena (para obtener más información acerca del valor exacto devuelto por el método `compareTo`, vea la página java.sun.com/javase/6/docs/api/java/lang/String.html).

La condición en la línea 47 utiliza el método `regionMatches` de `String` para comparar si ciertas porciones de dos cadenas son iguales. El primer argumento es el índice inicial en la cadena que invoca al método. El segundo argumento es una cadena de comparación. El tercer argumento es el índice inicial en la cadena de comparación. El último argumento es el número de caracteres a comparar entre las dos cadenas. El método devuelve `true` solamente si el número especificado de caracteres son lexicográficamente iguales.

Por último, la condición en la línea 54 utiliza una versión con cinco argumentos del método `regionMatches` de `String` para comparar si ciertas porciones de dos cadenas son iguales. Cuando el primer argumento es `true`, el método ignora el uso de mayúsculas y minúsculas en los caracteres que se van a comparar. Los argumentos restantes son idénticos a los que se describieron para el método `regionMatches` con cuatro argumentos.

En el siguiente ejemplo (figura 30.4) vamos a mostrar los métodos `startsWith` y `endsWith` de la clase `String`. El método `main` crea el arreglo cadenas que contiene las cadenas "empezo", "empezando", "termino" y "terminando". El resto del método `main` consiste en tres instrucciones `for` que prueban los elementos del arreglo para determinar si empiezan o terminan con un conjunto específico de caracteres.

```

1  // Fig. 30.4: CadenaInicioFin.java
2  // Los métodos startsWith y endsWith de String.
3
4  public class CadenaInicioFin
5  {
6      public static void main( String args[] )
7      {
8          String cadenas[] = { "empezo", "empezando", "termino", "terminando" };
9
10         // prueba el método startsWith
11         for ( String cadena: cadenas )
12         {
13             if ( cadena.startsWith( "em" ) )
14                 System.out.printf( "\\%s\\ " empieza con \\em\\ \\n", cadena );
15         } // fin de for
16
17         System.out.println();
18
19         // prueba el método startsWith empezando desde la posición 2 de la cadena
20         for ( String cadena: cadenas )

```

Figura 30.4 | Métodos `startsWith` y `endsWith` de la clase `String`. (Parte 1 de 2).

```

21      {
22          if ( cadena.startsWith( "pez", 2 ) )
23              System.out.printf(
24                  "\\\"%s\\\" empieza con \\\"pez\\\" en la posicion 2\\n", cadena );
25      } // fin de for
26
27      System.out.println();
28
29      // prueba el método endsWith
30      for ( String cadena: cadenas )
31      {
32          if ( cadena.endsWith( "do" ) )
33              System.out.printf( "\\\"%s\\\" termina con \\\"do\\\"\\n", cadena );
34      } // fin de for
35  } // fin de main
36 } // fin de la clase CadenaInicioFin

```

```

"empezo" empieza con "em"
"empezando" empieza con "em"

"empezo" empieza con "pez" en la posicion 2
"empezando" empieza con "pez" en la posicion 2

"empezando" termina con "do"
"terminando" termina con "do"

```

Figura 30.4 | Métodos `startsWith` y `endsWith` de la clase `String`. (Parte 2 de 2).

En las líneas 11 a 15 se utiliza la versión del método `startsWith` que recibe un argumento `String`. La condición en la instrucción `if` (línea 13) determina si cada objeto `String` en el arreglo empieza con los caracteres "em". De ser así, el método devuelve `true` y la aplicación imprime ese objeto `String`. En caso contrario, el método devuelve `false` y no ocurre nada.

En las líneas 20 a 25 se utiliza el método `startsWith` que recibe un objeto `String` y un entero como argumentos. El argumento entero especifica el índice en el que debe empezar la comparación en la cadena. La condición en la instrucción `if` (línea 22) determina si cada objeto `String` en el arreglo tiene los caracteres "pez", empezando con el tercer carácter en cada cadena. De ser así, el método devuelve `true` y la aplicación imprime el objeto `String`.

La tercera instrucción `for` (líneas 30 a 34) utiliza el método `endsWith` que recibe un argumento `String`. La condición en la línea 32 determina si cada objeto `String` en el arreglo termina con los caracteres "do". De ser así, el método devuelve `true` y la aplicación imprime el objeto `String`.

30.3.4 Localización de caracteres y subcadenas en las cadenas

A menudo es útil buscar un carácter o conjunto de caracteres en una cadena. Por ejemplo, si usted va a crear su propio procesador de palabras, tal vez quiera proporcionar una herramienta para buscar a través de los documentos. En la figura 30.5 se muestran las diversas versiones de los métodos `indexOf` y `lastIndexOf` de `String`, que buscan un carácter o subcadena especificados en una cadena. Todas las búsquedas en este ejemplo se realizan en la cadena `letras` (inicializada con "abcdefghijklmabcdefghijklm") que se encuentra en el método `main`. En las líneas 11 a 16 se utiliza el método `indexOf` para localizar la primera ocurrencia de un carácter en una cadena. Si `indexOf` encuentra el carácter, devuelve el índice de ese carácter en la cadena; en caso contrario `indexOf` devuelve -1. Hay dos versiones de `indexOf` que buscan caracteres en una cadena. La expresión en la línea 12 utiliza la versión de `indexOf` que recibe una representación entera del carácter que se va a buscar. La expresión en la línea 14 utiliza otra versión del método `indexOf`, que recibe dos argumentos enteros: el carácter y el índice inicial en el que debe empezar la búsqueda en la cadena.

Las instrucciones en las líneas 19 a 24 utilizan el método `lastIndexOf` para localizar la última ocurrencia de un carácter en una cadena. El método `lastIndexOf` realiza la búsqueda desde el final de la cadena, hacia el inicio

de la misma. Si el método `lastIndexOf` encuentra el carácter, devuelve el índice de ese carácter en la cadena; en caso contrario, devuelve `-1`. Hay dos versiones de `lastIndexOf` que buscan caracteres en una cadena. La expresión en la línea 20 utiliza la versión que recibe la representación entera del carácter. La expresión en la línea 22 utiliza la versión que recibe dos argumentos enteros: la representación entera de un carácter y el índice a partir del cual debe iniciarse la búsqueda inversa de ese carácter.

En las líneas 27 a 40 se muestran las versiones de los métodos `indexOf` y `lastIndexOf` que reciben, cada una de ellas, un objeto `String` como el primer argumento. Estas versiones de los métodos se ejecutan en forma idéntica a las descritas anteriormente, excepto que buscan secuencias de caracteres (o subcadenas) que se especifican mediante sus argumentos `String`. Si se encuentra la subcadena, estos métodos devuelven el índice en la cadena del primer carácter en la subcadena.

```

1 // Fig. 30.5: MetodosIndexString.java
2 // Métodos indexOf y lastIndexOf para buscar en cadenas.
3
4 public class MetodosIndexString
5 {
6     public static void main( String args[] )
7     {
8         String letras = "abcdefghijklmabcdefghijklm";
9
10        // prueba indexOf para localizar un carácter en una cadena
11        System.out.printf(
12            "'c' se encuentra en el índice %d\n", letras.indexOf( 'c' ) );
13        System.out.printf(
14            "'a' se encuentra en el índice %d\n", letras.indexOf( 'a', 1 ) );
15        System.out.printf(
16            "'$' se encuentra en el índice %d\n\n", letras.indexOf( '$' ) );
17
18        // prueba lastIndexOf para buscar un carácter en una cadena
19        System.out.printf( "La última 'c' se encuentra en el índice %d\n",
20            letras.lastIndexOf( 'c' ) );
21        System.out.printf( "La última 'a' se encuentra en el índice %d\n",
22            letras.lastIndexOf( 'a', 25 ) );
23        System.out.printf( "La última '$' se encuentra en el índice %d\n\n",
24            letras.lastIndexOf( '$' ) );
25
26        // prueba indexOf para localizar una subcadena en una cadena
27        System.out.printf( "\"def\" se encuentra en el índice %d\n",
28            letras.indexOf( "def" ) );
29        System.out.printf( "\"def\" se encuentra en el índice %d\n",
30            letras.indexOf( "def", 7 ) );
31        System.out.printf( "\"hola\" se encuentra en el índice %d\n\n",
32            letras.indexOf( "hola" ) );
33
34        // prueba lastIndexOf para buscar una subcadena en una cadena
35        System.out.printf( "La última ocurrencia de \"def\" se encuentra en el índice
36            %d\n",
37            letras.lastIndexOf( "def" ) );
38        System.out.printf( "La última ocurrencia de \"def\" se encuentra en el índice
39            %d\n",
40            letras.lastIndexOf( "def", 25 ) );
41        System.out.printf( "La última ocurrencia de \"hola\" se encuentra en el índice
42            %d\n",
43            letras.lastIndexOf( "hola" ) );
44    } // fin de main
45 } // fin de la clase MetodosIndexString

```

Figura 30.5 | Métodos de búsqueda de la clase `String`. (Parte I de 2).

```
'c' se encuentra en el índice 2
'a' se encuentra en el índice 13
'$' se encuentra en el índice -1

La última 'c' se encuentra en el índice 15
La última 'a' se encuentra en el índice 13
La última '$' se encuentra en el índice -1

"def" se encuentra en el índice 3
"def" se encuentra en el índice 16
"holá" se encuentra en el índice -1

La última ocurrencia de "def" se encuentra en el índice 16
La última ocurrencia de "def" se encuentra en el índice 16
La última ocurrencia de "holá" se encuentra en el índice -1
```

Figura 30.5 | Métodos de búsqueda de la clase String. (Parte 2 de 2).

30.3.5 Extracción de subcadenas de las cadenas

La clase String proporciona dos métodos `substring` para permitir la creación de un nuevo objeto String al copiar parte de un objeto String existente. Cada método devuelve un nuevo objeto String. Ambos métodos se muestran en la figura 30.6.

La expresión `letras.substring(20)` en la línea 12 utiliza el método `substring` que recibe un argumento entero. Este argumento especifica el índice inicial en la cadena original `letras`, a partir del cual se van a copiar caracteres. La subcadena devuelta contiene una copia de los caracteres, desde el índice inicial hasta el final de la cadena. Si el índice especificado como argumento se encuentra fuera de los límites de la cadena, el programa genera una excepción `StringIndexOutOfBoundsException`.

La expresión `letras.substring(3, 6)` en la línea 15 utiliza el método `substring` que recibe dos argumentos enteros. El primer argumento especifica el índice inicial a partir del cual se van a copiar caracteres en la cadena original. El segundo argumento especifica el índice que está una posición más allá del último carácter a copiar (es decir, copiar hasta, pero sin incluir a, ese índice en la cadena). La subcadena devuelta contiene copias de los caracteres especificados de la cadena original. Si los argumentos especificados están fuera de los límites de la cadena, el programa genera una excepción `StringIndexOutOfBoundsException`.

```
1 // Fig. 30.6: SubString.java
2 // Métodos substring de la clase String.
3
4 public class SubString
5 {
6     public static void main( String args[] )
7     {
8         String letras = "abcdefghijklmabcdefghijklm";
9
10        // prueba los métodos substring
11        System.out.printf( "La subcadena desde el índice 20 hasta el final es \"%s\\n\"",
12            letras.substring( 20 ) );
13        System.out.printf( "%s \\\"%s\\n\"",
14            "La subcadena desde el índice 3 hasta, pero sin incluir al 6 es",
15            letras.substring( 3, 6 ) );
16    } // fin de main
17 } // fin de la clase SubString
```

```
La subcadena desde el índice 20 hasta el final es "hijklm"
La subcadena desde el índice 3 hasta, pero sin incluir al 6 es "def"
```

Figura 30.6 | Métodos `substring` de la clase String.

30.3.6 Concatenación de cadenas

El método `concat` (figura 30.7) de `String` concatena dos objetos `String` y devuelve un nuevo objeto `String`, el cual contiene los caracteres de ambos objetos `String` originales. La expresión `s1.concat(s2)` de la línea 13 forma una cadena, anexando los caracteres de la cadena `s2` a los caracteres de la cadena `s1`. Los objetos `String` originales a los que se refieren `s1` y `s2` no se modifican.

```

1 // Fig. 30.7: ConcatenacionString.java
2 // Método concat de String.
3
4 public class ConcatenacionString
5 {
6     public static void main( String args[] )
7     {
8         String s1 = new String( "Feliz " );
9         String s2 = new String( "Cumpleaños" );
10
11         System.out.printf( "s1 = %s\ns2 = %s\n\n", s1, s2 );
12         System.out.printf(
13             "Resultado de s1.concat( s2 ) = %s\n", s1.concat( s2 ) );
14         System.out.printf( "s1 despues de la concatenacion= %s\n", s1 );
15     } // fin de main
16 } // fin de la clase ConcatenacionString

```

```

s1 = Feliz
s2 = Cumpleaños

```

```

Resultado de s1.concat( s2 ) = Feliz Cumpleaños
s1 despues de la concatenacion= Feliz

```

Figura 30.7 | Método `concat` de `String`.

30.3.7 Métodos varios de `String`

La clase `String` proporciona varios métodos que devuelven copias modificadas de cadenas, o que devuelven arreglos de caracteres. Estos métodos se muestran en la aplicación de la figura 30.8.

```

1 // Fig. 30.8: VariosString2.java
2 // Métodos replace, toLowerCase, toUpperCase, trim y toCharArray de String.
3
4 public class VariosString2
5 {
6     public static void main( String args[] )
7     {
8         String s1 = new String( "hola" );
9         String s2 = new String( "ADIOS" );
10        String s3 = new String( "   espacios   " );
11
12        System.out.printf( "s1 = %s\ns2 = %s\ns3 = %s\n\n", s1, s2, s3 );
13
14        // prueba del método replace
15        System.out.printf(
16            "Reemplazar 'l' con 'L' en s1: %s\n\n", s1.replace( 'l', 'L' ) );
17
18        // prueba de toLowerCase y toUpperCase
19        System.out.printf( "s1.toUpperCase() = %s\n", s1.toUpperCase() );

```

Figura 30.8 | Métodos `replace`, `toLowerCase`, `toUpperCase`, `trim` y `toCharArray` de `String`. (Parte I de 2).


```

20      System.out.printf( "s2.toLowerCase() = %s\n\n", s2.toLowerCase() );
21
22      // prueba del método trim
23      System.out.printf( "s3 despues de trim = \"%s\"\n\n", s3.trim() );
24
25      // prueba del método toCharArray
26      char arregloChar[] = s1.toCharArray();
27      System.out.print( "s1 como arreglo de caracteres = " );
28
29      for ( char caracter : arregloChar )
30          System.out.print( caracter );
31
32      System.out.println();
33  } // fin de main
34 } // fin de la clase VariosString2

```

```

s1 = hola
s2 = ADIOS
s3 =   espacios

Reemplazar 'l' con 'L' en s1: hoLa

s1.toUpperCase() = HOLA
s2.toLowerCase() = adios

s3 despues de trim = "espacios"

s1 como arreglo de caracteres = hola

```

Figura 30.8 | Métodos `replace`, `toLowerCase`, `toUpperCase`, `trim` y `toCharArray` de `String`. (Parte 2 de 2).

En la línea 16 se utiliza el método `replace` de `String` para devolver un nuevo objeto `String`, en el que cada ocurrencia del carácter 'l' en la cadena `s1` se reemplaza con el carácter 'L'. El método `replace` no modifica la cadena original. Si no hay ocurrencias del primer argumento en la cadena, el método `replace` devuelve la cadena original.

En la línea 19 se utiliza el método `toUpperCase` de `String` para generar un nuevo objeto `String` con letras mayúsculas, cuyas letras minúsculas correspondientes existen en `s1`. El método devuelve un nuevo objeto `String` que contiene la cadena convertida y deja la cadena original sin cambios. Si no hay caracteres que convertir, el método `toUpperCase` devuelve la cadena original.

En la línea 20 se utiliza el método `toLowerCase` de `String` para devolver un nuevo objeto `String` con letras minúsculas, cuyas letras mayúsculas correspondientes existen en `s2`. La cadena original permanece sin cambios. Si no hay caracteres que convertir en la cadena original, `toLowerCase` devuelve la cadena original.

En la línea 23 se utiliza el método `trim` de `String` para generar un nuevo objeto `String` que elimine todos los caracteres de espacio en blanco que aparecen al principio o al final en la cadena en la que opera `trim`. El método devuelve un nuevo objeto `String` que contiene a la cadena sin espacios en blanco a la izquierda o a la derecha. La cadena original permanece sin cambios.

En la línea 26 se utiliza el método `toCharArray` de `String` para crear un nuevo arreglo de caracteres, el cual contiene una copia de los caracteres en la cadena `s1`. En las líneas 29 y 30 se imprime cada `char` en el arreglo.

30.3.8 Método `valueOf` de `String`

Como hemos visto, todo objeto en Java tiene un método `toString` que permite a un programa obtener la representación de cadena del objeto. Desafortunadamente, esta técnica no puede utilizarse con tipos primitivos, ya que éstos no tienen métodos. La clase `String` proporciona métodos `static` que reciben un argumento de cualquier tipo y lo convierten en un objeto `String`. En la figura 30.9 se muestra el uso de los métodos `valueOf` de la clase `String`.

```

1 // Fig. 30.9: String.valueOf.java
2 // Métodos valueOf de String.
3
4 public class StringValueOf
5 {
6     public static void main( String args[] )
7     {
8         char arregloChar[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
9         boolean valorBoolean = true;
10        char valorCaracter = 'Z';
11        int valorEntero = 7;
12        long valorLong = 10000000000L; // el sufijo L indica long
13        float valorFloat = 2.5f; // f indica que 2.5 es un float
14        double valorDouble = 33.333; // no hay sufijo, double es el predeterminado
15        Object refObjeto = "hola"; // asigna la cadena a una referencia Object
16
17        System.out.printf(
18            "arreglo de valores char = %s\n", String.valueOf( arregloChar ) );
19        System.out.printf( "parte del arreglo char = %s\n",
20            String.valueOf( arregloChar, 3, 3 ) );
21        System.out.printf(
22            "boolean = %s\n", String.valueOf( valorBoolean ) );
23        System.out.printf(
24            "char = %s\n", String.valueOf( valorCaracter ) );
25        System.out.printf( "int = %s\n", String.valueOf( valorEntero ) );
26        System.out.printf( "long = %s\n", String.valueOf( valorLong ) );
27        System.out.printf( "float = %s\n", String.valueOf( valorFloat ) );
28        System.out.printf(
29            "double = %s\n", String.valueOf( valorDouble ) );
30        System.out.printf( "Object = %s\n", String.valueOf( refObjeto ) );
31    } // fin de main
32 } // fin de la clase StringValueOf

```

```

arreglo de valores char = abcdef
parte del arreglo char = def
boolean = true
char = Z
int = 7
long = 10000000000
float = 2.5
double = 33.333
Object = hola

```

Figura 30.9 | Métodos valueOf de la clase String.

La expresión `String.valueOf(arregloChar)` en la línea 18 utiliza el arreglo de caracteres `arregloChar` para crear un nuevo objeto `String`. La expresión `String.valueOf(arregloChar, 3, 3)` de la línea 20 utiliza una porción del arreglo de caracteres `arregloChar` para crear un nuevo objeto `String`. El segundo argumento especifica el índice inicial a partir del cual se van a utilizar caracteres. El tercer argumento especifica el número de caracteres a usar.

Existen otras siete versiones del método `valueOf`, las cuales toman argumentos de tipo `boolean`, `char`, `int`, `long`, `float`, `double` y `Object`, respectivamente. Estas versiones se muestran en las líneas 21 a 30. Observe que la versión de `valueOf` que recibe un objeto `Object` como argumento puede hacerlo debido a que todos los objetos `Object` pueden convertirse en objetos `String` mediante el método `toString`.

[Nota: en las líneas 12 y 13 se utilizan los valores literales `10000000000L` y `2.5f` como valores iniciales de la variable `valorLong` tipo `long` y de la variable `valorFloat` tipo `float`, respectivamente. De manera predeterminada, Java trata a las literales enteras como tipo `int` y a las literales de punto flotante como tipo `double`. Si se anexa la letra `L` a la literal `10000000000` y se anexa la letra `f` a la literal `2.5`, se indica al compilador que `10000000000`

debe tratarse como `long` y que `2.5` debe tratarse como `float`. Se puede usar una `L` mayúscula o una `l` minúscula para denotar una variable de tipo `long`, y una `F` mayúscula o una `f` minúscula para denotar una variable de tipo `float`].

30.4 La clase `StringBuilder`

Una vez que se crea un objeto `String`, su contenido ya no puede variar. Ahora hablaremos sobre las características de la clase `StringBuilder` para crear y manipular información de cadenas dinámicas; es decir, cadenas que pueden modificarse. Cada objeto `StringBuilder` es capaz de almacenar varios caracteres especificados por su capacidad. Si se excede la capacidad de un objeto `StringBuilder`, ésta se expande de manera automática para dar cabida a los caracteres adicionales. La clase `StringBuilder` también se utiliza para implementar los operadores `+` y `+=` para la concatenación de objetos `String`.



Tip de rendimiento 30.2

Java puede realizar ciertas optimizaciones en las que se involucran objetos `String` (como compartir un objeto `String` entre varias referencias), ya que sabe que estos objetos no cambiarán. Si los datos no van a cambiar, debemos usar objetos `String` (y no `StringBuilder`).



Tip de rendimiento 30.3

En los programas que realizan la concatenación de cadenas con frecuencia, o en otras modificaciones de cadenas, a menudo es más eficiente implementar las modificaciones con la clase `StringBuilder`.



Observación de ingeniería de software 30.2

Los objetos `StringBuilder` no son seguros para los subprocesos. Si varios subprocesos requieren acceso a la misma información de una cadena dinámica, use la clase `StringBuffer` en su código. Las clases `StringBuilder` y `StringBuffer` son idénticas, pero la clase `StringBuffer` es segura para los subprocesos.

30.4.1 Constructores de `StringBuilder`

La clase `StringBuilder` proporciona cuatro constructores. En la figura 30.10 mostramos tres de ellos. En la línea 8 se utiliza el constructor de `StringBuilder` sin argumentos para crear un objeto `StringBuilder` que no contiene caracteres, y tiene una capacidad inicial de 16 caracteres (el valor predeterminado para un objeto `StringBuilder`). En la línea 9 se utiliza el constructor de `StringBuilder` que recibe un argumento entero para crear un objeto `StringBuilder` que no contiene caracteres, y su capacidad inicial se especifica mediante el argumento entero (es decir, 10). En la línea 10 se utiliza el constructor de `StringBuilder` que recibe un argumento `String` (en este caso, una literal de cadena) para crear un objeto `StringBuilder` que contiene los caracteres en el argumento `String`. La capacidad inicial es el número de caracteres en el argumento `String`, más 16.

Las instrucciones en las líneas 12 a 14 utilizan el método `toString` de la clase `StringBuilder` para imprimir los objetos `StringBuilder` con el método `printf`. En la sección 30.4.4, hablaremos acerca de cómo Java usa los objetos `StringBuilder` para implementar los operadores `+` y `+=` para la concatenación de cadenas.

```

1 // Fig. 30.10: ConstructoresStringBuilder.java
2 // Constructores de StringBuilder.
3
4 public class ConstructoresStringBuilder
5 {
6     public static void main( String args[] )
7     {
8         StringBuilder bufer1 = new StringBuilder();
9         StringBuilder bufer2 = new StringBuilder( 10 );
10        StringBuilder bufer3 = new StringBuilder( "ho!a" );
11    }

```

Figura 30.10 | Constructores de la clase `StringBuilder`. (Parte 1 de 2).

```

12      System.out.printf( "bufer1 = \"%s\\n", bufer1.toString() );
13      System.out.printf( "bufer2 = \"%s\\n", bufer2.toString() );
14      System.out.printf( "bufer3 = \"%s\\n", bufer3.toString() );
15  } // fin de main
16  } // fin de la clase ConstructoresStringBuilder

```

```

bufer1 = ""
bufer2 = ""
bufer3 = "hola"

```

Figura 30.10 | Constructores de la clase `StringBuilder`. (Parte 2 de 2).

30.4.2 Métodos `length`, `capacity`, `setLength` y `ensureCapacity` de `StringBuilder`

La clase `StringBuilder` proporciona los métodos `length` y `capacity` para devolver el número actual de caracteres en un objeto `StringBuilder` y el número de caracteres que pueden almacenarse en un objeto `StringBuilder` sin necesidad de asignar más memoria, respectivamente. El método `ensureCapacity` permite al programador garantizar que un `StringBuilder` tenga, cuando menos, la capacidad especificada. El método `setLength` permite al programador incrementar o decrementar la longitud de un objeto `StringBuilder`. En la figura 30.11 se muestra el uso de estos métodos.

La aplicación contiene un objeto `StringBuilder` llamado `bufer`. En la línea 8 se utiliza el constructor de `StringBuilder` que toma un argumento `String` para inicializar el objeto `StringBuilder` con la cadena "Hola, como estas?". En las líneas 10 y 11 se imprimen el contenido, la longitud y la capacidad del objeto `StringBuilder`. Observe en la ventana de salida que la capacidad del objeto `StringBuilder` es inicialmente de 35. Recuerde que el constructor de `StringBuilder` que recibe un argumento `String` inicializa la capacidad con la longitud de la cadena que se le pasa como argumento, más 16.

En la línea 13 se utiliza el método `ensureCapacity` para expandir la capacidad del objeto `StringBuilder` a un mínimo de 75 caracteres. En realidad, si la capacidad original es menor que el argumento, este método asegura una capacidad que sea el valor mayor entre el número especificado como argumento, y el doble de la capacidad original más 2. Si la capacidad actual del objeto `StringBuilder` es más que la capacidad especificada, la capacidad del objeto `StringBuilder` permanece sin cambios.

```

1  // Fig. 30.11: StringBuilderCapLen.java
2  // Métodos length, setLength, capacity y ensureCapacity de StringBuilder.
3
4  public class StringBuilderCapLen
5  {
6      public static void main( String args[] )
7      {
8          StringBuilder bufer = new StringBuilder( "Hola, como estas?" );
9
10         System.out.printf( "bufer = %s\\nlongitud = %d\\ncapacidad = %d\\n\\n",
11                             bufer.toString(), bufer.length(), bufer.capacity() );
12
13         bufer.ensureCapacity( 75 );
14         System.out.printf( "Nueva capacidad = %d\\n\\n", bufer.capacity() );
15
16         bufer.setLength( 10 );
17         System.out.printf( "Nueva longitud = %d\\nbufer = %s\\n",
18                             bufer.length(), bufer.toString() );
19     } // fin de main
20 } // fin de la clase StringBuilderCapLen

```

Figura 30.11 | Métodos `length` y `capacity` de `StringBuilder`. (Parte 1 de 2).

```

bufer = Hola, como estas?
longitud = 17
capacidad = 33

Nueva capacidad = 75

Nueva longitud = 10
bufer = Hola, como

```

Figura 30.11 | Métodos `length` y `capacity` de `StringBuilder`. (Parte 2 de 2).



Tip de rendimiento 30.4

El proceso de incrementar en forma dinámica la capacidad de un objeto `StringBuilder` puede requerir una cantidad considerable de tiempo. La ejecución de un gran número de estas operaciones puede degradar el rendimiento de una aplicación. Si un objeto `StringBuilder` va a aumentar su tamaño en forma considerable, tal vez varias veces, si establecemos su capacidad a un nivel alto en un principio se incrementará el rendimiento.

