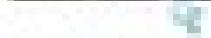


# Programación Java JDBC y Swing



www.ra-ma.com



División especializada en libros de programación y desarrollo profesional.

Francisco Blasco

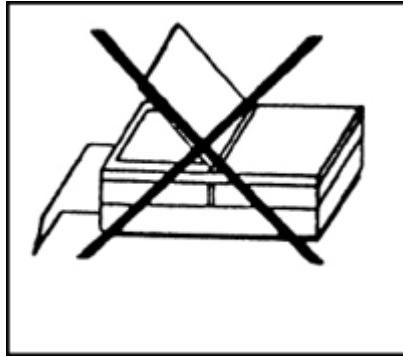


Ra-Ma®

# **Programación Java: JDBC y Swing**

*Francisco Blasco*





La ley prohíbe  
fotocopiar este libro

Programación Java: JDBC y Swing

© Francisco Blasco

© De la edición: Ra-Ma 2020

MARCAS COMERCIALES. Las designaciones utilizadas por las empresas para distinguir sus productos (hardware, software, sistemas operativos, etc.) suelen ser marcas registradas. RA-MA ha intentado a lo largo de este libro distinguir las marcas comerciales de los términos descriptivos, siguiendo el estilo que utiliza el fabricante, sin intención de infringir la marca y solo en beneficio del propietario de la misma. Los datos de los ejemplos y pantallas son ficticios a no ser que se especifique lo contrario.

RA-MA es marca comercial registrada.

Se ha puesto el máximo empeño en ofrecer al lector una información completa y precisa. Sin embargo, RA-MA Editorial no asume ninguna responsabilidad derivada de su uso ni tampoco de cualquier violación de patentes ni otros derechos de terceras partes que pudieran ocurrir. Esta publicación tiene por objeto proporcionar unos conocimientos precisos y acreditados sobre el tema tratado. Su venta no supone para el editor ninguna forma de asistencia legal, administrativa o de ningún otro tipo. En caso de precisarse asesoría legal u otra forma de ayuda experta, deben buscarse los servicios de un profesional competente.

Reservados todos los derechos de publicación en cualquier idioma.

Según lo dispuesto en el Código Penal vigente, ninguna parte de este libro puede ser reproducida, grabada en sistema de almacenamiento o transmitida en forma alguna ni por cualquier procedimiento, ya sea electrónico, mecánico, reprográfico, magnético o cualquier otro sin autorización previa y por escrito de RA-MA; su contenido está protegido por la ley vigente, que establece penas de prisión y/o multas a quienes, intencionadamente, reprodujeren o plagiaren, en todo o en parte, una obra literaria, artística o científica.

Editado por:

RA-MA Editorial

Calle Jarama, 3A, Polígono Industrial Igarsa  
28860 PARACUELLOS DE JARAMA, Madrid

Teléfono: 91 658 42 80

Fax: 91 662 81 39

Correo electrónico: *editorial@ra-ma.com*

Internet: *www.ra-ma.es* y *www.ra-ma.com*

ISBN: 978-84-9964-952-8

Depósito legal: M-13455-2020

Maquetación: Antonio García Tomé

Diseño de portada: Antonio García Tomé

Filmación e impresión: Safekat

Impreso en España en mayo de 2020

*A mi familia*

# Índice

- Prólogo
- Introducción
- Ejercicios JDBC
  - Modelo de desarrollo de software arquitectura a tres capas
  - Aplicación Ejercicio JDBC9
    - Insertar
    - Consultar todos
    - Consultar por identificador de libro
    - Transacción
    - Control de las reglas de negocio
  - Pool de conexiones
  - Aplicación PoolConexiones
  - Generalidades
    - InterfazGrafica1
    - InterfazGrafica2
    - InterfazGrafica3
  - Aplicación InterfazGrafica1
  - Aplicación InterfazGrafica2
  - Aplicación InterfazGrafica3
  - Aplicación InterfazGrafica4
  - Aplicación InterfazGrafica5
  - Aplicación InterfazGrafica6

[Aplicación InterfazGrafica7](#)  
[Aplicación InterfazGrafica8](#)  
[Aplicación InterfazGrafica9](#)  
[Aplicación InterfazGrafica10](#)  
[Aplicación InterfazGrafica11](#)  
[Aplicación InterfazGrafica12](#)  
[Aplicación MenuJTabbedPane](#)  
[Aplicación MenuJMenuBar](#)  
[Aplicación GestiónEventosFecha1](#)  
    Definición del evento  
    Definición del Listener  
    Definición de clases escucha  
    Definición del componente lanzador  
    del evento  
    Registro con clase escucha  
    Invocación a método susceptible de  
    lanzar el evento  
[Aplicación GestiónEventosFecha2](#)  
    Definición del evento  
    Definición del Listener  
    Definición de clase escucha  
    Definición del componente lanzador  
    del evento  
[Aplicación PredicciónMeteorológicaJTree](#)  
[Aplicación VotaciónPropuesta](#)  
    Abrir nueva votación

Registrar votación  
Leer resultado de votación  
previamente registrado en la base de  
datos

JTable  
Model  
Listener  
Acceso a Base de Datos  
Fichero repositorio.xml  
Gestión de incidencias  
Generación de ficheros PDF  
como documentos de salida

Aplicación GestiónLibros

Aspectos globales de la aplicación

OPCIÓN: Conexion  
OPCIÓN: Vista Formulario  
OPCIÓN: VistaUnicaTabla  
OPCIÓN: VistaPaginadaTabla  
OPCIÓN: VistaArbol  
OPCIÓN: Configurar Documento -  
Impresión  
OPCIÓN: Gestión Incidencias

JDBC

A1.1 Interfaces JDBC y otros  
aspectos contemplados en las  
aplicaciones

## Swing

A2. 1 métodos de clases Swing

A2.2 JTable

A2.3 JTree

A2.4 otras clases

A2.5 eventos

A2.5.1 Interfaces oyentes y métodos

A2.5.2 Componentes Swing y eventos que pueden lanzar

A2.5.3 Clases Swing y AWT utilizadas en las aplicaciones

A2.5.4 Listeners registrados en las aplicaciones

## Scripts SQL

material adicional

# PRÓLOGO

El objetivo de este libro no es ser un “áspero” manual de referencia de los contenidos tratados: JDBC y Swing, Tampoco se trata, como podría dar la impresión, de una mera colección de ejercicios. Sino que el planteamiento del autor en la redacción de esta obra técnica ha sido el presentar al lector una integración de la utilización de ambas APIs, contextualizada en un ámbito productivo, materializada en el ejemplo final GestiónLibros. El resto del libro constituye un camino de aproximación enfocada a la comprensión progresiva de dicha aplicación. Todo ello ajustado a la estructura que representa el modelo de desarrollo de software arquitectura a tres capas, que proporciona un marco organizativo de los componentes intervenientes.

Se ha acometido el desarrollo de interfaces gráficas desde un enfoque puro del código Java utilizado. El lector puede pensar que existen, en la mayor parte de IDEs, utilidades que permiten el desarrollo gráfico de una interfaz, generando automáticamente el código Java correspondiente a dicha implementación. Obviamente, es un método mucho más amigable y rápido que el que supone “bajar” a nivel de código, y desarrollar la interfaz “a golpe de tecla”. Somos conscientes de ello. El objetivo de este libro consiste en proporcionar al lector la cualificación necesaria para desarrollar este tipo de interfaces prescindiendo totalmente de este tipo de utilidades “rápidas”. Estaremos totalmente de acuerdo en que, desarrollando aplicaciones en el ámbito productivo, recurriremos al procedimiento “rápido” en la mayor parte de ocasiones. Pero nos encontramos, en situaciones puntuales, en las que el código generado automáticamente por el IDE al utilizar el procedimiento gráfico, no se adapta a los

requerimientos exigidos a la aplicación. En esas situaciones nos vemos obligados a “sumergirnos” en el código y aplicar los ajustes necesarios. Para “sumergirnos” en el código se requiere “ser un buen nadador” (siguiendo con el símil). Es ahí, exactamente, donde entra el planteamiento de este libro, en proporcionar la cualificación necesaria para poder ser resolutivos y “salir airosos” de la situación, sin mayores complicaciones. Conociendo en profundidad los mecanismos inherentes a las interfaces gráficas, siempre estamos en condiciones de recurrir al procedimiento gráfico.

Se aportan esquemas y gráficos en un intento de facilitar al lector la comprensión de mecanismos que pudiesen estar revestidos de cierta complejidad.

La lógica de negocio asociada a aplicación final GestiónLibros, está basada en desarrollar un CRUD aplicado al mantenimiento de una tabla de una Base de Datos. A primera vista, el lector, puede tener la impresión de que las subopciones de la opción **Mantenimiento** del menú de dicha aplicación son redundantes; pero pensemos que la aplicación no es un fin en sí misma, sino un medio para poder disponer de un escenario en que poder aplicar de forma integral todos los contenidos, mecanismos y estrategias objeto de estudio y tratamiento en este libro.

En pro de minimizar la extensión del libro, se han reducido a la mínima expresión los comentarios en el código de las clases de las aplicaciones aportadas, incluso en la mayoría de los casos, son inexistentes. Así mismo, también se han reducido otros recursos que proporcionan legibilidad al programa, tales como líneas en blanco adicionales. La ausencia de comentarios en el código, es suplida por las debidas explicaciones en el tema relacionadas con dicho código. Otra circunstancia, de la que el

lector puede tener la impresión de ser poco ortodoxa, es el hecho de que la mayor parte de ejercicios del primer bloque (JDBC) están totalmente exentos de modularidad. Se ha hecho de esta manera en un intento de “concentrar” y minimizar las actuaciones esenciales relacionadas con la funcionalidad que en cada momento se estaba tratando. Dicha falta de ortodoxia es compensada en el ejemplo EjercicioJDBC9, aplicación en la que se integran todas las funcionalidades tratadas individualmente en ejercicios anteriores, y se estructuran atendiendo a los criterios organizativos del modelo de desarrollo de software arquitectura a tres capas. El objetivo principal de esta aplicación es aportar al lector las directrices a aplicar en la estructuración de una aplicación a tres capas.

# Bloque 1º

## JDBC

# 1

## INTRODUCCIÓN

Java DataBase Connectivity (JDBC) es el API que permite el acceso desde una aplicación Java a Sistemas Gestores de Bases de Datos Relacionales. Es el cometido que nos hemos planteado en esta primera parte del libro. No se pretende convertir este libro en un mero manual de referencia, sino exponer al lector los mecanismos fundamentales que debe aplicar el programador con los recursos que nos ofrece este API. Todo ello ilustrado con los debidos ejemplos. El material descargable contiene toda una serie de ejercicios que brindan al lector la oportunidad de comprobar los citados mecanismos.

Todos los ejercicios de este libro que conectan con Base de Datos, están preparados (y probada exhaustivamente su ejecución libre de errores) para hacerlo con los Gestores de Bases de Datos Relacionales MySQL y Oracle. Nos encontramos con tres posibles variantes en cuando al procedimiento a utilizar para establecer el Gestor de Base de Datos concreto con que el lector pretende probar la aplicación:

- Aplicaciones cuyo nombre responde al formato **EjercicioJDBCn** siendo **n** el número de ejercicio de la serie. El lector no tiene más que dejar activadas las líneas

```
Class.forName( ... );
Connection connection =
```

```
DriverManager.getConnection( ... );
```

correspondientes al Gestor de BD a utilizar, y comentarizar las que corresponden a la otra BD. En EjercicioJDBC9, estructurado según el modelo de desarrollo de software arquitectura a tres capas, el lector deberá aplicar dicha actuación en la clase **ConexionBaseDatos** del package **datos**.

- Aplicación **PoolConexiones**. Es necesario cambiar la comentarización de una zona más extensa de código en la clase **ConexionBaseDatos**.
- Aplicaciones **VotacionPropuesta** y **GestionLibros**. Forman parte del bloque de ejercicios de este libro destinados a Swing. Aunque estructurados a tres capas, para establecer la Base de Datos con que conectar, el lector debe editar el fichero **repositorio.xml**, ubicado en el directorio **xml** del proyecto. Este fichero actúa como repositorio centralizado de información del proyecto susceptible de ser parametrizada. Mediante la edición del mencionado fichero podrá establecer el SGBD escogido:

```
<BASE_DATOS_SELECCIONADA>MySQL</BASE_DATOS_SELECCIONADA>
```

o bien

```
<BASE_DATOS_SELECCIONADA> Oracle  
</BASE_DATOS_SELECCIONADA>
```

Toda aplicación Java en que se pretenda conectar con una Base de Datos utiliza el driver correspondiente a dicha BD. Dichos drivers responden a unas librerías externas que deben ser añadidas al proyecto. Los drivers utilizados y probados en todos los ejercicios de este libro que conectan con Base de Datos son:

- Para MySQL: **mysql-connector-java-5.1.22-bin.jar**
- Para Oracle: **ojdbc6.jar**

Por cuestiones de licencia, solamente podemos aportar el primer fichero. Está añadido en todos los proyectos, y presente en el directorio **lib** de cada uno de ellos. Para poder conectar con Oracle, el lector deberá efectuar la descarga del fichero citado, y añadirlo al proyecto al igual que lo está el correspondiente a MySQL. Hemos podido aportar los drivers para MySQL amparándonos en:

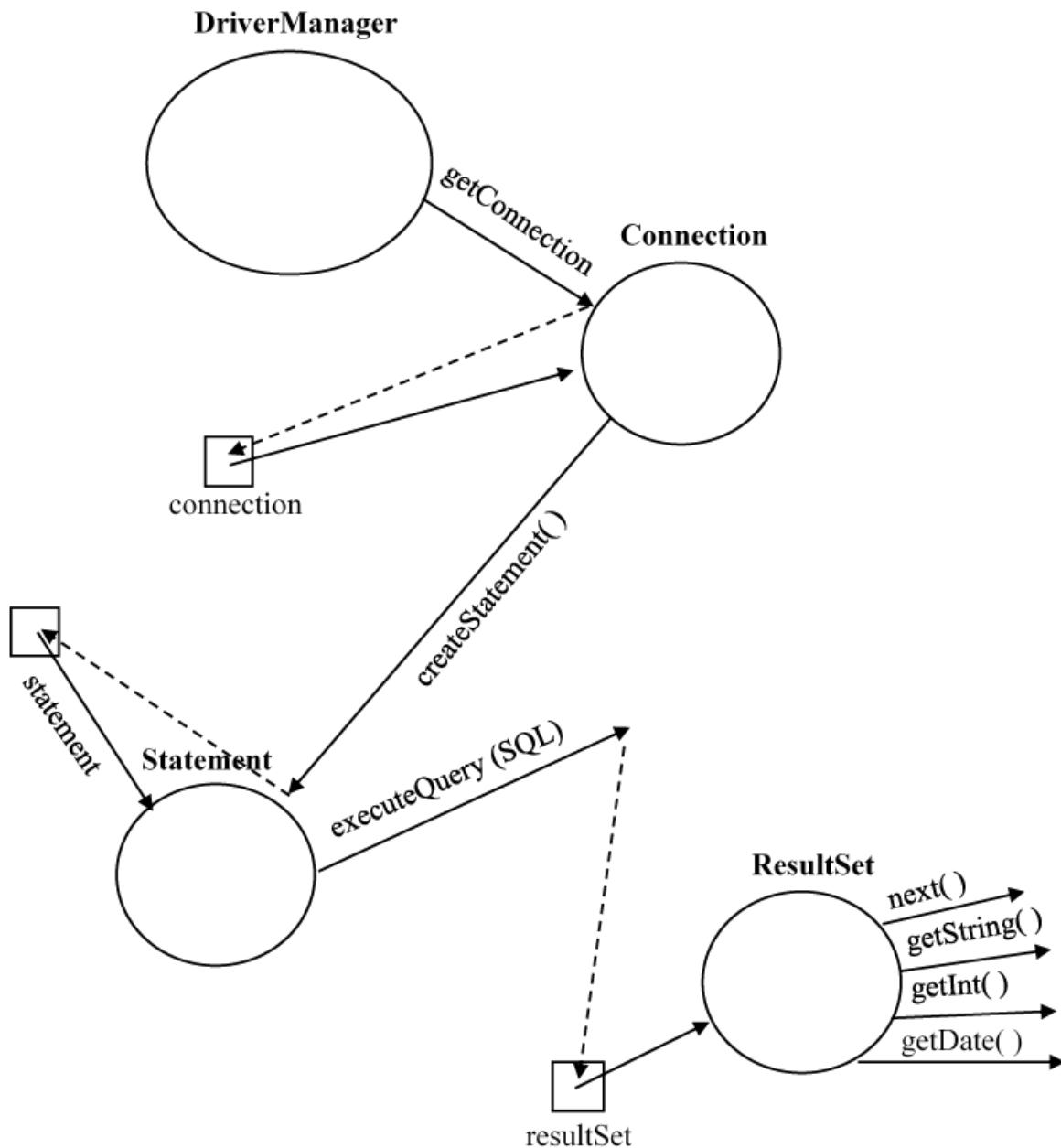
*“You may copy and distribute verbatim copies of the Program’s source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.”*

Sirva la citada reseña para cumplir con todos los requisitos legales.

Con el material descargable del curso, además del código de todas las aplicaciones, se aportan también scripts SQL que “recrean”, en ambos Sistemas Gestores de Bases de Datos, el escenario necesario (tablas, datos, secuencias, procedimientos almacenados) para la ejecución de las aplicaciones que conectan con dichos SGBD. Solamente la aplicación **VotacionPropuesta** requiere de la creación previa de la Base de Datos **VOTACIONES**. El resto de las aplicaciones acceden a la Base de Datos **BIBLIOTECA**. El lector podrá optar por probar la ejecución de estas aplicaciones en MySQL, en Oracle, o en los

dos. Para ello deberá tener instalado el SGBD que escoja, iniciar los servicios, y ejecutar los correspondientes scripts SQL.

Procedamos ya a abordar todos los detalles pertinentes para el acceso a la Base de Datos desde una aplicación Java. El siguiente esquema responde a una representación gráfica de las clases que intervienen en dicho proceso:



La clase que responde a la interface *Connection* es la que modela la conexión con la BD. Nos expresamos de esta forma porque en el API JDBC, salvo alguna excepción, nos encontramos con interfaces, y no con clases. El sentido que ello tiene es imponer, a cada “fabricante-desarrollador-distribuidor” de Gestores de Bases de Datos que pretendan aportar los correspondientes drivers para permitir el acceso desde Java, la obligación de que dichos drivers contengan unas clases con unos métodos, con unos nombres concretos, y que se comporten de determinada forma. Estamos asistiendo a un escenario polimórfico a gran escala. Presentamos a continuación el primer ejemplo, y con ello iremos analizando progresivamente cada una de las interfaces intervenientes, así como sus respectivos métodos.

# 2

## EJERCICIOS JDBC

### EjercicioJDBC1

```
package ejerciciojdbc1;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.text.SimpleDateFormat;
public class EjercicioJDBC1 {
public static void main(String[] args) {
try {
// CONEXION MySQL
Class.forName("com.mysql.jdbc.Driver");
Connection connection =
DriverManager.getConnection("jdbc:mysql://localhost:33
06/biblioteca", "root", "");
/*
// CONEXION Oracle
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection connection =
DriverManager.getConnection("jdbc:oracle:thin:@localho
st:1521:ORCL", "scott", "tiger");
*/
String sql = "SELECT * FROM libros";
Statement statement = connection.createStatement();
ResultSet resultSet = statement.executeQuery(sql);
while (resultSet.next()) {
System.out.println("id_libro : " +
```

```
resultSet.getString(1));
System.out.println("título : " +
resultSet.getString(2));
System.out.println("género : " +
resultSet.getString(3));
System.out.println("fecha edición : " + new
SimpleDateFormat("yyyy-MM-
dd").format(resultSet.getDate(4)));
System.out.println("numero páginas : " +
resultSet.getInt(5));
System.out.println("premiado: " +
resultSet.getInt(6));
System.out.println("-----");
}
connection.close();
} catch(Exception e){
e.printStackTrace();
}
}
}
```

Observará el lector, más que la falta, la total ausencia de modularidad en la implementación del código. Así como la total ausencia de recursos que proporcionan legibilidad al programa, tales como comentarios, líneas en blanco, etc. Todo ello no es consecuencia de una actitud descuidada y desorganizada. En los primeros ocho ejercicios de este bloque se ha ubicado la totalidad del código en el método *main()*. Tiempo habrá en ejercicios posteriores del bloque Swing, y sobre todo los organizados a tres capas, para aplicar modularidad, estructura y organización. Se ha hecho de esta forma en estos primeros ejercicios para que el lector disponga, en pro de la claridad de comprensión, de una perspectiva lo más concentrada y escueta posible del código utilizado. En lo relativo a la ausencia de recursos que

proporcionan legibilidad, se ha hecho con la intención de reducir al máximo la extensión del libro. La ausencia de comentarios en el código esperemos que sea compensada con las aclaraciones y explicaciones que efectúa el autor en el análisis de cada aplicación.

Para obtener una conexión, la primera actuación es:

```
Class.forName("com.mysql.jdbc.Driver");
```

en que proporcionamos al método *forName()* como parámetro un *String* con la clase del driver y la ruta en que está ubicado dentro del archivo-librería aportado. Si el lector no está familiarizado con la clase *Class* y sus métodos le remitimos al libro, del mismo autor, “Programación Orientada a Objetos en Java”, concretamente en el capítulo dedicado al polimorfismo.

Es habitual en el desarrollo de los drivers, la implementación de un bloque *static* en la clase correspondiente al driver. Ello permite que, al cargarse la clase, se ejecute el código de dicho bloque. Las actuaciones que se suelen implementar en el bloque *static* provocan que la clase se instancie a sí misma, y a continuación, la referencia obtenida de dicha instanciación se transfiera al método:

```
DriverManager.registerDriver(referenciaAInstanciaDelObreroDriver);
```

que ocasiona que el driver se registre con el administrador de controladores. Una vez registrado, podemos aplicar la siguiente actuación:

```
Connection connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/biblioteca", "root", "");
```

Proporcionamos como primer parámetro al método *getConnection()* una URL, y como segundo y tercer parámetro,

usuario y contraseña, respectivamente. El administrador de controladores busca de entre todos los drivers que tenga registrados, aquel en que encaje dicha URL, y lo utiliza para solicitar una conexión con la Base de Datos cuya dirección, puerto, y nombre de instancia se especifican. De conseguirla, modela dicha conexión en una instancia que se acoge a la interface *Connection*, tal y como aparece reflejado en la representación gráfica.

Antes de proseguir en el análisis minucioso del código utilizado en este ejemplo, focalizamos la atención del lector en el hecho de que no utilizamos el operador *new* para obtener ninguno de los objetos que intervienen en el acceso a la Base de Datos para conectar, lanzar la consulta, y gestionar los resultados. Es decir, no obtenemos los objetos por instanciación explícita mediante el operador **new**, sino que se produce una instanciación implícita de los mismos durante la ejecución de los métodos pertinentes, devolviendo éstos la referencia a dichos objetos, y limitándose el programador a recogerla en la variable declarada a tal efecto. Esta circunstancia es reflejada gráficamente mediante una flecha discontinua como continuación a la flecha continua que viene a representar la ejecución del método. El escenario descrito permite reforzar nuestras manifestaciones en relación a que declaramos las referencias del tipo de una interface y no de una clase.

El paso siguiente, y previo al de poder lanzar la consulta SQL a la BD, es obtener un objeto acogido a la interface *Statement*:

```
Statement statement = connection.createStatement();
```

Lanzaremos las SQL a la BD mediante los métodos *executeXXX()* de este objeto. Cuando lancemos una consulta, deberemos utilizar *executeQuery()*, pues este método devuelve la

referencia a un objeto que responde a la interface `ResultSet` que nos permite procesar el resultado devuelto por la consulta:

```
ResultSet resultSet = statement.executeQuery(sql);
```

De todas las variantes de proceso de dicho resultado, nos vamos a centrar en la implementada en este ejemplo, que será la que adaptaremos cuando en **Ejercicio JDBC9** presentemos un ejemplo organizado según el modelo de desarrollo de software arquitectura a tres capas. Los mecanismos a utilizar implican la utilización de los métodos:

- `next()`
- `getXXX()`

Ante el lanzamiento de una consulta, en relación al número de filas devueltas, nos podemos encontrar con los siguientes casos:

- Ninguna fila.
- Una sola fila. Circunstancia que se produce cuando se trata de una consulta en que establecemos una condición de búsqueda por valor de clave primaria, o cuando la expresión a devolver es una función de grupo sin utilizar la cláusula GROUP BY.
- Un conjunto de filas.

El *ResultSet* (por economía lingüística utilizaremos este modo de referirnos a un objeto que responde a una interface) dispone de un cursor. Para el proceso de los datos devueltos, y en previsión de cualquiera de los tres escenarios descritos, el posicionamiento del cursor es inicializado antes de la primera fila. El recorrido del conjunto de filas modelado por el *ResultSet* pasa,

inxorablemente, por establecer un “recorrido” de dicho conjunto, donde, antes de tratar los datos de una fila, proceden dos actuaciones:

1. “Interrogar” al *ResultSet* sobre la potencial existencia de dicha fila.
2. Si dicha fila existe, avanza, posicionando el cursor sobre la misma.

El *ResultSet* dispone de un método que representa un “paquete” de ambas actuaciones, estamos hablando del ya mencionado método *next()*. Este método cumple con el primero de los cometidos devolviendo **true** si existe la fila siguiente, y **false** en caso contrario. Así pues, la estructura de control repetitiva que mejor se adapta es:

```
while (resultSet.next()) {  
    PROCESO DE LOS DATOS DE UNA FILA  
}
```

Cada repetición del bloque nos permitirá procesar los datos de una cada fila, es decir las expresiones devueltas por SELECT. “Extraeremos” cada una de dichas expresiones mediante el método *getXXX()* procedente en función del tipo SQL a que responda la expresión, teniendo el cuenta el mapeo establecido entre tipos SQL y clases Java. En el caso del ejemplo, la consulta lanzada a la BD ha sido:

```
SELECT * FROM libros
```

En consecuencia, hemos utilizado:

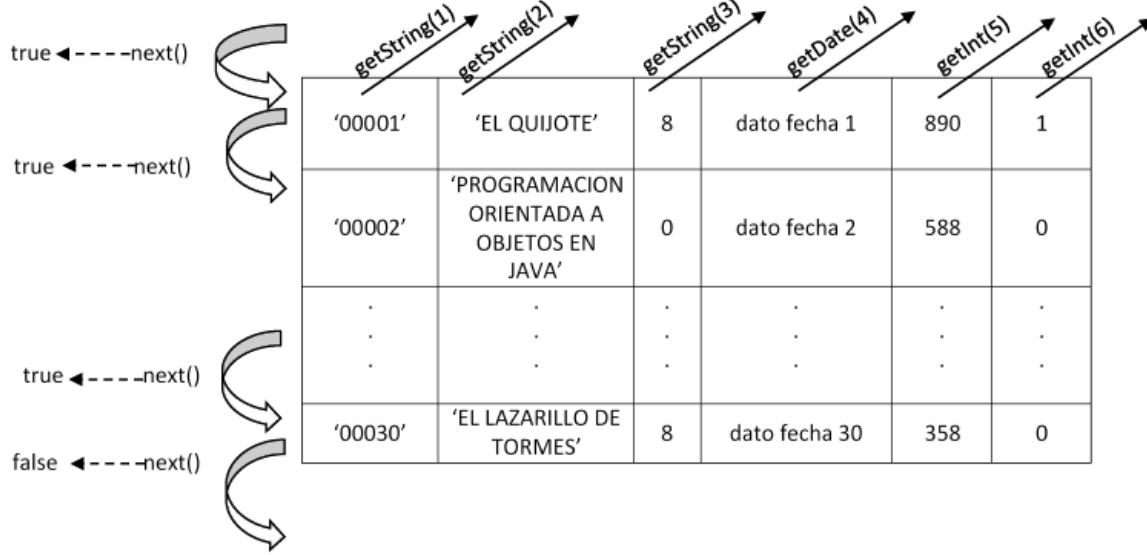
Expresión devuelta por la	Tipo SQL MySQL	Tipo SQL Oracle	Método
---------------------------	----------------	-----------------	--------

consulta			
id_libro	CHAR(5)	CHAR(5)	getString()
titulo	VARCHAR(60)	VARCHAR(60)	getString()
genero	CHAR(1)	CHAR(1)	getString()
fecha_edicion	DATETIME	DATE	getDate()
numero_paginas	INT	NUMBER	getInt()
premiado	INT	NUMBER	getInt()

Cada uno de los métodos *getXXX()* está sobrecargado, pudiendo ser el parámetro transferido:

- Un valor *int* que representa el número de posición en la SELECT de la expresión a “extraer”. El primer valor es el 1. Es la variante que utilizamos en todos los ejemplos de este libro.
- Un *String* que representa el nombre de la expresión en la SELECT, por ejemplo aplicado a la primera columna sería  
`getString("id_libro")`

El siguiente esquema representa el proceso de recorrido del *ResultSet* y extracción de los datos de cada fila



Una vez procesado el resultado de la consulta, procedemos a invocar al método

```
connection.close()
```

de la conexión que implica la liberación de los recursos utilizados por la misma.

Presentamos a continuación, otro ejercicio, en que se lanza a la BD la misma SQL. Pretendemos con este nuevo ejercicio presentar al alumno nuevos mecanismos para obtener la conexión con la BD.

## Ejercicio JDBC2

```
package ejerciciojdbc2;
import java.sql.Connection;
// import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.text.SimpleDateFormat;
import java.util.Properties;
public class EjercicioJDBC2 {
public static void main(String[] args) {
```

```
try {
Properties propertiesConexion = new Properties();
propertiesConexion.put("user", "root");
propertiesConexion.put("password", "");
propertiesConexion.put("charSet", "ISO-8859-1");
// CONEXION MySQL
// Class.forName("com.mysql.jdbc.Driver");
// Connection connection =
DriverManager.getConnection("jdbc:mysql://localhost:3306/biblioteca", propertiesConexion);
com.mysql.jdbc.Driver driver = new
com.mysql.jdbc.Driver();
Connection connection =
driver.connect("jdbc:mysql://localhost:3306/biblioteca",
", propertiesConexion);
/*
Properties propertiesConexion = new Properties();
propertiesConexion.put("user", "scott");
propertiesConexion.put("password", "tiger");
propertiesConexion.put("charSet", "ISO-8859-1");
// CONEXION Oracle
// Class.forName("oracle.jdbc.driver.OracleDriver");
// Connection connection =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:ORCL", propertiesConexion);
oracle.jdbc.driver.OracleDriver driver = new
oracle.jdbc.driver.OracleDriver();
Connection connection =
driver.connect("jdbc:oracle:thin:@localhost:1521:ORCL"
, propertiesConexion);
*/
String sql = "SELECT * FROM libros";
Statement statement = connection.createStatement();
ResultSet resultSet = statement.executeQuery(sql);
while (resultSet.next()) {
System.out.println("id_libro : " +
resultSet.getString(1));
```

```

System.out.println("título : " +
resultSet.getString(2));
System.out.println("género : " +
resultSet.getString(3));
System.out.println("fecha edición : " + new
SimpleDateFormat("yyyy-MM-
dd").format(resultSet.getDate(4)));
System.out.println("numero páginas : " +
resultSet.getInt(5));
System.out.println("premiado: " +
resultSet.getInt(6));
System.out.println("-----");
}
connection.close();
} catch(Exception e){
e.printStackTrace();
}
}
}

```

Obtenemos la conexión prescindiendo del administrador de controladores. Instanciamos explícitamente un objeto de la clase que corresponde al driver, para MySQL:

```

com.mysql.jdbc.Driver driver = new
com.mysql.jdbc.Driver();

```

y para Oracle:

```

oracle.jdbc.driver.OracleDriver driver = new
oracle.jdbc.driver.OracleDriver();

```

obteniendo la conexión mediante la invocación al método connect() del driver. Para MySQL:

```

Connection connection =
driver.connect("jdbc:mysql://localhost:3306/biblioteca
", propertiesConexion);

```

y para Oracle:

```
Connection connection =  
    driver.connect("jdbc:oracle:thin:@localhost:1521:ORCL"  
, propertiesConexion);
```

Observamos que al método *connect()* se le transfieren dos parámetros:

- La URL que permite acceder e identificar a la instancia de la BD.
- La referencia a un objeto de la clase *Properties* en que establecemos algunas características: usuario, contraseña, etc.

Aunque comentarizado, presentamos también en el ejercicio la posibilidad de obtener la conexión mediante *DriverManager*, pero con una variante sobrecargada del método *getConnection()* al que transferimos como segundo parámetro la referencia al objeto *Properties* ya descrito. Para MySQL:

```
Connection connection = DriverManager.getConnection  
("jdbc:mysql://localhost:3306/biblioteca",  
propertiesConexion);
```

y para Oracle:

```
Connection connection = DriverManager.getConnection  
("jdbc:oracle:thin:@localhost:1521:ORCL",  
propertiesConexion);
```

Para instrucciones SQL como la utilizada en los dos ejercicios previos, que no requieran de la aplicación de datos parametrizados se adaptan perfectamente objetos que respondan a la interface *Statement*. Pero en muchas ocasiones cuando nos encontramos con instrucciones SQL que sí requieran de la

aportación de datos parametrizados. El caso más obvio es `INSERT`, en que nos vemos obligados a proporcionar los valores a insertar, y que especificamos en la cláusula `VALUES`. Podríamos seguir utilizando `Statement`, pero nos veríamos obligados a concatenar al el `String` que conforma la SQL las variables que contienen dichos valores. Esta “incomodidad” puede ser subsanada utilizando `PreparedStatement`, que hereda de `Statement`. Presentamos la jerarquía en herencia correspondiente a esta última interface:



La utilización de `PreparedStatement` supone también una mejora en el rendimiento del Gestor de Base de Datos, pues cuando se lanza la SQL mediante clases de esta interface, se almacena la versión precompilada de la SQL, en previsión de que vuelva a utilizarse, evitando posteriores procesos de precompilación al lanzar nuevamente la misma SQL.

A efectos de ilustrar la utilización de `PreparedStatement`, presentamos un nuevo ejercicio en que la SQL que se lanza a la

BD es una consulta en que se intenta recuperar una fila estableciendo en la cláusula WHERE la condición de coincidencia de clave primaria.

## Ejercicio JDBC3

```
package ejerciciojdbc3;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.text.SimpleDateFormat;
public class EjercicioJDBC3 {
public static void main(String[] args) {
String idLibroAConsultar = "00001"; // Mediante esta
asignación simulamos la captura del dato desde una
interfaz de usuario
try {
// CONEXION MySQL
Class.forName("com.mysql.jdbc.Driver");
Connection connection =
DriverManager.getConnection("jdbc:mysql://localhost:33
06/biblioteca", "root", "");
/*
// CONEXION Oracle
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection connection =
DriverManager.getConnection("jdbc:oracle:thin:@localho
st:1521:ORCL", "scott", "tiger");
*/
String sql = "SELECT * FROM libros WHERE id_libro =
CAST(? AS CHAR(5))";
PreparedStatement preparedStatement =
connection.prepareStatement(sql);
preparedStatement.setString(1, idLibroAConsultar);
ResultSet resultSet =
preparedStatement.executeQuery();
```

```

if (resultSet.next()) {
    System.out.println("id_libro : " +
        resultSet.getString(1));
    System.out.println("título : " +
        resultSet.getString(2));
    System.out.println("género : " +
        resultSet.getString(3));
    System.out.println("fecha edición : " + new
        SimpleDateFormat("yyyy-MM-
        dd").format(resultSet.getDate(4)));
    System.out.println("numero páginas : " +
        resultSet.getInt(5));
    System.out.println("premiado: " +
        resultSet.getInt(6));
}
connection.close();
} catch(Exception e){
e.printStackTrace();
}
}
}

```

En primer lugar, hay que centrar la atención en la conformación del *String* que representa la SQL:

```

String sql = "SELECT * FROM libros WHERE id_libro =
CAST(? AS CHAR(5));"

```

Se observa que en el lugar donde ubicamos un literal cuando lanzamos directamente la SQL desde entornos especializados, aquí, dicho literal es sustituido por “?”, representando un parámetro que es necesario establecer.

A diferencia del *Statement*, al método que nos permite la obtención del *PreparedStatement*, se le transfiere el *String* que representa la SQL:

```

PreparedStatement preparedStatement =

```

```
connection.prepareStatement(sql);
```

En contraprestación, dicho String no se transferirá cuando invoquemos al método `executeXXX()`.

```
ResultSet resultSet =  
    preparedStatement.executeQuery();
```

Obviamente, antes de lanzar la SQL a la BD, es necesario establecerle los valores necesarios, representados mediante “?”, como ya hemos mencionado. Para ello recurrimos a los métodos `setXXX()`, en el caso de este ejercicio:

```
preparedStatement.setString(1, idLibroAConsultar);
```

A estos métodos se les transfieren dos parámetros:

- Un valor numérico que representa la posición del parámetro en la SQL, empezando por “1”.
- La expresión que corresponde al dato a establecer.

Análogamente a la selección del método `getXXX()` a aplicar en la “extracción” de datos del `ResultSet`, es necesario utilizar el método `setXXX()` pertinente, en función de la clase Java que mapea el tipo SQL del dato a establecer.

Un pequeño detalle a matizar es que en este ejercicio hemos aplicado la estructura alternativa simple en el “recorrido” del `ResultSet`, en lugar de la estructura repetitiva `while()` que se aplicó en los dos ejercicios anteriores. Nos hemos decantado por la alternativa simple en este ejercicio porque al tratarse de una consulta por coincidencia de clave primaria, solamente devolverá una fila como máximo, en el caso de devolver algún resultado.

A efectos de minimizar la extensión del código de los ejercicios expuestos, en todos ellos, a excepción de **EjercicioJDBC9**,

hemos simulado mediante asignaciones la recogida de datos desde una interfaz de usuario. En este ejercicio:

```
String idLibroAConsultar = "00001";
// Mediante esta asignación simulamos la captura del
dato desde una interfaz de usuario
```

Sugerimos al lector de que, en las pruebas de ejecución de dichos ejercicios, modifique los valores utilizados en dichas asignaciones.

En el siguiente ejercicio a presentar, volvemos a utilizar *PreparedStatement*, pero en este caso para el lanzamiento de *INSERT*.

## EjercicioJDBC4

```
package ejerciciojdbc4;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
public class EjercicioJDBC4 {
public static void main(String[] args) {
// Mediante las siguientes asignaciones simulamos la
captura de datos desde una interfaz de usuario
String idLibroAInsertar = "00501"; // Un valor de
clave primaria lo suficientemente alto para no
coincidir con los proporcionados por
// la secuencia las INSERT lanzadas para disponer de
datos de prueba en el correspondiente script SQL
// y por las que se se puedan lanzar desde la
aplicación GestiónLibros como consecuencia como
// consecuencia de la interacción con el usuario, en
este caso, el lector.
String tituloAInsertar = "LA TIA TULA";
String generoAInsertar = "8";
java.sql.Date fechaEdicionAInsertar =
java.sql.Date.valueOf("1972-11-05");
```

```

int numeroPaginasAInsertar = 570;
byte premiadoAInsertar = 0;
try {
// CONEXION MySQL
Class.forName("com.mysql.jdbc.Driver");
Connection connection =
DriverManager.getConnection("jdbc:mysql://localhost:33
06/biblioteca", "root", "");
/*
// CONEXION Oracle
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection connection =
DriverManager.getConnection("jdbc:oracle:thin:@localho
st:1521:ORCL", "scott", "tiger");
*/
String sql = "INSERT INTO libros VALUES(?, ?, ?, ?, ?, ?, ?)";
PreparedStatement preparedStatement =
connection.prepareStatement(sql);
preparedStatement.setString(1, idLibroAInsertar);
preparedStatement.setString(2, tituloAInsertar);
preparedStatement.setString(3, generoAInsertar);
preparedStatement.setDate(4, fechaEdicionAInsertar);
preparedStatement.setInt(5, numeroPaginasAInsertar);
preparedStatement.setByte(6, premiadoAInsertar);
int numeroFilasInsertadas =
preparedStatement.executeUpdate();
System.out.println("Se han insertado " +
numeroFilasInsertadas + " filas");
connection.close();
} catch(Exception e){
e.printStackTrace();
}
}
}
}

```

La SQL lanzada a la BD en este ejercicio:

```
String sql = "INSERT INTO libros VALUES(?, ?, ?, ?, ?, ?, ?);
```

Se han utilizado tantos parámetros representados por “?” como datos a contemplar en la cláusula VALUES. En consecuencia, aparecen tantos métodos *setXXX()* como parámetros hay, cada uno de ellos apropiado al tipo SQL.

Una novedad importante es que, por tratarse la SQL de una DML de actualización, hemos utilizado, para lanzar la SQL a la BD, el método *executeUpdate()*. En ejercicios anteriores utilizamos, en su lugar *executeQuery()*, que nos devolvía en ResultSet modelando el conjunto de datos devueltos por las consultas. El patrón de lanzamiento de SQL presentado en este ejercicio sería aplicable a las otras DML de actualización (UPDATE y DELETE). Dejamos de la mano del lector el desarrollar ejercicios análogos al presente en que se lancen las SQL citadas.

En el siguiente ejercicio que vamos a exponer, pretendemos mostrar al lector la posibilidad de poder lanzar instrucciones DDL desde una aplicación Java mediante JDBC.

## EjercicioJDBC5

```
package ejerciciojdbc5;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
public class EjercicioJDBC5 {
    public static void main(String[] args) {
        try {
            // CONEXION MySQL
            Class.forName("com.mysql.jdbc.Driver");
            Connection connection =
                DriverManager.getConnection("jdbc:mysql://localhost:33
06/biblioteca", "root", "");
            String sql = "CREATE TABLE clientes " +
            "(" +
            " dni CHAR(10), " +
```

```

" nombre VARCHAR(40), " +
" prioridad INT, " +
" saldo DOUBLE, " +
" CONSTRAINT pk_clientes PRIMARY KEY(dni) " +
") ";
/*
// CONEXION Oracle
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection connection =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:ORCL", "scott", "tiger");
String sql = "CREATE TABLE clientes " +
"( " +
" dni CHAR(10), " +
" nombre VARCHAR(40), " +
" prioridad NUMBER, " +
" saldo NUMBER, " +
" CONSTRAINT pk_clientes PRIMARY KEY(dni) " +
") ";
*/
Statement statement = connection.createStatement();
statement.executeUpdate(sql);
connection.close();
} catch(Exception e){
e.printStackTrace();
}
}
}
}

```

La ausencia de parámetros nos ha permitido utilizar *Statement*, desde el que hemos invocado a *executeUpdate()* para lanzar la SQL. Hemos aplicado CRATE TABLE como ejemplo representativo de cualquier DDL. La diferencia de tipos SQL disponibles en los dos SGBD contemplados nos obliga a plantear una SQL específica para cada uno de ellos. El nombre de la tabla a crear forma parte del literal *String*. Perfectamente podríamos

haber dispuesto la totalidad o parte del nombre de la tabla en el valor de una variable, que hubiésemos concatenado en la conformación del *String* en el lugar que le corresponde.

Seguimos avanzando en la presentación de las posibilidades que nos ofrece JDBC. Abordamos a continuación dos ejemplos en que accedemos a procedimientos almacenados. El primero de ellos invoca a una PROCEDURE. En primer lugar, exponemos dicha PROCEDURE en ambos SGBD, seguido del ejercicio Java que la invoca.

#### **MySQL - PROCEDURE consulta\_titulo\_pro**

```
CREATE PROCEDURE consulta_titulo_pro(IN  
id_libro_recibido CHAR(5), OUT titulo_encontrado  
VARCHAR(60))  
BEGIN  
SELECT titulo INTO titulo_encontrado FROM libros WHERE  
id_libro = id_libro_recibido;  
END//
```

#### **Oracle - PROCEDURE consulta\_titulo\_pro**

```
CREATE OR REPLACE PROCEDURE consulta_titulo_pro  
(id_libro_recibido IN CHAR, titulo_encontrado OUT  
VARCHAR2)  
AS retorna VARCHAR2(60);  
BEGIN  
SELECT titulo INTO retorna FROM libros WHERE id_libro  
= id_libro_recibido;  
titulo_encontrado := retorna;  
END consulta_titulo_pro;  
/
```

#### **EjercicioJDBC6**

```
package ejerciciojdbc6;
```

```
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Types;
public class EjercicioJDBC6 {
public static void main(String[] args) {
try {
// CONEXION MySQL
Class.forName("com.mysql.jdbc.Driver");
Connection connection =
DriverManager.getConnection("jdbc:mysql://localhost:33
06/biblioteca", "root", "");
/*
// CONEXION Oracle
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection connection =
DriverManager.getConnection("jdbc:oracle:thin:@localho
st:1521:ORCL", "scott", "tiger");
*/
String idLibroBuscado = "00002";
CallableStatement callableStatement =
connection.prepareCall("{CALL
consulta_titulo_pro(?,?)}");
callableStatement.registerOutParameter(2,
Types.VARCHAR);
callableStatement.setString(1, idLibroBuscado);
callableStatement.executeUpdate();
System.out.println("título encontrado: " +
callableStatement.getString(2));
connection.close();
} catch(Exception e){
e.printStackTrace();
}
}
}
```

Hace su aparición en este ejercicio el *CallableStatement* que ubicamos en la jerarquía asociada a *Statement* expuesta con anterioridad:

```
CallableStatement callableStatement =  
connection.prepareCall("{CALL  
consulta_titulo_pro(?,?)}");
```

Focalizamos la atención en que el contenido del String transferido como argumento al método *prepareCall()* se encuentra ubicado entre {}. Son estos los delimitadores que utilizamos para las secuencias de escape. El presente constituye uno de los casos. Dichos delimitadores fuerzan a que su contenido, o parte de él, reciba una interpretación especial. Seguimos constatando, como herencia de *PreparedStatement*, la presencia de "?" representando a los parámetros, que en este caso pueden ser de entrada o de salida. Las actuaciones necesarias para la invocación en la BD del procedimiento almacenado son:

1. Registrar el tipo del parámetro de salida:

```
callableStatement.registerOutParameter(2,  
Types.VARCHAR);
```

El primer argumento de este método hace referencia a la posición del parámetro de salida en cuestión en el conjunto de los que conforman los del procedimiento almacenado. En el segundo argumento especificamos su tipo SQL.

