COMERCIO ELECTRÓNICO CON J2EE

Desarrollo de la Tienda Virtual

Luis Velasco

PRIMERA EDICIÓN: Enero 2005

ISBN 84-689-0672-7

Para Anna,
porque la vida es un sueño…

Índice de Contenidos

PREFACIO

Este libro va dirigido a los alumnos de tercer curso de los estudios de Ingeniería Técnica de Telecomunicación, de la Universidad Pompeu Fabra de Barcelona y plasma los conceptos que deben aprender durante la asignatura de Aplicaciones Telemáticas III, en el desarrollo de una aplicación de comercio electrónico.

El libro describe una aproximación al diseño de aplicaciones web con la plataforma Java™ 2, Enterprise Edition, y está acompañado de una aplicación de ejemplo, La Tienda Virtual. El libro no proporciona información en profundidad del uso de las tecnologías Java, sino que se centra en proporcionar una guía sobre la arquitectura de las aplicaciones. Se asume, por lo tanto, que el lector tiene un conocimiento básico de la plataforma J2EE.

Este libro describe los principios de arquitectura y diseño que se emplean para construir aplicaciones web J2EE y los aplica en el desarrollo de la Tienda Virtual y de su aplicación de Administración de la Tienda.

Mostraremos cómo se realiza la implantación sobre el servidor Apache Tomcat 5.5.4 y utilizando la base de datos MySQL 4.1. Además, para el desarrollo de la aplicación se ha utilizado el Entorno Integrado de Desarrollo NetBeans IDE 4.0. Estas herramientas son gratuitas y están disponibles desde Internet.

En la contraportada del libro puede observarse que la edición se ha realizado bajo licencia pública general (GPL, *General Public License*). Esto implica que se aplican los mismos derechos y obligaciones asociadas a la documentación del sistema operativo Linux, diseñadas por la GNU, esto es, el libro puede ser utilizado con total libertad sin pagar derechos de propiedad intelectual y puede ser extendido por otras personas para cubrir temas adicionales, mejorar el formato, introducir nuevas opciones, etc. En el sitio web de GNU (WWW.GNU.ORG) y en el apartado “Licencia de documentación libre de GNU”, puede accederse a los detalles de licencia y posterior utilización del material para su extensión y enriquecimiento.

LICENCIA DE DOCUMENTACIÓN LIBRE DE GNU

Versión 1.2, Noviembre 2002

> This is an unofficial translation of the GNU Free Documentation License into Spanish. It was not published by the Free Software Foundation, and does not legally state the distribution terms for documentation that uses the GNU FDL -- only the original English text of the GNU FDL does that. However, we hope that this translation will help Spanish speakers understand the GNU FDL better.
>
> Ésta es una traducción no oficial de la GNU Free Document License a Español (Castellano). No ha sido publicada por la Free Software Foundation y no establece legalmente los términos de distribución para trabajos que usen la GFDL (sólo el texto de la versión original en Inglés de la GFDL lo hace). Sin embargo, esperamos que esta traducción ayude los hispanohablantes a entender mejor la GFDL. La versión original de la GFDL esta disponible en la Free Software Foundation.
>
> Esta traducción está basada en una de la versión 1.1 de Igor Támara y Pablo Reyes. Sin embargo la responsabilidad de su interpretación es de Joaquín Seoane.
>
>

A.1. PREÁMBULO

El propósito de esta Licencia es permitir que un manual, libro de texto, u otro documento escrito sea *libre* en el sentido de libertad: asegurar a todo el mundo la libertad efectiva de copiarlo y redistribuirlo, con o sin modificaciones, de manera comercial o no. En segundo término, esta Licencia proporciona al autor y al

editor[2] una manera de obtener reconocimiento por su trabajo, sin que se le considere responsable de las modificaciones realizadas por otros.

Esta Licencia es de tipo *copyleft*, lo que significa que los trabajos derivados del documento deben a su vez ser libres en el mismo sentido. Complementa la Licencia Pública General de GNU, que es una licencia tipo copyleft diseñada para el software libre.

Hemos diseñado esta Licencia para usarla en manuales de software libre, ya que el software libre necesita documentación libre: un programa libre debe venir con manuales que ofrezcan la mismas libertades que el software. Pero esta licencia no se limita a manuales de software; puede usarse para cualquier texto, sin tener en cuenta su temática o si se publica como libro impreso o no. Recomendamos esta licencia principalmente para trabajos cuyo fin sea instructivo o de referencia.

A.2. APLICABILIDAD Y DEFINICIONES

Esta Licencia se aplica a cualquier manual u otro trabajo, en cualquier soporte, que contenga una nota del propietario de los derechos de autor que indique que puede ser distribuido bajo los términos de esta Licencia. Tal nota garantiza en cualquier lugar del mundo, sin pago de derechos y sin límite de tiempo, el uso de dicho trabajo según las condiciones aquí estipuladas. En adelante la palabra *Documento* se referirá a cualquiera de dichos manuales o trabajos. Cualquier persona es un licenciatario y será referido como *Usted*. Usted acepta la licencia si copia. modifica o distribuye el trabajo de cualquier modo que requiera permiso según la ley de propiedad intelectual.

Una *Versión Modificada* del Documento significa cualquier trabajo que contenga el Documento o una porción del mismo, ya sea una copia literal o con modificaciones y/o traducciones a otro idioma.

Una *Sección Secundaria* es un apéndice con título o una sección preliminar del Documento que trata exclusivamente de la relación entre los autores o editores y el tema general del Documento (o temas relacionados) pero que no contiene nada que entre directamente en dicho tema general (por ejemplo, si el Documento es en parte un texto de matemáticas, una Sección Secundaria puede no explicar nada de matemáticas). La relación puede ser una conexión histórica con el tema o temas relacionados, o una opinión legal, comercial, filosófica, ética o política acerca de ellos.

Las *Secciones Invariantes* son ciertas Secciones Secundarias cuyos títulos son designados como Secciones Invariantes en la nota que indica que el documento es liberado bajo esta Licencia. Si una sección no entra en la definición de Secundaria, no puede designarse como Invariante. El documento puede no tener Secciones Invariantes. Si el Documento no identifica las Secciones Invariantes, es que no las tiene.

Los *Textos de Cubierta* son ciertos pasajes cortos de texto que se listan como Textos de Cubierta Delantera o Textos de Cubierta Trasera en la nota que indica que el documento es liberado bajo esta Licencia. Un Texto de Cubierta Delantera puede tener como mucho 5 palabras, y uno de Cubierta Trasera puede tener hasta 25 palabras.

Una copia *Transparente* del Documento, significa una copia para lectura en máquina, representada en un formato cuya especificación está disponible al público en general, apto para que los contenidos puedan ser vistos y editados directamente con editores de texto genéricos o (para imágenes compuestas por puntos) con programas genéricos de manipulación de imágenes o (para dibujos) con algún editor de dibujos ampliamente disponible, y que sea adecuado como entrada para formateadores de texto o para su traducción automática a formatos adecuados para formateadores de texto. Una copia hecha en un formato definido como Transparente, pero cuyo marcaje o ausencia de él haya sido diseñado para impedir o dificultar modificaciones posteriores por parte de los lectores no es Transparente. Un formato de imagen no es Transparente si se usa para una cantidad de texto sustancial. Una copia que no es *Transparente* se denomina *Opaca*.

Como ejemplos de formatos adecuados para copias Transparentes están ASCII puro sin marcaje, formato de entrada de Texinfo, formato de entrada de LaTeX, SGML o XML usando una DTD disponible públicamente, y HTML, PostScript o PDF simples, que sigan los estándares y diseñados para que los modifiquen personas. Ejemplos de formatos de imagen transparentes son PNG, XCF y JPG. Los formatos Opacos incluyen formatos propietarios que pueden ser leídos y editados únicamente en procesadores de palabras propietarios, SGML o XML para los cuáles las DTD y/o herramientas de procesamiento no estén ampliamente disponibles, y HTML, PostScript o PDF generados por algunos procesadores de palabras sólo como salida.

La *Portada* significa, en un libro impreso, la página de título, más las páginas siguientes que sean necesarias para mantener legiblemente el material que esta Licencia requiere en la portada. Para trabajos en formatos que no tienen página de portada como tal, *Portada* significa el texto cercano a la aparición más prominente del título del trabajo, precediendo el comienzo del cuerpo del texto.

Una sección *Titulada XYZ* significa una parte del Documento cuyo título es precisamente XYZ o contiene XYZ entre paréntesis, a continuación de texto que traduce XYZ a otro idioma (aquí XYZ se refiere a nombres de sección específicos mencionados más abajo, como *Agradecimientos*, *Dedicatorias* , *Aprobaciones* o *Historia*. *Conservar el Título* de tal sección cuando se modifica el Documento significa que permanece una sección *Titulada XYZ* según esta definición[3].

El Documento puede incluir Limitaciones de Garantía cercanas a la nota donde se declara que al Documento se le aplica esta Licencia. Se considera que estas Limitaciones de Garantía están incluidas, por referencia, en la Licencia, pero sólo en cuanto a limitaciones de garantía: cualquier otra implicación que estas Limitaciones de Garantía puedan tener es nula y no tiene efecto en el significado de esta Licencia.

A.3. COPIA LITERAL

Usted puede copiar y distribuir el Documento en cualquier soporte, sea en forma comercial o no, siempre y cuando esta Licencia, las notas de copyright y la nota que indica que esta Licencia se aplica al Documento se reproduzcan en todas las copias y que usted no añada ninguna otra condición a las expuestas en esta Licencia. Usted no puede usar medidas técnicas para obstruir o controlar la lectura o copia posterior de las copias que usted haga o distribuya. Sin embargo, usted puede

aceptar compensación a cambio de las copias. Si distribuye un número suficientemente grande de copias también deberá seguir las condiciones de la sección 3.

Usted también puede prestar copias, bajo las mismas condiciones establecidas anteriormente, y puede exhibir copias públicamente.

A.4. COPIADO EN CANTIDAD

Si publica copias impresas del Documento (o copias en soportes que tengan normalmente cubiertas impresas) que sobrepasen las 100, y la nota de licencia del Documento exige Textos de Cubierta, debe incluir las copias con cubiertas que lleven en forma clara y legible todos esos Textos de Cubierta: Textos de Cubierta Delantera en la cubierta delantera y Textos de Cubierta Trasera en la cubierta trasera. Ambas cubiertas deben identificarlo a Usted clara y legiblemente como editor de tales copias. La cubierta debe mostrar el título completo con todas las palabras igualmente prominentes y visibles. Además puede añadir otro material en las cubiertas. Las copias con cambios limitados a las cubiertas, siempre que conserven el título del Documento y satisfagan estas condiciones, pueden considerarse como copias literales.

Si los textos requeridos para la cubierta son muy voluminosos para que ajusten legiblemente, debe colocar los primeros (tantos como sea razonable colocar) en la verdadera cubierta y situar el resto en páginas adyacentes.

Si Usted publica o distribuye copias Opacas del Documento cuya cantidad exceda las 100, debe incluir una copia Transparente, que pueda ser leída por una máquina, con cada copia Opaca, o bien mostrar, en cada copia Opaca, una dirección de red donde cualquier usuario de la misma tenga acceso por medio de protocolos públicos y estandarizados a una copia Transparente del Documento completa, sin material adicional. Si usted hace uso de la última opción, deberá tomar las medidas necesarias, cuando comience la distribución de las copias Opacas en cantidad, para asegurar que esta copia Transparente permanecerá accesible en el sitio establecido por lo menos un año después de la última vez que distribuya una copia Opaca de esa edición al público (directamente o a través de sus agentes o distribuidores).

Se solicita, aunque no es requisito, que se ponga en contacto con los autores del Documento antes de redistribuir gran número de copias, para darles la oportunidad de que le proporcionen una versión actualizada del Documento.

A.5. MODIFICACIONES

Puede copiar y distribuir una Versión Modificada del Documento bajo las condiciones de las secciones 2 y 3 anteriores, siempre que usted libere la Versión Modificada bajo esta misma Licencia, con la Versión Modificada haciendo el rol del Documento, por lo tanto dando licencia de distribución y modificación de la Versión Modificada a quienquiera posea una copia de la misma. Además, debe hacer lo siguiente en la Versión Modificada:

- A. Usar en la Portada (y en las cubiertas, si hay alguna) un título distinto al del Documento y de sus versiones anteriores (que deberían, si hay alguna,

estar listadas en la sección de Historia del Documento). Puede usar el mismo título de versiones anteriores al original siempre y cuando quien las publicó originalmente otorgue permiso.

- B. Listar en la Portada, como autores, una o más personas o entidades responsables de la autoría de las modificaciones de la Versión Modificada, junto con por lo menos cinco de los autores principales del Documento (todos sus autores principales, si hay menos de cinco), a menos que le eximan de tal requisito.

- C. Mostrar en la Portada como editor el nombre del editor de la Versión Modificada.

- D. Conservar todas las notas de copyright del Documento.

- E. Añadir una nota de copyright apropiada a sus modificaciones, adyacente a las otras notas de copyright.

- F. Incluir, inmediatamente después de las notas de copyright, una nota de licencia dando el permiso para usar la Versión Modificada bajo los términos de esta Licencia, como se muestra en la Adenda al final de este documento.

- G. Conservar en esa nota de licencia el listado completo de las Secciones Invariantes y de los Textos de Cubierta que sean requeridos en la nota de Licencia del Documento original.

- H. Incluir una copia sin modificación de esta Licencia.

- I. Conservar la sección Titulada *Historia*, conservar su Título y añadirle un elemento que declare al menos el título, el año, los nuevos autores y el editor de la Versión Modificada, tal como figuran en la Portada. Si no hay una sección Titulada *Historia* en el Documento, crear una estableciendo el título, el año, los autores y el editor del Documento, tal como figuran en su Portada, añadiendo además un elemento describiendo la Versión Modificada, como se estableció en la oración anterior.

- J. Conservar la dirección en red, si la hay, dada en el Documento para el acceso público a una copia Transparente del mismo, así como las otras direcciones de red dadas en el Documento para versiones anteriores en las que estuviese basado. Pueden ubicarse en la sección *Historia*. Se puede omitir la ubicación en red de un trabajo que haya sido publicado por lo menos cuatro años antes que el Documento mismo, o si el editor original de dicha versión da permiso.

- K. En cualquier sección Titulada *Agradecimientos* o *Dedicatorias*, Conservar el Título de la sección y conservar en ella toda la sustancia y el tono de los agradecimientos y/o dedicatorias incluidas por cada contribuyente.

- L. Conservar todas las Secciones Invariantes del Documento, sin alterar su texto ni sus títulos. Números de sección o el equivalente no son considerados parte de los títulos de la sección.

- M. Borrar cualquier sección titulada *Aprobaciones*. Tales secciones no pueden estar incluidas en las Versiones Modificadas.

- N. No cambiar el título de ninguna sección existente a *Aprobaciones* ni a uno que entre en conflicto con el de alguna Sección Invariante.

- O. Conservar todas las Limitaciones de Garantía.

Si la Versión Modificada incluye secciones o apéndices nuevos que califiquen como Secciones Secundarias y contienen material no copiado del Documento, puede opcionalmente designar algunas o todas esas secciones como invariantes. Para hacerlo, añada sus títulos a la lista de Secciones Invariantes en la nota de licencia de la Versión Modificada. Tales títulos deben ser distintos de cualquier otro título de sección.

Puede añadir una sección titulada *Aprobaciones*, siempre que contenga únicamente aprobaciones de su Versión Modificada por otras fuentes --por ejemplo, observaciones de peritos o que el texto ha sido aprobado por una organización como la definición oficial de un estándar.

Puede añadir un pasaje de hasta cinco palabras como Texto de Cubierta Delantera y un pasaje de hasta 25 palabras como Texto de Cubierta Trasera en la Versión Modificada. Una entidad solo puede añadir (o hacer que se añada) un pasaje al Texto de Cubierta Delantera y uno al de Cubierta Trasera. Si el Documento ya incluye un textos de cubiertas añadidos previamente por usted o por la misma entidad que usted representa, usted no puede añadir otro; pero puede reemplazar el anterior, con permiso explícito del editor que agregó el texto anterior.

Con esta Licencia ni los autores ni los editores del Documento dan permiso para usar sus nombres para publicidad ni para asegurar o implicar aprobación de cualquier Versión Modificada.

A.6. COMBINACIÓN DE DOCUMENTOS

Usted puede combinar el Documento con otros documentos liberados bajo esta Licencia, bajo los términos definidos en la sección 4 anterior para versiones modificadas, siempre que incluya en la combinación todas las Secciones Invariantes de todos los documentos originales, sin modificar, listadas todas como Secciones Invariantes del trabajo combinado en su nota de licencia. Así mismo debe incluir la Limitación de Garantía.

El trabajo combinado necesita contener solamente una copia de esta Licencia, y puede reemplazar varias Secciones Invariantes idénticas por una sola copia. Si hay varias Secciones Invariantes con el mismo nombre pero con contenidos diferentes, haga el título de cada una de estas secciones único añadiéndole al final del mismo, entre paréntesis, el nombre del autor o editor original de esa sección, si es conocido, o si no, un número único. Haga el mismo ajuste a los títulos de sección en la lista de Secciones Invariantes de la nota de licencia del trabajo combinado.

En la combinación, debe combinar cualquier sección Titulada *Historia* de los documentos originales, formando una sección Titulada *Historia*; de la misma forma

combine cualquier sección Titulada *Agradecimientos*, y cualquier sección Titulada *Dedicatorias*. Debe borrar todas las secciones tituladas *Aprobaciones*.

A.7. COLECCIONES DE DOCUMENTOS

Puede hacer una colección que conste del Documento y de otros documentos liberados bajo esta Licencia, y reemplazar las copias individuales de esta Licencia en todos los documentos por una sola copia que esté incluida en la colección, siempre que siga las reglas de esta Licencia para cada copia literal de cada uno de los documentos en cualquiera de los demás aspectos.

Puede extraer un solo documento de una de tales colecciones y distribuirlo individualmente bajo esta Licencia, siempre que inserte una copia de esta Licencia en el documento extraído, y siga esta Licencia en todos los demás aspectos relativos a la copia literal de dicho documento.

A.8. AGREGACIÓN CON TRABAJOS INDEPENDIENTES

Una recopilación que conste del Documento o sus derivados y de otros documentos o trabajos separados e independientes, en cualquier soporte de almacenamiento o distribución, se denomina un *agregado* si el copyright resultante de la compilación no se usa para limitar los derechos de los usuarios de la misma más allá de lo que los de los trabajos individuales permiten. Cuando el Documento se incluye en un agregado, esta Licencia no se aplica a otros trabajos del agregado que no sean en sí mismos derivados del Documento.

Si el requisito de la sección 3 sobre el Texto de Cubierta es aplicable a estas copias del Documento y el Documento es menor que la mitad del agregado entero, los Textos de Cubierta del Documento pueden colocarse en cubiertas que enmarquen solamente el Documento dentro del agregado, o el equivalente electrónico de las cubiertas si el documento está en forma electrónica. En caso contrario deben aparecer en cubiertas impresas enmarcando todo el agregado.

A.9. TRADUCCIÓN

La Traducción es considerada como un tipo de modificación, por lo que usted puede distribuir traducciones del Documento bajo los términos de la sección 4. El reemplazo las Secciones Invariantes con traducciones requiere permiso especial de los dueños de derecho de autor, pero usted puede añadir traducciones de algunas o todas las Secciones Invariantes a las versiones originales de las mismas. Puede incluir una traducción de esta Licencia, de todas las notas de licencia del documento, así como de las Limitaciones de Garantía, siempre que incluya también la versión en Inglés de esta Licencia y las versiones originales de las notas de licencia y Limitaciones de Garantía. En caso de desacuerdo entre la traducción y la versión original en Inglés de esta Licencia, la nota de licencia o la limitación de garantía, la versión original en Inglés prevalecerá.

Si una sección del Documento está Titulada *Agradecimientos*, *Dedicatorias* o *Historia* el requisito (sección 4) de Conservar su Título (Sección 1) requerirá, típicamente, cambiar su título.

A.10. TERMINACIÓN

Usted no puede copiar, modificar, sublicenciar o distribuir el Documento salvo por lo permitido expresamente por esta Licencia. Cualquier otro intento de copia, modificación, sublicenciamiento o distribución del Documento es nulo, y dará por terminados automáticamente sus derechos bajo esa Licencia. Sin embargo, los terceros que hayan recibido copias, o derechos, de usted bajo esta Licencia no verán terminadas sus licencias, siempre que permanezcan en total conformidad con ella.

A.11. REVISIONES FUTURAS DE ESTA LICENCIA

De vez en cuando la Free Software Foundation puede publicar versiones nuevas y revisadas de la Licencia de Documentación Libre GNU. Tales versiones nuevas serán similares en espíritu a la presente versión, pero pueden diferir en detalles para solucionar nuevos problemas o intereses. Vea http://www.gnu.org/copyleft/.

Cada versión de la Licencia tiene un número de versión que la distingue. Si el Documento especifica que se aplica una versión numerada en particular de esta licencia o *cualquier versión posterior*, usted tiene la opción de seguir los términos y condiciones de la versión especificada o cualquiera posterior que haya sido publicada (no como borrador) por la Free Software Foundation. Si el Documento no especifica un número de versión de esta Licencia, puede escoger cualquier versión que haya sido publicada (no como borrador) por la Free Software Foundation.

A.12. ADENDA: Cómo usar esta Licencia en sus documentos

Para usar esta licencia en un documento que usted haya escrito, incluya una copia de la Licencia en el documento y ponga el siguiente copyright y nota de licencia justo después de la página de título:

> Copyright (c) AÑO SU NOMBRE. Se otorga permiso para copiar, distribuir y/o modificar este documento bajo los términos de la Licencia de Documentación Libre de GNU, Versión 1.2 o cualquier otra versión posterior publicada por la Free Software Foundation; sin Secciones Invariantes ni Textos de Cubierta Delantera ni Textos de Cubierta Trasera. Una copia de la licencia está incluida en la sección titulada GNU Free Documentation License.

Si tiene Secciones Invariantes, Textos de Cubierta Delantera y Textos de Cubierta Trasera, reemplace la frase *sin ... Trasera* por esto:

> siendo las Secciones Invariantes LISTE SUS TÍTULOS, siendo los Textos de Cubierta Delantera LISTAR, y siendo sus Textos de Cubierta Trasera LISTAR.

Si tiene Secciones Invariantes sin Textos de Cubierta o cualquier otra combinación de los tres, mezcle ambas alternativas para adaptarse a la situación.

Si su documento contiene ejemplos de código de programa no triviales, recomendamos liberar estos ejemplos en paralelo bajo la licencia de software libre que usted elija, como la Licencia Pública General de GNU (*GNU General Public License*), para permitir su uso en software libre.

Notas

[1] Ésta es la traducción del Copyright de la Licencia, no es el Copyright de esta traducción no autorizada.

[2] La licencia original dice *publisher*, que es, estrictamente, quien publica, diferente de *editor*, que es más bien quien prepara un texto para publicar. En castellano *editor* se usa para ambas cosas.

[3] En sentido estricto esta licencia parece exigir que los títulos sean exactamente *Acknowledgements*, *Dedications*, *Endorsements* e *History*, en inglés.

CAPÍTULO 1

Desarrollo de Aplicaciones Web

1.1 ARQUITECTURA

A lo largo este libro vamos a desarrollar una aplicación web de comercio electrónico mediante el modelo de arquitectura 2, una arquitectura Modelo-Vista-Controlador (MVC) que separa la generación del contenido de su presentación.

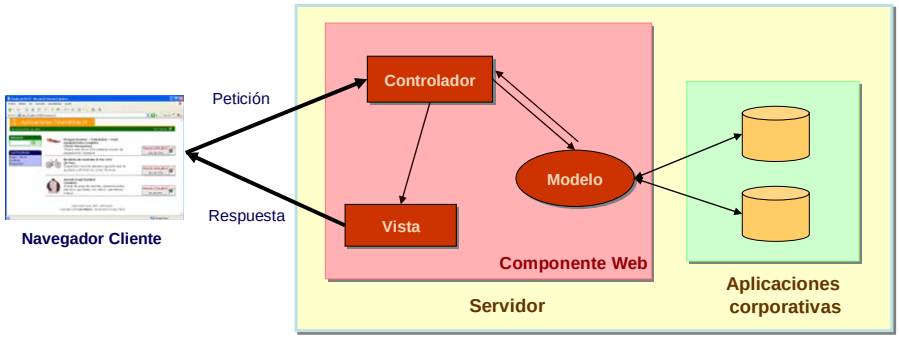

Figura 1-1 Arquitectura MVC

Un modelo de arquitectura 2 está indicado por la presencia de un controlador entre el navegador del cliente y las vistas que presentan el contenido. El controlador recibe cada petición HTTP e invoca la operación del modelo solicitada. A continuación selecciona la siguiente vista a mostrar en base al resultado de la operación y estado de la sesión. Este proceso se observa en la siguiente figura

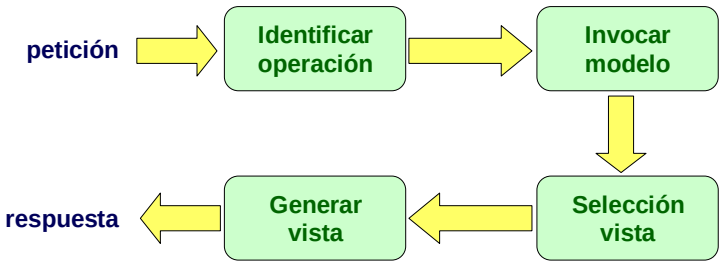

Figura 1-2 Ciclo de servicio

Las vistas muestran los datos producidos por el modelo, normalmente, con una apariencia común y, en este modelo, las vistas (páginas JSP o servlets) están aisladas unas de otras.