En la línea 16 se utiliza el método `setLength` para establecer la longitud del objeto `StringBuilder` en 10. Si la longitud especificada es menor que el número actual de caracteres en el objeto `StringBuilder`, el búfer se trunca a la longitud especificada (es decir, los caracteres en el objeto `StringBuilder` después de la longitud especificada se descartan). Si la longitud especificada es mayor que el número de caracteres actualmente en el objeto `StringBuilder`, se anexan caracteres nulos (caracteres con la representación numérica de 0) al objeto `StringBuilder` hasta que el número total de caracteres en el objeto `StringBuilder` sea igual a la longitud especificada.

30.4.3 Métodos `charAt`, `setCharAt`, `getChars` y `reverse` de `StringBuilder`

La clase `StringBuilder` proporciona los métodos `charAt`, `setCharAt`, `getChars` y `reverse` para manipular los caracteres en un objeto `StringBuffer`. Cada uno de estos métodos se muestra en la figura 30.12.

```

1 // Fig. 30.12: StringBuilderChars.java
2 // Métodos charAt, setCharAt, getChars y reverse de StringBuilder.
3
4 public class StringBuilderChars
5 {
6     public static void main( String args[] )
7     {
8         StringBuilder bufer = new StringBuilder( "hola a todos" );
9
10        System.out.printf( "bufer = %s\n", bufer.toString() );
11        System.out.printf( "Caracter en 0: %s\nCaracter en 3: %s\n\n",
12            bufer.charAt( 0 ), bufer.charAt( 3 ) );
13
14        char arregloChars[] = new char[ bufer.length() ];
15        bufer.getChars( 0, bufer.length(), arregloChars, 0 );
16        System.out.print( "Los caracteres son: " );
17
18        for ( char character : arregloChars )
19            System.out.print( character );
20
21        bufer.setCharAt( 0, 'H' );
22        bufer.setCharAt( 7, 'T' );
23        System.out.printf( "\n\nbufer = %s", bufer.toString() );

```

Figura 30.12 | Métodos para la manipulación de caracteres de `StringBuilder`. (Parte 1 de 2).

```

24
25     bufer.reverse();
26     System.out.printf( "\n\nbufer = %s\n", bufer.toString() );
27 } // fin de main
28 } // fin de la clase StringBuilderChars

```

```

bufer = hola a todos
Caracter en 0: h
Caracter en 3: a

```

Los caracteres son: hola a todos

```
bufer = Hola a Todos
```

```
bufer = sodoT a aloH
```

Figura 30.12 | Métodos para la manipulación de caracteres de `StringBuilder`. (Parte 2 de 2).

El método `charAt` (línea 12) recibe un argumento entero y devuelve el carácter que se encuentre en el objeto `StringBuilder` en ese índice. El método `getChars` (línea 15) copia los caracteres de un objeto `StringBuilder` al arreglo de caracteres que recibe como argumento. Este método recibe cuatro argumentos: el índice inicial a partir del cual deben copiarse caracteres en el objeto `StringBuilder`, el índice una posición más allá del último carácter a copiar del objeto `StringBuilder`, el arreglo de caracteres en el que se van a copiar los caracteres y la posición inicial en el arreglo de caracteres en donde debe colocarse el primer carácter. El método `setCharAt` (líneas 21 y 22) recibe un entero y un carácter como argumentos y asigna al carácter en la posición especificada en el objeto `StringBuilder` el carácter que recibe como argumento. El método `reverse` (línea 25) invierte el contenido del objeto `StringBuilder`.



Error común de programación 30.3

Al tratar de acceder a un carácter que se encuentra fuera de los límites de un objeto `StringBuilder` (es decir, con un índice menor que 0, o mayor o igual que la longitud del objeto `StringBuilder`) se produce una excepción `StringIndexOutOfBoundsException`.

30.4.4 Métodos `append` de `StringBuilder`

La clase `StringBuilder` proporciona métodos `append` sobrecargados (que mostramos en la figura 30.13) para permitir que se agreguen valores de diversos tipos al final de un objeto `StringBuilder`. Se proporcionan versiones para cada uno de los tipos primitivos y para arreglos de caracteres, objetos `String`, `Object`, `StringBuilder` y `CharSequence` (recuerde que el método `toString` produce una representación de cadena de cualquier objeto `Object`). Cada uno de los métodos recibe su argumento, lo convierte en una cadena y lo anexa al objeto `StringBuilder`.

```

1 // Fig. 30.13: StringBuilderAppend.java
2 // Métodos append de StringBuilder.
3
4 public class StringBuilderAppend
5 {
6     public static void main( String args[] )
7     {
8         Object refObjeto = "hola";
9         String cadena = "adios";
10        char arregloChar[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
11        boolean valorBoolean = true;

```

Figura 30.13 | Métodos `append` de la clase `StringBuilder`. (Parte 1 de 2).

```

12     char valorChar = 'Z';
13     int valorInt = 7;
14     long valorLong = 10000000000L;
15     float valorFloat = 2.5f; // el sufijo f indica que 2.5 es un float
16     double valorDouble = 33.333;
17
18     StringBuilder ultimoBufer = new StringBuilder( "ultimo bufer" );
19     StringBuilder bufer = new StringBuilder();
20
21     bufer.append( refObjeto );
22     bufer.append( "\n" ); // cada uno de estos contiene nueva línea
23     bufer.append( cadena );
24     bufer.append( "\n" );
25     bufer.append( arregloChar );
26     bufer.append( "\n" );
27     bufer.append( arregloChar, 0, 3 );
28     bufer.append( "\n" );
29     bufer.append( valorBoolean );
30     bufer.append( "\n" );
31     bufer.append( valorChar );
32     bufer.append( "\n" );
33     bufer.append( valorInt );
34     bufer.append( "\n" );
35     bufer.append( valorLong );
36     bufer.append( "\n" );
37     bufer.append( valorFloat );
38     bufer.append( "\n" );
39     bufer.append( valorDouble );
40     bufer.append( "\n" );
41     bufer.append( ultimoBufer );
42
43     System.out.printf( "bufer contiene %s\n", bufer.toString() );
44 } // fin de main
45 } // fin de StringBuilderAppend

```

```

bufer contiene hola
adios
abcdef
abc
true
Z
7
10000000000
2.5
33.333
ultimo bufer

```

Figura 30.13 | Métodos append de la clase StringBuilder. (Parte 2 de 2).

En realidad, el compilador utiliza los objetos `StringBuilder` y los métodos `append` para implementar los operadores `+` y `+=` para concatenar objetos `String`. Por ejemplo, suponga que se realizan las siguientes declaraciones:

```

String cadena1 = "hola";
String cadena2 = "BC"
int valor = 22;

```

la instrucción

```
String s = cadena1 + cadena2 + valor;
```

concatena a "hola", "BC" y 22. La concatenación se realiza de la siguiente manera:

```
new StringBuilder().append( "hola" ).append( "BC" ).append( 22 ).toString();
```

Primero, Java crea un objeto `StringBuilder` vacío y después anexa a este objeto `StringBuffer` las cadenas "hola", "BC" y el entero 22. A continuación, el método `toString` de `StringBuilder` convierte el objeto `StringBuilder` en un objeto `String` que se asigna al objeto `String s`. La instrucción

```
s += "!";
```

se ejecuta de la siguiente manera:

```
s = new StringBuilder().append( s ).append( "!" ).toString();
```

Primero, Java crea un objeto `StringBuilder` vacío y después anexa a ese objeto `StringBuilder` el contenido actual de `s`, seguido por "!". A continuación, el método `toString` de `StringBuilder` convierte el objeto `StringBuilder` en una representación de cadena, y el resultado se asigna a `s`.

30.4.5 Métodos de inserción y eliminación de `StringBuilder`

La clase `StringBuilder` proporciona métodos `insert` sobrecargados para permitir que se inserten valores de diversos tipos en cualquier posición de un objeto `StringBuilder`. Se proporcionan versiones para cada uno de los tipos primitivos, y para arreglos de caracteres, objetos `String`, `Object` y `CharSequence`. Cada uno de los métodos toma su segundo argumento, lo convierte en una cadena y la inserta justo antes del índice especificado por el primer argumento. El primer argumento debe ser mayor o igual que 0, y menor que la longitud del objeto `StringBuilder`; de no ser así, se produce una excepción `StringIndexOutOfBoundsException`. La clase `StringBuilder` también proporciona métodos `delete` y `deleteCharAt` para eliminar caracteres en cualquier posición de un objeto `StringBuilder`. El método `delete` recibe dos argumentos: el índice inicial y el índice que se encuentra una posición más allá del último de los caracteres que se van a eliminar. Se eliminan todos los caracteres que empiezan en el índice inicial hasta, pero sin incluir al índice final. El método `deleteCharAt` recibe un argumento: el índice del carácter a eliminar. El uso de índices inválidos hace que ambos métodos lancen una excepción `StringIndexOutOfBoundsException`. Los métodos `insert`, `delete` y `deleteCharAt` se muestran en la figura 30.14.

```
1 // Fig. 30.14: StringBuilderInsert.java
2 // Métodos insert, delete y deleteCharAt de StringBuilder
3
4 public class StringBuilderInsert
5 {
6     public static void main( String args[] )
7     {
8         Object refObjeto = "hola";
9         String cadena = "adios";
10        char arregloChars[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
11        boolean valorBoolean = true;
12        char valorChar = 'K';
13        int valorInt = 7;
14        long valorLong = 10000000;
15        float valorFloat = 2.5f; // el sufijo f indica que 2.5 es un float
16        double valorDouble = 33.333;
17
18        StringBuilder bufer = new StringBuilder();
19
20        bufer.insert( 0, refObjeto );
21        bufer.insert( 0, " " ); // cada uno de estos contiene nueva línea
22        bufer.insert( 0, cadena );
23        bufer.insert( 0, " " );
```

Figura 30.14 | Métodos `insert` y `delete` de `StringBuilder`. (Parte I de 2).


```

24     bufer.insert( 0, arregloChars );
25     bufer.insert( 0, " " );
26     bufer.insert( 0, arregloChars, 3, 3 );
27     bufer.insert( 0, " " );
28     bufer.insert( 0, valorBoolean );
29     bufer.insert( 0, " " );
30     bufer.insert( 0, valorChar );
31     bufer.insert( 0, " " );
32     bufer.insert( 0, valorInt );
33     bufer.insert( 0, " " );
34     bufer.insert( 0, valorLong );
35     bufer.insert( 0, " " );
36     bufer.insert( 0, valorFloat );
37     bufer.insert( 0, " " );
38     bufer.insert( 0, valorDouble );
39
40     System.out.printf(
41         "bufer despues de insertar:\n%s\n\n", bufer.toString() );
42
43     bufer.deleteCharAt( 10 ); // elimina el 5 en 2.5
44     bufer.delete( 2, 6 ); // elimina el .333 en 33.333
45
46     System.out.printf(
47         "bufer despues de eliminar:\n%s\n", bufer.toString() );
48     } // fin de main
49 } // fin de la clase StringBuilderInsert

```

```

bufer despues de insertar:
33.333 2.5 10000000 7 K true def abcdef adios hola

bufer despues de eliminar:
33 2. 10000000 7 K true def abcdef adios hola

```

Figura 30.14 | Métodos insert y delete de StringBuilder. (Parte 2 de 2).

30.5 La clase Character

En el capítulo 17 vimos que Java proporciona ocho clases de envoltura de tipos: `Boolean`, `Character`, `Double`, `Float`, `Byte`, `Short`, `Integer` y `Long`, los cuales permiten que los valores de tipo primitivo sean tratados como objetos. En esta sección presentaremos la clase `Character`: la clase de envoltura de tipos para el tipo primitivo `char`.

La mayoría de los métodos de la clase `Character` son `static`, diseñados para facilitar el procesamiento de valores `char` individuales. Estos métodos reciben cuando menos un argumento tipo carácter y realizan una prueba o manipulación del carácter. Esta clase también contiene un constructor que recibe un argumento `char` para inicializar un objeto `Character`. En los siguientes tres ejemplos presentaremos la mayor parte de los métodos de la clase `Character`. Para obtener más información sobre esta clase (y las demás clases de envoltura de tipos), consulte el paquete `java.lang` en la documentación del API de Java.

En la figura 30.15 se muestran algunos métodos `static` que prueban caracteres para determinar si son de un tipo de carácter específico y los métodos `static` que realizan conversiones de caracteres de minúscula a mayúscula, y viceversa. Puede introducir cualquier carácter y aplicar estos métodos a ese carácter.

En la línea 15 se utiliza el método `isDefined` de `Character` para determinar si el carácter `c` está definido en el conjunto de caracteres Unicode. De ser así, el método devuelve `true`; en caso contrario, devuelve `false`. En la línea 16 se utiliza el método `isDigit` de `Character` para determinar si el carácter `c` es un dígito definido en Unicode. De ser así el método devuelve `true` y, en caso contrario devuelve `false`.

En la línea 18 se utiliza el método `isJavaIdentifierStart` de `Character` para determinar si `c` es un carácter que puede ser el primer carácter de un identificador en Java; es decir, una letra, un guión bajo (`_`) o un signo de dólares (`$`). De ser así, el método devuelve `true`; en caso contrario devuelve `false`. En la línea 20 se utiliza

```

1 // Fig. 30.15: MetodosStaticChar.java
2 // Prueba de los métodos static de Character y los métodos de conversión de mayúsculas/
  minúsculas.
3 import java.util.Scanner;
4
5 public class MetodosStaticChar
6 {
7     public static void main( String args[] )
8     {
9         Scanner scanner = new Scanner( System.in ); // crea objeto scanner
10        System.out.println( "Escriba un caracter y oprima Intro" );
11        String entrada = scanner.next();
12        char c = entrada.charAt( 0 ); // obtiene el caracter de entrada
13
14        // muestra información sobre los caracteres
15        System.out.printf( "esta definido: %b\n", Character.isDefined( c ) );
16        System.out.printf( "es digito: %b\n", Character.isDigit( c ) );
17        System.out.printf( "es el primer caracter en un identificador de Java: %b\n",
18            Character.isJavaIdentifierStart( c ) );
19        System.out.printf( "es parte de un identificador de Java: %b\n",
20            Character.isJavaIdentifierPart( c ) );
21        System.out.printf( "es letra: %b\n", Character.isLetter( c ) );
22        System.out.printf(
23            "es letra o digito: %b\n", Character.isLetterOrDigit( c ) );
24        System.out.printf(
25            "es minuscula: %b\n", Character.isLowerCase( c ) );
26        System.out.printf(
27            "es mayuscula: %b\n", Character.isUpperCase( c ) );
28        System.out.printf(
29            "a mayuscula: %s\n", Character.toUpperCase( c ) );
30        System.out.printf(
31            "a minuscula: %s\n", Character.toLowerCase( c ) );
32    } // fin de main
33 } // fin de la clase MetodosStaticChar

```

Escriba un caracter y oprima Intro

```

A
esta definido: true
es digito: false
es el primer caracter en un identificador de Java: true
es parte de un identificador de Java: true
es letra: true
es letra o digito: true
es minuscula: false
es mayuscula: true
a mayuscula: A
a minuscula: a

```

Escriba un caracter y oprima Intro

```

8
esta definido: true
es digito: true
es el primer caracter en un identificador de Java: false
es parte de un identificador de Java: true
es letra: false
es letra o digito: true

```

Figura 30.15 | Métodos static de la clase Character para probar caracteres y convertir de mayúsculas a minúsculas, y viceversa. (Parte I de 2).

```

es minuscula: false
es mayuscula: false
a mayuscula: 8
a minuscula: 8

```

```

Escriba un caracter y oprima Intro
$
esta definido: true
es digito: false
es el primer caracter en un identificador de Java: true
es parte de un identificador de Java: true
es letra: false
es letra o digito: false
es minuscula: false
es mayuscula: false
a mayuscula: $
a minuscula: $

```

Figura 30.15 | Métodos `static` de la clase `Character` para probar caracteres y convertir de mayúsculas a minúsculas, y viceversa. (Parte 2 de 2).

el método `isJavaIdentifierPart` de `Character` para determinar si el carácter `c` puede utilizarse en un identificador en Java; es decir, un dígito, una letra, un guión bajo (`_`) o un signo de dólares (`$`). De ser así, el método devuelve `true`; en caso contrario devuelve `false`.

En la línea 21 se utiliza el método `isLetter` de `Character` para determinar si el carácter `c` es una letra. Si es así, el método devuelve `true`; en caso contrario devuelve `false`. En la línea 23 se utiliza el método `isLetterOrDigit` de `Character` para determinar si el carácter `c` es una letra o un dígito. Si es así, el método devuelve `true`; en caso contrario devuelve `false`.

En la línea 25 se utiliza el método `isLowerCase` de `Character` para determinar si el carácter `c` es una letra minúscula. Si es así, el método devuelve `true`; en caso contrario devuelve `false`. En la línea 27 se utiliza el método `isUpperCase` de `Character` para determinar si el carácter `c` es una letra mayúscula. Si es así, el método devuelve `true`; en caso contrario devuelve `false`.

En la línea 29 se utiliza el método `toUpperCase` de `Character` para convertir el carácter `c` en su letra mayúscula equivalente. El método devuelve el carácter convertido si éste tiene un equivalente en mayúscula; en caso contrario, el método devuelve su argumento original. En la línea 31 se utiliza el método `toLowerCase` de `Character` para convertir el carácter `c` en su letra minúscula equivalente. El método devuelve el carácter convertido si éste tiene un equivalente en minúscula; en caso contrario, el método devuelve su argumento original.

En la figura 30.16 se muestran los métodos estáticos `digit` y `forDigit` de `Character`, los cuales convierten caracteres a dígitos y dígitos a caracteres, respectivamente, en distintos sistemas numéricos. Los sistemas numéricos comunes son el decimal (base 10), octal (base 8), hexadecimal (base 16) y binario (base 2). La base de un número se conoce también como su *raíz*. Para obtener más información sobre las conversiones entre sistemas numéricos, vea el apéndice E.

```

1 // Fig. 30.16: MetodosStaticChar2.java
2 // Métodos de conversión estáticos de Character.
3 import java.util.Scanner;
4
5 public class MetodosStaticChar2
6 {
7     // crea el objeto MetodosStaticChar2 y ejecuta la aplicación
8     public static void main( String args[] )
9     {
10         Scanner scanner = new Scanner( System.in );

```

Figura 30.16 | Métodos de conversión `static` de la clase `Character`. (Parte 1 de 2).

```

11
12 // obtiene la raíz
13 System.out.println( "Escriba una raíz:" );
14 int raiz = scanner.nextInt();
15
16 // obtiene la selección del usuario
17 System.out.printf( "Seleccione una opcion:\n1 -- %s\n2 -- %s\n",
18 "Convertir digito a caracter", "Convertir caracter a digito" );
19 int opcion = scanner.nextInt();
20
21 // procesa la petición
22 switch ( opcion )
23 {
24     case 1: // convierte dígito a carácter
25         System.out.println( "Escriba un digito:" );
26         int digito = scanner.nextInt();
27         System.out.printf( "Convertir digito a caracter: %s\n",
28             Character.forDigit( digito, raiz ) );
29         break;
30
31     case 2: // convierte carácter a dígito
32         System.out.println( "Escriba un caracter:" );
33         char caracter = scanner.next().charAt( 0 );
34         System.out.printf( "Convertir caracter a digito: %s\n",
35             Character.digit( caracter, raiz ) );
36         break;
37     } // fin de switch
38 } // fin de main
39 } // fin de la clase MetodosStaticChar2

```

```

Escriba una raíz:
16
Seleccione una opcion:
1 -- Convertir digito a caracter
2 -- Convertir caracter a digito
2
Escriba un caracter:
A
Convertir caracter a digito: 10

```

```

Escriba una raíz:
16
Seleccione una opcion:
1 -- Convertir digito a caracter
2 -- Convertir caracter a digito
1
Escriba un digito:
13
Convertir digito a caracter: d

```

Figura 30.16 | Métodos de conversión static de la clase Character. (Parte 2 de 2).

En la línea 28 se utiliza el método `forDigit` para convertir el entero `digito` en un carácter del sistema numérico especificado por el entero `raiz` (la base del número). Por ejemplo, el entero decimal 13 en base 16 (la raíz) tiene el valor de carácter 'd'. Observe que las letras en minúsculas y mayúsculas representan el mismo valor en los sistemas numéricos. En la línea 35 se utiliza el método `digit` para convertir el carácter `c` en un entero del sistema numérico especificado por el entero `raiz` (la base del número). Por ejemplo, el carácter 'A' es la representación en base 16 (la raíz) del valor 10 en base 10. La raíz debe estar entre 2 y 36, inclusive.

En la figura 30.17 se muestra el constructor y varios métodos no `static` de la clase `Character`: `charValue`, `toString` y `equals`. En las líneas 8 y 9 se instancian dos objetos `Character` al realizar conversiones autoboxing en las constantes de caracteres 'A' y 'a', respectivamente. En la línea 12 se utiliza el método `charValue` de `Character` para devolver el valor `char` almacenado en el objeto `Character` llamado `c1`. En la línea 12 se devuelve la representación de cadena del objeto `Character` llamado `c2`, utilizando el método `toString`. La condición en la instrucción `if...else` de las líneas 14 a 17 utiliza el método `equals` para determinar si el objeto `c1` tiene el mismo contenido que el objeto `c2` (es decir, si los caracteres dentro de cada objeto son iguales).

```

1 // Fig. 30.17: OtrosMetodosChar.java
2 // Métodos no static de Character.
3
4 public class OtrosMetodosChar
5 {
6     public static void main( String args[] )
7     {
8         Character c1 = 'A';
9         Character c2 = 'a';
10
11         System.out.printf(
12             "c1 = %s\nc2 = %s\n\n", c1.charValue(), c2.toString() );
13
14         if ( c1.equals( c2 ) )
15             System.out.println( "c1 y c2 son iguales\n" );
16         else
17             System.out.println( "c1 y c2 no son iguales\n" );
18     } // fin de main
19 } // fin de la clase OtrosMetodosChar

```

```

c1 = A
c2 = a

```

```

c1 y c2 no son iguales

```

Figura 30.17 | Métodos no `static` de la clase `Character`.

30.6 La clase `StringTokenizer`

Cuando usted lee una oración, su mente la divide en **tokens** (palabras individuales y signos de puntuación, cada uno de los cuales transfiere a usted su significado). Los compiladores también llevan a cabo la descomposición de instrucciones en piezas individuales tales como palabras clave, identificadores, operadores y demás elementos de un lenguaje de programación. Ahora estudiaremos la clase `StringTokenizer` de Java (del paquete `java.util`), la cual descompone una cadena en los tokens que la componen. Los tokens se separan unos de otros mediante **delimitadores**, que generalmente son caracteres de espacio en blanco tales como los espacios, tabuladores, nuevas líneas y retornos de carro. También pueden utilizarse otros caracteres como delimitadores para separar tokens. La aplicación de la figura 30.18 muestra el uso de la clase `StringTokenizer`.

Cuando el usuario oprime *Intro*, el enunciado de entrada se almacena en la variable `enunciado`. En la línea 17 se crea un objeto `StringTokenizer` para `enunciado`. El constructor de `StringTokenizer` recibe un argumento de cadena y crea un objeto `StringTokenizer` para esa cadena, utilizando la cadena delimitadora predeterminada `"\t\n\r\f"` que consiste en un espacio, un tabulador, un retorno de carro y una nueva línea, para la descomposición en tokens. Hay otros dos constructores para la clase `StringTokenizer`. En la versión que recibe dos argumentos `String`, el segundo `String` es la cadena delimitadora. En la versión que recibe tres argumentos, el segundo `String` es la cadena delimitadora y el tercer argumento (`boolean`) determina si los delimitadores también se devuelven como tokens (sólo si este argumento es `true`). Esto es útil si usted necesita saber cuáles son los delimitadores.

En la línea 19 se utiliza el método `countTokens` de `StringTokenizer` para determinar el número de tokens en la cadena que se va a descomponer en tokens. La condición en la línea 21 utiliza el método `hasMoreTokens`

```

1 // Fig. 30.18: PruebaToken.java
2 // La clase StringTokenizer.
3 import java.util.Scanner;
4 import java.util.StringTokenizer;
5
6 public class PruebaToken
7 {
8     // ejecuta la aplicación
9     public static void main( String args[] )
10    {
11        // obtiene el enunciado
12        Scanner scanner = new Scanner( System.in );
13        System.out.println( "Escriba un enunciado y oprima Intro" );
14        String enunciado = scanner.nextLine();
15
16        // procesa el enunciado del usuario
17        StringTokenizer tokens = new StringTokenizer( enunciado );
18        System.out.printf( "Numero de elementos: %d\nLos tokens son:\n",
19                           tokens.countTokens() );
20
21        while ( tokens.hasMoreTokens() )
22            System.out.println( tokens.nextToken() );
23    } // fin de main
24 } // fin de la clase PruebaToken

```

```

Escriba un enunciado y oprima Intro
Este es un enunciado con siete tokens
Numero de elementos: 7
Los tokens son:
Este
es
un
enunciado
con
siete
tokens

```

Figura 30.18 | Objeto StringTokenizer utilizado para descomponer cadenas en tokens.

de StringTokenizer para determinar si hay más tokens en la cadena que va a descomponerse. De ser así, en la línea 22 se imprime el siguiente token en el objeto String. El siguiente token se obtiene mediante una llamada al método `nextToken` de StringTokenizer, el cual devuelve un objeto String. El token se imprime mediante el uso de `println`, de manera que los siguientes tokens aparezcan en líneas separadas.

Si desea cambiar la cadena delimitadora mientras descompone una cadena en tokens, puede hacerlo especificando una nueva cadena delimitadora en una llamada a `nextToken`, como se muestra a continuación:

```
tokens.nextToken( nuevaCadenaDelimitadora );
```

Esta característica no se muestra en la figura 30.18.

30.7 Expresiones regulares, la clase Pattern y la clase Matcher

Las **expresiones regulares** son secuencias de caracteres y símbolos que definen un conjunto de cadenas. Son útiles para validar la entrada y asegurar que los datos estén en un formato específico. Por ejemplo, un código postal debe consistir de cinco dígitos, y un apellido sólo debe contener letras, espacios, apóstrofes y guiones cortos. Una aplicación de las expresiones regulares es facilitar la construcción de un compilador. A menudo se utiliza una expresión regular larga y compleja para validar la sintaxis de un programa. Si el código del programa no coincide con la expresión regular, el compilador sabe que hay un error de sintaxis dentro del código.

La clase `String` proporciona varios métodos para realizar operaciones con expresiones regulares, siendo la más simple la operación de concordancia. El método `matches` de la clase `String` recibe una cadena que especifica la expresión regular, e iguala el contenido del objeto `String` que lo llama con la expresión regular. Este método devuelve un valor de tipo `boolean` indicando si hubo concordancia o no.

Una expresión regular consiste de caracteres literales y símbolos especiales. La tabla de la figura 30.19 especifica algunas **clases predefinidas de caracteres** que pueden usarse con las expresiones regulares. Una clase de carácter es una secuencia de escape que representa a un grupo de caracteres. Un dígito es cualquier carácter numérico. Un **carácter de palabra** es cualquier letra (mayúscula o minúscula), cualquier dígito o el carácter de guión bajo. Un carácter de espacio en blanco es un espacio, tabulador, retorno de carro, nueva línea o avance de página. Cada clase de carácter se iguala con un solo carácter en la cadena que intentamos hacer concordar con la expresión regular.