2. Establecer los valores que tendrán los parámetros de entrada, actuación totalmente análoga a la del *PreparedStatement*:

```
callableStatement.setString(1, idLibroBuscado);
```

3. Solicitar al SGBD la ejecución del procedimiento almacenado, idénticamente a como actuamos en sus

ascendientes de la jerarquía *Statement*:

```
callableStatement.executeUpdate();
```

4. Recoger los valores devueltos mediante los parámetros de salida:

```
callableStatement.getString(2)
```

actuando con los métodos *getXXX()* de forma similar a como lo hacíamos con el *ResultSet*, pero ahora desde el *CallableStatement*.

Procedemos seguidamente, de forma análoga a como hemos hecho en este ejercicio, pero ahora con otro, que, en lugar de invocar la ejecución en el SGBD de una PROCEDURE, se invoca a una FUNCTION.

### **MySQL - FUNCTION consulta\_titulo\_fun**

```
CREATE FUNCTION consulta_titulo_fun(id_libro_recibido
CHAR(5)) RETURNS VARCHAR(60)
BEGIN
DECLARE titulo_encontrado VARCHAR(60);
SELECT titulo INTO titulo_encontrado FROM libros WHERE
id_libro = id_libro_recibido;
RETURN titulo_encontrado;
END//
```

### **Oracle - FUNCTION consulta\_titulo\_fun**

```
CREATE OR REPLACE FUNCTION
consulta_titulo_fun(id_libro_recibido CHAR)
RETURN VARCHAR2
AS retorna VARCHAR2(60);
BEGIN
SELECT titulo INTO retorna FROM libros WHERE id_libro
= id_libro_recibido;
```

```
RETURN retorna;
END consulta_titulo_fun;
/
```

## EjercicioJDBC7

```
package ejerciciojdbc7;
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Types;
public class EjercicioJDBC7 {
public static void main(String[] args) {
try {
// CONEXION MySQL
Class.forName("com.mysql.jdbc.Driver");
Connection connection =
DriverManager.getConnection("jdbc:mysql://localhost:33
06/biblioteca", "root", "");
/*
// CONEXION Oracle
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection connection =
DriverManager.getConnection("jdbc:oracle:thin:@localho
st:1521:ORCL", "scott", "tiger");
*/
String idLibroBuscado = "00009";
CallableStatement callableStatement =
connection.prepareCall("{? = call
consulta_titulo_fun(?)}");
callableStatement.registerOutParameter(1, Types.VARCHAR
);
callableStatement.setString(2, idLibroBuscado);
callableStatement.executeUpdate();
System.out.println("título encontrado: " +
callableStatement.getString(1));
connection.close();
}
```

```
} catch(Exception e){  
e.printStackTrace();  
}  
}  
}  
}
```

Reproducimos en este ejercicio los mismos mecanismos que en el anterior. Tan sólo tener en cuenta la diferencia en la solicitud de invocación al procedimiento almacenado, que, por tratarse de una FUNCTION, ahora es:

```
CallableStatement callableStatement =  
connection.prepareCall("{? = call  
consulta_titulo_fun(?)}");
```

El parámetro de salida se ubica como receptor de la asignación de la expresión devuelta por la invocación a ejecución de la Function. Obviamente, cambia el orden de los parámetros de entrada y salida a aplicar en los métodos

- registerOutParameter()
- setString()
- getString()

en relación con el ejercicio anterior.

Con el siguiente ejercicio pretendemos presentar las interfaces *DatabaseMetaData* y *ResultSetMetaData*, cuyo nombre es lo suficiente ilustrativo como para no requerir aclaraciones sobre su cometido.

## EjercicioJDBC8

```
package ejerciciojdbc8;  
import java.sql.Connection;  
import java.sql.DatabaseMetaData;
```

```
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Statement;
public class EjercicioJDBC8 {
public static void main(String[] args) {
try {
// CONEXION MySQL
Class.forName("com.mysql.jdbc.Driver");
Connection connection =
DriverManager.getConnection("jdbc:mysql://localhost:33
06/biblioteca", "root", "");
/*
// CONEXION Oracle
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection connection =
DriverManager.getConnection("jdbc:oracle:thin:@localho
st:1521:ORCL", "scott", "tiger");
*/
System.out.println("-----
-----");
System.out.println("INFORMACIÓN OBTENIDA DEL
DatabaseMetaData");
System.out.println("-----
-----");
DatabaseMetaData databaseMetaData =
connection.getMetaData();
System.out.println("Gestor de Base de Datos : " +
databaseMetaData.getDatabaseProductName());
System.out.println("Driver : " +
databaseMetaData.getDriverName());
System.out.println("Máximo número de columnas en tabla
: " + databaseMetaData.getMaxColumnsInTable());
System.out.println("Máximo número de columnas en
SELECT : " +
databaseMetaData.getMaxColumnsInSelect());
System.out.println("-----");
```

```
System.out.println("PROCEDURES");
System.out.println("-----");
ResultSet procedimientos =
databaseMetaData.getProcedures(null, null, "%");
while (procedimientos.next()) {
System.out.println(procedimientos.getString(3));
}
System.out.println("-----");
System.out.println("FUNCTIONS");
System.out.println("-----");
ResultSet funciones =
databaseMetaData.getFunctions(null, null, "%");
while (funciones.next()) {
System.out.println(funciones.getString(3));
}
String tiposObjetosBD[] = {"TABLE", "VIEW"};
// ResultSet objetosBD =
databaseMetaData.getTables(null ,null, "L%", tiposObjetosBD); // Limitamos a aquellos cuyo nombre empieza por 'L'. Recomendable en la prueba con Oracle por el gran número de objetos.
ResultSet objetosBD = databaseMetaData.getTables(null ,null, "%", tiposObjetosBD); // Se puede utilizar en MySQL.
while (objetosBD.next())
{
String nombreObjeto = objetosBD.getString(3);
System.out.println("-----");
System.out.println("NOMBRE DEL OBJETO : " + nombreObjeto);
if (objetosBD.getString(4).compareTo("TABLE") == 0)
{
ResultSet columnasTabla =
databaseMetaData.getColumns(null, null, nombreObjeto, "%");
while (columnasTabla.next()) {
```

```
System.out.println("-----");
System.out.println("Nombre columna : " +
columnasTabla.getString(4));
System.out.println("Código tipo SQL : " +
columnasTabla.getString(5));
System.out.println("Tipo SQL : " +
columnasTabla.getString(6));
}
System.out.println("-----");
System.out.println("CLAVE PRIMARIA :");
ResultSet clavesPrimarias =
databaseMetaData.getPrimaryKeys(null, null,
nombreObjeto);
while (clavesPrimarias.next()) {
System.out.println(clavesPrimarias.getString(4));
}
System.out.println("-----");
System.out.println("CLAVES AJENAS SOBRE ESTA TABLA:");
ResultSet clavesAjenas =
databaseMetaData.getExportedKeys(null, null,
nombreObjeto);
while (clavesAjenas.next()) {
System.out.println("tabla : " +
clavesAjenas.getString(7));
System.out.println("columna : " +
clavesAjenas.getString(8));
System.out.println("nombre restriccion : " +
clavesAjenas.getString(12));
}
System.out.println("-----");
databaseMetaData.getURL();
System.out.println("-----");
System.out.println("CLAVES AJENAS PRESENTES EN ESTA
```

```
TABLA:");
ResultSet clavesAjenas2 =
databaseMetaData.getImportedKeys(null, null,
nombreObjeto);
while (clavesAjenas2.next()) {
System.out.println("columna : " +
clavesAjenas2.getString(8));
System.out.println("nombre restricción : " +
clavesAjenas2.getString(12));
}
}
}
System.out.println("-----
-----");
System.out.println("INFORMACIÓN OBTENIDA DEL
ResultSetMetaData");
System.out.println("-----
-----");
String sql = "SELECT * FROM libros";
Statement statement = connection.createStatement();
ResultSet resultSet = statement.executeQuery(sql);
// Columnas del ResultSet
ResultSetMetaData resultSetMetaData =
resultSet.getMetaData();
for (int i=1; i<= resultSetMetaData.getColumnCount(); i++) {
System.out.println("nombre columna : " +
resultSetMetaData.getColumnName(i));
System.out.println("clase que mapea en Java : " +
resultSetMetaData.getColumnClassName(i));
System.out.println("tipo SQL : " +
resultSetMetaData.getColumnTypeName(i));
System.out.println("código del tipo SQL : " +
resultSetMetaData.getColumnType(i));
System.out.println("tabla origen de la columna : " +
resultSetMetaData.getTableName(i));
System.out.println("-----")
```

```
-----”);
}
connection.close();
} catch(Exception e){
e.printStackTrace();
}
}
```

El *DatabaseMetaData* se obtiene:

```
DatabaseMetaData databaseMetaData =
connection.getMetaData();
```

y el *ResultSetMetaData*, para un determinado *ResultSet*:

```
ResultSetMetaData resultSetMetaData =
resultSet.getMetaData();
```

En este ejemplo, nos hemos limitado a invocar a algunos de los métodos de ambas interfaces, y a visualizar la información devuelta. Con ello, el código utilizado, no necesita ningún tipo de aclaración.

# 3

## MODELO DE DESARROLLO DE SOFTWARE ARQUITECTURA A TRES CAPAS

El modelo de desarrollo de software arquitectura a tres capas consiste en aportar unos criterios de organización-distribución-intercomunicación de los componentes de una aplicación distribuidos en tres capas con unas funciones claramente definidas:

- **PRESENTACIÓN.** La misión, desde una perspectiva de conjunto, de todos los componentes que integran esta capa es la interacción con el usuario. Nos podremos encontrar con diferentes variantes. En este libro utilizaremos dos de dichas variantes: interface tipo texto, en la aplicación **EjercicioJDBC9**, e interface gráfica con Swing, que estará presente en las aplicaciones **PrediccionMeteorologicaJTree**, **VotacionPropuesta**, y **GestionLibros** del segundo bloque de este libro. Seguir las directrices de la arquitectura a tres capas, permite intercambiar un tipo de interface de usuario por otro, manteniendo intactas las otras capas.

- **NEGOCIO.** Establece las reglas de negocio. Esta capa intermedia entre las otras dos, de tal manera que nunca existirá una interacción directa entre la capa de presentación y la de datos.
- **DATOS.** En esta capa se definen los tratamientos concretos relacionados con datos, generalmente externos, y que responden a tecnologías específicas. Dichos tratamientos están asociados, en la mayoría de las ocasiones a la persistencia de los mismos. El ejemplo más representativo de tratamiento en esta capa es el acceso a SGBD.

Intervienen otros componentes en la aplicación de forma transversal entre las citadas tres capas. Por una parte, los objetos **ENCAPSULADORES**, que permitirán la transferencia de información entre capas, y por otra las **EXCEPCIONES**, las cuales no son objeto de tratamiento en este libro. Si el lector no se encuentra ducho en todo lo que conlleva la gestión de las mismas, le remitimos al libro del autor “Programación Orientada a Objetos en Java”, donde encontrará un capítulo dedicado en exclusiva a este potente mecanismo de la orientación a objetos.

La implementación que aplicamos de la arquitectura a tres capas consiste en las siguientes actuaciones:

- Asociamos un *package* a cada una de las capas.
- En la capa de negocio definimos una clase para cada una de las “vertientes” o aspectos de negocio diferentes de la aplicación. Y un método de estas clases, para cada una de los grupos de acciones que deben ejecutarse en bloque, cuya ejecución, viene desencadenada, habitualmente, por

una acción de comando ejercida por el usuario desde la capa de presentación.

- En el *package* correspondiente a la capa de datos, definimos una clase asociada, normalmente, a cada una de las unidades externas de persistencia. El ejemplo más representativo de ello es, en JDBC, definir una clase para cada una de las tablas de la BD susceptibles de ser accedidas. Muestra de ello son las clases **LibrosDatos** y **GenerosDatos** integradas en la aplicación **EjercicioJDBC9**. En cada una de dichas clases, definimos un método, destinado en exclusiva al lanzamiento de cada una de las instrucciones SQL. También definimos una clase para gestionar la conexión con la BD, con un método destinado para cada una de las acciones inherentes a la gestión de la misma: abrir conexión, y cerrar conexión.
- Por cuestiones de organización, se suele destinar otro *package* para aglutinar las clases encapsuladoras que van a estar presentes en la aplicación. La configuración de atributos que presenta cada una de estas clases suelen ser una aproximación al conjunto de las columnas de una tabla. También es habitual encontrarse con una clase encapsuladora que vaya a aglutinar todas las expresiones devueltas por una consulta con combinación de tablas que vaya a ser lanzada desde la aplicación.

La comunicación entre capas, siempre es “vertical”, de tal forma que nunca un módulo de una capa invoca a otro de la misma capa, salvo excepción de descomposición de un módulo, y siempre en pro de la modularidad (valga la redundancia), donde se creará una jerarquía entre los bloques surgidos de la descomposición. Cuando hablamos de módulos como concepto

más abstracto nos estamos refiriendo, en programación Java, a métodos.

El ejemplo de este libro en que mejor se ilustra el carácter coordinador-vertebrador de acciones de la capa de datos, lo encontramos en la aplicación **VotacionPropuesta** tratada con amplitud en el bloque destinado a Swing de este libro. Nos estamos refiriendo, en concreto, al método **guardarVotacion()** de la clase **VotacionesNegocio**. Abordaremos a continuación el estudio de este método con detenimiento por tratarse del caso más representativo de módulo de la capa de negocio. El código de dicho método es;

```
public void guardarVotacion(BaseDatos baseDatos,
SistemaArchivos sistemaArchivos, Votacion votacion,
Object datos[][][], double[] contadorVotos) throws
Exception{
Connection connection = null;
ConexionBaseDatos conexionBaseDatos = new
ConexionBaseDatos();
try {
connection =
conexionBaseDatos.abrirConexion(baseDatos);
connection.setAutoCommit(false);
new VotacionesDatos().insertar(connection, votacion);
new VotosEmitidosDatos().insertar(connection,
votacion, datos);
new VotacionesPDF().generarPDF(sistemaArchivos,
votacion, contadorVotos);
connection.commit();
} catch (Exception excepcion)
{
connection.rollback();
throw excepcion;
}
finally
```

```
{  
conexionBaseDatos.cerrarConexion(connection);  
}  
}
```

Observemos que desde este módulo (método) de la capa de negocio se produce la invocación a métodos de la capa de datos que realizan las siguientes funciones:

- ABRIR CONEXIÓN
- INSERTAR FILA EN TABLA **votaciones**
- INSERTAR EN TABLA **votos\_emitidos** TANTAS FILAS COMO PERSONAS PARTICIPAN EN LA VOTACIÓN
- GENERAR FICHERO PDF CON INFORMACIÓN DE RESUMEN DEL PROCESO DE VOTACIÓN
- CERRAR CONEXIÓN

De la observación de este método, podemos matizar cuestiones ya expuestas con anterioridad:

- La comunicación entre capas es siempre vertical. La instanciación e invocación de métodos siempre es descendente en el sentido  
**PRESENTACION → NEGOCIO → DATOS**
- En el ejercicio de su cometido, los módulos de la capa de negocio representan una unidad de acción respecto a una acción de comando por parte del usuario de la aplicación, en el caso del método que estamos analizando, pulsar el *JButton botonGuardaVotacion*.

- Los módulos de la capa de datos representan la concreción en el tratamiento mediante las diversas tecnologías a aplicar en la E/S de la aplicación, siendo la persistencia el caso más generalizado. En el método de la capa de negocio que estamos analizando, además de la apertura y cierre de la conexión, dos de las acciones invocadas, representan acciones de inserción de filas en tablas de la BD, y otro método representa la salida a un documento PDF.
- Los objetos encapsuladores o estructuras aglutinadoras de datos actúan como elementos de intercambio de información entre capas. De tal forma que la información que es generada en capa superior, es transferida a capas inferiores como parámetro en la invocación a métodos, y la información que es generada en capas inferiores es devuelta a capas superiores asociado al identificador del método mediante *return* al finalizar la ejecución del método.

Intentaremos aclarar esta última matización mediante representaciones gráficas que utilizaremos en el estudio de la aplicación **EjercicioJDBC9** que exponemos a continuación.

# 4

## APLICACIÓN EJERCICIO JDBC9

La estructura que presenta esta aplicación corresponde a la establecida por el modelo de desarrollo de software arquitectura a tres capas:

- *package presentacion* aglutina las clases:
  - Inicio
  - Menu
- *package negocio* aglutina las clases:
  - LibrosNegocio
- *package datos* aglutina las clases:
  - ConexionBaseDatos
  - GenerosDatos
  - LibrosDatos
- *package encapsuladores* aglutina las clases:
  - Genero
  - Libro
- *package excepciones* aglutina las clases:
  - GenericaExcepcion

**Inicio**

```
package presentacion;
public class Inicio {
public static void main(String[] args) {
new Menu().ejecutarMenu();
}
}
```

## Menu

```
package presentacion;
import encapsuladores.Genero;
import encapsuladores.Libro;
import excepciones.GenericaExcepcion;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.text.SimpleDateFormat;
import java.util.List;
import negocio.LibrosNegocio;
public class Menu {
public void ejecutarMenu() {
BufferedReader bufferedReader = new BufferedReader(new
InputStreamReader(System.in));
byte opcionTecleada = 0;
do {
try {
System.out.println("-----");
System.out.println("GESTIÓN LIBROS -----");
System.out.println("1.- Insertar");
System.out.println("2.- Actualizar premiado");
System.out.println("3.- Eliminar libro");
System.out.println("4.- Consultar por identificador de
libro");
System.out.println("5.- Consultar todos");
System.out.println("6.- Consultar número de libros");
System.out.println("0.- Finalizar");
System.out.println("-----");
-----");
}
```

```
System.out.print("Seleccione opción : ");
opcionTecleada =
(byte)Integer.parseInt(bufferedReader.readLine());
switch(opcionTecleada)
{
case 1: insertar(bufferedReader);
break;
case 2: actualizarPremiado(bufferedReader);
break;
case 3: eliminar(bufferedReader);
break;
case 4: consultarPorIdLibro(bufferedReader);
break;
case 5: consultarTodos();
break;
case 6: consultarNumeroLibros();
break;
}
} catch (Exception exception)
{
gestionarExcepcion(exception);
}
} while (opcionTecleada != 0);
}

private void gestionarExcepcion(Exception excepcion)
{
int codigoError = 0;
String mensajeError = "";
if (excepcion instanceof GenericaExcepcion)
{
GenericaExcepcion genericaExcepcion =
(GenericExcepcion)excepcion;
codigoError = genericaExcepcion.getCodigoError();
switch (genericaExcepcion.getCodigoError())
{
case 50: mensajeError = "Se ha producido una situación
de error como consecuencia de problemas con la
```

```
conexión a la BD”;
break;
case 71: mensajeError = “No existe un libro con ese
identificador”;
break;
case 80: mensajeError = “Se ha producido una situación
de error en la BD al intentar insertar libro”;
break;
case 81: mensajeError = “Se ha producido una situación
de error en la BD al intentar actualizar premiado”;
break;
case 82: mensajeError = “Se ha producido una situación
de error en la BD al intentar eliminar libro”;
break;
case 83: mensajeError = “Se ha producido una situación
de error en la BD al intentar consultar por
identificador de libro”;
break;
case 84: mensajeError = “Se ha producido una situación
de error en la BD al intentar consultar todos los
libros”;
break;
case 85: mensajeError = “Se ha producido una situación
de error en la BD al intentar consultar el número de
libros”;
break;
case 90: mensajeError = “Se ha producido una situación
de error en la BD al intentar consultar la relación de
géneros”;
break;
}
}
else
{ if (excepcion instanceof NumberFormatException)
{
mensajeError = “La totalidad de los dígitos deben ser
numéricos”;
```

```
        }
    else
    {
        mensajeError = excepcion.getMessage();
    }
}
System.out.println("Código de error: " + codigoError +
" - " + mensajeError);
}
private void insertar(BufferedReader bufferedReader)
throws Exception
{
    System.out.println("----- INTRODUCCIÓN DATOS
DE NUEVO LIBRO -----");
    Libro libro = new Libro();
    System.out.print("identificador de libro : ");
    libro.setIdLibro(bufferedReader.readLine());
    System.out.print("título : ");
    libro.setTitulo(bufferedReader.readLine());
    List<Genero> listaGeneros = new
    LibrosNegocio().consultarGeneros();
    System.out.println("----- GÉNEROS -----
-----");
    for (int i=0; i<listaGeneros.size(); i++)
    {
        Genero genero = listaGeneros.get(i);
        System.out.println(genero.getCodigo() + " -
" + genero.getDescripcion());
    }
    System.out.print("introduzca código de género : ");
    libro.setGenero(bufferedReader.readLine());
    System.out.print("fecha edición (aaaa-mm-dd) : ");
    libro.setFechaEdicion(java.sql.Date.valueOf(bufferedRe
ader.readLine()));
    System.out.print("número páginas : ");
    libro.setNumeroPaginas(Integer.parseInt(bufferedReader
.readLine()));
```

```
System.out.print("premiado (0/1) : ");
int valorPremiado =
(Integer.parseInt(bufferedReader.readLine()));
if (valorPremiado == 1)
libro.setPremiado(true);
else
libro.setPremiado(false);
new LibrosNegocio().insertar(libro);
}
private void actualizarPremiado(BufferedReader
bufferedReader) throws Exception
{
System.out.println("----- ACTUALIZAR
PREMIADO -----");
Libro libro = new Libro();
System.out.print("identificador de libro : ");
libro.setIdLibro(bufferedReader.readLine());
System.out.print("premiado (0/1) : ");
int valorPremiado =
(Integer.parseInt(bufferedReader.readLine()));
if (valorPremiado == 1)
libro.setPremiado(true);
else
libro.setPremiado(false);
new LibrosNegocio().actualizarPremiado(libro);
}
private void eliminar(BufferedReader bufferedReader)
throws Exception
{
System.out.println("----- ELIMINAR
LIBRO -----");
Libro libro = new Libro();
System.out.print("identificador de libro : ");
libro.setIdLibro(bufferedReader.readLine());
new LibrosNegocio().eliminar(libro);
}
private void consultarPorIdLibro(BufferedReader
```

```
bufferedReader) throws Exception
{
System.out.println("----- CONSULTAR POR
IDENTIFICADOR DE LIBRO -----");
Libro libro = new Libro();
System.out.print("identificador de libro a consultar :
");
libro.setIdLibro(bufferedReader.readLine());
Libro libroObtenido = new
LibrosNegocio().consultarPorIdLibro(libro);
if (libroObtenido != null)
{
System.out.println("SE HA ENCONTRADO EL LIBRO : ");
System.out.println("identificador de libro : " +
libroObtenido.getIdLibro());
System.out.println("título : " +
libroObtenido.getTitulo());
System.out.println("género : " +
libroObtenido.getGenero() + " - " +
libroObtenido.getDescripcion());
System.out.println("fecha edición : " + new
SimpleDateFormat("yyyy-MM-
dd").format(libroObtenido.getFechaEdicion())));
System.out.println("número páginas : " +
libroObtenido.getNumeroPaginas());
String premiado = "NO";
if (libroObtenido.isPremiado())
premiado = "SI";
System.out.println("premiado : " + premiado);
}
else
System.out.println("NO EXISTE UN LIBRO CON EL
IDENTIFICADOR INTRODUCIDO");
}
private void consultarTodos() throws Exception
{
System.out.println("----- LISTADO
```

```

LIBROS -----”);
List<Libro> listaLibros = new
LibrosNegocio().consultarTodos();
for (int i=0; i<listaLibros.size(); i++)
{
Libro libro = listaLibros.get(i);
System.out.println("identificador de libro : " +
libro.getIdLibro());
System.out.println("título : " + libro.getTitulo());
System.out.println("género : " + libro.getGenero() + "-
" + libro.getDescripcion());
System.out.println("fecha edición : " + new
SimpleDateFormat("yyyy-MM-
dd").format(libro.getFechaEdicion()));
System.out.println("número páginas : " +
libro.getNumeroPaginas());
String premiado = "NO";
if (libro.isPremiado())
premiado = "SI";
System.out.println("premiado : " + premiado);
System.out.println("-----");
}
}
private void consultarNumeroLibros() throws Exception
{
System.out.println("----- CONSULTAR NÚMERO
DE LIBROS -----");
int numeroLibros = new
LibrosNegocio().consultarNumeroFilas();
System.out.println("número de libros : " +
numeroLibros);
}
}

```

## LibrosNegocio

```
package negocio;
import datos.ConexionBaseDatos;
import datos.GenerosDatos;
import datos.LibrosDatos;
import encapsuladores.Genero;
import encapsuladores.Libro;
import excepciones.GenericaExcepcion;
import java.sql.Connection;
import java.util.List;
public class LibrosNegocio {
    public String insertar(Libro libro) throws Exception{
        Connection connection = null;
        ConexionBaseDatos conexionBaseDatos = new
        ConexionBaseDatos();
        try {
            connection = conexionBaseDatos.abrirConexion();
            new LibrosDatos().insertar(connection, libro);
        } catch (Exception excepcion)
        {
            throw excepcion;
        }
        finally
        {
            conexionBaseDatos.cerrarConexion(connection);
        }
        return libro.getIdLibro();
    }
    public void eliminar(Libro libro) throws Exception{
        Connection connection = null;
        ConexionBaseDatos conexionBaseDatos = new
        ConexionBaseDatos();
        LibrosDatos librosDatos = new LibrosDatos();
        try {
            connection = conexionBaseDatos.abrirConexion();
            Libro libroAEliminar =
            librosDatos.consultarPorIdLibro(connection, libro);
            if (libroAEliminar == null)

```

```
throw new GenericaExcepcion(71);
librosDatos.eliminar(connection, libro);
} catch (Exception excepcion)
{
throw excepcion;
}
finally
{
conexionBaseDatos.cerrarConexion(connection);
}
}

public void actualizarPremiado(Libro libro) throws
Exception{
Connection connection = null;
ConexionBaseDatos conexionBaseDatos = new
ConexionBaseDatos();
LibrosDatos librosDatos = new LibrosDatos();
try {
connection = conexionBaseDatos.abrirConexion();
Libro libroAActualizar =
librosDatos.consultarPorIdLibro(connection, libro);
if (libroAActualizar == null)
throw new GenericaExcepcion(71);
librosDatos.actualizarPremiado(connection, libro);
} catch (Exception excepcion)
{
throw excepcion;
}
finally
{
conexionBaseDatos.cerrarConexion(connection);
}
}

public Libro consultarPorIdLibro(Libro libro) throws
Exception
{
Connection connection = null;
```

```
Libro libroObtenido = null;
ConexionBaseDatos conexionBaseDatos = new
ConexionBaseDatos();
try {
connection = conexionBaseDatos.abrirConexion();
libroObtenido = new
LibrosDatos().consultarPorIdLibro(connection, libro);
} catch (Exception excepcion)
{
throw excepcion;
}
finally
{
conexionBaseDatos.cerrarConexion(connection);
}
return libroObtenido;
}
public List<Libro> consultarTodos() throws Exception
{
Connection connection=null;
ConexionBaseDatos conexionBaseDatos = new
ConexionBaseDatos();
List<Libro> listaLibros = null;
try {
connection = conexionBaseDatos.abrirConexion();
listaLibros = new
LibrosDatos().consultarTodos(connection);
} catch (Exception excepcion)
{
throw excepcion;
}
finally
{
conexionBaseDatos.cerrarConexion(connection);
}
return listaLibros;
}
```

```
public Integer consultarNumeroFilas() throws Exception
{
Connection connection=null;
ConexionBaseDatos conexionBaseDatos = new
ConexionBaseDatos();
Integer numFilas = null;
try {
connection = conexionBaseDatos.abrirConexion();
numFilas = new
LibrosDatos().consultarNumeroFilas(connection);
} catch (Exception excepcion)
{
throw excepcion;
}
finally
{
conexionBaseDatos.cerrarConexion(connection);
}
return numFilas;
}
public List<Genero> consultarGeneros() throws
Exception
{
Connection connection=null;
ConexionBaseDatos conexionBaseDatos = new
ConexionBaseDatos();
List<Genero> listaGeneros = null;
try {
connection = conexionBaseDatos.abrirConexion();
listaGeneros = new
GenerosDatos().consultarGeneros(connection);
} catch (Exception excepcion)
{
throw excepcion;
}
finally
{
```

```
conexionBaseDatos.cerrarConexion(connection);
}
return listaGeneros;
}
}
```

## ConexionBaseDatos

```
package datos;
import excepciones.GenericaExcepcion;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
public class ConexionBaseDatos {
public Connection abrirConexion() throws Exception
{
Connection connection = null;
try {
// CONEXION MySQL
Class.forName("com.mysql.jdbc.Driver");
connection =
DriverManager.getConnection("jdbc:mysql://localhost:33
06/biblioteca", "root", "");
/*
// CONEXION Oracle
Class.forName("oracle.jdbc.driver.OracleDriver");
connection =
DriverManager.getConnection("jdbc:oracle:thin:@localho
st:1521:ORCL", "scott", "tiger");
*/
} catch (SQLException excepcion) {
throw new GenericaExcepcion(50);
}
return connection;
}
public void cerrarConexion(Connection connection)
throws GenericaExcepcion
```

```
{  
try {  
if (connection!= null)  
connection.close();  
} catch (SQLException excepcion) {  
throw new GenericaExcepcion(50);  
}  
}  
}
```

## LibrosDatos

```
package datos;  
import encapsuladores.Libro;  
import excepciones.GenericaExcepcion;  
import java.sql.Connection;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.sql.Statement;  
import java.util.ArrayList;  
import java.util.List;  
public class LibrosDatos {  
public void insertar(Connection connection, Libro  
libro) throws Exception  
{ PreparedStatement preparedStatement = null;  
try {  
String sql = “INSERT INTO libros VALUES(?, ?, ?, ?, ?, ?, ?)”;  
preparedStatement = connection.prepareStatement(sql);  
preparedStatement.setString(1, libro.getIdLibro());  
preparedStatement.setString(2, libro.getTitulo());  
preparedStatement.setString(3, libro.getGenero());  
preparedStatement.setDate(4, libro.getFechaEdicion());  
preparedStatement.setInt(5, libro.getNumeroPaginas());  
byte premiado = 0;  
if (libro.isPremiado())  
premiado = 1;
```

```
preparedStatement.setByte(6, premiado);
preparedStatement.executeUpdate();
} catch (SQLException excepcion) {
throw new GenericaExcepcion(80);
} finally
{
if (preparedStatement != null)
preparedStatement.close();
}
}

public void actualizarPremiado(Connection connection,
Libro libro) throws Exception
{ PreparedStatement preparedStatement = null;
try {
String sql = "UPDATE libros SET premiado = ? WHERE
id_libro = CAST(? AS CHAR(5))";
preparedStatement = connection.prepareStatement(sql);
preparedStatement.setString(2, libro.getIdLibro());
byte premiado = 0;
if (libro.isPremiado())
premiado = 1;
preparedStatement.setByte(1, premiado);
preparedStatement.executeUpdate();
} catch (SQLException excepcion) {
throw new GenericaExcepcion(81);
} finally
{
if (preparedStatement != null)
preparedStatement.close();
}
}

public void eliminar(Connection connection, Libro
libro) throws Exception
{ PreparedStatement preparedStatement = null;
try {
String sql = "DELETE FROM libros WHERE id_libro =
CAST(? AS CHAR(5))";
```

```
preparedStatement = connection.prepareStatement(sql);
preparedStatement.setString(1, libro.getIdLibro());
preparedStatement.executeUpdate();
} catch (SQLException excepcion) {
throw new GenericaExcepcion(82);
} finally
{
if (preparedStatement != null)
preparedStatement.close();
}
}

public Libro consultarPorIdLibro(Connection
connection, Libro libro) throws Exception
{
Libro libroObtenido = null;
ResultSet resultSet = null;
PreparedStatement preparedStatement = null;
try {
String sql = "SELECT id_libro, titulo, generos.codigo,
generos.descripcion, fecha_edicion, numero_paginas,
premiado FROM generos INNER JOIN libros ON
generos.codigo = libros.genero WHERE id_libro = CAST(? AS CHAR(5))";
preparedStatement = connection.prepareStatement(sql);
preparedStatement.setString(1, libro.getIdLibro());
resultSet = preparedStatement.executeQuery();
if (resultSet.next()) {
libroObtenido = new Libro();
libroObtenido.setIdLibro(resultSet.getString(1));
libroObtenido.setTitulo(resultSet.getString(2));
libroObtenido.setGenero(resultSet.getString(3));
libroObtenido.setDescripcion(resultSet.getString(4));
libroObtenido.setFechaEdicion(resultSet.getDate(5));
libroObtenido.setNumeroPaginas(resultSet.getInt(6));
byte premiado = resultSet.getByte(7);
if (premiado == 1)
libroObtenido.setPremiado(true);
}
}
}
```

```
else
libroObtenido.setPremiado(false);
}
} catch (SQLException excepcion) {
throw new GenericaExcepcion(83);
} finally
{
if (resultSet != null) resultSet.close();
if (preparedStatement != null)
preparedStatement.close();
}
return libroObtenido;
}
public List<Libro> consultarTodos(Connection
connection) throws Exception
{
List<Libro> listaLibros = new ArrayList();
ResultSet resultSet = null;
Statement statement = null;
try {
String sql = "SELECT id_libro, titulo, generos.codigo,
generos.descripcion, fecha_edicion, numero_paginas,
premiado FROM generos INNER JOIN libros ON
generos.codigo = libros.genero ORDER BY titulo";
statement = connection.createStatement();
resultSet = statement.executeQuery(sql);
while (resultSet.next()) {
Libro libro = new Libro();
libro = new Libro();
libro.setIdLibro(resultSet.getString(1));
libro.setTitulo(resultSet.getString(2));
libro.setGenero(resultSet.getString(3));
libro.setDescripcion(resultSet.getString(4));
libro.setFechaEdicion(resultSet.getDate(5));
libro.setNumeroPaginas(resultSet.getInt(6));
byte premiado = resultSet.getByte(7);
if (premiado == 1)
```

```
libro.setPremiado(true);
else
libro.setPremiado(false);
listaLibros.add(libro);
}
} catch (SQLException excepcion) {
throw new GenericaExcepcion(84);
} finally
{
if (resultSet != null) resultSet.close();
if (statement != null) statement.close();
}
return listaLibros;
}

public Integer consultarNumeroFilas(Connection
connection) throws Exception
{
Integer numFilas = null;
ResultSet resultSet = null;
Statement statement = null;
String sql = "SELECT COUNT(*) FROM libros";
try {
statement = connection.createStatement();
resultSet = statement.executeQuery(sql);
if (resultSet.next()) {
numFilas = new Integer(resultSet.getInt(1));
}
} catch (SQLException excepcion) {
throw new GenericaExcepcion(85);
} finally
{
if (resultSet != null) resultSet.close();
if (statement != null) statement.close();
}
return numFilas;
}
}
```

## GenerosDatos

```
package datos;
import encapsuladores.Genero;
import excepciones.GenericaExcepcion;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
public class GenerosDatos {
    public List<Genero> consultarGeneros(Connection
connection) throws Exception
    {
        List<Genero> listaGeneros = new ArrayList();
        ResultSet resultSet = null;
        Statement statement = null;
        try {
            String sql = "SELECT * FROM generos ORDER BY codigo";
            statement = connection.createStatement();
            resultSet = statement.executeQuery(sql);
            while (resultSet.next()) {
                Genero genero = new Genero();
                genero.setCodigo(resultSet.getString(1));
                genero.setDescripcion(resultSet.getString(2));
                listaGeneros.add(genero);
            }
        } catch (SQLException excepcion) {
            throw new GenericaExcepcion(90);
        } finally
        {
            if (resultSet != null) resultSet.close();
            if (statement != null) statement.close();
        }
        return listaGeneros;
    }
}
```

## Libro

```
package encapsuladores;
import java.sql.Date;
public class Libro {
private String idLibro;
private String titulo;
private String genero;
private String descripcion;
private Date fechaEdicion;
private int numeroPaginas;
private boolean premiado;
public String getIdLibro() {
return idLibro;
}
public void setIdLibro(String idLibro) {
this.idLibro = idLibro;
}
public String getTitulo() {
return titulo;
}
public void setTitulo(String titulo) {
this.titulo = titulo;
}
public String getGenero() {
return genero;
}
public void setGenero(String genero) {
this.genero = genero;
}
public String getDescripcion() {
return descripcion;
}
public void setDescripcion(String descripcion) {
this.descripcion = descripcion;
}
public Date getFechaEdicion() {
return fechaEdicion;
}
```

```
}

public void setFechaEdicion(Date fechaEdicion) {
this.fechaEdicion = fechaEdicion;
}
public int getNumeroPaginas() {
return numeroPaginas;
}
public void setNumeroPaginas(int numeroPaginas) {
this.numeroPaginas = numeroPaginas;
}
public boolean isPremiado() {
return premiado;
}
public void setPremiado(boolean premiado) {
this.premiado = premiado;
}
}
```

## Genero

```
package encapsuladores;
public class Genero {
private String codigo;
private String descripcion;
public String getCodigo() {
return codigo;
}
public void setCodigo(String codigo) {
this.codigo = codigo;
}
public String getDescripcion() {
return descripcion;
}
public void setDescripcion(String descripcion) {
this.descripcion = descripcion;
}
}
```

## GenericaExcepcion

```
package excepciones;
public class GenericaExcepcion extends Exception{
private int codigoError;
public GenericaExcepcion (int codigoError){
this.codigoError = codigoError;
}
public int getCodigoError(){
return codigoError;
}
}
```

En esta aplicación se han integrado las acciones SQL, presentadas individualmente en ejercicios anteriores, y que en su conjunto constituyen un CRUD sobre una tabla de una Base de Datos, pero ahora todo ello integrado y estructurado atendiendo al modelo de desarrollo de software arquitectura a tres capas. Este ejercicio representa la mínima expresión de una aplicación organizada siguiendo las directrices a tres capas. Ha sido diseñada a propósito, en un contexto lo más reducido posible, con la intención de facilitar al máximo la comprensión del lector de los aspectos más representativos de esta arquitectura. El estudio de la estructura organizativa a tres capas de esta aplicación, facultará al lector en la comprensión de aplicaciones de mayor complejidad, también organizadas a tres capas, correspondientes al bloque Swing de este libro. Debe tener presente el lector que los beneficios y ventajas de aplicar las recomendaciones de este modelo de desarrollo de software quedan patentes en aplicaciones de mayor envergadura.

La capa de presentación aplicación **EjercicioJDBC9**, utiliza, como ya se mencionado, una interfaz de usuario tipo texto, articulada sobre un menú de usuario que proporciona acceso a las funcionalidades correspondientes a un CRUD sobre la tabla

libros. Exponemos a continuación la visualización obtenida en consola de salida al ejecutar la aplicación:

## **GESTIÓN LIBROS**

- 1.- Insertar
  - 2.- Actualizar premiado
  - 3.- Eliminar libro
  - 4.- Consultar por identificador de libro
  - 5.- Consultar todos
  - 6.- Consultar número de libros
  - 0.- Finalizar
- 
- 

Seleccione opción : 1

## **INTRODUCCIÓN DATOS DE NUEVO LIBRO**

identificador de libro : 520

título : FUNDAMENTOS DE SISTEMAS OPERATIVOS

## **GÉNEROS**

- 0 - Generalidades
  - 1 - Filosofia. Psicologia
  - 2 - Religion. Teologia
  - 3 - Ciencias sociales
  - 4 - No clasificado
  - 5 - Matematicas. Ciencias naturales
  - 6 - Ciencias aplicadas
  - 7 - Bellas artes. Deportes
  - 8 - Literatura
  - 9 - Geografia. Historia
- introduzca código de género : 0
- fecha edición (aaaa-mm-dd) : 1980-11-05
- número páginas : 538
- premiado (0/1) : 0

## **GESTIÓN LIBROS**

- 1.- Insertar**
  - 2.- Actualizar premiado**
  - 3.- Eliminar libro**
  - 4.- Consultar por identificador de libro**
  - 5.- Consultar todos**
  - 6.- Consultar número de libros**
  - 0.- Finalizar**
- 
- 

Seleccione opción : 4

### **CONSULTAR POR IDENTIFICADOR DE LIBRO**

identificador de libro a consultar : 520  
SE HA ENCONTRADO EL LIBRO :  
identificador de libro : 520  
título : FUNDAMENTOS DE SISTEMAS OPERATIVOS  
género : 0 - Generalidades  
fecha edición : 1980-11-05  
número páginas : 538  
premiado : NO

## **GESTIÓN LIBROS**

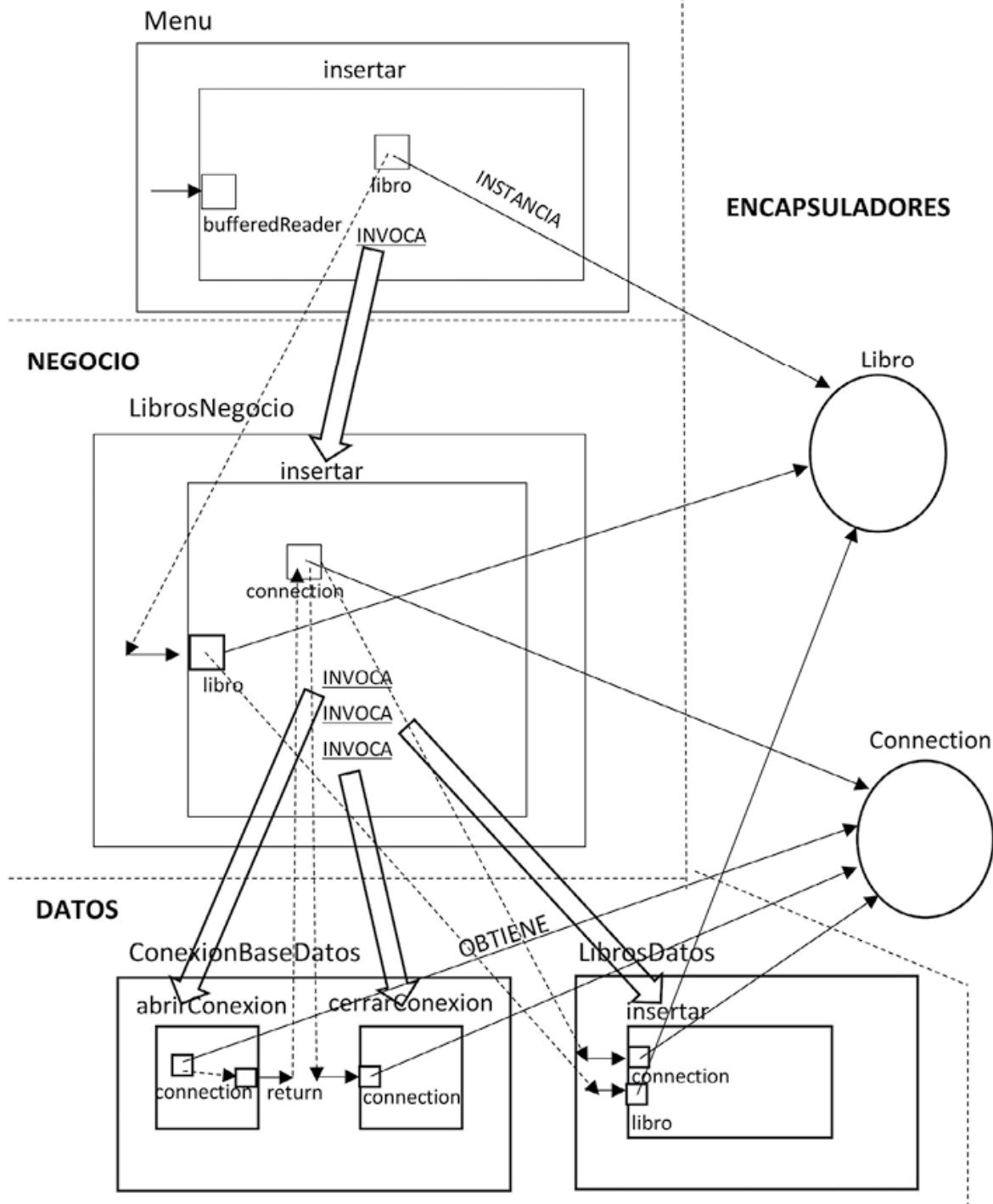
- 1.- Insertar**
  - 2.- Actualizar premiado**
  - 3.- Eliminar libro**
  - 4.- Consultar por identificador de libro**
  - 5.- Consultar todos**
  - 6.- Consultar número de libros**
  - 0.- Finalizar**
- 
- 

Seleccione opción : 0

Estudiaremos los mecanismos de la arquitectura a tres capas aplicados a las funcionalidades más representativas. Se han seleccionado dichas funcionalidades atendiendo a presentar todas las variantes en relación al sentido del flujo de información ascendente-descendente entre capas. Procedemos a un estudio individualizado de cada una de dichas funcionalidades.