Una vista compuesta compone subvistas separadas en una página con un diseño específico. Cada subvista (navegación, cuerpo, etc.) es un componente separado.

La vista compuesta controla la apariencia de todas las subsistas, de forma que se centraliza el control sobre la apariencia de toda la aplicación. Además, las subvistas se utilizan por referencia, en vez de mediante copy-paste.

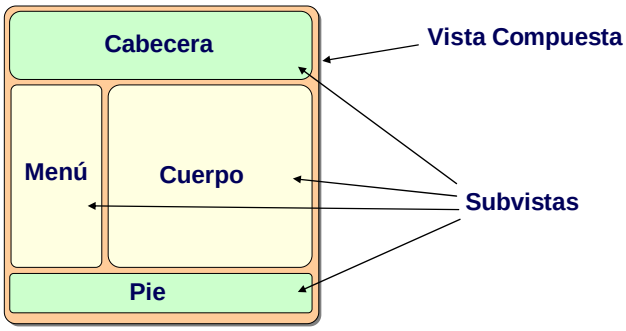

Figura 1-3 Vista compuesta

El Modelo representa los datos y las reglas de negocio. En las aplicaciones web, el modelo estará formado, generalmente, por un conjunto de clases que acceden a los datos de una forma consistente.

Las aplicaciones del modelo 2 son más flexibles y fáciles de mantener y extender, ya que las vistas no se referencian unas a otras de forma directa. El servlet controlador del modelo 2 proporciona un punto único de control para la seguridad y logs y, frecuentemente encapsula datos de entrada para que sean utilizados por componentes del modelo MVC.

Es verdad que el modelo de arquitectura 2 añade algo de complejidad a la aplicación, sin embargo existen frameworks de aplicación MVC que simplifican considerablemente la implementación de una aplicación de modelo 2.

El modelo de arquitectura 2 es el recomendado para aplicaciones complejas. En este documento veremos cómo este modelo de arquitectura se aplica al desarrollo de nuestro proyecto de tienda virtual.

1.2 PATRONES DE DISEÑO

Un patrón de diseño describe una solución probada a un problema recurrente de diseño. Los patrones de diseño contribuyen a reutilizar diseño, identificando aspectos claves de la estructura de un diseño que puede ser aplicado en una gran cantidad de situaciones. De esta forma se reducen los esfuerzos de desarrollo y mantenimiento, se mejora la seguridad, eficiencia y consistencia de nuestros diseños, y se alcanza un considerable ahorro en la inversión.

Los patrones de diseño describen problemas individuales, pero pueden combinarse de formas diferentes para proporcionar una solución completa.

Se han definido un conjunto completo de patrones de diseño para la plataforma J2EE. Aquí mencionaremos únicamente aquellos que aplicaremos en el desarrollo de nuestro proyecto de tienda virtual.

- **Intercepting filter**: aplica un pre y post procesado a las peticiones y proporciona servicios adicionales necesarios para procesar la petición. Por

ejemplo, un *intercepting filter* como un servlet filter, puede gestionar todas las peticiones entrantes a la aplicación web y proporcionar un mecanismo central de autorización y logging.

- **Front controller**: proporciona un control centralizado para gestionar las peticiones. Un front Controller recibe todas las peticiones entrantes de clientes, distribuye cada petición a un gestor de petición apropiado y presenta la respuesta al cliente.

- **View helper**: Encapsula las lógicas de presentación y el acceso a datos de las vistas, haciendo que las vistas sean más simples. La lógica de presentación formatea los datos para su presentación, mientras que la lógica del acceso a datos recupera los datos. Los view helpler suelen ser custom tags, para formatear datos para la presentación, o JavaBean para recuperar datos o cualquier otra utilidad.

- **Composite view**: Este patrón hace que la presentación sea más manejable creando una plantilla para gestionar los elementos comunes de una vista. A menudo, las páginas web contienen una mezcla de contenido estático y dinámico, como la cabecera, el pie de página, el logotipo, etc. La porción dinámica es particular para cada página, pero los elementos estáticos son los mismos para todas las páginas. La plantilla *composite view* captura las características comunes.

1.3 TECNOLOGÍAS WEB

Las tecnologías web de la plataforma J2EE proporcionan los beneficios de la programación en el lado del servidor, utilizando clases Java en un entorno seguro y estandarizado.

Una aplicación web es una colección de componentes web, contenido e información de configuración, que trabaja como una unidad funcional única. La especificación de la plataforma define un contrato entre el contenedor web y cada componente web que define el ciclo de vida de cada componente, el comportamiento que debe implementar el componente y los servicios que el servidor debe proporcionar al componente.

La especificación de la plataforma también define dos tipos de tecnologías de componentes web: la tecnología de Java Servlets (servlets) y la tecnología JavaServer Pages (páginas JSP).

Un servlet es una clase Java que extiende un Server J2EE, produciendo contenido dinámico en respuesta a peticiones del servidor. El servidor pasa peticiones de servicio al servlet a través de la interfaz estándar `javax.servlet`, que debe implementar cada servlet.

Un servlet también puede ser extendido por uno o más filtros, que son clases reutilizables que filtran las llamadas a un método de servicio de un servlet, transformando la petición o la respuesta. Los filtros de servlet pueden organizarse en cadenas de filtros, que realizan transformaciones sucesivas a las peticiones o las respuestas.

Una página JSP es una página HTML con marcas especiales que proporcionan comportamiento a medida para generar contenido dinámico en tiempo de ejecución. Una página JSP se convierte en un servlet cuando se despliega.

Las páginas JSP se diferencian de los servlet en su modelo de programación. Una página JSP es principalmente un documento que especifica contenido dinámico, en vez de un programa que produce contenido.

La tecnología JSP permite que los desarrolladores definan etiquetas a medida, *custom tags*, es decir, etiquetas que son reemplazadas por contenido dinámico cuando se sirve la página. El contenido dinámico es creado por una clase que maneja la etiqueta, *tag handler*.

Un programador define la sintaxis para una etiqueta e implementa el comportamiento de la etiqueta en la clase handler. A continuación, los autores de páginas pueden importar y utilizar las etiquetas, desde las librerías de etiquetas, como si se utilizase cualquier otra etiqueta.

1.3.1 Pautas de desarrollo

Como hemos visto durante el curso, los servlets están indicados para implementar lógica y generar contenido binario, mientras que las páginas JSP están más indicadas para generar contenidos de texto.

A continuación se resumen algunas pautas a tener en cuenta sobre el uso de las tecnologías:

- Utiliza servlets para implementar servicios, por ejemplo, seguridad, personalización, control de la aplicación, etc.
- Utiliza servlets como controladores, para determinar cómo manejar la petición y seleccionar la vista a mostrar.
- Utiliza servlets para generar contenido binario.
- Evita utilizar servlets para generar texto estático. Utiliza páginas JSP para crear contenido de texto estructurado, incluyendo HTML, XHTML y DHTML.
- Utiliza páginas JSP para generar XML, por ejemplo mensajes XML en formatos estándar.
- Utiliza páginas JSP como plantillas, para ensamblar contenidos de texto desde múltiples fuentes.
- Evita el uso de etiquetas lógicas, es decir, custom tags que realizan bucles, iteraciones, evalúan expresiones y toman decisiones. Su uso no aporta beneficios y viola la separación de la lógica y la presentación.
- Utiliza custom tags evitando scriptlets en las páginas JSP, ya que:
  - Los scriptlets no son reusables,

  - Fuerzan el uso de copiar y pegar código dificultando el mantenimiento del código,
  - Mezclan la lógica con la presentación,
  - Rompen la separación de roles del desarrollo,
  - Hacen que las páginas JSP sean difíciles de leer y mantener,
  - Los errores de compilación pueden ser difíciles de interpretar,
  - El código se hace difícil de probar, ya que se debe probar la página completa y comprobar los resultados.

- Evita hacer forward desde páginas JSP. Cuando una página JSP hace forward, tanto directamente como mediante un custom tag, está actuando como controlador. La mejor forma de implementar un controlador es mediante servlets, ya que es un componente lógico no de presentación.

CAPÍTULO 2

Arquitectura de la Tienda Virtual

2.1 INTRODUCCIÓN

La Tienda Virtual que vamos a desarrollar, es una aplicación de comercio electrónico pensada para vender artículos deportivos organizados en diferentes categorías, como los que se podrían encontrar en una tienda de deportes.

El proyecto ha sido desarrollado con motivos educativos y se han obviado cuestiones, como por ejemplo, las relativas a seguridad, trazabilidad, etc. para hacer más legible el código resultante. Por lo tanto, el software está dirigido al aprendizaje, no a su uso en entornos de producción.

2.2 ESPECIFICACIÓN DEL PROYECTO

El objetivo del proyecto es desarrollar una tienda virtual, que venderá artículos deportivos. La aplicación tiene dos interfaces web, a través de la interfaz de Tienda acceden los clientes y mediante la interfaz de Administración acceden los administradores de la Tienda.

Los artículos estarán organizados mediante categorías o departamentos y existirán artículos destacados, que aparecerán en el escaparate de la tienda.

Los visitantes podrán realizar las siguientes operaciones:

- Ver los artículos del escaparate,
- Navegar por la lista de productos organizados por categorías,
- Ver los detalles de un producto,
- Seleccionar un artículo y añadirlo a la cesta de la compra,
- Ver la cesta de la compra y modificar la cantidad de requerida de cada producto.
- Realizar el pedido con los productos de la cesta.

2.2.1 Diseño de la aplicación

El diseño de nuestra tienda virtual comienza evaluando los requisitos funcionales y determinando la implementación óptima para cumplir estos requisitos. Existen numerosas herramientas de análisis para reunir y evaluar requisitos de aplicación. El análisis de casos de uso es una de estas herramientas, e identifica los actores en un sistema y las operaciones que pueden realizar sobre éste.

Nuestra tienda virtual es un caso típico de aplicación de comercio electrónico. El cliente selecciona elementos de un catálogo, los lleva a la cesta de la compra donde selecciona la cantidad de cada artículo y la aplicación muestra el precio unitario y el subtotal y el precio total. Cuando el cliente tiene los artículos que desea en la cesta, realiza el pedido, suministrando los datos de envío y una tarjeta de crédito.

La siguiente figura muestra un diagrama de casos de uso de alto nivel, para nuestra Tienda Virtual. El diagrama muestra los actores potenciales del sistema y sus acciones.

- Un cliente navega por el catálogo, gestiona la cesta y realiza pedidos.
- Un administrador gestiona la tienda.

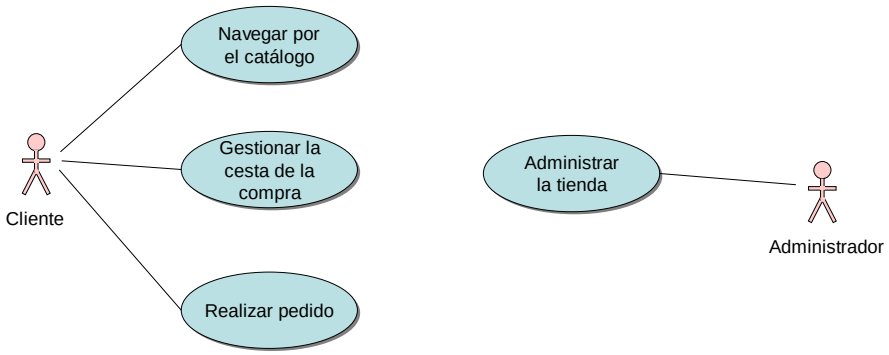

Figura 2-1 Diagrama de casos de uso

Una vez hemos determinado los requisitos del sistema podemos comenzar con el diseño de la aplicación. Puesto que la tienda virtual es una aplicación web interactiva, la arquitectura Modelo-Vista-Controlador es una buena elección.

2.2.2 Arquitectura de la Tienda Virtual

El desarrollo de la arquitectura completa de la aplicación implica subdividir la aplicación en objetos o componentes. El diseño de objetos se convierte en una actividad importante en el desarrollo de aplicaciones complejas. El desarrollo de aplicaciones orientadas a objeto de gran escala, requiere de *frameworks* que definan cómo interactúan los objetos entre sí. El *framework* debe permitir la reutilización del diseño y el código de los objetos, identificando la responsabilidad de cada componente. Además, en aplicaciones multicapa, es importante:

- Separar el código estable del que cambia frecuentemente. Normalmente, las reglas de negocio y esquemas de base de datos cambian mucho menos que la interfaz de usuario y presentación.
- Dividir los esfuerzos de desarrollo por perfiles. Por ejemplo, es común tener un grupo de diseñadores gráficos y desarrolladores HTML que son los responsables de diseñar y construir la interfaz de usuario de la aplicación Web. También existirá un grupo de programadores que escriben el código que añade funcionalidad a la aplicación. Ambos grupos son necesarios para construir aplicaciones web intuitivas y funcionales. Esta división de tareas permite que los diseñadores de páginas modifiquen la presentación de la aplicación sin afectar la funcionalidad que implementa el código Java. Al contrario, sin modificar la interfaz de los componentes, los desarrolladores pueden mejorar la implementación de un componente sin afectar al diseño de la aplicación.

La arquitectura MVC se adapta bien a nuestra tienda virtual. Podemos descomponer nuestro sistema en tres bloques lógicos, el primero es el relativo a los aspectos de presentación, el segundo es aquel que trata con el modelo y los datos, y el tercero es el que recibe e intercepta las peticiones del usuario, y controla el flujo para responder estas peticiones.

Normalmente, el aspecto de la interfaz gráfica de la aplicación cambia a menudo, su comportamiento cambia menos frecuentemente y el modelo de datos es relativamente estable. De esta forma, los diseñadores de páginas web y expertos en tecnología HTML y JSP han de implementar objetos de presentación después de que la aplicación está desplegada.

La siguiente fase del proceso de diseño es partir la aplicación en módulos y objetos que implementen los diferentes requisitos funcionales. La siguiente figura muestra los casos de uso de la interfaz de usuario de la tienda virtual.

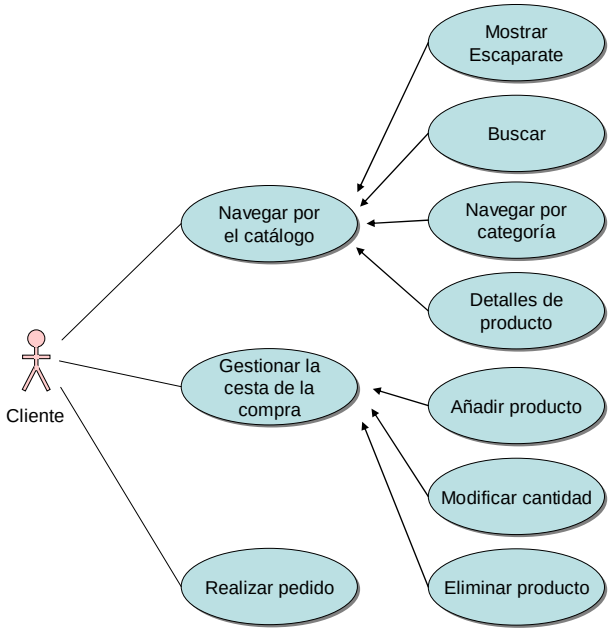

Figura 2-2 Casos de uso de la tienda

Un cliente que accede a la tienda, espera ver los siguientes elementos:

- Barra de navegación de tareas comunes,
- Un catálogo que proporciona una vista organizada de los contenidos de la aplicación,
- Un mecanismo de búsqueda,
- Una vista detallada de cada producto,

- Una vista de la cesta de la compra que deja a los clientes revisar y modificar los contenidos de su cesta.
- Una vista de pedido, que permite a los clientes introducir información sobre la entrega del pedido y medio de pago.
- Una vista que confirma la compra.

Una vez que los requisitos funcionales están identificados, debemos identificar unidades de modelo de datos y lógica de presentación y modelarlos como objetos software. Esto lo haremos identificando las opciones al nivel más alto y después iremos bajando hacia niveles inferiores.

La organización general de la tienda sigue la arquitectura MVC, ya que proporciona la estructura adecuada para manejar aplicaciones orientadas a la presentación complejas. Por otra parte, el diseño interno de algunos de sus componentes individuales siguen patrones J2EE.

El diseño de la aplicación consiste en partir la aplicación en bloques pertenecientes al modelo, las vistas y el controlador, como se muestra en la siguiente figura. Se debe tener en cuenta que estos límites no están siempre excesivamente claros.

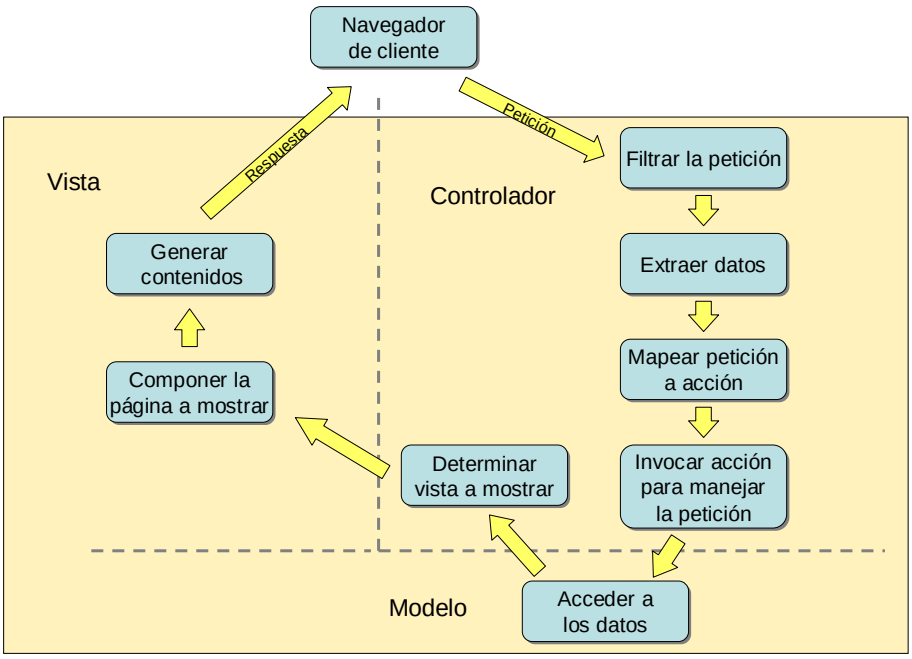

Figura 2-3 Arquitectura MVC de la Tienda

2.2.2.1 Las Vistas

Nuestra tienda virtual utilizará tecnología JavaServer Pages (JSP) para gestionar la presentación de las vistas de usuario. Además, se podría utilizar tecnología Servlets para la generación de gráficos u otras representaciones binarias, en caso necesario.

Cuando se diseñan las vistas de la aplicación, se debe tener en cuenta los siguientes puntos:

- **Separar la lógica de presentación de las lógicas de negocio y de control.**

  La Tienda Virtual utiliza JSP orientados a la presentación de la vista y no contienen lógica de control (la lógica de control la gestiona un controlador). De esta forma, es fácil reutilizar porciones de la lógica de presentación. También es importante evitar poner lógica en scriptlets embebidos.

  La aplicación utiliza un patrón view helpler que encapsula la lógica de presentación en un *custom tag* JSP, clases Java para contener los datos y un *JavaBean* para obtener la fecha y hora del sistema. La lógica de presentación implementada en *custom tags* y/o *JavaBeans* está separada de los datos y es modular y reutilizable, ya que se define una vez y se reutiliza por referencia en múltiples páginas JSP.

- **Gestión de la apariencia de las páginas.**

  El diseño gráfico de la tienda se ha realizado intentando mantener un aspecto común a través de todas las páginas de la aplicación. Todas las páginas presentan la misma estructura: una cabecera la parte más alta, una barra de navegación a la izquierda y un pie de página en la parte inferior de la página. Los contenidos de la página aparecen a la derecha de la barra de navegación entre la cabecera y el pie de página, y son independientes unos de otros.

  La siguiente figura muestra dos páginas de la Tienda Virtual. Aquí podemos ver que presentan un aspecto similar aunque el contenido de ambas es muy diferente.

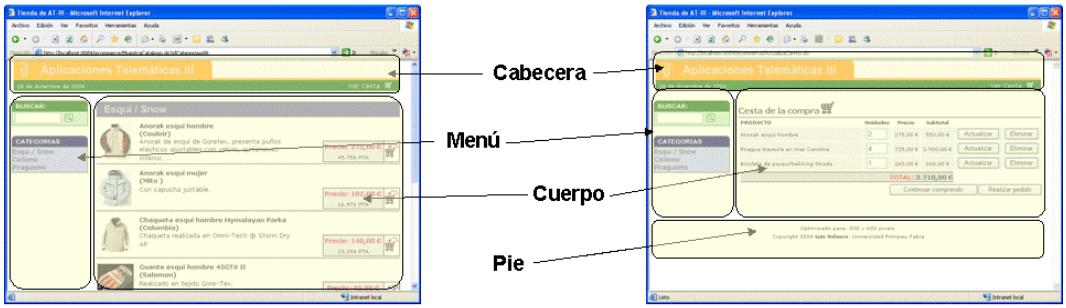

Figura 2-4 Dos ejemplos de páginas de la Tienda

Para que el aspecto de las páginas de la tienda sea similar, se utiliza un mecanismo de plantilla, como es el patrón de diseño de *Vista Compuesta*. La plantilla determina cómo ensamblar las vistas en una única vista compuesta y consolida el aspecto de la aplicación en un único punto, haciendo fácil modificar el aspecto desde un punto centralizado. Además, la plantilla separa el aspecto del contenido.

- **Separación de roles de desarrollo.**

  La arquitectura MVC proporciona los medios para separar el trabajo de los desarrolladores con diferentes perfiles, facilitando que cada grupo trabaje de forma independiente.

  La aplicación utiliza un patrón *View Helpler* para separar datos de presentación de la lógica de negocio, permitiendo que los diseñadores gráficos realicen cambios en la presentación de las vistas sin afectar a la funcionalidad. Por otra parte, los desarrolladores Java pueden implementar la lógica de la aplicación sin afectar el diseño de las páginas.

2.2.2.2 El Modelo

El modelo, en la arquitectura MVC, encapsula objetos de negocio y API de la funcionalidad de la aplicación. La Tienda virtual utiliza clases Java para implementar su lógica de negocio.

Cuando se diseña el modelo de la aplicación, se debe tener en cuenta los siguientes puntos:

- **Desarrollar código como componentes para facilitar la reutilización.**

  El desarrollo de las aplicaciones se mejora si los desarrolladores diseñan el código de forma modular, como componentes reutilizables, para ser independientes unos de otros. De esta forma, cuando los componentes están muy desacoplados entre sí, los cambios que se realizan en un componente no afectan al resto de los componentes.

- **Separación de roles de desarrollo.**

  De la misma forma que en el desarrollo de las vistas, el desarrollo de componentes es más efectivo cuando los roles de desarrollo están separados. Por ejemplo, si los desarrolladores de bases de datos implementan objetos de acceso a datos, se oculta a los desarrolladores de la lógica de negocio los detalles de implementación de las llamadas de acceso a la base de datos.

2.2.2.3 El Controlador

El controlador, en la arquitectura MVC, controla el flujo de la aplicación y sirve como cola de unión entre el modelo y las vistas, ya que ejecuta lógica de negocio en el modelo en respuesta a peticiones de usuario y, a continuación, selecciona la siguiente vista a mostrar. El controlador desacopla la presentación de datos de la lógica de negocio y de datos.

La Tienda Virtual utiliza un servlet como *Controlador*. El servlet controlador recibe peticiones http y las pasa al *Gestor de Flujo*, que selecciona la acción a realizar en base a la operación solicitada y la ejecuta. Después, el *Gestor de Flujo* selecciona la siguiente vista a mostrar y se la pasa de vuelta al servlet controlador, que hace la redirección apropiada.

El controlador de la aplicación incluye también un filtro de peticiones, que proporciona servicios de log a la aplicación, filtra los parámetros recibidos en la petición y los pone como atributos, y redirige la petición al controlador de administración si se recibe la cadena "`/admin/`" en la petición, al controlador de la tienda localizando la operación realizada, o deja pasar la petición si se trata de peticiones de imágenes gif o jpg.

2.2.2.4 Aplicación de la arquitectura MVC a la Tienda Virtual

La arquitectura de nuestra Tienda Virtual contiene un conjunto de componentes divididos en bloques de modelo, vista y controlador. Vamos a examinar la arquitectura de la aplicación in más detalle, siguiendo una petición de usuario recibida. La siguiente figura muestra la descomposición de la Tienda en componentes.