Las expresiones regulares no están limitadas a esas clases predefinidas de caracteres. Las expresiones utilizan varios operadores y otras formas de notación para igualar patrones complejos. Analizaremos varias de estas técnicas en la aplicación de las figuras 30.20 y 30.21, la cual valida la entrada del usuario mediante expresiones regulares. [Nota: esta aplicación no está diseñada para igualar todos los posibles datos de entrada del usuario].

Carácter	Concuerda con	Carácter	Concuerda con
<code>\d</code>	cualquier dígito	<code>\D</code>	cualquier carácter que no sea dígito
<code>\w</code>	cualquier carácter de palabra	<code>\W</code>	cualquier carácter que no sea de palabra
<code>\s</code>	cualquier espacio en blanco	<code>\S</code>	cualquier carácter que no sea de espacio en blanco

Figura 30.19 | Clases predefinidas de caracteres.

```

1  // Fig. 30.20: ValidacionEntrada.java
2  // Valida la información del usuario mediante expresiones regulares.
3
4  public class ValidacionEntrada
5  {
6      // valida el primer nombre
7      public static boolean validarPrimerNombre( String primerNombre )
8      {
9          return primerNombre.matches( "[A-Z][a-zA-Z]*" );
10     } // fin del método validarPrimerNombre
11
12     // valida el apellido
13     public static boolean validarApellidoPaterno( String apellidoPaterno )
14     {
15         return apellidoPaterno.matches( "[a-zA-z]+([ '][a-zA-Z]+)*" );
16     } // fin del método validarApellidoPaterno
17
18     // valida la dirección
19     public static boolean validarDireccion( String direccion )
20     {
21         return direccion.matches(
22             "\\d+\\s+([a-zA-Z]+|[a-zA-Z]+\\s[a-zA-Z]+)" );
23     } // fin del método validarDireccion
24
25     // valida la ciudad
26     public static boolean validarCiudad( String ciudad )
27     {
28         return ciudad.matches( "([a-zA-Z]+|[a-zA-Z]+\\s[a-zA-Z]+)" );
29     } // fin del método validarCiudad

```

Figura 30.20 | Valida la información del usuario mediante expresiones regulares. (Parte I de 2).

```

30
31 // valida el estado
32 public static boolean validarEstado( String estado )
33 {
34     return estado.matches( "([a-zA-Z]+|[a-zA-Z]+\\s[a-zA-Z]+)" );
35 } // fin del método validarEstado
36
37 // valida el código postal
38 public static boolean validarCP( String cp )
39 {
40     return cp.matches( "\\d{5}" );
41 } // fin del método validarCP
42
43 // valida el teléfono
44 public static boolean validarTelefono( String telefono )
45 {
46     return telefono.matches( "[1-9]\\d{2}-[1-9]\\d{2}-\\d{4}" );
47 } // fin del método validarTelefono
48 } // fin de la clase ValidacionEntrada

```

Figura 30.20 | Valida la información del usuario mediante expresiones regulares. (Parte 2 de 2).

```

1 // Fig. 30.21: Validacion.java
2 // Valida la información del usuario mediante expresiones regulares.
3 import java.util.Scanner;
4
5 public class Validacion
6 {
7     public static void main( String[] args )
8     {
9         // obtiene la entrada del usuario
10        Scanner scanner = new Scanner( System.in );
11        System.out.println( "Escriba el primer nombre:" );
12        String primerNombre = scanner.nextLine();
13        System.out.println( "Escriba el apellido paterno:" );
14        String apellidoPaterno = scanner.nextLine();
15        System.out.println( "Escriba la direccion:" );
16        String direccion = scanner.nextLine();
17        System.out.println( "Escriba la ciudad:" );
18        String ciudad = scanner.nextLine();
19        System.out.println( "Escriba el estado:" );
20        String estado = scanner.nextLine();
21        System.out.println( "Escriba el codigo postal:" );
22        String cp = scanner.nextLine();
23        System.out.println( "Escriba el telefono:" );
24        String telefono = scanner.nextLine();
25
26        // valida la entrada del usuario y muestra mensaje de error
27        System.out.println( "\nValidar resultado:" );
28
29        if ( !ValidacionEntrada.validarPrimerNombre( primerNombre ) )
30            System.out.println( "Primer nombre invalido" );
31        else if ( !ValidacionEntrada.validarApellidoPaterno( apellidoPaterno ) )
32            System.out.println( "Apellido paterno invalido" );
33        else if ( !ValidacionEntrada.validarDireccion( direccion ) )
34            System.out.println( "Direccion invalida" );
35        else if ( !ValidacionEntrada.validarCiudad( ciudad ) )
36            System.out.println( "Ciudad invalida" );

```

Figura 30.21 | Recibe datos del usuario y los valida mediante la clase ValidacionEntrada. (Parte 1 de 2).


```

37     else if ( !ValidacionEntrada.validarEstado( estado ) )
38         System.out.println( "Estado invalido" );
39     else if ( !ValidacionEntrada.validarCP( cp ) )
40         System.out.println( "Codigo postal invalido" );
41     else if ( !ValidacionEntrada.validarTelefono( telefono ) )
42         System.out.println( "Numero telefonico invalido" );
43     else
44         System.out.println( "La entrada es valida. Gracias." );
45 } // fin de main
46 } // fin de la clase Validacion

```

Escriba el primer nombre:
Jane
 Escriba el apellido paterno:
Doe
 Escriba la direccion:
123 Cierta Calle
 Escriba la ciudad:
Una ciudad
 Escriba el estado:
SS
 Escriba el codigo postal:
123
 Escriba el telefono:
123-456-7890

 Validar resultado:
 Codigo postal invalido

Escriba el primer nombre:
Jane
 Escriba el apellido paterno:
Doe
 Escriba la direccion:
123 Una calle
 Escriba la ciudad:
Una ciudad
 Escriba el estado:
SS
 Escriba el codigo postal:
12345
 Escriba el telefono:
123-456-7890

 Validar resultado:
 La entrada es valida. Gracias.

Figura 30.21 | Recibe datos del usuario y los valida mediante la clase ValidacionEntrada. (Parte 2 de 2).

En la figura 30.20 se valida la entrada del usuario. En la línea 9 se valida el nombre. Para hacer que concuerde un conjunto de caracteres que no tiene una clase predefinida de carácter, utilice los corchetes ([]). Por ejemplo, el patrón "[aeiou]" puede utilizarse para concordar con una sola vocal. Los rangos de caracteres pueden representarse colocando un guión corto (-) entre dos caracteres. En el ejemplo, "[A-Z]" concuerda con una sola letra mayúscula. Si el primer carácter entre corchetes es "^", la expresión acepta cualquier carácter distinto a los que se indiquen. Sin embargo, es importante observar que "[^Z]" no es lo mismo que "[A-Y]", la cual concuerda con las letras mayúsculas A-Y; "[^Z]" concuerda con cualquier carácter distinto de la letra Z mayúscula, incluyendo las letras minúsculas y los caracteres que no son letras, como el carácter de nueva línea. Los rangos en

las clases de caracteres se determinan mediante los valores enteros de las letras. En este ejemplo, "[A-Za-z]" concuerda con todas las letras mayúsculas y minúsculas. El rango "[A-z]" concuerda con todas las letras y también concuerda con los caracteres (como % y 6) que tengan un valor entero entre la letra Z mayúscula y la letra a minúscula (para obtener más información acerca de los valores enteros de los caracteres, consulte el apéndice B, Conjunto de caracteres ASCII). Al igual que las clases predefinidas de caracteres, las clases de caracteres delimitadas entre corchetes concuerdan con un solo carácter en el objeto de búsqueda.

En la línea 9 (figura 30.20), el asterisco después de la segunda clase de carácter indica que puede concordar cualquier número de letras. En general, cuando aparece el operador de expresión regular "*" en una expresión regular, el programa intenta hacer que concuerden cero o más ocurrencias de la subexpresión que va inmediatamente después de "*". El operador "+" intenta hacer que concuerden una o más ocurrencias de la subexpresión que va inmediatamente después de "+". Por lo tanto, "A*" y "A+" concordarán con "AAA", pero sólo "A*" concordará con una cadena vacía.

Si el método `validarPrimerNombre` devuelve `true` (línea 29 de la figura 30.21), la aplicación trata de validar el apellido (línea 31) llamando a `validarApellidoPaterno` (líneas 13 a 16 de la figura 30.20). La expresión regular para validar el apellido concuerda con cualquier número de letras divididas por espacios, apóstrofes o guiones cortos.

En la línea 33 se valida la dirección, llamando al método `validarDireccion` (líneas 19 a 23 de la figura 30.20). La primera clase de carácter concuerda con cualquier dígito una o más veces (`\d+`). Observe que se utilizan dos caracteres `\`, ya que `\` generalmente inicia una secuencia de escape en una cadena. Por lo tanto, `\\d` en una cadena de Java representa al patrón de expresión regular `\d`. Después concordamos uno o más caracteres de espacio en blanco (`\\s+`). El carácter `|` concuerda con la expresión a su izquierda o a su derecha. Por ejemplo, "Hola (Juan | Juana)" concuerda tanto con "Hola Juan" como con "Hola Juana". Los paréntesis se utilizan para agrupar partes de la expresión regular. En este ejemplo, el lado izquierdo de `|` concuerda con una sola palabra y el lado derecho concuerda con dos palabras separadas por cualquier cantidad de espacios en blanco. Por lo tanto, la dirección debe contener un número seguido de una o dos palabras. Por lo tanto, "10 Broadway" y "10 Main Street" son ambas direcciones válidas en este ejemplo. Los métodos `ciudad` (líneas 26 a 29 de la figura 30.20) y `estado` (líneas 32 a 35 de la figura 30.20) también concuerdan con cualquier palabra que tenga al menos un carácter o, de manera alternativa, con dos palabras cualesquiera con al menos un carácter, si éstas van separadas por un solo espacio. Esto significa que tanto `Waltham` como `West Newton` concordarían.

Cuantificadores

El asterisco (*) y el signo de suma (+) se conocen de manera formal como **cuantificadores**. En la figura 30.22 se presentan todos los cuantificadores. Ya hemos visto cómo funcionan el asterisco (*) y el signo de suma (+). Todos los cuantificadores afectan solamente a la subexpresión que va inmediatamente antes del cuantificador. El cuantificador signo de interrogación (?) concuerda con cero o una ocurrencia de la expresión que cuantifica. Un conjunto de llaves que contienen un número ({n}) concuerda exactamente con n ocurrencias de la expresión que cuantifica. En la figura 30.20 mostramos este cuantificador para validar el código postal, en la línea 40. Si se incluye una coma después del número encerrado entre llaves, el cuantificador concordará al menos con n ocurrencias de la expresión cuantificada. El conjunto de llaves que contienen dos números ({n,m}) concuerda entre n y m ocurrencias de la expresión que califica. Los cuantificadores pueden aplicarse a patrones encerrados entre paréntesis para crear expresiones regulares más complejas.

Todos los cuantificadores son **avaros**. Esto significa que concordarán con todas las ocurrencias que puedan, siempre y cuando haya concordancia. No obstante, si alguno de estos cuantificadores va seguido por un signo de interrogación (?), el cuantificador se vuelve **reacio** (o, en algunas ocasiones, **flojo**). De esta forma, concordará con la menor cantidad de ocurrencias posibles, siempre y cuando haya concordancia.

El código postal (línea 40 en la figura 30.20) concuerda con un dígito cinco veces. Esta expresión regular utiliza la clase de carácter de dígito y un cuantificador con el dígito 5 entre llaves. El número telefónico (línea 46 en la figura 30.20) concuerda con tres dígitos (el primero no puede ser cero) seguidos de un guión corto, seguido de tres dígitos más (de nuevo, el primero no puede ser cero), seguidos de cuatro dígitos más.

El método `matches` de `String` verifica si una cadena completa se conforma a una expresión regular. Por ejemplo, queremos aceptar "Smith" como apellido, pero no "9@Smith#". Si sólo una subcadena concuerda con la expresión regular, el método `matches` devuelve `false`.

Cuantificador	Concuerda con
*	Concuerda con cero o más ocurrencias del patrón.
+	Concuerda con una o más ocurrencias del patrón.
?	Concuerda con cero o una ocurrencia del patrón.
{n}	Concuerda con exactamente n ocurrencias.
{n,}	Concuerda con al menos n ocurrencias.
{n,m}	Concuerda con entre n y m (inclusive) ocurrencias.

Figura 30.22 | Cuantificadores utilizados en expresiones regulares.

Reemplazo de subcadenas y división de cadenas

En ocasiones es conveniente reemplazar partes de una cadena, o dividir una cadena en varias piezas. Para este fin, la clase String proporciona los métodos **replaceAll**, **replaceFirst** y **split**. Estos métodos se muestran en la figura 30.23.

```

1 // Fig. 30.23: SustitucionRegex.java
2 // Uso de los métodos replaceFirst, replaceAll y split.
3
4 public class SustitucionRegex
5 {
6     public static void main( String args[] )
7     {
8         String primeraCadena = "Este enunciado termina con 5 estrellas *****";
9         String segundaCadena = "1, 2, 3, 4, 5, 6, 7, 8";
10
11         System.out.printf( "Cadena 1 original: %s\n", primeraCadena );
12
13         // sustituye '*' con '^'
14         primeraCadena = primeraCadena.replaceAll( "\\*", "^" );
15
16         System.out.printf( "^ sustituyen a *: %s\n", primeraCadena );
17
18         // sustituye 'estrellas' con 'intercaladores'
19         primeraCadena = primeraCadena.replaceAll( "estrellas", "intercaladores" );
20
21         System.out.printf(
22             "\"intercaladores\" sustituye a \"estrellas\": %s\n", primeraCadena );
23
24         // sustituye las palabras con 'palabra'
25         System.out.printf( "Cada palabra se sustituye por \"palabra\": %s\n\n",
26             primeraCadena.replaceAll( "\\w+", "palabra" ) );
27
28         System.out.printf( "Cadena 2 original: %s\n", segundaCadena );
29
30         // sustituye los primeros tres dígitos con 'dígito'
31         for ( int i = 0; i < 3; i++ )
32             segundaCadena = segundaCadena.replaceFirst( "\\d", "dígito" );
33
34         System.out.printf(
35             "Los primeros 3 dígitos se sustituyeron por \"dígito\": %s\n", segundaCadena );

```

Figura 30.23 | Métodos replaceFirst, replaceAll y Split. (Parte I de 2).

```

36      String salida = "Cadena dividida en comas: [";
37
38      String[] resultados = segundaCadena.split( "\\s*" ); // se divide en las comas
39
40      for ( String cadena : resultados )
41          salida += "\"" + cadena + "\", "; // imprime resultados
42
43      // elimina la coma adicional y agrega un corchete
44      salida = salida.substring( 0, salida.length() - 2 ) + "]";
45      System.out.println( salida );
46  } // fin de main
47 } // fin de la clase SustitucionRegex

```

Cadena 1 original: Este enunciado termina con 5 estrellas *****
 ^ sustituyen a *: Este enunciado termina con 5 estrellas ^^^^^
 "intercaladores" sustituye a "estrellas": Este enunciado termina con 5 intercaladores ^^^^^
 Cada palabra se sustituye por "palabra": palabra palabra palabra palabra palabra palabra
 ^^^^^

Cadena 2 original: 1, 2, 3, 4, 5, 6, 7, 8
 Los primeros 3 dígitos se sustituyeron por "dígito" : dígito, dígito, dígito, 4, 5, 6, 7, 8
 Cadena dividida en comas: ["dígito", "dígito", "dígito", "4", "5", "6", "7", "8"]

Figura 30.23 | Métodos `replaceFirst`, `replaceAll` y `split`. (Parte 2 de 2).

El método `replaceAll` reemplaza el texto en una cadena con nuevo texto (el segundo argumento) en cualquier parte en donde la cadena original concuerde con una expresión regular (el primer argumento). En la línea 14 se reemplaza cada instancia de "*" en `primeraCadena` con "^". Observe que la expresión regular ("\\s*") coloca dos barras diagonales inversas (\) antes del carácter *. Por lo general, * es un cuantificador que indica que una expresión regular debe concordar con cualquier número de ocurrencias del patrón que se coloca antes de este carácter. Sin embargo, en la línea 14 queremos encontrar todas las ocurrencias del carácter literal *; para ello, debemos evadir el carácter * con el carácter \. Al evadir un carácter especial de expresión regular con una \, indicamos al motor de concordancia de expresiones regulares que busque el carácter en sí. Como la expresión está almacenada en una cadena de Java y \ es un carácter especial en las cadenas de Java, debemos incluir un \ adicional. Por lo tanto, la cadena de Java "\\s*" representa el patrón de expresión regular \s*, que concuerda con un solo carácter * en la cadena de búsqueda. En la línea 19, todas las coincidencias con la expresión regular "estrellas" en `primeraCadena` se reemplazan con "intercaladores".

El método `replaceFirst` (línea 32) reemplaza la primera ocurrencia de la concordancia de un patrón. Las cadenas de Java son inmutables, por lo cual el método `replaceFirst` devuelve una nueva cadena en la que se han reemplazado los caracteres apropiados. Esta línea toma la cadena original y la reemplaza con la cadena devuelta por `replaceFirst`. Al iterar tres veces, reemplazamos las primeras tres instancias de un dígito (\d) en `segundaCadena` con el texto "dígito".

El método `split` divide una cadena en varias subcadenas. La cadena original se divide en cualquier posición que concuerde con una expresión regular especificada. El método `split` devuelve un arreglo de cadenas que contiene las subcadenas que resultan de cada concordancia con la expresión regular. En la línea 38 utilizamos el método `split` para descomponer en tokens una cadena de enteros separados por comas. El argumento es la expresión regular que localiza el delimitador. En este caso, utilizamos la expresión regular "\\s*" para separar las subcadenas siempre que haya una coma. Al concordar con cualquier carácter de espacio en blanco, eliminamos los espacios adicionales de las subcadenas resultantes. Observe que las comas y los espacios en blanco no se devuelven como parte de las subcadenas. De nuevo, observe que la cadena de Java "\\s*" representa la expresión regular, \s*.

Las clases *Pattern* y *Matcher*

Además de las herramientas para el uso de expresiones regulares de la clase `String`, Java proporciona otras clases en el paquete `java.util.regex` que ayudan a los desarrolladores a manipular expresiones regulares. La clase

Pattern representa una expresión regular. La clase **Matcher** contiene tanto un patrón de expresión regular como un objeto **CharSequence** en el que se va a buscar ese patrón.

CharSequence es una interfaz que permite el acceso de lectura a una secuencia de caracteres. Esta interfaz requiere que se declaren los métodos `charAt`, `length`, `subSequence` y `toString`. Tanto **String** como **StringBuilder** implementan la interfaz **CharSequence**, por lo que puede usarse una instancia de cualquiera de estas clases con la clase **Matcher**.



Error común de programación 30.4

*Una expresión regular puede compararse con un objeto de cualquier clase que implemente a la interfaz **CharSequence**, pero la expresión regular debe ser un objeto **String**. Si se intenta crear una expresión regular como un objeto **StringBuilder** se produce un error.*

Si se va a utilizar una expresión regular sólo una vez, puede usarse el método `static matches` de la clase **Pattern**. Este método toma una cadena que especifica la expresión regular y un objeto **CharSequence** en la que se va a realizar la prueba de concordancia. Este método devuelve un valor de tipo `boolean`, el cual indica si el objeto de búsqueda (el segundo argumento) concuerda con la expresión regular.

Si se va a utilizar una expresión regular más de una vez, es más eficiente usar el método `static compile` de la clase **Pattern** para crear un objeto **Pattern** específico de esa expresión regular. Este método recibe una cadena que representa el patrón y devuelve un nuevo objeto **Pattern**, el cual puede utilizarse para llamar al método `matcher`. Este método recibe un objeto **CharSequence** en el que se va a realizar la búsqueda, y devuelve un objeto **Matcher**.

La clase **Matcher** cuenta con el método `matches`, el cual realiza la misma tarea que el método `matches` de **Pattern**, pero no recibe argumentos; el patrón y el objeto de búsqueda están encapsulados en el objeto **Matcher**. La clase **Matcher** proporciona otros métodos, incluyendo `find`, `lookingAt`, `replaceFirst` y `replaceAll`.

En la figura 30.24 presentamos un ejemplo sencillo en el que se utilizan expresiones regulares. Este programa compara las fechas de cumpleaños con una expresión regular. La expresión concuerda sólo con los cumpleaños que no ocurran en abril y que pertenezcan a personas cuyos nombres empiecen con "J".

```

1 // Fig. 30.24: ConcordanciasRegex.java
2 // Demostración de las clases Pattern y Matcher.
3 import java.util.regex.Matcher;
4 import java.util.regex.Pattern;
5
6 public class ConcordanciasRegex
7 {
8     public static void main( String args[] )
9     {
10         // crea la expresión regular
11         Pattern expresion =
12             Pattern.compile( "J.*\\d[0-35-9]-\\d\\d\\d-\\d\\d\\d" );
13
14         String cadena1 = "Jane nacio el 05-12-75\\n" +
15             "Dave nacio el 11-04-68\\n" +
16             "John nacio el 04-28-73\\n" +
17             "Joe nacio el 12-17-77";
18
19         // compara la expresión regular con la cadena e imprime las concordancias
20         Matcher matcher = expresion.matcher( cadena1 );
21
22         while ( matcher.find() )
23             System.out.println( matcher.group() );
24     } // fin de main
25 } // fin de la clase ConcordanciasRegex

```

Figura 30.24 | Expresiones regulares para verificar fechas de nacimiento. (Parte 1 de 2).

```
Jane nacio el 05-12-75
Joe nacio el 12-17-77
```

Figura 30.24 | Expresiones regulares para verificar fechas de nacimiento. (Parte 2 de 2).

En las líneas 11 y 12 se crea un objeto `Pattern` mediante la invocación al método estático `compile` de la clase `Pattern`. El carácter de punto "." en la expresión regular (línea 12) concuerda con cualquier carácter individual, excepto un carácter de nueva línea.

En la línea 20 se crea el objeto `Matcher` para la expresión regular compilada y la secuencia de concordancia (cadena1). En las líneas 22 y 23 se utiliza un ciclo `while` para iterar a través de la cadena. En la línea 22 se utiliza el método `find` de la clase `Matcher` para tratar de hacer que concuerde una pieza del objeto de búsqueda con el patrón de búsqueda. Cada una de las llamadas a este método empieza en el punto en el que terminó la última llamada, por lo que pueden encontrarse varias concordancias. El método `lookingAt` de la clase `Matcher` funciona de manera similar, sólo que siempre comienza desde el principio del objeto de búsqueda, y siempre encontrará la primera concordancia, si es que hay una.



Error común de programación 30.5

El método `matches` (de las clases `String`, `Pattern` o `Matcher`) devuelve `true` sólo si todo el objeto de búsqueda concuerda con la expresión regular. Los métodos `find` y `lookingAt` (de la clase `Matcher`) devuelven `true` si una parte del objeto de búsqueda concuerda con la expresión regular.

En la línea 23 se utiliza el método `group` de la clase `Matcher`, el cual devuelve la cadena del objeto de búsqueda que concuerda con el patrón de búsqueda. La cadena devuelta es la que haya concordado la última vez en una llamada a `find` o `lookingAt`. La salida en la figura 30.24 muestra las dos concordancias que se encontraron en `cadena1`.

Recursos Web sobre expresiones regulares

Los siguientes sitios Web proporcionan más información sobre las expresiones regulares.

java.sun.com/docs/books/tutorial/extra/regex/index.html

Este tutorial explica cómo utilizar el API de expresiones regulares de Java.

java.sun.com/javase/6/docs/api/java/util/regex/package-summary.html

Esta página es el panorama general del paquete `java.util.regex` mediante el uso de `javadoc`.

developer.java.sun.com/developer/technicalArticles/releases/1.4regex

Este sitio incluye una descripción detallada de las herramientas para expresiones regulares del lenguaje Java.

30.8 Conclusión

En este capítulo aprendió acerca de más métodos de `String` para seleccionar porciones de objetos `String` y manipularlos. También aprendió acerca de la clase `Character` y sobre algunos de los métodos que declara para manejar valores `char`. En este capítulo también hablamos sobre las herramientas de la clase `StringBuilder` para crear objetos `String`. En la parte final del capítulo hablamos sobre las expresiones regulares, las cuales proporcionan una poderosa herramienta para buscar y relacionar porciones de objetos `String` que coincidan con un patrón específico.

Resumen

Sección 30.2 Fundamentos de los caracteres y las cadenas

- El valor de una literal de carácter es el valor entero del carácter en el conjunto de caracteres Unicode. Las cadenas pueden incluir letras, dígitos y varios caracteres especiales, tales como `+`, `-`, `*`, `/` y `$`. Una cadena en Java es un objeto de la clase `String`. Las literales de cadena se conocen, por lo regular, como objetos `String`, y se escriben entre comillas dobles en un programa.

Sección 30.3 La clase *String*

- Los objetos *String* son inmutables: los caracteres que contienen no se pueden modificar una vez que se crean.
- El método *length* de *String* devuelve el número de caracteres en un objeto *String*.
- El método *charAt* de *String* devuelve el carácter en una posición específica.
- El método *equals* de *String* compara la igualdad entre dos objetos. Este método devuelve *true* si el contenido de los objetos *String* es igual, y *false* en caso contrario. El método *equals* utiliza una comparación lexicográfica para los objetos *String*.
- Cuando se comparan valores de tipo primitivo con *==*, el resultado es *true* si ambos valores son idénticos. Cuando las referencias se comparan con *==*, el resultado es *true* si ambas referencias son al mismo objeto en memoria.
- Java trata a todas las literales de cadena con el mismo contenido como un solo objeto *String*.
- El método *equalsIgnoreCase* de *String* realiza una comparación de cadenas insensible al uso de mayúsculas y minúsculas.
- El método *compareTo* de *String* usa una comparación lexicográfica y devuelve 0 si las cadenas que está comparando son iguales, un número negativo si la cadena con la que se invoca a *compareTo* es menor que el objeto *String* que recibe como argumento, y un número positivo si la cadena con la que se invoca a *compareTo* es mayor que la cadena que recibe como argumento.
- El método *regionMatches* de *String* compara la igualdad entre porciones de dos cadenas.
- El método *startsWith* de *String* determina si una cadena empieza con los caracteres especificados como argumento. El método *endsWith* de *String* determina si una cadena termina con los caracteres especificados como argumento.
- El método *indexOf* de *String* localiza la primera ocurrencia de un carácter, o de una subcadena en una cadena. El método *lastIndexOf* de *String* localiza la última ocurrencia de un carácter, o de una subcadena en una cadena.
- El método *substring* de *String* copia y devuelve parte de un objeto cadena existente.
- El método *concat* de *String* concatena dos objetos cadena y devuelve un nuevo objeto cadena, que contiene los caracteres de ambas cadenas originales.
- El método *replace* de *String* devuelve un nuevo objeto cadena que reemplaza cada ocurrencia en un objeto *String* de su primer argumento carácter, con su segundo argumento carácter.
- El método *toUpperCase* de *String* devuelve una nueva cadena con letras mayúsculas, en las posiciones en donde la cadena original tenía letras minúsculas. El método *toLowerCase* de *String* devuelve una nueva cadena con letras minúsculas en las posiciones en donde la cadena original tenía letras mayúsculas.
- El método *trim* de *String* devuelve un nuevo objeto cadena, en el que todos los caracteres de espacio en blanco (espacios, nuevas líneas y tabuladores) se eliminan de la parte inicial y la parte final de una cadena.
- El método *toArray* de *String* devuelve un arreglo *char* que contiene una copia de los caracteres de una cadena.
- El método *static valueOf* de *String* devuelve su argumento convertido en una cadena.