## **INSERTAR**

## PRESENTACION



Representa la variante de sentido de flujo totalmente descendente.  
De la observación de la representación gráfica, deducimos que en

la capa de presentación, concretamente en el método **insertar()** de la clase **Menu**, se produce la captura por teclado correspondiente a los datos a insertar. Dicha información es encapsulada en un objeto instanciado a tal propósito de la clase **Libro**, y “viaja descendentemente” a la capa de negocio (método **insertar()** de la clase **LibrosNegocio**). La capa de negocio actúa, como ha se ha comentado, “de intermediaria” entre las otras dos, en consecuencia, la información encapsulada en el objeto de la clase Libros, sigue “viajando descendente” hasta la capa de datos (método **insertar()** de la clase **LibrosDatos**). Permite el lector la licencia metafórica empleada relacionada con la expresión “viaja descendentemente”, que no responde a un rigor técnico pero contribuye a la comprensión del mecanismo. Obviamente, cuando nos referimos a que la información “viaja”, nos estamos refiriendo a la transferencia de referencias al objeto de un método a otro:

## PRESENTACION → NEGOCIO

```
new LibrosNegocio().insertar(libro) → public String  
insertar(Libro libro) {...}
```

## NEGOCIO → DATOS

```
new LibrosDatos().insertar(..., libro) → public void  
insertar(..., Libro libro) {...}
```

que permite que los datos encapsulados en el objeto sean accesibles desde otras capas mediante los “setters” y los “getters”. En el caso que estamos tratando de la funcionalidad INSERTAR, podemos decir (siguiendo con el sentido figurativo) que en la capa datos “finaliza el viaje” de la información encapsulada en el objeto Libro, habiendo “alcanzado su destino”, que era estar disponible para ser aplicada a los métodos *setXXX()* del

*PreparedStatement*, y poder lanzar a la Base de Datos la instrucción:

```
String sql = "INSERT INTO libros VALUES(?, ?, ?, ?, ?, ?, ?)"
```

Hemos descrito una de las tres invocaciones a módulos de la capa de datos que se producen desde la capa de negocio. Las otras dos invocaciones corresponden, como se observa en la representación gráfica, a las acciones de apertura y cierre de la conexión a la Base de Datos. Centrémonos en la operación de apertura de la conexión. El método **abrirConexion()** de la clase **ConexionBaseDatos** se encarga de implementar todas las actuaciones específicas relacionadas con JDBC que permiten obtener la citada conexión. Con lo que podríamos decir que es el encargado de “generar” dicha información (la referencia al objeto *Connection* que modela la conexión). Nos encontramos con un caso de información generada por capas “inferiores”, que debe “ascender”. Como ya se ha comentado, el mecanismo utilizado para “devolver” la información generada por método es asociarla al identificador del método mediante *return*:

## DATOS → NEGOCIO

```
public Connection abrirConexion() { . . . return  
connection} →  
connection = conexionBaseDatos.abrirConexion()
```

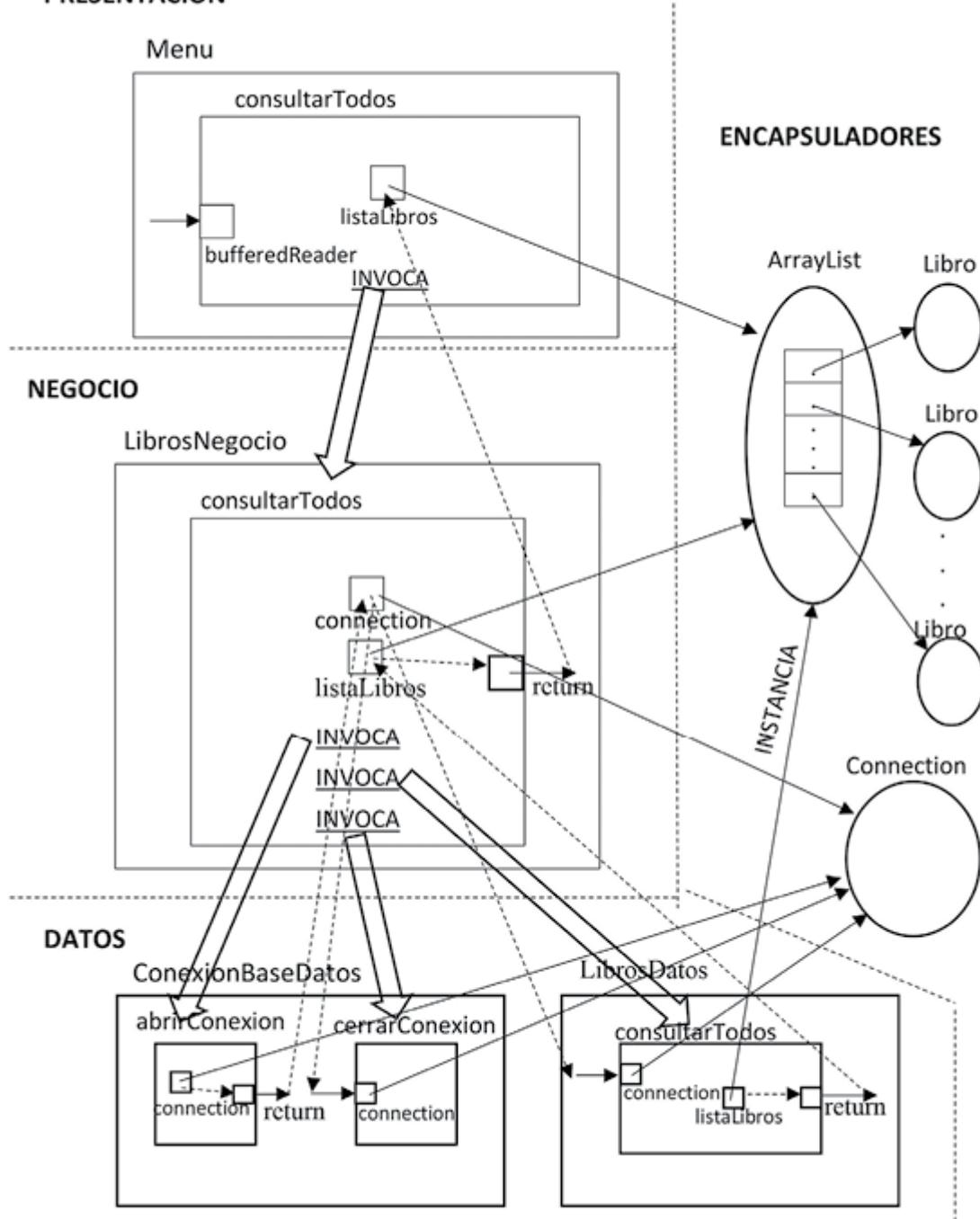
En los casos en que el método debe devolver más de un dato, recurrimos a aglutinar dicho conjunto de datos en un array de *Object*. Volvamos a la referencia al objeto *Connection* generado por el método de la capa de datos que es transferido al método de la capa de negocio, y que representa la conexión abierta a la Base de Datos. Dicha referencia será transferida, a su vez, pero ahora “descendentemente”, a todos los métodos de la capa de datos asociados a esta acción de negocio que vayan a lanzar SQL:

**NEGOCIO → DATOS**

```
new LibrosDatos().insertar(connection, ...); → public  
void insertar  
(Connection connection, ...) {...}
```

**CONSULTAR TODOS**

## PRESENTACION



Supone el ejemplo totalmente antagónico al anterior, en lo que al sentido del flujo de la información se refiere. El anterior, el flujo de la información era totalmente “descendente”, y ahora es

totalmente “ascendente”. La SQL que constituye el eje central de esta funcionalidad es:

```
String sql = "SELECT id_libro, titulo, generos.codigo,  
generos.descripcion,  
fecha_edicion, numero_paginas, premiado  
FROM generos INNER JOIN libros ON generos.codigo =  
libros.genero ORDER BY titulo
```

que, al no presentar parámetros, no requiere de información solicitada al usuario.

Prescindimos de comentar los mecanismos asociados al objeto *Connection* que modela la conexión a la BD, ya comentados en la funcionalidad anterior, y que se repiten en todos los módulos de la capa de negocio. Iniciemos “el viaje” de la información generada en la capa de datos al lanzar la SQL, y que “ascenderá” hasta la capa de presentación para ser visualizada al usuario. En el inicio del proceso, nos encontramos con la información obtenida por la consulta en el objeto *ResultSet*. Perfectamente podríamos “devolver hacia arriba” mediante los dos *return* sucesivos:

DATOS → NEGOCIO

```
public List<Libro> consultarTodos(...) { . . . return  
listaLibros } →  
listaLibros = new LibrosDatos().consultarTodos(...)
```

NEGOCIO → PRESENTACION

```
public List<Libro> consultarTodos() { . . . return  
listaLibros } →  
List<Libro> listaLibros = new  
LibrosNegocio().consultarTodos()
```

la referencia al *ResultSet* obtenido. Pero este objeto se encuentra contextualizado en una tecnología concreta, en este caso la

persistencia en Bases de Datos Relacionales mediante JDBC. La “filosofía” del modelo a tres capas comporta que la naturaleza tecnológica del origen de la información debe ser totalmente abstraída entre capas. Para acogernos a este precepto, nos vemos obligados a “trasvasar” el conjunto de los datos modelados mediante el *ResultSet*, a una estructura de datos genérica, en este caso un *ArrayList* que nos permitirá aglutinar cada uno de los objetos de la clase Libro que encapsularán los datos de cada una de las “filas” del *ResulSet*:

### *ResultSet → ArrayList*

que pueda “viajar” perfectamente entre capas sin transgredir la directriz comentada. Esta actuación es realiza en el mismo método de la capa de datos encargado de lanzar la SQL:

```
List<Libro> listaLibros = new ArrayList();
. . .
resultSet = statement.executeQuery(sql);
while (resultSet.next()) {
    Libro libro = new Libro();
    libro = new Libro();
    libro.setIdLibro(resultSet.getString(1));
    libro.setTitulo(resultSet.getString(2));
    libro.setGenero(resultSet.getString(3));
    libro.setDescripcion(resultSet.getString(4));
    libro.setFechaEdicion(resultSet.getDate(5));
    libro.setNumeroPaginas(resultSet.getInt(6));
    byte premiado = resultSet.getByte(7);
    if (premiado == 1)
        libro.setPremiado(true);
    else
        libro.setPremiado(false);
    listaLibros.add(libro);
}
```

Cuando la referencia al *ArrayList* “ascendente llega” a la capa de presentación, ya estamos en condiciones de aplicar el debido proceso al conjunto de datos que aglutina, en este vaso visualizar al usuario:

```
List<Libro> listaLibros = new
LibrosNegocio().consultarTodos();
for (int i=0; i<listaLibros.size(); i++)
{
    Libro libro = listaLibros.get(i);
    System.out.println("identificador de libro : " +
libro.getIdLibro());
    System.out.println("título : " + libro.getTitulo());
    System.out.println("género : " + libro.getGenero() + "-
" + libro.getDescripcion());
    System.out.println("fecha edición : " + new
SimpleDateFormat("yyyy-MM-
dd").format(libro.getFechaEdicion()));
    System.out.println("número páginas : " +
libro.getNumeroPaginas());
    String premiado = "NO";
    if (libro.isPremiado())
        premiado = "SI";
    System.out.println("premiado : " + premiado);
    System.out.println("-----");
}
}
```

La misma circunstancia de reclusión del *ResultSet* a la capa de datos, dada la naturaleza específica de JDBC, la hacemos extensiva a *SQLException*. El “trasvase” de la situación anómala que desencadenó la excepción queda implementado:

### ***SQLException → GenericaExpcion***

“desactivando” *SQLException* y “pasándole el testigo a un nuevo corredor que entra en pista” (utilizamos las competiciones

deportivas de carreras de relevos para ilustrar metafóricamente lo que les ocurre al *ResultSet* y a *SQLException*). Para que se produzca el “trasvase” que estamos tratando, capturamos la *SQLException* en el bloque *try{}*, con lo que quedará “desactivada”, y acto seguido, en el bloque *catch{}*, instanciamos y lanzamos una excepción definida por el programador que actuará con carácter genérico en la aplicación:

```
try {  
} catch (SQLException excepcion) {  
throw new GenericaExcepcion(80);  
}
```

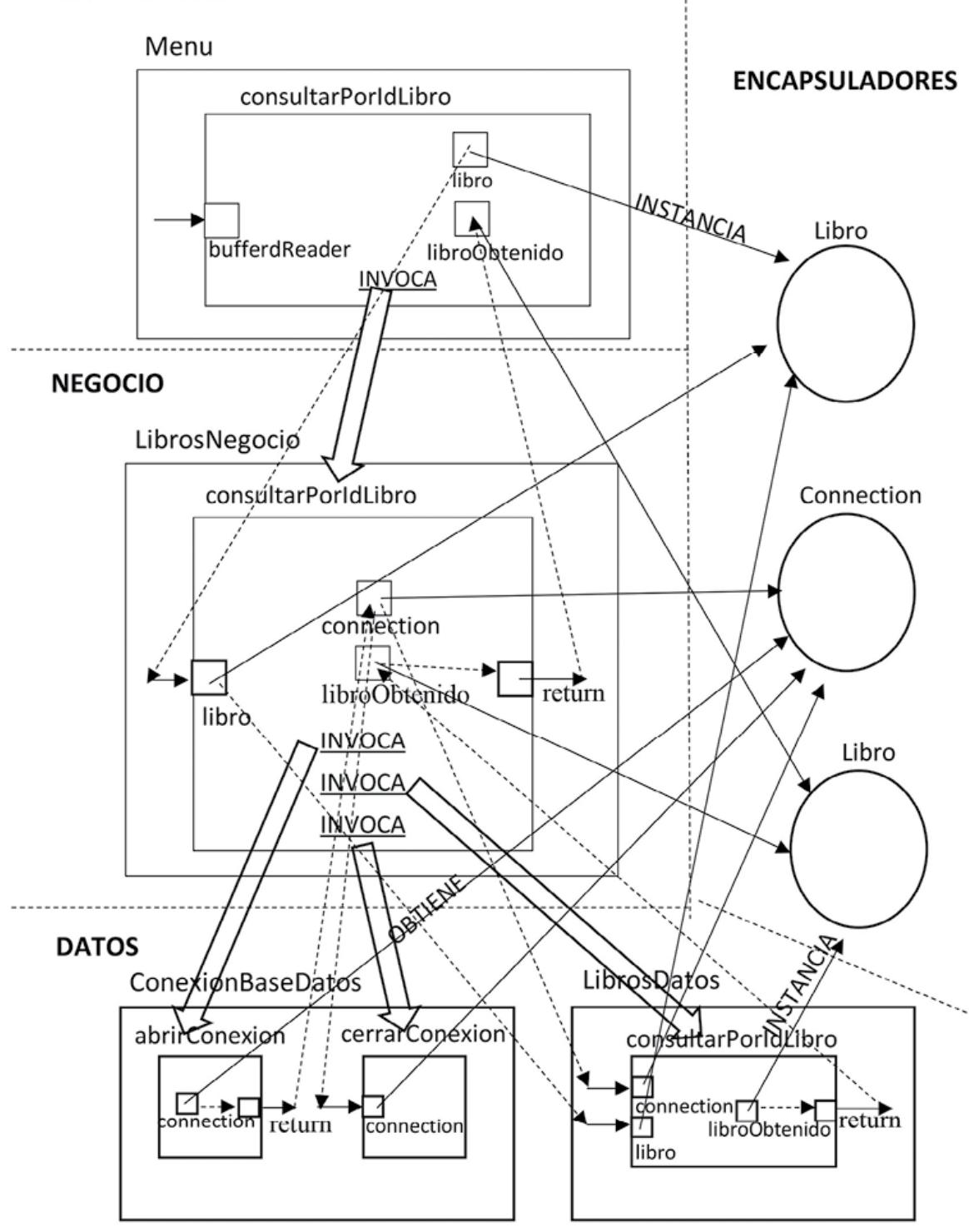
En la instanciación de **GenericaExcepcion** transferimos como parámetro al constructor un código asociado a la anomalía. El valor de dicho código es posteriormente desencapsulado mediante el método **getCodigoError()** en la capa de presentación para proceder al tratamiento correspondiente. Concretamente en esta aplicación, en el método **gestionarExcepcion()** de la clase **Menu**.

Es necesario matizar el “acoplamiento” polimórfico implementado al declarar como *List* la referencia receptora declarada en “capas superiores en el viaje ascendente” del *ArrayList*. Concretamente, ha sido declarada del tipo de la interface que implementa el objeto. Esta circunstancia nos proporciona versatilidad, permitiéndonos actuar con carácter genérico al posibilitar a dicha referencia “recoger” cualquier estructura de datos que implemente la interface en cuestión. Un escenario análogo nos lo encontramos en algunos IDEs, como por ejemplo Netbeans, en el momento de proporcionar una sugerencia de todos los métodos que puedan ser invocados desde una referencia, con los métodos en que se da este caso, califica como *List<...>* la estructura a devolver. Dicha circunstancia

permite al programador abstraerse de la concreción de la clase cuya referencia nos devuelven, y limitarnos a procesar la estructura, con los métodos y mecanismos asociados a la interface.

## **CONSULTAR POR IDENTIFICADOR DE LIBRO**

## PRESENTACION



La SQL que constituye el eje central de esta funcionalidad es:

```
String sql = "SELECT id_libro, titulo, generos.codigo,  
generos.descripcion,  
fecha_edicion, numero_paginas, premiado  
FROM generos INNER JOIN libros ON generos.codigo =  
libros.genero WHERE id_libro = CAST(?) AS CHAR(5))"
```

Esta funcionalidad responde a un esquema “híbrido” de los dos anteriores, en lo que al sentido ascendente-descendente del flujo de la información entre capas se refiere. Está sometido a un flujo descendente el valor de la clave primaria solicitado al usuario, al que se destina para ser encapsulado, un objeto de la clase Libro con carácter exclusivo a tal fin. El lector puede tener la impresión de que es un “derroche de espacio” utilizar todo un objeto encapsulador solamente para un atributo, pero como ya se ha comentado, responde a una de las directrices de la arquitectura a tres capas, el intentar que la comunicación entre las mismas sea mediante objetos encapsuladores, o estructuras de datos, en su caso. Volvamos al recorrido “descendente” de la clave primaria capturada por teclado y encapsulada en un objeto de la clase Libro:

## PRESENTACION → NEGOCIO

```
libro.setIdLibro(bufferedReader.readLine())  
new LibrosNegocio().consultarPorIdLibro(libro) →  
public Libro consultarPorIdLibro  
(Libro libro) {...}
```

## NEGOCIO → DATOS

```
new LibrosDatos().consultarPorIdLibro(..., libro) →  
public Libro  
consultarPorIdLibro(..., Libro libro) {...}
```

Cuando el valor de dicha clave primaria llega al final de su “trayecto descendente” es utilizada para establecer el correspondiente parámetro en la cláusula WHERE:

```
preparedStatement.setString(1, libro.getIdLibro())
```

Lanzada la consulta de búsqueda de una fila por coincidencia de valor de clave primaria, nos podemos encontrar con dos posibles escenarios:

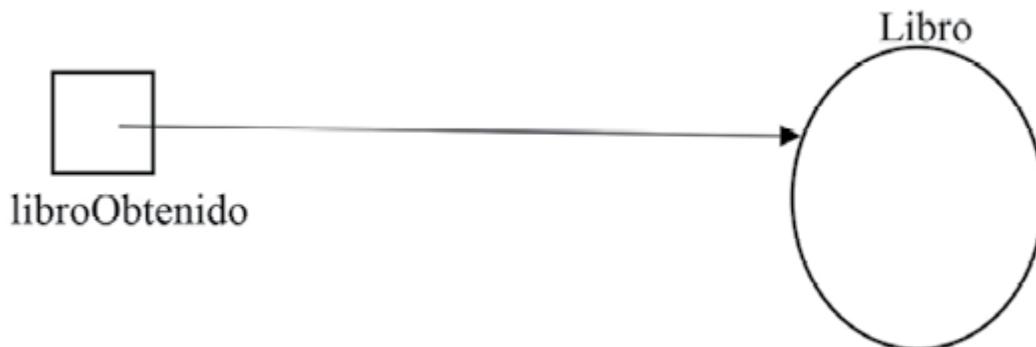
1. Encuentra dicha fila en la tabla de la BD. En este caso, la invocación al método

```
resultSet.next()
```

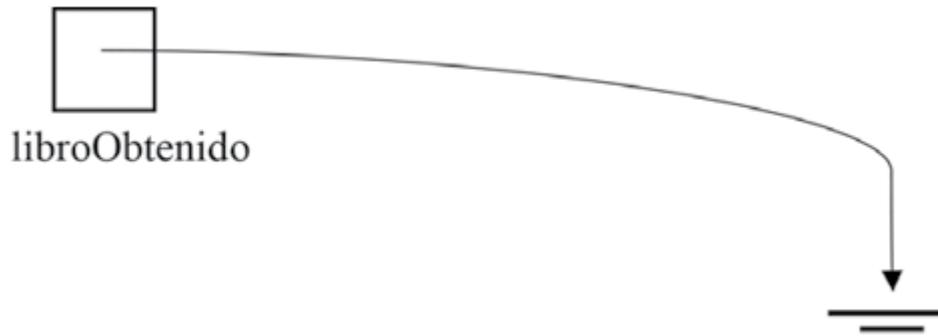
devuelve **true**, ejecutándose así el bloque de acciones de la estructura de control alternativa simple. En la ejecución de dicho bloque, la primera actuación será instanciar un objeto de la clase Libro para encapsular los valores de las columnas de la fila encontrada

```
libroObtenido = new Libro()
```

y la referencia de dicho objeto es asignada **libroObtenido**, declarada a tal efecto. A continuación, se procede a encapsular los datos devueltos por la consulta en el mencionado objeto Libro.



2. No encuentra la fila en la tabla de la BD. En este caso, no se ejecutará el bloque de acciones de la alternativa simple, y la referencia **libroObtenido** conservará el valor *null* que se le asignó inicialmente.



## El método de la capa de datos

```
public Libro consultarPorIdLibro(Connection
connection, Libro libro) throws Exception
{ . . .
return libroObtenido;
}
```

devuelve la referencia **libroObtenido**, “apuntando” a un objeto encapsulador, en el primer caso descrito, o a *null*, en el segundo caso. En uno u otro caso, el valor de la referencia “inicia su ascenso” hacia la capa de datos:

## DATOS → NEGOCIO

```
public Libro consultarPorIdLibro(...) { . . .
return libroObtenido } →
libroObtenido = new
LibrosDatos().consultarPorIdLibro(...)
```

## NEGOCIO → PRESENTACION

```
public Libro consultarPorIdLibro(...) { . . .
return libroObtenido } →
Libro libroObtenido = new
LibrosNegocio().consultarPorIdLibro(...)
```

En capas superiores se detectará si ha sido localizada la fila buscada en la BD o no ha sido localizada “sondeando” el valor de

la referencia devuelta:

```
if (libroObtenido != null)
{
    PROCESO DE LOS DATOS ENCAPSULADOS EN EL OBJETO
    REFERENCIADO POR libroObtenido
}
else
    System.out.println("NO EXISTE UN LIBRO CON EL
    IDENTIFICADOR INTRODUCIDO");
```

Se podía haber utilizado el mismo objeto encapsulador de la clase Libro para “bajar” a la capa de datos encapsulado el valor de la clave primaria, y después reutilizarlo para “subir” los valores de las columnas de la fila encontrada (en su caso) a la capa de presentación. Se ha optado por utilizar dos objetos diferentes, uno para cada cometido, en pro de facilitar al lector la comprensión de los mecanismos aplicados.

La conveniencia de proceder al cierre “ordenado” de objetos, después de haber sido utilizados, en las capas de datos y negocio, con total independencia de que se produzca un relanzamiento de excepción, nos obliga a implementar el bloque *finally{}}*. En este bloque se realizarán las mencionadas acciones de cierre de objetos, mediante la invocación al método *close()* de los mismos. La presencia del bloque *finally{}* en la capa de negocio debe ser acompañada por el bloque *try{}*. Consecuencia de ello, es que será “capturada” la excepción que ha sido relanzada desde la capa de datos, y quedará “desactivada”. Necesitamos volver a “reactivarla” para que pueda ser relanzada, a su vez, a la capa de presentación, en que recibirá el tratamiento oportuno. Conseguimos dicha “reactivación” en:

```
catch (Exception excepcion)
{
    throw excepcion;
```

}

## TRANSACCIÓN

Prosiguiendo con la estructuración de los métodos de la capa de negocio asociada al tratamiento de excepciones, comentamos los mecanismos que utilizamos en arquitectura a tres capas para implementar una transacción, es decir, un conjunto de acciones SQL que deben ejecutarse todas o ninguna para que la Base de Datos quede en estado coherente. Conocido será de sobra por el lector que COMMIT es la instrucción SQL encargada de la confirmación definitiva de acciones provisionales, y que ROLLBACK, encarga de “retrotraer” la Base de Datos al estado en que se produjo la última confirmación, “deshaciendo” las acciones provisionales pendientes. JDBC aporta, vinculados al objeto que modela la conexión, los métodos *commit()* y *rollback()* que producen tal efecto, sin necesidad de un lanzamiento explícito de las correspondientes SQL. Con la mención de estos conceptos, se planteará el lector que todas en las pruebas realizadas hasta ahora con DMLs de actualización no ha sido necesario lanzar un COMMIT explícito para que hayan quedado como definitivas las actualizaciones en la BD como consecuencia del lanzamiento de dichas SQL. Ello se debe a que JDBC tiene establecido en las conexiones la modalidad de “autocommit” por defecto. Para poder añadir al código de nuestra aplicación la invocación a los métodos *commit()* y *rollback()* en el lugar que les corresponde, es necesario desactivar el citado “autocommit”. Ello se consigue mediante:

```
connection.setAutoCommit(false);
```

El concepto de transacción ha sido implementado en el método **guardarVotacion()** de la clase **VotacionesNegocio()** de la aplicación **VotacionPropuesta**. Dicho método ya se ha expuesto

y analizado con anterioridad como ejemplo más representativo de implementación de lógica de negocio, coordinando-vertebrando la invocación a varias acciones de la capa de datos. En la observación de dicho método, centrándose en los aspectos que estamos comentando, localizará el lector la desactivación del “autocommit” antes de invocar a ningún método de la capa de datos que suponga el lanzamiento de una DML de actualización:

```
connection.setAutoCommit(false);
```

La confirmación de la transacción:

```
connection.commit();
```

es ubicada la última acción del bloque *try{}*, punto de paso obligado, si no se ha interceptado ninguna excepción. Y la retrotracción de las acciones SQL que pudiesen haber sido lanzadas, al inicio del bloque *catch{}*:

```
connection.rollback()
```

## CONTROL DE LAS REGLAS DE NEGOCIO

Hemos mencionado hasta ahora que una de las funciones de los módulos de la capa de negocio era coordinar-vertebrar la invocación a los módulos de la capa de datos que intervenían en una acción de negocio. Vamos a añadir a dicha función de la capa de negocio, otra, concretamente la de controlar que se cumplen las reglas de negocio. Los mecanismos que utilizamos para implementar dicho control quedan patentes en el método **autenticar()** de la clase **UsuariosBibliotecaNegocio** de la aplicación **GestionLibros**. Postergamos para el final del libro el estudio de dicha aplicación por ser la que presenta mayor complejidad como consecuencia de integrar todos los contenidos y mecanismos tratados a lo largo de este libro. No obstante, adelantamos el estudio del método que acabamos de citar dado

que su idiosincrasia encaja totalmente en el estudio de los mecanismos a aplicar en el control de las reglas de negocio. La codificación del mismo es:

```
public Contexto autenticar(BaseDatos baseDatos,
Contexto usuarioIntroducido) throws Exception
{
Connection connection = null;
Contexto usuarioAutenticado = null;
ConexionBaseDatos conexionBaseDatos = new
ConexionBaseDatos();
try {
connection =
conexionBaseDatos.abrirConexion(baseDatos);
usuarioAutenticado = new
UsuariosBibliotecaDatos().buscarPorIdUsuario(connectio
n, usuarioIntroducido);
if (usuarioAutenticado == null)
throw new GenericaExcepcion(20);
if
(usuarioAutenticado.getPassword().compareTo(usuarioInt
roducido.getPassword()) != 0)
throw new GenericaExcepcion(21);
usuarioAutenticado.setIpCliente(usuarioIntroducido.get
IpCliente());
} catch (Exception excepcion)
{
throw excepcion;
}
finally
{
conexionBaseDatos.cerrarConexion(connection);
}
return usuarioAutenticado;
}
```

La misión de este método es someter a un control de autenticación el conjunto usuario-password transferido desde la capa de presentación. Las principales acciones que coordina-vertebra el método son:

- ABRIR CONEXIÓN
- CONSULTAR USUARIO POR COINCIDENCIA DE CLAVE PRIMARIA
- SI NO EXISTE UNA FILA CON DICHO VALOR DE CLAVE PRIMARIA ENTONCES **LANZA EXCEPCIÓN**
- SI EL PASSWORD RECUPERADO DE LA CONSULTA NO COINCIDE CON EL PASSWORD TRANSFERIDO DESDE LA CAPA DE PRESENTACIÓN ENTONCES **LANZA EXCEPCIÓN**
- CERRAR CONEXIÓN

Comprobamos que la detección de cualquier anomalía asociada a la autenticación conlleva el lanzamiento de una excepción, la cual recibirá el tratamiento oportuno en la capa de presentación, donde será interceptada. De ser correcta la autenticación, en la capa de presentación se “abrirá” una sesión de usuario conectado.

# 5

## POOL DE CONEXIONES

Desarrollar la aplicación atendiendo al modelo estructural organizativo de arquitectura a tres capas que hemos planteado trae las siguientes implicaciones:

- Solamente permanece abierta una conexión con la Base de Datos el tiempo que supone la ejecución de los métodos de la capa de datos invocados coordinados-vertebrados desde un método de la capa de negocio. A lo que se le añade los posibles controles asociados a la lógica de negocio, si es que se producen. Hay que hacer constar que las directrices de la arquitectura a tres capas establecen que la actividad asociada a la interacción solamente debe y puede producirse en la capa de presentación. De todo ello podemos deducir que son mínimos los instantes en que permanece abierta una conexión asociada a una acción de la lógica de negocio.
- Es necesario abrir la conexión con la BD cada vez que se ejecuta una acción de dicha lógica de negocio. Abrir una conexión con la BD supone un coste de proceso, podríamos decir de cierta consideración. Esta circunstancia representa una cuestión importante de eficiencia a plantearse, o incluso, según el caso, un problema en servidores con elevada carga concurrente.

Una solución que permite paliar o subsanar el problema que hemos descrito es recurrir a un pool de conexiones. Este concepto hace referencia a la posibilidad de que existan permanentemente abiertas un número determinado de conexiones, de tal manera que cuando un hilo de ejecución requiere de una conexión, no necesita abrirla y cerrarla explícitamente, sino que se limita a solicitar al pool una, que obviamente ya esté abierta. Si hay una conexión disponible en esos instantes, es concedida al hilo de ejecución solicitante, éste la utiliza, y proceder a liberarla cuando ya no la necesita, a fin de que otro pueda reutilizarla.

Exponemos en el siguiente capítulo una aplicación en que se hace uso de un pool de conexiones.

# 6

## APLICACIÓN POOLCONEXIONES

La estructura que presenta esta aplicación corresponde a la establecida por el modelo de desarrollo de software arquitectura a tres capas:

- *package presentacion* aglutina las clases:
  - Inicio
- *package negocio* aglutina las clases:
  - LibrosNegocio
- *package datos* aglutina las clases:
  - ConexionBaseDatos
  - LibrosDatos
- *package encapsuladores* aglutina las clases:
  - Libro
- *package excepciones* aglutina las clases:
  - GenericaExcepcion

### Inicio

```
package presentacion;  
import encapsuladores.Libro;
```

```
import excepciones.GenericaExcepcion;
import java.util.List;
import negocio.LibrosNegocio;
public class Inicio {
public static void main(String[] args) {
try {
// Abrimos conexiones del pool hasta que lance una
excepción por tenerlas todas ocupadas
// Lanzamos una SQL desde cada una de las conexiones
abiertas
// Posteriormente las primeras diez se cierran y se
reutilizan inmediatamente volviendo a lanzar la SQL
List<String> listaMensajes = new
LibrosNegocio().probarOcupacionPoolConexiones();
for (int i=0; i<listaMensajes.size(); i++)
System.out.println(listaMensajes.get(i));
// Utilizamos una única conexión para lanzar la SQL.
Sería esta la implementación convencional y habitual.
System.out.println("LA SIGUIENTE INVOCACION AL METODO
DE LA CAPA DE NEGOCIO REPRESENTA LA IMPLEMENTACION
CONVENCIONAL Y HABITUAL");
System.out.println("----- RELACION DE LIBROS PRESENTES
EN LA TABLA -----");
List<Libro> listaLibros = new
LibrosNegocio().consultarTodos();
for (int i=0; i<listaLibros.size(); i++)
{
Libro libro = listaLibros.get(i);
System.out.print(libro.getIdLibro() + " - ");
System.out.println(libro.getTitulo());
}
} catch (Exception excepcion)
{
String mensajeError = "";
if (excepcion instanceof GenericaExcepcion)
{
GenericaExcepcion genericaExcepcion =
```

```

(GenericException)excepcion;
switch (genericaExcepcion.getCodigoError())
{
case 20: mensajeError = "Todas las conexiones del pool
están ocupadas en estos momentos. En breve habrá
alguna libre.";
break;
case 21: mensajeError = "Se ha producido una situación
de error al intentar abrir una conexión del pool.";
break;
case 22: mensajeError = "Se ha producido una situación
de error al cerrar una conexión del pool.";
break;
case 84: mensajeError = "Se ha producido una situación
de error en la BD al intentar consultar la relación de
libros.";
break;
case 85: mensajeError = "Se ha producido una situación
de error en la BD al intentar consultar número de
filas.";
break;
}
}
else
{ mensajeError = "Se ha producido una situación de
error: " + excepcion.getMessage(); }
System.out.println(mensajeError);
}
}
}

```

## LibrosNegocio

```

package negocio;
import datos.ConexionBaseDatos;
import datos.LibrosDatos;
import encapsuladores.Libro;

```

```
import excepciones.GenericaExcepcion;
import java.sql.Connection;
import java.util.ArrayList;
import java.util.List;
public class LibrosNegocio {
    public List<Libro> consultarTodos() throws Exception
    {
        Connection connection=null;
        List<Libro> listaLibros = null;
        try {
            connection =
            ConexionBaseDatos.getInstancia().abrirConexion();
            listaLibros = new
            LibrosDatos().consultarTodos(connection);
        } catch (Exception excepcion)
        {
            throw excepcion;
        }
        finally
        {
            ConexionBaseDatos.getInstancia().cerrarConexion(connection);
        }
        return listaLibros;
    }
    public List<String> probarOcupacionPoolConexiones()
    throws Exception
    {
        List<String> listaMensajes = new ArrayList();
        listaMensajes.add("----- PRUEBA DE TODAS LAS
CONEXIONES QUE SE PUEDEN ABRIR EN EL POOL DADA LA
CONFIGURACION QUE SE LE HA ESTABLECIDO -----");
        // APERTURA DE CONEXIONES
        Connection[] conexiones = new Connection[1000];
        int numeroConexionesAbiertas = 0;
        boolean interceptadaExcepcion = false;
        boolean abiertasTodasLasConexionesDisponibles = false;
```

```
while (!interceptadaExcepcion)
{
try {
conexiones[numeroConexionesAbiertas] =
ConexionBaseDatos.getInstancia().abrirConexion();
listaMensajes.add("Abierta conexión del pool : " +
numeroConexionesAbiertas);
numeroConexionesAbiertas++;
} catch (Exception excepcion)
{
interceptadaExcepcion = true;
if (excepcion instanceof GenericaExcepcion)
{
GenericaExcepcion genericaExcepcion =
(GenericExcepcion)excepcion;
if (genericaExcepcion.getCodigoError() == 20)
{
abiertasTodasLasConexionesDisponibles = true;
listaMensajes.add("Se han podido abrir : " +
numeroConexionesAbiertas + " conexiones");
listaMensajes.add("----- Al intentar abrir nueva
conexión en el pool, se ha interceptado excepcion
asociada a que todas las conexiones están ocupadas ---");
}
else
{
throw excepcion;
}
}
}
}
}
if (abiertasTodasLasConexionesDisponibles)
{
// LANZAMIENTO DE SQL DESDE CADA UNA DE LAS CONEXIONES
ABIERTAS
// Evidenciamos que siguen abiertas las conexiones
```

```
del pool lanzando una SQL desde cada una de ellas
for (int i=0; i<numeroConexionesAbiertas; i++)
{
    listaMensajes.add("Desde la conexión "+ i +" lanzamos
la SQL obteniendo el número de filas de la tabla : " +
(new
    LibrosDatos().consultarNumeroFilas(conexiones[i])).int
Value());
}
// CIERRE DE 10 DE LAS CONEXIONES ABIERTAS. ABRIENDO
UNA NUEVA PARA CADA UNA DE LAS QUE SE CIERRA, LANZANDO
UNA SQL DESDE CADA CONEXION QUE SE ABRE
listaMensajes.add("PROCEDEMOS A CERRAR CONEXIONES Y A
REUTILIZARLAS ABRIENDO UNA CADA VEZ QUE SE CIERRA
OTRA");
for (int i=0; i<10; i++)
{
    ConexionBaseDatos.getInstancia().cerrarConexion(conexi
ones[i]);
    listaMensajes.add("Se ha cerrado la conexión del pool
: " + i);
    conexiones[i] =
    ConexionBaseDatos.getInstancia().abrirConexion();
    listaMensajes.add("Se ha reutilizado la conexión del
pool : " + i);
    listaMensajes.add("Desde la conexión "+ i +" lanzamos
la SQL obteniendo el número de filas de la tabla : " +
(new
    LibrosDatos().consultarNumeroFilas(conexiones[i])).int
Value());
}
// CIERRE DE TODAS LAS CONEXIONES ABIERTAS
listaMensajes.add("PROCEDEMOS AL CIERRE DE TODAS LAS
CONEXIONES ABIERTAS");
for (int i=0; i<numeroConexionesAbiertas; i++)
{ // Cerramos todas las conexiones del pool
    ConexionBaseDatos.getInstancia().cerrarConexion(conexi
```

```
ones[i]);
listaMensajes.add("Se ha cerrado la conexión del pool
: " + i);
}
}
return listaMensajes;
}
}
```

## ConexionBaseDatos

```
package datos;
import excepciones.GenericaExcepcion;
import java.sql.Connection;
import java.sql.SQLException;
import oracle.ucp.jdbc.PoolDataSource;
import oracle.ucp.jdbc.PoolDataSourceFactory;
import org.apache.commons.dbcp2.BasicDataSource;
public class ConexionBaseDatos {
// **** MySQL ****
private static final BasicDataSource BASICDATASOURCE =
new BasicDataSource();
private static final String CLASSDRIVER =
"com.mysql.jdbc.Driver";
private static final String URLCONEXION =
"jdbc:mysql://localhost/biblioteca";
private static final String USUARIO = "root";
private static final String PASSWORD = "";
private static ConexionBaseDatos instancia;
private static boolean dataSourceInicializado = false;
static {
BASICDATASOURCE.setDriverClassName(CLASSDRIVER);
BASICDATASOURCE.setUrl(URLCONEXION);
BASICDATASOURCE.setUsername(USUARIO);
BASICDATASOURCE.setPassword(PASSWORD);
BASICDATASOURCE.setInitialSize(50);
```

```
BASICDATASOURCE.setMinIdle(2);
BASICDATASOURCE.setMaxIdle(50);
BASICDATASOURCE.setMaxTotal(50);
}
public Connection abrirConexion() throws Exception {
Connection connection = null;
try {
if (BASICDATASOURCE.getNumIdle() == 0 &&
dataSourceInicializado)
throw new GenericaExcepcion(20);
connection = BASICDATASOURCE.getConnection();
dataSourceInicializado = true;
} catch (SQLException excepcion) {
throw new GenericaExcepcion(21);
}
return connection;
}
//
*****
*****
/*
// **** Oracle
*****
private static final PoolDataSource POOLDATASOURCE =
PoolDataSourceFactory.getPoolDataSource();
private static final String CLASSDRIVER =
“oracle.jdbc.pool.OracleDataSource”;
private static final String URLCONEXION =
“jdbc:oracle:thin:@localhost:1521:ORCL”;
private static final String USUARIO = “scott”;
private static final String PASSWORD = “tiger”;
private static ConexionBaseDatos instancia;
private static boolean dataSourceInicializado = false;
static {
try {
POOLDATASOURCE.setConnectionFactoryClassName(CLASSDRIV-
ER);
```

```
POOLDATASOURCE.setURL(URLCONEXION);
POOLDATASOURCE.setUser(USUARIO);
POOLDATASOURCE.setPassword(PASSWORD);
POOLDATASOURCE.setInitialPoolSize(50);
POOLDATASOURCE.setMinPoolSize(1);
POOLDATASOURCE.setMaxPoolSize(50);
} catch (SQLException excepcion) {
System.out.println("Se ha producido un problema al
intentar inicializar el pool de conexiones");
}
}
public Connection abrirConexion() throws Exception {
Connection connection = null;
try {
if (POOLDATASOURCE.getAvailableConnectionsCount() == 0
&& dataSourceInicializado)
throw new GenericaExcepcion(20);
connection = POOLDATASOURCE.getConnection();
dataSourceInicializado = true;
} catch (SQLException excepcion) {
throw new GenericaExcepcion(20);
}
return connection;
}
//
*****
*****
*/
public static ConexionBaseDatos getInstancia() {
if (instancia == null)
instancia = new ConexionBaseDatos();
return instancia;
}
public void cerrarConexion(Connection connection)
throws GenericaExcepcion {
try {
if (connection!= null)
```

```
connection.close();
} catch (SQLException excepcion) {
throw new GenericaExcepcion(22);
}
}
}
```

## LibrosDatos

```
package datos;
import encapsuladores.Libro;
import excepciones.GenericaExcepcion;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
public class LibrosDatos {
public List<Libro> consultarTodos(Connection
connection) throws Exception
{
List<Libro> listaLibros = new ArrayList();
ResultSet resultSet = null;
Statement statement = null;
try {
String sql = "SELECT id_libro, titulo, generos.codigo,
generos.descripcion, fecha_edicion, numero_paginas,
premiado FROM generos INNER JOIN libros ON
generos.codigo = libros.genero ORDER BY titulo";
statement = connection.createStatement();
resultSet = statement.executeQuery(sql);
while (resultSet.next()) {
Libro libro = new Libro();
libro = new Libro();
libro.setIdLibro(resultSet.getString(1));
libro.setTitulo(resultSet.getString(2));
}
```

```
libro.setGenero(resultSet.getString(3));
libro.setDescripcion(resultSet.getString(4));
libro.setFechaEdicion(resultSet.getDate(5));
libro.setNumeroPaginas(resultSet.getInt(6));
byte premiado = resultSet.getByte(7);
if (premiado == 1)
libro.setPremiado(true);
else
libro.setPremiado(false);
listaLibros.add(libro);
}
} catch (SQLException excepcion) {
throw new GenericaExcepcion(84);
} finally
{
if (resultSet != null) resultSet.close();
if (statement != null) statement.close();
}
return listaLibros;
}
public Integer consultarNumeroFilas(Connection
connection) throws Exception
{
Integer numFilas = null;
ResultSet resultSet = null;
Statement statement = null;
String sql = "SELECT COUNT(*) FROM libros";
try {
statement = connection.createStatement();
resultSet = statement.executeQuery(sql);
if (resultSet.next()) {
numFilas = new Integer(resultSet.getInt(1));
}
} catch (SQLException excepcion) {
throw new GenericaExcepcion(85);
} finally
{
```

```
if (resultSet != null) resultSet.close();
if (statement != null) statement.close();
}
return numFilas;
}
}
```

## Libro

```
package encapsuladores;
import java.sql.Date;
public class Libro {
private String idLibro;
private String titulo;
private String genero;
private String descripcion;
private Date fechaEdicion;
private int numeroPaginas;
private boolean premiado;
public String getIdLibro() {
return idLibro;
}
public void setIdLibro(String idLibro) {
this.idLibro = idLibro;
}
public String getTitulo() {
return titulo;
}
public void setTitulo(String titulo) {
this.titulo = titulo;
}
public String getGenero() {
return genero;
}
public void setGenero(String genero) {
this.genero = genero;
}
```

```
public String getDescripcion() {
    return descripcion;
}
public void setDescripcion(String descripcion) {
    this.descripcion = descripcion;
}
public Date getFechaEdicion() {
    return fechaEdicion;
}
public void setFechaEdicion(Date fechaEdicion) {
    this.fechaEdicion = fechaEdicion;
}
public int getNumeroPaginas() {
    return numeroPaginas;
}
public void setNumeroPaginas(int numeroPaginas) {
    this.numeroPaginas = numeroPaginas;
}
public boolean isPremiado() {
    return premiado;
}
public void setPremiado(boolean premiado) {
    this.premiado = premiado;
}
```

## GenericaExcepcion

```
package excepciones;
public class GenericaExcepcion extends Exception{
    private int codigoError;
    public GenericaExcepcion (int codigoError){
        this.codigoError = codigoError;
    }
    public int getCodigoError(){
        return codigoError;
    }
```

}

Análogamente al resto de aplicaciones de este libro, ésta está preparada para conectar tanto con MySQL, como con Oracle en lo que a la implementación del código se refiere. Para cada uno de ellos, se requiere añadir las librerías que se enumeran:

- **MySQL**

- **mysql-connector-java-5.1.23-bin.jar** Es la equivalente a la que hemos añadido a todos los proyectos. En esta aplicación hemos aportado una versión posterior.
- **commons-dbcp2-2.7.0.jar** Disponible a descarga desde la web de Apache, al igual que las dos siguientes.
- **commons-logging-1.2.jar**
- **commons-pool2-2.7.0.jar**

- **Oracle**

- **ojdbc8.jar** Es la equivalente a la que hemos añadido a todos los proyectos. Hemos aportado una versión posterior por requerimientos del pool de conexiones.
- **ucp.jar**

Por cuestión de licencias, las librerías requeridas para Oracle no se aportan en los proyectos que conforman el material descargable del libro.