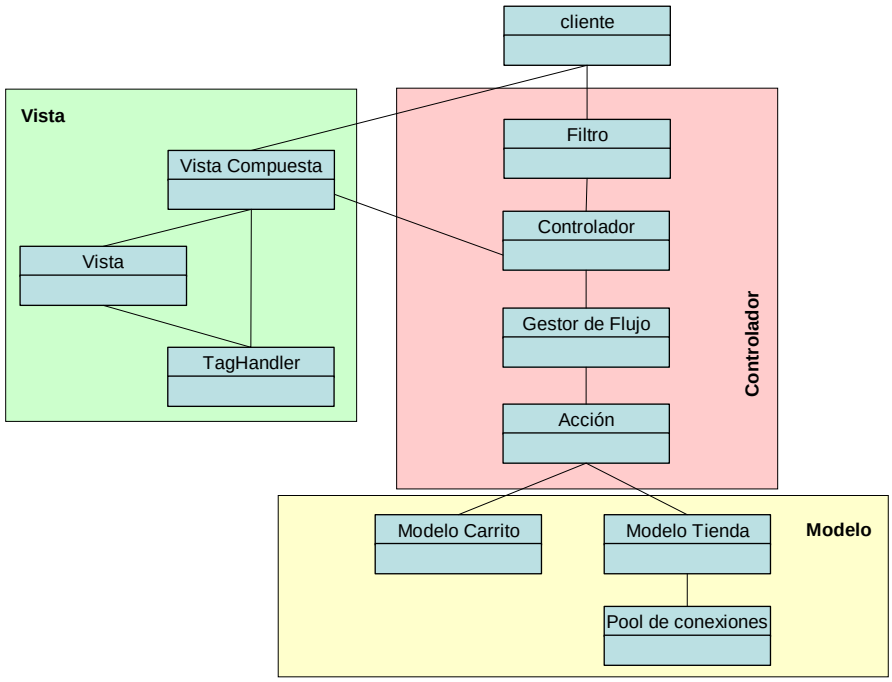

Figura 2-5 Diagrama de clases simplificado de la Tienda

Un cliente interactúa con diferentes vistas presentadas en el navegador y, eventualmente, envía una petición http. La petición va a la sección Controlador donde es interceptada por el Filtro. El Filtro recibe la petición, hace log de la petición y parámetros recibidos y selecciona el controlador (de Tienda o de Administración) o el recurso al que pasar la petición. El controlador de la tienda recibe la petición, recoge la operación a realizar y llama al Gestor de Flujo para que ejecute la operación. El Gestor de Flujo selecciona la Acción a realizar y la ejecuta. La Acción recoge los datos necesarios de la petición y ejecuta la lógica de negocio requerida en el Modelo.

El Modelo accede a los datos almacenados en la base de datos de la tienda mediante un pool de conexiones. También contiene la funcionalidad de la cesta de la compra.

De nuevo en el bloque Controlador, la acción devuelve los resultados de la operación en la propia petición, para que sean mostrados al usuario por la vista apropiada. El Gestor de Flujo selecciona la siguiente vista a mostrar, en función del resultado de la acción, y se la pasa de vuelta al servlet controlador, que hace la redirección apropiada.

En el bloque de Vista, la Vista Compuesta recibe la página de contenido a mostrar y la incluye para formar una única página que mostrar al cliente. La página de contenido recibe los datos y los presenta, ayudándose de custom tag que encapsulan la lógica necesaria y hacen uso de clases Java que encapsulan datos.

2.2.3 Arquitectura de la Administración

Para el diseño de la aplicación de Administración de la Tienda, se han seguido los mismos pasos que para la Tienda Virtual, teniendo en cuenta las siguientes consideraciones:

- El filtro es común para la parte pública (Tienda Virtual) que para la aplicación de Administración.

- El controlador de Administración gestiona la seguridad y limita el acceso a la aplicación a los usuarios registrados que tengan permiso de administración.

- Los datos de usuarios y permisos se almacenan en una base de datos diferente de la de la Tienda y son accedidos mediante un modelo también diferente.

- Se ha optado por no utilizar un Gestor de Flujo que seleccione la acción a ejecutar y la vista a mostrar. Esta funcionalidad se ha incluido en el propio servlet controlador. Para determinar que operación realizar, se utiliza un parámetro denominado “accion”.

- Con objeto de mostrar páginas JSP en las que se mezcla la lógica de negocio con la presentación, no se utilizan custom tags, sino que la funcionalidad está embebida en scriptlets dentro de las páginas.

Las siguientes figuras muestran los casos de uso y la descomposición de la Administración en componentes.

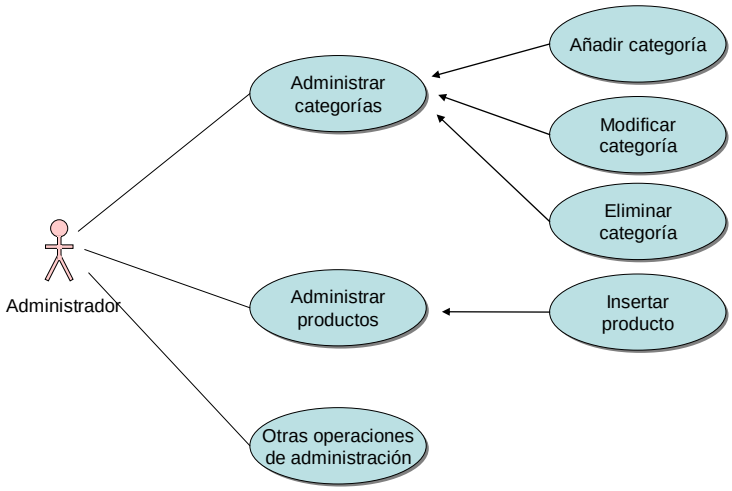

Figura 2-6 Casos de uso de la tienda

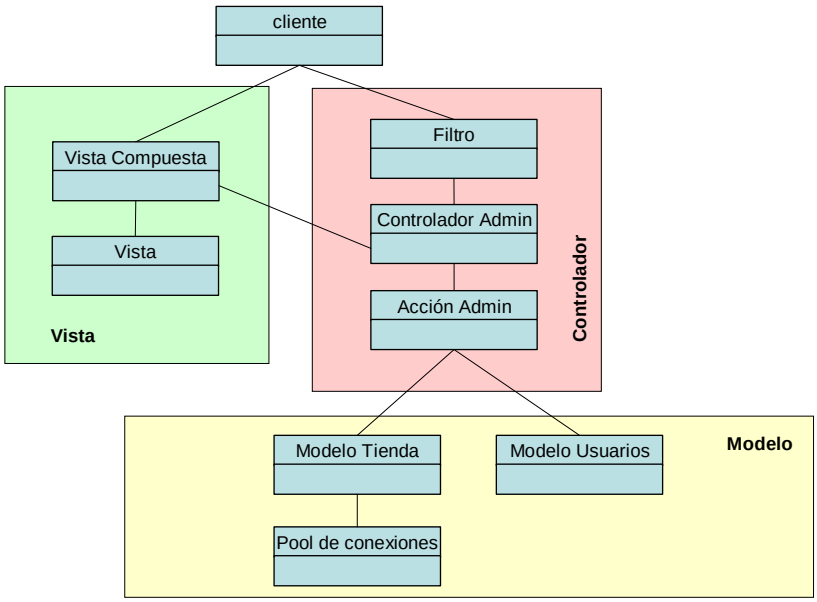

Figura 2-7 Diagrama de clases simplificado de la Administración

CAPÍTULO 3

Organización de la aplicación

3.1 INTRODUCCIÓN

Una vez hemos diseñado la aplicación y decidido su arquitectura, vamos a comenzar con la fase de desarrollo puramente dicha. En esta fase diseñaremos las bases de datos a utilizar, codificaremos las clases Java que hemos definido en el apartado anterior, para los bloques de Modelo, Vista y Controlador, y diseñaremos las páginas HTML que darán el aspecto definitivo a nuestra aplicación de Tienda Virtual y Administración de la Tienda.

3.2 ORGANIZACIÓN DE LA APLICACIÓN

Antes de comenzar vamos a dar un primer vistazo a la organización de la aplicación en cuanto a carpetas y paquetes de clases.

En el directorio "Tienda V.1.3" que acompaña este libro, se encuentra la Tienda Virtual. Bajo este directorio tenemos las siguientes carpetas:

Carpeta	Descripción
buid	Contiene una carpeta ("web") preparada para realizar el despliegue.
db	Contiene los scripts sql para crear y poblar las bases de datos de la Tienda Virtual, sobre MySQL.
dist	Contiene el fichero war y la documentación javadoc.
src	Código fuente de las clases Java.
web	Páginas html, jsp, *deployment descriptor* y otra información auxiliar.

Las carpetas "*web*" y "*src*" contienen el código fuente completo de la aplicación y junto con la carpeta "db" forman el bloque de fuentes de la Tienda Virtual y de la Administración de la Tienda. Las carpetas "*buid*" y "*dist*", han sido generadas por NetBeans. El fichero *war* y la carpeta "dist/web", son equivalentes. A continuación veremos la estructura de las carpetas que contienen los fuentes de la aplicación.

La carpeta "***web***" contiene la siguiente estructura:

Carpeta	Descripción
META-INF	Contiene el fichero `context.xml`, que definen opciones de configuración específicas de Tomcat.
images	Contiene ficheros con las imágenes comunes de la aplicación y dos carpetas denominadas `productos` y `categorías`, que contienen las imágenes de los productos y los rótulos de las categorías de productos.

jsp	Contiene las páginas jsp y html de la aplicación de Tienda Virtual y una subcarpeta `admin` con las páginas jsp de la Administración de la Tienda. En estas carpetas se encuentra la Vista Compuesta de la Tienda y de la Administración, las páginas de cabecera, menú y pie y las distintas vistas específicas de cada una de las acciones que es posible ejecutar sobre la Tienda Virtual y la Administración de la Tienda.
WEB-INF	Contiene los ficheros de configuración de la aplicación, (el *deployment descriptor* web.xml y la configuración de mapeos Accion-Vista).

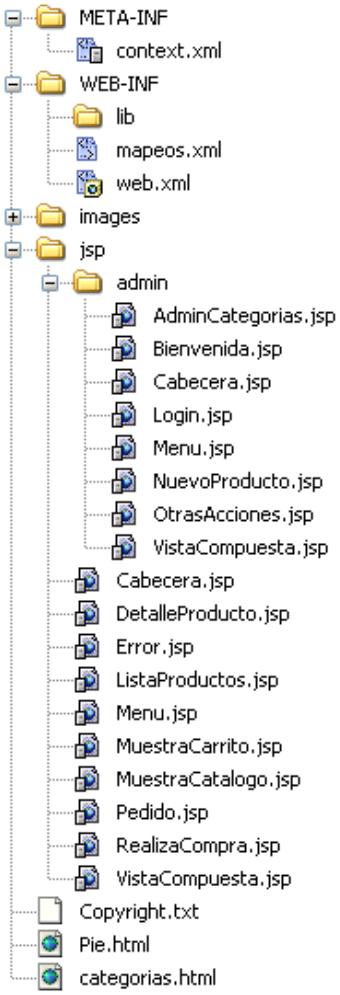

Figura 3-1 Árbol de directorios de la carpeta web

Dentro de la carpeta "***src***" tenemos el codigo fuente de las clases Java de la aplicación. Las clases se estructuran por paquetes, que se materializan en las siguientes carpetas:

Carpeta	Descripción
controlador	Contiene las clases Java del bloque Controlador de la aplicación (Controlador, Filtro y Gestor de Flujo) y clases auxiliares (Mapeo y MapeosHandler). También contiene una carpeta `acciones`, con las clases Java Acciones y una carpeta `admin` con el controlador de la Administración de la Tienda. La carpeta `admin` contiene el servlet controlador de la aplicación de Administración de la Tienda y clases Java con las acciones específicas de administración.
excepciones	Contiene las clases Java de excepciones de la aplicación. Estas excepciones (en esta versión solo una) son lanzadas cuando ocurre un error en la aplicación.
modelo	Contiene las clases Java del bloque Modelo de la aplicación (ModeloCarrito, ModeloTienda, ModeloUsuarios). También contiene otras clases auxiliares para contener datos (ElementoCarrito. Producto y Usuarios).
utilidades	Contiene las clases para el envío de ficheros al servidor, para hacer el cache a fichero de la tabla Categorías y otra para acceder a la fecha del servidor.
taglibrary	Contiene la librería de tags de la Tienda y una carpeta con los Handlers de los tags. En esta versión se incluyen tres handlers (iterator, enumerator e include).

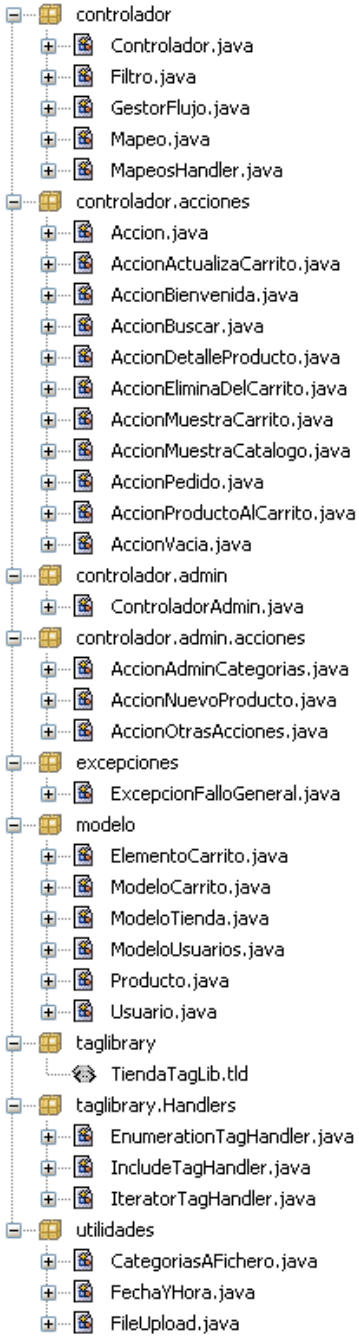

Figura 3-2 Paquetes de la aplicación

3.3 EJEMPLOS DE VISTAS DE LA APLICACIÓN

Antes de comenzar nuestro desarrollo daremos un vistazo general al aspecto que tiene la aplicación, para familiarizarnos con ella de forma más visual.

3.3.1 La Tienda Virtual

Cuando tecleamos la dirección URL de nuestra Tienda Virtual, la aplicación nos devuelve una página con los productos destacados de la tienda, es decir, el escaparate. De esta forma estamos atrayendo a los clientes, igual que un escaparate de una tienda de una calle intenta atraer clientes colocando productos en su escaparate.

En esta primera página, ya podemos observar las partes comunes de que consta la interfaz gráfica de nuestra Tienda Virtual.

- Vemos una cabecera con un banner y una barra con la fecha actual y un enlace para acceder a la bolsa de la compra.

- A la izquierda vemos una herramienta de búsqueda y unas opciones de navegación por categorías. El usuario de forma intuitiva, comprende que puede buscar productos en la tienda o buscar productos "paseando por los pasillos", es decir, navegando por las distintas secciones o categorías de productos.

- A la derecha vemos una lista de productos, en la que aparece información básica del producto, el fabricante, una imagen del producto, su precio y una opción de llevar el producto a la cesta de la compra.

- Por último, en la parte inferior, podemos ver un pie de página.

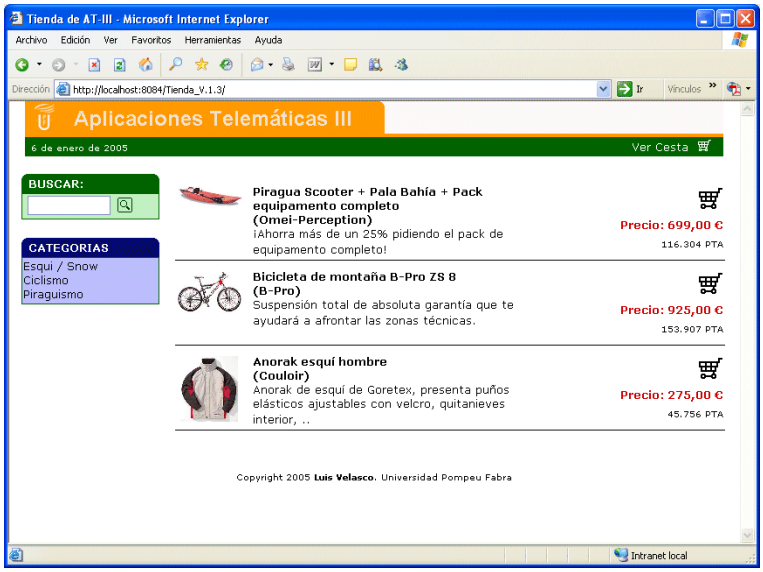

Figura 3-3 Escaparate

Si utilizamos la opción de navegar por categorías, nos aparece una página como la siguiente, con una lista de productos idéntica a la anterior, pero con un rótulo de la categoría de productos que se ha seleccionado.

Podemos observar en este momento que la página anterior de escaparate y la de navegación por categorías comparte, prácticamente, las mismas características y será posible reutilizar una parte importante de la página desarrollada.

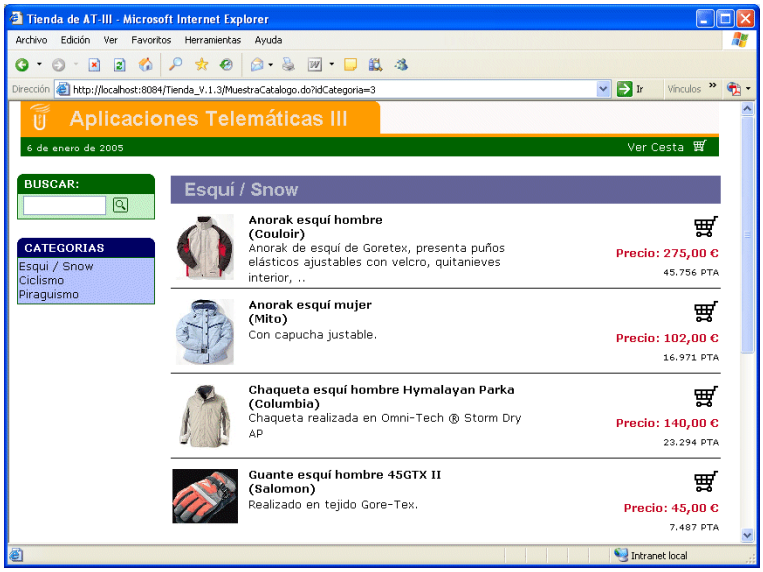

Figura 3-4 Productos de una categoría

Si ahora seleccionamos un producto, podemos ver una página específica donde se muestra el detalle del producto. Esta página contiene una descripción más completa que la que aparecía anteriormente y podría contener otra información adicional.

Figura 3-5 Detalle de un producto

Una vez hemos hallado un producto que deseamos adquirir, tenemos la posibilidad de llevar el producto a la cesta de la compra. Además, es posible que en cualquier momento deseemos ver el contenido de la cesta de la compra. En ambas situaciones, nos aparecerá una página como la siguiente, en la que se listan los artículos de la cesta y podemos modificar la cantidad de cada uno de ellos, así como eliminar alguno o todos los artículos de nuestra cesta de la compra.

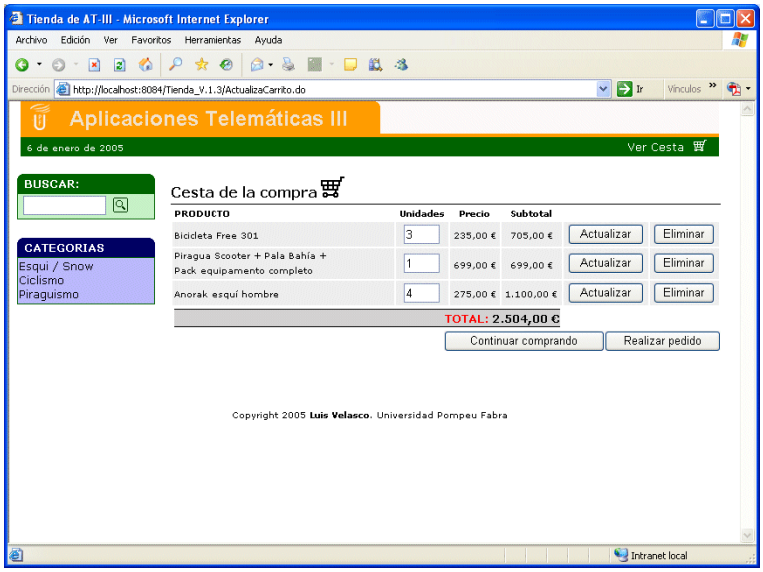

Figura 3-6 Cesta de la compra

Cuando ya tenemos en nuestra cesta de la compra todos los artículos que deseamos comprar, podemos utilizar la opción de realizar el pedido. Esta opción nos muestra una página que contiene un formulario de pedido para que el cliente proporcione información acerca del envío de los productos y de la forma de pago.

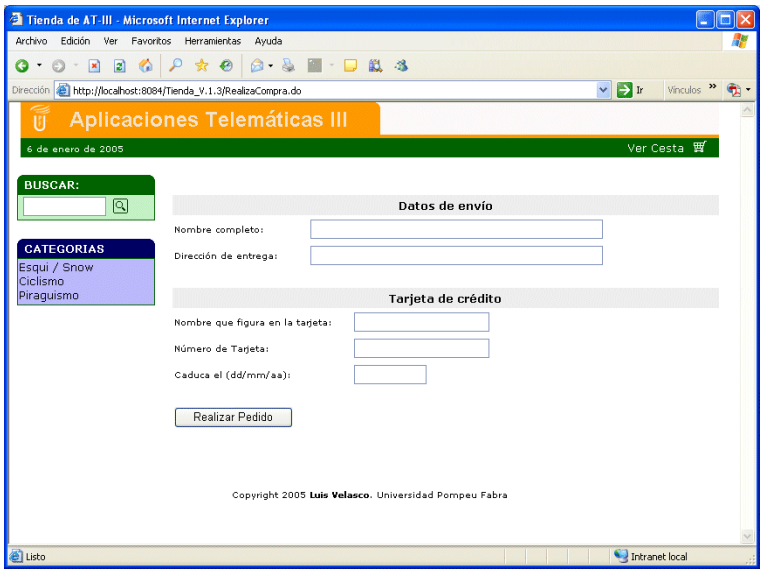

Figura 3-7 Pedido

Si rellenamos correctamente los datos del formulario de pedido, la aplicación nos devolverá un mensaje con información sobre la fecha esperada de recepción de la mercancía. Si falta algún dato, aparecerá un mensaje de error.

3.3.2 La Administración de la Tienda

Cuando tecleamos la dirección URL de nuestra Tienda Virtual, pero incluimos la subcadena "/admin/" en esta URL, la aplicación nos devuelve una página de login para acceder a la aplicación de Administración de la Tienda.

La siguiente imagen muestra la página de login de la aplicación de Administración.

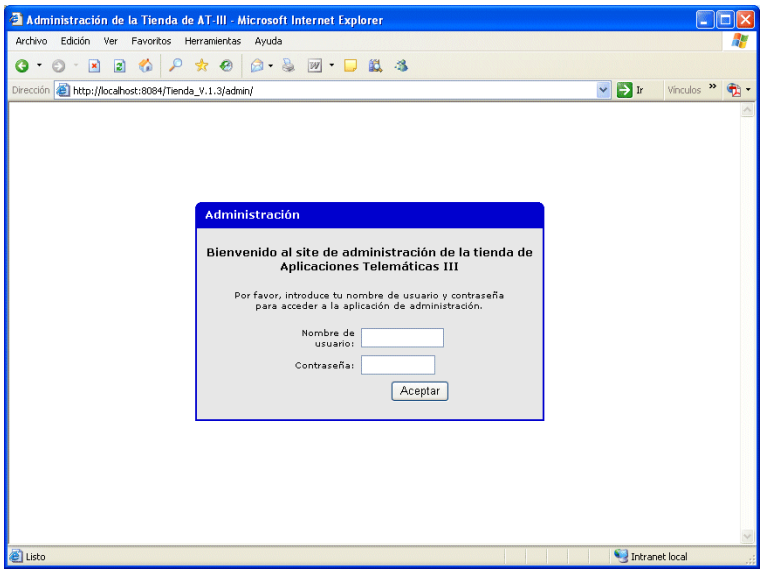

Figura 3-8 Acceso a la aplicación de administración

En esta página debemos introducir un nombre de usuario y una contraseña de un usuario registrado con permiso de administración de la Tienda Virtual. Cuando proporcionamos un usuario que cumple las condiciones anteriores, la aplicación nos deja acceder a la Administración devolviéndonos la siguiente página de bienvenida.