Sección 30.4 La clase *StringBuilder*

- La clase *StringBuilder* proporciona constructores que permiten inicializar objetos *StringBuilders* sin caracteres, y con una capacidad inicial de 16 caracteres, sin caracteres y con una capacidad inicial especificada en el argumento entero, o con una copia de los caracteres del argumento *String* y una capacidad inicial equivalente al número de caracteres en el argumento *String*, más 16.
- El método *length* de *StringBuilder* devuelve el número de caracteres actualmente almacenados en un objeto *StringBuilder*. El método *capacity* de *StringBuilder* devuelve el número de caracteres que se pueden almacenar en un objeto *StringBuilder* sin necesidad de asignar más memoria.
- El método *ensureCapacity* de *StringBuilder* asegura que un objeto *StringBuilder* tenga por lo menos la capacidad especificada. El método *setLength* de *StringBuilder* incrementa o decrementa la longitud de un objeto *StringBuilder*.
- El método *charAt* de *StringBuilder* devuelve el carácter que se encuentra en el índice especificado. El método *setCharAt* de *StringBuilder* establece el carácter en la posición especificada. El método *getChars* de *StringBuilder* copia los caracteres que están en el objeto *StringBuilder* y los coloca en el arreglo de caracteres que se pasa como argumento.
- La clase *StringBuilder* proporciona métodos *append* para agregar valores de tipo primitivo, arreglos de caracteres, *String*, *Object* y *CharSequence* al final de un objeto *StringBuilder*. El compilador de Java utiliza los objetos *StringBuilder* y los métodos *append* para implementar los operadores de concatenación *+* y *+=*.
- La clase *StringBuilder* proporciona métodos *insert* sobrecargados para insertar valores de tipo primitivo, arreglos de caracteres, *String*, *Object* y *CharSequence* en cualquier posición en un objeto *StringBuilder*.

Sección 30.5 La clase *Character*

- La clase *Character* proporciona un constructor que recibe un argumento *char*.
- El método *isDefined* de *Character* determina si un carácter está definido en el conjunto de caracteres Unicode. De ser así, el método devuelve *true*; en caso contrario, devuelve *false*.
- El método *isDigit* de *Character* determina si un carácter es un dígito definido en Unicode. De ser así, el método devuelve *true*; en caso contrario devuelve *false*.
- El método *isJavaIdentifierStart* de *Character* determina si un carácter se puede utilizar como el primer carácter de un identificador en Java [es decir, una letra, un guión bajo (*_*) o un signo de dólar (\$)]. De ser así, el método devuelve *true*; en caso contrario, devuelve *false*.
- El método *isJavaIdentifierPart* de *Character* determina si se puede utilizar un carácter en un identificador en Java [es decir, un dígito, una letra, un guión bajo (*_*) o un signo de dólar (\$)]. El método *isLetter* de *Character* determina si un carácter es una letra. El método *isLetterOrDigit* de *Character* determina si un carácter es una letra o un dígito. En cada caso, si es así, el método devuelve *true*; en caso contrario devuelve *false*.
- El método *isLowerCase* de *Character* determina si un carácter es una letra minúscula. El método *isUpperCase* de *Character* determina si un carácter es una letra mayúscula. En ambos casos, de ser así el método devuelve *true*; en caso contrario devuelve *false*.
- El método *toUpperCase* de *Character* convierte un carácter en su equivalente en mayúscula. El método *toLowerCase* convierte un carácter en su equivalente en minúscula.
- El método *digit* de *Character* convierte su argumento carácter en un entero en el sistema numérico especificado por su argumento entero raiz. El método *forDigit* de *Character* convierte su argumento entero dígito en un carácter en el sistema numérico especificado por su argumento entero raiz.
- El método *charValue* de *Character* devuelve el valor *char* almacenado en un objeto *Character*. El método *toString* de *Character* devuelve una representación *String* de un objeto *Character*.

Sección 30.6 La clase *StringTokenizer*

- El constructor predeterminado de *StringTokenizer* crea un objeto *StringTokenizer* para su argumento cadena que utilizará la cadena delimitadora predeterminada "\t\n\r\f", la cual consiste en un espacio, un tabulador, un carácter de nueva línea y un retorno de carro, para dividir la cadena en tokens.
- El método *countTokens* de *StringTokenizer* devuelve el número de tokens en una cadena que se va a dividir en tokens.
- El método *hasMoreTokens* de *StringTokenizer* determina si hay más tokens en la cadena que se va a dividir en tokens.
- El método *nextToken* de *StringTokenizer* devuelve un objeto *String* con el siguiente token.

Sección 30.7 Expresiones regulares, la clase *Pattern* y la clase *Matcher*

- Las expresiones regulares son secuencias de caracteres y símbolos que definen un conjunto de cadenas. Son útiles para validar la entrada y asegurar que los datos se encuentren en un formato específico.
- El método *matches* de *String* recibe una cadena que especifica una expresión regular y relaciona el contenido del objeto *String* en el que se llama con la expresión regular. El método devuelve un valor de tipo *boolean*, el cual indica si hubo concordancia o no.
- Una clase de carácter es una secuencia de escape que representa a un grupo de caracteres. Cada clase de carácter concuerda con un solo carácter en la cadena que estamos tratando de igualar con la expresión regular.
- Un carácter de palabra (*\w*) es cualquier letra (mayúscula o minúscula), dígito o el carácter de guión bajo.
- Un carácter de espacio en blanco (*\s*) es un espacio, un tabulador, un retorno de carro, un carácter de nueva línea o un avance de página.
- Un dígito (*\d*) es cualquier carácter numérico.
- Para relacionar un conjunto de caracteres que no tienen una clase de carácter predefinida, use corchetes (*[]*). Para representar los rangos, coloque un guión corto (*-*) entre dos caracteres. Si el primer carácter en los corchetes es "*^*", la expresión acepta a cualquier carácter distinto de los que se indican.
- Cuando aparece el operador "***" en una expresión regular, el programa trata de relacionar cero o más ocurrencias de la subexpresión que está justo antes del "***".
- El operador "*+*" trata de relacionar una o más ocurrencias de la subexpresión que está antes de éste.
- El carácter "*|*" permite una concordancia de la expresión a su izquierda o a su derecha.
- Los paréntesis (*()*) se utilizan para agrupar partes de la expresión regular.
- El asterisco (***) y el signo positivo (*+*) se conocen formalmente como cuantificadores.
- Todos los cuantificadores afectan sólo a la subexpresión que va justo antes del cuantificador.
- El cuantificador signo de interrogación (*?*) concuerda con cero o una ocurrencias de la expresión que cuantifica.

- Un conjunto de llaves que contienen un número ($\{n\}$) concuerda exactamente con n ocurrencias de la expresión que cuantifica. Si se incluye una coma después del número encerrado entre llaves, concuerda con al menos n ocurrencias de la expresión cuantificada.
- Un conjunto de llaves que contienen dos números ($\{n,m\}$) concuerda con entre n y m ocurrencias de la expresión que califica.
- Todos los cuantificadores son avaros, lo cual significa que concordarán con tantas ocurrencias como puedan, mientras que haya concordancia.
- Si cualquiera de estos cuantificadores va seguido de un signo de interrogación (?), el cuantificador se vuelve renuente, y concuerda con el menor número posible de ocurrencias, mientras que haya concordancia.
- El método `replaceAll` de `String` reemplaza texto en una cadena con nuevo texto (el segundo argumento), en cualquier parte en donde la cadena original concuerde con una expresión regular (el primer argumento).
- Al escapar un carácter de expresión regular especial con una `\`, indicamos al motor de concordancia de expresiones regulares que encuentre el carácter actual, en contraste a lo que representa en una expresión regular.
- El método `replaceFirst` de `String` reemplaza la primera ocurrencia de la concordancia de un patrón. Los objetos `String` de Java son inmutables, por lo cual el método `replaceFirst` devuelve una nueva cadena en la que se han reemplazado los caracteres apropiados.
- El método `split` de `String` divide una cadena en varias subcadenas. La cadena original se divide en cualquier ubicación que concuerde con una expresión regular especificada. El método `split` devuelve un arreglo de cadenas que contienen las subcadenas entre las concordancias para la expresión regular.
- La clase `Pattern` representa a una expresión regular.
- La clase `Matcher` contiene tanto un patrón de expresión regular como un objeto `CharSequence`, en el cual puede buscar el patrón.
- `CharSequence` es una interfaz que permite el acceso de lectura a una secuencia de caracteres. Tanto `String` como `StringBuilder` implementan a la interfaz `CharSequence`, por lo que se puede utilizar una instancia de cualquiera de estas clases con la clase `Matcher`.
- Si una expresión regular se va a utilizar sólo una vez, el método estático `matches` de `Pattern` recibe una cadena que especifica la expresión regular y un objeto `CharSequence` en el que se va a realizar la concordancia. Este método devuelve un valor de tipo `boolean` que indica si el objeto de búsqueda concuerda o no con la expresión regular.
- Si una expresión regular se va a utilizar más de una vez, es más eficiente usar el método estático `compile` de `Pattern` para crear un objeto `Pattern` específico para esa expresión regular. Este método recibe una cadena que representa el patrón y devuelve un nuevo objeto `Pattern`.
- El método `matcher` de `Pattern` recibe un objeto `CharSequence` para realizar la búsqueda y devuelve un objeto `Matcher`.
- El método `matches` de `Matcher` realiza la misma tarea que el método `matches` de `Pattern`, pero no recibe argumentos.
- El método `find` de `Matcher` trata de relacionar una pieza del objeto de la búsqueda con el patrón de búsqueda. Cada llamada a este método empieza en el punto en el que terminó la última llamada, por lo que se pueden encontrar varias concordancias.
- El método `lookingAt` de `Matcher` realiza lo mismo que `find`, excepto que siempre empieza desde el inicio del objeto de búsqueda, y siempre encuentra la primera concordancia, si hay una.
- El método `group` de `Matcher` devuelve la cadena del objeto de búsqueda que concuerda con el patrón de búsqueda. La cadena que se devuelve es la última que concordó mediante una llamada a `find` o a `lookingAt`.

Terminología

`append`, método de la clase `StringBuilder`
 cadena vacía
`capacity`, método de la clase `StringBuilder`
 carácter de palabra
 carácter especial
`charAt`, método de la clase `StringBuilder`
`CharSequence`, interfaz
`charValue`, método de la clase `Character`
 clase de carácter predefinida
 comparación lexicográfica
`concat`, método de la clase `String`

cuantificador avaro
 cuantificador flojo
 cuantificador para expresión regular
 cuantificador renuente
`delete`, método de la clase `StringBuilder`
`deleteCharAt`, método de la clase `String`
 delimitador para tokens
`digit`, método de la clase `Character`
`endsWith`, método de la clase `String`
`ensureCapacity`, método de la clase `StringBuilder`
 expresiones regulares

find, método de la clase `Matcher`
 forDigit, método de la clase `Character`
 getChars, método de la clase `String`
 getChars, método de la clase `StringBuilder`
 hasMoreTokens, método de la clase `StringTokenizer`
 indexOf, método de la clase `String`
 immutable
 isDefined, método de la clase `Character`
 isDigit, método de la clase `Character`
 isJavaIdentifierPart, método de la clase `Character`
 isJavaIdentifierStart, método de la clase `Character`
 isLetter, método de la clase `Character`
 isLetterOrDigit, método de la clase `Character`
 isLowerCase, método de la clase `Character`
 isUpperCase, método de la clase `Character`
 lastIndexOf, método de la clase `String`
 length, método de la clase `String`
 length, método de la clase `StringBuilder`
 literal de cadena
 literal de carácter
 lookingAt, método de la clase `Matcher`
`Matcher`, clase

`matcher`, método de la clase `Pattern`
 matches, método de la clase `Matcher`
 matches, método de la clase `Pattern`
 matches, método de la clase `String`
 nextToken, método de la clase `StringTokenizer`
`Pattern`, clase
 raíz
 regionMatches, método de la clase `String`
 replaceAll, método de la clase `String`
 replaceFirst, método de la clase `String`
 reverse, método de la clase `StringBuilder`
 setCharAt, método de la clase `StringBuilder`
 split, método de la clase `String`
 startsWith, método de la clase `String`
`StringIndexOutOfBoundsException`, clase
 token de un objeto `String`
 toLowerCase, método de la clase `Character`
 toUpperCase, método de la clase `Character`
 trim, método de la clase `StringBuilder`
`Unicode`, conjunto de caracteres
 valueOf, método de la clase `String`

Ejercicios de autoevaluación

- 30.1** Conteste con *verdadero* o *falso* a cada una de las siguientes proposiciones; en caso de ser *falso*, explique por qué.
- Cuando los objetos `String` se comparan utilizando `==`, el resultado es `true` si los objetos `String` contiene los mismos valores.
 - Un objeto `String` puede modificarse una vez creado.
- 30.2** Para cada uno de los siguientes enunciados, escriba una instrucción que realice la tarea indicada:
- Comparar la cadena en `s1` con la cadena en `s2` para ver si su contenido es igual.
 - Anexar la cadena `s2` a la cadena `s1`, utilizando `+=`.
 - Determinar la longitud de la cadena en `s1`.

Respuestas a los ejercicios de autoevaluación

- 30.1**
- Falso. Los objetos `String` se comparan con el operador `==` para determinar si son el mismo objeto en la memoria.
 - Falso. Los objetos `String` son inmutables y no pueden modificarse una vez creados. Los objetos `StringBuilder` sí pueden modificarse una vez creados.
- 30.2**
- `s1.equals(s1)`
 - `s1 += s2;`
 - `s1.length()`

Ejercicios

- 30.3** Escriba una aplicación que utilice el método `compareTo` de la clase `String` para comparar dos cadenas introducidas por el usuario. Muestre si la primera cadena es menor, igual o mayor que la segunda.
- 30.4** Escriba una aplicación que utilice el método `regionMatches` de la clase `String` para comparar dos cadenas introducidas por el usuario. La aplicación deberá recibir como entrada el número de caracteres a comparar y el índice inicial de la comparación. La aplicación deberá indicar si las cadenas son iguales. Ignore si los caracteres están en mayúsculas o en minúsculas al momento de realizar la comparación.
- 30.5** Escriba una aplicación que utilice la generación de números aleatorios para crear enunciados. Use cuatro arreglos de cadenas llamados `articulo`, `sustantivo`, `verbo` y `preposicion`. Cree una oración seleccionando una palabra al azar de cada uno de los arreglos, en el siguiente orden: `articulo`, `sustantivo`, `verbo`, `preposicion`, `articulo` y

sustantivo. A medida que se elija cada palabra, concáténela con las palabras anteriores en el enunciado. Las palabras deberán separarse mediante espacios. Cuando se muestre el enunciado final, deberá empezar con una letra mayúscula y terminar con un punto. El programa deberá generar 20 enunciados y mostrarlos en un área de texto.

30.6 El arreglo de artículos debe contener los artículos "el", "un", "algún" y "ningún"; el arreglo de sustantivos deberá contener los sustantivos "ninio", "ninia", "perro", "ciudad" y "auto"; el arreglo de verbos deberá contener los verbos "manejo", "salto", "corrio", "camino" y "omitio"; el arreglo de preposiciones deberá contener las preposiciones "a", "desde", "encima de", "debajo de" y "sobre".

30.7 Una vez que escriba el programa anterior, modifíquelo para producir una historia breve que consista de varias de estas oraciones (¿qué hay sobre la posibilidad de un escritor de exámenes finales al azar?)

30.8 (*Quintillas*) Una quintilla es un verso humorístico de cinco líneas en el cual la primera y segunda línea riman con la quinta, y la tercera línea rima con la cuarta. Utilizando técnicas similares a las desarrolladas en el ejercicio 30.5, escriba una aplicación en Java que produzca quintillas al azar. Mejorar el programa para producir buenas quintillas es un gran desafío, ¡pero el resultado valdrá la pena!

30.9 (*Latín cerdo*) Escriba una aplicación que codifique frases en español a frases en latín cerdo. El latín cerdo es una forma de lenguaje codificado. Existen muchas variaciones en los métodos utilizados para formar frases en latín cerdo. Por cuestiones de simpleza, utilice el siguiente algoritmo:

Para formar una frase en latín cerdo a partir de una frase en español, divida la frase en palabras con un objeto de la clase `StringTokenizer`. Para traducir cada palabra en español a una palabra en latín cerdo, coloque la primera letra de la palabra en español al final de la palabra, y agregue las letras "ae". De esta forma, la palabra "salta" se convierte a "altasae", la palabra "el" se convierte en "leae" y la palabra "computadora" se convierte en "omputadoracae". Los espacios en blanco entre las palabras permanecen como espacios en blanco. Suponga que la frase en español consiste en palabras separadas por espacios en blanco, que no hay signos de puntuación y que todas las palabras tienen dos o más letras. El método `imprimirPalabraEnLatín` deberá mostrar cada palabra. Cada token devuelto de `nextToken` se pasará al método `imprimirPalabraEnLatín` para imprimir la palabra en latín cerdo. Permita al usuario introducir el enunciado. Use un área de texto para ir mostrando cada uno de los enunciados convertidos.

30.10 Escriba una aplicación que reciba como entrada un número telefónico como una cadena de la forma (555) 555-5555. La aplicación deberá utilizar un objeto de la clase `StringTokenizer` para extraer el código de área como un token, los primeros tres dígitos del número telefónico como otro token y los últimos cuatro dígitos del número telefónico como otro token. Los siete dígitos del número telefónico deberán concatenarse en una cadena. Deberán imprimirse tanto el código de área como el número telefónico. Recuerde que tendrá que modificar los caracteres delimitadores al dividir la cadena en tokens.

30.11 Escriba una aplicación que reciba como entrada una línea de texto, que divida la línea en tokens mediante un objeto de la clase `StringTokenizer` y que muestre los tokens en orden inverso. Use caracteres de espacio como delimitadores.

30.12 Use los métodos de comparación de cadenas que se describieron en este capítulo, junto con las técnicas para ordenar arreglos que se desarrollaron en el capítulo 16 para escribir una aplicación que ordene alfabéticamente una lista de cadenas. Permita al usuario introducir las cadenas en un campo de texto. Muestre los resultados en un área de texto.

30.13 Escriba una aplicación que reciba como entrada una línea de texto y que la imprima dos veces; una vez en letras mayúsculas y otra en letras minúsculas.

30.14 Escriba una aplicación que reciba como entrada una línea de texto y un carácter de búsqueda, y que utilice el método `indexOf` de la clase `String` para determinar el número de ocurrencias de ese carácter en el texto.

30.15 Escriba una aplicación con base en el programa del ejercicio 30.14, que reciba como entrada una línea de texto y utilice el método `indexOf` de la clase `String` para determinar el número total de ocurrencias de cada letra del alfabeto en ese texto. Las letras mayúsculas y minúsculas deben contarse como una sola. Almacene los totales para cada letra en un arreglo, e imprima los valores en formato tabular después de que se hayan determinado los totales.

30.16 Escriba una aplicación que lea una línea de texto, que divida la línea en tokens utilizando caracteres de espacio como delimitadores, y que imprima sólo aquellas palabras que comiencen con la letra "b".

30.17 Escriba una aplicación que lea una línea de texto, que divida la línea en tokens utilizando caracteres de espacio como delimitadores, y que imprima sólo aquellas palabras que comiencen con las letras "ED".

30.18 Escriba una aplicación que reciba como entrada un código entero para un carácter y que muestre el carácter correspondiente. Modifique esta aplicación de manera que genere todos los posibles códigos de tres dígitos en el rango de 000 a 255, y que intente imprimir los caracteres correspondientes.

30.19 Escriba sus propias versiones de los métodos de búsqueda `indexOf` y `lastIndexOf` de la clase `String`.

30.20 Escriba un programa que lea una palabra de cinco letras proveniente del usuario, y que produzca todas las posibles cadenas de tres letras que puedan derivarse de las letras de la palabra con cinco letras. Por ejemplo, las palabras de tres letras producidas a partir de la palabra “trigo” son “rio”, “tio” y “oir”.

Sección especial: manipulación avanzada de cadenas

Los siguientes ejercicios son clave para el libro y están diseñados para evaluar la comprensión del lector sobre los conceptos fundamentales de la manipulación de cadenas. Esta sección incluye una colección de ejercicios intermedios y avanzados de manipulación de cadenas. El lector encontrará estos ejercicios desafiantes, pero divertidos. Los problemas varían considerablemente en dificultad. Algunos requieren una hora o dos para escribir e implementar la aplicación. Otros son útiles como tareas de laboratorio que pudieran requerir dos o tres semanas de estudio e implementación. Algunos son proyectos de fin de curso desafiantes.

30.21 (*Análisis de textos*) La disponibilidad de computadoras con capacidades de manipulación de cadenas ha dado como resultado algunos métodos interesantes para analizar los escritos de grandes autores. Se ha dado mucha importancia para saber si realmente vivió William Shakespeare. Algunos estudiosos creen que existe una gran evidencia que indica que en realidad fue Christopher Marlowe quien escribió las obras maestras que se atribuyen a Shakespeare. Los investigadores han utilizado computadoras para buscar similitudes en los escritos de estos dos autores. En este ejercicio se examinan tres métodos para analizar textos mediante una computadora.

- a) Escriba una aplicación que lea una línea de texto desde el teclado e imprima una tabla que indique el número de ocurrencias de cada letra del alfabeto en el texto. Por ejemplo, la frase:

Ser o no ser: ése es el dilema:

contiene una “a”, ninguna “b”, ninguna “c”, etcétera.

- b) Escriba una aplicación que lea una línea de texto e imprima una tabla que indique el número de palabras de una letra, de dos letras, de tres letras, etcétera, que aparezcan en el texto. Por ejemplo, en la figura 30.25 se muestra la cuenta para la frase:

¿Qué es más noble para el espíritu?

- c) Escriba una aplicación que lea una línea de texto e imprima una tabla que indique el número de ocurrencias de cada palabra distinta en el texto. La primera versión de su programa debe incluir las palabras en la tabla, en el mismo orden en el cual aparecen en el texto. Por ejemplo, las líneas:

Ser o no ser: ése es el dilema:

¿Qué es más noble para el espíritu?

Longitud de palabra	Ocurrencias
1	0
2	2
3	2
4	1
5	1
6	0
7	0
8	1

Figura 30.25 | La cuenta de longitudes de palabras para la cadena “¿Qué es más noble para el espíritu?”.

contiene la palabra “ser” dos veces, La palabra “o” una vez, la palabra “ése” una vez, etcétera. Una muestra más interesante (y útil) podría ser intentar con las palabras ordenadas alfabéticamente.

30.22 (*Impresión de fechas en varios formatos*) Las fechas se imprimen en varios formatos comunes. Dos de los formatos más utilizados son:

04/25/1955 y Abril 25, 1955

Escriba una aplicación que lea una fecha en el primer formato e imprima dicha fecha en el segundo formato.

30.23 (*Protección de cheques*) Las computadoras se utilizan con frecuencia en los sistemas de escritura de cheques, tales como aplicaciones para nóminas y para cuentas por pagar. Existen muchas historias extrañas acerca de cheques de nómina que se imprimen (por error) con montos que se exceden por millones. Los sistemas de emisión de cheques computarizados imprimen cantidades incorrectas debido al error humano o a una falla de la máquina. Los diseñadores de sistemas construyen controles en sus sistemas para evitar la emisión de dichos cheques erróneos.

Otro problema grave es la alteración intencional del monto de un cheque por alguien que planea cobrar un cheque de manera fraudulenta. Para evitar la alteración de un monto, la mayoría de los sistemas computarizados que emiten cheques emplean una técnica llamada protección de cheques. Los cheques diseñados para impresión por computadora contienen un número fijo de espacios en los cuales la computadora puede imprimir un monto. Suponga que un cheque contiene ocho espacios en blanco en los cuales la computadora puede imprimir el monto de un cheque de nómina semanal. Si el monto es grande, entonces se llenarán los ocho espacios. Por ejemplo:

1,230.60 (*monto del cheque*)

12345678 (*números de posición*)

Por otra parte, si el monto es menor de \$1,000, entonces varios espacios quedarían vacíos. Por ejemplo:

99.87

12345678

contiene tres espacios en blanco. Si se imprime un cheque con espacios en blanco, es más fácil para alguien alterar el monto del cheque. Para evitar que se altere el cheque, muchos sistemas de escritura de cheques insertan *asteriscos al principio* para proteger la cantidad, como se muestra a continuación:

***99.87

12345678

Escriba una aplicación que reciba como entrada un monto a imprimir sobre un cheque y que lo escriba mediante el formato de protección de cheques, con asteriscos al principio si es necesario. Suponga que existen nueve espacios disponibles para imprimir el monto.

30.24 (*Escritura en letras del código de un cheque*) Para continuar con la discusión del ejercicio 30.23, reiteramos la importancia de diseñar sistemas de escritura de cheques para evitar la alteración de los montos de los cheques. Un método común de seguridad requiere que el monto del cheque se escriba tanto en números como en letras. Aun cuando alguien pueda alterar el monto numérico del cheque, es extremadamente difícil modificar el monto en letras. Escriba una aplicación que reciba como entrada un monto numérico para el cheque, y que escriba el equivalente del monto en letras. Por ejemplo, el monto 112.43 debe escribirse como

CIENTO DOCE CON 43/100

30.25 (*Clave Morse*) Quizá el más famoso de todos los esquemas de codificación es el código Morse, desarrollado por Samuel Morse en 1832 para usarlo con el sistema telegráfico. El código Morse asigna una serie de puntos y guiones a cada letra del alfabeto, cada dígito y algunos caracteres especiales (tales como el punto, la coma, los dos puntos y el punto y coma). En los sistemas orientados a sonidos, el punto representa un sonido corto y el guión representa un sonido largo. Otras representaciones de puntos y guiones se utilizan en los sistemas orientados a luces y sistemas de señalización con banderas. La separación entre palabras se indica mediante un espacio o, simplemente, con la ausencia de un punto o un guión. En un sistema orientado a sonidos, un espacio se indica por un tiempo breve durante el cual no se transmite sonido alguno. La versión internacional del código Morse aparece en la figura 30.26.

Escriba una aplicación que lea una frase en español y que codifique la frase en clave Morse. Además, escriba una aplicación que lea una frase en código Morse y que la convierta en su equivalente en español. Use un espacio en blanco entre cada letra en clave Morse, y tres espacios en blanco entre cada palabra en clave Morse.

Carácter	Código	Carácter	Código
A	.-	T	-
B	-...	U	..-
C	-.-	V	...-
D	-..	W	.-.-
E	.	X	-.--
F	..-	Y	-.--
G	--.	Z	--..
H		
I	..	<i>Dígitos</i>	
J	.---	1	.----
K	-.-	2	..----
L	.-..	3	...----
M	--	4----
N	-.	5
O	---	6	-.....
P	.-.-	7	--.....
Q	--.-	8	---.....
R	.-.	9	----.....
S	...	0	-----

Figura 30.26 | Las letras del alfabeto expresadas en código Morse internacional.

30.26 (*Aplicación de conversión al sistema métrico*) Escriba una aplicación que ayude al usuario a realizar conversiones métricas. Su aplicación debe permitir al usuario especificar los nombres de las unidades como cadenas (es decir, centímetros, litros, gramos, etcétera, para el sistema métrico, y pulgadas, cuartos, libras, etcétera, para el sistema inglés) y debe responder a preguntas simples tales como:

“¿Cuántas pulgadas hay en 2 metros?”
 “¿Cuántos litros hay en 10 cuartos?”