Esta aplicación es una réplica de EjercicioJDBC9 en la que solamente se han conservado los módulos imprescindibles para mostrar al lector los detalles inherentes al trabajo con pool de

conexiones. Solamente nos centraremos en aquellos métodos en los que aparecen variaciones respecto al anterior. Se ha prescindido de menú de usuario, manteniendo en la clase **Inicio** dos invocaciones a métodos de la capa de negocio:

- **new LibrosNegocio().consultarTodos()** Representa lo que sería la implementación habitual de métodos de la capa de negocio de aplicaciones que contemplan pool de conexiones. Las únicas divergencias en la codificación del método invocado respecto a los métodos homólogos de las aplicaciones de este libro en que no se utiliza pool de conexiones, radican en la invocación a apertura y cierre de la conexión:
  - connection =  
    ConexionBaseDatos.getInstancia().abrirConexion();
  - ConexionBaseDatos.getInstancia().cerrarConexion(connection);
- **new LibrosNegocio().probarOcupacionPoolConexiones()** El lector puede tener la impresión de que se trata de un método en que se implementan actuaciones “insólitas”, totalmente distanciadas del resto de las contempladas en este libro, las cuales pretenden responder a la mayor aproximación posible a la realidad productiva. Debe tener en cuenta el lector que las actuaciones implementadas en este método responden a una intención exclusivamente didáctica que también estudiaremos en profundidad.

Para abordar los detalles de la gestión del pool, hemos de centrarnos en la clase **ConexionBaseDatos**. En la codificación de esta clase están activadas todas las actuaciones necesarias para conectar con MySQL, y comentarizadas las correspondientes a la conexión con Oracle. Comentaremos las que están activadas, asociadas al primer SGBD. Comprobará el lector, que, salvo cuestiones de terminología sintáctica, existe un total paralelismo en las actuaciones a realizar en ambos SGBD, de tal modo que lo que se comente para MySQL podrá hacerse extensivo a Oracle.

El pool de conexiones es la solución a que se suele recurrir en aplicaciones con alto grado de concurrencia. El pool requiere una gestión centralizada, circunstancia que implica que solamente pueda existir instanciado un objeto de la clase que gestiona dicho pool, en el caso de nuestra aplicación ejemplo, asume este papel la clase **ConexionBaseDatos**. A tal fin, la hemos configurado atendiendo al patrón de diseño *Singleton*, manifestándose en las siguientes actuaciones:

- Definición de una referencia static que pueda “apuntar” al único objeto de la propia clase que puede ser instanciado:

```
private static ConexionBaseDatos instancia;
```

- Dicha instanciación se producirá en el primer acceso a dicha referencia “canalizado” mediante su correspondiente método “getter”:

```
public static ConexionBaseDatos getInstancia() {  
    if (instancia == null)  
        instancia = new ConexionBaseDatos();  
    return instancia;  
}
```

La invocación a este método permitirá dar acceso, a su vez, a los métodos de la misma clase:

- public Connection abrirConexion()
- public void cerrarConexion(Connection connection)

cuyas actuaciones no van a consistir realmente, en abrir y cerrar conexión, sino en solicitar conexión disponible al pool, y liberarla cuando ya no va a ser utilizada.

Cualquier clase que modele un pool de conexiones debe implementar la interface *javax.sql.DataSource*. Para MySQL, disponemos de la clase *BasicDataSource* aportada por Apache. Invocamos a sus métodos “setters” para establecer las principales características que determinarán el comportamiento del pool:

- **setDriverClassName( CLASSDRIVER )**
- **setUrl( URLCONEXION )**
- **setUsername( USUARIO )**
- **setPassword( PASSWORD )**
- **setMaxTotal( 50 )**
- **setInitialSize( 50 )** Número de conexiones que se abren cuando se inicializa el pool.
- **setMinIdle( 2 )** Número mínimo de conexiones no utilizadas en un instante determinado. Se abrirán conexiones en la medida haga falta mantener dicho número mínimo establecido, que actuará como resguardo para dar servicio instantáneo a potenciales peticiones de conexión.

- **setMaxIdle( 50 )** Número máximo de conexiones no utilizadas que puede haber en un instante determinado. De sobrepasarse este número, se cerrarían conexiones.

Tal y como hemos señalado, la petición de una conexión disponible al pool se efectúa en el método **abrirConexion()**, el cual devolverá dicha conexión, si ello es posible, en función de la disponibilidad existente de las mismas condicionada por las limitaciones inherentes a los parámetros de funcionamiento establecidos al pool. Observamos en la codificación del método, que se solicita conexión al pool mediante:

```
connection = BASICDATASOURCE.getConnection()
```

previa comprobación de que hay conexiones disponibles. Esta acción se ejecuta si se supera “la barrera” del lanzamiento de la excepción encapsulando un valor cuya codificación implica que no hay conexiones disponibles. El susodicho lanzamiento de excepción se produce si se detecta que no hay conexiones susceptibles de ser concedidas (el método *getNumIdle()* nos devuelve número de conexiones disponibles) en este momento, estando el pool inicializado:

```
if (BASICDATASOURCE.getNumIdle() == 0 &&
dataSourceInicializado)
throw new GenericaExcepcion(20);
```

Procedemos a continuación a analizar en detalle el método

```
public List<String> probarOcupacionPoolConexiones()
```

de la clase LibrosNegocio, en el que se desarrollan unas actuaciones exclusivamente enfocadas a mostrar al lector el comportamiento del pool:

1. Abrir tantas conexiones como sea posible. Para ello se implementa una estructura repetitiva *while()*. Declaramos el array

```
Connection[] conexiones = new Connection[1000]
```

a efectos de almacenar las conexiones que se vayan obteniendo. En cada repetición del bloque se solicita una nueva conexión, y si es concedida, su referencia se registra en el citado array

```
conexiones[numeroConexionesAbiertas] =  
ConexionBaseDatos.getInstancia().abrirConexion()
```

Cuando se intenta ir más allá del máximo disponible se lanza la excepción anteriormente mencionada, que al ser interceptada finaliza la ejecución del bloque repetitivo.

2. A efectos de comprobar que todas las conexiones obtenidas se encuentran disponibles para ser utilizadas, desde cada una de ellas, se lanza una SQL a la BD mediante la invocación al método

```
new  
LibrosDatos().consultarNumeroFilas(conexiones[i])  
.intValue()
```

3. En un intento de simulación de la dinámica habitual en el trabajo con pool de conexiones, las diez primeras conexiones que se abrieron, se van cerrando progresivamente. Conforme se va liberando cada una de ellas, se solicita otra nueva, y se invoca al mismo método de la capa de datos de la fase anterior para comprobar que es concedida y utilizable.
4. Finalmente, se cierran todas las conexiones.

En un intento de ceñirnos a las directrices del modelo de desarrollo de software arquitectura a tres capas, eludimos totalmente la interacción con el usuario en este método de la capa de negocio, en este caso concreto, la visualización de mensajes. Para ello se declara el

```
List<String> listaMensajes = new ArrayList();
```

en el que se almacenarán todos los mensajes que se pretenden visualizar al usuario para demostrar el funcionamiento de las actuaciones implementadas. Este objeto aglutinador de los mensajes, es devuelto asociado al identificador del método, para poder ser recorrido en la capa de presentación y proceder a la visualización de los mensajes. Exponemos a continuación la visualización en la consola de salida de dichos mensajes:

**PRUEBA DE TODAS LAS CONEXIONES QUE SE PUEDEN  
ABRIR EN EL POOL DADA LA CONFIGURACION QUE SE  
LE HA ESTABLECIDO**

```
Abierta conexión del pool : 0
Abierta conexión del pool : 1
Abierta conexión del pool : 2
. . .
Abierta conexión del pool : 48
Abierta conexión del pool : 49
Se han podido abrir : 50 conexiones
```

**Al intentar abrir nueva conexión en el pool,  
se ha interceptado excepcion asociada a que  
todas las conexiones están ocupadas**

Desde la conexión 0 lanzamos la SQL obteniendo el número de filas de la tabla : 36

Desde la conexión 1 lanzamos la SQL obteniendo el número de filas de la tabla : 36

Desde la conexión 2 lanzamos la SQL obteniendo el número de filas de la tabla : 36

. . .

Desde la conexión 48 lanzamos la SQL obteniendo el número de filas de la tabla : 36

Desde la conexión 49 lanzamos la SQL obteniendo el número de filas de la tabla : 36

**PROCEDEMOS A CERRAR CONEXIONES Y A  
REUTILIZARLAS ABRIENDO UNA CADA VEZ QUE SE  
CIERRA OTRA**

Se ha cerrado la conexión del pool : 0

Se ha reutilizado la conexión del pool : 0

Desde la conexión 0 lanzamos la SQL obteniendo el número de filas de la tabla : 36

Se ha cerrado la conexión del pool : 1

Se ha reutilizado la conexión del pool : 1

Desde la conexión 1 lanzamos la SQL obteniendo el número de filas de la tabla : 36

Se ha cerrado la conexión del pool : 2

Se ha reutilizado la conexión del pool : 2

Desde la conexión 2 lanzamos la SQL obteniendo el número de filas de la tabla : 36

Se ha cerrado la conexión del pool : 3

Se ha reutilizado la conexión del pool : 3

Desde la conexión 3 lanzamos la SQL obteniendo el número de filas de la tabla : 36

. . .

Se ha cerrado la conexión del pool : 9

Se ha reutilizado la conexión del pool : 9

Desde la conexión 9 lanzamos la SQL obteniendo el número de filas de la tabla : 36

**PROCEDEMOS AL CIERRE DE TODAS LAS CONEXIONES  
ABIERTAS**

Se ha cerrado la conexión del pool : 0  
Se ha cerrado la conexión del pool : 1  
Se ha cerrado la conexión del pool : 2  
. . .  
Se ha cerrado la conexión del pool : 48  
**Se ha cerrado la conexión del pool : 49**

Bloque 2º

**SWING**

# 7

## GENERALIDADES

Swing es el API de java que nos permite desarrollar aplicaciones Java en que la interacción con el usuario sea implementada mediante interfaz gráfica de usuario (GUI). Las primeras versiones de Java solamente disponían de AWT (Abstract Window Toolkit). Posteriormente se incorporó Swing, “asentado” sobre AWT.

El planteamiento del autor en este bloque del libro es centrarse en los aspectos eminentemente funcionales asociados tanto a los componentes, como a la gestión de eventos. Todo aquello relacionado con la apariencia y la estética de los componentes y las vistas, es abordado mínimamente a efectos de que los mecanismos asociados a la funcionalidad no queden “diluidos” con interminables líneas de código que necesariamente habría que aportar si se quisiera también tratar la apariencia y estética de la aplicación. Consideramos que el aspecto final de una aplicación, aunque importante como “retoque” final a presentar al usuario, debe tener un carácter secundario. Además, todas las cuestiones asociadas con el aspecto son mucho más sencillas y triviales; por no mencionar, que prácticamente todos los entornos de desarrollo permiten afrontar el diseño de las vistas de forma gráfica, generándose las líneas de código asociadas a la estética de la aplicación, automáticamente. Estas son las razones por las

que las vistas de los ejemplos aportados pueden dar la impresión de “austeras”.

Se presentan inicialmente toda una serie de ejemplos **InterfazGraficaX** en que se va produciendo una progresiva introducción a todos las clases y mecanismos utilizados en Swing.

Los primeros ejemplos son muy elementales, pero en complejidad creciente. El objetivo es que todos ellos supongan una aproximación progresiva para el lector, para finalmente, presentar la aplicación **GestionLibros** en que se ofrece una solución que encajaría con las necesidades y funcionalidades de la realidad productiva, accediendo a Base de Datos mediante el API JDBC, tratado en la primera parte del libro, y estructurada acorde a las directrices del modelo de desarrollo de software arquitectura a tres capas. Se abordarán todos los detalles de dicha aplicación en su momento.

Para desarrollar una interfaz gráfica en Swing disponemos de:

- **CONTENEDORES :**

- **JFrame.** Es la clase que modela una ventana. A dicho componente le añadimos el resto de componentes o subventanas de la GUI. Una aplicación debería tener un único JFrame, actuando como ventana principal.
- **JPanel.** Esta clase actúa como subventana, constituyendo la unidad que constituirán cada una de las secciones en que hemos fraccionado el espacio de la ventana principal, en el caso de que ello fuese necesario. Al JPanel le añadimos

componentes al igual que podemos hacer directamente con el *JFrame*.

- **JDialog.** Remitiéndonos a lo que hemos comentado sobre el *JFrame* en relación a que una aplicación solamente debería tener uno. El *JDialog* nos ofrece la posibilidad de poder implementar más de una ventana, además de la principal, que, en ese caso, desempeñarían el papel de ventanas secundarias. Un *JDialog* actúa como ventana modal cuando utilizamos un constructor que disponga del correspondiente argumento *boolean* que habrá que establecer a true.
- **COMPONENTES :** Son los elementos que, añadidos a un contenedor, nos permiten la interacción con el usuario en la E/S de datos.
  - AbstractButton
    - JButton
    - JMenuItem
      - JMenu
    - JToggleButton
      - JCheckBox
      - JRadioButton
  - JComboBox
  - JFileChooser
  - JLabel
  - JList
  - JMenuBar
  - JOptionPane

- JProgressBar
  - JScrollPane
  - JSlider
  - JSplitPane
  - JTabbedPane
  - JTable
  - JTextComponent
    - JTextArea
    - JTextField
    - JPasswordField
  - JTree
- **LAYOUTS** : Constituyen los administradores de diseño que, aplicados a un contenedor, nos permiten establecer un criterio de distribución de los componentes del mismo. Utilizamos para ello el método *setLayout(LAYOUT)*.
    - **AUSENCIA DE LAYOUT** Se define mediante  
`setLayout( null )`

Si nos decantamos por esta opción, es necesario posicionar cada uno de los componentes concretando las coordenadas de posicionamiento mediante el método

```
componente.setBounds(coordenada X, coordenada Y,  
ancho, alto)
```

Es la variante utilizada en todos los ejercicios de este libro. La justificación de ello, tal y como ya se ha mencionado, es centrarnos en los aspectos meramente funcionales de la aplicación, y que estos no se vean “diluidos” con cuestiones asociadas a la apariencia de la aplicación. Solamente se hace uso de Layouts en el

ejemplo **InterfazGrafica12**, destinado en exclusiva a mostrar al lector la utilización de los mismos.

- **FlowLayout.** Dispone los elementos en fila.
  - **BorderLayout.** Proporciona una división de la ventana en cinco secciones, distribuidas cuatro de ellas en la periferia del contenedor, dimensionándose dichas secciones automáticamente de manera que puedan albergar a los componentes que en ellas se ubiquen. Una quinta sección ocupa el espacio central restante.
  - **GridLayout.** Distribuye los componentes a dos dimensiones, acomodando su tamaño para que ocupen todos ellos el mismo espacio.
  - **CardLayout.** Implica una superposición de los componentes que aglutina. Dichos componentes suelen ser generalmente *JPanel*. La superposición implica que en cada momento solamente uno de ellos puede estar visible. A la implementación de un menú mediante este administrador de diseño, se destina un ejercicio en exclusiva, concretamente **MenuJMenuBar**, además del ejercicio final integrador **GestionLibros**. En ésta aplicación, las opciones del menú son implementadas cada una de ellas por un *JPanel*, adecuadamente integrados todos ellos por el *CardLayout*.
- **EVENTOS :** Son el mecanismo que proporciona interactividad a una interfaz gráfica de usuario. Para mayor comprensión por parte del lector, inicialmente estableceremos símiles metafóricos, tales como el

mecanismo de estímulo-respuesta que observamos en la naturaleza, o el principio físico de acción-reacción. Trasladando los símiles expuestos a una interfaz gráfica, el estímulo o acción es la interacción que ejerce el usuario sobre un componente. El más representativo, pulsar un botón de la interfaz gráfica. Esta acción del usuario sobre el botón implica una respuesta por parte de la aplicación que comporta toda una serie de acciones. El ejemplo más representativo de ello es pulsar el botón “Aceptar” de un formulario de recogida de datos. En este caso, las acciones respuesta a pulsar el botón son recoger la información introducida por el usuario de los componentes de la interfaz, filtrar los que proceda, en su caso, y lanzar una SQL de inserción en una tabla de la Base de Datos. Pues bien, en este ejemplo el mecanismo asociado al evento es el encargado de automatizar a la pulsación del botón, las acciones subsiguientes citadas. El botón, tal vez el más representativo, no es, ni muchísimo menos el único componente de una interfaz de usuario que pueda disparar el evento. Existe toda una variedad de componentes o situaciones susceptibles de generar un evento, como iremos viendo a lo largo de los ejercicios ejemplos de este bloque del libro, y, obviamente una diversidad de tipos de evento, modelados cada uno de ellos por una clase específica.

Habiendo hecho la correspondiente introducción intuitiva al concepto de evento, procedamos a analizar con más detalle y rigor los mecanismos inherentes a todo ello. Java utiliza el llamado modelo de delegación de eventos. Este modelo consiste básicamente en que el objeto que genera o dispara el evento, no implementa las acciones asociadas al tratamiento de dicho evento, sino que “delega” dicha

responsabilidad en otro objeto. Encontrándonos, pues, con una dualidad de objetos el objeto fuente del evento, y el objeto oyente que se encuentra a la espera o escucha de que se produzca el evento desde el objeto fuente, también llamado “escucha” o “listener”. A su vez, el evento es modelado por un objeto cuya referencia recibe el método del objeto escucha encargado de su tratamiento.

Para mejor comprensión de todos estos conceptos por parte del lector expondremos, como ejemplo representativo, el código que implementa el registro de escucha de eventos de un *JButton*, concretamente el referenciado desde **jButtonContar**, presente en los ejercicios ejemplo **InterfazGrafica1**, **InterfazGrafica2**, e **InterfazGrafica3** (se aporta el código de estas aplicaciones, al igual que el del resto de ejemplos más adelante). Aunque nos centremos en estos momentos en un tipo concreto de evento, el producido al pulsar el botón, a lo largo de los ejercicios de esta parte del libro irán apareciendo otros tipos de eventos y componentes que los disparan. Sugerimos al lector que consulte el anexo de la última parte del libro en que aparece un cuadro con los diferentes tipos de eventos y los componentes origen de los mismos.

Volvamos a los ejemplos **InterfazGrafica1**, **InterfazGrafica2**, e **InterfazGrafica3**. Podrá observar el lector al ejecutar cada uno de los tres ejercicios ejemplos, que todos ellos hacen exactamente lo mismo al pulsarse dicho botón: incrementar una variable y presentar el valor en una etiqueta. Es más, las tres aplicaciones presentan la misma apariencia. Pero en cada una de ellas la clase escucha del evento supone una implementación diferente respecto a las demás. Procedamos al detalle. Los pasos a

seguir en la configuración del *JButton* son los mismos que los que seguimos en la mayoría de los componentes:

- Instanciar el componente
- Establecer posicionamiento y dimensión (si hemos escogido ausencia de Layout, cuestión comentado con anterioridad)
- Añadir el componente al contenedor en que será aglutinado.
- Registro del componente con la clase o clases (puede ser más de una) que actuarán de escucha de los eventos del componente en cuestión.

- **INSTANCIAR COMPONENTE :**

```
 JButton jButtonContar = new JButton("Contar");
```

En **InterfazGrafica3** implementamos la instanciación:

```
componentes.setjButtonContar(new  
JButton("Contar"));
```

Comentaremos en breve las razones por las que lo hacerlo así.

- **ESTABLECER POSICION Y DIMENSIONES DEL COMPONENTE:**

```
jButtonContar.setBounds(100,300,100,40);
```

y en **InterfazGrafica3**:

```
componentes.getjButtonContar().setBounds(100,300,1  
00,40);
```

- **AÑADIR COMPONENTE AL CONTENEDOR EN QUE ES AGLUTINADO :**

```
add(jButtonContar);
```

y en **InterfazGrafica3**:

```
add(componentes.get jButtonContar());
```

- **REGISTRO DEL COMPONENTE CON LA CLASE ESCUCHA CORRESPONDIENTE :**

Es en este aspecto donde estriban las diferencias en cada una de las tres aplicaciones. Estudiaremos con detalle el despliegue de clases y métodos que intervienen en cada una de las tres configuraciones, representadas por cada una de las tres aplicaciones mencionadas, **InterfazGrafica1**, **InterfazGrafica2**, e **InterfazGrafica3**.

## **INTERFAZGRAFICA1**

Registraremos la escucha del *JButton*:

```
jButtonContar.addActionListener(listenerJButtonContar)  
;
```

transfiriendo al método la referencia a un objeto que actúa como clase de escucha, en este caso, habiendo previamente definido dicha clase de escucha:

```
ActionListener listenerJButtonContar = new  
ActionListener () {  
public void actionPerformed(ActionEvent e) {  
contador ++;  
jLabelVisualizaContador.setText("Número de  
pulsaciones: "+contador);  
}  
};
```

La pulsación del *JButton* genera un evento de la clase *ActionEvent*, que es recogido por una clase que implemente la interface *ActionListener*. Esta clase obliga a la implementación del método

```
actionPerformed(ActionEvent e)
```

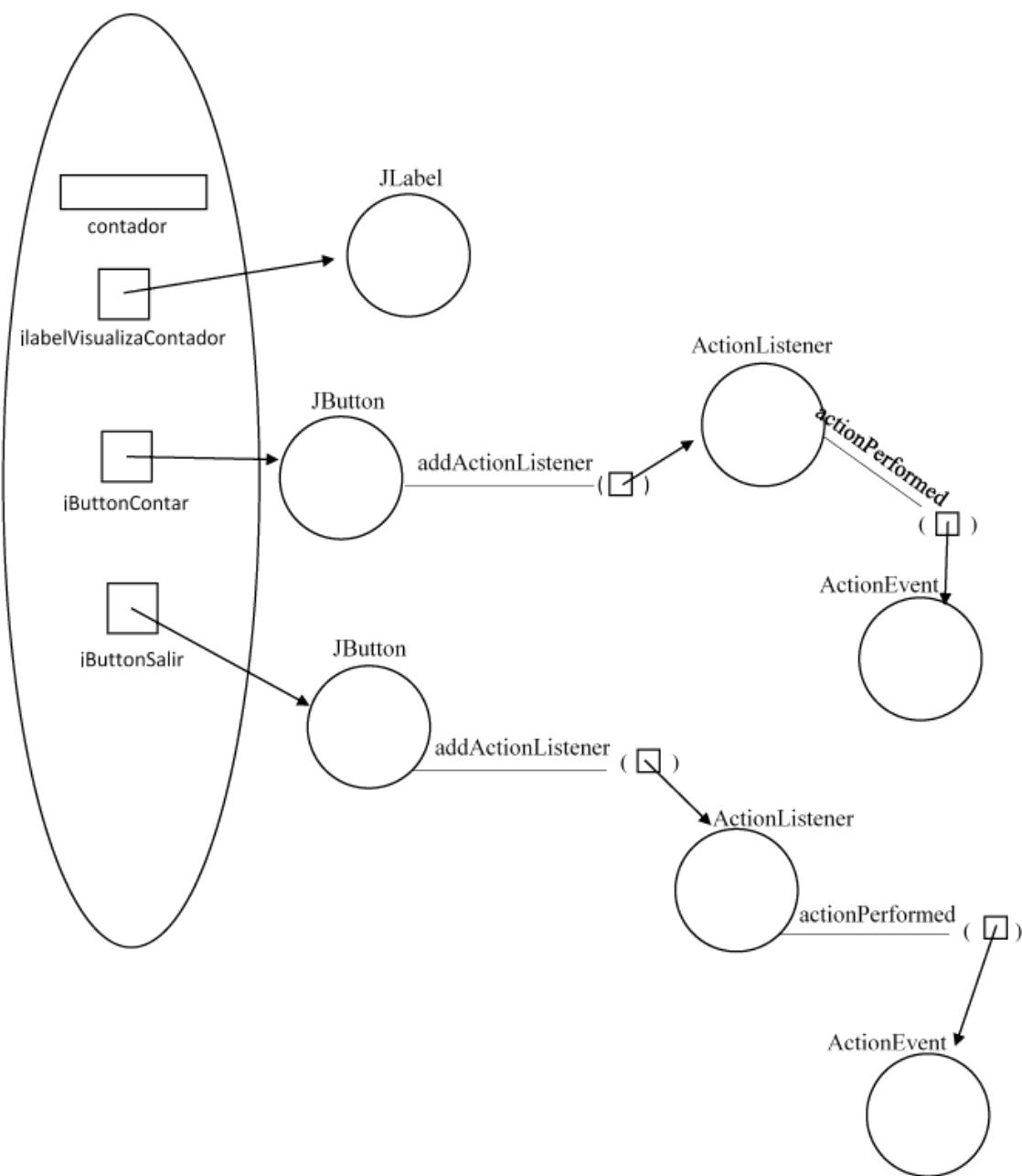
que es el que se ejecuta cuando dicha clase escucha recibe el evento generado.

También podemos proceder en el registro y definición del Listener de forma abreviada:

```
jButtonContar.addActionListener(new ActionListener ()  
{  
    public void actionPerformed(ActionEvent e) {  
        contador++;  
        jLabelVisualizaContador.setText("Número de  
        pulsaciones: "+contador);  
    }  
});
```

El esquema correspondiente a la configuración utilizada en el ejemplo **InterfazGrafica1**:

JFRAME  
**InterfazGrafica1**



## INTERFAZGRAFICA2

El procedimiento que aplicamos en el registro del *JButton* con la clase escucha del evento es:

```
jButtonSalir.addActionListener(this);
```

Deducirá el lector por la referencia transferida al método, que es el propio contenedor el que actúa como clase de escucha. Para ello es necesario que la clase en cuestión, el contenedor en este caso, implemente la correspondiente interface a la escucha de eventos de *JButton*:

```
implements ActionListener
```

Por lo que la clase que se erige como escucha, al asumir dicho compromiso, se ve obligada a implementar el método

```
public void actionPerformed(ActionEvent e)
```

en el que concretamos las acciones concretas a realizar al haberse disparado el evento en el *JButton*.

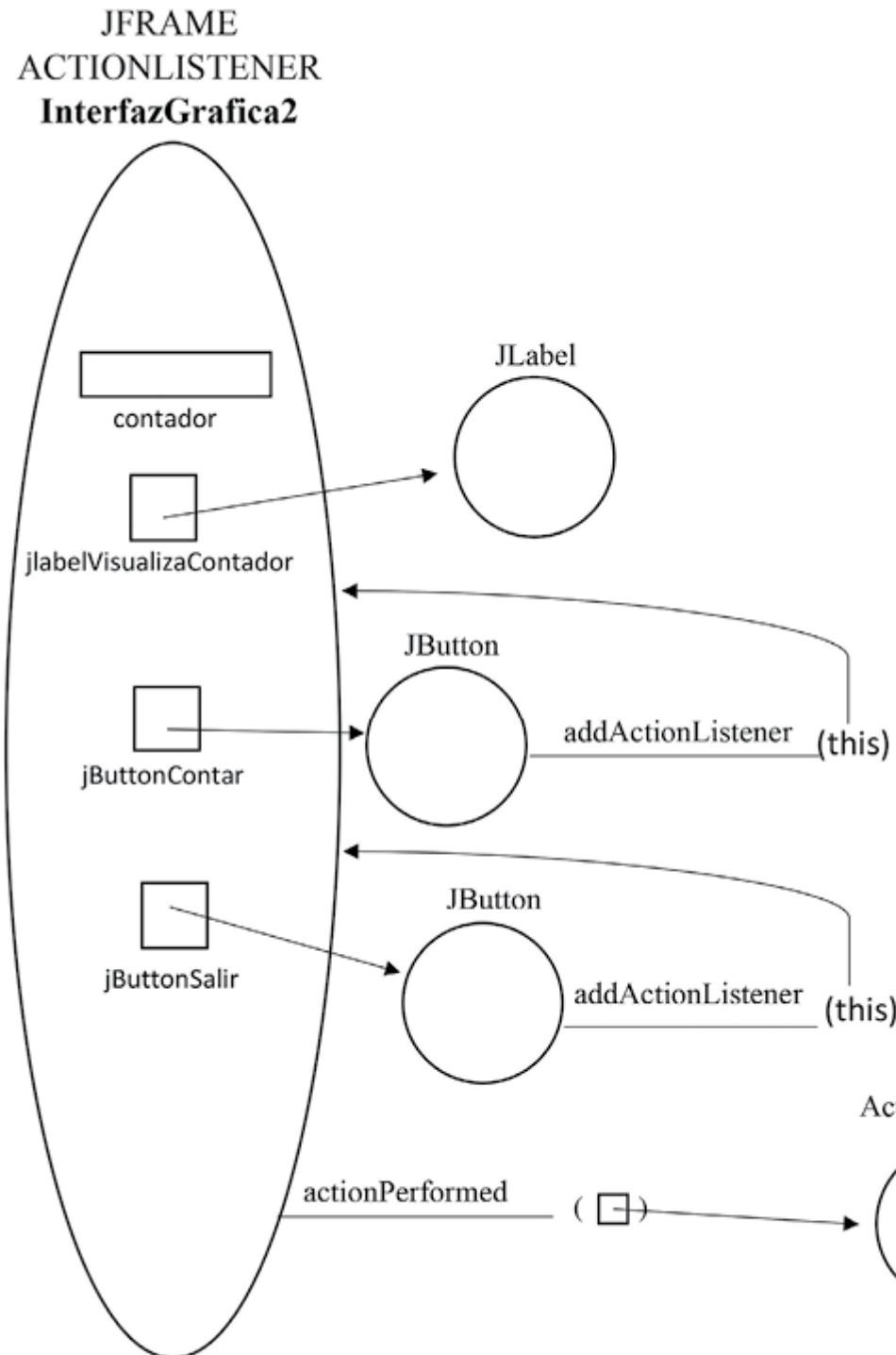
Al igual que *actionPerformed()*, todos los métodos que implementan las clases de escucha, reciben como argumento una referencia al objeto que modela el evento, y que encapsula toda la información concerniente al mismo. Para las escuchas *ActionListener*, la clase que modela el objeto en cuestión es *ActionEvent*.

En este ejercicio ejemplo **InterfazGrafica2**, ya nos encontramos con la necesidad de recurrir a la información que encapsula el objeto *ActionEvent*, concretamente, la necesidad de detectar cuál ha sido el componente fuente del evento, dado que son dos los *JButton* registrados con la misma clase escucha, y cada uno requiere un tratamiento diferente. Así pues, obtenemos la información de cuál ha sido el componente fuente mediante el método:

```
JButton jButton = (JButton) e.getSource();
```

para posteriormente proceder a un tratamiento diferenciado, tal y como se observa en él código.

El esquema de la configuración aplicada a este ejemplo es:



El lector podrá empezar ya a vislumbrar que nos podemos decantar por una de las dos posibles tendencias en el tratamiento

de eventos:

- **Dispersar** el tratamiento de eventos a una clase escucha diferente para cada uno de los componentes. Es el tipo de implementación que nos encontramos en **InterfazGrafica1**.
- **Centralizar** a una sola clase escucha el tratamiento de eventos de todos los componentes. Es el caso de **InterfazGrafica2**, **InterfazGrafica3**, y el resto de ejercicios del libro. Esta opción tiene como primer inconveniente la necesidad de diferenciar el componente fuente del evento, pero a su vez las ventajas inherentes a la centralización, que nos permitirán alcanzar configuraciones como la de la aplicación ejemplo **GestionLibros**, que ya se comentará en su momento

En relación con todo ello, la pretensión del autor es mostrar al lector todas las posibilidades al respecto, y dotarle de criterio para que, finalmente sea él mismo el que tome la decisión de decantarse por una tendencia u otra en la aplicación en sus propios desarrollos en función de las circunstancias concretas de cada caso.

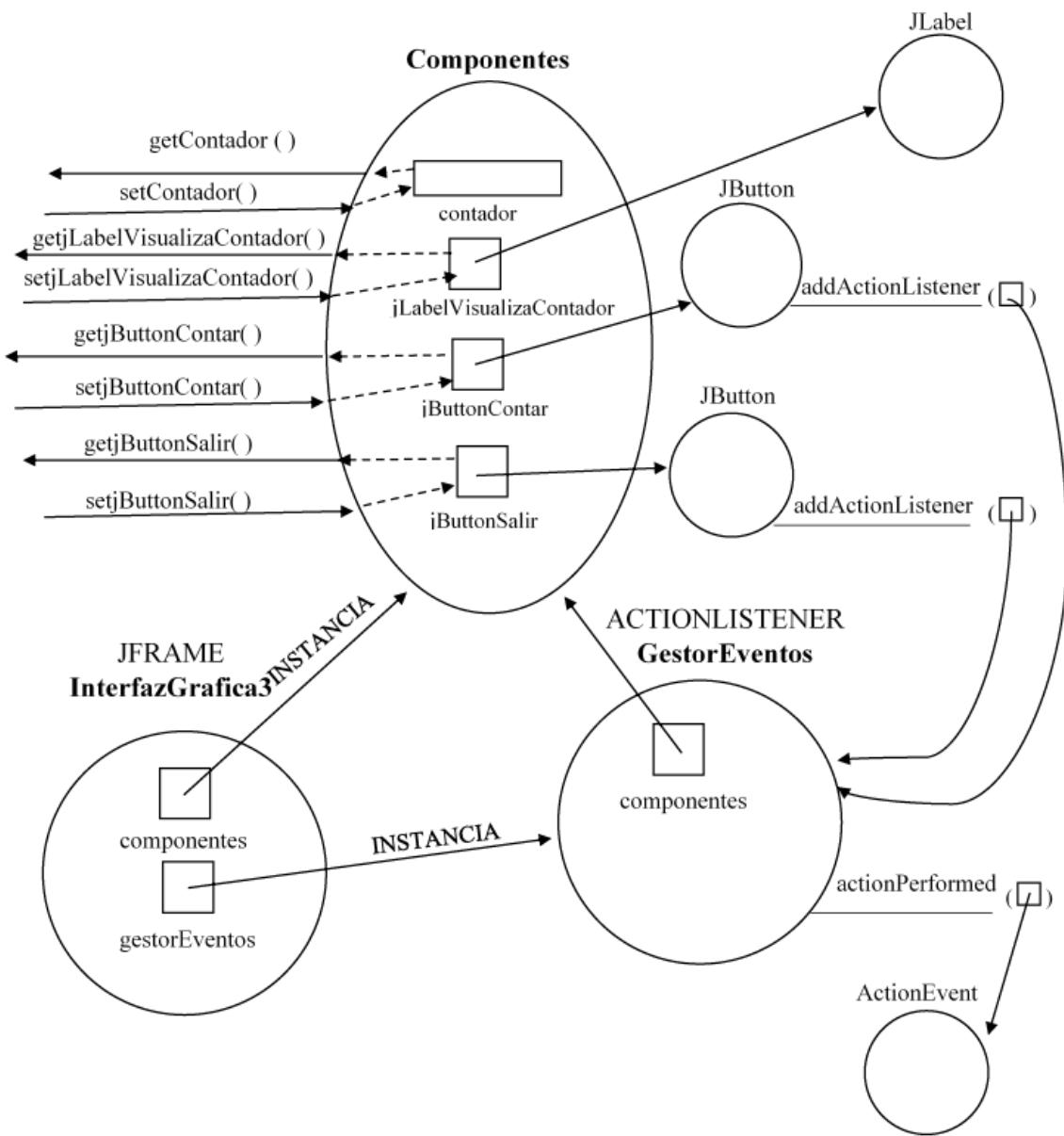
## **INTERFAZGRAFICA3**

En lo que a la dispersión o centralización del tratamiento de eventos se refiere, la configuración utilizada en **InterfazGrafica3** es la misma que en **InterfazGrafica2**, es decir, tendencia a la centralización. La diferencia entre ambas es que en **InterfazGrafica3** “externalizamos” la gestión de eventos a una clase exclusiva a tal fin, **GestorEventos**, como su propio nombre

indica. Encontrándonos ahora con la necesidad de invocar a métodos de los componentes involucrados en la gestión del evento:

```
componentes.getjButtonContar()  
componentes.getContador()  
componentes.setContador(contador)  
componentes.getjLabelVisualizaContador().setText("Número de pulsaciones: "+componentes.getContador())
```

desde una clase diferente (**GestorEventos**) a aquella en que se han declarado las referencias a dichos componentes (el propio contenedor **InterfazGrafica3**). Para solventar esta cuestión podemos optar por varias soluciones diferentes. En esta aplicación y en el resto de aplicaciones ejemplo del libro nos decantamos por arbitrar una clase destinada única y exclusivamente a actuar como “repositorio”, lugar centralizado de las referencias de todos los componentes cuyos métodos requieran ser accedidos desde objetos diferentes de la aplicación. Será la clase Componentes la que se encargue de tal cometido. Realmente, dicha clase, podríamos decir que actúa como clase encapsuladora de todas las referencias de todos los componentes de la aplicación que requieran un acceso “disperso”, valga la expresión, es decir, desde diversas clases de la aplicación. Aunque finalmente, la instanciación de los componentes se produce desde un método del contenedor que los aglutina, en consecuencia, se añaden a dicho contenedor. Recurrimos nuevamente a un esquema, que como siempre, la perspectiva gráfica nos ayudará a una mejor comprensión de todo ello.



Con estos tres ejercicios diferentes en su configuración e implementación, pero cuya funcionalidad es exactamente la misma, pretendemos aportar al lector diferentes perspectivas y soluciones a la gestión del tratamiento de eventos. De todas ellas, la primera es la más extendida y generalizada, pero las otras dos nos aportan la posibilidad de centralizar todo el tratamiento de eventos que se produzcan en todos los componentes de la aplicación. Adoptaremos la tercera versión en el resto de

ejemplos del libro, tal vez la más insólita, y también cuestionada en determinados círculos, pero es la que más nos permitirá aproximarnos, como ya hemos comentado, a la solución que aplicaremos al tratamiento de eventos en la aplicación final **GestionLibros**. En dicha aplicación pretendemos implementar en Swing una aproximación del patrón de diseño Front Controller, y para ello nos vemos obligados, irremisiblemente a la centralización, con las indudables ventajas que ello nos comportará y que analizaremos con detalle en su momento.

Todos los componentes enumerados en el esquema inicial de este bloque, así como los contenedores, y eventos aparecen a lo largo de los ejercicios de ese libro, algunos de ellos en repetidas ocasiones. El análisis con detalle de cada uno de dichos ejercicios proporcionará al lector la oportunidad de conocer los mecanismos inherentes a dichos contenedores, componentes y eventos. Así pues, procederemos a continuación a realizar un análisis detallado de cada uno de dichos ejercicios ejemplo, presentando el código de todas las clases de cada uno de ellos, y comentando los aspectos más relevantes.

# 8

## APLICACIÓN INTERFAZGRAFICA1

Esta aplicación solamente contempla la interacción con el usuario mediante una interfaz gráfica en Swing, en consecuencia, solamente dispondría de un *package* que se correspondería con la capa de presentación integrante del modelo de desarrollo software arquitectura a tres capas. Esta misma estructura es aplicable a todas las aplicaciones ejemplo del bloque Swing de este libro, con excepción de las aplicaciones **VotacionPropuesta** y **GestionLibros**, que, por acceder a Bases de Datos, sí que se adecúan al citado modelo a tres capas. Otra aplicación aportada que también se estructura a tres capas es **PrediccionMeteorologicaJTree**, aplicación que, aunque no accede a Base de Datos, sí que implementa lecturas de ficheros XML, cuya presencia justifica dicha organización.

Así pues, el *package* de este ejemplo que equivaldría a la capa de presentación es:

- *package* **interfazgrafica1** aglutina solamente la clase:
  - InterfazGrafical

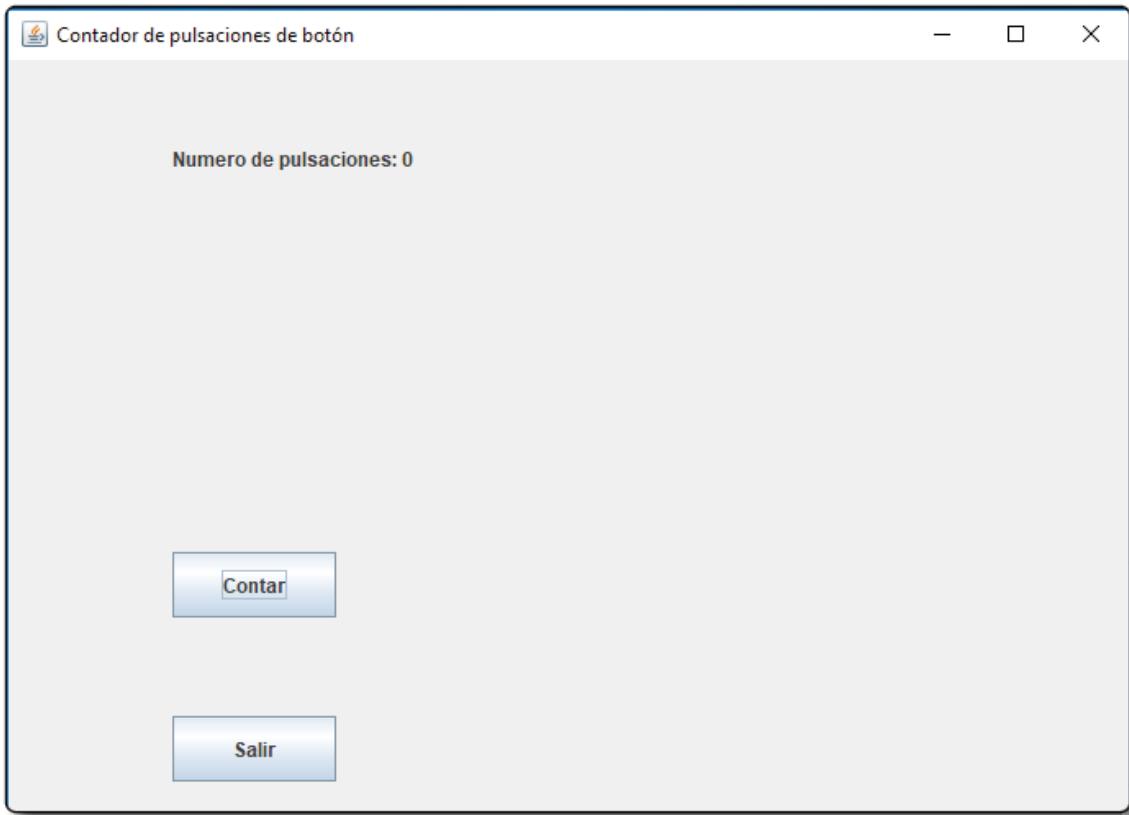
### **InterfazGrafica1**

```
package interfazgrafica1;
```

```
import java.awt.EventQueue;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
public class InterfazGrafica1 extends JFrame {
private int contador = 0;
public InterfazGrafica1() {
setSize(700,500);
setTitle("Contador de pulsaciones de botón");
ubicarComponentes();
setVisible(true);
}
private void ubicarComponentes() {
setLayout(null);
// CONFIGURACIÓN JLabel jLabelVisualizaContador
JLabel jLabelVisualizaContador = new JLabel("Número de
pulsaciones: "+contador);
jLabelVisualizaContador.setBounds(100,50,300,20);
add(jLabelVisualizaContador);
// CONFIGURACIÓN JButton jButtonContar
JButton jButtonContar = new JButton("Contar");
jButtonContar.setBounds(100,300,100,40);
/*
ActionListener listenerJButtonContar = new
ActionListener () {
public void actionPerformed(ActionEvent e) {
contador++;
jLabelVisualizaContador.setText("Número de
pulsaciones: "+contador);
}
};
jButtonContar.addActionListener(listenerJButtonContar)
; // REGISTRO DE ESCUCHA DE EVENTO DE BOTÓN
*/
jButtonContar.addActionListener(new ActionListener ()
```

```
{  
public void actionPerformed(ActionEvent e) {  
contador++;  
jLabelVisualizaContador.setText("Número de  
pulsaciones: "+contador);  
}  
});  
add(jButtonContar);  
// CONFIGURACIÓN JButton jButtonSalir  
JButton jButtonSalir = new JButton("Salir");  
jButtonSalir.setBounds(100,400,100,40);  
ActionListener listenerJButtonSalir = new  
ActionListener () {  
public void actionPerformed(ActionEvent e) {  
System.exit(0);  
}  
};  
jButtonSalir.addActionListener(listenerJButtonSalir);  
// REGISTRO DE ESCUCHA DE EVENTO DE BOTON  
add(jButtonSalir);  
}  
public static void main(String[] args) {  
EventQueue.invokeLater(new Runnable() {  
@Override  
public void run() {  
new InterfazGrafica1();  
}  
});  
}  
}
```

La ejecución de la aplicación presenta la siguiente ventana:



La ventana aglutina los siguientes componentes:

- *JButton* *jButtonContar* que cada vez que se pulsa, provoca un incremento del atributo contador.
- *JButton* *jButtonSalir*, cuya pulsación desemboca en la ejecución de  
`System.exit(0)`  
que comporta el cierre del programa y la finalización del programa.
- *JLabel* *jLabelVisualizaContador* que permite visualizar en todo momento el valor que va tomando el atributo contador.