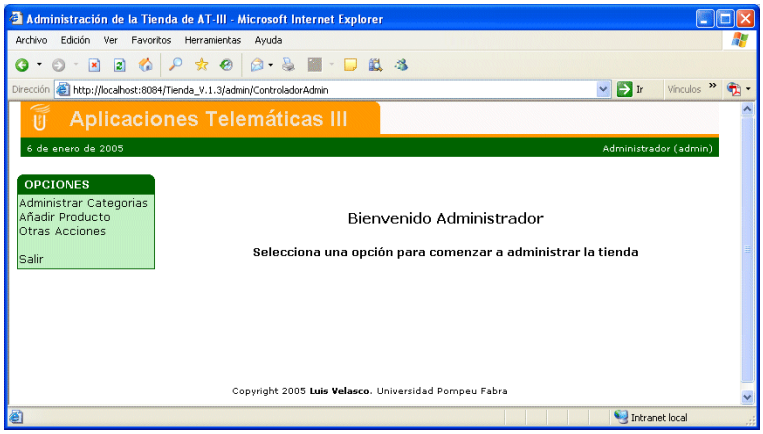

Figura 3-9 Bienvenida a la aplicación de administración

- **Administrar Categorías**: Esta opción permite añadir, modificar y eliminar categorías de la Tienda Virtual. Como veremos en siguientes capítulos, la aplicación hace cache de la tabla de categorías en un fichero de texto, que es incluido por la página de menú. De esta forma se evita el acceso a la base de datos para solicitar información que cambia de forma muy infrecuente. Mediante esta opción de administración, se modifica la tabla de categorías y se vuelca a fichero el nuevo estado de esta tabla de la base de datos.

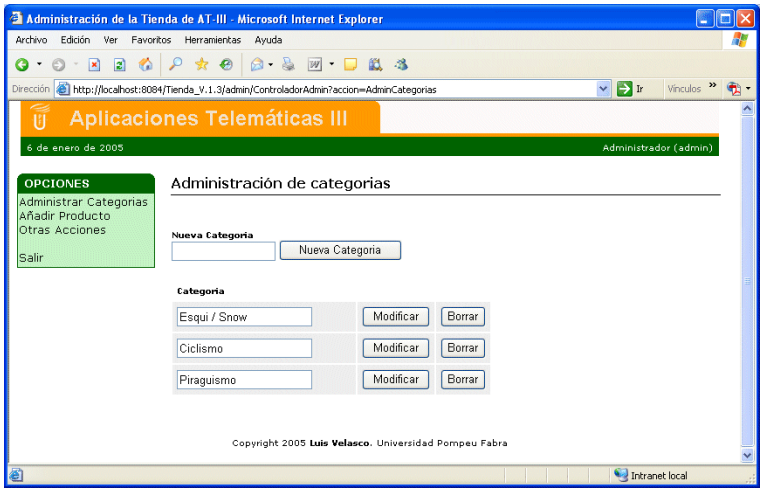

Figura 3-10 Administración de categorías

- **Añadir Producto**: Esta opción muestra un formulario que permite añadir un nuevo producto en la tabla de Productos de la base de datos.

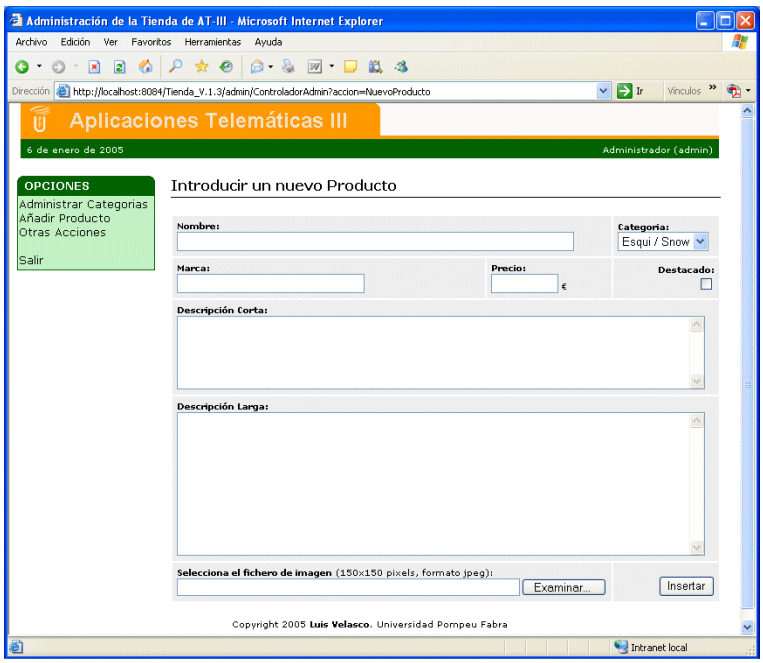

Figura 3-11 Introducir un producto

Puesto que la acción de introducir un nuevo producto lleva consigo enviar una nueva imagen del producto al servidor, este formulario incorpora la posibilidad de enviar fieros al servidor. Los ficheros recibidos serántratados como imágenes por el servidor y almacenados en la carpeta correspondiente.

- **Otras Acciones**: Puesto que la administración de la Tienda lleva consigo muchas acciones, se ha optado por incluir una herramienta que permite hacer consultas sobre la base de datos de Productos. Esta herramienta pasa de forma transparente las consultas realizadas (admite cualquier consulta sql, como: *select*, *insert*, *update*, *delete*, *create table*, etc.) y muestra el resultado de

la consulta, bien sea mediante una tabla con los resultados de una consulta `select` o bien un mensaje indicando el número de registros afectados por la consulta.

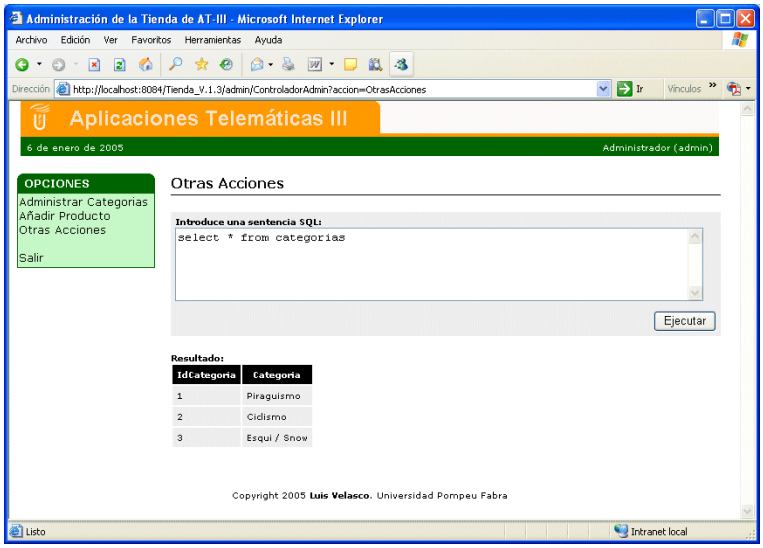

Figura 3-12 Otras acciones de administración

CAPÍTULO 4

Diseño de las bases de datos

4.1 INTRODUCCIÓN

Este capítulo contiene el diseño de las bases de datos del sistema y suele estar realizado por un equipo específico de expertos en Bases de Datos.

En una aplicación con una estructura de bases de datos tan pequeña como la nuestra, esta labor no parece necesaria, sin embargo, en aplicaciones un poco más complejas y utilizando gestores de bases de datos más potentes, esta labor resulta imprescindible.

La aplicación accede a dos bases de datos diferentes. La base de datos Tienda contiene el catálogo de productos y los pedidos de los clientes. La segunda base de datos, Usuarios, contiene la información relativa a los usuarios de la aplicación.

Se ha optado por utilizar la base de datos MySQL ya que está disponible de forma gratuita a través de Internet, aunque es posible utilizar cualquier otra base de datos relacional más escalable y potente. Los ficheros con los scripts sql para generar las bases de datos de la Tienda Virtual y para poblar las tablas, se encuentran en la carpeta "**db**" bajo la carpeta.

Comenzaremos diseñando las bases de datos que componen la Tienda Virtual, para después definir las consultas que serán utilizadas. Los scripts sql se comentarán en el capítulo dedicado al despliegue de la aplicación.

4.2 DISEÑO DE LAS BASES DE DATOS

4.2.1 Base de datos Tienda

La base de datos Tienda está formada por cuatro tablas: Categorías, Productos, Pedidos y DetallePedidos. Las relaciones entre las tablas se muestra en la siguiente figura.

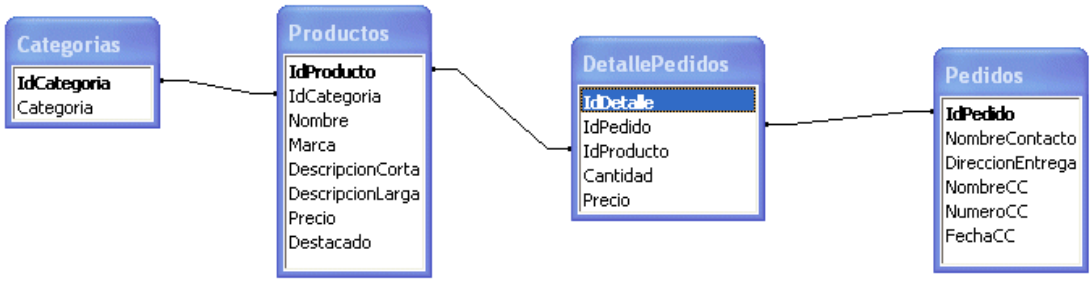

Figura 4-1 Tablas y relaciones entre tablas de Tienda

4.2.1.1 La tabla Categorias

La tabla de categorías se utiliza para almacenar las categorías o departamentos en que se organizar los productos de la tienda.

Nombre del campo	Tipo de Datos	Descripción
IdCategoria	Autonumérico	Clave Primaria
Categoria	Texto	Nombre de la categoría

Tabla 4-1. Tabla Categorias

4.2.1.2 La tabla Productos

La tabla Productos almacena los productos que están disponibles en la tienda. Cada producto contiene la descripción y precio, así como la categoría a la que pertenece un producto. Además, es posible seleccionar que el producto aparezca en el escaparate de la tienda, mediante el campo destacado.

Nombre del campo	Tipo de Datos	Descripción
IdProducto	Autonumérico	Clave Primaria
IdCategoria	Número	Identificador de categoría
Nombre	Texto	Nombre del producto
Marca	Texto	Marca
DescripcionCorta	Texto	Descripción corta de características
DescripcionLarga	Texto	Descripción larga de características
Precio	Doble	Precio del producto
Destacado	Booleano	Si *true*, el producto aparece en el escaparate.

Tabla 4-2. Tabla Productos

4.2.1.3 La tabla Pedidos

La tabla de pedidos almacena la información a cerca del pedido, como los datos de entrega del pedido y los datos de la tarjeta de crédito.

Nombre del campo	Tipo de Datos	Descripción
IdPedido	Autonumérico	Clave Primaria
NombreContacto	Texto	Persona de contacto para la entrega del pedido
DireccionEntrega	Texto	Dirección de entrega del pedido
NombreCC	Texto	Nombre que figura en la tarjeta de crédito
NumeroCC	Texto	Número de tarjeta de crédito
FechaCC	Texto	Fecha de caducidad de la tarjeta de crédito

Tabla 4-3. Tabla Pedidos

4.2.1.4 La tabla DetallePedidos

Esta tabla se utiliza para almacenar los elemetos adquiridos en cada pedido.

Nombre del	Tipo de Datos	Descripción

campo		
IdDetalle	Autonumérico	Clave Primaria
IdPedido	Número	Identificador del pedido relacionado
IdProducto	Número	Identificador del producto adquirido
Cantidad	Número	Cantidad de unidades
Precio	Doble	Precio al que se adquiere el producto

Tabla 4-4. Tabla DetallePedidos

4.2.2 Base de datos de Usuarios

La base de datos Usuarios se utiliza para gestionar el acceso a la aplicación de administración de la tienda. Está formada por dos tablas: Usuarios y Permisos.

El acceso a la aplicación de administración se realiza mediante un sistema de permisos para los usuarios registrados. Se definen tres niveles de permisos diferentes, aunque de momento, solamente se utilizará el permiso de Administración.

La siguiente figura muestra las relaciones entre las tablas de la base de datos Usuarios

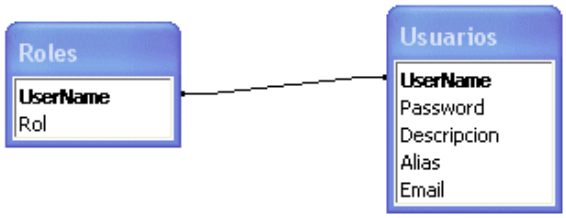

Figura 4-2 Tablas y relaciones entre tablas de Usuarios

4.2.2.1 La tabla Usuarios

Almacena información de los usuarios. Contiene el nombre de usuario y contraseña y otros datos administrativos.

Nombre del campo	Tipo de Datos	Descripción
UserName	Texto	Nombre del usuario Clave Primaria
Password	Texto	Contraseña
Descripcion	Texto	Función que desempeña
Alias	Texto	Alias del usuario
Email	Texto	Dirección de e-mail

Tabla 4-5. Tabla Usuarios

4.2.2.2 La tabla Roles

Almacena los roles asociados a los usuarios en la aplicación. Existen diferentes roles, adminTienda, etc.

Nombre del campo	Tipo de Datos	Descripción
UserName	Texto	Clave Primaria
Rol	Texto	Nombre del rol Clave Primaria

Tabla 4-6 .Tabla Roles

4.3 LAS CONSULTAS SQL

Una vez hemos definido la estructura de las bases de datos a utilizar, con sus tablas, los campos de las tablas y las relaciones entre las tablas, debemos definir las consultas a realizar sobre estas bases de datos para acceder a la los datos que se desean, para introducir el ellas datos nuevos, y para actualizar o eliminar datos existentes.

4.3.1 Lista categorías

La siguiente sentencia SQL devuelve el par IdCategoria, Categoria de todas las categorías de la tienda.

```
SELECT IdCategoria, Categoria FROM Categorias ORDER BY Categoria ASC
```

4.3.2 Id de categoría

La siguiente sentencia SQL devuelve el IdCategoria de una categoría dada.

```
SELECT IdCategoria FROM Categorias WHERE Categoria=?
```

4.3.3 Inserta categoría

La siguiente sentencia SQL inserta una nueva categoría, dado su nombre de categoría.

```
INSERT INTO Categorias (Categoria) VALUES (?)
```

4.3.4 Elimina categoría

La siguiente sentencia SQL elimina una categoría dado si id.

```
DELETE FROM Categorias WHERE IdCategoria=?
```

4.3.5 Actualiza categoría

La siguiente sentencia SQL actualiza el nombre de una categoría dado si id.

```
UPDATE Categorias SET Categoria=? WHERE IdCategoria=?
```

4.3.6 Busca productos

La siguiente sentencia SQL busca productos que contengan una clave dada en cualquiera de los campos Nombre, Marca, DescripcionCorta y DescripcionLarga.

```
SELECT IdProducto, Nombre, Marca, DescripcionCorta, Precio FROM Productos
      WHERE Nombre LIKE '%?%'
      OR Marca LIKE '%?%'
      OR DescripcionCorta LIKE '%?%'
      OR DescripcionLarga LIKE '%?%'
```

4.3.7 Productos en categoría

La siguiente sentencia SQL devuelve los productos que pertenezcan a una categoría identificada por su id.

```
SELECT IdProducto, Nombre, Marca, DescripcionCorta, Precio FROM Productos
WHERE IdCategoria=?
```

4.3.8 Productos destacados

La siguiente sentencia SQL devuelve los productos que tengan marcado a trae el campo destacado.

```
SELECT IdProducto, Nombre, Marca, DescripcionCorta, Precio FROM Productos
WHERE Destacado=True
```

4.3.9 Id Producto

La siguiente sentencia SQL devuelve el id del producto cuyo nombre se le proporciona.

```
SELECT IdProducto FROM Productos WHERE Nombre=?
```

4.3.10 Inserta producto

La siguiente sentencia SQL devuelve el id del producto cuyo nombre se le proporciona.

```
INSERT INTO Productos (Nombre, Marca, DescripcionCorta, DescripcionLarga,
Precio, IdCategoria, Destacado) VALUES (?,?,?,?,?,?,?)
```

4.3.11 Detalles de producto

La siguiente sentencia SQL devuelve los detalles de un producto, dado su id.

```
SELECT IdProducto, Nombre, Marca, DescripcionCorta, DescripcionLarga,
Precio FROM Productos WHERE IdProducto=?
```

4.3.12 Inserta Pedido (transacción)

Esta transacción primero inserta un nuevo pedido en la tabla Pedidos y después inserta el detalle del pedido, un insert por cada producto en el pedido, en la tabla DetallePedido.

```
INSERT INTO Pedidos (IdPedido, NombreContacto, DireccionEntrega, NombreCC,
NumeroCC, FechaCC) VALUES (?,?,?,?,?,?)


INSERT INTO DetallePedidos (IdPedido, IdProducto, Cantidad, Precio) VALUES
(?,?,?,?)
```

4.3.13 Usuario en el sistema

La siguiente sentencia SQL devuelve los detalles de un usuario, dado su nombre de usuario y contraseña.

```
SELECT IdUsuario, UserName, Password, Descripcion, Alias, Email FROM
Usuarios WHERE UserName=? AND Password=?
```

4.3.14 Usuario en Rol

La siguiente sentencia SQL devuelve un registro si el usuario tiene asignado el rol proporcionado.

```
SELECT * FROM Roles WHERE UserName=? AND Rol=?
```

CAPÍTULO 5

Desarrollando el Modelo

5.1 INTRODUCCIÓN

Una vez hemos definido la estructura de las bases de datos y las consultas necesarias que vamos a utilizar en la aplicación, comenzaremos con la codificación de las clases Java que permiten el acceso a los datos.

En este apartado codificaremos clases pertenecientes a los tres partes del modelo, que utiliza la aplicación: el modelo de la tienda, el modelo de carrito de la compra y el modelo de usuarios.

5.2 EL MODELO DE LA TIENDA

El modelo de la tienda contiene las clases necesarias para acceder a la base de datos Tienda, de una forma consistente.

5.2.1 La clase Producto

La clase auxiliar `Producto`, almacena de forma temporal los datos de un producto, que provienen de la base de datos y se van a utilizar para la presentación, o bien provienen de un formulario y se van a almacenar en la base de datos.

Esta clase contiene los mismos campos que tiene la tabla Productos de la base de datos Tienda, como se aprecia en el siguiente listado.

Listado 5-1 La clase auxiliar Producto

```
public class Producto {
    public int id;
    public int idCategoria;
    public String nombre;
    public String marca;
    public String descripcionCorta;
    public String descripcionLarga;
    public double precio;
    public boolean destacado;
}
```

5.2.2 DataSources

Un *DataSource* no es más que un pool de conexiones para conexiones JDBC con una base de datos.

El *datasource* establece un conjunto de conexiones, o pool, antes de que se necesiten. Cuando un cliente requiere una conexión a la base de datos, se la pide al pool, el cual selecciona una conexión libre del conjunto, la marca como ocupada y se la devuelve al cliente.

Cuando no se necesite más la conexión, el cliente se la devuelve al pool, que la marca como disponible. Para devolver una conexión al pool, basta con cerrarla de forma explícita.

Además de distribuir conexiones, el *datasource* es responsable del mantenimiento de las mismas, comprobando que funcionan adecuadamente, creando nuevas conexiones cuando se acaben las disponibles, etc.

La siguiente figura esquematiza es funcionamiento de un pool de conexiones.

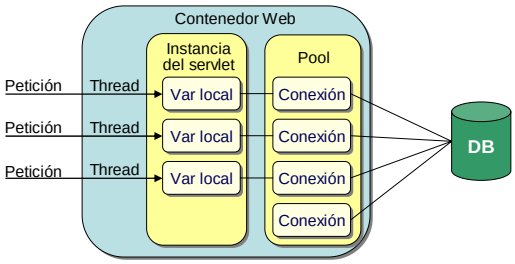

Figura 5-1 Pool de conexiones

Para utilizar un datasource en una aplicación web, primero debe configurarse y después se accede al recurso mediante JNDI. Los servidores de aplicaciones J2EE proporcionan una instancia de *InitialContext* JNDI para cada aplicación web. Para referenciar los recursos, se utilizan elementos `<env-entry>` en el *deployment descriptor* de la aplicación. Los recursos se definen en el fichero de contexto de la aplicación (META-INF/context.xml).

Los recursos del contexto inicial de la aplicación web, se situan en la porción del espacio de nombres JNDI "`java:comp/env`". El recurso, en este cado el *datasource*, se obtiene referenciandolo a través del contexto inicial de la aplicación, mediate el espacio de nombres y el nombre que le hemos asignado en el fichero de contexto, context.xml. Por convención, todos estos nombres se deben resolver en el subcontexto `jdbc`, relativo al contexto `java:comp/env`, que es la raíz de todas las factorías de recursos que se proporcionan.

Una vez obtenido el *datasorce*, se utiliza el método `getConnection`, para pedir una conexión libre del pool con la base de datos. Cuando la conexión ya no se necesita más, se debe devolver al pool cerrándola mediante el método `close` de la propia conexión.

En el siguiente apartado y, más adelante, en el capítulo decicado al despliegue de la aplicación, veremos más en detalle estos pasos.

5.2.3 La clase ModeloTienda

El uso de la factoria de recursos JNDI de fuentes de datos JDBC requiere que previamente se haya instalado el driver JDBC apropiado y se haya declarado el nombre JNDI, en el *deployment descriptor* de la aplicación, bajo el que se desea consultar la fuente de datos preconfigurada. En el caso de la Tienda Virtual, se ha denominado `jdbc/Tienda`. Veremos estos pasos en el capítulo referente al despliegue de la aplicación.

La clase ModeloTienda cuenta con el método `init`, que consulta el contexto para obtener una instacia del *datasource* con el que el resto de los métodos acceden a la base de datos de la Tienda. Antes de llamar a este método, se utiliza el método `setDbRefName`, para poner el nombre por el que consultar el *DataSource*.

Listado 5-2 El método init

```
  public void init() throws NamingException {
    Context initCtx = new InitialContext();
    Context envCtx = (Context) initCtx.lookup("java:comp/env");
```

```
    ds = (DataSource) envCtx.lookup(dbRefName);
  }
```

El resto de métodos de esta clase, hacen uso de las consultas SQL diseñadas en capítulos anteriores y se pueden clasificar en:

- gestión de categorías,
- búsquedas,
- gestión de productos,
- ejecución de sentencias de usuario.

Veremos a continuación ejemplos de algunos métodos de esta clase.

5.2.3.1 Actualización de categorías

El método `actualizaCategoria` actualiza la categoría identificada por su *id* con el valor de categoría nuevo. Podemos observar cómo los métodos de esta clase piden una conexión al pool del *DataSource*, la utilizan y la devuelven al pool cuando finalizan su trabajo o si ocurre un error.

Otro punto que podemos observar es que este método utiliza una sentencia preparada.

- La variable `pstmt` contiene la sentencia SQL, que ha sido enviada al controlador de la base de datos, y ha sido precompilada.
- A continuación, necesitamos suministrar los valores que se utilizarán en los lugares donde están las marcas de interrogación, mediante los métodos `setXXX`.
- Después de que los valores hayan sido asignados a sus dos parámetros, utilizamos el método `executeUpdate` para ejecutar la sentencia.

El siguiente listado muestra el código del método `actualizaCategoria`.

Listado 5-3 El método actualizaCategoria

```
  public int actualizaCategoria(String idCategoria, String categoria)
     throws ExcepcionFalloGeneral {
    if (ds==null)
        throw new ExcepcionFalloGeneral(
         "Imposible acceder a la Base de Datos. No se encuentra el DataSource");

    int returnValue = 0;
    Connection connection = null;
    try {
        connection = ds.getConnection();
        PreparedStatement pstmt = connection.prepareStatement(
                           "UPDATE Categorias SET Categoria=? WHERE
IdCategoria=?");
        pstmt.setString(1, categoria);
        pstmt.setString(2, idCategoria);
        returnValue = pstmt.executeUpdate();
        pstmt.close();
        connection.close();
```

```
    }
    catch (SQLException e1) {
        try {
            if (connection!=null) connection.close();
        } catch (SQLException e2) {}
    }
    return returnValue;
  }
```

5.2.3.2 Inserción de pedidos

El método `insertaPedido` inserta un nuevo pedido en la tabla *Pedidos* y la lista de productos asociados a este pedido, en la tabla *DetallePedidos*. La inserción se realiza como una única transacción, de forma que si existe cualquier error durante la inserción, se deshacen todas las operaciones realizadas. El siguiente listado muestra el código del método `insertaPedido`.