Su programa debe reconocer conversiones inválidas. Por ejemplo, la pregunta:

“¿Cuántos pies hay en 5 kilogramos?”

no es correcta, debido a que los "pies" son unidades de longitud, mientras que los "kilogramos" son unidades de masa.

Sección especial: proyectos desafiantes de manipulación de cadenas

30.27 (*Proyecto: un corrector ortográfico*) Muchos paquetes populares de software de procesamiento de palabras cuentan con correctores ortográficos integrados. En este proyecto usted debe desarrollar su propia herramienta de corrección ortográfica. Le haremos unas sugerencias para ayudarlo a empezar. Sería conveniente que después le agregara más características. Use un diccionario computarizado (si tiene acceso a uno) como fuente de palabras.

¿Por qué escribimos tantas palabras en forma incorrecta? En algunos casos es porque simplemente no conocemos la manera correcta de escribirlas, por lo que tratamos de adivinar lo mejor que podemos. En otros casos, es porque transponemos dos letras (por ejemplo, “perdeterminado” en lugar de “predeterminado”). Algunas veces escribimos una letra doble por accidente (por ejemplo, “útil” en vez de “util”). Otras veces escribimos una tecla que está cerca de la que pretendíamos escribir (por ejemplo, “cunpleaños” en vez de “cumpleaños”), etcétera.

Diseñe e implemente una aplicación de corrección ortográfica en Java. Su aplicación debe mantener un arreglo de cadenas llamado `listaDePalabras`. Permita al usuario introducir estas cadenas. [Nota: en el capítulo 14 presentamos el procesamiento de archivos. Con esta capacidad, puede obtener las palabras para el corrector ortográfico de un diccionario computarizado almacenado en un archivo].

Su aplicación debe pedir al usuario que introduzca una palabra. La aplicación debe entonces buscar esa palabra en el arreglo `listaDePalabras`. Si la palabra se encuentra en el arreglo, su aplicación deberá imprimir “La palabra está escrita correctamente”. Si la palabra no se encuentra en el arreglo, su aplicación debe imprimir “La palabra no está escrita correctamente”. Después su aplicación debe tratar de localizar otras palabras en la `listaDePalabras` que puedan ser la palabra que el usuario trataba de escribir. Por ejemplo, puede probar con todas las transposiciones simples posibles de letras adyacentes para descubrir que la palabra “predeterminado” concuerda directamente con una palabra en `listaDePalabras`. Desde luego que esto implica que su programa comprobará todas las otras transposiciones posibles, como “rpedeterminado”, “perdeterminado”, “predetreminado”, “predetemrinado” y “predetermniado”. Cuando encuentre una nueva palabra que concuerde con una en la `listaDePalabras`, imprima esa palabra en un mensaje como

“¿Quiso decir “predeterminado”?”.

Lleve a cabo otras pruebas, como reemplazar cada letra doble con una sola letra y cualquier otra prueba que pueda desarrollar para aumentar el valor de su corrector ortográfico.

30.28 (*Proyecto: un generador de crucigramas*) La mayoría de las personas han resuelto crucigramas, pero pocos han intentado generar uno. Aquí lo sugerimos como un proyecto de manipulación de cadenas que requiere una cantidad considerable de sofisticación y esfuerzo.

Hay muchas cuestiones que el programador tiene que resolver para hacer que funcione incluso hasta la aplicación generador de crucigramas más simple. Por ejemplo, ¿cómo representaría la cuadrícula de un crucigrama dentro de la computadora? ¿Debería utilizar una serie de cadenas o arreglos bidimensionales?

El programador necesita una fuente de palabras (es decir, un diccionario computarizado) a la que la aplicación pueda hacer referencia de manera directa. ¿De qué manera deben almacenarse estas palabras para facilitar las manipulaciones complejas que requiere la aplicación?

Si usted es realmente ambicioso, querrá generar la porción de “claves” del crucigrama, en la que se imprimen pistas breves para cada palabra “horizontal” y cada palabra “vertical”. La sola impresión de la versión del crucigrama en blanco no es una tarea fácil.

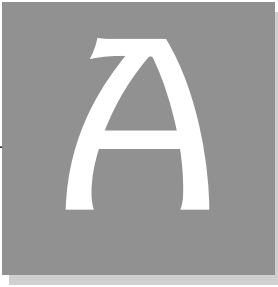


Tabla de precedencia de los operadores

A.1 Precedencia de operadores

Los operadores se muestran en orden decreciente de precedencia, de arriba hacia abajo (figura A.1).

Operador	Descripción	Asociatividad
++ --	unario de postincremento unario de postdecremento	de derecha a izquierda
++ -- + - ! ~ (tipo)	unario de preincremento unario de predecremento unario de suma unario de resta unario de negación lógica unario de complemento a nivel de bits unario de conversión	de derecha a izquierda
* / %	multiplicación división residuo	de izquierda a derecha
+ -	suma o concatenación de cadenas resta	de izquierda a derecha
<< >> >>> < <= > >= instanceof	desplazamiento a la izquierda desplazamiento a la derecha con signo desplazamiento a la derecha sin signo menor que menor o igual que mayor que mayor o igual que comparación de tipos	de izquierda a derecha

Figura A.1 | Tabla de precedencia de los operadores. (Parte 1 de 2).

Operador	Descripción	Asociatividad
==	es igual a	de izquierda a derecha
!=	no es igual a	
&	AND a nivel de bits AND lógico booleano	de izquierda a derecha
^	OR excluyente a nivel de bits OR excluyente lógico booleano	de izquierda a derecha
	OR incluyente a nivel de bits OR incluyente lógico booleano	de izquierda a derecha
&&	AND condicional	de izquierda a derecha
	OR condicional	de izquierda a derecha
?:	condicional	de derecha a izquierda
=	asignación	de derecha a izquierda
+=	asignación, suma	
-=	asignación, resta	
*=	asignación, multiplicación	
/=	asignación, división	
%=	asignación, residuo	
&=	asignación, AND a nivel de bits	
^=	asignación, OR excluyente a nivel de bits	
=	asignación, OR incluyente a nivel de bits	
<<=	asignación, desplazamiento a la izquierda a nivel de bits	
>>=	asignación, desplazamiento a la derecha a nivel de bits con signo	
>>>=	asignación, desplazamiento a la derecha a nivel de bits sin signo	

Figura A.1 | Tabla de precedencia de los operadores. (Parte 2 de 2).

B

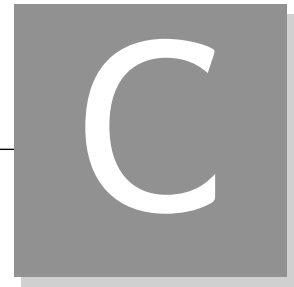
Conjunto de caracteres ASCII

	0	1	2	3	4	5	6	7	8	9
0	nul	soh	stx	ext	eot	enq	ack	bel	bs	ht
1	n1	vt	ff	cr	so	si	d1e	dc1	dc2	dc3
2	dc4	nak	syn	etb	can	em	sub	esc	fs	gs
3	rs	us	sp	!	"	#	\$	%	&	'
4	()	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[\]	^	_	'	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{		}	~	del		

Figura B.1 | El conjunto de caracteres ASCII.

Los dígitos a la izquierda de la tabla son los dígitos izquierdos del equivalente decimal (0-127) del código de caracteres, y los dígitos en la parte superior de la tabla son los dígitos derechos del código de caracteres. Por ejemplo, el código de carácter para la “F” es 70, mientras que para el “&” es 38.

La mayoría de los usuarios de este libro estarán interesados en el conjunto de caracteres ASCII utilizado para representar los caracteres del idioma español en muchas computadoras. El conjunto de caracteres ASCII es un subconjunto del conjunto de caracteres Unicode utilizado por Java para representar caracteres de la mayoría de los lenguajes existentes en el mundo. Para obtener más información acerca del conjunto de caracteres Unicode, vea el apéndice I, Unicode®, que se incluye como bono Web.



Palabras clave y palabras reservadas

Palabras clave en Java				
abstract	assert	boolean	break	byte
case	catch	char	class	continue
default	do	double	else	enum
extends	final	finally	float	for
if	implements	import	instanceof	int
interface	long	native	new	package
private	protected	public	return	short
static	strictfp	super	switch	synchronized
this	throw	throws	transient	try
void	volatile	while		
<i>Palabras clave que no se utilizan actualmente</i>				
const	goto			

Figura C.1 | Palabras clave de Java.

Java también contiene las palabras reservadas `true` y `false`, las cuales son literales boolean, y `null`, que es la literal que representa una referencia a nada. Al igual que las palabras clave, esas palabras reservadas no se pueden utilizar como identificadores.



Tipos primitivos

Tipo	Tamaño en bits	Valores	Estándar
boolean		true o false	
[Nota: una representación boolean es específica para la Máquina virtual de Java en cada plataforma].			
char	16	'\u0000 ' a '\uFFFF ' (0 a 65535)	(ISO, conjunto de caracteres Unicode)
byte	8	−128 a +127 (-2^7 a $2^7 - 1$)	
short	16	−32,768 a +32,767 (-2^{15} a $2^{15} - 1$)	
int	32	−2,147,483,648 a +2,147,483,647 (-2^{31} a $2^{31} - 1$)	
long	64	−9,223,372,036,854,775,808 a +9,223,372,036,854,775,807 (-2^{63} a $2^{63} - 1$)	(IEEE 754, punto flotante)
float	32	Rango negativo: −3.4028234663852886E+38 a −1.40129846432481707e−45 Rango positivo: 1.40129846432481707e−45 a 3.4028234663852886E+38	
double	64	Rango negativo: −1.7976931348623157E+308 a −4.94065645841246544e−324 Rango positivo: 4.94065645841246544e−324 a 1.7976931348623157E+308	

Figura D.1 | Tipos primitivos de Java.

Para obtener más información acerca de IEEE 754, visite grouper.ieee.org/groups/754/. Para obtener más información sobre Unicode, vea el apéndice I, Unicode®.



He aquí sólo los números ratificados.

—William Shakespeare

La naturaleza tiene un cierto tipo de sistema de coordenadas aritméticas-geométricas, ya que cuenta con todo tipo de modelos. Lo que experimentamos de la naturaleza está en los modelos, y todos los modelos de la naturaleza son tan bellos.

Se me ocurrió que el sistema de la naturaleza debe ser una verdadera belleza, porque en la química encontramos que las asociaciones se encuentran siempre en hermosos números enteros; no hay fracciones.

—Richard Buckminster Fuller

Sistemas numéricos

OBJETIVOS

En este apéndice aprenderá a:

- Comprender los conceptos acerca de los sistemas numéricos como base, valor posicional y valor simbólico.
- Trabajar con los números representados en los sistemas numéricos binario, octal y hexadecimal.
- Abreviar los números binarios como octales o hexadecimales.
- Convertir los números octales y hexadecimales en binarios.
- Realizar conversiones hacia y desde números decimales y sus equivalentes en binario, octal y hexadecimal.
- Comprender el funcionamiento de la aritmética binaria y la manera en que se representan los números binarios negativos, utilizando la notación de complemento a dos.

- E.1 Introducción
- E.2 Abreviatura de los números binarios como números octales y hexadecimales
- E.3 Conversión de números octales y hexadecimales a binarios
- E.4 Conversión de un número binario, octal o hexadecimal a decimal
- E.5 Conversión de un número decimal a binario, octal o hexadecimal
- E.6 Números binarios negativos: notación de complemento a dos

Resumen | Terminología | Ejercicios de autoevaluación | Respuestas a los ejercicios de autoevaluación | Ejercicios

E.1 Introducción

En este apéndice presentaremos los sistemas numéricos clave que utilizan los programadores de Java, especialmente cuando trabajan en proyectos de software que requieren de una estrecha interacción con el hardware a nivel de máquina. Entre los proyectos de este tipo están los sistemas operativos, el software de redes computacionales, los compiladores, sistemas de bases de datos y aplicaciones que requieren de un alto rendimiento.

Cuando escribimos un entero, como 227 o -63, en un programa de Java, se asume que el número está en el sistema numérico decimal (base 10). Los dígitos en el sistema numérico decimal son 0, 1, 2, 3, 4, 5, 6, 7, 8 y 9. El dígito más bajo es el 0 y el más alto es el 9 (uno menos que la base, 10). En su interior, las computadoras utilizan el sistema numérico binario (base 2). Este sistema numérico sólo tiene dos dígitos: 0 y 1. El dígito más bajo es el 0 y el más alto es el 1 (uno menos que la base, 2).

Como veremos, los números binarios tienden a ser mucho más extensos que sus equivalentes decimales. Los programadores que trabajan con lenguajes ensambladores y en lenguajes de alto nivel como Java, que les permiten llegar hasta el nivel de máquina, encuentran que es complicado trabajar con números binarios. Por eso existen otros dos sistemas numéricos, el sistema numérico octal (base 8) y el sistema numérico hexadecimal (base 16), que son populares debido a que permiten abreviar los números binarios de una manera conveniente.

En el sistema numérico octal, los dígitos utilizados son del 0 al 7. Debido a que tanto el sistema numérico binario como el octal tienen menos dígitos que el sistema numérico decimal, sus dígitos son los mismos que sus correspondientes en decimal.

El sistema numérico hexadecimal presenta un problema, ya que requiere de 16 dígitos: el dígito más bajo es 0 y el más alto tiene un valor equivalente al 15 decimal (uno menos que la base, 16). Por convención utilizamos las letras de la A a la F para representar los dígitos hexadecimales que corresponden a los valores decimales del 10 al 15. Por lo tanto, en hexadecimal podemos tener números como el 876, que consisten solamente de dígitos similares a los decimales; números como 8A55F que consisten de dígitos y letras; y números como FFE que consisten solamente de letras. En ocasiones un número hexadecimal puede coincidir con una palabra común como FACE o FEED (en inglés); esto puede parecer extraño para los programadores acostumbrados a trabajar con números. Los dígitos de los sistemas numéricos binario, octal, decimal y hexadecimal se sintetizan en las figuras E.1 y E.2.

Cada uno de estos sistemas numéricos utilizan la notación posicional: cada posición en la que se escribe un dígito tiene un valor posicional distinto. Por ejemplo, en el número decimal 937 (el 9, el 3 y el 7 se conocen como valores simbólicos) decimos que el 7 se escribe en la posición de las unidades; el 3, en la de las decenas; y el 9, en la de las centenas. Observe que cada una de estas posiciones es una potencia de la base (10) y que estas potencias empiezan en 0 y aumentan de 1 en 1 a medida que nos desplazamos hacia la izquierda por el número (figura E.3).

Para números decimales más extensos, las siguientes posiciones a la izquierda serían: de millares (10 a la tercera potencia), de decenas de millares (10 a la cuarta potencia), de centenas de millares (10 a la quinta potencia), de los millones (10 a la sexta potencia), de decenas de millones (10 a la séptima potencia), y así sucesivamente.

En el número binario 101 decimos que el 1 más a la derecha se escribe en la posición de los unos, el 0 se escribe en la posición de los dos y el 1 de más a la izquierda se escribe en la posición de los cuatros. Observe que cada una de estas posiciones es una potencia de la base (2) y que estas potencias empiezan en 0 y aumentan de 1 en 1 a medida que nos desplazamos hacia la izquierda por el número (figura E.4). Por lo tanto, $101 = 2^2 + 2^0 = 4 + 1 = 5$.

Para números binarios más extensos, las siguientes posiciones a la izquierda serían la posición de los ochos (2 a la tercera potencia), la posición de los dieciséis (2 a la cuarta potencia), la posición de los treinta y dos (2 a la quinta potencia), la posición de los sesenta y cuatros (2 a la sexta potencia), y así sucesivamente.

Dígito binario	Dígito octal	Dígito decimal	Dígito hexadecimal
0	0	0	0
1	1	1	1
	2	2	2
	3	3	3
	4	4	4
	5	5	5
	6	6	6
	7	7	7
		8	8
		9	9
			A (valor de 10 en decimal)
			B (valor de 11 en decimal)
			C (valor de 12 en decimal)
			D (valor de 13 en decimal)
			E (valor de 14 en decimal)
			F (valor de 15 en decimal)

Figura E.1 | Dígitos de los sistemas numéricos binario, octal, decimal y hexadecimal.

Atributo	Binario	Octal	Decimal	Hexadecimal
Base	2	8	10	16
Dígito más bajo	0	0	0	0
Dígito más alto	1	7	9	F

Figura E.2 | Comparación de los sistemas binario, octal, decimal y hexadecimal.

Valores posicionales en el sistema numérico decimal			
Dígito decimal	9	3	7
Nombre de la posición	Centenas	Decenas	Unidades
Valor posicional	100	10	1
Valor posicional como potencia de la base (10)	10^2	10^1	10^0

Figura E.3 | Valores posicionales en el sistema numérico decimal.

En el número octal 425, decimos que el 5 se escribe en la posición de los unos, el 2 se escribe en la posición de los ochos y el 4 se escribe en la posición de los sesenta y cuatros. Observe que cada una de estas posiciones es una potencia de la base (8) y que estas potencias empiezan en 0 y aumentan de 1 en 1 a medida que nos desplazamos hacia la izquierda por el número (figura E.5).

Para números octales más extensos, las siguientes posiciones a la izquierda sería la posición de los quinientos doces (8 a la tercera potencia), la posición de los cuatro mil noventa y seis (8 a la cuarta potencia), la posición de los treinta y dos mil setecientos sesenta y ochos (8 a la quinta potencia), y así sucesivamente.

Valores posicionales en el sistema numérico binario			
Dígito binario	1	0	1
Nombre de la posición	Cuatro	Dos	Unos
Valor posicional	4	2	1
Valor posicional como potencia de la base (2)	2^2	2^1	2^0

Figura E.4 | Valores posicionales en el sistema numérico binario.

Valores posicionales en el sistema numérico octal			
Dígito octal	4	2	5
Nombre de la posición	Sesenta y cuatros	Ochos	Unos
Valor posicional	64	8	1
Valor posicional como potencia de la base (8)	8^2	8^1	8^0

Figura E.5 | Valores posicionales en el sistema numérico octal.

Valores posicionales en el sistema numérico hexadecimal			
Dígito hexadecimal	3	D	A
Nombre de la posición	Doscientos cincuenta y seis	Dieciséis	Unos
Valor posicional	256	16	1
Valor posicional como potencia de la base (16)	16^2	16^1	16^0

Figura E.6 | Valores posicionales en el sistema numérico hexadecimal.

En el número hexadecimal 3DA, decimos que la A se escribe en la posición de los unos, la D se escribe en la posición de los dieciséis y el 3 se escribe en la posición de los doscientos cincuenta y seis. Observe que cada una de estas posiciones es una potencia de la base (16) y que estas potencias empiezan en 0 y aumentan de 1 en 1 a medida que nos desplazamos hacia la izquierda por el número (figura E.6).

Para números hexadecimales más extensos, las siguientes posiciones a la izquierda serían la posición de los cuatro mil noventa y seis (16 a la tercera potencia), la posición de los sesenta y cinco mil quinientos treinta y seis (16 a la cuarta potencia), y así sucesivamente.

E.2 Abreviatura de los números binarios como números octales y hexadecimales

En computación, el uso principal de los números octales y hexadecimales es para abreviar representaciones binarias demasiado extensas. La figura E.7 muestra que los números binarios extensos pueden expresarse más concisamente en sistemas numéricos con bases mayores que en el sistema numérico binario.

Una relación especialmente importante que tienen tanto el sistema numérico octal como el hexadecimal con el sistema binario es que las bases de los sistemas octal y hexadecimal (8 y 16, respectivamente) son potencias de la base del sistema numérico binario (base 2). Considere el siguiente número binario de 12 dígitos y sus equivalentes en octal y hexadecimal. Vea si puede determinar cómo esta relación hace que sea conveniente el abreviar los números binarios en octal o hexadecimal. La respuesta sigue después de los números.

Número decimal	Representación binaria	Representación octal	Representación hexadecimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10

Figura E.7 | Equivalentes en decimal, binario, octal y hexadecimal.

Número binario	Equivalente en octal	Equivalente en hexadecimal
100011010001	4321	8D1

Para convertir fácilmente el número binario en octal, sólo divida el número binario de 12 dígitos en grupos de tres bits consecutivos y escriba esos grupos por encima de los dígitos correspondientes del número octal, como se muestra a continuación:

100	011	010	001
4	3	2	1

Observe que el dígito octal que escribió debajo de cada grupo de tres bits corresponde precisamente al equivalente octal de ese número binario de 3 dígitos que se muestra en la figura E.7.

El mismo tipo de relación puede observarse al convertir números de binario a hexadecimal. Divida el número binario de 12 dígitos en grupos de cuatro bits consecutivos y escriba esos grupos por encima de los dígitos correspondientes del número hexadecimal, como se muestra a continuación:

1000	1101	0001
8	D	1

Observe que el dígito hexadecimal que escribió debajo de cada grupo de cuatro bits corresponde precisamente al equivalente hexadecimal de ese número binario de 4 dígitos que se muestra en la figura E.7.

E.3 Conversión de números octales y hexadecimales a binarios

En la sección anterior vimos cómo convertir números binarios a sus equivalentes en octal y hexadecimal, formando grupos de dígitos binarios y simplemente volviéndolos a escribir como sus valores equivalentes en dígitos octales o hexadecimales. Este proceso puede utilizarse en forma inversa para producir el equivalente en binario de un número octal o hexadecimal.

Por ejemplo, el número octal 653 se convierte en binario simplemente escribiendo el 6 como su equivalente binario de 3 dígitos 110, el 5 como su equivalente binario de 3 dígitos 101 y el 3 como su equivalente binario de 3 dígitos 011 para formar el número binario de 9 dígitos 110101011.

El número hexadecimal FAD5 se convierte en binario simplemente escribiendo la F como su equivalente binario de 4 dígitos 1111, la A como su equivalente binario de 4 dígitos 1010, la D como su equivalente binario de 4 dígitos 1101 y el 5 como su equivalente binario de 4 dígitos 0101, para formar el número binario de 16 dígitos 1111101011010101.

E.4 Conversión de un número binario, octal o hexadecimal a decimal

Como estamos acostumbrados a trabajar con el sistema decimal, a menudo es conveniente convertir un número binario, octal o hexadecimal en decimal para tener una idea de lo que “realmente” vale el número. Nuestros diagramas en la sección E.1 expresan los valores posicionales en decimal. Para convertir un número en decimal desde otra base, multiplique el equivalente en decimal de cada dígito por su valor posicional y sume estos productos. Por ejemplo, el número binario 110101 se convierte en el número 53 decimal, como se muestra en la figura E.8.

Para convertir el número 7614 octal en el número 3980 decimal utilizamos la misma técnica, esta vez utilizando los valores posicionales apropiados para el sistema octal, como se muestra en la figura E.9.

Para convertir el número AD3B hexadecimal en el número 44347 decimal utilizamos la misma técnica, esta vez empleando los valores posicionales apropiados para el sistema hexadecimal, como se muestra en la figura E.10.

Conversión de un número binario en decimal						
Valores posicionales:	32	16	8	4	2	1
Valores simbólicos:	1	1	0	1	0	1
Productos:	$1 \times 32 = 32$	$1 \times 16 = 16$	$0 \times 8 = 0$	$1 \times 4 = 4$	$0 \times 2 = 0$	$1 \times 1 = 1$
Suma:	$= 32 + 16 + 0 + 4 + 0 + 1 = 53$					

Figura E.8 | Conversión de un número binario en decimal.

Conversión de un número octal en decimal				
Valores posicionales:	512	16	32	58
Valores simbólicos:	7	32	32	32
Productos:	$7 \times 512 = 3584$	$6 \times 64 = 384$	$1 \times 8 = 8$	$4 \times 1 = 4$
Suma:	$= 3584 + 384 + 8 + 4 = 3980$			

Figura E.9 | Conversión de un número octal en decimal.

Conversión de un número hexadecimal; en decimal				
Valores posicionales:	4096	256	16	1
Valores simbólicos:	A	D	3	B
Productos:	$A \times 4096 = 40960$	$D \times 256 = 3328$	$3 \times 16 = 48$	$B \times 1 = 11$
Suma:	$= 40960 + 3328 + 48 + 11 = 44347$			

Figura E.10 | Conversión de un número hexadecimal en decimal.

E.5 Conversión de un número decimal a binario, octal o hexadecimal

Las conversiones de la sección E.4 siguen naturalmente las convenciones de la notación posicional. Las conversiones de decimal a binario, octal o hexadecimal también siguen estas convenciones.

Suponga que queremos convertir el número 57 decimal en binario. Empezamos escribiendo los valores posicionales de las columnas de derecha a izquierda, hasta llegar a una columna cuyo valor posicional sea mayor que el número decimal. Como no necesitamos esa columna, podemos descartarla. Por lo tanto, primero escribimos:

Valores posicionales:	64	32	16	8	4	2	1
-----------------------	----	----	----	---	---	---	---

Luego descartamos la columna con el valor posicional de 64, dejando:

Valores posicionales:	32	16	8	4	2	1
-----------------------	----	----	---	---	---	---

A continuación, empezamos a trabajar desde la columna más a la izquierda y nos vamos desplazando hacia la derecha. Dividimos 57 entre 32 y observamos que hay un 32 en 57, con un residuo de 25, por lo que escribimos 1 en la columna de los 32. Dividimos 25 entre 16 y observamos que hay un 16 en 25, con un residuo de 9, por lo que escribimos 1 en la columna de los 16. Dividimos 9 entre 8 y observamos que hay un 8 en 9 con un residuo de 1. Las siguientes dos columnas producen el cociente de cero cuando se divide 1 entre sus valores posicionales, por lo que escribimos 0 en las columnas de los 4 y de los 2. Por último, 1 entre 1 es 1, por lo que escribimos 1 en la columna de los 1. Esto nos da:

Valores posicionales:	32	16	8	4	2	1
Valores simbólicos:	1	1	1	0	0	1

y, por lo tanto, el 57 decimal es equivalente al 111001 binario.

Para convertir el número decimal 103 en octal, empezamos por escribir los valores posicionales de las columnas hasta llegar a una columna cuyo valor posicional sea mayor que el número decimal. Como no necesitamos esa columna, podemos descartarla. Por lo tanto, primero escribimos:

Valores posicionales:	512	64	8	1
-----------------------	-----	----	---	---

Luego descartamos la columna con el valor posicional de 512, lo que nos da:

Valores posicionales:	64	8	1
-----------------------	----	---	---

A continuación, empezamos a trabajar desde la columna más a la izquierda y nos vamos desplazando hacia la derecha. Dividimos 103 entre 64 y observamos que hay un 64 en 103 con un residuo de 39, por lo que escribimos 1 en la columna de los 64. Dividimos 39 entre 8 y observamos que el 8 cabe cuatro veces en 39 con un residuo de 7, por lo que escribimos 4 en la columna de los 8. Por último, dividimos 7 entre 1 y observamos que el 1 cabe siete veces en 7 y no hay residuo, por lo que escribimos 7 en la columna de los 1. Esto nos da:

Valores posicionales:	64	8	1
Valores simbólicos:	1	4	7

y por lo tanto, el 103 decimal es equivalente al 147 octal.