Los detalles más relevantes sobre la gestión de eventos de este ejemplo, así como de InterfazGrafica2 e InterfazGrafica2 ya se han comentado con anterioridad, con la aportación de esquemas que proporcionan al lector perspectiva gráfica. Por ser esta la primera aplicación ejemplo que presentamos comentaremos las siguientes líneas de código que aparecerán en todos los ejemplos, que aunque triviales, no debemos dejarlos ignorados:

- Establecer tamaño ventana:

```
setSize(700,500);
```

- Establecer título ventana:

```
setTitle("Contador de pulsaciones de botón");
```

- Dejar visible la ventana:

```
setVisible(true);
```

- El inicio de la aplicación, implementado por el método *main*:

```
public static void main(String[] args) {  
    EventQueue.invokeLater(new Runnable() {  
        @Override  
        public void run() {  
            new InterfazGrafica1();  
        }  
    });  
}
```

para una adecuada gestión de la cola de eventos.

# 9

## APLICACIÓN INTERFAZGRAFICA2

La estructura que presenta es la misma que InterfazGrafica1:

- *package interfazgrafica2* aglutina solamente la clase:
  - InterfazGrafica2

### InterfazGrafica2

```
package interfazgrafica2;
import java.awt.EventQueue;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
public class InterfazGrafica2 extends JFrame
implements ActionListener {
private JLabel jLabelVisualizaContador;
private JButton jButtonContar;
private JButton jButtonSalir;
private int contador = 0;
public InterfazGrafica2() {
setSize(700,500);
setTitle("Contador de pulsaciones de botón");
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
ubicarComponentes();
```

```
setVisible(true);
}
private void ubicarComponentes() {
setLayout(null);
// CONFIGURACIÓN JLabel jLabelVisualizaContador
jLabelVisualizaContador = new JLabel("Número de pulsaciones: "+contador);
jLabelVisualizaContador.setBounds(100,50,300,20);
add(jLabelVisualizaContador);
// CONFIGURACIÓN JButton jButtonContar
jButtonContar = new JButton("Contar");
jButtonContar.setBounds(100,300,100,40);
jButtonContar.addActionListener(this); // REGISTRO DE ESCUCHA DE EVENTO DE BOTON
add(jButtonContar);
// CONFIGURACIÓN JButton jButtonSalir
jButtonSalir = new JButton("Salir");
jButtonSalir.setBounds(100,400,100,40);
jButtonSalir.addActionListener(this); // REGISTRO DE ESCUCHA DE EVENTO DE BOTON
add(jButtonSalir);
}
public void actionPerformed(ActionEvent e) {
JButton jButton = (JButton) e.getSource();
if (jButton == jButtonContar)
{ contador++;
jLabelVisualizaContador.setText("Número de pulsaciones: "+contador);
}
else
if (jButton == jButtonSalir)
System.exit(0);
}
public static void main(String[] args) {
EventQueue.invokeLater(new Runnable() {
@Override
public void run() {
```

```
new InterfazGrafica2();
}
});
}
}
```

La ventana obtenida al ejecutar la aplicación es exactamente la misma que la de InterfazGrafica1, los mismos componentes y la misma apariencia. Los detalles de la comparativa con InterfazGrafica1 en lo que a la gestión de eventos se refiere, que es donde radican las diferencias entre ambas, ya se comentaron en su momento. Una innovación en relación con InterfazGrafica1 es que el *JFrame* implementa

```
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)
```

lo cual permite cerrar la aplicación cuando se pulsa la cruz de la esquina superior derecha de la pantalla asociado al *JFrame*. En la anterior aplicación esto no se produce. Es un mecanismo redundante a la acción de pulsar el botón “Salir”.

# 10

## APLICACIÓN INTERFAZGRAFICA3

Comparativamente a los dos ejercicios anteriores, aporta dos clases adicionales en el mismo package:

- *package interfazgrafica3* aglutina las clases:
  - InterfazGrafica3
  - Componentes
  - GestorEventos

### InterfazGrafica3

```
package interfazgrafica3;
import java.awt.EventQueue;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
public class InterfazGrafica3 extends JFrame {
public InterfazGrafica3() {
setSize(700,500);
setTitle("Contador de pulsaciones de botón");
ubicarComponentes();
setVisible(true);
}
private void ubicarComponentes() {
setLayout(null);
```

```
Componentes componentes = new Componentes();
componentes.setContador(0);
GestorEventos gestorEventos = new
GestorEventos(componentes);
// CONFIGURACIÓN JLabel jLabelVisualizaContador
componentes.setjLabelVisualizaContador(new
JLabel("Número de pulsaciones:
"+componentes.getContador()));
componentes.getjLabelVisualizaContador().setBounds(100
,50,300,20);
add(componentes.getjLabelVisualizaContador());
// CONFIGURACIÓN JButton jButtonContar
componentes.setjButtonContar(new JButton("Contar"));
componentes.getjButtonContar().setBounds(100,300,100,4
0);
componentes.getjButtonContar().addActionListener(gesto
rEventos); // REGISTRO DE ESCUCHA DE EVENTO DE BOTON
add(componentes.getjButtonContar());
// CONFIGURACIÓN JButton jButtonSalir
componentes.setjButtonSalir(new JButton("Salir"));
componentes.getjButtonSalir().setBounds(100,400,100,40
);
componentes.getjButtonSalir().addActionListener(gestor
Eventos); // REGISTRO DE ESCUCHA DE EVENTO DE BOTON
add(componentes.getjButtonSalir());
}
public static void main(String[] args) {
EventQueue.invokeLater(new Runnable() {
@Override
public void run() {
new InterfazGrafica3();
}
});
}
```

## Componentes

```
package interfazgrafica3;
import javax.swing.JButton;
import javax.swing.JLabel;
public class Componentes {
private JLabel jLabelVisualizaContador;
private JButton jButtonContar;
private JButton jButtonSalir;
private int contador;
public JLabel getjLabelVisualizaContador() {
return jLabelVisualizaContador;
}
public void setjLabelVisualizaContador(JLabel
jLabelVisualizaContador) {
this.jLabelVisualizaContador =
jLabelVisualizaContador;
}
public JButton getjButtonContar() {
return jButtonContar;
}
public void setjButtonContar(JButton jButtonContar) {
this(jButtonContar = jButtonContar;
}
public JButton getjButtonSalir() {
return jButtonSalir;
}
public void setjButtonSalir(JButton jButtonSalir) {
this(jButtonSalir = jButtonSalir;
}
public int getContador() {
return contador;
}
public void setContador(int contador) {
this.contador = contador;
}
}
```

## GestorEventos

```
package interfazgrafica3;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
public class GestorEventos implements ActionListener {
private Componentes componentes;
public GestorEventos(Componentes componentes) {
this.componentes = componentes;
}
public void actionPerformed(ActionEvent e) {
JButton jButton = (JButton) e.getSource();
if (jButton == componentes.get jButtonContar())
{ int contador = componentes.getContador();
contador++;
componentes.setContador(contador);
componentes.get jLabelVisualizaContador().setText("Número de pulsaciones: "+componentes.getContador());
}
else
if (jButton == componentes.get jButtonSalir())
System.exit(0);
}
}
```

La ventana obtenida al ejecutar la aplicación sigue siendo exactamente la misma que la de las dos aplicaciones ejemplo predecesoras, los mismos componentes y la misma apariencia. Y los detalles de la comparativa con sus predecesoras en lo que a la gestión de eventos se refiere, también se comentaron en su momento. Con la nueva aportación de las clases **Componentes** y **GestorEventos**, obtenemos una nueva configuración para la gestión de eventos. Configuración ya tratada con anterioridad cuando se abordaron conceptos sobre eventos, y que se extenderá, como ya se mencionó a lo largo de los ejercicios del libro, con pequeños cambios en la aplicación final.

**GestionLibros.** Como se podrá observar en el esquema correspondiente a esta configuración, la secuencia de instanciaciones de clases que se producen son:

en clase **InterfazGrafica3** – método **ubicarComponentes()**

```
Componentes componentes = new Componentes();
```

y la referencia obtenida con esta instancia se transfiere al constructor de **GestorEventos**:

```
GestorEventos gestorEventos = new  
GestorEventos(componentes);
```

Y ya con esta referencia al objeto Componentes accedemos a los componentes de la aplicación desde cualquier “otro lugar” de la misma mediante los métodos “setters” y “getters”, como se podrá observar en los siguientes casos:

desde la clase **InterfazGrafica3** – método **ubicarComponentes()**

```
componentes.setContador(0);
```

desde la clase **GestorEventos** – método **actionPerformed()**

```
int contador = componentes.getContador();
```

desde la clase **InterfazGrafica3** – método **ubicarComponentes()**

```
componentes.setjButtonContar(new JButton("Contar"));
```

desde la clase **GestorEventos** – método **actionPerformed()**

```
if (jButton == componentes.getjButtonContar())  
{ . . . }
```

# 11

## APLICACIÓN INTERFAZGRAFICA4

Presenta la misma estructura que el ejercicio anterior. Las diferencias con el ejercicio anterior solamente radican en una línea añadida en InterfazGrafica4 en relación a su predecesora, que es la única línea de código que aportamos a continuación. Sí que aportamos la clase **GestorEventos** en su totalidad, dado que presenta mayor número de líneas añadidas comparativamente a su predecesora en la aplicación anterior.

- *package interfazgrafica4* aglutina las clases:
  - InterfazGrafica4
  - Componentes
  - GestorEventos

### InterfazGrafica4

```
package interfazgrafica4;
import java.awt.EventQueue;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
public class InterfazGrafica4 extends JFrame{
public InterfazGrafica4() {
setSize(700,500);
```

```
setTitle("Contador de pulsaciones de botón");
ubicarComponentes();
setVisible(true);
}
private void ubicarComponentes() {
setLayout(null);
Componentes componentes = new Componentes();
componentes.setContador(0);
GestorEventos gestorEventos = new
GestorEventos(componentes);
addWindowListener(gestorEventos); // REGISTRO DE
ESCUCHA DE EVENTOS DE VENTANA
// CONFIGURACIÓN JLabel jLabelVisualizaContador
componentes.setjLabelVisualizaContador(new
JLabel("Número de pulsaciones:
"+componentes.getContador()));
componentes.getjLabelVisualizaContador().setBounds(100
,50,300,20);
add(componentes.getjLabelVisualizaContador());
// CONFIGURACIÓN JButton jButtonContar
componentes.setjButtonContar(new JButton("Contar"));
componentes.getjButtonContar().setBounds(100,300,100,4
0);
componentes.getjButtonContar().addActionListener(gesto
rEventos); // REGISTRO DE ESCUCHA DE EVENTO DE BOTÓN
add(componentes.getjButtonContar());
// CONFIGURACIÓN JButton jButtonSalir
componentes.setjButtonSalir(new JButton("Salir"));
componentes.getjButtonSalir().setBounds(100,400,100,40
);
componentes.getjButtonSalir().addActionListener(gestor
Eventos); // REGISTRO DE ESCUCHA DE EVENTO DE BOTÓN
add(componentes.getjButtonSalir());
}
public static void main(String[] args) {
EventQueue.invokeLater(new Runnable() {
@Override
```

```
public void run() {  
    new InterfazGrafica4();  
}  
}  
}  
}
```

## GestorEventos

```
package interfazgrafica4;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.awt.event.WindowAdapter;  
import java.awt.event.WindowEvent;  
import javax.swing.JButton;  
public class GestorEventos extends WindowAdapter  
implements ActionListener {  
    private Componentes componentes;  
    public GestorEventos(Componentes componentes) {  
        this.componentes = componentes;  
    }  
    // MÉTODOS DE WindowAdapter  
    public void windowClosing(WindowEvent e) {  
        System.exit(0);  
    }  
    // MÉTODO DE ActionListener  
    public void actionPerformed(ActionEvent e) {  
        JButton jButton = (JButton) e.getSource();  
        if (jButton == componentes.get jButtonContar())  
        { int contador = componentes.getContador();  
            contador++;  
            componentes.setContador(contador);  
            componentes.get jLabelVisualizaContador().setText("Número de pulsaciones: "+componentes.getContador());  
        }  
        else  
        if (jButton == componentes.get jButtonSalir())
```

```
System.exit(0);
}
}
```

La ventana obtenida al ejecutar la aplicación sigue siendo exactamente la misma que la de las dos aplicaciones ejemplo predecesoras, los mismos componentes y la misma apariencia. La aportación novedosa en relación con el ejercicio anterior es la escucha de eventos de ventana, presente en la clase **GestorEventos** al aplicar a la definición de esta clase:

```
extends WindowAdapter
```

Lo cual se procede al registrarse el *JFrame* con esta clase escucha:

```
addWindowListener(gestorEventos);
```

La presencia en la clase escucha de eventos del método

```
public void windowClosing(WindowEvent e) {
System.exit(0);
}
```

da lugar a que cuando se pulse en la cruz del extremo superior derecho de la ventana (cierre de ventana), se ejecute el código que presenta el método, en este caso concreto:

```
System.exit(0);
```

que finaliza la ejecución del programa. Conseguimos el mismo efecto en **InterfazGrafica2** cuando al *JFrame* se le estableció:

```
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)
```

pero con la implementación del método *windowClosing(WindowEvent e)* el programador puede codificar en dicho método cuantas acciones crea oportunas asociadas al cierre de la aplicación. En la clase **GestorEventos** del ejercicio ejemplo InterfazGrafica6, en lugar de aplicar

`extends WindowAdapter`

la clase se erige en escucha de eventos de ventana mediante

`implements WindowListener`

Si el lector revisa el código de esta clase, presentada con posterioridad, podrá comprobar que además de implementar el método *windowClosing(WindowEvent e)*, que es el único que nos merece el interés en este caso, también nos vemos obligados a implementar, aunque vacíos, el resto de los métodos de la interface *WindowListener*:

- `windowActivated()`
- `windowDeactivated()`
- `windowClosed()`
- `windowIconified()`
- `windowDeiconified()`
- `windowOpened()`.

Como se podrá deducir, el sentido de la existencia de los Adapter correspondientes a un Listener, es implementar vacíos todos los métodos a que obliga la interface. La utilización del Adapter, en lugar del correspondiente Listener le permite al programador la posibilidad de implementar solamente los métodos de la interface de los que tiene necesidad, soslayando la definición vacía de los métodos que no necesita. En el caso de la clase **GestorEventos** de **InterfazGrafica6**, el Adapter utilizado es *MouseInputAdapter*, y para los eventos de ventana ha de aplicar *implements* de la interface *WindowListener*, dado que una clase solamente puede aplicar *extends* de una superclase.

# 12

## APLICACIÓN INTERFAZGRAFICA5

Seguimos manteniendo la misma estructura que en las últimas aplicaciones ejemplo:

- *package interfazgrafica5* aglutina las clases:
  - InterfazGrafica5
  - Componentes
  - GestorEventos

### InterfazGrafica5

```
package interfazgrafica5;
import java.awt.EventQueue;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
public class InterfazGrafica5 extends JFrame{
public InterfazGrafica5() {
setSize(700,500);
setTitle("Contador de pulsaciones de botón y ratón - Visualización posición del ratón");
ubicarComponentes();
setVisible(true);
}
private void ubicarComponentes() {
```

```
setLayout(null);
Componentes componentes = new Componentes();
componentes.setContadorPulsacionesBoton(0);
componentes.setContadorPulsacionesRaton(0);
GestorEventos gestorEventos = new
GestorEventos(componentes);
addWindowListener(gestorEventos); // REGISTRO DE
ESCUCHA DE EVENTOS DE VENTANA
addMouseListener(gestorEventos); // REGISTRO DE
ESCUCHA DE EVENTO CLICK DE RATON
addMouseMotionListener(gestorEventos); // REGISTRO DE
ESCUCHA DE EVENTO MOVIMIENTO DE RATON
// CONFIGURACIÓN JLabel jLabelVisualizaContador
componentes.setjLabelVisualizaPulsacionesBoton(new
JLabel("Número de pulsaciones botón:
"+componentes.getContadorPulsacionesBoton()));
componentes.getjLabelVisualizaPulsacionesBoton().setBo
unds(100,50,350,20);
add(componentes.getjLabelVisualizaPulsacionesBoton());
// CONFIGURACIÓN JLabel
jLabelVisualizaPulsacionesRaton
componentes.setjLabelVisualizaPulsacionesRaton(new
JLabel("Número de pulsaciones ratón:
"+componentes.getContadorPulsacionesRaton()));
componentes.getjLabelVisualizaPulsacionesRaton().setBo
unds(100,150,350,20);
add(componentes.getjLabelVisualizaPulsacionesRaton());
// CONFIGURACIÓN JLabel jLabelVisualizaPosicionRaton
componentes.setjLabelVisualizaPosicionRaton(new
JLabel("Posición ratón: "));
componentes.getjLabelVisualizaPosicionRaton().setBound
s(100,250,400,20);
add(componentes.getjLabelVisualizaPosicionRaton());
// CONFIGURACIÓN JButton jButtonContar
componentes.setjButtonContar(new JButton("Contar"));
componentes.getjButtonContar().setBounds(100,350,100,4
0);
```

```
componentes.get jButtonContar().addActionListener(gesto  
rEventos); // REGISTRO DE ESCUCHA DE EVENTO DE BOTON  
add(componentes.get jButtonContar());  
}  
public static void main(String[] args) {  
EventQueue.invokeLater(new Runnable() {  
@Override  
public void run() {  
new InterfazGrafica5();  
}  
});  
}  
}  
}
```

## Componentes

```
package interfazgrafica5;  
import javax.swing.JButton;  
import javax.swing.JLabel;  
public class Componentes {  
private JLabel jLabelVisualizaPulsacionesBoton;  
private JLabel jLabelVisualizaPulsacionesRaton;  
private JLabel jLabelVisualizaPosicionRaton;  
private JButton jButtonContar;  
private int contadorPulsacionesBoton;  
private int contadorPulsacionesRaton;  
public JLabel getjLabelVisualizaPulsacionesBoton() {  
return jLabelVisualizaPulsacionesBoton;  
}  
public void setjLabelVisualizaPulsacionesBoton(JLabel  
jLabelVisualizaPulsacionesBoton) {  
this.jLabelVisualizaPulsacionesBoton =  
jLabelVisualizaPulsacionesBoton;  
}  
public JLabel getjLabelVisualizaPulsacionesRaton() {  
return jLabelVisualizaPulsacionesRaton;  
}
```

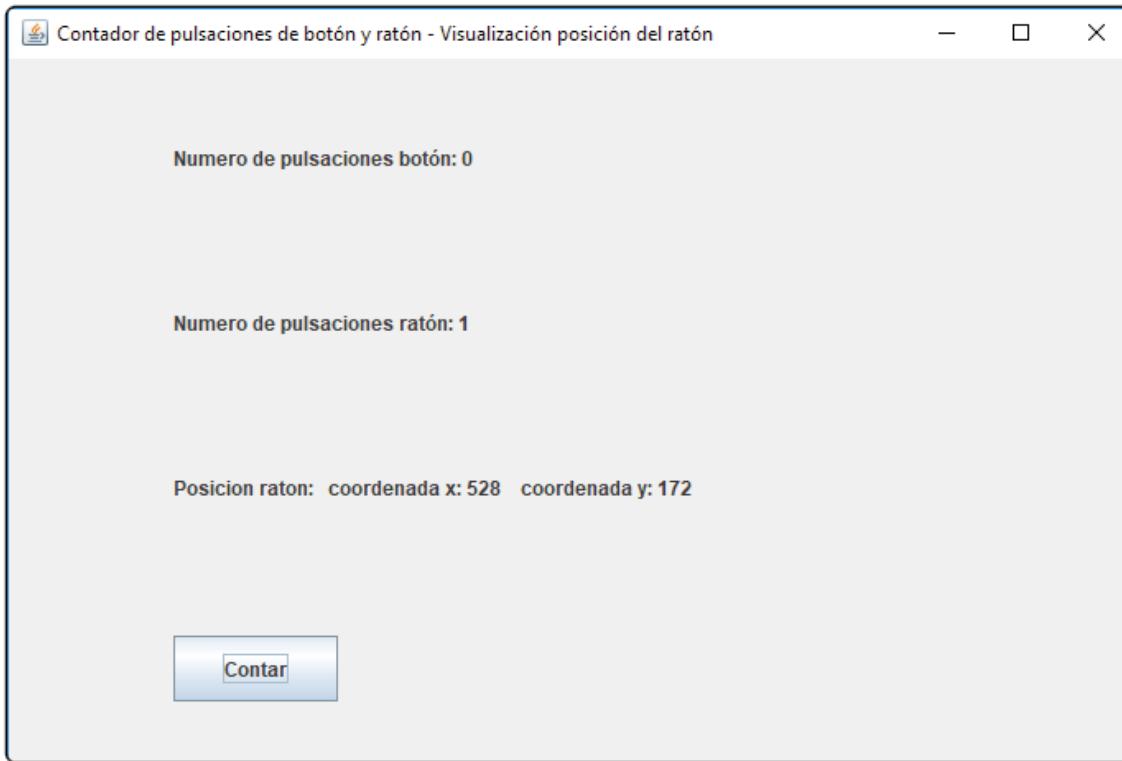
```
public void setjLabelVisualizaPulsacionesRaton(JLabel
jLabelVisualizaPulsacionesRaton) {
this.jLabelVisualizaPulsacionesRaton =
jLabelVisualizaPulsacionesRaton;
}
public JLabel getjLabelVisualizaPosicionRaton() {
return jLabelVisualizaPosicionRaton;
}
public void setjLabelVisualizaPosicionRaton(JLabel
jLabelVisualizaPosicionRaton) {
this.jLabelVisualizaPosicionRaton =
jLabelVisualizaPosicionRaton;
}
public JButton getjButtonContar() {
return jButtonContar;
}
public void setjButtonContar(JButton jButtonContar) {
this(jButtonContar = jButtonContar;
}
public int getContadorPulsacionesBoton() {
return contadorPulsacionesBoton;
}
public void setContadorPulsacionesBoton(int
contadorPulsacionesBoton) {
this.contadorPulsacionesBoton =
contadorPulsacionesBoton;
}
public int getContadorPulsacionesRaton() {
return contadorPulsacionesRaton;
}
public void setContadorPulsacionesRaton(int
contadorPulsacionesRaton) {
this.contadorPulsacionesRaton =
contadorPulsacionesRaton;
}
}
```

## GestorEventos

```
package interfazgrafica5;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.event.MouseInputListener;
public class GestorEventos extends WindowAdapter
implements ActionListener, MouseInputListener {
private Componentes componentes;
public GestorEventos(Componentes componentes) {
this.componentes = componentes;
}
// MÉTODOS DE WindowAdapter
public void windowClosing(WindowEvent e) {
System.exit(0);
}
// MÉTODO DE ActionListener
public void actionPerformed(ActionEvent e) {
int contador =
componentes.getContadorPulsacionesBoton();
contador++;
componentes.setContadorPulsacionesBoton(contador);
componentes.getjLabelVisualizaPulsacionesBoton().setText(
“Número de pulsaciones botón:
“+componentes.getContadorPulsacionesBoton());
}
// MÉTODOS DE MouseInputListener
public void mouseMoved(MouseEvent e) {
componentes.getjLabelVisualizaPosicionRaton().setText(
“Posición ratón: coordenada x: “+e.getX())+” coordenada
y: “+e.getY());
}
public void mouseClicked(MouseEvent e) {
int contador =
componentes.getContadorPulsacionesRaton();
```

```
contador++;
componentes.setContadorPulsacionesRaton(contador);
componentes.getjLabelVisualizaPulsacionesRaton().setText("Número de pulsaciones ratón:
"+componentes.getContadorPulsacionesRaton());
}
public void mousePressed(MouseEvent e) {
}
public void mouseReleased(MouseEvent e) {
}
public void mouseEntered(MouseEvent e) {
}
public void mouseExited(MouseEvent e) {
}
public void mouseDragged(MouseEvent e) {
}
}
```

La ejecución de la aplicación presenta la siguiente ventana:



Partiendo del supuesto inicial, seguimos “creciendo” en la gestión de eventos, encontrándonos en la clase **GestionEventos** las siguientes escuchas:

- Eventos de ventana mediante la implementación de acciones en el método *windowClosing()*.
- Eventos de *JButton* mediante la implementación de acciones en el método *actionPerformed()*.
- Eventos de movimiento de ratón mediante la implementación de acciones en el método *mouseMoved()*.
- Eventos de click de ratón mediante la implementación de acciones en el método *mouseClicked()*.

Para ello, adicionalmente respecto al ejemplo anterior, la clase **GestorEventos**, que centraliza todas las escuchas debe implementar la *interface* *MouseListener*, definiendo implementación de acciones en los métodos *mouseMoved()* y *mouseClicked()*, y vacíos el resto de métodos de la *interface*.

En este caso, es el *JFrame* el que se registra con las escuchas de eventos de ratón:

```
addMouseListener(gestorEventos); // REGISTRO DE  
ESCUCHA DE EVENTO CLICK DE RATON  
addMouseMotionListener(gestorEventos); // REGISTRO DE  
ESCUCHA DE EVENTO MOVIMIENTO DE RATON
```

La gestión que hacemos en ese ejemplo de dichos eventos es análoga a la que hasta ahora se ha hecho con las pulsaciones del botón: el click del ratón sobre el *JFrame*, incrementa un contador arbitrado a tal efecto, y visualizando el valor que va tomando en cada momento sobre una etiqueta. Otra etiqueta nos permite visualizar en todo momento las nuevas coordenadas que va

tomando la posición del ratón en su movimiento sobre el *JFrame*.

En el método *actionPerformed()* no nos encontramos con la necesidad de detectar el componente fuente del evento, tal y como hemos hecho en los ejemplos anteriores, porque solamente un botón se registra con la escucha *ActionListener*.

Otra solución alternativa a la que supone la centralización de escucha de eventos en la clase **GestorEventos** de InterfazGrafica5, podría haber consistido en “dispersar” dicha gestión de eventos en tres clases diferentes:

- public class GestorEventosBoton implements ActionListener
- public class GestorEventosVentana extends WindowAdapter
- public class GestorEventosRaton extends MouseInputAdapter

Ello hubiese eliminado la necesidad de definir métodos vacíos de implementación. Otra variante de esta configuración de gestión de eventos “dispersa”, podría consistir en prescindir de la primera clase, de las tres expuestas, añadiendo la escucha *ActionListener* a cualquiera de las otras dos, quedando:

- public class GestorEventosVentanaYBoton extends WindowAdapter implements ActionListener
- public class GestorEventosRaton extends MouseInputAdapter

sin necesidad de implementar métodos vacíos dado que la *interface ActionListener*, solamente exige la implementación del método *actionPerformed()*.

# 13

## APLICACIÓN INTERFAZGRAFICA6

Una nueva clase se añade a la estructura de las últimas aplicaciones ejemplo presentadas:

- *package interfazgrafica6* aglutina las clases:
  - InterfazGrafica6
  - Componentes
  - GestorEventos
  - AreaDibujo
  - JButtonPersonalizado

### InterfazGrafica6

```
package interfazgrafica6;
import java.awt.Color;
import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JLabel;
public class InterfazGrafica6 extends JFrame {
public InterfazGrafica6() {
setSize(700,590);
setTitle("Dibujo líneas - Click ratón en inicio,
soltar en final");
ubicarComponentes();
```

```
setVisible(true);
}
private void ubicarComponentes() {
setLayout(null);
Componentes componentes = new Componentes();
GestorEventos gestorEventos = new
GestorEventos(componentes);
addWindowListener(gestorEventos); // REGISTRO DE
ESCUCHA DE EVENTOS DE VENTANA
componentes.setAreaDibujo(new AreaDibujo());
componentes.getAreaDibujo().setBounds(15,50,650,395);
componentes.getAreaDibujo().setBackground(Color.green)
;
componentes.getAreaDibujo().addMouseListener(gestorEve
ntos); // REGISTRO DE ESCUCHA DE EVENTOS DE RATON
componentes.getAreaDibujo().addMouseMotionListener(ges
torEventos); // REGISTRO DE ESCUCHA DE EVENTO
MOVIMIENTO DE RATON
this.getContentPane().add(componentes.getAreaDibujo())
;
// CONFIGURACIÓN JLabel jLabelVisualizaPosicionRaton
componentes.setjLabelVisualizaPosicionRaton(new
JLabel("Posicion raton: "));
componentes.getjLabelVisualizaPosicionRaton().setBound
s(50,10,400,20);
add(componentes.getjLabelVisualizaPosicionRaton());
// CONFIGURACIÓN JLabel jLabelVisualizaContador
componentes.setjLabelVisualizaPulsacionesBoton(new
JLabel("Numero de pulsaciones botón:
"+componentes.getContadorPulsacionesBoton()));
componentes.getjLabelVisualizaPulsacionesBoton().setBo
unds(220,480,450,20);
add(componentes.getjLabelVisualizaPulsacionesBoton());
// CONFIGURACIÓN JButtonPersonalizado
jButtonPersonalizadoContar
componentes.setjButtonPersonalizadoContar(new
JButtonPersonalizado("Botón personalizado"));
```

```

componentes.get jButtonPersonalizadoContar().setBounds(
30,470,150,50);
componentes.get jButtonPersonalizadoContar().setActionC
ommand("botonPersonalizado");
componentes.get jButtonPersonalizadoContar().addActionL
istener(gestorEventos); // REGISTRO DE ESCUCHA DE
EVENTO DE BOTON
add(componentes.get jButtonPersonalizadoContar());
}
public static void main(String[] args) {
EventQueue.invokeLater(new Runnable() {
@Override
public void run() {
new InterfazGrafica6();
}
});
}
}
}

```

## Componentes

```

package interfazgrafica6;
import javax.swing.JLabel;
public class Componentes {
private JLabel jLabelVisualizaPosicionRaton;
private AreaDibujo areaDibujo;
private JLabel jLabelVisualizaPulsacionesBoton;
private JButtonPersonalizado
jButtonPersonalizadoContar;
private int contadorPulsacionesBoton;
public JLabel get jLabelVisualizaPosicionRaton() {
return jLabelVisualizaPosicionRaton;
}
public void set jLabelVisualizaPosicionRaton(JLabel
jLabelVisualizaPosicionRaton) {
this.jLabelVisualizaPosicionRaton =
jLabelVisualizaPosicionRaton;
}

```

```
}

public AreaDibujo getAreaDibujo() {
    return areaDibujo;
}

public void setAreaDibujo(AreaDibujo areaDibujo) {
    this.areaDibujo = areaDibujo;
}

public JLabel getjLabelVisualizaPulsacionesBoton() {
    return jLabelVisualizaPulsacionesBoton;
}

public void setjLabelVisualizaPulsacionesBoton(JLabel jLabelVisualizaPulsacionesBoton) {
    this.jLabelVisualizaPulsacionesBoton =
        jLabelVisualizaPulsacionesBoton;
}

public JButtonPersonalizado
get	JButtonPersonalizadoContar() {
    return jButtonPersonalizadoContar;
}

public void
set.JButtonPersonalizadoContar(JButtonPersonalizado
jButtonPersonalizadoContar) {
    this.JButtonPersonalizadoContar =
        jButtonPersonalizadoContar;
}

public int getContadorPulsacionesBoton() {
    return contadorPulsacionesBoton;
}

public void setContadorPulsacionesBoton(int
contadorPulsacionesBoton) {
    this.contadorPulsacionesBoton =
        contadorPulsacionesBoton;
}
```

## GestorEventos

```
package interfazgrafica6;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseEvent;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
import javax.swing.JButton;
import javax.swing.event.MouseInputAdapter;
public class GestorEventos extends MouseInputAdapter
implements WindowListener, ActionListener {
private Componentes componentes;
public GestorEventos(Componentes componentes) {
this.componentes = componentes;
}
// MÉTODOS DE WindowListener
public void windowClosing(WindowEvent e) {
System.exit(0);
}
public void windowActivated(WindowEvent e) {
}
public void windowDeactivated(WindowEvent e) {
}
public void windowClosed(WindowEvent e) {
}
public void windowIconified(WindowEvent e) {
}
public void windowDeiconified(WindowEvent e) {
}
public void windowOpened(WindowEvent e) {
}
// MÉTODO DE ActionListener
public void actionPerformed(ActionEvent e) {
JButton jButton = (JButton) e.getSource();
if (jButton ==
componentes.getjButtonPersonalizadoContar())
{
int contador =
```

```

componentes.getContadorPulsacionesBoton());
contador++;
componentes.setContadorPulsacionesBoton(contador);
componentes.getjLabelVisualizaPulsacionesBoton().setText("Número de pulsaciones botón:
"+componentes.getContadorPulsacionesBoton());
actionCommand : "+jButton.getActionCommand());
}
}
// MÉTODOS DE MouseInputAdapter
public void mouseMoved(MouseEvent e) {
componentes.getjLabelVisualizaPosicionRaton().setText(
"Posicion raton: coordenada x: "+e.getX()+" coordenada
y: "+e.getY());
}
public void mousePressed(MouseEvent e) {
componentes.getAreaDibujo().recibirCoordenadas1(e.getX(),
e.getY());
}
public void mouseReleased(MouseEvent e) {
componentes.getAreaDibujo().recibirCoordenadas2(e.getX(),
e.getY());
componentes.getAreaDibujo().repaint();
}
}
}

```

## AreaDibujo

```

package interfazgrafica6;
import java.awt.Canvas;
import java.awt.Graphics;
public class AreaDibujo extends Canvas {
private int x1, y1, x2, y2;
private boolean primerDibujado = true;
private boolean inicioLinea = false;
public void paint(Graphics g) {
if (primerDibujado)
{ g.drawString("AREA DE DIBUJO", 250, 20);
}
}
}

```

```

g.drawLine(1,1,649,1);
g.drawLine(1,1,1,394);
g.drawLine(649,1,649,394);
g.drawLine(1,394,649,394);
primerDibujado = false;
}
if (inicioLinea)
{ g.drawLine(x1,y1,x2,y2);
inicioLinea = false;
}
}
public void update(Graphics g) {
paint(g);
}
void recibirCoordenadas1(int x1, int y1) {
this.x1 = x1;
this.y1 = y1;
inicioLinea = true;
}
void recibirCoordenadas2(int x2, int y2) {
this.x2 = x2;
this.y2 = y2;
}
}

```

## JButtonPersonalizado

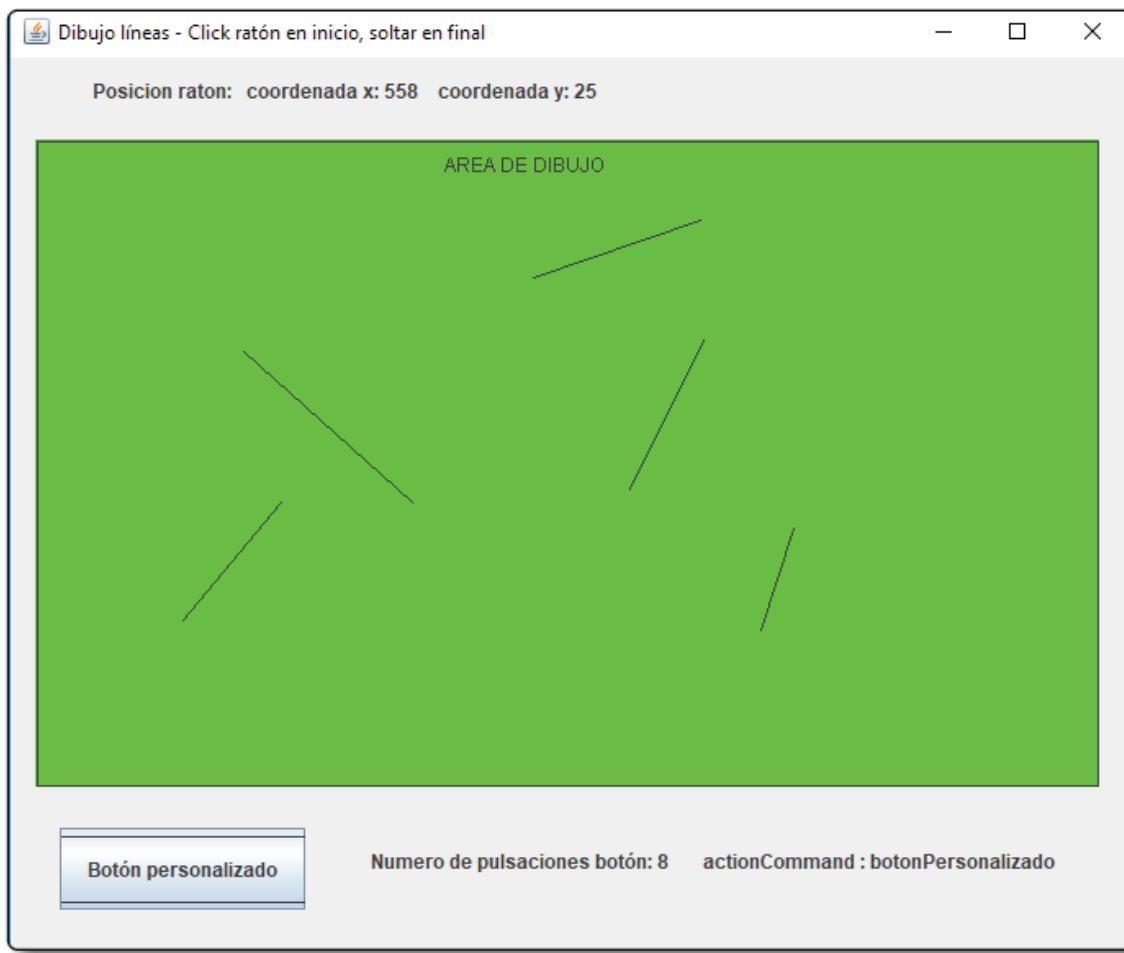
```

package interfazgrafica6;
import java.awt.Graphics;
import javax.swing.JButton;
public class JButtonPersonalizado extends JButton {
public JButtonPersonalizado(String texto) {
super(texto);
}
public void paint(Graphics g) {
super.paint(g);
g.drawLine(1, 5, this.getWidth(), 5);
}

```

```
g.drawLine(1, this.getHeight()-5, this.getWidth(),  
this.getHeight()-5);  
}  
}
```

La ejecución de la aplicación presenta la siguiente ventana:



Permite el dibujo de líneas sobre un objeto *Canvas*, materializado en la clase:

```
public class AreaDibujo extends Canvas
```

El código implementado en esta aplicación, permite al usuario el trazado de líneas, además de visualizar las coordenadas de la posición del ratón, tal y como hemos hecho en el último ejemplo. Para el trazado de una línea, el usuario debe pulsar un botón del

ratón en el punto inicial de la línea que pretende dibujar, a continuación, mueve el ratón, manteniendo pulsado el botón del ratón, hasta la posición del punto final de la susodicha línea, quedando marcado dicho punto final al soltar el botón del ratón.

Incorporamos los eventos:

- Pulsar un botón del ratón sin soltarlo, implementando las correspondientes acciones al interceptar dicho evento en el método.

```
public void mouseMoved(MouseEvent e)
```

- soltar el botón del ratón, implementando las correspondientes acciones al interceptar dicho evento en el método

```
public void mousePressed(MouseEvent e)
```

Hemos optado por definir la clase **GestorEventos** inversamente a como la definimos en los ejemplos anteriores: ahora la erigimos en heredera de *MouseInputAdapter*, y en consecuencia, forzada a implementar la interface *WindowListener*. Heredar de *MouseInputAdapter* permite evitar la definición de los métodos con implementación vacía no utilizados de la interface *MouseInputListener*, pero en contrapartida, nos hemos visto obligados a definir con implementación vacía el resto de métodos de la interface *WindowListener* adicionalmente al método *windowClosing()*, circunstancia ya comentada en el estudio en detalle de la aplicación **InterfazGrafica4**.

Para que el trazado de líneas, así como el movimiento del ratón sean detectados en el área de dibujo, es decir, en el *Canvas*, debemos registrar con la escucha de eventos de ratón, no el *JFrame*, sino el citado *Canvas*:

```
componentes.getAreaDibujo().addMouseListener(gestorEventos); // REGISTRO DE ESCUCHA DE EVENTOS DE RATON  
componentes.getAreaDibujo().addMouseMotionListener(gestorEventos); // REGISTRO DE ESCUCHA DE EVENTO  
MOVIMIENTO DE RATON
```

El *Canvas* procede añadirlo a la ventana principal como un componente más:

```
this.getContentPane().add(componentes.getAreaDibujo())  
;
```

Para entender cómo procedemos al trazado de líneas en este ejercicio, es necesario abordar en primer lugar las clases involucradas en dicho proceso. Para el trazado de figuras, disponemos en java de la clase *Canvas*. El dibujado de cualquier componente Swing se produce por una invocación automática a su método *paint()*, disponible en todos los componentes por herencia desde *JComponent*. Cuando el programador pretende intervenir en el dibujado de un componente, sobrescribe dicho método *paint()*, que es exactamente lo que estamos haciendo en nuestro ejemplo, sobrescribir el método *paint()* de nuestro *Canvas* personalizado en la clase **AreaDibujo**. Este método recibe la referencia a un objeto *Graphics* (realmente es un objeto que hereda de esta clase dado que *Graphics* es una clase *abstract*) que irá vinculado al componente. Para el trazado de figuras sobre el componente utilizamos los métodos del citado objeto perteneciente a la jerarquía *Graphics*, tal y como podemos observar en nuestro ejemplo para el trazado de una línea:

```
g.drawLine(x1,y1,x2,y2);
```

Como ya se ha mencionado, el método *paint()* se ejecuta automáticamente cuando presentamos el *Canvas*, aplicando las siguientes acciones:

```
g.drawString("AREA DE DIBUJO", 250, 20);
```

```
g.drawLine(1,1,649,1);
g.drawLine(1,1,1,394);
g.drawLine(649,1,649,394);
g.drawLine(1,394,649,394);
```

En nuestro ejemplo no necesitamos de unas acciones previas contempladas en el método *paint()* de *Canvas*, superclase en que basamos **AreaDibujo**. Pero en el caso de otros componentes, sí que sería necesario contemplar dichas acciones previas inherentes a la superclase. Para ello, deberíamos codificar en primer lugar en la sobrescritura de *paint()*:

```
super.paint(g);
```

A efectos de mostrar al lector un caso práctico de lo que acabamos de mencionar, añadimos a esta aplicación la clase

```
public class JButtonPersonalizado extends JButton
```

Obtenemos un *JButton* “extendido” (no es éste un término muy común cuando se está tratando la herencia, pero sí que es lo suficientemente ilustrativo) al que le hemos añadido un pequeño detalle a la apariencia, concretamente dos líneas horizontales paralelas en la parte superior e inferior. Ello lo hemos conseguido implementando dichas aportaciones a la apariencia sobrescribiendo el método *paint()*:

```
public void paint(Graphics g) {
super.paint(g);
g.drawLine(1, 5, this.getWidth(), 5);
g.drawLine(1, this.getHeight()-5, this.getWidth(),
this.getHeight()-5);
}
```

En dicha sobrescritura nos vemos obligados a la invocación de la implementación de dicho método en la superclase:

```
super.paint(g);
```

tal y como hemos mencionado, para disponer del sustrato básico de la apariencia del *JButton*.

La instancia del botón personalizado se registra con la clase escucha **GestorEventos**, como hemos venido haciendo en anteriores ejemplos, y también le establecemos:

```
componentes.getjButtonPersonalizadoContar().setActionCommand("botonPersonalizado");
```

mecanismo que trataremos más adelante. Aplicamos ambas actuaciones simplemente para comprobar que el botón personalizado obtenido conserva todas las funcionalidades heredadas de su superclase. También ha sido necesario definir el constructor con argumento *String*.

Volvamos al trazado de una línea en el *Canvas*. Cuando pretendemos que aparezca una línea sobre el *Canvas* que previamente hemos trazado mediante *g.drawLine()*, invocamos explícitamente al método *repaint()*, que provoca una ejecución automática del método *update()*, en que se procede al repintado. La lógica algorítmica aplicada provoca que en el primer “pintado” se produzcan unas acciones, y en el “repintado” se trace la línea, todo ello determinado por los valores de las variables *boolean primerDibujado* e *inicioLinea*.