Listado 5-4 El método insertaPedido

```
  public boolean insertaPedido (String nombreContacto, String direccionEntrega,
    String nombreCC, String numeroCC, String fechaCC, Hashtable elementos)
    throws ExcepcionFalloGeneral {
    if (ds==null)
        throw new ExcepcionFalloGeneral(
         "Imposible acceder a la Base de Datos. No se encuentra el DataSource");

    boolean returnValue = false;
    if ((nombreContacto==null) || (direccionEntrega==null) || (nombreCC==null) ||
        (numeroCC==null) || (fechaCC==null) || (elementos==null))
        return returnValue;

    long idPedido = System.currentTimeMillis();
    Connection connection = null;
    try {
      connection = ds.getConnection();
      connection.setAutoCommit(false);
      PreparedStatement pstmt1 = connection.prepareStatement(
        "INSERT INTO Pedidos (IdPedido, NombreContacto, DireccionEntrega," +
        " NombreCC, NumeroCC, FechaCC) VALUES (?,?,?,?,?,?)");

      pstmt1.setString(1, Long.toString(idPedido));
      pstmt1.setString(2, nombreContacto);
      pstmt1.setString(3, direccionEntrega);
      pstmt1.setString(4, nombreCC);
      pstmt1.setString(5, numeroCC);
      pstmt1.setString(6, fechaCC);
      pstmt1.executeUpdate();
      pstmt1.close();
      // ahora insertar el detalle del pedido

      PreparedStatement pstmt2 = connection.prepareStatement(
        "INSERT INTO DetallesPedido (IdPedido, IdProducto, Cantidad, Precio)" +
        " VALUES (?,?,?,?)");
      Enumeration enumera = elementos.elements();
      while (enumera.hasMoreElements()) {
        ElementoCarrito elem = (ElementoCarrito) enumera.nextElement();
        pstmt2.setString(1, Long.toString(idPedido));
        pstmt2.setString(2, Integer.toString(elem.id));
        pstmt2.setString(3, Integer.toString(elem.cantidad));
        pstmt2.setString(4, Double.toString(elem.precio));
        pstmt2.executeUpdate();
        pstmt2.clearParameters();
      }

      pstmt2.close();
      connection.commit();
      connection.close();
```

```
      returnValue = true;
    }
    catch (SQLException e1) {
      try {
        if (connection!=null){
            connection.rollback();
            connection.close();
        }
      }
      catch (SQLException e2) {}
    }
    return returnValue;
  }
```

5.2.3.3 Ejecución de consultas de usuario

Vamos a ver ahora un método que permite que el usuario realice consultas directamente sobre la base de datos. El resultado de una consulta se muestra en forma de tabla HTML o indicando el número de registros afectados, en función del tipo de consulta.

El siguiente listado muestra el código del método ejecutaSQL. Podemos ver que se ejecuta la consulta mediante el método `execute`, que devuelve *true* si el resultado es un `ResultSet` y *false* si el resultado es un contador. Los métodos `getResultSet` y `getUpdateCount` de `Statement` proporcionan acceso a los resultados del método `execute`.

Por otra parte, la interfaz `ResultSetMetaData` proporciona una vía para obtener la estructura del resultado de una consulta, cuando ésta devuelve un `ResultSet`.

Para chequear valores nulos se utiliza el método `getObject`, que devuelve *null* si el campo es *null*.

Listado 5-5 El método ejecutaSQL

```
  public String ejecutaSQL (String sql) throws ExcepcionFalloGeneral {
   if (ds==null)
       throw new ExcepcionFalloGeneral(
        "Imposible acceder a la Base de Datos. No se encuentra el DataSource");

      StringBuffer out = new StringBuffer();

      if (sql==null)
          return "";

      Connection connection = null;

      try {
           connection = ds.getConnection();
           Statement s = connection.createStatement();
           if (s.execute(sql)) {
               // Hay un ResultSet
               ResultSet rs = s.getResultSet();
               out.append("<table border=0 cellpadding=5 cellspacing=2>");
               ResultSetMetaData rsmd = rs.getMetaData();
               int numcols = rsmd.getColumnCount();

               // Titulos de las columnas
               out.append("<tr bgcolor=black>");
               for (int i = 1; i <= numcols; i++){
                   out.append("<td align=center>");
                   out.append("<font face=Verdana color=white size=1><b>");
                   out.append(rsmd.getColumnName(i));
```

```
                    out.append("</b></font></td>");
                }

                out.append("</tr>");
                while(rs.next()) {
                    out.append("<tr bgcolor=#eeeeee>");  // Nueva fila
                    for(int i = 1; i <= numcols; i++) {
                        out.append("<td valign=top align=left>");
                        out.append("<font face=Verdana size=1>");
                        Object obj = rs.getObject(i);
                        if (obj != null)
                            out.append(obj.toString());
                        else
                            out.append(" ");
                        out.append("</font></td>");
                    }
                    out.append("</tr>");
                }
                // Final de la tabla
                out.append("</table>");
                rs.close();
            }
            else {
                // Hay un contador
                out.append("<font face=Verdana size=1><b>");
                int i=s.getUpdateCount();
                if (i<0) i=0;
                out.append(i);
                out.append(" registro(s) afectado(s)</b></font>");
            }
            s.close();
            connection.close();
        }
        catch (SQLException e1) {
            try {
                if (connection!=null) connection.close();
                out.append("<font face=Verdana color=red size=2><b>");
                out.append("<b>Error al ejecutar la consulta</b><br>");
                out.append("</b></font>");
                System.out.println(e1.toString());
            } catch (SQLException e2) {
              }
         }
         return out.toString();
    }
```

5.3 EL MODELO DE LA CESTA DE LA COMPRA

5.3.1 La clase ElementoCarrito

La clase auxiliar `ElementoCarrito`, almacena los datos de un producto en el carrito de la compra.

Esta clase contiene la información de un producto que es necesario mantener en el carrito de la compra, incluyendo la cantidad de unidades deseadas por el cliente. El siguiente listado muestra la clase `ElementoCarrito`.

Listado 5-6 La clase ElementoCarrito

```
public class ElementoCarrito {
  public int id;
  public String nombre;
  public String descripcion;
  public double precio;
  public int cantidad;
}
```

5.3.2 La clase ModeloCarrito

La clase `ModeloCarrito` implementa los métodos para gestionar el carrito de la compra. El carito se almacena en una tabla hash, que se direcciona mediante el *id* del producto y cada elemento contiene una instancia de la clase auxiliar `ElementoCarrito`, conteniendo información del producto, el precio y la cantidad de unidades.

El siguiente listado muestra el código de la clase `ModeloCarrito`, que contiene tres métodos para introducir un nuevo producto en el carrito, actualizar la cantidad de uno de los elementos que ya estaban en el carrito y para eliminar un elemento del carrito.

Listado 5-7 La clase ModeloCarrito

```
public class ModeloCarrito {
  public Hashtable elementos=new Hashtable();//tabla hash que contiene el carrito

  public void nuevoProducto(Producto producto) {
      if (producto != null) {
        ElementoCarrito elem = new ElementoCarrito();
        elem.id = producto.id;
        elem.cantidad = 1;
        elem.precio = producto.precio;
        elem.nombre = producto.nombre;
        elem.descripcion = producto.descripcionCorta;
        elementos.remove(Integer.toString(producto.id));
        elementos.put(Integer.toString(producto.id), elem);
      }
  }

  public void actualizaProducto (int id, int cantidad) {
        ElementoCarrito elem = (ElementoCarrito)
                               elementos.get(Integer.toString(id));
        if (elem!=null)
            elem.cantidad = cantidad;
  }

  public void eliminaProducto (int id) {
        elementos.remove(Integer.toString(id));
  }
}
```

5.4 EL MODELO DE USUARIOS

5.4.1 La clase Usuario

La clase auxiliar `Usuario`, almacena de forma temporal los datos de un usuario, que provienen de la base de datos y se van a utilizar para la presentación.

Esta clase contiene los mismos campos que tiene la tabla `Usuario` de la base de datos `Usuarios`, como se aprecia en el siguiente listado.

Listado 5-8 La clase Usuario

```
public class Usuario {
  public String userName;
  public String password;
  public String descripcion;
  public String alias;
  public String email;
}
```

5.4.2 La clase ModeloUsuarios

La clase `ModeloUsuarios` utiliza un mecanismo más simple que la clase `ModeloTienda` para acceder a la base de datos de Usuarios. En este caso se ha considerado que no es necesario crear un pool de conexiones con la base de datos, ya que esta aplicación la utiliza únicamente el administrador. Por ello, se utiliza el DriverManager para coger una única conexión con la base de datos.

La clase `ModeloUsuarios` contiene los métodos para el acceso a la base de datos `Usuarios`. El constructor crea una conexión con la base de datos, que se utiliza por el resto de los métodos. Los métodos hacen uso de las consultas SQL diseñadas en capítulos anteriores, como se observa en el siguiente listado.

Listado 5-9 La clase ModeloUsuarios

```
public class ModeloUsuarios {
    private Connection connection=null; // conexión con la db de usuarios

  public ModeloUsuarios(String dbURL, String dbUsuario, String dbPassword)
  throws SQLException, ClassNotFoundException {

      connection = DriverManager.getConnection(dbURL, dbUsuario, dbPassword);
  }

  public boolean UsuarioEnRol (String userName, String rol) {

        if (connection==null)
            return false;

        boolean returnValue=false;

        try {
            PreparedStatement pstmt = connection.prepareStatement(
                "SELECT * FROM Roles WHERE UserName=? AND Rol=?");
            pstmt.setString(1, userName);
            pstmt.setString(2, rol);

            ResultSet rs = pstmt.executeQuery();
            if (rs.next())
              returnValue=true;

            rs.close();
            pstmt.close();
        }
        catch (SQLException e) {
            System.out.println(e.toString());
        }
        return returnValue;
   }

   public Usuario UsuarioAutorizado (String userName, String password) {
        if (connection==null)
            return null;

        Usuario usuario=null;

        try {
            PreparedStatement pstmt = connection.prepareStatement(
                "SELECT UserName, Password, Descripcion, Alias, Email FROM " +
                "Usuarios WHERE UserName=? AND Password=?");
            pstmt.setString(1, userName);
            pstmt.setString(2, password);
            ResultSet rs = pstmt.executeQuery();
            if (rs.next()) {
                usuario = new Usuario();
                usuario.userName=rs.getString(1);
```

```
                usuario.password=rs.getString(2);
                usuario.descripcion=rs.getString(3);
                usuario.alias=rs.getString(4);
                usuario.email=rs.getString(5);
            }
            rs.close();
            pstmt.close();
        }
        catch (SQLException e) {
            System.err.println(e.toString());
        }

        return usuario;
    }
}
```

CAPÍTULO 6

Desarrollando las Vistas

6.1 INTRODUCCIÓN

En este capítulo desarrollaremos el bloque de vistas de la aplicación de Tienda Virtual. Cuando hablamos de vistas, podemos diferenciar entre presentación puramente dicha y clases Java que contienen funcionalidad o datos.

En nuestra Tienda Virtual se utilizan custom tags y Javabeans como clases de ayuda. Comenzaremos desarrollando primero los handlers y la librería de de etiquetas, a continuación los JavaBeans, para después desarrollar las páginas jsp que utilizan los anteriores componentes.

6.2 CUSTOM TAGS

Las custom tags proporcionan un mecanismo para encapsular funcionalidad compleja y ser utilizada en páginas JSP. La funcionalidad de las etiquetas a medida se define mediante clases Java que implementan la interface Tag.

Los programadores Java pueden crear y encapsular funcionalidad compleja. Los diseñadores Web incorporan estas funcionalidades en las páginas, sin necesidad de conocimientos de programación Java.

Las custom tags se agrupan en librerías que después serán incluidas en páginas JSP utiliza la directiva taglib.

Para describir una librería de tags se utiliza un fichero descriptor de la librería (tld), que describe los tags que incluye la librería y otra información adicional, como la clase del manejador, atributos, variables, etc.

El siguiente listado muestra una parte del contenido del descriptor de la librería de tags de la Tienda Virtual. Concretamente muestra la información que aparece en el descriptor para las etiquetas `incluye` e `iterator`.

Listado 6-1 TLD de la Tienda Virtual

```
<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE taglib
        PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
        "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">

<taglib>
    <tlib-version>1.0</tlib-version>
    <jsp-version>1.2</jsp-version>
    <short-name>TiendaTagLib</short-name>
    <uri>/WEB-INF/classes/taglibrary/TiendaTagLib</uri>
    <display-name>TiendaTagLib</display-name>
    <description>Librería de Tags para Aplicaciones Telemáticas III
        Curso 2004-2005
        Universidad Pompeu Fabra</description>
    <tag>
        <name>include</name>
        <tag-class>taglibrary.Handlers.IncludeTagHandler</tag-class>
        <body-content>empty</body-content>
        <display-name></display-name>
        <description>Incluye la página cuyo nombre recibe en el parámetro
        page</description>
```

```
        <attribute>
            <name>page</name>
            <required>true</required>
            <rtexprvalue>true</rtexprvalue>
            <type>java.lang.String</type>
        </attribute>
        <attribute>
            <name>errorpage</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
            <type>java.lang.String</type>
        </attribute>
        <example></example>
    </tag>
    <tag>
        <name>iterator</name>
        <tag-class>taglibrary.Handlers.IteratorTagHandler</tag-class>
        <body-content>JSP</body-content>
        <display-name></display-name>
        <description></description>
        <variable>
            <name-given>nombre</name-given>
            <variable-class>java.lang.String</variable-class>
            <declare>true</declare>
            <scope>NESTED</scope>
        </variable>
        <variable>
            <name-given>descripcionCorta</name-given>
            <variable-class>java.lang.String</variable-class>
            <declare>true</declare>
            <scope>NESTED</scope>
        </variable>
        <variable>
            <name-given>precio</name-given>
            <variable-class>java.lang.String</variable-class>
            <declare>true</declare>
            <scope>NESTED</scope>
        </variable>
        <variable>
            <name-given>id</name-given>
            <variable-class>java.lang.String</variable-class>
            <declare>true</declare>
            <scope>NESTED</scope>
        </variable>
        <variable>
            <name-given>precioPta</name-given>
            <variable-class>java.lang.String</variable-class>
            <declare>true</declare>
            <scope>NESTED</scope>
        </variable>
        <variable>
            <name-given>numProductos</name-given>
            <variable-class>java.lang.String</variable-class>
            <declare>true</declare>
            <scope>AT_END</scope>
        </variable>
        <variable>
            <name-given>marca</name-given>
            <variable-class>java.lang.String</variable-class>
            <declare>true</declare>
            <scope>NESTED</scope>
        </variable>
        <example></example>
    </tag>
</taglib>
```

Este fichero descriptor es normalmente generado de forma automática cuando se utiliza un entorno integrado de desarrollo, por ejemplo NetBeans.

Una vez se han descrito las etiquetas de la librería, se deben desarrollar las clases Java que manejan cada etiqueta, denominados Handlers. Estas clases serán llamadas cuando se encuentre la etiqueta dentro de la página jsp.

Los handlers deben tener implementar la interfaz `Tag` extendiendo la clase `TagSupport` o `BodyTagSupport`, según si se evalúa o no el cuerpo de la etiqueta.

6.2.1 El manejador de la etiqueta include

La etiqueta `include`, no procesa el cuerpo de la etiqueta y solo tiene implementa el método `doStartTag` y dos métodos setter para los atributos `page` y `errorpage`. El siguiente listado muestra el código del manejador de la etiqueta `include`.

Listado 6-2 include Tag Handler

```
public class IncludeTagHandler extends TagSupport {
   private String page="";
   private String errorpage="";

   public int doStartTag() throws JspException {
        ServletRequest request = pageContext.getRequest();
        ServletResponse response = pageContext.getResponse();
        try {
            request.getRequestDispatcher(page).include(request, response);
        }
        catch(Throwable t) {
          try {
              request.getRequestDispatcher(errorpage).include(request, response);
          }
            catch(Throwable e) {}
        }
        return SKIP_BODY;
   }

   public void setPage(String page) {
        this.page=page;
   }

   public void setErrorpage(String errorpage) {
        this.errorpage=errorpage;
   }
}
```

6.2.2 El manejador de la etiqueta iterator

La etiqueta `iterator`, realiza el procesamiento del cuerpo de la etiqueta e implementa los métodos `doStartTag`, `doAfterBody` y `doEndTag`. El siguiente listado muestra el código del manejador de la etiqueta `iterator`.

Listado 6-3 iterator Tag Handler

```
public class EnumerationTagHandler extends BodyTagSupport {
   private Enumeration enum;
   private double total = 0;
   private DecimalFormat EUR;
   private int numProductos = 0;

   public int doStartTag() throws JspException
   {
      EUR = new DecimalFormat("#,###.00");
      ServletRequest request = pageContext.getRequest();
      ServletResponse response = pageContext.getResponse();
```

```
      try {
            ModeloCarrito carrito =
                           (ModeloCarrito)request.getAttribute("carrito");
            if (carrito==null)
                return SKIP_BODY;
            enum = carrito.elementos.elements();

            if ( enum.hasMoreElements() ) {
               processNextProduct();
               return EVAL_BODY_BUFFERED; //continua procesando el cuerpo del tag
            }
            else
               return SKIP_BODY; // termina de procesar el cuerpo del tag
      }
      catch(Exception e) {
         return SKIP_BODY;   // ignora el cuerpo del tag
      }
   }

   public int doAfterBody()
   {
      // intenta escribir los datos de salida
      try {
         bodyContent.writeOut( getPreviousOut() );
      }
      catch (IOException e) {
         return SKIP_BODY; //  termina de procesar el cuerpo del tag
      }
      bodyContent.clearBody();

      if (enum.hasMoreElements()) {
         processNextProduct();

         return EVAL_BODY_BUFFERED; // ontinua procesando el cuerpo del tag
      }
      else
         return SKIP_BODY;        // termina de procesar el cuerpo del tag
   }

   private void processNextProduct()
   {
      // coge el siguiente producto
        ElementoCarrito elem = (ElementoCarrito) enum.nextElement();

        pageContext.setAttribute("nombre", elem.nombre);
        pageContext.setAttribute("precio", EUR.format(elem.precio));
        pageContext.setAttribute("cantidad", Integer.toString(elem.cantidad));
        pageContext.setAttribute("id", Integer.toString(elem.id));
        pageContext.setAttribute("subtotal",
                                 EUR.format(elem.precio*elem.cantidad));

        total += elem.precio*elem.cantidad;
        numProductos++;
   }

   public int doEndTag() throws JspException {
        pageContext.setAttribute("total", EUR.format(total));
        pageContext.setAttribute("numProductos", Integer.toString(numProductos));
        numProductos = 0;
        total = 0;
        return EVAL_PAGE;
    }
}
```

6.3 JAVABEANS

Otra forma de encapsular funcionalidad compleja y ser utilizada en páginas JSP, la proporcionan los JavaBeans. Un *bean* no es más que una clase Java. No es necesario extender ninguna clase base ni implementar ninguna interfaz, pero deben cumplirse ciertas reglas que detalla la especificación de *JavaBeans*.

En relación a los *JavaBeans* que pueden utilizarse desde páginas JSP, las reglas son:

- La clase *bean* debe tener un **constructor sin argumentos** o no tener constructor.
- Opcionalmente, un *bean* puede tener un método público que puede ser utilizado para poner el valor de una propiedad. Este método se denomina *setter*. El método no devuelve ningún valor y su nombre comienza con "set" seguido del nombre de la propiedad:

```
public void setPropertyName (PropertyType value);
```

- Opcionalmente, un bean puede tener un método público que pueda ser llamado para obtener el valor de la propiedad. Este método se denomina *getter*, y devuelve una variable del mismo tipo que el tipo de la propiedad. Su nombre comienza con "get" seguido del nombre de la propiedad:

```
public PropertyType getPropertyName();
```

- Los métodos *setter* y *getter* se denominan **métodos de acceso**. En JSP, las acciones jsp:getProperty y jsp:setProperty se utilizan para invocar al método *getter* y *setter* respectivamente, aunque es posible llamar a estos métodos de la misma forma que se llama a un método ordinario.

Nuestra Tienda Virtual utiliza una clase Java que es invocada como un JavaBen desde las páginas JSP, para acceder a la fecha y hora del servidor, en el momento de servir la página. En el siguiente listado vemos el código de la clase FechaYHora.

Listado 6-4 La clase FechaYHora

```
public class FechaYHora {

    public String Fecha (HttpServletRequest request) {
        Locale locale = request.getLocale();
        DateFormat dateFormat =
            DateFormat.getDateInstance(DateFormat.LONG, locale);
        return dateFormat.format(new java.util.Date());
    }
       ...
}
```

6.4 PÁGINAS JSP

6.4.1 La vista compuesta

La vista compuesta se encarga de ensamblar la página html que será enviada al usuario. Contiene la definición del aspecto que tendrá la aplicación completa.

Normalmente, se hace uso de hojas de cascada de estilos para especificar el aspecto de cada tipo de párrafo. En nuestra vista compuesta utilizaremos css únicamente para que los enlaces no aparezcan subrayados, que es el comportamiento por defecto de la mayoría de los navegadores.

La vista compuesta hace uso del custom tag incluye, desarrollado en el apartado anterior, para incluir una página de forma dinámica, cuyo nombre nos viene dado mediante un atributo de la petición. Esta página es el denominado cuerpo, que realmente es el contenido principal de cada una de las páginas.

Las páginas de cabecera y de menú son incluidas mediante la acción jsp incluye, de forma que son procesadas con cada petición.

Por el contrario, el fichero de pie de página se incluye mediante la directiva *include*, que inserta los ficheros en el momento en que la página es traducida. Si el pie de página cambia, necesitamos re-traducir de nuevo la vista compuesta para que se reflejen los cambios realizados.

Por último, la mayoría de las páginas hacen uso de la variable baseUrl. Esta variable contiene el path del contexto de la aplicación y es necesario que figure en todos los enlaces, en el path de las imágenes, etc.

El siguiente listado muestra el código de la vista compuesta de la Tienda Virtual. La aplicación de administración tiene su propia vista compuesta, de forma que el aspecto de cada aplicación pueda ser totalmente diferente. De cualquier forma, el contenido de ambas vistas compuestas es semejante.