Para convertir el número decimal 375 en hexadecimal, empezamos por escribir los valores posicionales de las columnas hasta llegar a una columna cuyo valor posicional sea mayor que el número decimal. Como no necesitamos esa columna, podemos descartarla. Por consecuencia, primero escribimos:

Valores posicionales:	4096	256	16	1
-----------------------	------	-----	----	---

Luego descartamos la columna con el valor posicional de 4096, lo que nos da:

Valores posicionales:	256	16	1
-----------------------	-----	----	---

A continuación, empezamos a trabajar desde la columna más a la izquierda y nos vamos desplazando hacia la derecha. Dividimos 375 entre 256 y observamos que 256 cabe una vez en 375 con un residuo de 119, por lo que escribimos 1 en la columna de los 256. Dividimos 119 entre 16 y observamos que el 16 cabe siete veces en 119 con un residuo de 7, por lo que escribimos 7 en la columna de los 16. Por último, dividimos 7 entre 1 y

observamos que el 1 cabe siete veces en 7 y no hay residuo, por lo que escribimos 7 en la columna de los 1. Esto produce:

Valores posicionales:	256	16	1
Valores simbólicos:	1	7	7

y, por lo tanto, el 375 decimal es equivalente al 177 hexadecimal.

E.6 Números binarios negativos: notación de complemento a dos

La discusión en este apéndice se ha enfocado hasta ahora en números positivos. En esta sección explicaremos cómo las computadoras representan números negativos mediante el uso de la notación de *complementos a dos*. Primero explicaremos cómo se forma el complemento a dos de un número binario y después mostraremos por qué representa el valor negativo de dicho número binario.

Considere una máquina con enteros de 32 bits. Suponga que se ejecuta la siguiente instrucción:

```
int valor = 13;
```

La representación en 32 bits de `valor` es:

```
00000000 00000000 00000000 00001101
```

Para formar el negativo de `valor`, primero formamos su *complemento a uno* aplicando el operador de complemento a nivel de bits de Java (`~`):

```
complementoAUnoDeValor = ~valor;
```

Internamente, `~valor` es ahora `valor` con cada uno de sus bits invertidos; los unos se convierten en ceros y los ceros en unos, como se muestra a continuación:

```
valor:
00000000 00000000 00000000 00001101
~valor (es decir, el complemento a uno de valor):
11111111 11111111 11111111 11110010
```

Para formar el complemento a dos de `valor`, simplemente sumamos uno al complemento a uno de `valor`. Por lo tanto:

```
El complemento a dos de valor es:
11111111 11111111 11111111 11110011
```

Ahora, si esto de hecho es igual a -13 , deberíamos poder sumarlo al 13 binario y obtener como resultado 0. Comprobemos esto:

```
00000000 00000000 00000000 00001101
+11111111 11111111 11111111 11110011
-----
00000000 00000000 00000000 00000000
```

El bit de acarreo que sale de la columna que está más a la izquierda se descarta y evidentemente obtenemos 0 como resultado. Si sumamos el complemento a uno de un número a ese mismo número, todos los dígitos del resultado serían iguales a 1. La clave para obtener un resultado en el que todos los dígitos sean cero es que el complemento a dos es 1 más que el complemento a 1. La suma de 1 hace que el resultado de cada columna sea 0 y se acarrea un 1. El acarreo sigue desplazándose hacia la izquierda hasta que se descarta en el bit que está más a la izquierda, con lo que todos los dígitos del número resultante son iguales a cero.

En realidad, las computadoras realizan una suma como:

```
x = a - valor;
```

mediante la suma del complemento a dos de `valor` con `a`, como se muestra a continuación:

```
x = a + (~valor + 1);
```

Suponga que a es 27 y que valor es 13 como en el ejemplo anterior. Si el complemento a dos de valor es en realidad el negativo de éste, entonces al sumar el complemento de dos de valor con a se produciría el resultado de 14. Comprobemos esto:

$$\begin{array}{r}
 a \text{ (es decir, 27)} \quad 00000000 \ 00000000 \ 00000000 \ 00011011 \\
 +(\sim\text{valor} + 1) \quad +11111111 \ 11111111 \ 11111111 \ 11110011 \\
 \hline
 00000000 \ 00000000 \ 00000000 \ 00001110
 \end{array}$$

lo que ciertamente da como resultado 14.

Resumen

- Cuando escribimos un entero como 19, 227 o -63, en un programa de Java, suponemos que el número se encuentra en el sistema numérico decimal (base 10). Los dígitos en el sistema numérico decimal son 0, 1, 2, 3, 4, 5, 6, 7, 8 y 9. El dígito más bajo es el 0 y el más alto es el 9 (uno menos que la base, 10).
- En su interior, las computadoras utilizan el sistema numérico binario (base 2). Este sistema numérico sólo tiene dos dígitos: 0 y 1. El dígito más bajo es el 0 y el más alto es el 1 (uno menos que la base, 2).
- El sistema numérico octal (base 8) y el sistema numérico hexadecimal (base 16) son populares debido a que permiten abreviar los números binarios de una manera conveniente.
- Los dígitos que se utilizan en el sistema numérico octal son del 0 al 7.
- El sistema numérico hexadecimal presenta un problema, ya que requiere de dieciséis dígitos: el dígito más bajo es 0 y el más alto tiene un valor equivalente al 15 decimal (uno menos que la base, 16). Por convención utilizamos las letras de la A a la F para representar los dígitos hexadecimales que corresponden a los valores decimales del 10 al 15.
- Cada uno de estos sistemas numéricos utilizan la notación posicional: cada posición en la que se escribe un dígito tiene un distinto valor posicional.
- Una relación especialmente importante que tienen tanto el sistema numérico octal como el hexadecimal con el sistema binario es que las bases de los sistemas octal y hexadecimal (8 y 16, respectivamente) son potencias de la base del sistema numérico binario (base 2).
- Para convertir un número octal en binario, sustituya cada dígito octal con su equivalente binario de tres dígitos.
- Para convertir un número hexadecimal en binario, simplemente sustituya cada dígito hexadecimal con su equivalente binario de cuatro dígitos.
- Como estamos acostumbrados a trabajar con el sistema decimal, es conveniente convertir un número binario, octal o hexadecimal en decimal para tener una idea de lo que “realmente” vale el número.
- Para convertir un número en decimal desde otra base, multiplique el equivalente en decimal de cada dígito por su valor posicional y sume estos productos.
- Las computadoras representan números negativos mediante el uso de la notación de complementos a dos.
- Para formar el negativo de un valor en binario, primero formamos su complemento a uno aplicando el operador de complemento a nivel de bits de Java (~). Esto invierte los bits del valor. Para formar el complemento a dos de un valor, simplemente sumamos uno al complemento a uno de ese valor.

Terminología

base

conversiones

dígito

notación de complementos a dos

notación de complementos a uno

notación posicional

operador de complemento a nivel de bits (~)

sistema numérico binario

sistema numérico de base 10

sistema numérico de base 16

sistema numérico de base 2

sistema numérico de base 8

sistema numérico decimal

sistema numérico hexadecimal

sistema numérico octal

valor negativo

valor posicional

valor simbólico

Ejercicios de autoevaluación

- E.1** Las bases de los sistemas numéricos decimal, binario, octal y hexadecimal son _____, _____, _____ y _____, respectivamente.
- E.2** En general, las representaciones en decimal, octal y hexadecimal de un número binario dado contienen (más/menos) dígitos de los que contiene el número binario.
- E.3** (*Verdadero/falso*) Una de las razones populares de utilizar el sistema numérico decimal es que forma una notación conveniente para abreviar números binarios, en la que simplemente se sustituye un dígito decimal por cada grupo de cuatro dígitos binarios.
- E.4** La representación (octal/hexadecimal/decimal) de un valor binario grande es la más concisa (de las alternativas dadas).
- E.5** (*Verdadero/falso*) El dígito de mayor valor en cualquier base es uno más que la base.
- E.6** (*Verdadero/falso*) El dígito de menor valor en cualquier base es uno menos que la base.
- E.7** El valor posicional del dígito que se encuentra más a la derecha en cualquier número, ya sea binario, octal, decimal o hexadecimal es siempre _____.
- E.8** El valor posicional del dígito que está a la izquierda del dígito que se encuentra más a la derecha en cualquier número, ya sea binario, octal, decimal o hexadecimal es siempre igual a _____.
- E.9** Complete los valores que faltan en esta tabla de valores posicionales para las cuatro posiciones que están más a la derecha en cada uno de los sistemas numéricos indicados:
- | | | | | |
|-------------|------|-----|-----|-----|
| decimal | 1000 | 100 | 10 | 1 |
| hexadecimal | ... | 256 | ... | ... |
| binario | ... | ... | ... | ... |
| octal | 512 | ... | 8 | ... |
- E.10** Convierta el número binario 110101011000 en octal y en hexadecimal.
- E.11** Convierta el número hexadecimal FACE en binario.
- E.12** Convierta el número octal 7316 en binario.
- E.13** Convierta el número hexadecimal 4FEC en octal. (*Sugerencia:* primero convierta el número 4FEC en binario y después convierta el número resultante en octal).
- E.14** Convierta el número binario 1101110 en decimal.
- E.15** Convierta el número octal 317 en decimal.
- E.16** Convierta el número hexadecimal EFD4 en decimal.
- E.17** Convierta el número decimal 177 en binario, en octal y en hexadecimal.
- E.18** Muestre la representación binaria del número decimal 417. Después muestre el complemento a uno de 417 y el complemento a dos del mismo número.
- E.19** ¿Cuál es el resultado cuando se suma el complemento a dos de un número con ese mismo número?

Respuestas a los ejercicios de autoevaluación

- E.1** 10, 2, 8, 16.
- E.2** Menos.
- E.3** Falso. El hexadecimal hace esto.
- E.4** Hexadecimal.
- E.5** Falso. El dígito de mayor valor en cualquier base es uno menos que la base.
- E.6** Falso. El dígito de menor valor en cualquier base es cero.
- E.7** 1 (La base elevada a la potencia de cero).
- E.8** La base del sistema numérico.
- E.9** Complete los valores que faltan en esta tabla de valores posicionales para las cuatro posiciones que están más a la derecha en cada uno de los sistemas numéricos indicados:

decimal	1000	100	10	1
hexadecimal	4096	256	16	1
binario	8	4	2	1
octal	512	64	8	1

E.10 6530 octal; D58 hexadecimal.

E.11 1111 1010 1100 1110 binario.

E.12 111 011 001 110 binario.

E.13 0 100 111 111 101 100 binario; 47754 octal.

E.14 $2+4+8+32+64=110$ decimal.

E.15 $7+1*8+3*64=7+8+192=207$ decimal.

E.16 $4+13*16+15*256+14*4096=61396$ decimal.

E.17 177 decimal

en binario:

```

256 128 64 32 16 8 4 2 1
128 64 32 16 8 4 2 1
(1*128)+(0*64)+(1*32)+(1*16)+(0*8)+(0*4)+(0*2)+(1*1)
10110001

```

en octal:

```

512 64 8 1
64 8 1
(2*64)+(6*8)+(1*1)
261

```

en hexadecimal:

```

256 16 1
16 1
(11*16)+(1*1)
(B*16)+(1*1)
B1

```

E.18 Binario:

```

512 256 128 64 32 16 8 4 2 1
256 128 64 32 16 8 4 2 1
(1*256)+(1*128)+(0*64)+(1*32)+(0*16)+(0*8)+(0*4)+(0*2)+
(1*1)
110100001

```

Complemento a uno: 001011110

Complemento a dos: 001011111

Comprobación: Número binario original + su complemento a dos:

```

110100001
001011111
-----
000000000

```

E.19 Cero.

Ejercicios

E.20 Algunas personas argumentan que muchos de nuestros cálculos se realizarían más fácilmente en el sistema numérico de base 12, ya que el 12 puede dividirse por muchos más números que el 10 (por la base 10). ¿Cuál es el dígito de menor valor en la base 12? ¿Cuál podría ser el símbolo con mayor valor para un dígito en la base 12? ¿Cuáles son los valores posicionales de las cuatro posiciones más a la derecha de cualquier número en el sistema numérico de base 12?

E.21 Complete la siguiente tabla de valores posicionales para las cuatro posiciones más a la derecha en cada uno de los sistemas numéricos indicados:

decimal	1000	100	10	1
base 6	6	...
base 13	...	169
base 3	27

E.22 Convierta el número binario 100101111010 en octal y en hexadecimal.

E.23 Convierta el número hexadecimal 3A7D en binario.

E.24 Convierta el número hexadecimal 765F en octal. (*Sugerencia:* primero conviértalo en binario y después convierta el número resultante en octal).

E.25 Convierta el número binario 1011110 en decimal.

E.26 Convierta el número octal 426 en decimal.

E.27 Convierta el número hexadecimal FFFF en decimal.

E.28 Convierta el número decimal 299 en binario, en octal y en hexadecimal.

E.29 Muestre la representación binaria del número decimal 779. Después muestre el complemento a uno de 779 y el complemento a dos del mismo número.

E.30 Muestre el complemento a dos del valor entero -1 en una máquina con enteros de 32 bits.

GroupLayout

F.1 Introducción

Java SE 6 incluye un nuevo y poderoso administrador de esquemas llamado **GroupLayout**, el cual es el administrador de esquemas predeterminado en el IDE Netbeans 5.5 (www.netbeans.org). En este apéndice veremos las generalidades acerca de GroupLayout, y después demostraremos cómo usar el **diseñador de GUI Matisse** del IDE Netbeans 5.5 para crear una GUI mediante el uso de GroupLayout para posicionar los componentes. NetBeans genera el código de GroupLayout por el programador de manera automática. Aunque podemos escribir código de GroupLayout en forma manual, en la mayoría de los casos es mejor utilizar una herramienta de diseño de GUI tal como la que proporciona Netbeans, para sacar provecho al poder de GroupLayout. Para obtener más detalles acerca de GroupLayout, consulte la lista de recursos Web al final de este apéndice.

F.2 Fundamentos de GroupLayout

En los capítulos 11 y 22 presentamos varios administradores de esquemas que proporcionan herramientas de esquemas de GUI. También vimos cómo combinar administradores de esquemas y varios contenedores para crear esquemas más complejos. La mayoría de los administradores de esquemas no nos proporcionan un control preciso sobre el posicionamiento de los componentes. En el capítulo 22 vimos GridBagLayout, que proporciona un control más preciso sobre la posición y el tamaño de los componentes de GUI del programador. Nos permite especificar la posición vertical y horizontal de cada componente, el número de filas y columnas que ocupa cada componente en la cuadrícula, y la forma en que los componentes aumentan y reducen su tamaño, a medida que cambia el tamaño del contenedor. Todo esto se especifica al mismo tiempo con un objeto GridBagConstraints. La clase GroupLayout es el siguiente paso en la administración de esquemas. GroupLayout es más flexible, ya que el programador puede especificar los esquemas horizontal y vertical de sus componentes de manera independiente.

Arreglos en serie y en paralelo

Los componentes se ordenan en secuencia o en paralelo. Los tres objetos JButton de la figura F.1 tienen una **orientación horizontal secuencial**: aparecen de izquierda a derecha en secuencia. En sentido vertical, los componentes están ordenados en paralelo, por lo que en cierto sentido, “ocupan el mismo espacio vertical”. Los componentes también se pueden ordenar secuencialmente en dirección vertical, y en paralelo en dirección horizontal, como veremos en la sección F.3. Para evitar traslapar los componentes, por lo general, los componentes con orientación vertical en paralelo tienen una orientación horizontal secuencial (y viceversa).

Grupos y alineación

Para crear interfaces de usuario más complejas, GroupLayout nos permite crear **grupos** que contengan elementos secuenciales o en paralelo. Dentro de un grupo, podemos tener componentes de GUI, otros grupos y huecos.

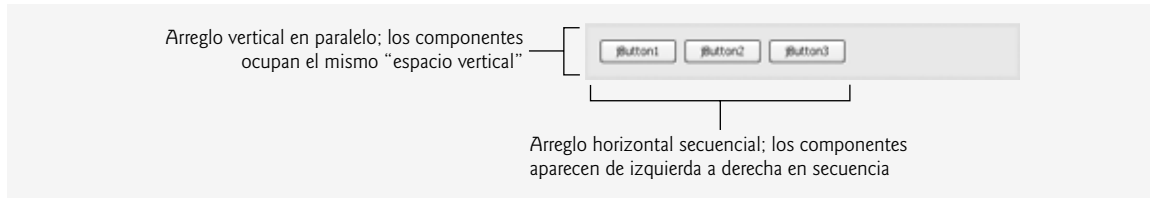


Figura F.1 | Objetos `JButton` ordenados en secuencia para su orientación horizontal, y en paralelo para su orientación vertical.

Colocar un grupo dentro de otro grupo es similar a crear una GUI usando contenedores anidados, como un objeto `JPanel` que contiene otros objetos `JPanel`, que a su vez contienen componentes de GUI.

Al crear un grupo, podemos especificar la **alineación** de sus elementos. La clase `GroupLayout` contiene cuatro constantes para este fin: `LEADING`, `TRAILING`, `CENTER` y `BASELINE`. La constante `BASELINE` se aplica sólo a orientaciones verticales. En la orientación horizontal, las constantes `LEADING`, `TRAILING` y `CENTER` representan la justificación a la izquierda, justificación a la derecha y centrado, respectivamente. En la orientación vertical, `LEADING`, `TRAILING` y `CENTER` alinean los componentes en su parte superior, inferior o centro vertical, respectivamente. Al alinear componentes con `BASELINE` estamos indicando que deben alinearse mediante el uso de la línea base de la fuente para el texto del componente. Para obtener más información acerca de las líneas base, vea la sección 12.4.

Espaciado

`GroupLayout` utiliza de manera predeterminada los lineamientos de diseño de GUIs de la plataforma subyacente para aplicar espacio entre un componente y otro. El método `addGap` de las clases de `GroupLayout` anidadas `GroupLayout.Group`, `GroupLayout.SequentialGroup` y `GroupLayout.ParallelGroup` nos permite controlar el espaciado entre componentes.

Ajustar el tamaño de los componentes

De manera predeterminada, `GroupLayout` utiliza los métodos `getMinimumSize`, `getMaximumSize` y `getPreferredSize` de cada componente para ayudar a determinar el tamaño del componente. Podemos redefinir la configuración predeterminada.

F.3 Creación de un objeto `SelectorColores`

Ahora vamos a presentar una aplicación llamada `SelectorColores` para demostrar el administrador de esquemas `GroupLayout`. Esta aplicación consiste en tres objetos `JSlider`, cada uno de los cuales representa los valores de 0 a 255 para especificar los valores rojo, verde y azul de un color. Los valores seleccionados para cada objeto `JSlider` se utilizarán para mostrar un rectángulo sólido del color especificado. Vamos a crear esta aplicación usando Netbeans 5.5. Para obtener una introducción más detallada acerca de cómo desarrollar aplicaciones de GUI en el IDE Netbeans, vea www.netbeans.org/kb/trails/matisse.html.

Cree un nuevo proyecto

Empiece por abrir un nuevo proyecto en Netbeans. Seleccione **File > New Project...** En el cuadro de diálogo **New Project**, seleccione **General** de la lista **Categories** y **Java Application** de la lista **Projects**; después haga clic en **Next >**. Especifique `SelectorColores` como el nombre del proyecto y desactive la casilla de verificación **Create Main Class**. También puede especificar la ubicación de su proyecto en el campo **Project Location**. Haga clic en **Finish** para crear el proyecto.

Agregue una nueva subclase de `JFrame` al proyecto

En la ficha **Projects** del IDE, justo debajo del menú **File** y la barra de herramientas (figura F.2), expanda el nodo **Source Packages**. Haga clic con el botón derecho del ratón en el nodo **<default package>** que aparece y seleccione **New > JFrame Form**. En el cuadro de diálogo **New JPanel Form**, especifique `SelectorColores` como el nombre de la clase y haga clic en **Finish**. Esta subclase de `JFrame` mostrará los componentes de la GUI de la aplicación. La ventana de Netbeans ahora deberá ser similar a la figura F.3, mostrando la clase `SelectorColores` en vista de diseño (**Design**). Los botones **Source** y **Design** en la parte superior de la ventana `SelectorColores.java` nos permiten alternar entre editar el código fuente y diseñar la GUI.

La vista **Design** sólo muestra el área cliente de **SelectorColores** (es decir, el área que aparecerá dentro de los bordes de la ventana). Para crear una GUI en forma visual, puede arrastrar componentes de GUI desde la ventana **Palette** hacia el área cliente. Para configurar las propiedades de cada componente, hay que seleccionarlo y después modificar los valores de las propiedades que aparecen en la ventana **Properties** (figura F.3). Al seleccionar un componente, la ventana **Properties** muestra tres botones: **Properties**, **Events** y **Code** (vea la figura F.4); éstos le permiten configurar varios aspectos del componente.



Figura F.2 | Agregue un nuevo formulario **JFrame** al proyecto **SelectorColores**.

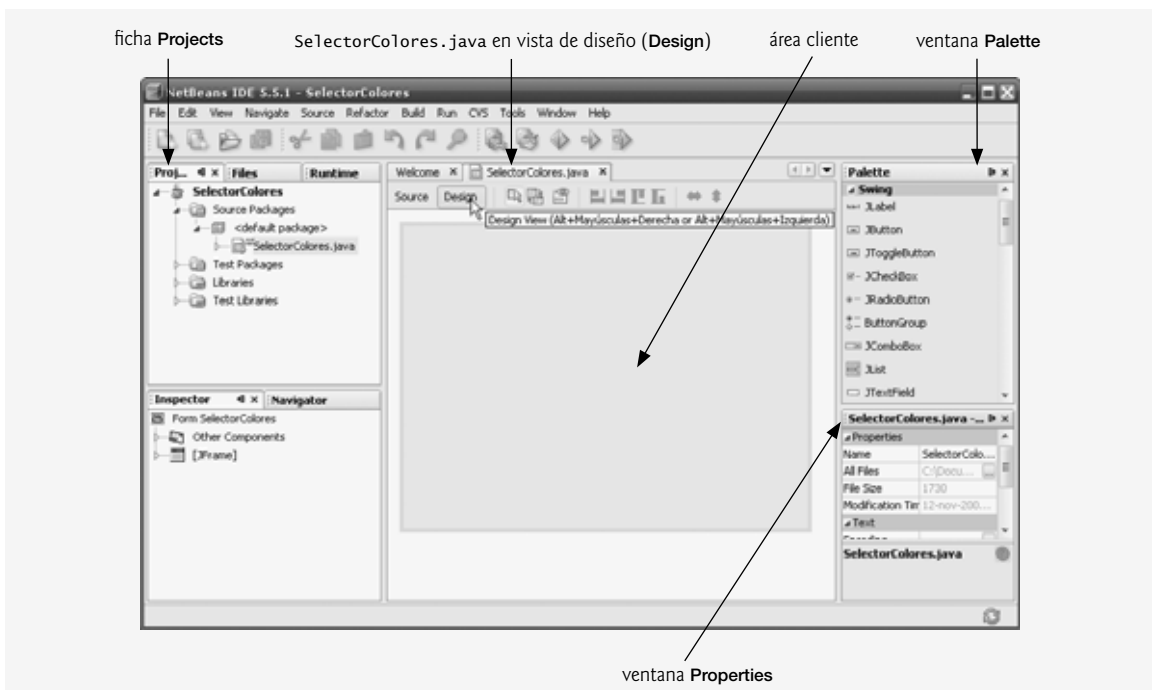


Figura F.3 | La clase **SelectorColores** se muestra en vista **Design** de Netbeans.

Cree la GUI

Arrastre tres componentes `JSlider` de la paleta (**Palette**) hacia el formulario `JFrame` (tal vez necesite desplazarse por la paleta). A medida que arrastramos componentes cerca de los bordes del área cliente, o cerca de otros componentes, Netbeans muestra **líneas guía** (figura F.4) que indican las distancias y alineaciones recomendadas entre el componente que estamos arrastrando, los bordes del área cliente y los demás componentes. A medida que siga los pasos para crear la GUI, use las líneas guía para ordenar los componentes en tres filas y tres columnas, como en la figura F.5. Use la ventana **Properties** para cambiar el nombre de los componentes `JSlider` a `rojoJSlider`, `verdeJSlider` y `azulJSlider`. Seleccione el primer componente `JSlider`, después haga clic en el botón **Code** de la ventana **Properties** y cambie la propiedad **Variable Name** a `rojoJSlider`. Repita este proceso para cambiar el nombre a los otros dos componentes `JSlider`. Después seleccione cada componente `JSlider` y cambie su propiedad **maximum** a 255, para que produzca valores en el rango de 0 a 255, y cambie su propiedad **value** a 0, de manera que el indicador del componente `JSlider` se encuentre inicialmente a la izquierda.

Arrastre tres componentes `JLabel` de la paleta al formulario `JFrame` para etiquetar cada componente `JSlider` con el color que representa. Use los nombres `rojoJLabel`, `verdeJLabel` y `azulJLabel` para los componentes `JLabel`, respectivamente. Cada componente `JLabel` debe colocarse a la izquierda del componente `JSlider` correspondiente (figura F.5). Cambie la propiedad **text** de cada componente `JLabel, ya sea haciendo doble clic en el componente JLabel y escribiendo el nuevo texto, o seleccionando el componente JLabel y cambiando la propiedad text en la ventana Properties.`

Agregue un componente `JTextField` a cada uno de los componentes `JSlider` para mostrar su valor. Use los nombres `rojoJTextField`, `verdeJTextField` y `azulJTextField` para estos componentes. Cambie la propiedad **text** de cada componente `JTextField` a 0, usando las mismas técnicas que para los componentes `JLabel`. Cambie la propiedad **columns** de cada componente `JTextField` a 4.



Figura F.4 | Posicione el primer componente `JTextField`.



Figura F.5 | Distribución de los componentes `JLabel`, `JSlider` y `JTextField`.

Haga doble clic en el borde del área cliente para que aparezca el cuadro de diálogo **Set Form Designer Size** y cambie el primer número (que representa la anchura) a 410; después haga clic en **OK**. Esto hace al área cliente lo suficientemente amplia como para poder alojar el componente `JPanel` que agregará a continuación. Por último, agregue un componente `JPanel` llamado `colorJPanel` a la derecha de este grupo de componentes. Use las líneas guía como se muestra en la figura F.6 para colocar el componente `JPanel`. Cambie el color de fondo de este componente para mostrar el color seleccionado. Por último, arrastre el borde inferior del área cliente hacia la parte superior del área **Design**, de manera que pueda ver la línea de ajuste que muestra la altura recomendada del área cliente (con base en sus componentes), como se muestra en la figura F.7.



Figura F.6 | Posicione el panel `JLabel`.