El mecanismo utilizado en este ejemplo para el trazado de líneas consiste en definir en la clase que actúa como área de dibujo las coordenadas que definen los extremos del segmento:

```
private int x1, y1, x2, y2;
```

Las coordenadas del primer punto (*x1*, *y1*) son asignadas a los correspondientes atributos cuando se detecta la escucha “*mousePressed*” del ratón desde la clase **GestorEventos**:

```
public void mousePressed(MouseEvent e) {
```

```
componentes.getAreaDibujo().recibirCoordenadas1(e.getX()
(),e.getY());
}
```

De forma análoga se procede con las coordenadas ( $x_2$ ,  $y_2$ ) del otro extremo cuando se detecta el evento “*mouseReleased*”. Una vez ya tenemos las coordenadas de los dos extremos del segmento, ya procede “repintar” el *Canvas*:

```
componentes.getAreaDibujo().repaint();
```

a continuación, prosiguiendo con la ejecución del método escucha de este último evento. La invocación al *método repaint()*, como ya hemos comentado, supone volver a ejecutar el método *paint()*, en que condicionamos el trazado de la línea al valor de la variable *boolean inicioLinea*:

```
if (inicioLinea)
{ g.drawLine(x1,y1,x2,y2);
inicioLinea = false;
}
```

# 14

## APLICACIÓN INTERFAZGRAFICA7

A la estructura de la aplicación anterior, añadimos tres clases que heredan de *JPanel*:

- *package interfazgrafica7* aglutina las clases:
  - InterfazGrafica7
  - Componentes
  - GestorEventos
  - AreaDibujo
  - PanelJToggleButton
  - PanelJRadioButton
  - PanelJCheckBox

### InterfazGrafica7

```
package interfazgrafica7;
import java.awt.Color;
import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
public class InterfazGrafica7 extends JFrame {
public InterfazGrafica7() {
```

```
setSize(700,750);
setTitle("Dibujo líneas rectángulos óvalos - Click
ratón en inicio, soltar en final");
ubicarComponentes();
setVisible(true);
}
private void ubicarComponentes() {
setLayout(null);
Componentes componentes = new Componentes();
GestorEventos gestorEventos = new
GestorEventos(componentes);
componentes.setGestorEventos(gestorEventos);
addWindowListener(gestorEventos); // REGISTRO DE
ESCUCHA DE EVENTOS DE VENTANA
AreaDibujo areaDibujo = new AreaDibujo(componentes);
componentes.setAreaDibujo(areaDibujo);
componentes.getAreaDibujo().setBounds(15, 50, 650,
395);
componentes.getAreaDibujo().setBackground(Color.green)
;
componentes.getAreaDibujo().addMouseListener(gestorEve
ntos); // REGISTRO DE ESCUCHA DE EVENTOS DE RATON
componentes.getAreaDibujo().addMouseMotionListener(ges
torEventos); // REGISTRO DE ESCUCHA DE EVENTO
MOVIMIENTO DE RATON
this.getContentPane().add(componentes.getAreaDibujo())
;
// CONFIGURACIÓN JLabel jLabelVisualizaPosicionRaton
componentes.setjLabelVisualizaPosicionRaton(new
JLabel("Posicion raton: "));
componentes.getjLabelVisualizaPosicionRaton().setBound
s(50,10,400,20);
add(componentes.getjLabelVisualizaPosicionRaton());
JPanel jPanelSeleccionFiguras = new
PanelJToggleButton(componentes); // SELECCION DE
FIGURAS MEDIANTE JToggleButton
// JPanel jPanelSeleccionFiguras = new
```

```

PanelJRadioButton(componentes); // SELECCION DE
FIGURAS MEDIANTE JRadioButton
// JPanel jPanelSeleccionFiguras = new
PanelJCheckBox(componentes); // SELECCION DE FIGURAS
MEDIANTE JCheckBox
jPanelSeleccionFiguras.setBounds(15, 450, 650, 395);
add(jPanelSeleccionFiguras);
}
public static void main(String[] args) {
EventQueue.invokeLater(new Runnable() {
@Override
public void run() {
new InterfazGrafica7();
}
});
}
}
}

```

## Componentes

```

package interfazgrafica7;
import javax.swing.JCheckBox;
import javax.swing.JLabel;
import javax.swing.JRadioButton;
import javax.swing.JToggleButton;
public class Componentes {
private GestorEventos gestorEventos;
private AreaDibujo areaDibujo;
private JLabel jLabelVisualizaPosicionRaton;
private JToggleButton jToggleButtonLineas;
private JToggleButton jToggleButtonRectangulos;
private JToggleButton jToggleButtonOvalos;
private JRadioButton jRadioButtonLineas;
private JRadioButton jRadioButtonRectangulos;
private JRadioButton jRadioButtonOvalos;
private JCheckBox jCheckBoxLineas;
private JCheckBox jCheckBoxRectangulos;

```

```
private JCheckBox jCheckBoxOvalos;
private byte tipoFigura = 1;
public byte getTipoFigura() {
    return tipoFigura;
}
public void setTipoFigura(byte tipoFigura) {
    this.tipoFigura = tipoFigura;
}
public GestorEventos getGestorEventos() {
    return gestorEventos;
}
public void setGestorEventos(GestorEventos
gestorEventos) {
    this.gestorEventos = gestorEventos;
}
public JLabel getjLabelVisualizaPosicionRaton() {
    return jLabelVisualizaPosicionRaton;
}
public void setjLabelVisualizaPosicionRaton(JLabel
jLabelVisualizaPosicionRaton) {
    this.jLabelVisualizaPosicionRaton =
    jLabelVisualizaPosicionRaton;
}
public AreaDibujo getAreaDibujo() {
    return areaDibujo;
}
public void setAreaDibujo(AreaDibujo areaDibujo) {
    this.areaDibujo = areaDibujo;
}
public JToggleButton getjToggleButtonLineas() {
    return jToggleButtonLineas;
}
public void setjToggleButtonLineas(JToggleButton
jToggleButtonLineas) {
    this.jToggleButtonLineas = jToggleButtonLineas;
}
public JToggleButton getjToggleButtonRectangulos() {
```

```
return jToggleButtonRectangulos;
}
public void setjToggleButtonRectangulos(JToggleButton
jToggleButtonRectangulos) {
this.jToggleButtonRectangulos =
jToggleButtonRectangulos;
}
public JToggleButton getjToggleButtonOvalos() {
return jToggleButtonOvalos;
}
public void setjToggleButtonOvalos(JToggleButton
jToggleButtonOvalos) {
this.jToggleButtonOvalos = jToggleButtonOvalos;
}
public JRadioButton getjRadioButtonLineas() {
return jRadioButtonLineas;
}
public void setjRadioButtonLineas(JRadioButton
jRadioButtonLineas) {
this.jRadioButtonLineas = jRadioButtonLineas;
}
public JRadioButton getjRadioButtonRectangulos() {
return jRadioButtonRectangulos;
}
public void setjRadioButtonRectangulos(JRadioButton
jRadioButtonRectangulos) {
this.jRadioButtonRectangulos =
jRadioButtonRectangulos;
}
public JRadioButton getjRadioButtonOvalos() {
return jRadioButtonOvalos;
}
public void setjRadioButtonOvalos(JRadioButton
jRadioButtonOvalos) {
this.jRadioButtonOvalos = jRadioButtonOvalos;
}
public JCheckBox getjCheckBoxLineas() {
```

```
return jCheckBoxLineas;
}
public void setjCheckBoxLineas(JCheckBox
jCheckBoxLineas) {
this.jCheckBoxLineas = jCheckBoxLineas;
}
public JCheckBox getjCheckBoxRectangulos() {
return jCheckBoxRectangulos;
}
public void setjCheckBoxRectangulos(JCheckBox
jCheckBoxRectangulos) {
this.jCheckBoxRectangulos = jCheckBoxRectangulos;
}
public JCheckBox getjCheckBoxOvalos() {
return jCheckBoxOvalos;
}
public void setjCheckBoxOvalos(JCheckBox
jCheckBoxOvalos) {
this.jCheckBoxOvalos = jCheckBoxOvalos;
}
}
```

## GestorEventos

```
package interfazgrafica7;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.awt.event.MouseEvent;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
import javax.swing.JRadioButton;
import javax.swing.JToggleButton;
import javax.swing.event.MouseInputAdapter;
public class GestorEventos extends MouseInputAdapter
implements WindowListener, ActionListener,
```

```
ItemClickListener {
private Componentes componentes;
public GestorEventos(Componentes componentes) {
this.componentes = componentes;
}
// MÉTODOS DE WindowListener
public void windowClosing(WindowEvent e) {
System.exit(0);
}
public void windowActivated(WindowEvent e) {
}
public void windowDeactivated(WindowEvent e) {
}
public void windowClosed(WindowEvent e) {
}
public void windowIconified(WindowEvent e) {
}
public void windowDeiconified(WindowEvent e) {
}
public void windowOpened(WindowEvent e) {
}
// MÉTODOS DE MouseInputAdapter
public void mouseMoved(MouseEvent e) {
componentes.getjLabelVisualizaPosicionRaton().setText(
“Posicion raton: coordenada x: “+e.getX()+” coordenada
y: “+e.getY());
}
public void mousePressed(MouseEvent e) {
componentes.getAreaDibujo().recibirCoordenadas1(e.getX(),
e.getY());
}
public void mouseReleased(MouseEvent e) {
componentes.getAreaDibujo().recibirCoordenadas2(e.getX(),
e.getY());
componentes.getAreaDibujo().repaint();
}
// MÉTODO DE ActionListener
```

```
public void actionPerformed(ActionEvent e) {
switch(e.getSource().getClass().getName())
{
case "javax.swing.JToggleButton" : // SELECCION DE
FIGURAS MEDIANTE JToggleButton
JToggleButton jToggleButton = (JToggleButton)
e.getSource();
if (jToggleButton ==
componentes.getjToggleButtonLineas())
{ componentes.setTipoFigura((byte)1);
}
else
if (jToggleButton ==
componentes.getjToggleButtonRectangulos())
{ componentes.setTipoFigura((byte)2);
}
else
if (jToggleButton ==
componentes.getjToggleButtonOvalos())
{ componentes.setTipoFigura((byte)3);
}
break;
case "javax.swing.JRadioButton" : // SELECCION DE
FIGURAS MEDIANTE JRadioButton
JRadioButton jRadioButton = (JRadioButton)
e.getSource();
if (jRadioButton ==
componentes.getjRadioButtonLineas())
{ componentes.setTipoFigura((byte)1);
}
else
if (jRadioButton ==
componentes.getjRadioButtonRectangulos())
{ componentes.setTipoFigura((byte)2);
}
else
if (jRadioButton ==
```

```

componentes.getjRadioButtonOvalos())
{ componentes.setTipoFigura((byte)3);
}
break;
}
}

// MÉTODO DE ItemListener
public void itemStateChanged(ItemEvent e)
{ Object fuente = e.getItemSelectable();
// SELECCION DE FIGURAS MEDIANTE JCheckBox
if (fuente == componentes.getjCheckBoxLineas())
componentes.setTipoFigura((byte)1);
else
if (fuente == componentes.getjCheckBoxRectangulos())
componentes.setTipoFigura((byte)2);
else
if (fuente == componentes.getjCheckBoxOvalos())
componentes.setTipoFigura((byte)3);
}
}

```

## AreaDibujo

```

package interfazgrafica7;
import java.awt.Canvas;
import java.awt.Graphics;
public class AreaDibujo extends Canvas {
private Componentes componentes;
private int x1, y1, x2, y2;
private boolean primerDibujado = true;
private boolean inicioFigura = false;
public AreaDibujo(Componentes componentes) {
this.componentes = componentes;
}
public void paint(Graphics g) {
if (primerDibujado)
{ g.drawString("AREA DE DIBUJO", 250, 20);

```

```
g.drawLine(1, 1, 649, 1);
g.drawLine(1, 1, 1, 394);
g.drawLine(649, 1, 649, 394);
g.drawLine(1, 394, 649, 394);
primerDibujado = false;
}
if (inicioFigura)
{
switch(componentes.getTipoFigura())
{
case 1: g.drawLine(x1, y1, x2, y2);
break;
case 2: g.drawRect(x1, y1, x2-x1, y2-y1);
break;
case 3: g.drawOval(x1, y1, x2-x1, y2-y1);
break;
}
inicioFigura = false;
}
}

public void update(Graphics g) {
paint(g);
}

public void recibirCoordenadas1(int x1, int y1) {
this.x1 = x1;
this.y1 = y1;
inicioFigura = true;
}

void recibirCoordenadas2(int x2, int y2) {
this.x2 = x2;
this.y2 = y2;
}
```

## PanelJToggleButton

```
package interfazgrafica7;
```

```
import javax.swing.ButtonGroup;
import javax.swing.JPanel;
import javax.swing.JToggleButton;
public class PanelJToggleButton extends JPanel {
private Componentes componentes;
public PanelJToggleButton(Componentes componentes) {
this.componentes = componentes;
ubicarComponentes();
}
private void ubicarComponentes() {
setLayout(null);
componentes.setjToggleButtonLineas(new
JToggleButton("Línea", true));
componentes.getjToggleButtonLineas().setBounds(255,
45, 100, 40);
add(componentes.getjToggleButtonLineas());
componentes.getjToggleButtonLineas().addActionListener
(componentes.getGestorEventos());
componentes.setjToggleButtonRectangulos(new
JToggleButton("Rectángulo", false));
componentes.getjToggleButtonRectangulos().setBounds(25
5, 105, 100, 40);
add(componentes.getjToggleButtonRectangulos());
componentes.getjToggleButtonRectangulos().addActionList
tener(componentes.getGestorEventos());
componentes.setjToggleButtonOvalos(new
JToggleButton("Óvalo", false));
componentes.getjToggleButtonOvalos().setBounds(255,
165, 100, 40);
add(componentes.getjToggleButtonOvalos());
componentes.getjToggleButtonOvalos().addActionListener
(componentes.getGestorEventos());
ButtonGroup buttonGroup = new ButtonGroup();
buttonGroup.add(componentes.getjToggleButtonLineas());
buttonGroup.add(componentes.getjToggleButtonRectangulo
s());
buttonGroup.add(componentes.getjToggleButtonOvalos());
```

```
}
```

## PanelJRadioButton

```
package interfazgrafica7;
import javax.swing.ButtonGroup;
import javax.swing.JPanel;
import javax.swing.JRadioButton;
public class PanelJRadioButton extends JPanel {
private Componentes componentes;
public PanelJRadioButton(Componentes componentes) {
this.componentes = componentes;
ubicarComponentes();
}
private void ubicarComponentes() {
setLayout(null);
componentes.setjRadioButtonLineas(new
JRadioButton("Línea", true));
componentes.getjRadioButtonLineas().setBounds(255, 45,
100, 40);
add(componentes.getjRadioButtonLineas());
componentes.getjRadioButtonLineas().addActionListener(
componentes.getGestorEventos());
componentes.setjRadioButtonRectangulos(new
JRadioButton("Rectángulo", false));
componentes.getjRadioButtonRectangulos().setBounds(255
, 105, 100, 40);
add(componentes.getjRadioButtonRectangulos());
componentes.getjRadioButtonRectangulos().addActionList
ener(componentes.getGestorEventos());
componentes.setjRadioButtonOvalos(new
JRadioButton("Óvalo", false));
componentes.getjRadioButtonOvalos().setBounds(255,
165, 100, 40);
add(componentes.getjRadioButtonOvalos());
componentes.getjRadioButtonOvalos().addActionListener(
```

```
componentes.getGestorEventos());
ButtonGroup buttonGroup = new ButtonGroup();
buttonGroup.add(componentes.getjRadioButtonLineas());
buttonGroup.add(componentes.getjRadioButtonRectangulos());
buttonGroup.add(componentes.getjRadioButtonOvalos());
}
}
```

## PanelJCheckBox

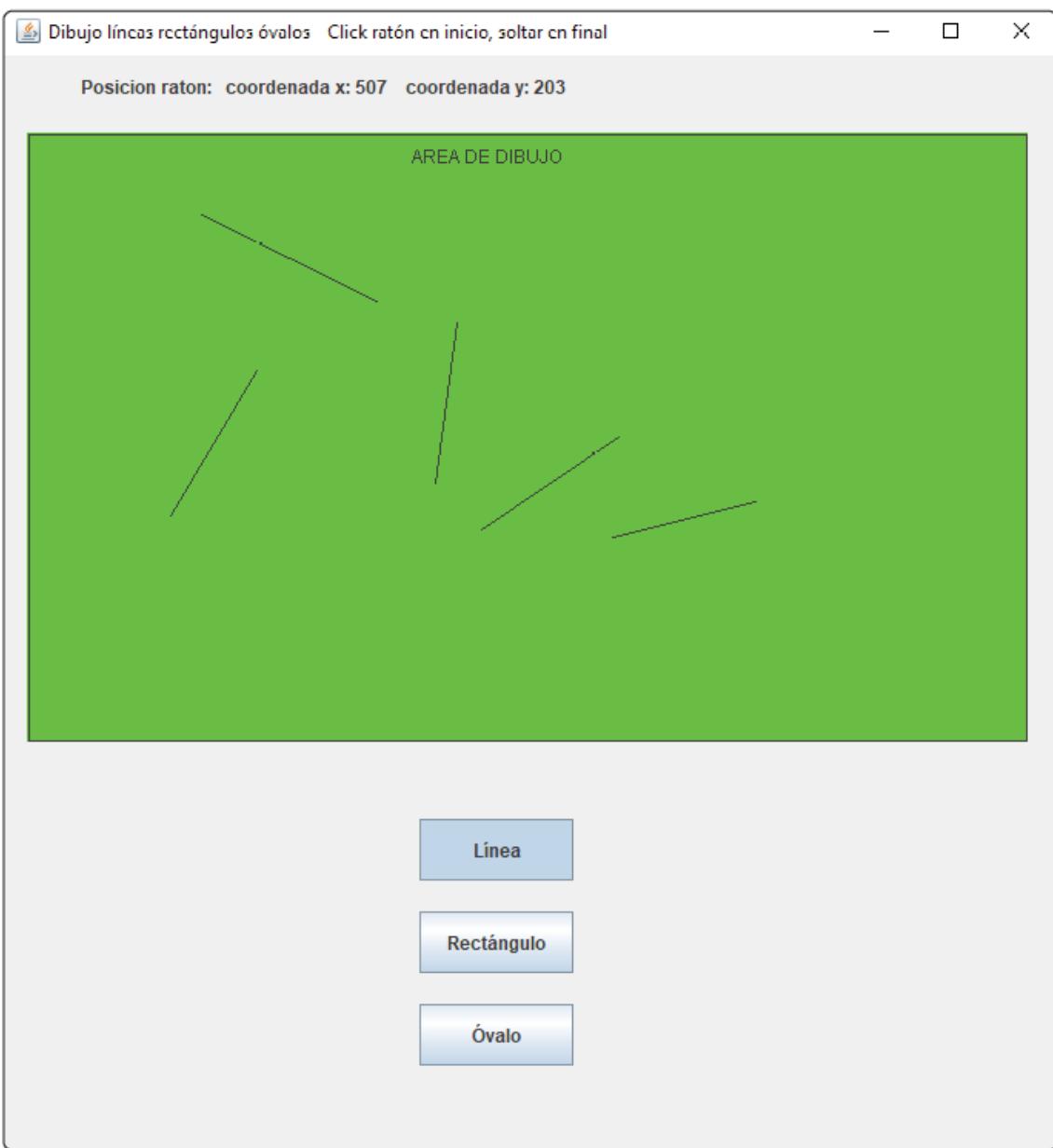
```
package interfazgrafica7;
import javax.swing.ButtonGroup;
import javax.swing.JPanel;
import javax.swing.JCheckBox;
public class PanelJCheckBox extends JPanel {
private Componentes componentes;
public PanelJCheckBox(Componentes componentes) {
this.componentes = componentes;
ubicarComponentes();
}
private void ubicarComponentes() {
setLayout(null);
componentes.setjCheckBoxLineas(new JCheckBox("Línea",
true));
componentes.getjCheckBoxLineas().setBounds(255, 45,
100, 40);
add(componentes.getjCheckBoxLineas());
componentes.getjCheckBoxLineas().addItemListener(componentes.getGestorEventos());
componentes.setjCheckBoxRectangulos(new
JCheckBox("Rectángulo", false));
componentes.getjCheckBoxRectangulos().setBounds(255,
105, 100, 40);
add(componentes.getjCheckBoxRectangulos());
componentes.getjCheckBoxRectangulos().addItemListener(componentes.getGestorEventos());
```

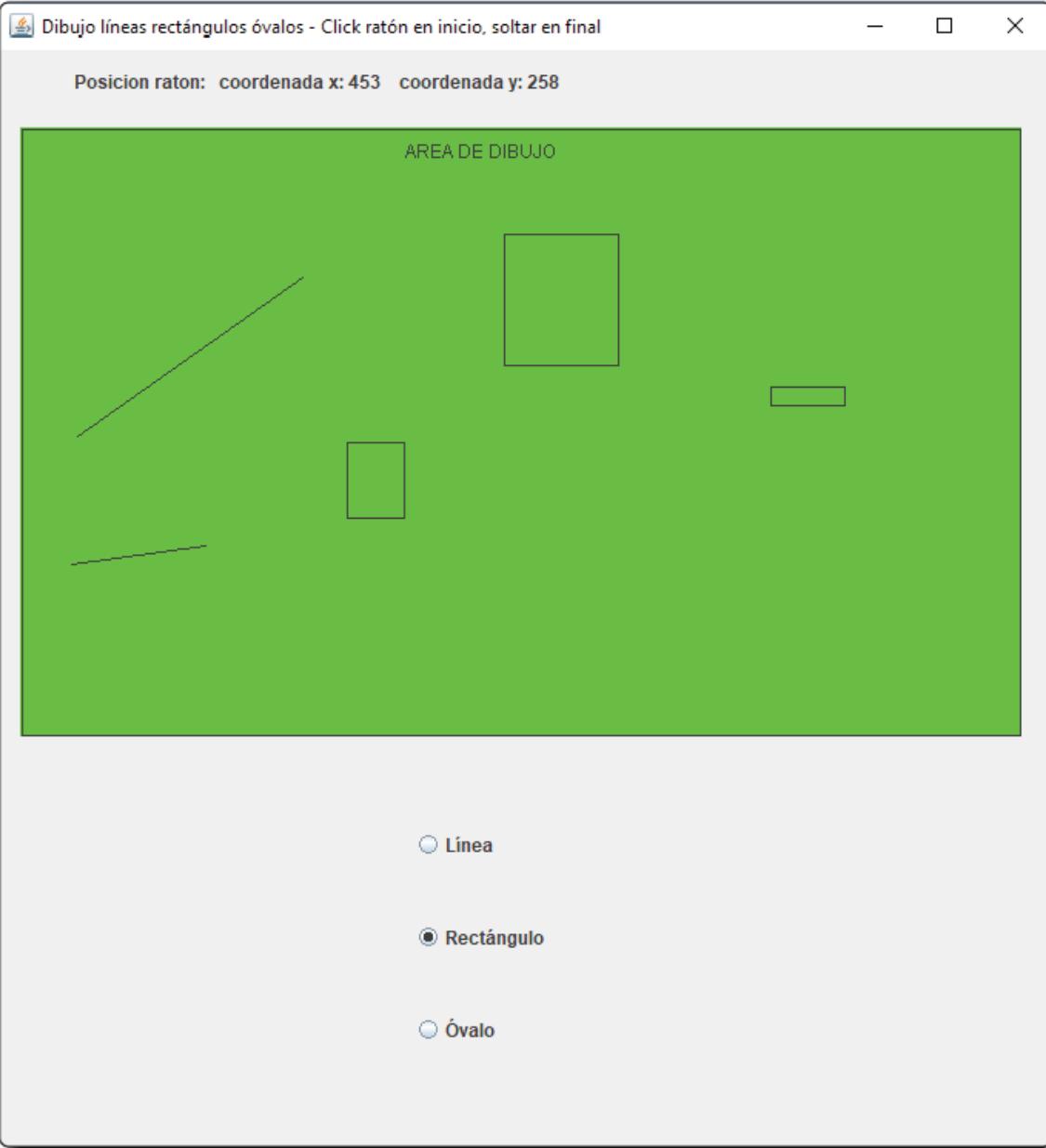
```
componentes.setjCheckBoxOvalos(new JCheckBox("Óvalo",
false));
componentes.getjCheckBoxOvalos().setBounds(255, 165,
100, 40);
add(componentes.getjCheckBoxOvalos());
componentes.getjCheckBoxOvalos().addItemListener(compo-
nentes.getGestorEventos());
ButtonGroup buttonGroup = new ButtonGroup();
buttonGroup.add(componentes.getjCheckBoxLineas());
buttonGroup.add(componentes.getjCheckBoxRectangulos())
;
buttonGroup.add(componentes.getjCheckBoxOvalos());
}
}
```

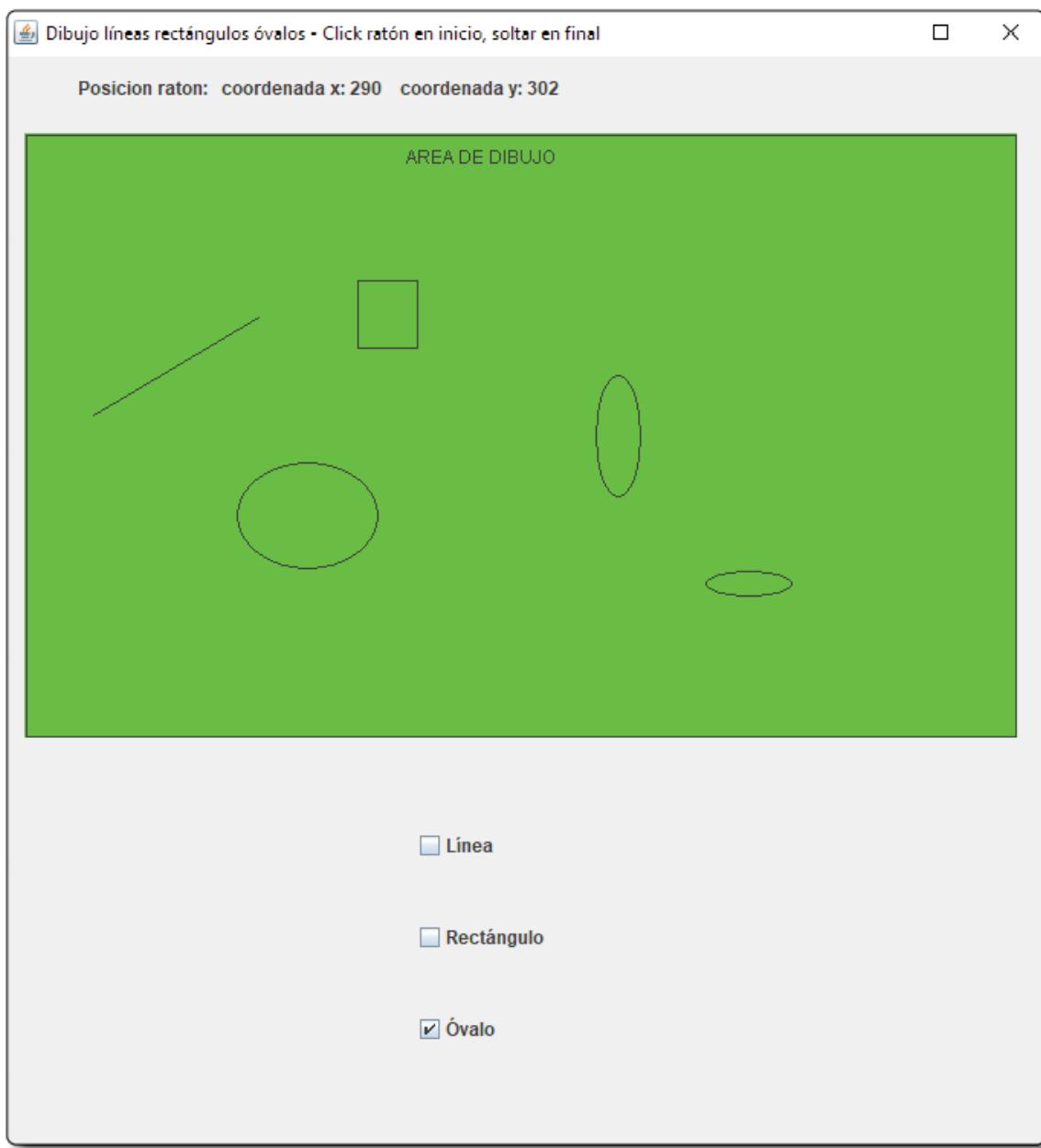
Disponemos de tres variantes para la ejecución de esta aplicación, en función de que el *JFrame* añada uno de los siguientes *JPanel*:

- PanelJToggleButton
- PanelJRadioButton
- PanelJCheckBox

Presentamos la ventana que se presenta al usuario de la aplicación para cada una de las tres posibles ejecuciones alternativas:







De las tres ventanas presentadas, en la segunda aparecen seleccionadas Rectángulos, pero vemos que también aparecen dibujadas líneas además de rectángulos. Ello se debe a que previamente a la selección que aparece, se han trazado líneas, encontrándose en dicha selección. Lo mismo podemos decir de la tercera ventana, en que aparecen seleccionados Óvalos, pero previamente se han dibujado los otros dos tipos de figuras,

encontrándose en cada momento bajo la selección correspondiente.

La ampliación de esta aplicación ejemplo respecto a la anterior se basa en los siguientes aspectos:

- Utilización de componentes tales como *JToggleButton*, *JRadioButton*, *JCheckBox* susceptibles de ser agrupados en un *ButtonGroup*, a efectos de conseguir que solamente uno de los componentes aglutinados por el grupo pueda estar seleccionado en cada momento, circunstancia que nos permitirá implementar criterios de selección. En este ejemplo, nos permitirá seleccionar el tipo de figura a dibujar. En el ejemplo anterior solamente se contemplaba el trazado de líneas sobre el *Canvas*. Ahora, adicionalmente podremos dibujar rectángulos u óvalos también, en función del componente del grupo seleccionado.
- Evidenciar la utilización de *JPanel* como subcontenedor. En este ejemplo aportamos tres, intercambiables según el tipo de componente a utilizar en la selección del tipo de figura a dibujar. El programador puede escoger la clase de los componentes a utilizar para la selección del tipo de figura actuando en las líneas de código del método **ubicarComponentes()** de la clase **InterfazGrafica7**:

```
JPanel jPanelSeleccionFiguras = new
PanelJToggleButton(componentes); // SELECCION DE
FIGURAS MEDIANTE JToggleButton
// JPanel jPanelSeleccionFiguras = new
PanelJRadioButton(componentes); // SELECCION DE
FIGURAS MEDIANTE JRadioButton
// JPanel jPanelSeleccionFiguras = new
PanelJCheckBox(componentes); // SELECCION DE
FIGURAS MEDIANTE JCheckBox
```

dejando activada una de las tres líneas, y comentarizando las otras dos.

- Presentar un caso de método de clase escucha de eventos, en que como consecuencia se centralizar la escucha de varias clases de componentes, nos vemos obligados a dilucidar la clase del componente fuente del evento antes de la identificación del componente en concreto que lo ha generado. En esta aplicación hemos utilizado una de las posibles soluciones:

```
switch(e.getSource().getClass().getName())
{
    case "javax.swing.JToggleButton" : // SELECCION DE FIGURAS MEDIANTE JToggleButton
        JToggleButton jToggleButton = (JToggleButton)
e.getSource();
        . . .
        break;
    case "javax.swing.JRadioButton" : // SELECCION DE FIGURAS MEDIANTE JRadioButton
        JRadioButton jRadioButton = (JRadioButton)
e.getSource();
        . . .
        break;
}
```

Existen más soluciones, otra de ellas consistiría en establecer el *actionCommand* para cada uno de los componentes, solución que se será utilizada y presentada en ejercicios posteriores.

- Ampliar el repertorio de escuchas de eventos con la aportación del *ItemListener*, y su correspondiente método de escucha *itemStateChanged()*. En el ejemplo, se registran con dicha clase escucha los *JCheckBox*.

La lógica del algoritmo utilizada para determinar el tipo de figura a dibujar se centra en establecer el valor del atributo

```
private byte tipoFigura = 1;
```

declarado en la clase Componentes, que, como siempre actúa como espacio centralizado de variables y referencias que deben quedar accesibles desde cualquier clase de la aplicación. Dicho atributo para disponer de un valor por defecto, que es modificado en los métodos de escucha de eventos cuando se detecta un cambio en la selección, como por ejemplo hacemos en:

```
if (jToggleButton ==  
componentes.getjToggleButtonRectangulos())  
{ componentes.setTipoFigura((byte)2);  
}
```

Dicho valor determinará el método a utilizar en función del tipo de figura seleccionado:

```
switch(componentes.getTipoFigura())  
{  
case 1: g.drawLine(x1, y1, x2, y2);  
break;  
case 2: g.drawRect(x1, y1, x2-x1, y2-y1);  
break;  
case 3: g.drawOval(x1, y1, x2-x1, y2-y1);  
break;  
}
```

Es necesario tener en cuenta que, así como las líneas las dibuja marcando punto inicial y punto final en cualquier sentido de trayectoria, para los rectángulos y los óvalos solamente los dibuja cuando el punto inicial está más a la izquierda y más arriba que el final. Esta limitación obedece a la intención de no proporcionar mayor complejidad al código.

# 15

## APLICACIÓN INTERFAZGRAFICA8

Volvemos a la estructura que presentaban aplicaciones anteriores:

- *package* **interfazgrafica8** aglutina las clases:
  - InterfazGrafica8
  - Componentes
  - GestorEventos

### InterfazGrafica8

```
package interfazgrafica8;
import java.awt.EventQueue;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JTextField;
public class InterfazGrafica8 extends JFrame {
public InterfazGrafica8() {
setSize(700,500);
setTitle("Ejemplo utilización JComboBox");
ubicarComponentes();
setVisible(true);
}
private void ubicarComponentes() {
setLayout(null);
```

```
Componentes componentes = new Componentes();
GestorEventos gestorEventos = new
GestorEventos(componentes);
addWindowListener(gestorEventos); // REGISTRO DE
ESCUCHA DE EVENTOS DE VENTANA
// CONFIGURACIÓN JComboBox
componentes.setjComboBox(new JComboBox());
componentes.getjComboBox().setBounds(5, 150, 250, 20);
componentes.getjComboBox().setActionCommand("combo");
componentes.getjComboBox().addActionListener(gestorEventos); // REGISTRO DE ESCUCHA DE EVENTO EN JComboBox
add(componentes.getjComboBox());
// CONFIGURACIÓN JTextField
componentes.setjTextfield(new JTextField());
componentes.getjTextfield().setBounds(400, 150, 200,
20);
add(componentes.getjTextfield());
// CONFIGURACIÓN JButton jButtonAñadir
JButton jButtonAñadir = new JButton("Añadir");
jButtonAñadir.setBounds(285, 150, 90, 40);
jButtonAñadir.setActionCommand("botonAñadir");
jButtonAñadir.addActionListener(gestorEventos); // REGISTRO DE ESCUCHA DE EVENTO DE BOTON
add(jButtonAñadir);
// CONFIGURACIÓN JButton jButtonVaciar
JButton jButtonVaciar = new JButton("Vaciar");
jButtonVaciar.setBounds(285, 200, 90, 40);
jButtonVaciar.setActionCommand("botonVaciar");
jButtonVaciar.addActionListener(gestorEventos); // REGISTRO DE ESCUCHA DE EVENTO DE BOTON
add(jButtonVaciar);
// CONFIGURACIÓN JButton jButtonEliminar
JButton jButtonEliminar = new JButton("Eliminar");
jButtonEliminar.setBounds(285, 250, 90, 40);
jButtonEliminar.setActionCommand("botonEliminar");
jButtonEliminar.addActionListener(gestorEventos); // REGISTRO DE ESCUCHA DE EVENTO DE BOTON
```

```
add(jButtonEliminar);
}
public static void main(String[] args) {
EventQueue.invokeLater(new Runnable() {
@Override
public void run() {
new InterfazGrafica8();
}
});
}
```

## Componentes

```
package interfazgrafica8;
import javax.swing.JComboBox;
import javax.swing.JTextField;
public class Componentes {
private JComboBox jComboBox;
private JTextField jTextField;
public JComboBox getjComboBox() {
return jComboBox;
}
public void setjComboBox(JComboBox jComboBox) {
this.jComboBox = jComboBox;
}
public JTextField getjTextField() {
return jTextField;
}
public void setjTextField(JTextField jTextField) {
this(jTextField = jTextField;
}
}
```

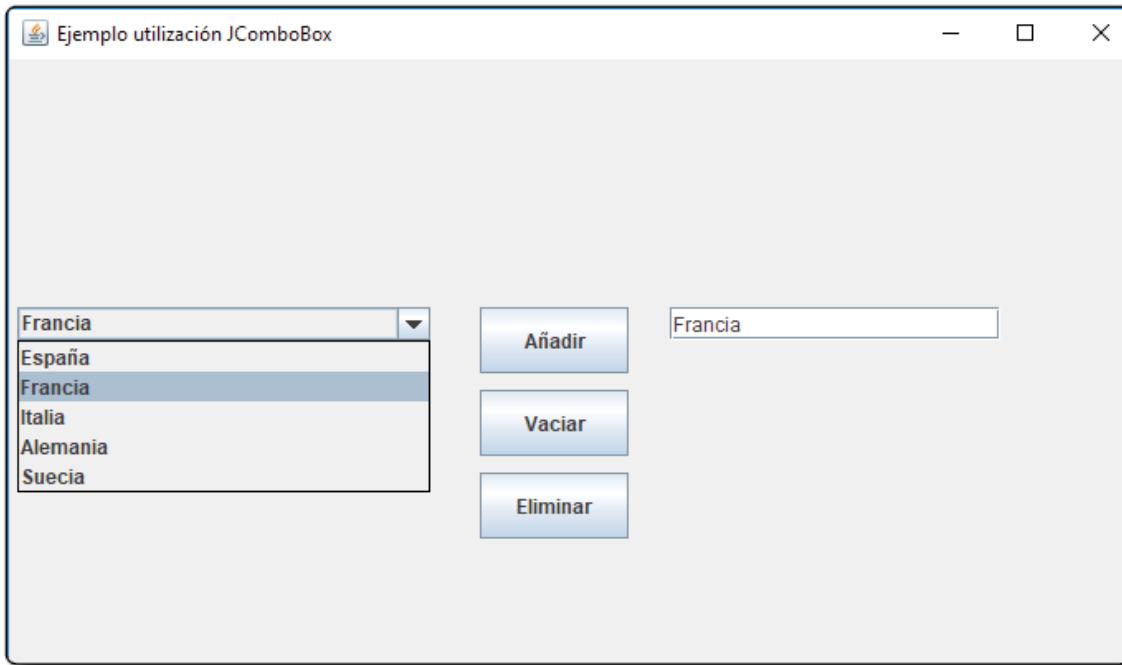
## GestorEventos

```
package interfazgrafica8;
import java.awt.event.ActionEvent;
```

```
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
public class GestorEventos extends WindowAdapter
implements ActionListener {
private Componentes componentes;
public GestorEventos(Componentes componentes) {
this.componentes = componentes;
}
// MÉTODOS DE WindowAdapter
public void windowClosing(WindowEvent e) {
System.exit(0);
}
// MÉTODO DE ActionListener
public void actionPerformed(ActionEvent e) {
if
(componentes.getjComboBox().getActionListeners().length > 0)
componentes.getjComboBox().removeActionListener(this);
switch(e.getActionCommand())
{
case "combo" :
componentes.getjTextfield().setText(componentes.getjComboBox().getSelectedItem().toString());
break;
case "botonAñadir" :
if
(componentes.getjTextfield().getText().compareTo("")!=0)
{
componentes.getjComboBox().addItem(componentes.getjTextfield().getText());
componentes.getjComboBox().setSelectedIndex(componentes.getjComboBox().getItemCount()-1);
componentes.getjTextfield().setText("");
}
break;
}
```

```
case "botonVaciar" :  
componentes.getjComboBox().removeAllItems();  
componentes.getjTextfield().setText("");  
break;  
case "botonEliminar" :  
if (componentes.getjComboBox().getItemCount()>0)  
{  
if (componentes.getjComboBox().getItemCount()>1)  
{  
componentes.getjComboBox().removeItemAt(componentes.getjComboBox().getSelectedIndex());  
}  
else  
{ componentes.getjComboBox().removeAllItems();  
componentes.getjTextfield().setText("");  
}  
}  
break;  
}  
componentes.getjComboBox().addActionListener(this);  
}  
}
```

La ejecución de la aplicación presenta la siguiente ventana:



En este ejemplo, se pretende mostrar al lector la implementación de un *JComboBox* en que se produce actividad de inserción y borrado de ítems individualmente, y eliminación de la totalidad de los ítems que alberga en ese momento. Se inserta el dato que previamente ha sido introducido al *JTextField*. Si se selecciona un ítem del *JComboBox*, éste es visualizado en el *JTextField*, que puede ser eliminado, si se desea, al pulsar el botón **Eliminar**.

Los tres *JButton* utilizados y el *JComboBox* se registran todos ellos en escucha *ActionListener*. De tal modo que nos encontramos que el método *actionPerformed()* debe implementar las acciones derivadas de la escucha de las dos clases de componentes: *JButton* y *JComboBox*. Circunstancia que ya se producía en el ejemplo anterior, pero ahora, y novedosamente en relación con ejemplos anteriores utilizamos un nuevo recurso para la detección del componente concreto que ha disparado el evento, concretamente el *ActionCommand*. Ello se realiza

estableciendo dicha propiedad para cada uno de ellos, tanto si es *JComboBox*, como si se trata de un *JButton*

```
componentes.getjComboBox().setActionCommand("combo");
jButtonAñadir.setActionCommand("botonAñadir");
```

y posteriormente darle un tratamiento homogéneo en la clase escucha con que se registran todos ellos (**GestorEventos**), con total independencia de la clase del componente:

```
public void actionPerformed(ActionEvent e) {
    switch(e.getActionCommand())
    { case "combo" : ACCIONES
        break;
    case "botonAñadir" : ACCIONES
        break;
    . . .
    }
```

Otra actuación novedosa que se produce en este ejemplo es la eliminación de un registro de escucha de eventos:

```
if
(componentes.getjComboBox().getActionListeners().length > 0)
componentes.getjComboBox().removeActionListener(this);
```

Para posteriormente proceder a registrarlo nuevamente:

```
componentes.getjComboBox().addActionListener(this);
```

Dichas actuaciones tienen lugar, concretamente, al principio y al final del método *actionPerformed()*, con el objeto de que tanto el añadido como la eliminación de ítems del *JComboBox* disparen un evento por dicho componente, cuando dicho evento no va a ser gestionado.

Esta actuación volverá a repetirse en la aplicación **GestionLibros**, concretamente en la clase **VistaFormulario**, y en

el método **inicializarPantalla()** con el componente referenciado desde **jComboBoxLibros**, exactamente con el mismo propósito.