Listado 6-5 La Vista Compuesta

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@page buffer="none"%>
<%@taglib uri="/WEB-INF/classes/taglibrary/TiendaTagLib.tld" prefix="t" %>

<html>
<head>
<title>Tienda de AT-III</title>

<style TYPE="text/css">
    a:link,a:visited,a:active {color:black;text-decoration:none}
</style>
</head>

<body bgcolor="#ffffff" link="#ffffff" vlink="#ffffff" alink="#ffffff"
leftmargin="0" topmargin="0" marginwidth="0" marginheight="0">
<%
    String baseUrl = (String)
                     request.getAttribute("javax.servlet.forward.context_path");
    String cuerpo = (String)request.getAttribute("cuerpo");
%>
```

```
<center>
<%@include file="Cabecera.jsp"%>
<table border="0" cellpadding="0" cellspacing="0" width="800">
<tr align="left">
 <td valign="top">
  <img src="<%=(baseUrl + "/images/pt.gif")%>" width="160" height="10"
    border="0"><jsp:include page="Menu.jsp"/></td>
 <td valign="top">
  <table border="0" >
  <tr>
  <td><img src="<%=(baseUrl + "/images/pt.gif")%>" width="10" height="10"
        border="0"></td>
  <td><img src="<%=(baseUrl + "/images/pt.gif")%>" width="630" height="10"
        border="0"><t:include page="<%=cuerpo%>"/></td>
  </tr>
  </table>
 </td>
</tr>
</table>
<br>
<%@include file="Pie.html"%>
</center>
</body>
</html>
```

6.4.2 La cabecera

La página de cabecera muestra una imagen con el nombre de la tienda y una barra con la fecha y la opción de ver la cesta de la compra, en el caso de la Tienda Virtual, y el alias y el nombre del usuario, en el caso de la aplicación de administración.

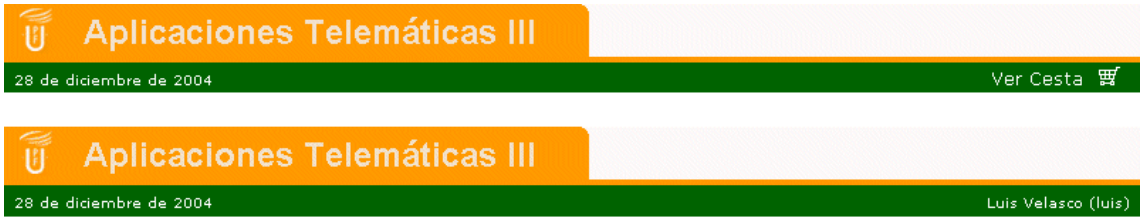

Figura 6-1 Cabeceras de la tienda y de administración

La funcionalidad de la fecha del servidor está implementada mediante la clase FechaYHora, que se invoca mediante la acción `jsp:useBean` desde la página de cabecera, y después se utilizan sus métodos para acceder a la fecha y hora del sistema.

En el siguiente listado podemos ver el código de la página de cabecera de la tienda.

Listado 6-6 Página de cabecera de la tienda

```
<jsp:useBean id="fecha" class="utilidades.FechaYHora" scope="page"/>

<table width="800" border="0" cellspacing="0" cellpadding="0">
<tr bgcolor="#ff9900">
<td colspan=2 align="left" valign="top">
  <img src="<%=(baseUrl + "/images/AT-III.gif")%>" width="800" height="37">
</td>
</tr>
<tr bgcolor="#006600">
<td height="22" align="left">
  <img src="<%=(baseUrl + "/images/pt.gif")%>" width="3" height="13" border="0">
  <font face="Verdana" color="white" size=1><%=fecha.Fecha(request)%></font>
</td>
```

```
<td height="22" align="right"><a href="<%=(baseUrl + "/MuestraCarrito.do")%>">
  <font color="white" face="Verdana" size=2>Ver Cesta</font>
  <img src="<%=(baseUrl + "/images/pt.gif")%>" width="3" border="0">
  <img src="<%=(baseUrl + "/images/VerCarrito.gif")%>"
       width="15" height="13" border="0"></a>
  <img src="<%=(baseUrl + "/images/pt.gif")%>" width="10" border="0">
</td>
</tr>
</table>
```

6.4.3 El menú

La página de menú de la tienda tiene dos grupos de acciones. El primero contiene la acción de buscar, mientras que el segundo contiene la navegación por las categorías de artículos.

En este caso, se incluye un fichero estático que contiene las categorías (cache de categorías). Este fichero se vuelca de la base de datos desde la aplicación de Administración de la Tienda, de forma que se evitan accesos a la base de datos con cada iteración del cliente sobre la tienda.

La página de menú de la aplicación de Administración es muy semejante a la de la Tienda Virtual, que muestra en el siguiente listado.

Listado 6-7 Página de menú de la tienda

```
<%
  String baseUrl = (String)
request.getAttribute("javax.servlet.forward.context_path");
%>

<table border="0" cellpadding="0" cellspacing="1" width="100%">
<tr>
<td align=center>
 <table border="0" cellspacing="0" cellpadding="0" width="100%">
 <tr valign="bottom" align="center"><td>
  <table width="100%" border="0" cellspacing="0" cellpadding="0">
  <tr valign="top" bgcolor="#006600"><td width="5">
   <img src="<%=(baseUrl + "/images/esquina-superior-izquierda.gif")%>"
       height="5" border="0" width="5"></td>
  <td>
   <table width="99%" border="0" cellspacing="3" cellpadding="0">
   <tr><td valign="bottom">
    <font color="white" face="Verdana" size=2><b>BUSCAR:</b></font></td></tr>
  </table>
  </td>
  <td width="5" align="right">
    <img src="<%=(baseUrl + "/images/esquina-superior-derecha.gif")%>"
       height="5" border="0" width="5"></td></tr>
  </table>
 </td></tr>
 </table>

 <table border="0" cellpadding="0" cellspacing="1" width="100%"
     bgcolor="#006600">
 <form action="<%=(baseUrl + "/Buscar.do")%>" method=post>
 <tr bgcolor="#006600"><td bgcolor="#c4f2c4">
  <table border="0" cellpadding="0" cellspacing="1" width="100%">
  <tr align=left><td>
   <img src="<%=(baseUrl + "/images/pt.gif")%>" width="5">
   <input type="text" name="clave" size="10">
   <input type=image name="clave" value="Ir!" border=0 alt="Ir!"
      src="<%=(baseUrl + "/images/Buscar.gif")%>" border=0 align=absmiddle>
  </td></tr>
  <tr align=left><td width="2"></td></tr>
```

```
 </table>
 </td></tr>
 </form>
 </table>
</td>
</tr>
<tr><td><br></td></tr>
<tr>
<td align=center>
 <table border="0" cellspacing="0" cellpadding="0" width="100%">
 <tr valign="bottom" align="center"><td>
  <table width="100%" border="0" cellspacing="0" cellpadding="0">
  <tr valign="top" bgcolor="#000066">
  <td width="5">
   <img src="<%=(baseUrl + "/images/esquina-superior-izquierda.gif")%>"
         height="5" border="0" width="5" /></td>
  <td>
  <table width="99%" border="0" cellspacing="3" cellpadding="0">
  <tr><td valign="bottom">
   <font color="white" face="Verdana" size=2><b>CATEGORIAS</b></font></td></tr>
  </table>
 </td>
 <td width="5" align="right">
 <img src="<%=(baseUrl + "/images/esquina-superior-derecha.gif")%>" height="5"
     border="0" width="5" /></td></tr>
  </table>
 </td></tr>
 </table>
 <table border="0" cellpadding="0" cellspacing="1" width="100%"
    bgcolor="#006600">
 <tr bgcolor="#000066"><td bgcolor="#babafb">
   <table border="0" cellpadding="0" cellspacing="1" width="100%">
   <tr align=left><td>
   <font color="#000066" face="Verdana" size=2>
   <%@include file="categorias.html" %></font>
  </td></tr>
  <tr align=left><td width="2"></td></tr>
  </table>
 </td></tr>
 </table>
</td>
</tr>
</table>
```

6.4.4 El cuerpo de página

La página de cuerpo de página depende de la acción realizada sobre la tienda. Básicamente, existen cuatro tipos de páginas de cuerpo en la tienda y otros tres en la aplicación de administración.

- Páginas de cuerpo de la tienda:
  - *MuestraCatalogo*: Muestra la imagen de la categoría e incluye la página ListaProductos.
  - *ListaProductos:* Lista los productos que recibe en un ArrayList de productos. Hace uso del custom tag `iterator`.
  - *DetalleProducto:* Muestra el detalle de un producto.
  - *MuestraCarrito:* Muestra el contenido del carrito que recibe en una Hashtable. Hace uso del custom tag `enumeration`.
  - *RealizaCompra:* Muestra el formulario para hacer el pedido.
  - *Pedido:* Confirma el pedido o muestra un mensaje de error.

El siguiente listado muestra el contenido de la página *ListaProductos* y cómo esta página utiliza el custom tag `iterator`. Podemos observar lo clara que es esta página, ya que todo el tratamiento de iteraciones del bucle ha sido extraído y llevado al menejador del custom tag.

También podemos observar cómo el manejador del custom tag deja información a la que es posible acceder después de finalizar el propio tag.

Listado 6-8 Página ListaProductos

```
<%@taglib uri="/WEB-INF/classes/taglibrary/TiendaTagLib.tld" prefix="t" %>

<%
    String baseUrl = (String)
request.getAttribute("javax.servlet.forward.context_path");
%>

<t:iterator>
<table border="0" cellpadding="0" cellspacing="0" width="100%">
<tr>
 <td colspan="4" valign="top"><img height="10" width="20" src="<%=(baseUrl +
"/images/pt.gif")%>"></td>
</tr>
<tr>
 <td valign="top" align="center" width="65">
 <a href="<%=(baseUrl + "/DetalleProducto.do?id=" + id)%>">
 <img height="75" width="75" src="<%=(baseUrl+"/images/Productos/"+id+".jpg")%>">
 </a>
</td>
<td width="10"></td>
<td valign="top" width="334">
 <p><a href="<%=(baseUrl + "/DetalleProducto.do?id=" + id)%>">
 <font size="2" face="Verdana"><b><%=nombre%><br>(<%=marca%>)<br></b></font></a>
 <font size="2" face="Verdana"><%=descripcionCorta%></font>
</td>
<td valign="bottom" align="right">
 <table border="0" cellpadding="0" cellspacing="0" bgcolor="black">
 <tr>
 <td>
  <table border="0" cellpadding="0" cellspacing="1" width="160">
  <tr>
  <td bgcolor="#e8e8e8" align="right">
   <table border="0" cellpadding="1" cellspacing="0" width="100%">
   <tr>
   <td align="center">
    <font face="Verdana" size="2" color="#cc0000"><b>Precio: <%=precio%>
       &euro;</b></font>
   </td>
   </tr>
   <tr>
   <td align="center">
   <font face="Verdana" size="1" color="black"><%=precioPta%> PTA</font>
  </td>
  </tr>
  </table>
 </td>
 <td bgcolor="white" width="28">
  <a href="<%=(baseUrl + "/ProductoAlCarrito.do?id=" + id)%>">
  <img height="29" width="28" src="<%=(baseUrl + "/images/AlCarrito.gif")%>"></a>
 </td>
 </tr>
 </table>
</td>
</tr>
</table>
</td>
</tr>
```

```
<tr>
<td colspan="4">
 <img height="3" width="20" src="<%=(baseUrl + "/images/pt.gif")%>"></td>
</tr>
<tr>
<td colspan="4" bgcolor="black">
 <img height="1" width="20" src="<%=(baseUrl + "/images/pt.gif")%>"></td>
</tr>
</table>
</t:iterator>

<%
    if (numProductos.equals("0"))
    {
%>
<br><br><br>
<center>
 <font size="3" face="Verdana"><b>No se han encontrado productos</font>
</center>
<br><br><br><br><br>
<%
    }
%>
```

- Páginas de la aplicación de Administración. Estas páginas no utilizan custom tags para encapsular funcionalidad, por lo que ésta se debe desarrollar en scriplets dentro de la propia página.
  - *Bienvenida:* Muestra un mensaje de bienvenida al usuario.
  - *AdminCategorias:* Lista las categorías actuales y permite añadir nuevas categorías y modificar o eliminar las existentes.
  - *NuevoProducto:* Muestra un formulario para insertar un nuevo producto.
  - *OtrasAcciones:* Permite realizar consultas sobre la base de datos.

6.4.5 El Pie de página

El pie de página es una página html que muestra la nota de copyright de la Tienda Virtual y de la Administración de la Tienda.

CAPÍTULO 7

Desarrollando el Controlador

7.1 INTRODUCCIÓN

En los capítulos anteriores hemos aprendido a desarrollar el modelo y las vistas de la aplicación. En este capítulo veremos cómo se enlazan ambos bloques para terminar de construir nuestra Tienda Virtual.

Primero comenzaremos echando un vistazo al fichero de configuración de mapeos entre acciones y vistas y a su manejador, para a continuación describir el resto de las clases Java que contiene nuestro proyecto de Tienda Virtual y Administración de la Tienda.

7.2 MAPEOS OPERACIÓN-ACCIÓN-VISTA

7.2.1 El fichero de mapeos

El fichero mapeos.xml contiene el flujo configurable de la aplicación, asociando nombre de operación con la acción a realizar y la vista que se mostrará a continuación.

Este fichero de configuración lo utiliza el Gestor de Flujo para determinar la acción a realizar y la vista siguiente, dada una operación solicitada por el usuario.

También define la operación por defecto que se ejecutará si no se recibe ninguna petición de operación reconocible y la página de error que incluir, en caso de que ocurra un error al ejecuta la acción definida.

Listado 7-1 El fichero de mapeos

```
<?xml version="1.0" encoding="UTF-8"?>

<mapeos oDefault="Bienvenida.do" pError="Error.jsp">
    <mapeo operacion="ActualizaCarrito.do">
        <vista>MuestraCarrito.jsp</vista>
        <accion-class>
            controlador.acciones.AccionActualizaCarrito
        </accion-class>
    </mapeo>
    <mapeo operacion="Bienvenida.do">
        <vista>ListaProductos.jsp</vista>
        <accion-class>
            controlador.acciones.AccionBienvenida
        </accion-class>
    </mapeo>

        ...

</mapeos>
```

7.2.2 El Manejador de Mapeos

Esta clase es el manejador del fichero de configuración de mapeos acción-vista. Es llamada por el parser SAX para procesar las etiquetas xml del fichero de mapeos.

El fichero de mapeos será procesado y su contenido se almacenará en una tabla hash. La tabla hash se direcciona mediante el nombre de operación y cada elemento

contiene una instancia de la clase auxiliar `Mapeo`, que contiene la clase Java de la acción a realizar y la vista que le sigue, como se ve en el siguiente listado.

Listado 7-2 La clase Mapeo

```
public class Mapeo {
    public String accion;
    public String vista;
}
```

Ya hemos visto antes el fichero de mapeos que se utiliza realmente en la Tienda Virtual, para configurar el Gestor de Flujo. Sin embargo vamos a repasar las etiquetas que contiene para entender el funcionamiento del handler de este fichero xml.

El siguiente listado muestra el DTD del fichero de mapeos.

Listado 7-3 DTD del fichero de Mapeos

```
<?xml version='1.0' encoding='UTF-8'?>

<!ELEMENT accion-class (#PCDATA)>

<!ELEMENT vista (#PCDATA)>

<!ELEMENT mapeo (accion-class|vista)*>
<!ATTLIST mapeo
    operacion CDATA #IMPLIED
  >

<!ELEMENT mapeos (mapeo)*>
<!ATTLIST mapeos
    pError CDATA #IMPLIED
    oDefault CDATA #IMPLIED
  >
```

En este listado podemos ver las etiquetas que vamos a procesar en el Handler:

- `mapeos`: Al detectar esta etiqueta debemos recoger los atributos de operación por defecto y página de error.

- `mapeo`: Al detectar esta etiqueta crearemos una nueva instancia de la clase Mapeo y se guarda el nombre del atributo operación en una variable temporal. El nombre de la operación se utiliza para direccionar el mapeo en la tabla hash.

  Cuando se detecta el final de esta etiqueta, se almacena el mapeo en la tabla hash.

- `vista`: Al detectar el final esta etiqueta se recoge el contenido del cuerpo de la etiqueta, que debe contener el nombre de la vista asociada, y se almacena en el campo `vista` de la instancia de la clase Mapeo.

- `accion-class`: Al detectar el final esta etiqueta se recoge el contenido del cuerpo de la etiqueta, que debe contener el nombre de la clase Java de la acción a realizar, y se almacena en el campo `accion` de la instancia de la clase Mapeo.

El siguiente listado muestra los métodos más importantes del manejador de mapeos.

Listado 7-4 El Manejador de Mapeos

```
public void startElement(String namespaceURI, String sName, String qName,
                         Attributes attrs) throws SAXException {
  buffer=null;
  if (qName.equalsIgnoreCase("mapeos")) {
    if (attrs != null) {
      oDefault = attrs.getValue("oDefault").trim();
      pError = attrs.getValue("pError").trim();
    }
  } else if (qName.equalsIgnoreCase("mapeo")) {
      if (attrs != null) {
        mapeo = new Mapeo();
        operacion = attrs.getValue("operacion").trim();
      }
  }
}

public void endElement(String namespaceURI, String sName, String qName)
    throws SAXException   {
  if (mapeo==null)
    return;
  String cuerpo = buffer.toString();
  buffer=null;
  if (qName.equalsIgnoreCase("vista")) {
    mapeo.vista = cuerpo.trim();
  } else if (qName.equalsIgnoreCase("accion-class")) {
    mapeo.accion = cuerpo.trim();
  } else if (qName.equalsIgnoreCase("mapeo")) {
    mapeos.put(operacion, mapeo);
    mapeo=null;
  }
}
```

7.3 EL CONTROLADOR DE LA TIENDA

7.3.1 El Filtro

Al inicializarse el filtro, recoge los parámetros iniciales `controlador` y `controladorAdmin` del *deployment descriptor*, que contienen las URL relativas del controlador de la Tienda y del controlador de la Administración de la Tienda respectivamente.

Una vez inicializado, realiza el filtrado de la petición. Recoge todos los parámetros recibidos, y los pone como atributos eliminando los espacios iniciales y finales y hace log de estos parámetros.

Deja pasar la petición si el recurso solicitado es una imagen, redirige la petición al controlador de administración si se recibe la subcadena "/admin/" o sino al controlador de la tienda. Si la petición pertenece a la tienda, busca la operación en la URL recibida y la pone como atributo en la petición.

El siguiente listado muestra el método `doFilter` del Filtro.

Listado 7-5 El método doFilter del Filtro

```
public void doFilter(ServletRequest request, ServletResponse response,
```

```
    FilterChain chain) throws IOException, ServletException {

  for (Enumeration en=request.getParameterNames();en.hasMoreElements();) {
       String name = (String)en.nextElement();
       String value = request.getParameter(name);
       request.setAttribute(name, value.trim());
       StringBuffer buf = new StringBuffer();
       buf.append(name);
       buf.append("=");
       buf.append(value);
       log(buf.toString());
  }

  HttpServletRequest httpRequest = (HttpServletRequest)request;
  String URI = httpRequest.getRequestURI();
  if (URI.endsWith(".gif") || URI.endsWith(".jpg"))
       chain.doFilter(request, response);
  else if (URI.lastIndexOf("/admin/") > 0) {
       RequestDispatcher rd =
              application.getRequestDispatcher(controladorAdmin);
       rd.forward(request, response);
  } else {
       request.setAttribute("op", getOperation(URI));
       RequestDispatcher rd =
              application.getRequestDispatcher(controlador);
       rd.forward(request, response);
  }
}
```

7.3.2 El servlet Controlador

El servlet controlador está configurado en el *deployment descriptor* para ser el primero objeto de la aplicación en ser cargado en memoria, cuando se recibe la primera petición dirigida a la Tienda Virtual o a la aplicación de Administración de la Tienda. El método `init` del servlet Controlador es el encargado de crear, inicializar y poner en el contexto de la aplicación una instancia de la claseModeloTienda, que gestiona los datos de la Tienda.

El método `init` recoge del *deployment descriptor* los parámetros de inicialización, inicializa el modelo de la Tienda y se guarda el modelo en el contexto de la aplicación para que sea accesible por otros objetos de la aplicación.

El método `init` también es el encargado de inicializar el Gestor de Flujo de la Tienda Virtual. Para ello, debe pasar a su constructor el la ruta real del fichero de mapeos (por ejemplo “C:\\Midir\\mapeos.xml”) que contiene la configuración de los mapeos entre la operación recibida, la acción a realizar y la vista a mostrar, y que necesita el Gestor de Flujo.

Listado 7-6 Método init del Controlador

```
    public void init(ServletConfig config) throws ServletException {

        tienda = new ModeloTienda();
        tienda.setDbRefName(config.getInitParameter("dbRefName"));

        application = config.getServletContext();

        try {
            tienda.init();
        }
        catch (Exception e) {
            e.printStackTrace();
```

```
        }

        application.setAttribute("tienda", tienda);

        gestorFlujo = new GestorFlujo(application.getRealPath("/") +
                                      "WEB-INF\\mapeos.xml");
        super.init(config);
    }
```

Una vez inicializado el servlet controlador, está listo para comenzar a servir peticiones dirigidas a la Tienda Virtual.

Cada petición entrante debe contener un atributo "op" con la operación a realizar, que pone el filtro. El método `doPost` recoge este atributo de operación y llama al gestor de flujo para que ejecute la acción correspondiente.

A continuación, pone en el atributo cuerpo de la petición, la vista que sigue a la acción realizada y hace forward a la vista principal. La vista principal de la tienda es la vista compuesta, que muestra la aplicación y llama a la página de cuerpo, que muestra los contenidos en función de la acción a realizar.

Listado 7-7 Método doPost del Controlador

```
protected void doPost(HttpServletRequest request,
          HttpServletResponse response) throws ServletException, IOException {

   String op= (String) request.getAttribute("op");
   String URL="/jsp/VistaCompuesta.jsp";

   String cuerpo = gestorFlujo.doAccion(tienda, request, op);
   request.setAttribute("cuerpo", cuerpo);

   RequestDispatcher rd = request.getRequestDispatcher(URL);
   rd.forward(request, response);
}
```

7.3.3 La clase Gestor de Flujo

Gestiona el flujo de la aplicación. El flujo se configura mediante el fichero de mapeos, que relaciona la operación solicitada por el usuario, con la acción a ejecutar y la vista a mostrar.

Cuando se crea una instancia del Gestor de Flujo, se recibe el path absoluto del fichero html de mapeos. El constructor llama al parseador SAX con el manejador de mapeos `MapeosHandler`, para cargar los mapeos en una tabla hash. Recoge además la operación por defecto y la página de error.

El gestor de flujo utiliza un patrón *Command*. Con cada llamada al método `doAccion` se recibe la operación solicitada por el usuario. La operación se busca por su nombre en la tabla hash y se carga el objeto acción asociado, se inicializa con el modelo y se ejecuta la acción.

Si no encuentra operación solicitada, el método `doAccion` ejecuta la operación por defecto, que ha sido inicializada desde el fichero de mapeos.

Una vez se ha ejecutado la acción, devuelve la vista a mostrar. Si se produce un error al ejecutar la acción (normalmente porque no se encuentra la clase), devuelve la página de error.

Listado 7-8 El Gestor de Flujo

```
    public String doAccion(ModeloTienda tienda,
                           HttpServletRequest request, String op) {
        Accion accion = null;
        Mapeo mapeo = null;

        try {
            mapeo = (Mapeo) mapeos.get(op);
            if (mapeo==null)
                mapeo = (Mapeo) mapeos.get(oDefault);

            accion = (Accion)
              getClass().getClassLoader().loadClass(mapeo.accion).newInstance();
        }
        catch (Exception e) {
        }

        try {
            accion.setModelo(tienda);
            accion.perform(request);
            return mapeo.vista;
        } catch (Exception e) {
            String error = "Se ha producido un error al ejecutar: " +
                    mapeo.accion + "<br><br>" + e.toString();
            request.setAttribute("error", error);
            return pError;
        }
    }
```

7.3.4 El servlet ControladorAdmin

El servlet controlador de la aplicación de Administración de la Tienda es la puerta de entrada de la administración y es el responsable de validar los usuarios. La información relativa a los usuarios está controlada por la clase `ModeloUsuarios`, del bloque del modelo de la aplicación.

En su método `init`, el servlet `ControladorAdmin` inicializa una instancia de esta clase, recogiendo del deployment descriptor los parámetros relativos a la base de datos, como el driver jdbc a utiliza, la URL de la base de datos y el usename y password con los que acceder a la base de datos de Usuarios.

Además, recoge del contexto de la aplicación la instancia del modelo de la tienda con la que acceder a los datos de la tienda. Esta instancia del modelo de la tienda ha sido guardada en el contexto de la aplicación el servlet Controlador de la Tienda.

Con cada petición se comprueba si existe un usuario registrado, accediendo a la sesión y comprobando si existe el atributo `usuario`. Este atributo contiene una instancia de la clase usuario, con los datos del usuario validado.

Si no existe un usuario validado, se buscan los atributos `Login` y `Password` en la petición. Si se encuentran, se intenta validar el usuario en el modelo de usuarios y si se consigue se almacena el usuario en la sesión, se crean las instancias de las clases `CategoriasAFichero` y `FileUpload` y se colocan también en la sesión. Cuando el usuario sale de la sesión se deben eliminar estas instancias de la sesión.

A continuación se realiza el despacho hacia la acción solicitada. Este controlador se diferencia del controlador de la Tienda en varios aspectos y se ha construido de esta forma para mostrar estos diferentes planteamientos.

- Se utiliza el parámetro `accion`, que debe ser enviado con cada petición. Este atributo contiene el nombre de la acción solicitada.

- La selección de la acción a realizar y de la vista siguiente, están configurada dentro del código, en vez de estarlo en un fichero de configuración.

El siguiente listado muestra el código del método `doPost` del Controlador de Administración.

Listado 7-9 El método doPost del ControladorAdmin

```
protected void doPost(HttpServletRequest request,
          HttpServletResponse response) throws ServletException, IOException {

  boolean invalidar = false;
  String accion= (String)request.getAttribute("accion");
  HttpSession session = request.getSession(true);
  Usuario usuario = (Usuario) session.getAttribute("usuario");

  if (usuario==null) {
    String login = (String)request.getAttribute("Login");
    String passwd = (String)request.getAttribute("Password");
    if ((login == null) || (passwd == null)) {
      invalidar = true;
      accion = null;
    } else {
      usuario = usuarios.UsuarioAutorizado(login, passwd);
      if (usuario!=null) {
        session.setAttribute ("usuario", usuario);
        CategoriasAFichero cache = new CategoriasAFichero();
        cache.setNombrefichero(application.getRealPath("/") +
                               "jsp\\categorias.html");
        session.setAttribute("cache", cache);
        FileUpload upload = new FileUpload();
        upload.setCarpeta(application.getRealPath("/") +
                          "images\\productos\\");
        session.setAttribute("upload", upload);
      } else {
        invalidar = true;
        accion = null;
      }
    }
  }

  // Dispatch petición
  String URL = "/jsp/admin/Login.jsp"; // página de login por defecto

  if (accion!=null && usuario!=null &&
          usuarios.UsuarioEnRol(usuario.userName, rolAdministrador)) {
    URL="/jsp/admin/VistaCompuesta.jsp"; // Página de la aplicación
    String cuerpo="Bienvenida.jsp"; // vista por defecto

    if (accion.equals("AdminCategorias")) {
      AccionAdminCategorias adminCategorias =
                           new AccionAdminCategorias();
      adminCategorias.setModelo(tienda);
      adminCategorias.perform(request);
      cuerpo="AdminCategorias.jsp";
    } else if (accion.equals("NuevoProducto")){
                . . .
    }
    request.setAttribute("cuerpo", cuerpo);
  } else
    invalidar = true;

  // Si hay que invalidar el usuario, se saca su id de la sesión, si estaba.
  if (invalidar) {
```

```
    session.removeAttribute("usuario");
    session.removeAttribute("cache");
    session.removeAttribute("upload");
  }

  RequestDispatcher rd = request.getRequestDispatcher(URL);
  rd.forward(request, response);
}
```

7.4 LAS ACCIONES

Como hemos visto, tanto el servlet Controlador como el servlet ControladorAdmin delegan la ejecución de las operaciones solicitadas por el usuario, en acciones. Las acciones recogen la información que envía el usuario y la aplican sobre el modelo.

Nuestra Tienda Virtual define una clase abstracta, denominada Accion y se implementa una acción diferente por cada operación. El siguiente listado muestra la clase abstracta Accion.

Listado 7-10 La clase abstracta Accion

```
public abstract class Accion {
    protected ModeloTienda tienda; // modelo de acceso a los datos de la tienda
    protected ServletContext application;

    public Accion() {}

    /** Almacena el modelo de la tienda
     *
     * @param tienda
     */
    public void setModelo(ModeloTienda tienda) {
        this.tienda = tienda;
    }

    /** Devuelve el nombre de la acción
     */
    public abstract String getName();

    /** Realiza la acción.
     *
     * @param request
     */
    public abstract void perform(HttpServletRequest request);
};
```

El siguiente listado muestra una implementación de la clase `Accion`, la clase `AccionProductoAlCarrito`. Esta clase recoge el carrito de la sesión o lo crea nuevo si no existía, recoge el identificador del producto que el cliente desea añadir al carrito y llama al modelo de la tienda para obtener toda la información del producto. A continuación, añade el producto al carrito, volviendo a poner el carrito en la sesión. También pone el carrito en la petición para que lo muestre la vista correspondiente.

Listado 7-11 La acción de añadir un producto al carrito

```
public class AccionProductoAlCarrito extends Accion {

    public String getName() {
        return "ProductoAlCarrito";
    }
```

```
    public void perform(HttpServletRequest request)
      throws ExcepcionFalloGeneral {
        HttpSession session=request.getSession(true);
        ModeloCarrito carrito = (ModeloCarrito) session.getAttribute("carrito");
        int id;
        if (carrito==null)
            carrito = new ModeloCarrito();
        try {
            id = Integer.parseInt((String)request.getAttribute("id"));
        }
        catch (Exception e) {
            throw new ExcepcionFalloGeneral(
                   "No se encontraron los parámetros esperados o son erróneos");
        }
        Producto producto = tienda.detallesProducto(id);
        carrito.nuevoProducto(producto);
        session.setAttribute("carrito", carrito);
        request.setAttribute("carrito", carrito);
    }
}
```

CAPÍTULO 8

Utilidades

8.1 INTRODUCCIÓN

En este capítulo codificaremos dos clases Java que proporcionan la funcionalidad de envío de ficheros al servidor, la clase `UploafFile`, y de cache de las categorías de productos a fichero, la clase `CategoriasAFichero`.

Al paquete de utilidades también pertenece la clase `FechaYHora`. Sin embargo, esta clase se utiliza como un JavaBean desde las páginas jsp y se ha explicado en el capítulo relativo a las vistas.

8.2 LA CLASE UPLOADFILE

Para enviar un fichero de texto o de datos al servidor utilizamos la etiqueta HTML `<INPUT NAME=filename TYPE=FILE>`, dentro de un formulario con el tipo MIME "`MULTIPART/FORM-DATA`".

El siguiente listado muestra un ejemplo de formulario parecido al que se utiliza en la página *NuevoProducto.jsp*, de la aplicación de Administración de la Tienda.

Listado 8-1 Formulario para el envío de ficheros

```
                ......