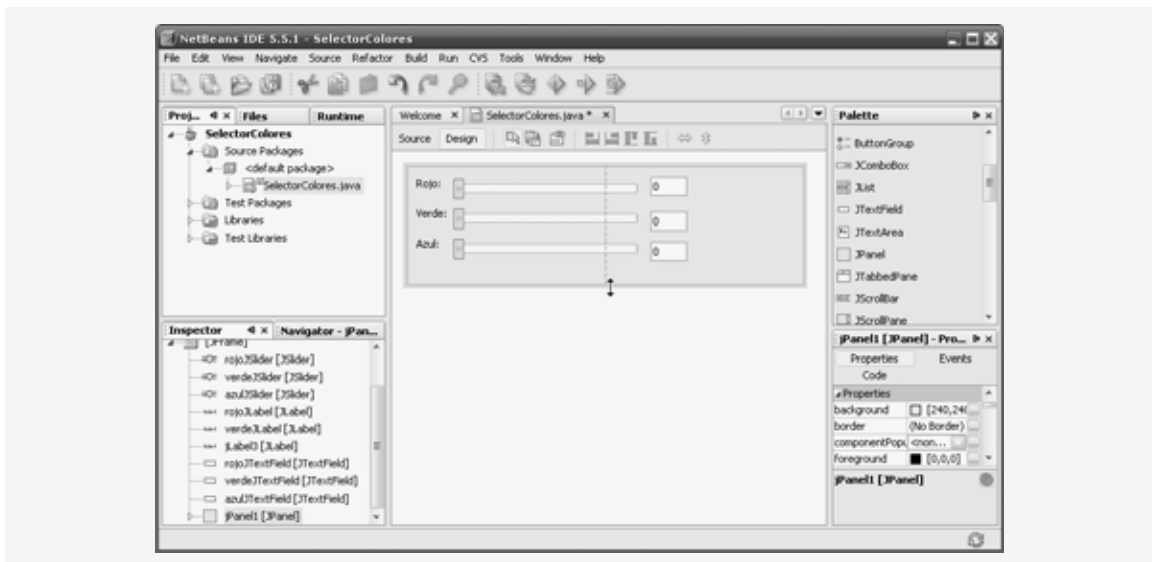


Figura F.7 | Ajuste de la altura del área cliente.

Edite el código fuente y agregue manejadores de eventos

El IDE generó de manera automática el código de la GUI, incluyendo métodos para inicializar componentes y alinearlos mediante el administrador de esquemas GroupLayout. Debemos agregar la funcionalidad deseada a los manejadores de eventos de los componentes. Para agregar un manejador de eventos para un componente, haga clic con el botón derecho sobre él y coloque el ratón sobre la opción **Events** en el menú contextual. A continuación, podrá seleccionar la categoría de evento que desee manejar, y el evento específico dentro de esa categoría. Por ejemplo, para agregar los manejadores de eventos para los componentes JSlider para este ejemplo, haga clic en cada componente JSlider y seleccione **Events > Change > stateChanged**. Al hacer esto, Netbeans agrega un objeto ChangeListener al componente JSlider y cambia de vista de diseño (**Design**) a vista de código fuente (**Source**), en donde podemos colocar código en el manejador de eventos. Use el botón **Design** para regresar a la vista de diseño y repita los pasos anteriores para agregar los manejadores de eventos para los otros dos componentes JSlider. Para completar los manejadores de eventos, agregue primero el método de la figura F.8. En cada manejador de eventos de JSlider, establezca el componente JTextField correspondiente con el nuevo valor del componente JSlider, y después llame al método cambiarColor. Por último, en el constructor después de la llamada a initComponents, agregue la línea:

```
colorJPanel.setBackground( java.awt.Color.BLACK );
```

La figura F.9 muestra la clase SelectorColores completa, exactamente como la genera Netbeans. Cada vez una mayor parte del desarrollo de software se lleva a cabo con herramientas que generan código complicado como éste, lo cual ahorra al lector el tiempo y esfuerzo de hacerlo por sí mismo.

```
1 // cambia el color de fondo del componente colorJPanel, con base en los valores
2 // actuales de los componentes JSlider
3 public void cambiarColor()
4 {
5     colorJPanel.setBackground( new java.awt.Color(
6         rojoJSlider.getValue(), verdeJSlider.getValue(),
7         azulJSlider.getValue() ) );
8 } // fin del método cambiarColor
```

Figura F.8 | Método que cambia el color de fondo de colorJPanel, con base en los valores de los tres componentes JSlider.

```
1 /*
2  * SelectorColores.java
3  *
4  * Created on 12 de noviembre de 2007, 3:51
5  */
6
7 /**
8  *
9  * @author Administrador
10 */
11 public class SelectorColores extends javax.swing.JFrame
12 {
13
14     /** Creates new form SelectorColores */
15     public SelectorColores()
16     {
17         initComponents();
18         colorJPanel.setBackground( java.awt.Color.BLACK );
19     }
20 }
```

Figura F.9 | Clase SelectorColores que utiliza a GroupLayout para su esquema de GUI. (Parte I de 5).

```

20
21 // cambia el color de fondo del componente colorJPanel, con base en los valores
22 // actuales de los componentes JSlider
23 public void cambiarColor()
24 {
25     colorJPanel.setBackground( new java.awt.Color(
26         rojoJSlider.getValue(), verdeJSlider.getValue(),
27         azulJSlider.getValue() ) );
28 } // fin del método cambiarColor
29
30 /** This method is called from within the constructor to
31  * initialize the form.
32  * WARNING: Do NOT modify this code. The content of this method is
33  * always regenerated by the Form Editor.
34  */
35 // <editor-fold defaultstate="collapsed" desc=" Generated Code ">
36 private void initComponents() {
37     rojoJSlider = new javax.swing.JSlider();
38     verdeJSlider = new javax.swing.JSlider();
39     azulJSlider = new javax.swing.JSlider();
40     rojoJLabel = new javax.swing.JLabel();
41     verdeJLabel = new javax.swing.JLabel();
42     jLabel3 = new javax.swing.JLabel();
43     rojoJTextField = new javax.swing.JTextField();
44     verdeJTextField = new javax.swing.JTextField();
45     azulJTextField = new javax.swing.JTextField();
46     colorJPanel = new javax.swing.JPanel();
47
48     setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
49     rojoJSlider.setMaximum(255);
50     rojoJSlider.setValue(0);
51     rojoJSlider.setName("null");
52     rojoJSlider.addChangeListener(new javax.swing.event.ChangeListener() {
53         public void stateChanged(javax.swing.event.ChangeEvent evt) {
54             rojoJSliderStateChanged(evt);
55         }
56     });
57
58     verdeJSlider.setMaximum(255);
59     verdeJSlider.setValue(0);
60     verdeJSlider.setName("null");
61     verdeJSlider.addChangeListener(new javax.swing.event.ChangeListener() {
62         public void stateChanged(javax.swing.event.ChangeEvent evt) {
63             verdeJSliderStateChanged(evt);
64         }
65     });
66
67     azulJSlider.setMaximum(255);
68     azulJSlider.setValue(0);
69     azulJSlider.addChangeListener(new javax.swing.event.ChangeListener()
70     {
71         public void stateChanged(javax.swing.event.ChangeEvent evt) {
72             azulJSliderStateChanged(evt);
73         }
74     });
75
76     rojoJLabel.setText("Rojo:");
77
78     verdeJLabel.setText("Verde:");

```

Figura F.9 | Clase SelectorColores que utiliza a GroupLayout para su esquema de GUI. (Parte 2 de 5).

```

79
80     jLabel3.setText("Azul:");
81
82     rojoJTextField.setColumns(4);
83     rojoJTextField.setText("0");
84
85     verdeJTextField.setColumns(4);
86     verdeJTextField.setText("0");
87
88     azulJTextField.setColumns(4);
89     azulJTextField.setText("0");
90
91     javax.swing.GroupLayout colorJPanelLayout = new javax.swing.GroupLayout(colorJPanel);
92     colorJPanel.setLayout(colorJPanelLayout);
93     colorJPanelLayout.setHorizontalGroup(
94         colorJPanelLayout.createParallelGroup(
95             javax.swing.GroupLayout.Alignment.LEADING)
96             .addGroup(
97                 colorJPanelLayout.createParallelGroup(
98                     javax.swing.GroupLayout.Alignment.LEADING)
99                     .addGap(0, 100, Short.MAX_VALUE)
100                );
101
102     javax.swing.GroupLayout layout = new
103     javax.swing.GroupLayout(getContentPane());
104     getContentPane().setLayout(layout);
105     layout.setHorizontalGroup(
106         layout.createParallelGroup(
107             javax.swing.GroupLayout.Alignment.LEADING)
108             .addGroup(layout.createSequentialGroup()
109                 .addGroup(layout.createParallelGroup(
110                     javax.swing.GroupLayout.Alignment.TRAILING, false)
111                     .addComponent(rojoJLabel, javax.swing.GroupLayout.DEFAULT_SIZE, javax.
112                         swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
113                     .addComponent(verdeJLabel, javax.swing.GroupLayout.DEFAULT_SIZE, javax.
114                         swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
115                     .addComponent(jLabel3, javax.swing.GroupLayout.DEFAULT_SIZE, javax.
116                         swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
117                 .addPreferredGap(
118                     javax.swing.LayoutStyle.ComponentPlacement.RELATED)
119                 .addGroup(layout.createParallelGroup(
120                     javax.swing.GroupLayout.Alignment.LEADING)
121                     .addGroup(layout.createSequentialGroup()
122                         .addGroup(layout.createParallelGroup(
123                             javax.swing.GroupLayout.Alignment.LEADING)
124                             .addComponent(azulJSlider, javax.swing.GroupLayout.PREFERRED_SIZE,
125                                 javax.swing.GroupLayout.PREFERRED_SIZE)
126                             .addPreferredGap(
127                                 javax.swing.LayoutStyle.ComponentPlacement.RELATED)
128                             .addComponent(verdeJTextField, javax.swing.GroupLayout.PREFERRED_

```

Figura F.9 | Clase SelectorColores que utiliza a GroupLayout para su esquema de GUI. (Parte 3 de 5).


```

SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
122         .addGroup(layout.createSequentialGroup()
123             .addComponent(rojoJSlider, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
124             .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
125             .addComponent(rojoJTextField, javax.swing.GroupLayout.PREFERRED_
SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)))
126         .addPreferredGap(
javax.swing.LayoutStyle.ComponentPlacement.RELATED, 10, Short.MAX_VALUE)
127         .addComponent(colorJPanel, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
128         .addContainerGap())
129     );
130     layout.setVerticalGroup(
131         layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
132         .addGroup(layout.createSequentialGroup()
133             .addContainerGap()
134             .addGroup(layout.createParallelGroup(
javax.swing.GroupLayout.Alignment.LEADING)
135                 .addComponent(colorJPanel, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
136                 .addGroup(layout.createSequentialGroup()
137                     .addGroup(layout.createParallelGroup(
javax.swing.GroupLayout.Alignment.LEADING, false)
138                         .addGroup(layout.createSequentialGroup()
139                             .addComponent(rojoJTextField, javax.swing.GroupLayout.
PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
140                             .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.
RELATED, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
141                             .addComponent(verdeJTextField, javax.swing.GroupLayout.
PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_
SIZE)))
142                             .addGroup(layout.createSequentialGroup()
143                                 .addGroup(layout.createParallelGroup(
javax.swing.GroupLayout.Alignment.LEADING)
144                                     .addComponent(rojoJSlider, javax.swing.GroupLayout.PREFERRED_
SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
145                                     .addComponent(rojoJLabel))
146                                     .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.
RELATED)
147                                     .addGroup(layout.createParallelGroup(
javax.swing.GroupLayout.Alignment.LEADING)
148                                         .addComponent(verdeJSlider, javax.swing.GroupLayout.PREFERRED_
SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
149                                         .addComponent(verdeJLabel))))))
150                             .addPreferredGap(
javax.swing.LayoutStyle.ComponentPlacement.RELATED)
151                             .addGroup(layout.createParallelGroup(
javax.swing.GroupLayout.Alignment.LEADING)
152                                 .addComponent(azulJLabel)
153                                 .addGroup(layout.createParallelGroup(
javax.swing.GroupLayout.Alignment.TRAILING)
154                                     .addComponent(azulJTextField, javax.swing.GroupLayout.
PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
155                                     .addComponent(azulJSlider, javax.swing.GroupLayout.PREFERRED_
SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))))))
156                             .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.
MAX_VALUE))
157     );

```

Figura F.9 | Clase SelectorColores que utiliza a GroupLayout para su esquema de GUI. (Parte 4 de 5).

```

158     pack();
159 }// </editor-fold>
160
161 private void azulJSliderStateChanged(javax.swing.event.ChangeEvent evt) {
162     azulJTextField.setText( "" + azulJSlider.getValue() );
163     cambiarColor();
164 }
165
166 private void verdeJSliderStateChanged(javax.swing.event.ChangeEvent evt) {
167     verdeJTextField.setText( "" + verdeJSlider.getValue() );
168     cambiarColor();
169 }
170
171 private void rojoJSliderStateChanged(javax.swing.event.ChangeEvent evt) {
172     rojoJTextField.setText( "" + rojoJSlider.getValue() );
173     cambiarColor();
174 }
175
176 /**
177  * @param args the command line arguments
178  */
179 public static void main(String args[]) {
180     java.awt.EventQueue.invokeLater(new Runnable() {
181         public void run() {
182             new SelectorColores().setVisible(true);
183         }
184     });
185 }
186
187 // Variables declaration - do not modify
188 private javax.swing.JSlider azulJSlider;
189 private javax.swing.JTextField azulJTextField;
190 private javax.swing.JPanel colorJPanel;
191 private javax.swing.JLabel jLabel3;
192 private javax.swing.JLabel rojoJLabel;
193 private javax.swing.JSlider rojoJSlider;
194 private javax.swing.JTextField rojoJTextField;
195 private javax.swing.JLabel verdeJLabel;
196 private javax.swing.JSlider verdeJSlider;
197 private javax.swing.JTextField verdeJTextField;
198 // End of variables declaration
199
200 }

```

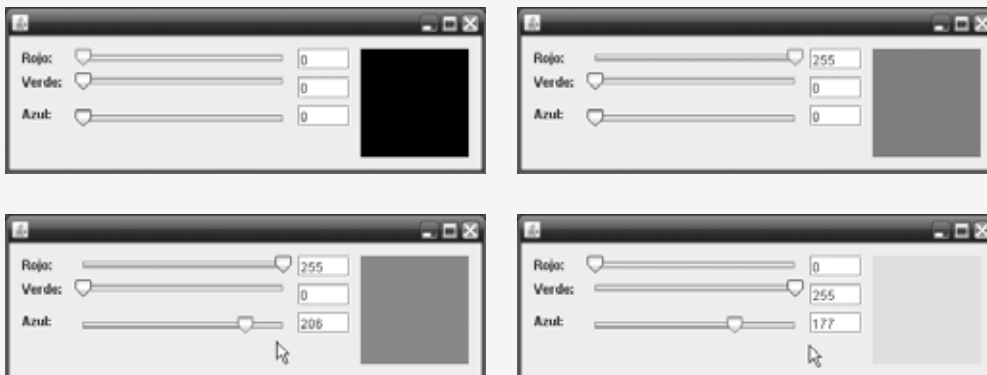


Figura F.9 | Clase `SelectorColores` que utiliza a `GroupLayout` para su esquema de GUI. (Parte 5 de 5).

El método `initComponents` (líneas 36 a 159) fue generado completamente por Netbeans, con base en las interacciones del lector con el diseñador de GUIs. Este método contiene el código que crea y da formato a la GUI. En las líneas 38 a 93 se construyen e inicializan los componentes de la GUI. En las líneas 91 a 159 se especifica la distribución de esos componentes mediante el uso de `GroupLayout` . En las líneas 104 a 129 se especifica el grupo horizontal y en las líneas 130 a 157 se especifica el grupo vertical.

Agregamos en forma manual la instrucción que modifica el color de fondo del componente `colorJPanel` en la línea 18, y el método `cambiarColor` en las líneas 23 a 28. Cuando el usuario desplaza el indicador en uno de los componentes `JSlider` , el manejador de eventos de ese componente establece el texto en su correspondiente componente `JTextField` con el nuevo valor del componente `JSlider` (líneas 162, 167 y 172), después llama el método `cambiarColor` (líneas 163, 168 y 173) para actualizar el color de fondo del componente `colorJPanel` . El método `cambiarColor` obtiene el valor actual de cada componente `JSlider` (líneas 26 y 27), y utiliza estos valores como argumentos para el constructor de `Color` y crear un nuevo objeto `Color` .

F.4 Recursos Web sobre GroupLayout

weblogs.java.net/blog/tpavek/archive/2006/02/getting_to_know_1.html

Parte 1 del mensaje publicado en el blog sobre `GroupLayout` de Tomas Pavel; presenta las generalidades detrás de la teoría de `GroupLayout` .

weblogs.java.net/blog/tpavek/archive/2006/03/getting_to_know.html

Parte 2 del mensaje publicado en el blog sobre `GroupLayout` de Tomas Pavel; presenta una GUI completa, implementada con `GroupLayout` .

wiki.java.net/bin/view/Javadesktop/GroupLayoutExample

Proporciona una demostración de una Libreta de direcciones, de una GUI creada en forma manual con `GroupLayout` , con código fuente.

java.sun.com/developer/technicalArticles/Interviews/violet_pavek_qa.html

Artículo: “La siguiente ola de GUIs: el proyecto Matisse y el IDE Netbeans 5.0”, por Roman Strobl.

www.netbeans.org/kb/50/quickstart-gui.html

Tutorial: “Creación de GUIs en Netbeans 5.0”, por Talley Mulligan. Un recorrido a través de la creación de GUIs en Netbeans.

testwww.netbeans.org/kb/41/flash-matisse.html

Demostración en Flash del diseñador de la GUI Matisse de Netbeans, la cual utiliza a `GroupLayout` para ordenar componentes.

www.developer.com/java/ent/article.php/3589961

Tutorial sobre `GroupLayout` basado en Flash.

weblogs.java.net/blog/claudio/archive/nb-layouts.html

Tutorial: “Building Java GUIs with Matisse: A Gentle Introduction”, por Dick Wall.



Componentes de integración Java Desktop (JDIC)

G.1 Introducción

Los **Componentes de integración Java Desktop (JDIC)** son parte de un proyecto de código fuente abierto, orientado a permitir una mejor integración entre las aplicaciones de Java y las plataformas en las que se ejecutan. Algunas características de JDIC son:

- Interacción con la plataforma subyacente para iniciar aplicaciones nativas (como navegadores Web y clientes de correo electrónico).
- Mostrar una pantalla de inicio cuando una aplicación empieza a ejecutarse para indicar al usuario que se está cargando.
- Creación de iconos en la bandeja del sistema (también llamada área de estado de la barra de tareas, o área de notificación) para proporcionar acceso a las aplicaciones Java que se ejecutan en segundo plano.
- Registro de asociaciones de tipos de archivos, para que los archivos de tipos especificados se abran automáticamente en las correspondientes aplicaciones de Java.
- Creación de paquetes instaladores, y otras cosas más.

La página inicial de JDIC (jdk.dev.java.net/) incluye una introducción a JDIC, descargas, documentación, FAQs, demos, artículos, blogs, anuncios, proyectos Incubator, una página para el desarrollador, foros, listas de correo y mucho más. Java SE 6 ahora incluye algunas de las características antes mencionadas. Aquí hablaremos sobre varias de estas características.

G.2 Pantallas de inicio

Los usuarios de aplicaciones de Java perciben con frecuencia un problema en el rendimiento, ya que no aparece nada en la pantalla cuando se inicia una aplicación por primera vez. Una manera de mostrar a un usuario que su programa se está cargando es mediante una **pantalla de inicio**: una ventana sin bordes que aparece temporalmente mientras se inicia una aplicación. Java SE 6 proporciona la nueva opción de línea de comandos **-splash** para que el comando java pueda llevar a cabo esta tarea. Esta opción permite al programador especificar una imagen PNG, GIF o JPG que debe aparecer al momento en que una aplicación empieza a cargarse. Para demostrar esta nueva opción, creamos un programa (figura G.1) que permanece inactivo durante 5 segundos (para que el usuario pueda ver la pantalla de inicio) y después muestra un mensaje en la línea de comandos. El directorio para este ejemplo incluye una imagen en formato PNG para utilizarla como pantalla de inicio. Para mostrar la pantalla de inicio a la hora de cargar esta aplicación, use el siguiente comando:

```
java -splash:DeitelBug.png DemoSplash
```

```

1 // Fig. G.1: DemoInicio.java
2 // Demostración de la pantalla de inicio.
3 public class DemoInicio
4 {
5     public static void main( String[] args )
6     {
7         try
8         {
9             Thread.sleep( 5000 );
10        } // fin de try
11        catch ( InterruptedException e )
12        {
13            e.printStackTrace();
14        } // fin de catch
15
16        System.out.println(
17            "Esta fue la demostracion de la pantalla de inicio." );
18    } // fin del método main
19 } // fin de la clase DemoInicio

```

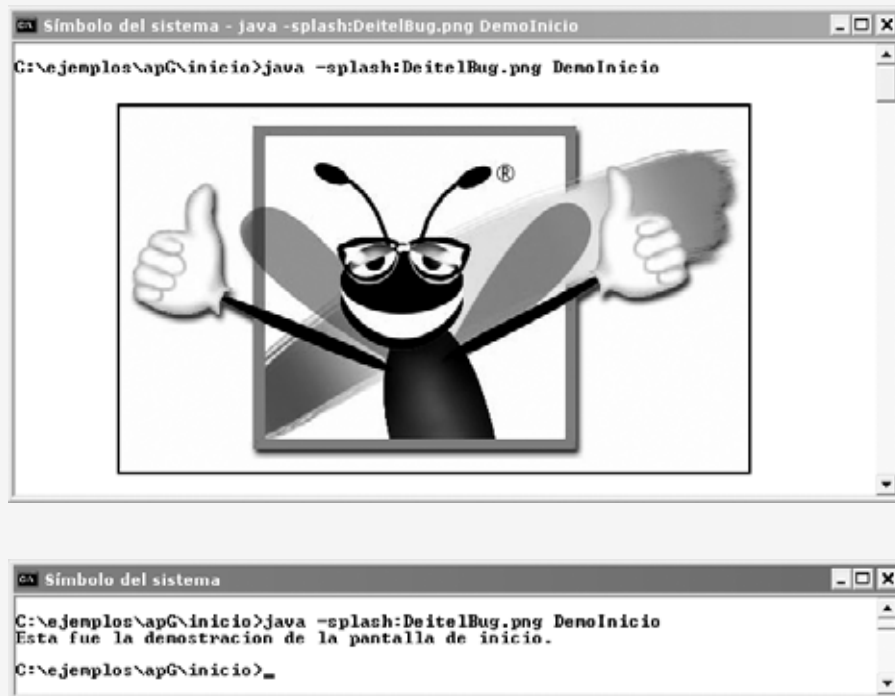


Figura G.1 | Pantalla de inicio que se muestra mediante la opción `-splash` del comando `java`.

Una vez que haya iniciado la visualización de la pantalla de inicio, podrá interactuar con ésta por medio de programación, mediante la clase `SplashScreen` del paquete `java.awt`. Para ello, puede agregar contenido dinámico a la pantalla de inicio. Para obtener más información acerca de cómo trabajar con las pantallas de inicio, vea los siguientes sitios:

java.sun.com/developer/technicalArticles/J2SE/Desktop/javase6/splashscreen/
java.sun.com/javase/6/docs/api/java/awt/SplashScreen.html

G.3 La clase Desktop

La nueva clase **Desktop** de Java SE 6 nos permite especificar un archivo o URI que deseemos abrir, mediante el uso de la aplicación apropiada de la plataforma subyacente. Por ejemplo, si el navegador Web predeterminado de su computadora es Firefox, puede usar el método `browse` de la clase `Desktop` para abrir un sitio Web en Firefox. Además, puede abrir una ventana de composición de correo electrónico en el cliente de correo electrónico predeterminado de su sistema, abrir un archivo en su aplicación asociada e imprimir un archivo mediante el uso del comando `imprimir` de la aplicación asociada. En la figura G.2 se demuestran las primeras tres de estas capacidades.

El manejador de eventos en las líneas 22 a 52 obtiene el número de índice de la tarea que el usuario selecciona en el componente `tareasJComboBox` (línea 25), y el objeto `String` que representa el archivo o URI a procesar (línea 26). En la línea 28 se utiliza el método `static isDesktopSupported` de `Desktop` para determinar si se soportan las características de la clase `Desktop` en la plataforma en la que se ejecute la aplicación. De ser así, en la línea 32 se utiliza el método `static getDesktop` de `Desktop` para obtener un objeto `Desktop`. Si el usuario seleccionó la opción para abrir el navegador Web predeterminado, en la línea 37 se crea un nuevo objeto URI mediante el uso del objeto `String` llamado `entrada` como el sitio a mostrar en el navegador, y después se pasa el objeto URI al método `browse` de `Desktop`, el cual invoca al navegador Web predeterminado del sistema y le pasa el URI para que lo muestre. Si el usuario selecciona la opción para abrir un archivo en su programa asociado, en la línea 40 se crea un nuevo objeto `File` usando el objeto `String` llamado `entrada` como el archivo a abrir, y después se pasa este objeto `File` al método `open` de `Desktop`, el cual pasa el archivo a la aplicación apropiada para que lo abra. Por último, si el usuario selecciona la opción para componer un correo electrónico, en la línea 43 se crea un nuevo objeto URI usando el objeto `String` llamado `entrada` como la dirección de correo a la cuál se enviará el correo electrónico, y después se pasa el objeto URI al método `mail` de `Desktop`, el cual invoca al cliente de correo electrónico predeterminado del sistema y pasa el URI a ese cliente de correo electrónico como el recipiente del mensaje. Para aprender más acerca de la clase `Desktop`, visite el sitio:

java.sun.com/javase/6/docs/api/java/awt/Desktop.html

```

1 // Fig. G.2: DemoDesktop.java
2 // Usa a Desktop para iniciar el navegador predeterminado, abrir un archivo en su
3 // aplicación asociada y componer un email en el cliente de email predeterminado.
4 import java.awt.Desktop;
5 import java.io.File;
6 import java.io.IOException;
7 import java.net.URI;
8
9 public class DemoDesktop extends javax.swing.JFrame
10 {
11     // constructor
12     public DemoDesktop()
13     {
14         initComponents();
15     } // fin del constructor de DemoDesktop
16
17     // Para ahorrar espacio, no mostramos aquí las líneas 20 a 84 del código de GUI
18     // generado por Netbeans de manera automática. El código completo para este ejemplo se
19     // encuentra en el archivo DemoDesktop.java en el directorio de este ejemplo.
20
21     // determina la tarea seleccionada y la lleva a cabo
22     private void hacerTareaJButtonActionPerformed(
23         java.awt.event.ActionEvent evt)
24     {
25         int indice = tareasJComboBox.getSelectedIndex();
26         String entrada = entradaJTextField.getText();

```

Figura G.2 | Use a `Desktop` para iniciar el navegador Web predeterminado, abrir un archivo en su aplicación asociada y componer un correo electrónico en el cliente de correo predeterminado. (Parte I de 3).

```

27
28     if ( Desktop.isDesktopSupported() )
29     {
30         try
31         {
32             Desktop escritorio = Desktop.getDesktop();
33
34             switch ( indice )
35             {
36                 case 0: // abre el navegador
37                     escritorio.browse( new URI( entrada ) );
38                     break;
39                 case 1: // abre el archivo
40                     escritorio.open( new File( entrada ) );
41                     break;
42                 case 2: // abre la ventana de composición de email
43                     escritorio.mail( new URI( entrada ) );
44                     break;
45             } // fin de switch
46         } // fin de try
47         catch ( Exception e )
48         {
49             e.printStackTrace();
50         } // fin de catch
51     } // end if
52 } // fin del método hacerTareaJButtonActionPerformed
53
54 public static void main(String args[])
55 {
56     java.awt.EventQueue.invokeLater(
57         new Runnable()
58         {
59             public void run()
60             {
61                 new DemoDesktop().setVisible(true);
62             }
63         }
64     );
65 } // fin del método main
66
67 // Variables declaration - do not modify//GEN-BEGIN:variables
68 private javax.swing.JLabel entradaJLabel;
69 private javax.swing.JTextField entradaJTextField;
70 private javax.swing.JButton hacerTareaJButton;
71 private javax.swing.JLabel instruccionesJLabel;
72 private javax.swing.JComboBox tareasJComboBox;
73 // End of variables declaration//GEN-END:variables
74 }

```

Figura G.2 | Use a Desktop para iniciar el navegador Web predeterminado, abrir un archivo en su aplicación asociada y componer un correo electrónico en el cliente de correo predeterminado. (Parte 2 de 3).