Novedosamente, aparecen en este ejemplo métodos del *JTextField* para

- Establecer un contenido:

```
componentes.getjTextfield().setText("")
```

- Leer la cadena previamente introducida:

```
componentes.getjTextfield().getText()
```

Y métodos del *JComboBox* que permiten:

- Leer el ítem previamente seleccionado:

```
componentes.getjComboBox().getSelectedItem().toString()
```

- Obtener la posición del ítem seleccionado:

```
componentes.getjComboBox().getSelectedIndex()
```

- Añadir un ítem:

```
componentes.getjComboBox().addItem( NUEVO ITEM )
```

- Establecer un ítem como seleccionado:

```
componentes.getjComboBox().setSelectedIndex( ITEM  
A SELECCIONAR )
```

- Obtener el número de ítems:

```
componentes.getjComboBox().getItemCount()
```

- Eliminar un ítem determinado:

```
componentes.getjComboBox().removeItemAt( ITEM A  
ELIMINAR )
```

- Eliminar todos los ítems:

```
componentes.getjComboBox().removeAllItems()
```

# 16

## APLICACIÓN INTERFAZGRAFICA9

Seguimos con la misma estructura:

- *package* **interfazgrafica9** aglutina las clases:
  - InterfazGrafica9
  - Componentes
  - GestorEventos

### InterfazGrafica9

```
package interfazgrafica9;
import java.awt.EventQueue;
import javax.swing.DefaultListModel;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JList;
import javax.swing.JScrollPane;
import javax.swing.JTextField;
public class InterfazGrafica9 extends JFrame {
public InterfazGrafica9() {
setSize(700,500);
setTitle("Ejemplo utilización JList");
ubicarComponentes();
setVisible(true);
}
```

```
private void ubicarComponentes() {  
    setLayout(null);  
    Componentes componentes = new Componentes();  
    GestorEventos gestorEventos = new  
    GestorEventos(componentes);  
    addWindowListener(gestorEventos); // REGISTRO DE  
    ESCUCHA DE EVENTOS DE VENTANA  
    // CONFIGURACIÓN JList  
    componentes.setDefaultListModel(new  
    DefaultListModel());  
    componentes.setjList(new  
    JList(componentes.getDefaultListModel()));  
    JScrollPane jScrollPane = new  
    JScrollPane(componentes.getjList());  
    jScrollPane.setBounds(5, 150, 250, 80);  
    componentes.getjList().addListSelectionListener(gestor  
    Eventos); // REGISTRO DE ESCUCHA DE EVENTO  
    ListSelectionListener  
    add(jScrollPane);  
    // CONFIGURACIÓN JTextField  
    componentes.setjTextfield(new JTextField());  
    componentes.getjTextfield().setBounds(400, 150, 200,  
    20);  
    add(componentes.getjTextfield());  
    // CONFIGURACIÓN JButton jButtonAñadir  
    JButton jButtonAñadir = new JButton("Añadir");  
    jButtonAñadir.setBounds(285, 150, 90, 40);  
    jButtonAñadir.setActionCommand("botonAñadir");  
    jButtonAñadir.addActionListener(gestorEventos); //  
    REGISTRO DE ESCUCHA DE EVENTO DE BOTON  
    add(jButtonAñadir);  
    // CONFIGURACIÓN JButton jButtonVaciar  
    JButton jButtonVaciar = new JButton("Vaciar");  
    jButtonVaciar.setBounds(285, 200, 90, 40);  
    jButtonVaciar.setActionCommand("botonVaciar");  
    jButtonVaciar.addActionListener(gestorEventos); //  
    REGISTRO DE ESCUCHA DE EVENTO DE BOTON
```

```

add(jButtonVaciar);
// CONFIGURACIÓN JButton jButtonEliminar
JButton jButtonEliminar = new JButton("Eliminar");
jButtonEliminar.setBounds(285, 250, 90, 40);
jButtonEliminar.setActionCommand("botonEliminar");
jButtonEliminar.addActionListener(gestorEventos); // REGISTRO DE ESCUCHA DE EVENTO DE BOTON
add(jButtonEliminar);
}
public static void main(String[] args) {
EventQueue.invokeLater(new Runnable() {
@Override
public void run() {
new InterfazGrafica9();
}
});
}
}
}

```

## Componentes

```

package interfazgrafica9;
import javax.swing.DefaultListModel;
import javax.swing.JList;
import javax.swing.JTextField;
public class Componentes {
private JList jList;
private DefaultListModel defaultListModel;
private JTextField jTextField;
public JList getjList() {
return jList;
}
public void setjList(JList jList) {
this.jList = jList;
}
public DefaultListModel getDefaultListModel() {
return defaultListModel;
}

```

```
}

public void setDefaultListModel(DefaultListModel defaultListModel) {
    this.defaultListModel = defaultListModel;
}
public JTextField getjTextfield() {
    return jTextField;
}
public void setjTextfield(JTextField jTextField) {
    this.jTextfield = jTextField;
}
}
```

## GestorEventos

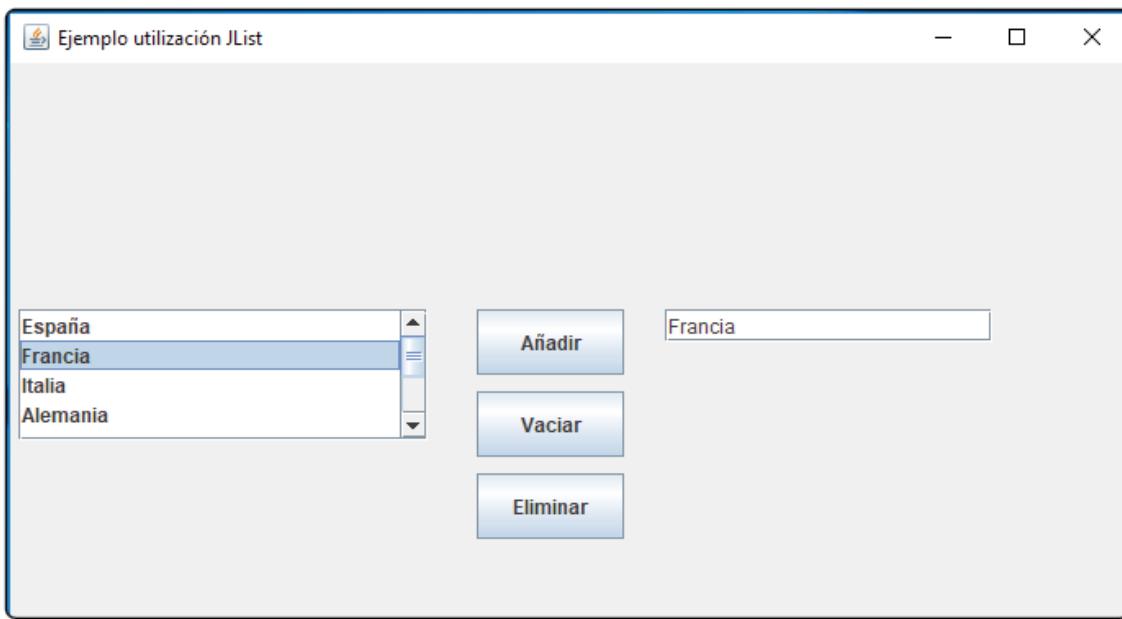
```
package interfazgrafica9;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JList;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
public class GestorEventos extends WindowAdapter
implements ActionListener, ListSelectionListener {
private Componentes componentes;
public GestorEventos(Componentes componentes) {
    this.componentes = componentes;
}
// MÉTODOS DE WindowAdapter
public void windowClosing(WindowEvent e) {
    System.exit(0);
}
// MÉTODO DE ActionListener
public void actionPerformed(ActionEvent e) {
    switch(e.getActionCommand())
{
```

```
case "botonAñadir" :
if
(componentes.getjTextfield().getText().compareTo("")!=
0)
{
componentes.getDefaultListModel().addElement(componentes.getjTextfield().getText());
componentes.getjList().setSelectedIndex(0);
componentes.getjTextfield().setText("");
}
break;
case "botonVaciar" :
if (componentes.getDefaultListModel().getSize()>0)
{
componentes.getDefaultListModel().removeAllElements();
componentes.getjTextfield().setText("");
}
break;
case "botonEliminar" :
if (componentes.getDefaultListModel().getSize()>0)
{
componentes.getDefaultListModel().remove(componentes.getjList().getSelectedIndex());
if (componentes.getDefaultListModel().getSize()>0)
{ componentes.getjList().setSelectedIndex(0); }
}
break;
}
}

// MÉTODO DE ListSelectionListener
public void valueChanged(ListSelectionEvent e) {
JList jList = (JList) e.getSource();
if (jList == componentes.getjList())
if (!componentes.getjList().isSelectionEmpty())
componentes.getjTextfield().setText(componentes.getjList().getSelectedValue().toString());
}
```

}

La ejecución de la aplicación muestra la siguiente ventana:



Este ejemplo supone una réplica del ejemplo anterior, con exactamente las mismas funcionalidades, pero ahora se pretende mostrar al lector la utilización, como componente aglutinador de ítems, un *JList*. A este componente, se le acopla un *JScrollPane* que, de ser necesario por la altura del componente y el número de ítems que contenga en cada momento, nos aporta una barra de desplazamiento en la visualización de los ítems del *JList*.

Novedosamente en relación con la escucha de eventos, con el *JList* se da la circunstancia de que se produce un registro de escucha *ListSelectionListener*, en consecuencia, hemos de aportar la correspondiente implementación del método de dicha clase escucha *valueChanged(ListSelectionEvent e)*.

Como aspectos susceptibles de matizar que aparecen novedosamente en este ejemplo, podemos mencionar:

- La necesidad de un *Model* que representa, almacena y permite la gestión de los datos que el componente se limita a visualizar. A lo largo de sucesivos ejemplos irán apareciendo otros *Model* que se acoplarán a sus correspondientes componentes. Para el caso que nos ocupa del *JList*, utilizamos en este ejemplo un *DefaultListModel*:

```
componentes.setDefaultListModel(new
DefaultListModel());
```

que acoplamos como argumento en el constructor:

```
componentes.setjList(new
JList(componentes.getDefaultListModel()));
```

- La barra de desplazamiento para visualización de los ítems del *JList*, modelada por el *JScrollPane*, que recibe en el constructor referencia a la instancia del *JList*:

```
JScrollPane jScrollPane = new
JScrollPane(componentes.getjList());
```

Y los métodos del *JList* que permiten:

- Leer el ítem previamente seleccionado:

```
componentes.getjList().getSelectedValue().toString
()
```

- Obtener la posición del ítem seleccionado:

```
componentes.getjList().getSelectedIndex()
```

- Añadir un ítem:

```
componentes.getDefaultListModel().addElement (
NUEVO ITEM )
```

- Establecer un ítem como seleccionado:

```
componentes.getjList().setSelectedIndex( ITEM A )
```

**SELECCIONAR )**

- Obtener el número de ítems:

```
componentes.getDefaultListModel().getSize()
```

- Eliminar un ítem determinado:

```
componentes.getDefaultListModel().remove ( ITEM A  
ELIMINAR )
```

- Eliminar todos los ítems:

```
componentes.getDefaultListModel().removeAllElements()
```

Podremos comprobar que la mayor parte de las actuaciones para el mantenimiento de los ítems aglutinados por el componente, es necesario ejercerlas por el *Model*.

# 17

## APLICACIÓN INTERFAZGRAFICA10

Se sigue manteniendo la misma estructura:

- *package* **interfazgrafica10** aglutina las clases:
  - InterfazGrafica10
  - Componentes
  - GestorEventos

### **InterfazGrafica10**

```
package interfazgrafica10;
import java.awt.EventQueue;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JProgressBar;
import javax.swing.JSlider;
import javax.swing.border.TitledBorder;
public class InterfazGrafica10 extends JFrame {
public InterfazGrafica10() {
setSize(700,500);
setTitle("JProgresBar combinado con JSlider");
Componentes componentes = new Componentes();
ubicarComponentes(componentes);
inicializarPantalla(componentes);
```

```
setVisible(true);
}
private void ubicarComponentes(Componentes componentes) {
componentes.setjFrame(this);
setLayout(null);
GestorEventos gestorEventos = new GestorEventos(componentes);
addWindowListener(gestorEventos); // REGISTRO DE ESCUCHA DE EVENTOS DE VENTANA
// CONFIGURACIÓN JSlider
componentes.setjSlider(new JSlider(JSlider.HORIZONTAL,0 ,250 ,125));
componentes.getjSlider().setBounds(20, 140, 600, 48);
componentes.getjSlider().setBorder(new TitledBorder("Dato numérico a establecer"));
componentes.getjSlider().setPaintTicks(true);
componentes.getjSlider().setMajorTickSpacing(20);
componentes.getjSlider().setMinorTickSpacing(5);
componentes.getjSlider().addChangeListener(gestorEventos); // REGISTRO DE ESCUCHA DE EVENTO ChangeListener
add(componentes.getjSlider());
// CONFIGURACIÓN JProgressBar
componentes.setjProgresBar(new JProgressBar());
componentes.getjProgresBar().setBounds(20, 300, 600, 30);
componentes.getjProgresBar().setMinimum(0);
componentes.getjProgresBar().setMaximum(250);
add(componentes.getjProgresBar());
// CONFIGURACIÓN JLabel
componentes.setjLabel(new JLabel());
componentes.getjLabel().setBounds(640, 160, 40, 20);
add(componentes.getjLabel());
// CONFIGURACIÓN JButton jVisualizaVentanaSecundaria
JButton jButtonVisualizarVentanaModal = new JButton("Visualizar ventana modal");
jButtonVisualizarVentanaModal.setBounds(220, 400, 180,
```

```
40);
jButtonVisualizarVentanaModal.addActionListener(gestor
Eventos); // REGISTRO DE ESCUCHA DE EVENTO DE BOTON
add(jButtonVisualizarVentanaModal);
}
private void inicializarPantalla(Componentes
componentes) {
componentes.getjLabel().setText(Integer.toString(compo
nentes.getjSlider().getValue()));
componentes.getjProgresBar().setValue(componentes.getj
Slider().getValue());
}
public static void main(String[] args) {
EventQueue.invokeLater(new Runnable() {
@Override
public void run() {
new InterfazGrafica10();
}
});
}
}
```

## Componentes

```
package interfazgrafica10;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JProgressBar;
import javax.swing.JSlider;
public class Componentes {
private JFrame jFrame;
private JSlider jSlider;
private JProgressBar jProgresBar;
private JLabel jLabel;
public JFrame getjFrame() {
return jFrame;
}
}
```

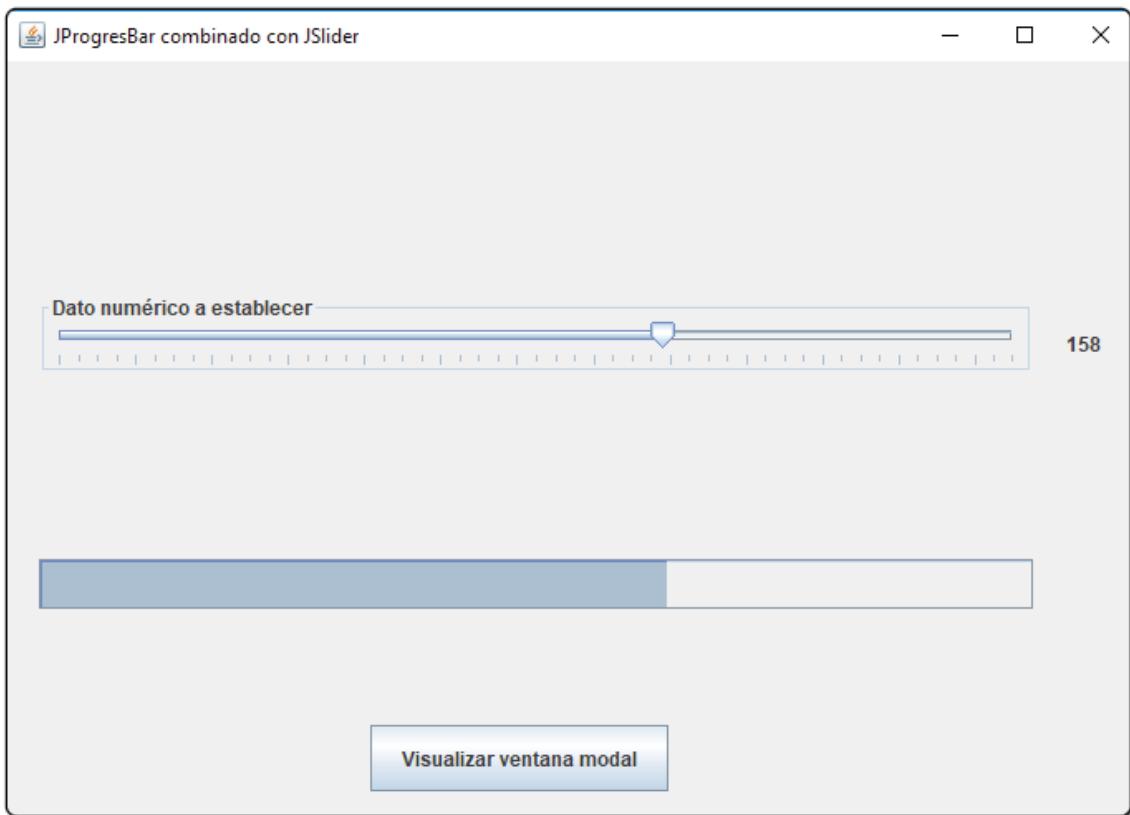
```
public void setjFrame(JFrame jFrame) {  
    this.jFrame = jFrame;  
}  
public JSlider getjSlider() {  
    return jSlider;  
}  
public void setjSlider(JSlider jSlider) {  
    this.jSlider = jSlider;  
}  
public JProgressBar getjProgresBar() {  
    return jProgresBar;  
}  
public void setjProgresBar(JProgressBar jProgresBar) {  
    this.jProgresBar = jProgresBar;  
}  
public JLabel getjLabel() {  
    return jLabel;  
}  
public void setjLabel(JLabel jLabel) {  
    this.jLabel = jLabel;  
}  
}
```

## GestorEventos

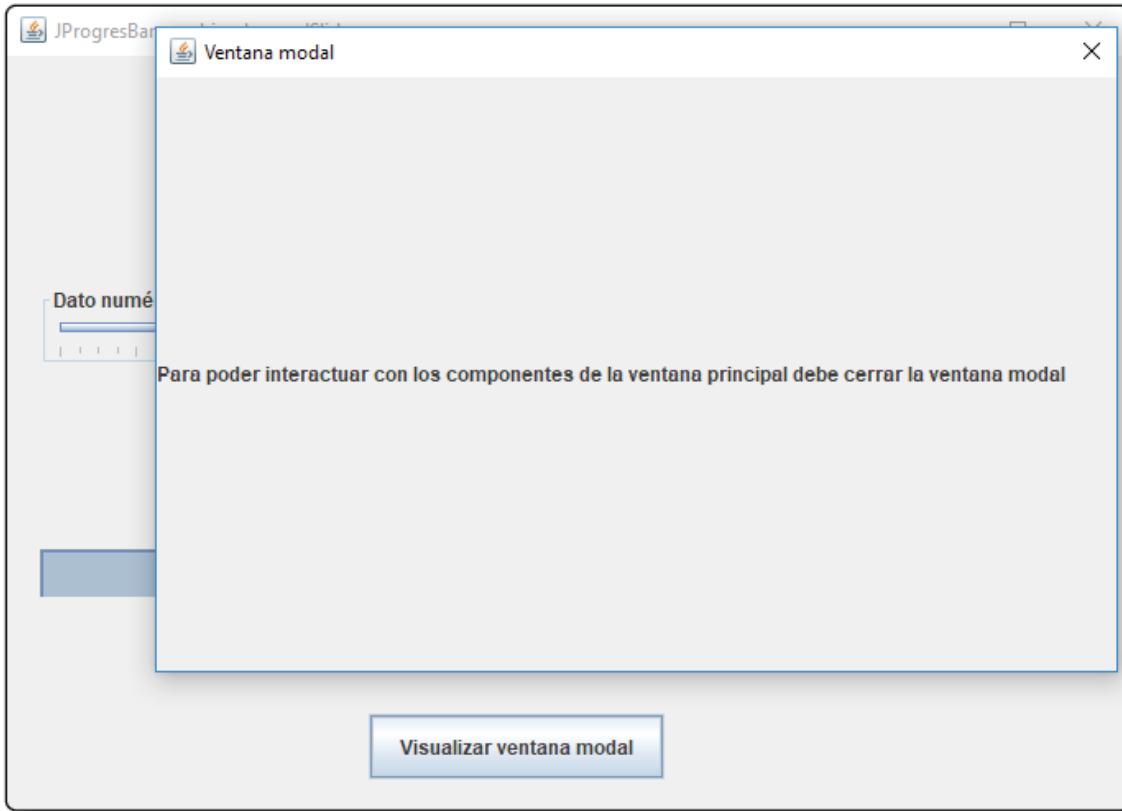
```
package interfazgrafica10;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.awt.event.WindowAdapter;  
import java.awt.event.WindowEvent;  
import javax.swing.JDialog;  
import javax.swing.JLabel;  
import javax.swing.event.ChangeEvent;  
import javax.swing.event.ChangeListener;  
public class GestorEventos extends WindowAdapter  
implements ActionListener, ChangeListener {  
    private Componentes componentes;
```

```
public GestorEventos(Componentes componentes) {  
    this.componentes = componentes;  
}  
// MÉTODOS DE WindowAdapter  
public void windowClosing(WindowEvent e) {  
    System.exit(0);  
}  
// MÉTODO DE ChangeListener  
public void stateChanged(ChangeEvent e) {  
    componentes.getjLabel().setText(Integer.toString(componentes.getjSlider().getValue()));  
    componentes.getjProgressBar().setValue(componentes.getjSlider().getValue());  
}  
// MÉTODO DE ActionListener  
public void actionPerformed(ActionEvent e) {  
    // No es necesario detectar el componente que disparó  
    // el evento ActionListener  
    // dado que en toda la aplicación solamente se ha  
    // registrado con esta escucha  
    // de eventos el JButton  
    jButtonVisualizarVentanaModal.  
    JDialog jDialog = new JDialog(componentes.getjFrame(),  
        "Ventana modal", true);  
    jDialog.setSize(600, 400);  
    JLabel jLabelMensaje = new JLabel("Para poder  
    interactuar con los componentes de la ventana  
    principal debe cerrar la ventana modal");  
    jDialog.add(jLabelMensaje);  
    jDialog.setVisible(true);  
}  
}
```

La ejecución del programa presenta la siguiente ventana:



Y cuando al pulsar el botón, aparece la ventana modal:



Este ejemplo muestra una actuación combinada entre un *JSlider* y un *JProgressBar*. La posición del cursor del *JSlider* en cada momento determina un valor numérico que se aplica al movimiento del *JProgressBar*, así como a la visualización en un *JLabel*, todo ello cuando se detecta el evento asociado al movimiento del *JSlider*, en el correspondiente método de la clase escucha. En la aplicación final **GestionLibros**, concretamente en la pantalla **VistaFormulario** se utiliza un *JSlider* para la introducción del dato número de páginas del libro.

El *JSlider* utilizado en este ejemplo es horizontal, tal y como vemos reflejado en el correspondiente argumento del constructor, así los valores de inicio, final, y la posición inicial del cursor:

```
componentes.setjSlider(new  
JSlider.JSlider.HORIZONTAL, 0 , 250 , 125));
```

Se prosigue configurando el *JSlider* con los métodos:

```
componentes.getjSlider().setPaintTicks(true);  
componentes.getjSlider().setMajorTickSpacing(20);  
componentes.getjSlider().setMinorTickSpacing(5);
```

La configuración del *JProgressBar* se establece con los métodos:

```
componentes.getjProgresBar().setMinimum(0);  
componentes.getjProgresBar().setMaximum(250);
```

En lo que a la gestión de eventos se refiere, el aspecto novedoso de este ejemplo es la escucha de eventos *ChangeListener* disparado por el movimiento del cursor del *JSlider*, y en consecuencia el método de escucha de este evento *stateChanged(ChangeEvent e)*.

Aprovechamos también este ejercicio para mostrar al lector cómo implementar una ventana modal mediante un *JDialog*, para ello es necesario utilizar el constructor de esta clase con el correspondiente argumento *boolean* a **true**. Dicha ventana modal se abre al pulsar el botón de la ventana principal. El lector podrá observar las acciones inherentes a todo ello en el método *actionPerformed()*.

Es recomendable que toda aplicación disponga de un solo *JFrame*, actuando como ventana principal. La clase *JDialog* ofrece la solución para la implementación de ventanas secundarias, caso de ser necesarias.

# 18

## APLICACIÓN INTERFAZGRAFICA11

Seguimos manteniendo la misma estructura:

- *package* **interfazgrafica11** aglutina las clases:
  - InterfazGrafica11
  - Componentes
  - GestorEventos

### **InterfazGrafica11**

```
package interfazgrafica11;
import java.awt.EventQueue;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JSpinner;
import javax.swing.SpinnerDateModel;
import javax.swing.SpinnerListModel;
import javax.swing.SpinnerNumberModel;
public class InterfazGrafica11 extends JFrame {
public InterfazGrafica11(){
Componentes componentes = new Componentes();
setSize(700,500);
```

```
setTitle("Ejemplo utilización JSpinner");
ubicarComponentes(componentes);
inicializar(componentes);
setVisible(true);
}
private void ubicarComponentes(Componentes componentes) {
setLayout(null);
GestorEventos gestorEventos = new
GestorEventos(componentes);
addWindowListener(gestorEventos); // REGISTRO DE
ESCUCHA DE EVENTOS DE VENTANA
// CONFIGURACIÓN JSpinner jSpinnerNumeros
SpinnerNumberModel spinnerNumberModel = new
SpinnerNumberModel(3, 0, 30, 5);
spinnerNumberModel.addChangeListener(gestorEventos);
componentes.setjSpinnerNumeros(new
JSpinner(spinnerNumberModel));
componentes.getjSpinnerNumeros().setBounds(150, 100,
50, 30);
add(componentes.getjSpinnerNumeros());
// CONFIGURACIÓN JSpinner jSpinnerGeneros
String[] generos = {"0 - Generalidades", "1 -
Filosofia. Psicologia", "2 - Religion. Teologia", "3 -
Ciencias sociales", "4 - No clasificado", "5 -
Matematicas. Ciencias naturales", "6 - Ciencias
aplicadas", "7 - Bellas artes. Deportes", "8 -
Literatura", "9 - Geografia. Historia"};
SpinnerListModel spinnerListModel = new
SpinnerListModel(generos);
spinnerListModel.addChangeListener(gestorEventos);
componentes.setjSpinnerGeneros(new
JSpinner(spinnerListModel));
componentes.getjSpinnerGeneros().setBounds(150, 200,
250, 30);
add(componentes.getjSpinnerGeneros());
// CONFIGURACIÓN JSpinner jSpinnerFechas
```

```
Date fechaActual = new Date();
/*
// La siguiente configuración del SpinnerDateModel
permite establecer límites inferior y superior al
rango de fechas accesibles desde el Spinner.
// Con estos parámetros, para que no se produzca
error, es necesario supervisar en el establecimiento de
dichas fechas límites que se encuentran dentro del
mismo mes que fecha actual.
// Dicha supervisión aparece implementada a efectos de
no incrementar la extensión del código.
Date fechaInicial = new Date();
int diasPreviosMaximo = 2;
fechaInicial.setTime(fechaActual.getTime() - (long)
(24*60*60*1000*diasPreviosMaximo));
Date fechaFinal = new Date();
int diasPosterioresMaximo = 2;
fechaFinal.setTime(fechaActual.getTime() + (long)
(24*60*60*1000*diasPosterioresMaximo));
SpinnerDateModel spinnerDateModel = new
SpinnerDateModel(fechaActual, fechaInicial,
fechaFinal, Calendar.YEAR);
*/
SpinnerDateModel spinnerDateModel = new
SpinnerDateModel(fechaActual, null, null,
Calendar.YEAR);
spinnerDateModel.addChangeListener(gestorEventos);
componentes.setjSpinnerFechas(new
JSpinner(spinnerDateModel));
componentes.getjSpinnerFechas().setBounds(150, 300,
100, 30);
componentes.getjSpinnerFechas().setEditor(new
JSpinner.DateEditor(componentes.getjSpinnerFechas(),
"dd-MM-yyyy"));
add(componentes.getjSpinnerFechas());
componentes.setjLabelNumeros(new JLabel());
componentes.getjLabelNumeros().setBounds(150, 150,
```

```
300, 30);
add(componentes.getjLabelNumeros());
componentes.setjLabelGeneros(new JLabel());
componentes.getjLabelGeneros().setBounds(150, 250,
500, 30);
add(componentes.getjLabelGeneros());
componentes.setjLabelFechas(new JLabel());
componentes.getjLabelFechas().setBounds(150, 350, 300,
30);
add(componentes.getjLabelFechas());
}
private void inicializar(Componentes componentes) {
componentes.getjLabelNumeros().setText("Dato
seleccionado en el JSpinner : " +
((Integer)componentes.getjSpinnerNumeros().getValue())
.toString()); // Es innecesario el cast a Integer pero
permite dejar constancia de que este JSpinner devuelve
un objeto de esta clase
componentes.getjLabelGeneros().setText("Dato
seleccionado en el JSpinner : " +
(String)componentes.getjSpinnerGeneros().getValue());
componentes.getjLabelFechas().setText("Dato
seleccionado en el JSpinner : " + new
SimpleDateFormat("dd-MM-
yyyy").format((java.util.Date)componentes.getjSpinnerF
echas().getValue()));
}
public static void main(String[] args) {
EventQueue.invokeLater(new Runnable() {
@Override
public void run() {
new InterfazGrafica11();
}
});
}
}
```

## Componentes

```
package interfazgrafica11;
import javax.swing.JLabel;
import javax.swing.JSpinner;
public class Componentes {
    private JSpinner jSpinnerNumeros;
    private JSpinner jSpinnerGeneros;
    private JSpinner jSpinnerFechas;
    private JLabel jLabelNumeros;
    private JLabel jLabelGeneros;
    private JLabel jLabelFechas;
    public JSpinner getjSpinnerNumeros() {
        return jSpinnerNumeros;
    }
    public void setjSpinnerNumeros(JSpinner jSpinnerNumeros) {
        this.jSpinnerNumeros = jSpinnerNumeros;
    }
    public JSpinner getjSpinnerGeneros() {
        return jSpinnerGeneros;
    }
    public void setjSpinnerGeneros(JSpinner jSpinnerGeneros) {
        this.jSpinnerGeneros = jSpinnerGeneros;
    }
    public JSpinner getjSpinnerFechas() {
        return jSpinnerFechas;
    }
    public void setjSpinnerFechas(JSpinner jSpinnerFechas) {
        this.jSpinnerFechas = jSpinnerFechas;
    }
    public JLabel getjLabelNumeros() {
        return jLabelNumeros;
    }
    public void setjLabelNumeros(JLabel jLabelNumeros) {
        this(jLabelNumeros = jLabelNumeros;
```

```
}

public JLabel getjLabelGeneros() {
    return jLabelGeneros;
}

public void setjLabelGeneros(JLabel jLabelGeneros) {
    this.jLabelGeneros = jLabelGeneros;
}

public JLabel getjLabelFechas() {
    return jLabelFechas;
}

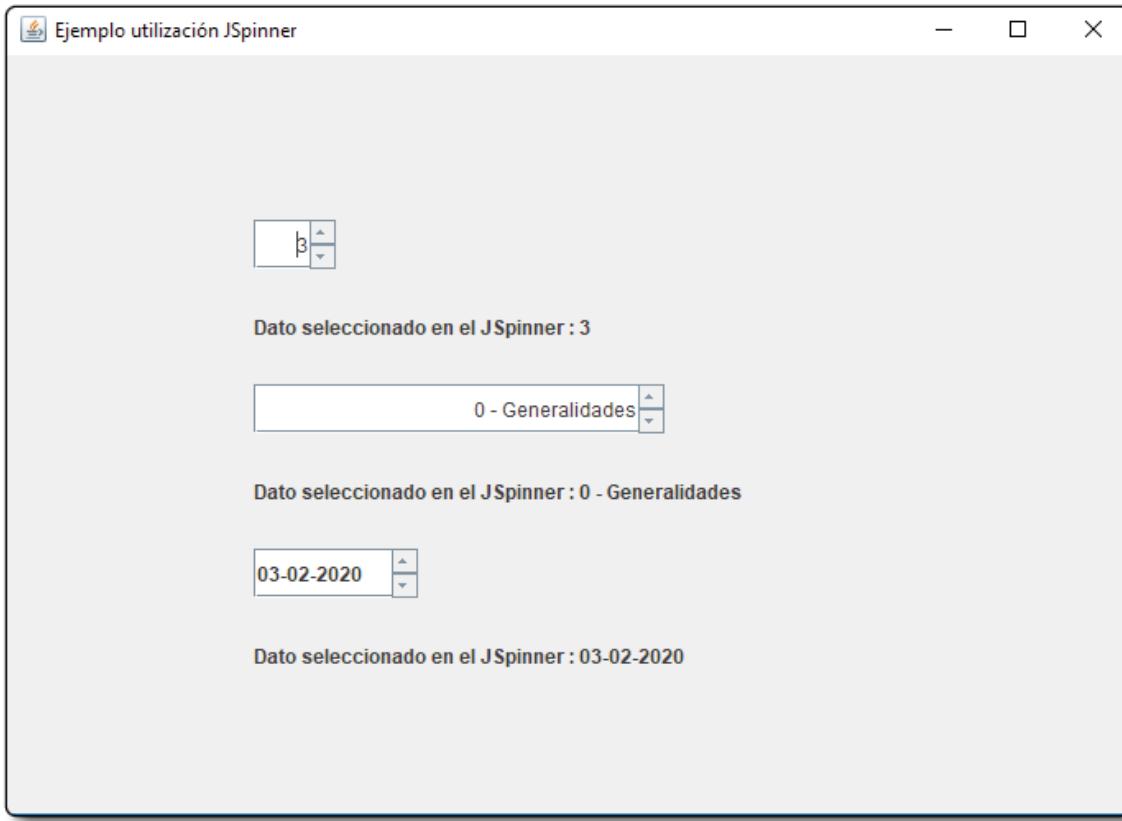
public void setjLabelFechas(JLabel jLabelFechas) {
    this.jLabelFechas = jLabelFechas;
}
}
```

## GestorEventos

```
package interfazgrafica11;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.text.SimpleDateFormat;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
public class GestorEventos extends WindowAdapter
implements ChangeListener {
private Componentes componentes;
public GestorEventos(Componentes componentes) {
this.componentes = componentes;
}
// MÉTODOS DE WindowAdapter
public void windowClosing(WindowEvent e) {
System.exit(0);
}
// MÉTODO DE ChangeListener
public void stateChanged(ChangeEvent e)
{
switch(e.getSource().getClass().getName())
```

```
{  
case "javax.swing.SpinnerNumberModel":  
componentes.getjLabelNumeros().setText("Dato  
seleccionado en el JSpinner : " +  
((Integer)componentes.getjSpinnerNumeros().getValue())  
.toString()); // Es innecesario el cast a Integer pero  
permite dejar constancia de que este JSpinner devuelve  
un objeto de esta clase  
break;  
case "javax.swing.SpinnerListModel":  
componentes.getjLabelGeneros().setText("Dato  
seleccionado en el JSpinner : " +  
(String)componentes.getjSpinnerGeneros().getValue());  
break;  
case "javax.swing.SpinnerDateModel":  
componentes.getjLabelFechas().setText("Dato  
seleccionado en el JSpinner : " + new  
SimpleDateFormat("dd-MM-  
yyyy").format((java.util.Date)componentes.getjSpinnerF  
echas().getValue()));  
break;  
}  
}  
}
```

La ejecución de la aplicación presenta la siguiente ventana:



Este ejemplo se dedica al *JSpinner*. Se trata de un componente que permite interactuar con un conjunto de datos previamente asignado modelado por uno de los tres tipos de *Model* disponibles para el *JSpinner*:

- SpinnerNumberModel
- SpinnerListModel
- SpinnerDateModel

El ejemplo presenta tres *JSpinner*, a cada uno de los cuales se le asigna cada una de las tres clases de *Model* descritas. En cualquiera de los tres casos, la asignación del *Model* al *JSpinner* se realiza transfiriéndolo como argumento al constructor:

```
new JSpinner( MODELO )
```

Así mismo, el cambio de dato seleccionado dispara un evento *ChangeListener*, que en el ejemplo, es debidamente tratado por el correspondiente método de evento *stateChanged(ChangeEvent e)*. En la implementación de dicho método determinamos el componente origen del evento por el *Model* que tiene asignado mediante:

```
switch(e.getSource().getClass().getName())
{
case "javax.swing.SpinnerNumberModel":
    . . .
break;
case "javax.swing.SpinnerListModel":
    . . .
break;
case "javax.swing.SpinnerDateModel":
    . . .
break;
}
```

Las acciones que en este ejemplo se ejecutan como consecuencia del cambio del valor seleccionado, y en consecuencia disparo del evento, son visualizar en un *JLabel* el nuevo valor de la selección. Detallamos a continuación cómo extraemos los citados valores en cada una de las tres variantes:

```
((Integer)componentes.getjSpinnerNumeros().getValue())
.toString() // Es innecesario el cast a Integer pero
permite dejar constancia de que este JSpinner devuelve
un objeto de esta clase
(String)componentes.getjSpinnerGeneros().getValue()
new SimpleDateFormat("dd-MM-
yyyy").format((java.util.Date)componentes.getjSpinnerF
echas().getValue())
```

Al *JSpinner* que permite obtener un valor de una secuencia de valores numéricos mediante un *SpinnerNumberModel* ha sido

parametrizado con un valor actual inicial 3, y con un incremento o decremento en los valores de la secuencia de 5:

```
SpinnerNumberModel spinnerNumberModel = new  
SpinnerNumberModel(3, 0, 30, 5);
```

En la inmensa mayoría de las ocasiones, el valor actual inicial será 0 ó 1, y los incrementos-decrementos de 1 en 1. Los parámetros “insólitos” utilizados en el ejemplo se han aplicado con la intención de evidenciar la posibilidad de parametrizar los valores inicial e incremento, además de los valores límite.

Como se podrá observar siguiendo el código, un *SpinnerListModel* se inicializa asignándole un *String[]*:

```
SpinnerListModel spinnerListModel = new  
SpinnerListModel(generos);
```

En el caso del ejemplo se le ha asignado el array de géneros que en la aplicación **GestionLibros** se utiliza para asignar a un libro el género que le corresponda. En dicha aplicación se utiliza, según la vista de que se trate, en la selección del género en cuestión un *JComboBox*, un *JList*, o cada género constituye un subárbol con todos los libros a los que se les asigna dicho género dentro del *JTree* principal. El *JSpinner* de este ejemplo que permite interactuar con el *SpinnerListModel*, no contemplado en la aplicación **GestionLibros** podría haber sido perfectamente una posibilidad diferente de selección del género.

En relación con el *JSpinner* respaldado por el *SpinnerDateModel* lo inicializamos:

```
SpinnerDateModel spinnerDateModel = new  
SpinnerDateModel(fechaActual, null, null,  
Calendar.YEAR);
```

No hemos aportado valores para los límites inicial y final, pero presentamos comentarizado lo que supondría un

*SpinnerDateModel* al que sí se le establecen los valores límite.

Podemos establecer un formato para la presentación de las fechas a visualizar en el *JSpinner*:

```
componentes.getjSpinnerFechas().setEditor(new  
JSpinner.DateEditor(componentes.getjSpinnerFechas(),  
"dd-MM-yyyy"));
```

Podría ser la variante de modelo *SpinnerDateModel* a gestionar por el *JSpinner* una solución muy apropiada en aplicaciones en que es necesario recoger la fecha del sistema, pero que en ocasiones está sometida a ligeras aproximaciones. Podríamos poner el ejemplo de la facturación de alojamiento en un hotel, en que la fecha de salida del alojamiento, constituye objeto del cálculo del número de días de la estancia al restarle la fecha de entrada (aritmética de fechas, nos remitimos al libro PROGRAMACION ORIENTADA A OBJETOS EN JAVA del autor). Supongamos que dicha fecha de salida se basa en la fecha del sistema, y que la condición de que dicha fecha del sistema coincida con el dato es que el cliente abandone la habitación y se presente en la recepción del hotel a entregar las llaves antes de las 12:00 horas. Si un cliente se persona más tarde de dicha hora se considera que ocupa la habitación un día más, con lo que habrá que facturarle un día más, y, en consecuencia, la fecha de salida deberá ser la del sistema del día siguiente. O, el proceso inverso, es decir, que el cliente se persona en la noche anterior a la salida para liquidar su estancia, la fecha de salida también deberá ser la del sistema del día siguiente.

# 19

## APLICACIÓN INTERFAZGRAFICA12

En este ejemplo aportamos una sola clase:

- *package* **interfazgrafica12** aglutina la clase:
  - InterfazGrafica12

### InterfazGrafica12

```
package interfazgrafica12;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.EventQueue;
import java.awt.FlowLayout;
import java.awt.GridLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.WindowConstants;
public class InterfazGrafica12 extends JFrame {
public InterfazGrafica12() {
Dimension dimension = getToolkit().getScreenSize();
setSize(dimension);
setMinimumSize(new Dimension(700, 500));
setExtendedState(this.MAXIMIZED_BOTH);
```

```
setTitle("Prueba Layouts");
ubicarComponentes();
setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
);
pack();
setVisible(true);
}
private void ubicarComponentes() {
// JFrame configurado BorderLayout
setLayout(new BorderLayout());
JButton[] jButton = new JButton[10];
JLabel[] jLabel = new JLabel[20];
// JPanel ocupa BorderLayout.NORTH configurado
FlowLayout
JPanel jPanel1 = new JPanel();
jPanel1.setLayout(new FlowLayout());
jPanel1.setBackground(Color.CYAN);
add(jPanel1, BorderLayout.NORTH);
for (int i=0; i<3; i++)
{
jButton[i] = new JButton("botón "+i);
jPanel1.add(jButton[i]);
}
// JPanel ocupa BorderLayout.SOUTH configurado
FlowLayout
JPanel jPanel2 = new JPanel();
jPanel2.setLayout(new FlowLayout());
jPanel2.setBackground(Color.CYAN);
add(jPanel2, BorderLayout.SOUTH);
for (int i=3; i<6; i++)
{
jButton[i] = new JButton("botón "+i);
jPanel2.add(jButton[i]);
}
// JPanel ocupa BorderLayout.WEST configurado
FlowLayout
JPanel jPanel3 = new JPanel();
```

```
jPanel3.setLayout(new FlowLayout());
jPanel3.setBackground(Color.green);
add(jPanel3, BorderLayout.WEST);
for (int i=6; i<8; i++)
{
jButton[i] = new JButton("botón "+i);
jPanel3.add(jButton[i]);
}
// JPanel ocupa BorderLayout.EAST configurado
FlowLayout
JPanel jPanel4 = new JPanel();
jPanel4.setLayout(new FlowLayout());
jPanel4.setBackground(Color.green);
add(jPanel4, BorderLayout.EAST);
for (int i=8; i<10; i++)
{
jButton[i] = new JButton("botón "+i);
jPanel4.add(jButton[i]);
}
// JPanel ocupa BorderLayout.CENTER configurado
GridLayout
JPanel jPanel5 = new JPanel();
jPanel5.setLayout(new GridLayout(5, 4));
add(jPanel5, BorderLayout.CENTER);
for (int i=0; i<20; i++)
{
jLabel[i] = new JLabel("etiqueta "+i);
jPanel5.add(jLabel[i]);
}
}
}

public static void main(String[] args) {
EventQueue.invokeLater(new Runnable() {
@Override
public void run() {
new InterfazGrafica12();
}
});
}
```

```
}
```

La ejecución de la aplicación presenta una ventana que presentamos a continuación:



Pretendemos tratar en esta aplicación ejemplo dos aspectos que se han venido obviando en el resto de los ejercicios por cuestiones de claridad en el tratamiento de los conceptos que se pretendían abordar en cada momento, pero que no podemos soslayar en el presente libro. Aunque se le presenten al lector de manera muy escueta en el presente ejemplo, sí que son cuestiones que debe conocer un programador que utilice el API SWING. Dichos aspectos son:

- Ajuste del tamaño de la ventana principal

- Administradores de diseño (*Layouts*)

En lo que al ajuste del tamaño de la ventana principal se refiere, se realizan las siguientes actuaciones:

- Se obtiene el tamaño de pantalla de la consola que se está utilizando para ejecutar la aplicación:

```
Dimension dimension =
getToolkit().getScreenSize();
```

- Se establece el tamaño de la ventana principal (*JFrame*) a las dimensiones obtenidas en el paso anterior:

```
setSize(dimension);
```

- En previsión de los reajustes del tamaño de la ventana que pueda realizar el usuario, establecemos una dimensión mínima del *JFrame* que permita que no quede oculto ninguno de los *Component* que integran la ventana:

```
setMinimumSize(new Dimension(700, 500));
```

- Fijamos como preferencia que la ventana aparezca maximizada a todo el tamaño de la pantalla mediante la siguiente actuación:

```
setExtendedState(this.MAXIMIZED_BOTH);
```

Para mostrar al lector la utilización de *Layouts*, hemos aplicado en este ejemplo las siguientes distribuciones:

- A la ventana principal (*JFrame*) se le aplica *BorderLayout*, y en cada una de las zonas del mismo se añade un *JPanel*, diferenciándolos por colores para mejor perspectiva del lector.

- A los cuatro *JPanel* periféricos del *BorderLayout*, se les aplica *FlowLayout*, y se añaden JButton a todos ellos a efectos de que el lector pueda observar la distribución de los componentes consecuencia del *Layout* aplicado.
- Al *JPanel* central se le aplica *GridLayout*, añadiendo un *JLabel* a cada una de las subdivisiones generadas.