<form action="<%=(baseUrl + "/admin/ControladorAdmin?accion=NuevoProducto")%>"
 enctype="MULTIPART/FORM-DATA" method=post>

Nombre: <input type=text name=nombre size=70><br>
Marca: <input type=text name=marca size=30><br>
Destacado: <input type=checkbox name=destacado size=14><br>
Descripción: <textarea name=descripcionCorta rows="5" cols="73"></textarea><br>
Selecciona el fichero de imagen: <input type=file size=60 Name=imagen><br>
<input type=submit value="Insertar">

</form>
                ......
```

Cuando se encuentra un tipo FILE, el navegador muestra una caja de texto para introducir el nombre del fichero y un botón que despliega una caja de diálogo para explorar el sistema de ficheros.

Los datos del formulario se envían al servidor en una petición http que tiene un formato diferente del que se emplea en el resto de peticiones POST.

El cuerpo de la petición HTTP contiene todos los valores recogidos del formulario, incluyendo el fichero enviado. Estos valores están separados por un delimitador, que consiste en varios guiones (-) seguidos de un número aleatorio.

```
-----------------------------7d15340138
```

El mismo delimitador actúa como separador entre dos valores del formulario. El último delimitador que indica el final del cuerpo de la petición, contiene dos guiones más:

```
-----------------------------7d15340138--
```

Un valor que viene de un elemento distinto de “File”, el delimitador es seguido por la siguiente línea:

```
Content-Disposition: form-data; name=inputName
```

Esta línea es seguida por dos retornos de carro y por el valor del elemento.

Para un fichero, al delimitador le siguen dos líneas.

- La primera contiene el nombre del elemento FILE y el path completo del fichero en el ordenador del usuario.
- La segunda contiene el tipo de contenido del fichero. Este valor depende del fichero a ser enviado.

```
Content-Disposition: form-data; name="Filename"; filename="C:\hola.html“
Content-Type: text/html
```

La información (parámetros y ficheros) recibidos mediante un formulario `MULTIPART/FORM-DATA` no es accesible mediante el método `getParameter` y similares, sino que es necesario crear una clase específica que recoja esta información.

La clase `UploadFile` implementa el método `doUpload`, que recoge los parámetros y ficheros recibidos y almacena los pares nombre-valor de parámetros y de ficheros (el valor para un fichero es el path absoluto con el que se guardó en disco).

Una vez se ha recibido y procesado la petición, los parámetros están disponibles mediante el método `getParameter`.

Cuando se recibe un fichero, se almacena en la carpeta temporal por defecto con un nombre temporal o se almacena en la carpeta seleccionada por el usuario mediante `setCarpeta` y se le asigna el nombre con el que viene desde la petición.

Si se opta por la opción de fichero temporal se debe utilizar el método `setFicheroTemporal` para indicarlo. Una vez recibidos los ficheros, se puede utilizar el método `renombraFichero`, pasándole el nombre de parámetro con el que se recibió el fichero.

El siguiente listado muestra el código completo para la clase `UploadFile`.

Listado 8-2 La clase UploadFile

```
public class FileUpload {
    private static int MAXLINEA=256; // Número máximo de caracteres a leer
    private String carpeta = null; //Carpeta donde almacenar los ficheros
    private String nombrefichero = null; // nombre del fichero recibido
    private boolean ficheroTemporal = false;//false:renombrar el fichero temporal

    private String delimitador = null; // delimitador de parámetros
    private ServletInputStream in=null;
    private byte[] line; // buffer donde se almacena cada lectura de línea

    private Hashtable parametros=new Hashtable();

    public void setCarpeta (String carpeta) {
        this.carpeta = carpeta;
```

```
    }

    public void setNombrefichero (String nombrefichero) {
        this.nombrefichero = nombrefichero;
    }

    public void setFicheroTemporal () {
        ficheroTemporal=true;
    }

    public int doUpload (HttpServletRequest request) throws IOException {
        if (!enable)
            return -1;

        in = request.getInputStream();
        line = new byte[MAXLINEA];

        int i = in.readLine(line, 0, MAXLINEA);
        if (i==-1) return i;

        int longDelimitador = i - 2;
        delimitador = new String(line, 0, longDelimitador);

        while (i != -1) {
            String linea = new String(line, 0, i);
            if (linea.startsWith("Content-Disposition: form-data; name=\"")) {
                String paramFichero=esFichero(linea);
                if (paramFichero!=null)
                    doFichero(linea, paramFichero);
                else
                    doParameter(linea);
            }
            i = in.readLine(line, 0, MAXLINEA);
        }
        return 0;
    }

    private void doParameter(String lineaActual) throws IOException {
        int i;

        int pos = lineaActual.indexOf("name=\"");
        String nombre = lineaActual.substring(pos+6, lineaActual.length()-3);

        i = in.readLine(line, 0, MAXLINEA);
        i = in.readLine(line, 0, MAXLINEA);
        String valor = new String(line, 0, i-2).trim();
        parametros.put(nombre,valor);
        i = in.readLine(line, 0, MAXLINEA);
    }

    private String esFichero(String lineaActual) throws IOException {
        String ef = null;

        int pos = lineaActual.indexOf("\"; filename=\"");
        if (pos != -1) {
            int posi = lineaActual.indexOf("name=\"");
            String nombreParam = lineaActual.substring(posi+6, pos);
            String pathfichero = lineaActual.substring(pos+13,
                                                       lineaActual.length()-3);
            pos = pathfichero.lastIndexOf("\\");
            if (pos != -1)
                nombrefichero = pathfichero.substring(pos + 1);
            else
                nombrefichero = pathfichero;
            ef = nombreParam;
        }
        return ef;
    }

    private int doFichero(String lineaActual, String nombreParametro)
```

```
    throws IOException {
        int i;

        i = in.readLine(line, 0, MAXLINEA);
        i = in.readLine(line, 0, MAXLINEA);
        i = in.readLine(line, 0, MAXLINEA);

        ByteArrayOutputStream buffer = new ByteArrayOutputStream();
        String linea = new String(line, 0, i);

        if ((i-2)==0) // El fichero está vacío
            return 0;

        while (i != -1 && !linea.startsWith(delimitador)) {
            buffer.write(line, 0, i);
            i = in.readLine(line, 0, MAXLINEA);
            linea = new String(line, 0, i);
        }
        try {
            // Crear fichero temporal.
            File temp = File.createTempFile("atiii", "");
            temp.deleteOnExit();
            RandomAccessFile f = new RandomAccessFile(temp, "rw");
            byte[] bytes = buffer.toByteArray();
            int l=bytes.length-2;
            if (l<0) l=0;
            f.write(bytes, 0, l);
            f.close();
            if (ficheroTemporal){ // almacena en la tabla de parámetros
                parametros.put(nombreParametro, temp.getPath());
            }
            else { // almacena en la tabla de parámetros y renombra el fichero
                parametros.put(nombreParametro, carpeta + nombrefichero);
                File file = new File(nombrefichero);
                File dir = new File(carpeta);
                temp.renameTo(new File(dir, file.getName()));
            }
            return 1;
        }
        catch (Exception e) {
            System.err.println("FileUpload::dumpRequest: Excepción: "+
                               e.toString());
            return -1;
        }
    }

    public void renombraFichero (String nombreParametro) {
        String nombreFichTemp=(String) parametros.get(nombreParametro);
        if (nombreFichTemp!=null){
            File temp = new File(nombreFichTemp);
            parametros.remove(nombreParametro);
            parametros.put(nombreParametro, carpeta + nombrefichero);
            File file = new File(nombrefichero);
            File dir = new File(carpeta);
            temp.renameTo(new File(dir, file.getName()));
        }
    }

    public String getParameter(String nombre) {
        return (String) parametros.get(nombre);
    }

    public Enumeration getParameterNames() {
        return parametros.keys();
    }
}
```

8.3 LA CLASE CATEGORIASAFICHERO

Normalmente los datos se almacenan en una base de datos. Sin embargo, el acceso a la base de datos es una operación lenta, por lo que reducir el número de accesos que requieren las aplicaciones, implica hacer que éstas sean más rápidas y escalables.

Una solución para reducir el número de accesos es cachear los datos en un fichero de texto.

Esta solución escribe datos a los que se accede frecuentemente pero que son pocas veces cambiados, en ficheros de texto. Cuando la aplicación necesite los datos, hará un *include* del fichero de texto, en vez de acceder a la base de datos. La operación de incluir un fichero es miles de veces más rápida que abrir una conexión con la base de datos o de acceder a la base de datos con una conexión ya establecida.

Sin embargo, esta técnica presenta el problema del mantenimiento de los datos. Cuando los datos en la base de datos cambian, su copia en el fichero de texto debe cambiar. Por lo tanto, es necesario disponer de un procedimiento de actualización.

La solución a este problema de mantenimiento es modificar las páginas de administración, de forma que cuando algo cambie en la base de datos, la copia en el fichero de texto también cambie.

Nuestra Tienda Virtual utiliza la técnica de cache sobre la tabla de categorías de productos. Cada vez que un usuario realiza una petición a la Tienda es necesario consultar las categorías para componer el menú. Si fuese necesario acceder a la base de datos para consultar las categorías con cada petición de usuario, la aplicación sería más lenta. Además, las categorías suelen presentar pocos cambios, por lo que son unas buenas candidatas a ser cacheadas en fichero.

Para la actualización del fichero de cache cuando cambie la tabla de categorías, vamos a desarrollar una clase java que reciba las categorías de la base de datos y genere un nuevo fichero de cache.

El siguiente listado muestra la clase `CategoriasAFichero`, que proporciona esta funcionalidad.

Listado 8-3 La clase CategoriasAFichero

```
public class CategoriasAFichero {
  private String nombrefichero = null; //nombde del fichero a generar

  public void setNombrefichero (String nombrefichero) {
   this.nombrefichero = nombrefichero;
  }

  public void doCache (Hashtable categorias, String link){
    if (nombrefichero==null)
        return;

    StringBuffer buffer = new StringBuffer(2048);
    Enumeration enum;
    String idCategoria;
    String categoria;

    try {
```

```
        enum = categorias.keys();
        while (enum.hasMoreElements()) {
            idCategoria = (String)enum.nextElement();
            categoria = (String)categorias.get(idCategoria);
            buffer.append("<a href=\"");
            buffer.append(link);
            buffer.append(idCategoria);
            buffer.append("\">");
            buffer.append(categoria);
            buffer.append("</a><br>\n");
        }
        FileWriter fw = new FileWriter(nombrefichero);
        fw.write(buffer.toString());
        fw.close();
    }
    catch (Exception e) {
        System.err.println(e.toString());
    }
  }
}
```

Una acción del controlador de administración realiza el acceso a la base de datos, a través del modelo y llama a esta clase para realizar el volcado de los datos. El siguiente listado muestra el uso de la clase `CategoriasAFichero`.

Listado 8-4 Acción de cache de categorías

```
Hashtable categorias = tienda.listaCategorias();
String baseUrl = (String)
                  request.getAttribute("javax.servlet.forward.context_path");
if(cache!=null)
    cache.doCache(categorias, baseUrl+"/MuestraCatalogo.do?idCategoria=");
```

El resultado, es un fichero html con la lista de categorías y enlaces como el que puede verse en el siguiente listado.

Listado 8-5 La cache de categorías

```
<a href="/Tienda_V.1.3/MuestraCatalogo.do?idCategoria=3">Esqui / Snow</a><br>
<a href="/Tienda_V.1.3/MuestraCatalogo.do?idCategoria=2">Ciclismo</a><br>
<a href="/Tienda_V.1.3/MuestraCatalogo.do?idCategoria=1">Piraguismo</a><br>
```

CAPÍTULO 9

Despliegue de la aplicación

9.1 INTRODUCCIÓN

Este capítulo describe el proceso de despliegue de la Tienda. Comenzaremos con el descriptor de despliegue, veremos cómo utilizar los scripts sql para crear las bases de datos en MySQL, continuaremos con las instrucciones para desplegar la aplicación utilizando Tomcat 5.x. y configuraremos las fuentes de datos para acceder a las bases de datos.

9.2 EL DESCRIPTOR DE DESPLIEGUE

El descriptor de despliegue es un documento XML que contiene información que describe la aplicación. El descriptor de despliegue de la tienda (web.xml) especifica el valor de los parámetros iniciales. En concreto, se especifica:

- **Referencia a la fuente de datos**. Se especifica el nombre por el cual la fuente de datos será referenciada y la clase Java de la factoría de objetos *DataSource*.
- **dbRefName**: Nombre de la referencia a la fuente de datos.
- **controlador**: Url con la que acceder al controlador de la tienda.
- **controladorAdmin**: Url con la que acceder al controlador de la aplicación de administración de la tienda.
- **jdbcDriver**: Driver de acceso a la base de datos de Usuarios.
- **dbUrl**: Url de la base de datos de Usuarios.
- **dbUserName**: Nombre de usuario para acceder a la base de datos de Usuarios.
- **dbPassword**: Contraseña para acceder a la base de datos de Usuarios.
- **rolAdministrador**: Rol del asignado al administrador de la Tienda.

Listado 9-1 El descriptor de despliegue

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
  <display-name>Tienda Virtual V.1.3</display-name>
  <description>
      Tienda Virtual V.1.3
  </description>
  <resource-ref>
      <description>
        Referencia a una factoria de instancias java.sql.Connection
      </description>
      <res-ref-name>jdbc/Tienda</res-ref-name>
      <res-type>javax.sql.DataSource</res-type>
      <res-auth>Container</res-auth>
  </resource-ref>
  <filter>
    <filter-name>Filtro</filter-name>
    <filter-class>controlador.Filtro</filter-class>
```

```
    <init-param>
      <param-name>controlador</param-name>
      <param-value>/servlet/Controlador</param-value>
    </init-param>
    <init-param>
      <param-name>controladorAdmin</param-name>
      <param-value>/servlet/admin/Controlador</param-value>
    </init-param>
  </filter>
  <filter-mapping>
    <filter-name>Filtro</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
  <servlet>
    <servlet-name>Controlador</servlet-name>
    <servlet-class>controlador.Controlador</servlet-class>
    <init-param>
      <param-name>dbRefName</param-name>
      <param-value>jdbc/Tienda</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet>
    <servlet-name>ControladorAdmin</servlet-name>
    <servlet-class>controlador.admin.ControladorAdmin</servlet-class>
    <init-param>
      <param-name>jdbcDriver</param-name>
      <param-value>org.gjt.mm.mysql.Driver</param-value>
    </init-param>
    <init-param>
      <param-name>dbUrl</param-name>
      <param-value>jdbc:mysql://localhost:3306/Usuarios</param-value>
    </init-param>
    <init-param>
      <param-name>dbUserName</param-name>
      <param-value/>
    </init-param>
    <init-param>
      <param-name>dbPassword</param-name>
      <param-value/>
    </init-param>
    <init-param>
      <param-name>rolAdministrador</param-name>
      <param-value>adminTienda</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>Controlador</servlet-name>
    <url-pattern>/servlet/Controlador</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>ControladorAdmin</servlet-name>
    <url-pattern>/servlet/admin/Controlador</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>30</session-timeout>
  </session-config>
</web-app>
```

9.3 DESPLIEGUE

9.3.1 Creación de las bases de datos

Bajo la carpeta “Tienda V.1.3\db” se encuentran los scripts sql que permiten crear y poblar las bases de datos de la Tienda Virtual. Antes de ejecuta los scripts debemos asegurarnos que existe una instancia de la base de datos MySQL en

ejecución. En el momento de editar este libro, la última versión disponible de MySQL era la 4.1.

El fichero CreaBasesDatos.bat, crea las bases de datos, crea las tablas y puebla las tablas con datos. Su contenido se muestra en el siguiente listado.

Listado 9-2 El fichero CreaBasesDatos.bat

```
rem Script para crear la base de datos de la Tienda Virtual y de Usuarios de la
Tienda

cls

mysqladmin drop Tienda
mysqladmin drop Usuarios
mysqladmin create Tienda
mysqladmin create Usuarios
mysqladmin reload

mysql -e="\. CreaTablasTienda.sql"   Tienda
mysql -e="\. InsertaDatosTienda.sql" Tienda

mysql -e="\. CreaTablasUsuarios.sql"   Usuarios
mysql -e="\. InsertaDatosUsuarios.sql" Usuarios
```

Como se puede observar, este fichero ejecuta los scripts sql sobre la base de datos MySQL. Los siguientes listados muestran el contenido de los scripts "CreaTablasTienda.sql", "CreaTablasUsuarios.sql" y "InsertaDatosUsuarios.sql".

Listado 9-3 El script CreaTablasTienda.sql

```
# Crea las tablas de la Tienda

drop table if exists Categorias;
create table Categorias
(
       IdCategoria   int not null auto_increment,
       Categoria     char(30) not null,
       primary key(IdCategoria)
);

drop table if exists Productos;
create table Productos
(
       IdProducto    int not null auto_increment,
       IdCategoria   int references Categorias,
       Nombre        char(100) not null,
       Marca         char(50),
       DescripcionCorta  char(255),
       DescripcionLarga  text,
       Precio        double not null,
       Destacado     bit,
       primary key(IdProducto)
);

drop table if exists Pedidos;
create table Pedidos
(
       IdPedido         bigint not null unique,
       NombreContacto   char(100) not null,
       DireccionEntrega char(50) not null,
       NombreCC         char(50) not null,
       NumeroCC         char(50) not null,
       FechaCC          char(50) not null,
       primary key(IdPedido)
);
```

```
drop table if exists DetallesPedido;
create table DetallesPedido
(
       IdDetalle     int not null auto_increment,
       IdPedido      bigint not null references Pedidos,
       IdProducto    int not null references Productos,
       Cantidad      int not null,
       Precio        double not null,
       primary key(IdDetalle)
);
```

Listado 9-4 El script CreaTablasUsuarios.sql

```
# Crea las tablas de Usuarios de la Tienda

drop table if exists Usuarios;
create table Usuarios
(
       UserName     char(15) not null unique,
       Password     char(15) not null,
       Descripcion  char(100),
       Alias        char(20),
       Email        char(50),
       primary key (UserName)
);

drop table if exists Roles;
create table Roles
(
       UserName     varchar(15) not null,
       Rol          varchar(15) not null,
       primary key (UserName, Rol)
);
```

Listado 9-5 El script InsertaDatosUsuarios.sql

```
# Introduce los datos de los usuarios

delete from Usuarios;

insert into Usuarios (UserName, Password, Descripcion, Alias, Email)
       values ('admin', 'admin', 'Administrador de la Tienda',
                'Administrador', 'administrador.atiii@upf.edu');

delete from Roles;

insert into Roles (UserName, Rol)
       values ('admin', 'adminTienda');

insert into Roles (UserName, Rol)
       values ('admin', 'otroRol');
```

Para crear las bases de datos de la Tienda Virtual ejecutamos el fichero CreaBasesDatos.bat. Nos preguntará si queremos realmente eliminar las bases de datos “Tienda” y “Usuarios”. Contestamos afirmativamente y cuando finalice la ejecución del fichero bat se habrán creado las bases de datos, las tablas y se habrán poblado con datos.

9.3.2 Configuración del DataSource de la Tienda

Previamente a este apartado, se debe comprobar que el driver JDBC de la base de datos está instalado. El driver JDBC oficial es Connector/J.

La base de datos de la tienda se accede mediante un *DataSource*, es decir, un pool de conexiones con la base de datos. En este apartado vamos a configurar este *DataSource*.

El primer paso consiste en crear un recurso JDBC (un *datasource*) en Tomcat, para ello utilizaremos la herramienta de administración de Tomcat.

Una vez hemos accedido a la herramienta de administración, tecleando el login y password de administrador, elegimos la opción "*DataSources*". A continuación, seleccionamos la opción "Create New DataSource" e introducimos la información necesaria, como se observa en la siguiente figura.

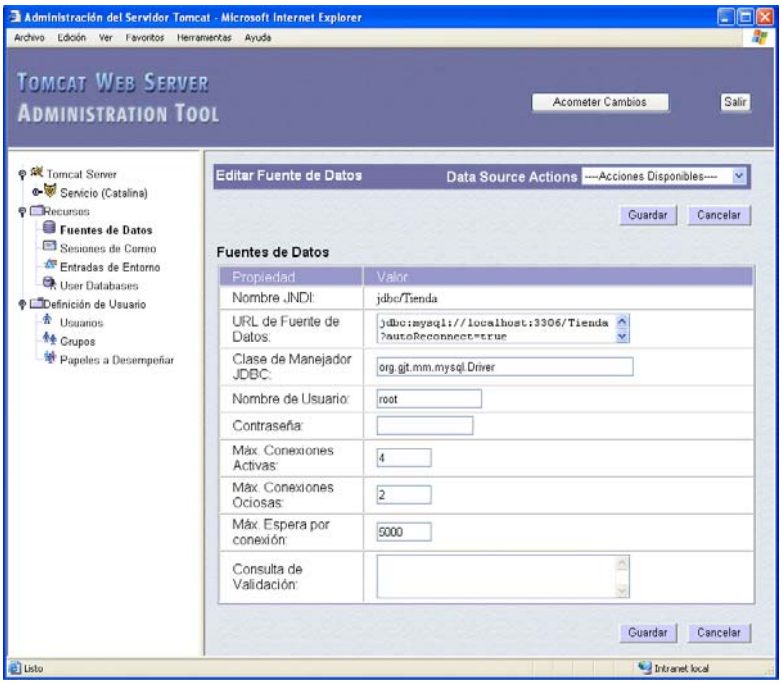

Figura 9-1 Administración de DataSources

- **Nombre JNDI**: `jdbc/Tienda`
- **URL de Fuentes de Datos**: `jdbc:mysql://localhost:3306/Tienda?autoReconnect=true`
- **Clase de Manejador JDBC**: `org.gjt.mm.mysql.Driver`
- **Nombre de usuario**: Nombre de usuario que utilizará el driver para acceder a la base de datos.
- **Contraseña**: Contraseña del usuario de la base de datos.
- **Máx**. **Conexiones Activas**: Número máximo de conexiones en el pool (0: sin límite).
- **Máx Conexiones Ociosas:** Número máximo de conexiones inactivas del pool (-1: sin límite).