G.4 Iconos de la bandeja

Los **iconos de la bandeja** comúnmente aparecen en la bandeja de sistema de nuestro sistema, en el área de estado de la barra de tareas o en el área de notificación. Por lo general, proporcionan un acceso rápido a las aplicaciones que se ejecutan en segundo plano en el sistema del programador. Al posicionar el ratón sobre uno de estos iconos, aparece una barra de herramientas indicando qué aplicación representa el icono. Si hace clic en el icono, aparecerá un menú contextual con opciones para esa aplicación.

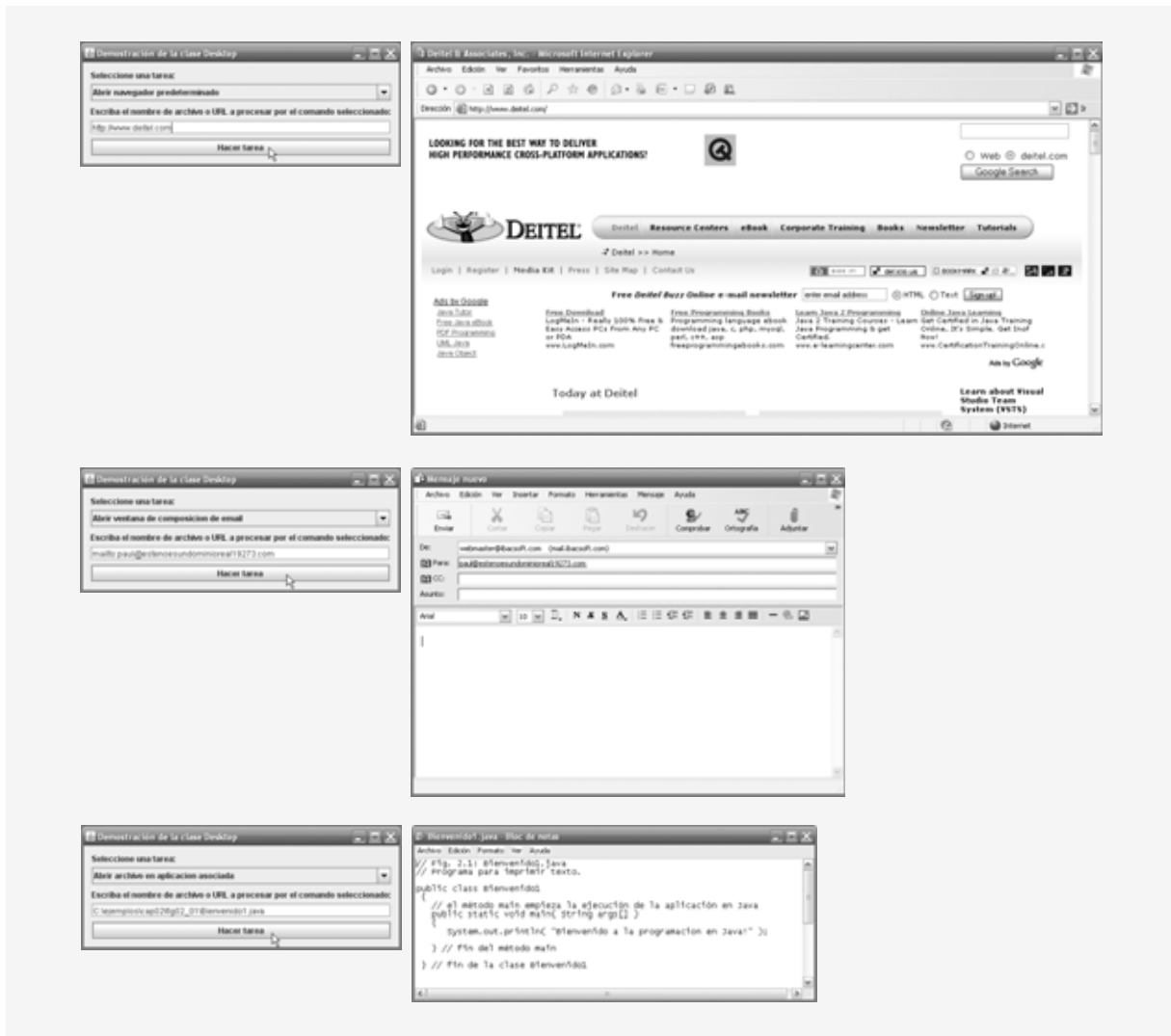


Figura G.2 | Use a Desktop para iniciar el navegador Web predeterminado, abrir un archivo en su aplicación asociada y componer un correo electrónico en el cliente de correo predeterminado. (Parte 3 de 3).

Las clases `SystemTray` y `TrayIcon` (ambas del paquete `java.awt`) nos permiten crear y administrar nuestros propios iconos de la bandeja, de una manera independiente a la plataforma. La clase `SystemTray` proporciona acceso a la bandeja del sistema de la plataforma subyacente; la clase consiste en tres métodos:

- El método `static getDefaultSystemTray` devuelve la bandeja del sistema.
- El método `addTrayIcon` agrega un nuevo objeto `TrayIcon` a la bandeja del sistema.
- El método `removeTrayIcon` elimina un icono de la bandeja del sistema.

La clase `TrayIcon` consiste en varios métodos que permiten a los usuarios especificar un icono, un cuadro de información sobre las herramientas y un menú contextual para el icono. Además, los iconos de la bandeja soportan objetos `ActionListener`, `MouseListener` y `MouseMotionListener`. Para aprender más acerca de las clases `SystemTray` y `TrayIcon`, visite:

java.sun.com/javase/6/docs/api/java/awt/SystemTray.html
java.sun.com/javase/6/docs/api/java/awt/TrayIcon.html

G.5 Proyectos JDIC Incubator

Los Proyectos JDIC Incubator son desarrollados, administrados y pertenecen a los miembros de la comunidad de Java. Estos proyectos se asocian (pero no se distribuyen con) JDIC. Los Proyectos Incubator pueden en algún momento dado formar parte del proyecto JDIC, una vez que se hayan desarrollado por completo y cumplan con ciertos criterios. Para obtener más información acerca de los Proyectos Incubator, y para aprender cómo puede establecer un Proyecto Incubator, visite el sitio:

`jdic.dev.java.net/#incubator`

Los Proyectos Incubator actuales son:

- FileUtil: la API de una herramienta de archivos, que extiende a la clase `java.io.File`.
- Floating Dock Top-level Window: una API de Java para desarrollar una ventana flotante acoplable de nivel superior.
- Icon Service: devuelve un objeto icono de Java, de una especificación de icono nativa.
- Misc API: Contiene APIs simples (un método, un tipo de clase).
- Music Player Control API: API de Java que controla los reproductores de música nativos.
- SaverBeans Screensaver SDK: kit de desarrollo de protectores de pantalla en Java.
- SystemInfo: comprueba la información del sistema.

G.6 Demos de JDIC

El sitio de JDIC incluye aplicaciones de demostración para FileExplorer, el paquete de navegador, el paquete TrayIcon, la clase Floating Dock y la API Wallpaper (`jdic.dev.java.net/#demos`). El código fuente para estos demos se incluye en la descarga de JDIC (`jdic.dev.java.net/servlets/ProjectDocumentList`). Para obtener más demos, dé un vistazo a algunos de los proyectos Incubator.



Mashups

Introducción

La creación de mashups de aplicaciones Web es uno de los temas insignia de Web 2.0. El término mashup se originó en el mundo de la música; un mashup es un remix de dos o más canciones para crear una nueva canción. Puede escuchar algunos mashups de música en www.ccmixter.org/. Un mashup de aplicación Web combina la funcionalidad complementaria que, por lo general, se utiliza a través de servicios Web (capítulo 28) y transmisiones RSS (www.deitel.com/rss y www.rssbus.com) de varios sitios Web. Podemos crear poderosas e innovadoras aplicaciones mashup Web 2.0 con mucha más rapidez que si tuviéramos que escribir las aplicaciones desde cero. Por ejemplo, www.housingmaps.com combina los listados de apartamentos Craigslist con Google Maps para mostrar en un mapa todos los apartamentos en renta en un vecindario.

Mashups populares

En la figura H.1 se muestran algunos mashups populares.

URL	APIs	Descripción
<i>Mashups populares de Google Maps:</i>		
www.mappr.com/	Google Maps, Flickr	Búsqueda de fotografías de sitios en EE.UU.
www.housingmaps.com/	Google Maps, Craigslist	Búsqueda de apartamentos y casas disponibles por vecindario. Incluye precios, fotografías, la dirección y la información de contacto del agente de bienes raíces.
www.broadwayzone.com/	Google Maps	Búsqueda de ubicaciones de teatros en la ciudad de Nueva York y los espectáculos de cada teatro. Vínculos a los detalles acerca del espectáculo, información sobre boletos e indicaciones para el metro.
www.cribseek.com	Google Maps	Mapas con propiedades en venta.
www.shackprices.com/	Google Maps	Búsqueda del valor aproximado de una casa, con base en las ventas recientes de casas en el área.
www.mashmap.com/	Google Maps	Haga clic en un cine en el mapa para buscar películas y mostrar horarios.

Figura H.1 | Mashups populares. (Parte de 1 de 2).

URL	APIs	Descripción
paul.kedrosky.com/publicloos/	Google Maps	Búsqueda de sanitarios públicos en San Francisco. Incluye la dirección, una clasificación y comentarios sobre cada sanitario.
<i>Otros mashups populares:</i>		
www.doubletrust.net	Yahoo! Search, Google Search	Combina los resultados de búsquedas de Yahoo! y Google en una página.
api.local.yahoo.com/eb/	Yahoo! Maps	Búsqueda de la ubicación de ciertos eventos (por fecha) en un área geográfica.
www.csthota.com/geotagr	Microsoft Virtual Earth	Almacene y explore fotografías por ubicación geográfica.
www.kokogiak.com/amazon4/default.asp	Amazon Web Services	Agregue artículos de Amazon a su lista de regalos, coloque el vínculo a un libro en su blog en Blogger, agregue un vínculo a sus sitios favoritos de del.icio.us o busque el libro en su biblioteca local.

Figura H.1 | Mashups populares. (Parte de 2 de 2).

Ahora que ha leído la mayor parte de este libro, tal vez esté familiarizado con las categorías de APIs, incluyendo gráficos, GUI, colecciones, multimedia, bases de datos y muchas más. Casi todas ellas proporcionan una *funcionalidad de cómputo* mejorada. Muchas APIs de servicios Web proporcionan *funcionalidad comercial*: eBay proporciona herramientas para subastas, Amazon proporciona ventas de libros (y ventas de otros tipos de productos, como CDs, DVDs, dispositivos electrónicos, y otros más), Google proporciona herramientas de búsqueda, PayPal proporciona servicios de pago, etcétera. Por lo general, estos servicios Web son gratuitos para su uso no comercial; algunos establecen cuotas (por lo general, razonables) para su uso comercial. Esto crea enormes posibilidades para las personas que crean aplicaciones y comercios basados en Internet.

APIs de uso común en mashups

Hemos enfatizado la importancia de la reutilización de software. Los mashups son otra forma más de reutilización de software que nos ahorra tiempo, dinero y esfuerzo; podemos crear rápidamente versiones prototipo de nuestras aplicaciones, integrar funcionalidad de negocios, de búsqueda y mucho más. En la figura H.2 se muestran algunas APIs de uso común en mashups.

Origen de API	URL	Funcionalidad
Google Maps	www.google.com/apis/maps	Mapas.
Yahoo! Maps	developer.yahoo.net/maps/	Mapas.
Microsoft Virtual Earth	virtualearth.msn.com/	Búsqueda local, mapas.
Amazon	aws.amazon.com/	Comercio electrónico.
TypePad ATOM	www.sixapart.com/pronet/docs/typepad_atom_api	Blogging.
Blogger ATOM feed	code.blogspot.com	Blogging.
Flickr	developer.yahoo.net/flickr/index.html	Compartir fotografías.
YouTube	www.youtube.com/dev	Compartir videos.
PayPal	developer.paypal.com	Pagos.

Figura H.2 | APIs de uso común para crear mashups. (Parte de 1 de 2).

Origen de API	URL	Funcionalidad
del.icio.us	del.icio.us/help/api/	Sitios favoritos de interés social.
Backpack	backpackit.com/	Programación de eventos.
Dropcash	www.dropcash.com/	Organizador de recaudación de fondos.
Upcoming.org	upcoming.org/services/api/	Listados de eventos de sindicatos.
Google AdWords	www.google.com/apis/adwords/	Administrar programas de publicidad de Google AdWords.
eBay	developer.ebay.com/common/api	Subastas.
SalesForce	www.salesforce.developer/	Administración de relaciones con los clientes (CRM).
Technorati	developers.technorati.com/ wiki/TechoratiApi	Búsqueda en Blogs.

Figura H.2 | APIs de uso común para crear mashups. (Parte de 2 de 2).

Centro de recursos Deitel sobre mashups

Nuestro Centro de recursos sobre mashups, que se encuentra en

www.deitel.com/mashups/MashUpsResourceCenter.html

se enfoca en la enorme cantidad de contenido de mashups gratuito, disponible en línea. Encontrará tutoriales, artículos, documentación, los libros más recientes, blogs, directorios, herramientas, foros, etcétera, que le ayudarán a desarrollar rápidamente aplicaciones de mashups.

- Dé un vistazo a los mashups más recientes y populares, incluyendo decenas de mashups basados en Google Maps, mostrando la ubicación de cines, bienes raíces para venta o renta, propiedades que se han vendido en su área, ¡e incluso las ubicaciones de los sanitarios públicos en San Francisco!
- Busque mashups en ProgrammableWeb por categoría.
- Dé un vistazo a las APIs de Flickr para agregar fotografías a sus aplicaciones, actualizar fotografías, reemplazarlas, ver peticiones de ejemplos y enviar en forma asíncrona.
- Lea el artículo “Building Mashups for Non-Programmers”.
- Dé un vistazo a la herramienta Smashforce que permite a los usuarios de Salesforce.com usar aplicaciones como Google Maps con sus aplicaciones Multiforce y Sforce empresariales.
- Busque sitios sobre mashups tales como ProgrammableWeb, Givezilla, Podbop y Strmz.
- Dé un vistazo a la Herramienta de Mashups empresarial de IBM.
- Dé un vistazo a las APIs de búsqueda y mapas de Microsoft, Yahoo! Y Google que puede usar en sus aplicaciones de mashups.
- Use las APIs de Technorati para buscar todos los blogs que vinculen a un sitio específico, busque la mención de ciertas palabras en blogs, vea cuáles blogs están vinculados con un blog dado y busque blogs asociados con un sitio Web específico.
- Use la API de Backpack para que le ayude a organizar tareas y eventos, planear su itinerario, colaborar con otros, monitorear a sus competidores en línea, y mucho más.

Centro de recursos Deitel sobre RSS

Las transmisiones RSS son también fuentes de información populares para los mashups. Para aprender más acerca de las transmisiones RSS, visite nuestro Centro de recursos de RSS en www.deitel.com/RSS/. Cada semana anunciamos el (los) Centro(s) de Recursos más reciente(s) en nuestro boletín de correo electrónico gratuito, *Deitel Buzz Online*:

www.deitel.com/newsletter/subscribe.html

Envíe sus sugerencias sobre Centros de recursos adicionales y mejoras a los Centros de recursos existentes a deitel@deitel.com. ¡Muchas gracias!

Cuestiones de rendimiento y confiabilidad de los mashups

Hay varios retos a vencer al crear aplicaciones de mashups. Sus aplicaciones se hacen susceptibles a los problemas de tráfico y confiabilidad en Internet; circunstancias que, por lo general, están fuera de su control. Las compañías podrían cambiar repentinamente las APIs que sus aplicaciones utilizan. Su aplicación depende de las herramientas de hardware y software de otras compañías. Además, las compañías podrían establecer estructuras de cuotas para servicios Web anteriormente gratuitos, o podrían incrementar las cuotas existentes.

Tutoriales sobre mashups

En ésta y en las siguientes secciones, listaremos una gran cantidad de recursos sobre mashups, de nuestro Centro de recursos sobre mashups. Una vez que haya dominado los servicios Web en el capítulo 28, encontrará que la creación de mashups es un proceso bastante simple. Para cada API que desee utilizar, sólo visite el sitio correspondiente, regístrese y obtenga su clave de acceso (si se requiere), dé un vistazo a las implementaciones de ejemplo y asegúrese de aceptar sus acuerdos de “condiciones del servicio”.

www.programmableweb.com/howto

El tutorial “How to Make your Own Web Mashup”, de Programmableweb.com, es un tutorial de 5 pasos para crear un mashup. Los temas incluyen seleccionar un asunto, buscar sus datos, analizar sus habilidades de codificación, registrarse para una API, y empezar a codificar. Incluye una lista de APIs disponibles.

blogs.msdn.com/jhawk/archive/2006/03/26/561658.aspx

El tutorial “Building a Mashup of National Parks Using the Atlas Virtual Earth Map Control” de Jonathan Hawkins le muestra cómo mostrar chinchetas en un mapa de Microsoft Virtual Earth. Incluye un breve recorrido de la aplicación y una guía paso a paso para crear esta aplicación (incluye código en C#).

www-128.ibm.com/developerworks/edu/x-dw-x-ultimashup1.html?ca=dgr1nxw07WebMashupsPart1&s_cmp=gr&s_tact=105agx59

El tutorial “The Ultimate Mashup—Web Services and the Semantic Web” de seis partes, publicado por IBM, trata acerca del concepto de los mashups, crear una caché de XML, RDF, Web Ontology Language (OWL), control de usuario y mucho más. El tutorial es principalmente para empleados de IBM; otros deben registrarse.

conferences.oreillynet.com/cs/et2005/view/e_sess/6241

Descargue la presentación sobre Mashups de la Conferencia sobre tecnologías emergentes de O'Reilly.

www.theurer.cc/blog/2005/11/03/how-to-build-a-maps-mash-up/

Tutorial “How to Build a Maps Mashup”, por Dan Theurer. Incluye el código de JavaScript y una aplicación mashup de ejemplo.

Directorios sobre mashups

www.programmableweb.com/mashups

ProgrammableWeb (www.programmableweb.com/mashups) lista los mashups y APIs más recientes, además de las noticias y desarrollos Web. Incluye un directorio de nuevos mashups, los mashups más populares y otras cosas más. Puede buscar mashups por etiquetas comunes, incluyendo mapas, fotografías, búsqueda, compras, deportes, viajes, mensajería, noticias, tránsito y bienes raíces. De un vistazo a la matriz de Web 2.0 Mashup, con vínculos a numerosos mashups. Para cada sitio, encontrará los mashups que se han creado con los otros sitios en la matriz.

www.programmableweb.com/matrix

ProgrammableWeb incluye una matriz Web 2.0 Mashup con vínculos a numerosos mashups. Para cada uno de los sitios listados, puede buscar las mashups que se hayan creado con otros sitios en la matriz.

googlemapsmania.blogspot.com/

Lista numerosos mashups que utilizan Google Maps. Algunos ejemplos incluyen mashups de Google Maps con hoteles en EE.UU., información de tránsito, noticias sobre el Reino Unido y mucho más.

www.webmashup.com

Un directorio abierto para mashups y APIs de Web 2.0.

Recursos sobre mashups

code.google.com/

Las APIs de Google incluyen a Google Maps, Google AJAX Search API, Google Toolbar API, AdWords API, Google Data APIs, Google Checkout API y WikiWalki (APIs de google utilizadas en Google Maps).

www.flickr.com/services/api/

Las APIs disponibles de Flickr incluyen la actualización de fotografías, reemplazo de fotografías, peticiones de ejemplo y envío asíncrono. Los kits de APIs incluyen Java, ActionScript, Cold Fusion, Common Lisp, cUrl, Delphi, .NET, Perl, PHO, PHP5, Python, REALbasic y Ruby.

developers.technorati.com/wiki/TechnoratiApi

Las APIs disponibles en Technorati incluyen CosmosQuery, SearchQuery, GetInfoQuery, OutboundQuery y BlogInfoQuery. Las APIs nuevas y experimentales incluyen TagQuery, AttentionQuery y KeyInfo.

mashworks.net/wiki/Building_mashups_for_Non-Programmers

Artículo “Building Mashups for Non-Programmers” de MashWorks. Proporciona fuentes a los no programadores para crear mashups, como vínculos a servicios de mapas y ejemplos de mashups creados por no programadores, mediante el uso de Google Maps y Flickr.

Herramientas y descargas sobre mashups

mashup-tools.pbwiki.com/

Mashup Tools Wiki es una fuente de herramientas y tips para el desarrollador, para crear mashups de tecnología.

news.com.com/2100-1032_3.6046693.html

Artículo: “Yahoo to Offer New Mashup Tools” por Anne Broache. Habla sobre el anuncio de Yahoo de que proporcionará APIs para crear mashups a través de su Red de desarrolladores. Además, Yahoo establecerá una Galería de aplicaciones para ver programas creados con las APIs.

www.imediaconnection.com/content/10217.asp

Artículo: “Marketing Mashup Tools” por Rob Rose. Habla acerca del uso de mashups para comercializar en sitios Web. Los temas incluyen sistemas de búsqueda en el sitio, administración de campañas de correo electrónico, sistemas de administración de contenido, sistemas para análisis de sitios Web y lo que hay que considerar al usar estas herramientas.

datamashups.com/overview.html

Herramienta gratuita: DataMashup.com es un servicio hospedado que ofrece una herramienta de código fuente abierto (AppliBuilder), la cual permite a los usuarios crear mashups. Hay una demo disponible.

blogs.zdnet.com/Hinchcliffe/?p=63

Blog: “Assembling Great Software: A Round-up of Eight Mashup Tools” por Dion Hinchcliffe. Habla acerca de lo que hacen los mashups, los sitios de origen de APIs tales como ProgrammableWeb, y su reseña de ocho herramientas de mashups, incluyendo Above All Studio (de Above All Software), Dapper (una herramienta para mashups en línea), DataMashups.com (excelente para mashups de aplicaciones pequeñas de negocios), JackBuilder (de JackBe): una herramienta de mashups basados en navegador, aRex (de Nexaweb), Process Engine (de Procession) para automatización de tareas, Ratchet-X Studio (de RatchetSoft) para la rápida integración de las aplicaciones, y RSSBus (de RSSBus) para crear mashups a partir de transmisiones RSS.

www.ning.com

Use esta herramienta gratuita para crear sus propias “aplicaciones sociales”. Dé un vistazo a algunas de las aplicaciones que han creado las personas mediante el uso de Ning, incluyendo un mapa de las rutas de excursiones en el área de la bahía de San Francisco, reseñas de restaurantes con mapas, y mucho más. El cofundador de Ning fue Marc Andressen; uno de los fundadores de Netscape.

Artículos sobre mashups

www.factiva.com/infopro/articles/Sept2006Feature.asp?node=menuElem1103

Artículo: “Mashups—The API Buffer”, de Factiva. Explica qué son los mashups y cómo se crean.

[www-128.ibm.com/developerworks/library/x-mashups.html?](http://www-128.ibm.com/developerworks/library/x-mashups.html?ca=dgr1nxw16MashupChallenges)

[ca=dgr1nxw16MashupChallenges](http://www-128.ibm.com/developerworks/library/x-mashups.html?ca=dgr1nxw16MashupChallenges)

Artículo: “Mashups: The New Breed of Web App: An Introduction to Mashups” por Duane Merrill. Habla sobre lo que son los mashups, los tipos de mashups (mapas, video, fotografía, búsqueda, compras y noticias), las tecnologías relacionadas (como la arquitectura, AJAX, protocolos Web, screen scraping, Web semántica, RDF, RSS y ATOM) y los desafíos técnicos y sociales.

ajax.sys-con.com/read/203935.htm

Artículo: “Mashup Data Formats: JSON versus XML/XMLHttpRequest” por Daniel B. Markham. Compara las tecnologías JSON (Notación de objetos JavaScript) y XML/XMLHttpRequest para utilizarlas en aplicaciones Web.

www.techsoup.org/learningcenter/webbuilding/page5788.cfm

Artículo: “Mashups: An Easy, Free Way to Create Custom Web Apps”, por Brian Satterfield. Habla sobre los recursos para crear mashups. Lista varios sitios sobre mashups, incluyendo Givezilla (para fines sin lucro), Podbop (listados de conciertos y archivos MP3) y Strmz (video de flujo continuo, o “streaming video”, blogs de video y podcasts de video).

www.msnbc.msn.com/id/11569228/site/newsweek/

Artículo: “Technology: Time For Your Mashup?” por N’gai Croal. Habla sobre la historia de los mashups, los mashups de música, de video y las aplicaciones Web.

www.slate.com/id/2114791/

Artículo sobre “newsmashing”: un mashup de blogs con las historias de noticias a las cuales hacen referencia. Esto nos permite ver un artículo completo y leer comentarios relacionados de la blogósfera.

images.businessweek.com/ss/05/07/mashups/index_01.htm

Artículo de Business Week Online titulado “Sampling the Web’s Best Mashups”, en el cual se listan mashups populares.

www.usatoday.com/tech/columnist/kevinmaney/2005-08-16-maney-google-mashups_x.htm

Artículo que habla sobre la proliferación de los mashups de Google Maps.

www.clickz.com/experts/brand/brand/article.php/3528921

Artículo titulado “The Branding and Mapping Mashup”. Habla sobre cómo se utilizan los mashups para llevar marcas a los usuarios con base en su ubicación. Por ejemplo, los usuarios pueden buscar la gasolina más económica en su área.

www.usernomics.com/news/2005/10/mash-up-apps-and-competitive-advantage.html

Artículo: “Mashup Apps and Competitive Advantage: Benefits of mashups including user experience”.

Mashups en la blogósfera

web2.wsj2.com/the_web_20_mashup_ecosystem_ramps_up.htm

En el Blog Web 2.0 de Dion Hinchcliffe (presidente y CTO de Hinchcliffe & Company) se habla sobre los mashups. Incluye un excelente gráfico del Ecosistema de Mashups.

www.techcrunch.com/2005/10/04/ning-1aunches/

Blog que da seguimiento a las compañías y noticias sobre Web 2.0. Estos mensajes publicados hablan acerca de Ning, una herramienta gratuita que podemos usar para crear aplicaciones sociales.

www.engadget.com/entry/1234000917034960/

Aprenda a crear sus propios mashups de Google Maps.

blogs.zdnet.com/web2explorer/?p=16&part=rss&tag=feed&subj=zdblog

Mensaje publicado en el blog de ZDNet, titulado “Fun with mashups”. Incluye vínculos a varios mashups.

FAQs y grupos de noticias sobre mashups

groups.google.com/group/Google-Maps-API?lnk=gschg&hl=en

Grupo de noticias de la API de Google Maps en Google Groups. Converse con otros desarrolladores sobre el uso de la API de Google Maps, obtenga respuestas a sus preguntas y comparta sus aplicaciones con otros.

programmableweb.com/faq

La FAQ sobre los mashups en ProgrammableWeb proporciona una introducción a los mashups y las APIs, y habla acerca de cómo crear sus propias mashups, además de otros temas.