- **Máx. Espera por Conexión**: Máximo tiempo a esperar (en ms) para establecer una conexión. Cuando se sobrepase se lanzará una excepción. (-1: espera indefinidamente).

- **Consulta de Validación**: <Consulta que puede utilizar el pool para validar conexiones, antes de entregarlas a la aplicación. Si se especifica, la consulta debe ser una consulta SELECT que devuelva, al menos, una columna>

La opción `autoReconnect=true`, se asegura de que el driver JDBC reconectará automáticamente las conexiones que MySQL cierra después de 8 horas de inactividad. La siguiente figura muestra un ejemplo de administración del *DataSource* de la Tienda.

Hacemos clic en "Guardar", después en "Acometer Cambios" y después en "Salir". En este momento tendremos la base de datos de la Tienda preparada para ser accedidas mediante un *DataSource* desde nuestra aplicación Web Tienda Virtual.

9.3.3 Configuración de la factoría de recursos en Tomcat

Para configurar la factoría de recursos de Tomcat, se debe añadir un nuevo elemento "*ResourceLink*" al fichero META-INF/context.xml de la aplicación, como se muestra en el siguiente listado.

Listado 9-6 El fichero context.xml

```
<Context path="/Tienda_V.1.3">
...
  <ResourceLink name="jdbc/Tienda"
                type="org.gjt.mm.mysql.Driver"
                global="jdbc/Tienda"/>
...
</Context>
```

9.3.4 Despliegue en Tomcat

En este apartado se asume que Tomcat está ya instalado y configurado de forma apropiada.

Junto con este documento se proporciona el código fuente de la Tienda Virtual y de la aplicación de Administración de la Tienda.

La versión desplegable se encuentra en "Tienda V.1.3\build\web". Para desplegar la aplicación en Tomcat 5.x tenemos que copiar esta carpeta y su contenido a la carpeta %CATALINA_HOME%/webapps/ y renombrar la carpeta a "Tienda V.1.3". El resultado final se observa en la siguiente figura.

Una vez se ha copiado la carpeta, es conveniente comprobar que la aplicación ha sido reconocida por Tomcat y se está ejecutando. Para realizar esta comprobación se puede utilizar la aplicación manager incluida en Tomcat.

Figura 9-2 Árbol de carpetas en Tomcat

9.3.5 Acceso a la aplicación

Una vez hemos seguido los pasos anteriores, podemos acceder a la aplicación desde un navegador, tecleando la URL http://localhost:<puerto>/Tienda_V.1.3. Aparecerá el escaparate de la tienda.

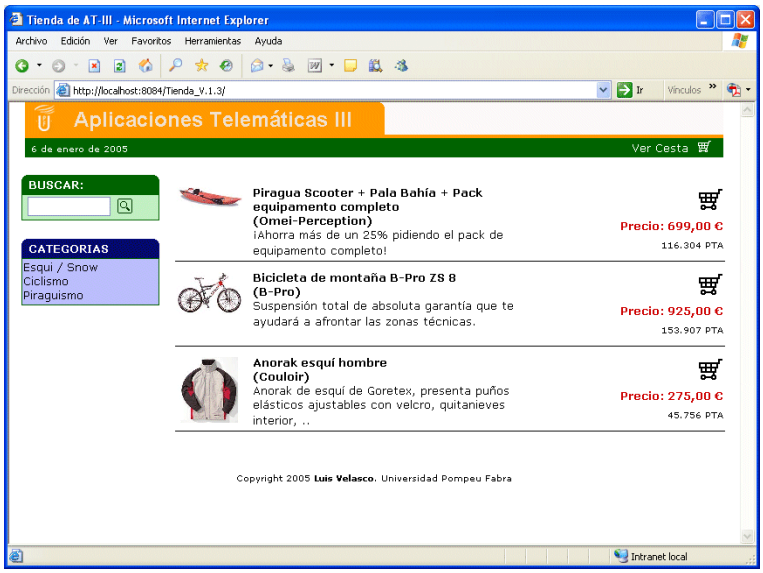

Figura 9-3 Escaparate de la Tienda

A partir de aquí podemos navegar por el resto de la Tienda. Si deseamos acceder a la aplicación de Administración de la Tienda, tecleamos la URL de administración http://localhost:<puerto>/Tienda_V.1.3/admin/. Aparecerá la página de login.

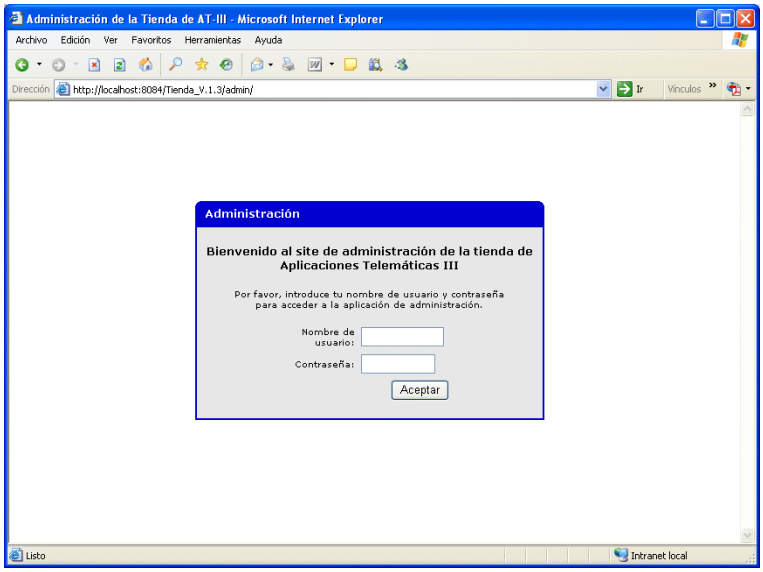

Figura 9-4 Acceso a la Administración de la Tienda

CONCLUSIONES

Este libro ha presentado una introducción al diseño y desarrollo de aplicaciones web con la plataforma J2EE. Su objetivo es servir de introducción a los conceptos y la tecnología utilizada en el diseño de aplicaciones web J2EE y proporciona ejemplos prácticos aplicables a una aplicación típica.

Aplicando los conceptos, hemos diseñado y desarrollado una aplicación web de comercio electrónico y hemos discutido las razones para elegir entre las diferentes opciones de diseño. Por último, se ha cubierto en detalle el diseño e implementación de cada componente.

REFERENCIAS

REFERENCIAS

[1] Singh, I. y Stearns, B. "Designing Enterprise Applications with the J2EE™ Platform". 2002. Addison-Wesley.

[2] Kurniawan, B. "Java for the Web with Sevlets, JSP, and EJB". 2002. New Riders Publishing.

[3] Hunter, J.. "Java™Servlet Programming". 2001. O'Reilly & Associates.

[4] Alur, D. et al. "Core J2EE patterns. Best Practices and Design Strategies". 2ª Ed. 2004. Sun Mycrosystems.

ANEXO A

GNU LICENCIA PÚBLICA GENERAL

Esta licencia aplica a la aplicación web Tienda Virtual y Administración de la Tienda, cuyo código fuente acompaña a este libro.

Palmira Granados
Licenciada en Derecho por la Escuela libre de derecho (Mexico))

> NOTA IMPORTANTE:
>
> Esta es una traducción no oficial al español de la GNU General Public License. No ha sido publicada por la Free Software Foundation, y no establece legalmente las condiciones de distribución para el software que usa la GNU GPL. Estas condiciones se establecen solamente por el texto original, en inglés, de la GNU GPL. Sin embargo, esperamos que esta traducción ayude a los hispanoparlantes a entender mejor la GNU GPL.
>
> IMPORTANT NOTICE:
>
> This is an unofficial translation of the GNU General Public License into Spanish. It was not published by the Free Software Foundation, and does not legally state the distribution terms for software that uses the GNU GPL--only the original English text of the GNU GPL does that. However, we hope that this translation will help Spanish speakers understand the GNU GPL better.
>
>
>
> 675 Mass Ave, Cambridge, MA 02139, EEUU
>
> Se permite la copia y distribución de copias literales de este documento, pero no se permite su modificación.

PREÁMBULO

Las licencias de la mayoría de los programas de cómputo están diseñadas para coartar la libertad de compartirlos y cambiarlos. Por el contrario, la Licencia Pública General GNU pretende garantizar esa libertad de compartir y cambiar Software Libre a fin de asegurar que el software sea libre para todos sus usuarios. Esta Licencia Pública General se aplica a la mayor parte del software de la Free Software

Foundation y a cualquier otro programa cuyos autores se comprometan a usarla. (Algunos otros paquetes de software de la Free Software Foundation están protegidos bajo la Licencia Pública General de Librería GNU.) Esta última licencia también puede aplicarse a nuevos paquetes de software.

Cuando se hable de Software Libre, se hace referencia a libertad, no a precio. Las Licencias Públicas Generales GNU están diseñadas para asegurar que el usuario tenga libertad de distribuir copias de Software Libre (y de recibir una remuneración por este servicio, si así se desea), que ese mismo usuario reciba el código fuente o que tenga la posibilidad de recibirlo, si así lo desea, que pueda cambiar o modificar el software o utilice sólo partes del mismo en nuevos paquetes de Software Libre; y que dicho usuario tenga pleno conocimiento de estas facultades.

Con la finalidad de proteger los derechos antes mencionados, es necesario establecer restricciones que prohíban a cualquiera negar esos derechos o pedir la renuncia a los mismos. Estas restricciones se traducen en ciertas responsabilidades para el usuario que distribuye o modifica copias de software protegido bajo estas licencias.

Por ejemplo, si una persona distribuye copias de un paquete de Software Libre protegido bajo esta licencia, ya sea de manera gratuita o a cambio de una contraprestación, esa persona debe dar a los receptores de esa distribución todos y cada uno de los derechos que él o ella misma tenga. Asimismo, esa persona debe asegurarse que dichos receptores reciban o tengan la posibilidad de recibir el código fuente. De igual manera, debe mostrarles esta licencia a fin de que tengan conocimiento de los derechos de los que son titulares.

La protección que otorga la presente licencia se hace de dos maneras simultáneas: (1) se otorga protección al software bajo la ley de copyright, y (2) se ofrece la protección bajo esta licencia, la cual otorga permiso legal para copiar, distribuir y/o modificar el software.

Asimismo, a fin de proteger a cada uno de los autores y a los creadores mismos de esta licencia, es importante hacer notar y que todos entiendan que no existe ninguna garantía de cualquier paquete de Software Libre por la cual se deba responder. Esto es, si el software es modificado por alguna persona distinta del autor y distribuido con esas modificaciones, los receptores de esa distribución deben saber que lo que han recibido no es la obra original, y que por lo tanto, cualquier problema surgido de las modificaciones no se reflejará en la reputación del autor original.

Finalmente, cualquier programa de Software Libre es amenazado por patentes de Software. Esta licencia tiene la finalidad de evitar el peligro que representa que los redistribuidores de programas de Software Libre obtengan individualmente licencias de patentes, haciendo de esta forma, programas de Software Propietario. Para lograr esto, queda totalmente claro que cualquier patente debe otorgar licencias que permitan el uso libre del programa para todos o no otorgar licencia alguna.

TÉRMINOS Y CONDICIONE PARA COPIA, MODIFICACIÓN Y DISTRIBUCIÓN.

0. Esta licencia se aplica a cualquier programa u otra obra que contenga un aviso puesto por el titular de los derechos de autor en el que se establezca que el mismo

puede ser distribuido bajo los términos de esta Licencia Pública General. El "Programa" se refiere a cualquier programa u obra, y "Obra basada en el Programa" se refiere por su parte, a, ya sea al "Programa" mismo a cualquier obra derivada del mismo según la ley de Derechos de Autor; esto es, una obra que contenga el "Programa" o una porción del mismo, ya sea que esta porción sea exactamente igual o modificada y/o traducida a otro idioma. (En adelante, una traducción se considerará de manera enunciativa, mas no limitativa, como una "modificación".)

1. Está permitido copiar y distribuir por cualquier medio copias fieles del código fuente del "Programa" tal y como fue recibido, siempre y cuando se publique en cada copia, de manera conspicua y apropiada, el aviso apropiado de derechos de autor y la renuncia a responder por la garantía correspondiente al "Programa", se mantengan intactos los avisos referentes a esta licencia y a la respectiva ausencia de cualquier garantía; y se entregue a los receptores del "Programa" una copia de esta licencia.

2. Está permitido modificar la copia o copias del "Programa" o cualquier parte del mismo, creando de esta forma, una "Obra basada en el Programa." Asimismo, está permitido copiar y distribuir las modificaciones antes mencionadas o la obra misma bajo los términos de la Sección 1 mencionada anteriormente, y siempre y cuando se cumplan de igual manera las condiciones siguientes:

a) Colocación de avisos, en la obra misma y por parte de quien realiza las modificaciones, en los que se informe que los archivos fueron modificados y la fecha de esas modificaciones.

b) Otorgamiento de una licencia bajo los términos establecidos en esta Licencia Pública General que abarque la obra en su totalidad y sin cargo a terceras personas para el caso en el que se distribuya o publique una obra que contenga todo o parte del "Programa" o que constituya una obra derivada del mismo.

c) Si el programa modificado normalmente lee comandos de manera interactiva cuando corre, cuando empiece a correr con dicho propósito interactivo, es necesario que aparezca un aviso que incluya la leyenda de derechos de autor correspondiente, así como la ausencia de responsabilidad por la garantía. Asimismo, dicho aviso deberá establecer que los usuarios de dicho programa tienen autorización para redistribuirlo bajo las mismas condiciones en las que les fue distribuido y les deberá informar cómo podrán tener acceso a una copia de esta licencia. (La excepción a esta condición tiene lugar cuando se trata de una "Obra basada en un Programa" que es en sí mismo interactivo, pero no envía normalmente un aviso.)

3. Copiar y distribuir el "Programa" (o una "Obra basada en el Programa" de acuerdo a la sección 2), bajo los términos de las secciones 1 y 2 mencionadas anteriormente, ya sea en código objeto o en su forma ejecutable está permitido, siempre y cuando dicho "Programa" se acompañe también por cualquiera de los siguientes:

a) El código fuente respectivo completo y leíble por una máquina, el cual debe ser distribuido bajo los términos establecidos en las secciones 1 y 2 mencionadas anteriormente y a través de un medio normalmente usado para el intercambio de software;

b) Una oferta por escrito y con una validez mínima de tres años, de proporcionar a cualquier tercera persona, por una cuota que no exceda el costo del acto fisico de distribuir, bajo los términos de las secciones 1 y 2 antes mencionadas; y a través de un medio normalmente usado para el intercambio de software; una copia del respectivo código fuente completo y leíble por una máquina; o,

c) Toda la información recibida respecto a la oferta de distribución del código fuente correspondiente. (Esta alternativa está permitida únicamente para distribuciones no comerciales y siempre y cuando el "Programa" se haya recibido en código objeto o en forma ejecutable junto con esta oferta de acuerdo a la subsección b antes mencionada.)

El código fuente de una obra se refiere a la forma preferida para hacerle modificaciones. En una obra ejecutable, el código fuente completo se refiere a todo el código fuente de todos los módulos que contiene, además de cualquier archivo de definición de interfaz asociado y de los **scripts** utilizados para controlar la compilación e instalación del ejecutable. Sin embargo, como una excepción especial, el código fuente distribuido no debe incluir cualquier cosa que sea normalmente distribuida (ya sea en forma de binarios o de código fuente) con los principales componentes del sistema operativo (como compilador, kernel, etc.) sobre el cual el ejecutable corre, a menos que el mismo componente acompañe al ejecutable.

4. El "Programa" no puede copiarse, modificarse, sublicenciarse ni distribuirse a menos que se haga bajo los términos y condiciones de esta licencia. Cualquier intento por hacer lo anterior de otra forma, será nulo y extinguirá automáticamente los derechos surgidos de esta licencia. Sin embargo, las licencias de las personas que hayan recibido copias o derechos bajo esta licencia, seguirán vigentes mientras dichas personas cumplan con sus obligaciones.

5. Mientras no se firme la presente licencia no existe obligación de aceptarla. Sin embargo, no existe autorización, y por lo tanto está legalmente prohibido, modificar o distribuir el "Programa" o una "Obra basada en el Programa" a menos que se acepten los términos y condiciones de la presente licencia. Por lo anterior, del acto de modificar o distribuir el "Programa" o una "Obra basada en el Programa" se presume la aceptación de los términos y condiciones de la presente licencia para copiar, distribuir o modificar dicho "Programa" u "Obra basada en el Programa".

6. Cada vez que se distribuya el "Programa" (o cualquier "Obra basada en el Programa"), quien recibe la copia del mismo recibe también, de manera automática una licencia de parte del licenciante original para copiar, distribuir o modificar el "Programa" bajo los términos y condiciones de esta licencia. No podrán imponerse más restricciones al ejercicio de los derechos del licenciatario que los establecidos en esta licencia. Quien distribuye el "Programa" no es responsable por el cumplimiento de la presente licencia por parte de terceras personas.

7. En el caso en el que como consecuencia de orden judicial o de las pretensiones demandadas por violación a una patente o por cualquier otra razón (de manera enunciativa, mas no limitativa) se imponen condiciones (ya sea por orden judicial, contrato o por otro medio) que se contradicen con las condiciones de esta licencia, estas últimas no se eximen de su cumplimiento. Como consecuencia de la imposibilidad de cumplir con ambas obligaciones mencionadas, el "Programa" no

podrá distribuirse. Por ejemplo, si una licencia de una patente prohíbe la redistribución gratuita del "Programa" por parte de quienes reciben copias del mismo de manera directa o indirecta, entonces la única forma de cumplir con ambas licencias, ésta y la de la patente, será abstenerse de distribuir el "Programa".

8. En el caso en el que la distribución y/o uso del "Programa esté restringida en ciertos países, ya sea por patentes o interfases protegidas por el sistema de propiedad intelectual, el titular original de los derechos de autor del "Programa" que lo sujeta a esta licencia tiene la facultad de agregar una limitación de tipo geográfico a la distribución, por virtud de la cual se excluya a dichos países; de manera que la distribución del mismo se permita únicamente en los países no excluidos. En este caso, dicha limitación se tiene como parte integrante de esta licencia.

9. Es facultad de la Free Software Foundation publicar, en cualquier momento, tanto versiones revisadas como versiones de reciente creación, de la Licencia Pública General. Las versiones nuevas pueden diferir en detalles a fin de afrontar y resolver nuevos problemas o preocupaciones, pero conservando siempre el espíritu de la presente versión.

10. En el caso en el que se deseen incorporar partes del "Programa" a otros paquetes de Software Libre cuyas condiciones de distribución difieran a estas, es necesario solicitar permiso por escrito al autor. Cuando se trate de software cuyo titular de los de los derechos de autor correspondientes sea la Free Software Foundation, la solicitud de permiso deberá dirigirse a ésta última, quien en algunas ocasiones hace excepciones como esta. La decisión emitida por la Free Software Foundation se basará tomando en cuenta la finalidad de preservar el estatus libre de todos los derivados del Software Libre y de promocionar que se comparta y se reutilice el software en general.

11. COMO CONSECUENCIA DE QUE EL "PROGRAMA" SE LICENCIE COMO GRATUITO, EN LA MEDIDA EN QUE LA LEY APLICABLE LO PERMITA, NO EXISTIRÁ GARANTÍA ALGUNA POR LA QUE SE DEBA RESPONDER. SALVO DISPOSICIÓN ESCRITA EN CONTRARIO, LOS TITULARES DE LOS DERECHOS DE AUTOR RESPECTIVOS Y/U OTRAS PARTES PONEN A DISPOSICIÓN EL "PROGRAMA" SIN GARANTÍA DE NINGÚN TIPO, EXPRESA O IMPLÍCITA, INCLUYENDO DE MANERA ENUNCIATIVA MAS NO LIMITATIVA, LAS GARANTÍAS IMPLÍCITAS DE TIPO COMERCIAL U OTRAS INHERENTES A ALGÚN PROPÓSITO ESPECÍFICO. EL RIESGO DE QUE EL "PROGRAMA" ESTÉ EN PERFECTAS CONDICIONES Y FUNCIONE TAL Y COMO DEBE FUNCIONAR CORRE POR CUENTA DE QUIEN LO RECIBE, AL IGUAL QUE LOS GASTOS NECESARIOS PARA SU SERVICIO, REPARACIÓN O CORRECCIÓN EN EL DADO CASO EN EL QUE DICHO "PROGRAMA" CONTENGA DEFECTOS.

12. A MENOS QUE ASÍ LO DISPONGA LA LEY APLICABLE O EXISTA ACUERDO ESCRITO EN CONTRARIO, NINGÚN TITULAR DE LOS DERECHOS DE AUTOR O PERSONA FACULTADA, SEGÚN LAS SECCIONES ANTERIORES DE LA PRESENTE, PARA MODIFICAR Y/O DISTRIBUIR EL "PROGRAMA" SERÁ RESPONSABLE POR LOS DAÑOS YA SEAN GENERALES, ESPECIALES, INCIDENTALES O CONSECUENCIALES RESULTADO DEL USO O INCAPACIDAD DE USO DEL "PROGRAMA" (INCLUYENDO DE MANERA ENUNCIATIVA MAS NO LIMITATIVA LA PÉRDIDA DE INFORMACIÓN, INEXACTITUD EN LA INFORMACIÓN, PÉRDIDAS

SUFRIDAS POR EL USUARIO DEL "PROGRAMA" O POR TERCERAS PERSONAS O LA INCAPACIDAD DEL "PROGRAMA" PARA OPERAR CON OTROS PROGRAMAS), AUN CUANDO DICHO TITULAR O CUALQUIER OTRA PERSONA HAYA ADVERTIDO DICHA POSIBILIDAD DE DAÑO.

FIN DE LOS TÉRMINOS Y CONDICIONES

Cómo aplicar estos términos a los nuevos "programas"

En el caso en el que se esté desarrollando un "Programa" nuevo y se tenga la intención de hacerlo de uso público, la mejor forma de lograrlo es haciéndolo Libre, y de esta forma, permitir a cualquiera redistribuirlo y cambiarlo bajo los términos y condiciones de esta Licencia.

A fin de lograr lo anterior, se deben incluir los siguientes avisos al "Programa". Es más seguro incluir dichos avisos al principio de cada archivo de código fuente para aclarar de manera más eficiente la exclusión de garantía mencionada. Asimismo, cada archivo debe tener por lo menos la frase "Derechos Reservados" (Copyright para los países con ese sistema) relativa a los derechos de autor, así como la referencia al lugar donde se encuentre la leyenda y especificaciones completas de los mismos.

La referencia al nombre del "Programa" y a una idea de lo que el mismo hace.

```
Derechos Reservados © yyyy nombre del autor

Este es un Software Libre; como tal redistribuirlo y/o
modificarlo está permitido, siempre y cuando se haga
bajo los términos y condiciones de la Licencia Pública
General GNU publicada por la Free Software Foundation,
ya sea en su versión 2 ó cualquier otra de las
posteriores a la misma.

Este "Programa" se distribuye con la intención de que
sea útil, sin embargo carece de garantía, ni siquiera
tiene la garantía implícita de tipo comercial o
inherente al propósito del mismo "Programa". Ver la
Licencia Pública General GNU para más detalles.

Se debe haber recibido una copia de la Licencia
Pública General GNU con este "Programa", si este no
fue el caso, favor de escribir a la Free Software
Foundation, Inc., 59 Temple Place – Suite 330, Boston,
MA 02111-1307, USA.
```

Asimismo, se deben incluir las direcciones de correo electrónico y convencional del autor del "Programa" a fin de contactarlo.

En el caso en el que el "Programa" sea interactivo se debe incluir un aviso cuando inicie el modo interactivo como el siguiente:

```
Gnomovision versión 69, Derechos Reservados © primer
año de publicación y nombre del autor
```

> Gnomovision carece totalmente de garantía; para más detalles teclee “show w”. Este es Software Libre y está permitido redistribuirlo bajo ciertas condiciones; teclee “show c” para más detalles.

Los comandos hipotéticos “show w” y “show c” deberán desplegar las partes correspondientes de la Licencia Pública General. Obviamente, los comandos que se utilicen pueden ser distintos a los mencionados, inclusive pueden ser clicks del ratón o elementos del menú, según sea más conveniente.

Asimismo, el autor del “Programa” debe obtener de su empleador (en caso en que dicho autor trabaje como programador) o de su escuela, si ese es el caso, una renuncia firmada a los derechos de autor por el “Programa”, si es que fuera necesario. El siguiente es un ejemplo:

> Yoyodyne, Inc., por medio de la presente se renuncia a cualquier derecho de autor que corresponda al programa “Gnomovision” cuyo autor es James Hacker.
>
> Firma Ty Coon, Presidente de Vice, 1 de Abril de 1989.

Esta Licencia Pública General prohíbe incorporar el “Programa” a programas propietarios. En el caso en el que se trate de un programa que a su vez sea una librería de subrutinas, es más conveniente permitir ligas de aplicaciones propietarias con la librería. Si es esto lo que se desea, entonces se debe usar la Licencia Pública General Menor de GNU, en lugar de esta licencia.

Fin