# 20

## APLICACIÓN MENUJTABBEDPANE

Nos encontramos con una estructura ampliada respecto a lo que viene siendo habitual:

- *package presentacion* aglutina las clases:
  - Menu
  - Componentes
  - GestorEventos
  - Opcion1
  - Opcion2
  - Opcion3

### Menu

```
package presentacion;
import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JTabbedPane;
public class Menu extends JFrame {
private Componentes componentes;
public Menu() {
setSize(700,500);
setTitle("Menu JTabbedPane");
```

```
componentes = new Componentes();
componentes.setGestorEventos(new
GestorEventos(componentes));
ubicarComponentes();
inicializar();
setVisible(true);
}
private void ubicarComponentes() {
addWindowListener(componentes.getGestorEventos()); //  
REGISTRO DE ESCUCHA DE EVENTOS DE VENTANA
componentes.setjTabbedPane(new JTabbedPane());
componentes.getjTabbedPane().setTabLayoutPolicy(JTabbe
dPane.SCROLL_TAB_LAYOUT);
componentes.setOpcion1(new Opcion1(componentes));
componentes.getjTabbedPane().add("Opcion 1",
componentes.getOpcion1());
componentes.setOpcion2(new Opcion2(componentes));
componentes.getjTabbedPane().add("Opcion 2",
componentes.getOpcion2());
componentes.setOpcion3(new Opcion3(componentes));
componentes.getjTabbedPane().add("Opcion 3",
componentes.getOpcion3());
componentes.getjTabbedPane().addChangeListener(compone
ntes.getGestorEventos());
add(componentes.getjTabbedPane());
}
private void inicializar() {
componentes.getOpcion1().iniciarPantalla();
componentes.getjTabbedPane().setSelectedIndex(0);
}
public static void main(String[] args) {
EventQueue.invokeLater(new Runnable() {
@Override
public void run() {
new Menu();
}
});
}
```

```
}
```

## Componentes

```
package presentacion;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JTabbedPane;
public class Componentes {
    private GestorEventos gestorEventos;
    private JTabbedPane jTabbedPane;
    private Opcion1 opcion1;
    private Opcion2 opcion2;
    private Opcion3 opcion3;
    private JButton jButtonOpcion1;
    private JLabel jLabelParcialOpcion1;
    private JLabel jLabelGlobalOpcion1;
    private JButton jButtonOpcion2;
    private JLabel jLabelParcialOpcion2;
    private JLabel jLabelGlobalOpcion2;
    private JButton jButtonOpcion3;
    private JLabel jLabelParcialOpcion3;
    private JLabel jLabelGlobalOpcion3;
    private int contadorOpcion1 = 0;
    private int contadorOpcion2 = 0;
    private int contadorOpcion3 = 0;
    private int contadorGlobal = 0;
    public GestorEventos getGestorEventos() {
        return gestorEventos;
    }
    public void setGestorEventos(GestorEventos
        gestorEventos) {
        this.gestorEventos = gestorEventos;
    }
    public JTabbedPane getjTabbedPane() {
        return jTabbedPane;
    }
}
```

```
}

public void setjTabbedPane(JTabbedPane jTabbedPane) {
this.jTabbedPane = jTabbedPane;
}
public Opcion1 getOpcion1() {
return opcion1;
}
public void setOpcion1(Opcion1 opcion1) {
this.opcion1 = opcion1;
}
public Opcion2 getOpcion2() {
return opcion2;
}
public void setOpcion2(Opcion2 opcion2) {
this.opcion2 = opcion2;
}
public Opcion3 getOpcion3() {
return opcion3;
}
public void setOpcion3(Opcion3 opcion3) {
this.opcion3 = opcion3;
}
public JButton getjButtonOpcion1() {
return jButtonOpcion1;
}
public void setjButtonOpcion1(JButton jButtonOpcion1)
{
this(jButtonOpcion1 = jButtonOpcion1;
}
public JLabel getjLabelParcialOpcion1() {
return jLabelParcialOpcion1;
}
public void setjLabelParcialOpcion1(JLabel
jLabelParcialOpcion1) {
this.jLabelParcialOpcion1 = jLabelParcialOpcion1;
}
public JLabel getjLabelGlobalOpcion1() {
```

```
return jLabelGlobalOpcion1;
}
public void setjLabelGlobalOpcion1(JLabel
jLabelGlobalOpcion1) {
this.jLabelGlobalOpcion1 = jLabelGlobalOpcion1;
}
public JButton getjButtonOpcion2() {
return jButtonOpcion2;
}
public void setjButtonOpcion2(JButton jButtonOpcion2)
{
this(jButtonOpcion2 = jButtonOpcion2;
}
public JLabel getjLabelParcialOpcion2() {
return jLabelParcialOpcion2;
}
public void setjLabelParcialOpcion2(JLabel
jLabelParcialOpcion2) {
this.jLabelParcialOpcion2 = jLabelParcialOpcion2;
}
public JLabel getjLabelGlobalOpcion2() {
return jLabelGlobalOpcion2;
}
public void setjLabelGlobalOpcion2(JLabel
jLabelGlobalOpcion2) {
this.jLabelGlobalOpcion2 = jLabelGlobalOpcion2;
}
public JButton getjButtonOpcion3() {
return jButtonOpcion3;
}
public void setjButtonOpcion3(JButton jButtonOpcion3)
{
this(jButtonOpcion3 = jButtonOpcion3;
}
public JLabel getjLabelParcialOpcion3() {
return jLabelParcialOpcion3;
}
```

```
public void setjLabelParcialOpcion3(JLabel
jLabelParcialOpcion3) {
this.jLabelParcialOpcion3 = jLabelParcialOpcion3;
}
public JLabel getjLabelGlobalOpcion3() {
return jLabelGlobalOpcion3;
}
public void setjLabelGlobalOpcion3(JLabel
jLabelGlobalOpcion3) {
this.jLabelGlobalOpcion3 = jLabelGlobalOpcion3;
}
public int getContadorOpcion1() {
return contadorOpcion1;
}
public void setContadorOpcion1(int contadorOpcion1) {
this.contadorOpcion1 = contadorOpcion1;
}
public int getContadorOpcion2() {
return contadorOpcion2;
}
public void setContadorOpcion2(int contadorOpcion2) {
this.contadorOpcion2 = contadorOpcion2;
}
public int getContadorOpcion3() {
return contadorOpcion3;
}
public void setContadorOpcion3(int contadorOpcion3) {
this.contadorOpcion3 = contadorOpcion3;
}
public int getContadorGlobal() {
return contadorGlobal;
}
public void setContadorGlobal(int contadorGlobal) {
this.contadorGlobal = contadorGlobal;
}
}
```

## GestorEventos

```
package presentacion;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JTabbedPane;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
public class GestorEventos extends WindowAdapter
implements ChangeListener, ActionListener {
private Componentes componentes;
public GestorEventos(Componentes componentes) {
this.componentes = componentes;
}
// MÉTODOS DE WindowAdapter
public void windowClosing(WindowEvent e) {
System.exit(0);
}
// MÉTODO DE ChangeListener
public void stateChanged(ChangeEvent e) {
JTabbedPane componenteFuente=(JTabbedPane)
e.getSource();
if (componenteFuente == componentes.getjTabbedPane())
{
switch(componentes.getjTabbedPane().getSelectedIndex())
{
case 0: componentes.getOpcion1().iniciarPantalla();
break;
case 1: componentes.getOpcion2().iniciarPantalla();
break;
case 2: componentes.getOpcion3().iniciarPantalla();
break;
}
}
}
}
// MÉTODO DE ActionListener
```

```
public void actionPerformed(ActionEvent e) {  
switch(e.getActionCommand())  
{  
case "botonOpcion1" :  
componentes.setContadorOpcion1(componentes.getContador  
Opcion1() + 1);  
componentes.getjLabelParcialOpcion1().setText("Pulsaci  
ones botón opción 1:  
"+componentes.getContadorOpcion1());  
componentes.setContadorGlobal(componentes.getContadorG  
lobal() + 1);  
componentes.getjLabelGlobalOpcion1().setText("Pulsacio  
nes acumuladas de los botones de todas las opciones:  
"+componentes.getContadorGlobal());  
break;  
case "botonOpcion2" :  
componentes.setContadorOpcion2(componentes.getContador  
Opcion2() + 1);  
componentes.getjLabelParcialOpcion2().setText("Pulsaci  
ones botón opción 2:  
"+componentes.getContadorOpcion2());  
componentes.setContadorGlobal(componentes.getContadorG  
lobal() + 1);  
componentes.getjLabelGlobalOpcion2().setText("Pulsacio  
nes acumuladas de los botones de todas las opciones:  
"+componentes.getContadorGlobal());  
break;  
case "botonOpcion3" :  
componentes.setContadorOpcion3(componentes.getContador  
Opcion3() + 1);  
componentes.getjLabelParcialOpcion3().setText("Pulsaci  
ones botón opción 3:  
"+componentes.getContadorOpcion3());  
componentes.setContadorGlobal(componentes.getContadorG  
lobal() + 1);  
componentes.getjLabelGlobalOpcion3().setText("Pulsacio  
nes acumuladas de los botones de todas las opciones:
```

```
“+componentes.getContadorGlobal());
break;
}
}
}
```

## Opcion1

```
package presentacion;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
public class Opcion1 extends JPanel {
private Componentes componentes;
public Opcion1(Componentes componentes) {
this.componentes = componentes;
ubicarComponentes();
}
private void ubicarComponentes() {
setLayout(null);
// CONFIGURACIÓN JButton
componentes.setjButtonOpcion1(new JButton("Contar"));
componentes.getjButtonOpcion1().setBounds(255, 220,
100, 40);
componentes.getjButtonOpcion1().setActionCommand("botonOpcion1");
componentes.getjButtonOpcion1().addActionListener(componentes.getGestorEventos()); // REGISTRO DE ESCUCHA DE EVENTO DE BOTON
add(componentes.getjButtonOpcion1());
// CONFIGURACIÓN JLabel visualización contador parcial
componentes.setjLabelParcialOpcion1(new JLabel());
componentes.getjLabelParcialOpcion1().setBounds(100,
100, 400, 20);
add(componentes.getjLabelParcialOpcion1());
// CONFIGURACIÓN JLabel visualización contador global
componentes.setjLabelGlobalOpcion1(new JLabel());
```

```

componentes.getLabelGlobalOpcion1().setBounds(100,
150, 400, 20);
add(componentes.getLabelGlobalOpcion1());
}
public void iniciarPantalla() {
componentes.setContadorOpcion1(0);
componentes.getLabelParcialOpcion1().setText("Pulsaciones botón opción 1:
"+componentes.getContadorOpcion1());
componentes.getLabelGlobalOpcion1().setText("Pulsaciones acumuladas de los botones de todas las opciones:
"+componentes.getContadorGlobal());
}
}

```

## Opcion2

```

package presentacion;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
public class Opcion2 extends JPanel {
private Componentes componentes;
public Opcion2(Componentes componentes) {
this.componentes = componentes;
ubicarComponentes();
}
private void ubicarComponentes() {
setLayout(null);
// CONFIGURACIÓN JButton
componentes.set JButtonOpcion2(new JButton("Contar"));
componentes.get JButtonOpcion2().setBounds(255, 220,
100, 40);
componentes.get JButtonOpcion2().setActionCommand("botonOpcion2");
componentes.get JButtonOpcion2().addActionListener(componentes.getGestorEventos()); // REGISTRO DE ESCUCHA DE
}

```

```

EVENTO DE BOTON
add(componentes.getjButtonOpcion2());
// CONFIGURACIÓN JLabel visualización contador parcial
componentes.setjLabelParcialOpcion2(new JLabel());
componentes.getjLabelParcialOpcion2().setBounds(100,
100, 400, 20);
add(componentes.getjLabelParcialOpcion2());
// CONFIGURACIÓN JLabel visualización contador global
componentes.setjLabelGlobalOpcion2(new JLabel());
componentes.getjLabelGlobalOpcion2().setBounds(100,
150, 400, 20);
add(componentes.getjLabelGlobalOpcion2());
}
public void iniciarPantalla() {
componentes.setContadorOpcion2(0);
componentes.getjLabelParcialOpcion2().setText("Pulsaciones botón opción 2:
"+componentes.getContadorOpcion2());
componentes.getjLabelGlobalOpcion2().setText("Pulsaciones acumuladas de los botones de todas las opciones:
"+componentes.getContadorGlobal());
}
}

```

### Opcion3

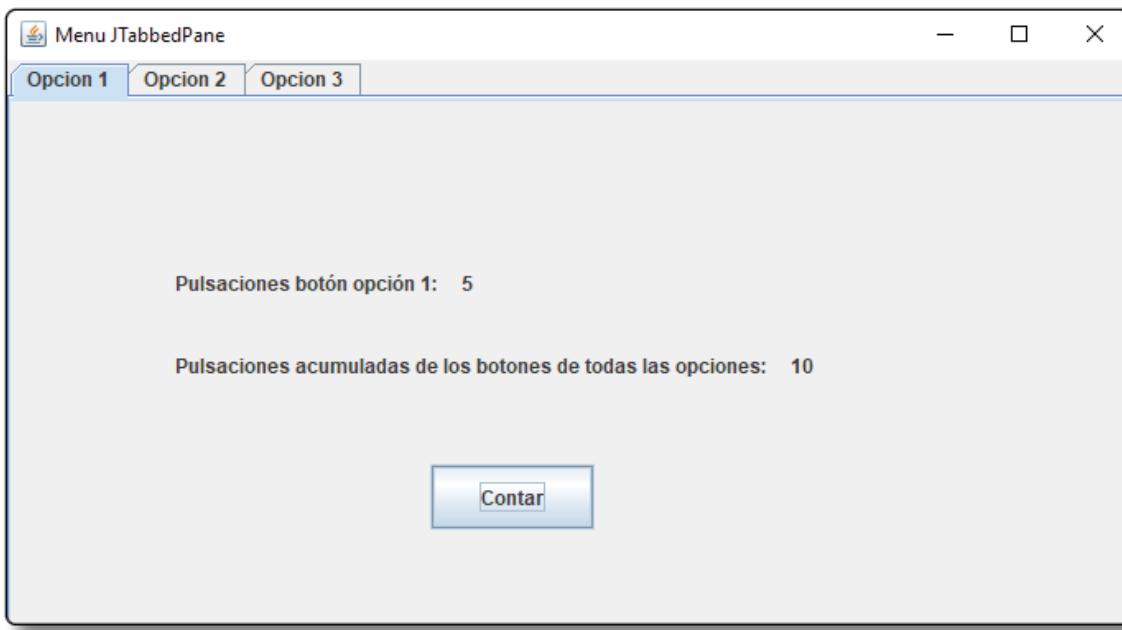
```

package presentacion;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
public class Opcion3 extends JPanel {
private Componentes componentes;
public Opcion3(Componentes componentes) {
this.componentes = componentes;
ubicarComponentes();
}
private void ubicarComponentes() {

```

```
setLayout(null);
// CONFIGURACIÓN JButton
componentes.setjButtonOpcion3(new JButton("Contar"));
componentes.getjButtonOpcion3().setBounds(255, 220,
100, 40);
componentes.getjButtonOpcion3().setActionCommand("botonOpcion3");
componentes.getjButtonOpcion3().addActionListener(componentes.getGestorEventos()); // REGISTRO DE ESCUCHA DE
EVENTO DE BOTON
add(componentes.getjButtonOpcion3());
// CONFIGURACIÓN JLabel visualización contador parcial
componentes.setjLabelParcialOpcion3(new
JLabel("Pulsaciones botón opción 3:
"+componentes.getContadorOpcion3()));
componentes.getjLabelParcialOpcion3().setBounds(100,
100, 400, 20);
add(componentes.getjLabelParcialOpcion3());
// CONFIGURACIÓN JLabel visualización contador global
componentes.setjLabelGlobalOpcion3(new
JLabel("Pulsaciones acumuladas de los botones de todas
las opciones: "+componentes.getContadorGlobal()));
componentes.getjLabelGlobalOpcion3().setBounds(100,
150, 400, 20);
add(componentes.getjLabelGlobalOpcion3());
}
public void iniciarPantalla() {
componentes.setContadorOpcion3(0);
componentes.getjLabelParcialOpcion3().setText("Pulsaciones botón opción 3:
"+componentes.getContadorOpcion3());
componentes.getjLabelGlobalOpcion3().setText("Pulsaciones acumuladas de los botones de todas las opciones:
"+componentes.getContadorGlobal());
}
}
```

La ejecución de la aplicación presenta la siguiente ventana:



Con este ejemplo pretendemos mostrar al lector la implementación de un menú de usuario en un *JFrame* mediante *JTabbedPane*. El menú permite el acceso a tres opciones (**Opcion 1**, **Opcion 2**, **Opcion 3**), accesibles cada una de ellas pulsando en la pestaña correspondiente que queda visible en la parte superior. La funcionalidad que implementan dichas opciones es meramente testimonial, concretamente una de las tratadas en los primeros ejemplos de la parte de Swing de este libro consistente en un contador de pulsaciones de un botón. Cada opción del menú tiene su propio botón, y cada botón tiene su propio contador. Al mismo tiempo también se dispone de un contador global de las pulsaciones globales entre todos los botones. La intención didáctica de la presencia del contador global es plantear la necesidad de tener que compartir referencias a objetos entre los diferentes componentes de una aplicación, en este caso mediante la clase Componentes, estrategia ésta ya descrita y abordada previamente. Así mismo, cada vez que se

accede a una opción, ya accedida con anterioridad, el contador parcial se inicializa. La intención de esta última actuación es plantear la necesidad de implementar el método **iniciarPantalla()**.

Vamos a centrarnos solamente en los aspectos novedosos de esta aplicación, en este ejemplo, son los relacionados con el *JTabbedPane*, vertebrador de las opciones del menú. En primer lugar, comentar que, una vez instanciado, las opciones se añaden:

```
componentes.getjTabbedPane().add("Opcion 1",
componentes.getOpcion1());
```

donde el primer parámetro es la etiqueta que aparece en la parte superior, con que quedará accesible el *JPanel* correspondiente a la subventana de la opción. Dicho *JPanel* constituye el segundo parámetro.

Podemos establecer la selección explícita de una opción desde el código de la aplicación mediante:

```
componentes.getjTabbedPane().setSelectedIndex(0);
```

Queda comentar que en lo que a la gestión de eventos se refiere, el cambio de selección de una opción dispara un evento *ChangeListener*, que debe ser convenientemente tratado por el correspondiente método de evento *stateChanged(ChangeEvent e)*.

Continuar comentando sobre el resto de aspectos, mecanismos, componentes, y evento que están presentes en la aplicación supondría caer en redundancia respecto a cuestiones ya tratadas y abordadas en ejemplos anteriores.

# 21

## APLICACIÓN MENUJMenuBar

Responde exactamente a la misma estructura que la aplicación ejemplo anterior:

- *package presentacion* aglutina las clases:
  - Menu
  - Componentes
  - GestorEventos
  - Opcion1
  - Opcion2
  - Opcion3

### Menu

```
package presentacion;
import java.awt.CardLayout;
import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JPanel;
import javax.swing.JSeparator;
public class Menu extends JFrame {
private Componentes componentes;
```

```
public Menu() {
    setSize(700,500);
    setTitle("Menu JMenuBar");
    componentes = new Componentes();
    componentes.setGestorEventos(new
    GestorEventos(componentes));
    ubicarComponentes();
    inicializar();
    setVisible(true);
}
private void ubicarComponentes() {
    addWindowListener(componentes.getGestorEventos()); //  
REGISTRO DE ESCUCHA DE EVENTOS DE VENTANA
    JMenuBar jMenuBar = new JMenuBar();
    // OPCIONES CON FUNCIONALIDAD, EXISTE UN JPanel  
ASOCIADO A CADA UNA DE ELLAS
    JMenu jMenuOpciones = new JMenu("Opciones con  
funcionalidad");
    JMenuItem jMenuItemOPcion1 = new JMenuItem("Opcion  
1");
    jMenuOpciones.add(jMenuItemOPcion1);
    jMenuItemOPcion1.addActionListener(componentes.getGest  
orEventos());
    JMenuItem jMenuItemOPcion2 = new JMenuItem("Opcion  
2");
    jMenuOpciones.add(jMenuItemOPcion2);
    jMenuItemOPcion2.addActionListener(componentes.getGest  
orEventos());
    JMenuItem jMenuItemOPcion3 = new JMenuItem("Opcion  
3");
    jMenuOpciones.add(jMenuItemOPcion3);
    jMenuItemOPcion3.addActionListener(componentes.getGest  
orEventos());
    jMenuBar.add(jMenuOpciones);
    // OPCIONES SIN FUNCIONALIDAD, NO EXISTE UN JPanel  
ASOCIADO A CADA UNA DE ELLAS
    // Este grupo de opciones se ha implementado para
```

disponer de un segundo grupo de opciones para el menú principal, y también permite ilustrar la utilización del JSeparator.

```
JMenu jMenuOpcionesSinJPanel = new JMenu("Opciones sin funcionalidad");
jMenuOpcionesSinJPanel.add(new JMenuItem("Opcion A"));
jMenuOpcionesSinJPanel.add(new JMenuItem("Opcion B"));
jMenuOpcionesSinJPanel.add(new JSeparator());
jMenuOpcionesSinJPanel.add(new JMenuItem("Opcion C"));
jMenuOpcionesSinJPanel.add(new JMenuItem("Opcion D"));
jMenuOpcionesSinJPanel.add(new JMenuItem("Opcion E"));
jMenuBar.add(jMenuOpcionesSinJPanel);
setJMenuBar(jMenuBar);
componentes.setjPanelPrincipal(new JPanel(new CardLayout()));
componentes.setOpcion1(new Opcion1(componentes));
componentes.getjPanelPrincipal().add("opcion1",
componentes.getOpcion1());
componentes.setOpcion2(new Opcion2(componentes));
componentes.getjPanelPrincipal().add("opcion2",
componentes.getOpcion2());
componentes.setOpcion3(new Opcion3(componentes));
componentes.getjPanelPrincipal().add("opcion3",
componentes.getOpcion3());
add(componentes.getjPanelPrincipal());
componentes.setCardLayout((CardLayout)componentes.getjPanelPrincipal().getLayout());
}
private void inicializar() {
componentes.getOpcion1().iniciarPantalla();
componentes.getCardLayout().show(componentes.getjPanelPrincipal(), "opcion1");
}
public static void main(String[] args) {
EventQueue.invokeLater(new Runnable() {
@Override
public void run() {
```

```
new Menu();
}
});
}
}
```

## Componentes

```
package presentacion;
import java.awt.CardLayout;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
public class Componentes {
private GestorEventos gestorEventos;
private JPanel jPanelPrincipal;
private CardLayout cardLayout;
private Opcion1 opcion1;
private Opcion2 opcion2;
private Opcion3 opcion3;
private JButton jButtonOpcion1;
private JLabel jLabelParcialOpcion1;
private JLabel jLabelGlobalOpcion1;
private JButton jButtonOpcion2;
private JLabel jLabelParcialOpcion2;
private JLabel jLabelGlobalOpcion2;
private JButton jButtonOpcion3;
private JLabel jLabelParcialOpcion3;
private JLabel jLabelGlobalOpcion3;
private int contadorOpcion1 = 0;
private int contadorOpcion2 = 0;
private int contadorOpcion3 = 0;
private int contadorGlobal = 0;
public GestorEventos getGestorEventos() {
return gestorEventos;
}
public void setGestorEventos(GestorEventos
```

```
gestorEventos) {
this.gestorEventos = gestorEventos;
}
public JPanel getjPanelPrincipal() {
return jPanelPrincipal;
}
public void setjPanelPrincipal(JPanel jPanelPrincipal)
{
this(jPanelPrincipal = jPanelPrincipal;
}
public CardLayout getCardLayout() {
return cardLayout;
}
public void setCardLayout(CardLayout cardLayout) {
this.cardLayout = cardLayout;
}
public Opcion1 getOpcion1() {
return opcion1;
}
public void setOpcion1(Opcion1 opcion1) {
this.opcion1 = opcion1;
}
public Opcion2 getOpcion2() {
return opcion2;
}
public void setOpcion2(Opcion2 opcion2) {
this.opcion2 = opcion2;
}
public Opcion3 getOpcion3() {
return opcion3;
}
public void setOpcion3(Opcion3 opcion3) {
this.opcion3 = opcion3;
}
public JButton getjButtonOpcion1() {
return jButtonOpcion1;
}
```

```
public void setjButtonOpcion1(JButton jButtonOpcion1)
{
this(jButtonOpcion1 = jButtonOpcion1;
}
public JLabel getjLabelParcialOpcion1() {
return jLabelParcialOpcion1;
}
public void setjLabelParcialOpcion1(JLabel
jLabelParcialOpcion1) {
this(jLabelParcialOpcion1 = jLabelParcialOpcion1;
}
public JLabel getjLabelGlobalOpcion1() {
return jLabelGlobalOpcion1;
}
public void setjLabelGlobalOpcion1(JLabel
jLabelGlobalOpcion1) {
this(jLabelGlobalOpcion1 = jLabelGlobalOpcion1;
}
public JButton getjButtonOpcion2() {
return jButtonOpcion2;
}
public void setjButtonOpcion2(JButton jButtonOpcion2)
{
this(jButtonOpcion2 = jButtonOpcion2;
}
public JLabel getjLabelParcialOpcion2() {
return jLabelParcialOpcion2;
}
public void setjLabelParcialOpcion2(JLabel
jLabelParcialOpcion2) {
this(jLabelParcialOpcion2 = jLabelParcialOpcion2;
}
public JLabel getjLabelGlobalOpcion2() {
return jLabelGlobalOpcion2;
}
public void setjLabelGlobalOpcion2(JLabel
jLabelGlobalOpcion2) {
```

```
this.jLabelGlobalOpcion2 = jLabelGlobalOpcion2;
}
public JButton getjButtonOpcion3() {
return jButtonOpcion3;
}
public void setjButtonOpcion3(JButton jButtonOpcion3) {
this(jButtonOpcion3 = jButtonOpcion3;
}
public JLabel getjLabelParcialOpcion3() {
return jLabelParcialOpcion3;
}
public void setjLabelParcialOpcion3(JLabel
jLabelParcialOpcion3) {
this.jLabelParcialOpcion3 = jLabelParcialOpcion3;
}
public JLabel getjLabelGlobalOpcion3() {
return jLabelGlobalOpcion3;
}
public void setjLabelGlobalOpcion3(JLabel
jLabelGlobalOpcion3) {
this.jLabelGlobalOpcion3 = jLabelGlobalOpcion3;
}
public int getContadorOpcion1() {
return contadorOpcion1;
}
public void setContadorOpcion1(int contadorOpcion1) {
this.contadorOpcion1 = contadorOpcion1;
}
public int getContadorOpcion2() {
return contadorOpcion2;
}
public void setContadorOpcion2(int contadorOpcion2) {
this.contadorOpcion2 = contadorOpcion2;
}
public int getContadorOpcion3() {
return contadorOpcion3;
```

```
}

public void setContadorOpcion3(int contadorOpcion3) {
this.contadorOpcion3 = contadorOpcion3;
}
public int getContadorGlobal() {
return contadorGlobal;
}
public void setContadorGlobal(int contadorGlobal) {
this.contadorGlobal = contadorGlobal;
}
}
```

## GestorEventos

```
package presentacion;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JMenuItem;;
public class GestorEventos extends WindowAdapter
implements ActionListener {
private Componentes componentes;
public GestorEventos(Componentes componentes) {
this.componentes = componentes;
}
// MÉTODOS DE WindowAdapter
public void windowClosing(WindowEvent e) {
System.exit(0);
}
// MÉTODO DE ActionListener
public void actionPerformed(ActionEvent e) {
String componenteFuente = e.getActionCommand();
if (e.getSource() instanceof JMenuItem)
componenteFuente = ((JMenuItem)
e.getSource()).getActionCommand();
switch(componenteFuente)
```

```
{  
case "Opcion 1" :  
componentes.getOpcion1().iniciarPantalla();  
componentes.getCardLayout().show(componentes.getjPanelPrincipal(), "opcion1");  
break;  
case "Opcion 2" :  
componentes.getOpcion2().iniciarPantalla();  
componentes.getCardLayout().show(componentes.getjPanelPrincipal(), "opcion2");  
break;  
case "Opcion 3" :  
componentes.getOpcion3().iniciarPantalla();  
componentes.getCardLayout().show(componentes.getjPanelPrincipal(), "opcion3");  
break;  
case "botonOpcion1" :  
componentes.setContadorOpcion1(componentes.getContadorOpcion1() + 1);  
componentes.getjLabelParcialOpcion1().setText("Pulsaciones botón opción 1:  
"+componentes.getContadorOpcion1());  
componentes.setContadorGlobal(componentes.getContadorGlobal() + 1);  
componentes.getjLabelGlobalOpcion1().setText("Pulsaciones acumuladas de los botones de todas las opciones:  
"+componentes.getContadorGlobal());  
break;  
case "botonOpcion2" :  
componentes.setContadorOpcion2(componentes.getContadorOpcion2() + 1);  
componentes.getjLabelParcialOpcion2().setText("Pulsaciones botón opción 2:  
"+componentes.getContadorOpcion2());  
componentes.setContadorGlobal(componentes.getContadorGlobal() + 1);  
componentes.getjLabelGlobalOpcion2().setText("Pulsacio
```

```

nes acumuladas de los botones de todas las opciones:
“+componentes.getContadorGlobal());
break;
case “botonOpcion3” :
componentes.setContadorOpcion3(componentes.getContador
Opcion3() + 1);
componentes.getjLabelParcialOpcion3().setText(“Pulsaci
ones botón opción 3:
“+componentes.getContadorOpcion3());
componentes.setContadorGlobal(componentes.getContadorG
lobal() + 1);
componentes.getjLabelGlobalOpcion3().setText(“Pulsacio
nes acumuladas de los botones de todas las opciones:
“+componentes.getContadorGlobal());
break;
}
}
}

```

## Opcion1

```

package presentacion;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
public class Opcion1 extends JPanel {
private Componentes componentes;
public Opcion1(Componentes componentes) {
this.componentes = componentes;
ubicarComponentes();
}
private void ubicarComponentes() {
setLayout(null);
// CONFIGURACIÓN JButton
componentes.set jButtonOpcion1(new JButton(“Contar”));
componentes.get jButtonOpcion1().setBounds(255, 220,
100, 40);

```

```

componentes.get jButtonOpcion1().setActionCommand("botonOpcion1");
componentes.get jButtonOpcion1().addActionListener(componentes.getGestorEventos()); // REGISTRO DE ESCUCHA DE EVENTO DE BOTON
add(componentes.get jButtonOpcion1());
// CONFIGURACIÓN JLabel visualización contador parcial
componentes.set jLabelParcialOpcion1(new JLabel());
componentes.get jLabelParcialOpcion1().setBounds(100, 100, 400, 20);
add(componentes.get jLabelParcialOpcion1());
// CONFIGURACIÓN JLabel visualización contador global
componentes.set jLabelGlobalOpcion1(new JLabel());
componentes.get jLabelGlobalOpcion1().setBounds(100, 150, 400, 20);
add(componentes.get jLabelGlobalOpcion1());
}
public void iniciarPantalla() {
componentes.setContadorOpcion1(0);
componentes.get jLabelParcialOpcion1().setText("Pulsaciones botón opción 1:
"+componentes.getContadorOpcion1());
componentes.get jLabelGlobalOpcion1().setText("Pulsaciones acumuladas de los botones de todas las opciones:
"+componentes.getContadorGlobal());
}
}

```

## Opcion2

```

package presentacion;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
public class Opcion2 extends JPanel {
private Componentes componentes;
public Opcion2(Componentes componentes) {

```

```
this.componentes = componentes;
ubicarComponentes();
}
private void ubicarComponentes() {
setLayout(null);
// CONFIGURACIÓN JButton
componentes.setjButtonOpcion2(new JButton("Contar"));
componentes.getjButtonOpcion2().setBounds(255, 220,
100, 40);
componentes.getjButtonOpcion2().setActionCommand("boto
nOpcion2");
componentes.getjButtonOpcion2().addActionListener(comp
ONENTES.getGestorEventos()); // REGISTRO DE ESCUCHA DE
EVENTO DE BOTON
add(componentes.getjButtonOpcion2());
// CONFIGURACIÓN JLabel visualización contador parcial
componentes.setjLabelParcialOpcion2(new JLabel());
componentes.getjLabelParcialOpcion2().setBounds(100,
100, 400, 20);
add(componentes.getjLabelParcialOpcion2());
// CONFIGURACIÓN JLabel visualización contador global
componentes.setjLabelGlobalOpcion2(new JLabel());
componentes.getjLabelGlobalOpcion2().setBounds(100,
150, 400, 20);
add(componentes.getjLabelGlobalOpcion2());
}
public void iniciarPantalla() {
componentes.setContadorOpcion2(0);
componentes.getjLabelParcialOpcion2().setText("Pulsaci
ones botón opción 2:
"+componentes.getContadorOpcion2());
componentes.getjLabelGlobalOpcion2().setText("Pulsacio
nes acumuladas de los botones de todas las opciones:
"+componentes.getContadorGlobal());
}
}
```

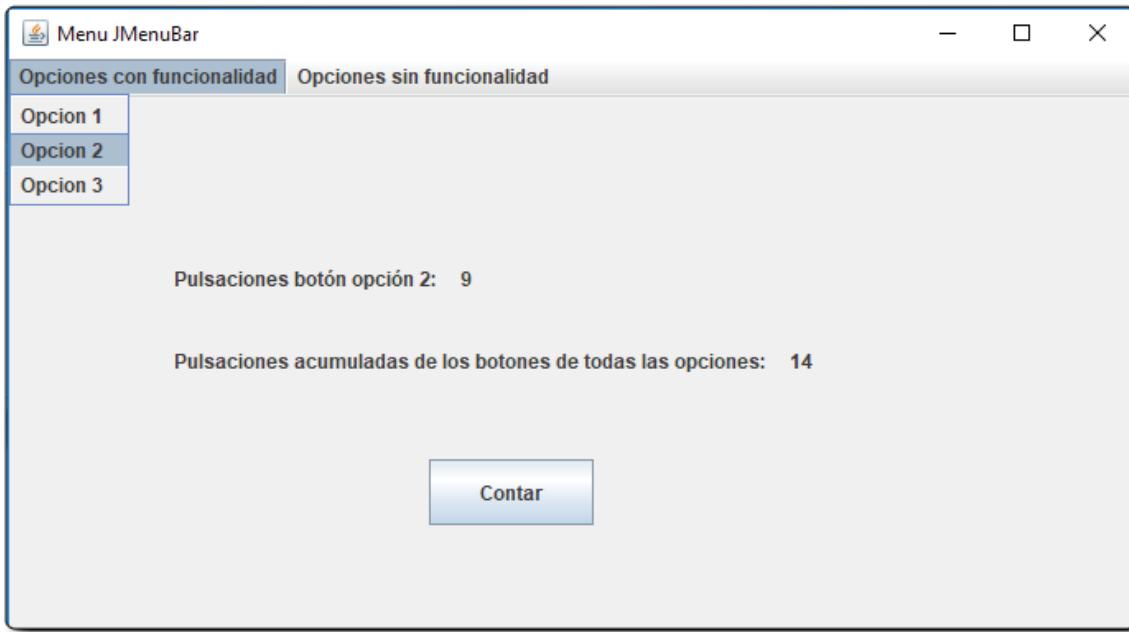
## Opcion3

```
package presentacion;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
public class Opcion3 extends JPanel {
private Componentes componentes;
public Opcion3(Componentes componentes) {
this.componentes = componentes;
ubicarComponentes();
}
private void ubicarComponentes() {
setLayout(null);
// CONFIGURACIÓN JButton
componentes.setjButtonOpcion3(new JButton("Contar"));
componentes.getjButtonOpcion3().setBounds(255, 220,
100, 40);
componentes.getjButtonOpcion3().setActionCommand("botonOpcion3");
componentes.getjButtonOpcion3().addActionListener(componentes.getGestorEventos()); // REGISTRO DE ESCUCHA DE
EVENTO DE BOTON
add(componentes.getjButtonOpcion3());
// CONFIGURACIÓN JLabel visualización contador parcial
componentes.setjLabelParcialOpcion3(new
JLabel("Pulsaciones botón opción 3:
"+componentes.getContadorOpcion3()));
componentes.getjLabelParcialOpcion3().setBounds(100,
100, 400, 20);
add(componentes.getjLabelParcialOpcion3());
// CONFIGURACIÓN JLabel visualización contador global
componentes.setjLabelGlobalOpcion3(new
JLabel("Pulsaciones acumuladas de los botones de todas
las opciones: "+componentes.getContadorGlobal()));
componentes.getjLabelGlobalOpcion3().setBounds(100,
150, 400, 20);
add(componentes.getjLabelGlobalOpcion3());
```

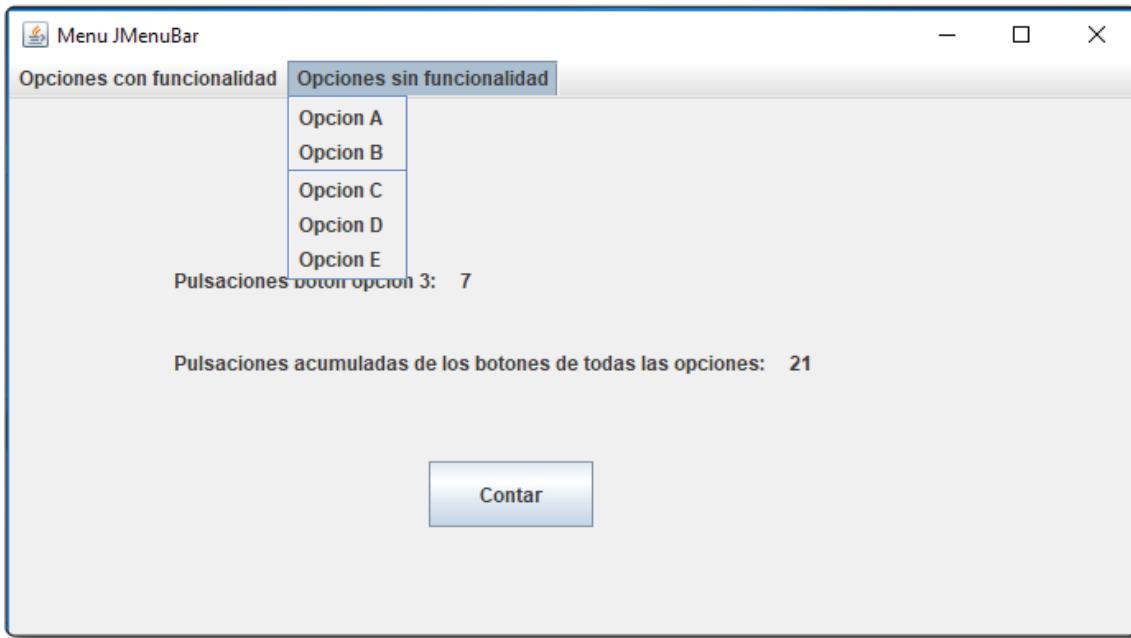
```
}

public void iniciarPantalla() {
    componentes.setContadorOpcion3(0);
    componentes.getjLabelParcialOpcion3().setText("Pulsaciones botón opción 3:
"+componentes.getContadorOpcion3());
    componentes.getjLabelGlobalOpcion3().setText("Pulsaciones acumuladas de los botones de todas las opciones:
"+componentes.getContadorGlobal());
}
}
```

La ejecución de la aplicación presenta la siguiente ventana, en que hemos desplegado la opción de menú Opciones con funcionalidad:



Y a continuación desplegada la opción Opciones sin funcionalidad:



El planteamiento es idéntico al del ejercicio anterior, pero ahora utilizando un menú implementado mediante los componentes:

- JMenuBar
- JMenu
- JMenuItem

combinados con el administrador de diseño *CardLayout*.

El menú plantea dos grupos de opciones:

- Opciones con funcionalidad.- Aglutina las opciones descritas en el ejercicio anterior.
- Opciones sin funcionalidad.- Con la pretensión de diversificar el menú, y al mismo tiempo mostrar la utilización de un *JSeparator* entre la relación de opciones del grupo.

Esta variante de menú es la utilizada en la aplicación final **GestionLibros**. Aunque ya esté presente este tipo de menú en dicha aplicación, se le dedica un ejercicio en exclusiva para facilitar al lector la comprensión de su implementación.

Entremos en detalle en la estructura y mecanismos inherentes al tipo de menú que estamos describiendo. El mayor nivel de integración es modelado por el *JMenuBar*, que es acoplado al *JFrame*:

```
setJMenuBar(jMenuBar);
```

Como grupo aglutinador de opciones en segundo nivel, disponemos del *JMenu*. Los dos presentes en este ejemplo son añadidos al *JMenuBar*:

```
jMenuBar.add(jMenuOpciones);
jMenuBar.add(jMenuOpcionesSin JPanel);
```

Y en tercer nivel nos encontramos *JMenuItem*, representando las opciones individuales, que se integran en el *JMenu* correspondiente:

```
jMenuOpciones.add(jMenuItemOPcion1);
```

La selección por parte del usuario de un *JMenuItem* conllevará la ejecución de una serie de acciones, en los ejemplos de este libro, la presentación de un *JPanel* asociado.

Procedamos a analizar cómo se organizan los *JPanel* que van a conformar el *CardLayout*. En primer lugar, requerimos la presencia de un *JPanel* que actúa como aglutinador del resto:

```
componentes.setjPanelPrincipal(new JPanel(new
CardLayout()));
```

vertebrador del *CardLayout*, que le asignaremos como parámetro del constructor. Será añadido al *JFrame*:

```
add(componentes.getjPanelPrincipal());
```

El resto de paneles implementarán cada una de las vistas que se irán mostrando alternativamente a tenor del *JMenuItem* seleccionado por el usuario. La instanciación y aglutinamiento al principal de estos paneles es implementada:

```
componentes.setOpcion1(new Opcion1(componentes));
componentes.getjPanelPrincipal().add("opcion1",
componentes.getOpcion1());
componentes.setOpcion2(new Opcion2(componentes));
componentes.getjPanelPrincipal().add("opcion2",
componentes.getOpcion2());
componentes.setOpcion3(new Opcion3(componentes));
componentes.getjPanelPrincipal().add("opcion3",
componentes.getOpcion3());
```

Al método *add()* utilizado para agregar los *JPanel* vistas individuales al principal, le transferimos dos parámetros:

- Un *String*, que actuará como identificador de cada uno de ellos.
- La referencia a la instancia de cada *JPanel* individual.

Como ya hemos mencionado con anterioridad solamente dispondrán de funcionalidad, en este ejemplo, la visualización del *JPanel* asociado, los *JMenuItem* integrados en el

### JMenu jMenuOpciones

La selección de un *JMenuItem* por parte del usuario dispara un evento *ActionListener*. Ello trae consigo que todos los eventos disparados en esta aplicación respondan a esta clase: los disparados por *JMenuItem* que acabamos de citar, y los disparados por cada uno de los *JButton* presentes en cada uno de los *JPanel* vista del menú. Así pues, nos vemos abocados a un tratamiento centralizado de todos ellos en el método

*actionPerformed()* de la clase **GestorEventos**. Para dilucidar cuál ha sido el componente concreto origen del evento, recurrimos a sondear el *ActionCommand*, accedido desde el *ActionEvent* transferido al método como parámetro, solución ésta ya tratada en ejemplos anteriores:

```
switch(componenteFuente)
{
case "Opcion 1" :
    ...
break;
case "Opcion 2" :
    ...
break;
case "Opcion 3" :
    ...
break;
case "botonOpcion1" :
    ...
break;
case "botonOpcion2" :
    ...
break;
case "botonOpcion3" :
    ...
break;
}
```

Pero ahora, con un leve matiz, y es que a los *JButton* es necesario establecerles expresamente el *ActionCommand*:

```
componentes.getjButtonOpcion1().setActionCommand("botonOpcion1");
```

y a los *JMenuItem* ya se les asigna en la instanciación, con el *String* que se transfiere como parámetro al constructor:

```
JMenuItem jMenuItemOpcion1 = new JMenuItem("Opcion
```

1”);

Analicemos cómo se implementa:

## SELECCIÓN DE *JMenuItem* → VISUALIZACION DE *JPanel* ASOCIADO

La acción de visualizar cada uno de los *JPanel* que hemos dispuesto como vistas a presentar bajo selección del usuario mediante el menú, se produce al invocar al método `show()` del *CardLayout*:

```
componentes.getCardLayout().show(componentes.getjPanelPrincipal(), "opcion1");
```

al que se le transfieren los parámetros:

- El *JPanel* aglutinador al que se le ha asignado el *CardLayout*.
- El *String* que actúa identificador del *JPanel* vista concreto a mostrar.

# 22

## APLICACIÓN GESTIONEVENTOSFECHA1

Presenta un solo *package*:

- *package gestioneventosfecha* aglutina las clases:
  - GestionEventosFecha1
  - Fecha
  - EventoFecha
  - EventoFechaListener
  - GestorEventos1
  - GestorEventos2

### GestionEventosFecha1

```
package gestioneventosfecha;
public class GestionEventosFecha1 {
public static void main(String[] args) {
GestorEventos1 gestorEventos1 = new GestorEventos1();
GestorEventos2 gestorEventos2 = new GestorEventos2();
System.out.println("-----
-----");
Fecha fecha1 = new Fecha("12-24-1968"); // dispara el
evento
fecha1.addEventoFechaListener(gestorEventos1);
```

```
fecha1.addEventoFechaListener(gestorEventos2);
fecha1.filtrarFecha();
Fecha fecha2 = new Fecha("12-04-1968"); // no dispara
el evento
fecha2.addEventoFechaListener(gestorEventos1);
fecha1.addEventoFechaListener(gestorEventos2);
fecha2.filtrarFecha();
System.out.println("-----");
-----");
}
}
```

## Fecha

```
package gestioneventosfecha;
import java.util.ArrayList;
import java.util.List;
public class Fecha {
private String fecha;
private List<EventoFechaListener> listaListeners = new
ArrayList();
public Fecha(String fecha) {
this.fecha = fecha;
}
public String getFecha() {
return fecha;
}
public void setFecha(String fecha) {
this.fecha = fecha;
}
public void filtrarFecha() {
int codigoError = 0;
int dia = 0;
int mes = 0;
int año = 0;
if (fecha.length() == 10 && fecha.charAt(2) == '-' &&
fecha.charAt(5) == '-')
{
StringTokenizer st = new StringTokenizer(fecha, "-");
st.nextToken();
dia = Integer.parseInt(st.nextToken());
mes = Integer.parseInt(st.nextToken());
año = Integer.parseInt(st.nextToken());
}
else
codigoError = 1;
}
}
```

```
{  
try {  
dia = Integer.parseInt(fecha.substring(0, 2));  
mes = Integer.parseInt(fecha.substring(3, 5));  
año = Integer.parseInt(fecha.substring(6,  
fecha.length()));  
} catch (NumberFormatException excepcion)  
{ códigoError = códigoError | 1; } // dígito no  
numérico en fecha  
}  
else  
{  
códigoError = códigoError | 1; // formato fecha  
erróneo  
}  
if (códigoError == 0)  
{  
if (Integer.toString(año).length() != 4) // año  
erróneo  
códigoError = códigoError | 2;  
if (mes < 1 || mes > 12) // mes erróneo  
códigoError = códigoError | 4;  
if (dia < 1 || dia > getDiaMaximoMes(mes, año)) // dia  
erróneo  
códigoError = códigoError | 8;  
}  
System.out.println("código error fecha "+códigoError);  
if (códigoError > 0)  
fireEventoFecha(códigoError);  
}  
private int getDiaMaximoMes(int mes, int año) {  
int diaMaximoMes = 31;  
switch(mes)  
{ case 1:  
case 3:  
case 5:  
case 7:
```

```
case 8:  
case 10:  
case 12: diaMaximoMes = 31;  
break;  
case 4:  
case 6:  
case 9:  
case 11: diaMaximoMes = 30;  
break;  
case 2: if (((año % 4 == 0) && !(año % 100 == 0)) ||  
(año % 400 == 0))  
diaMaximoMes = 29;  
else  
diaMaximoMes = 28;  
break;  
}  
return diaMaximoMes;  
}  
public void addEventoFechaListener(EventoFechaListener  
eventoFechaListener) {  
listaListeners.add(eventoFechaListener);  
}  
public void  
removeEventoFechaListener(EventoFechaListener  
eventoFechaListener) {  
listaListeners.remove(eventoFechaListener);  
}  
private void fireEventoFecha(int codigoError) {  
for (int i=0; i<listaListeners.size(); i++) {  
listaListeners.get(i).actionPerformed(new  
EventoFecha(this, 1, "fechaIncorrecta", codigoError));  
}  
}  
}
```

## EventoFecha

```
package gestioneventosfecha;
import java.awt.event.ActionEvent;
public class EventoFecha extends ActionEvent {
private int codigoError;
public EventoFecha(Object source, int id, String
command, int codigoError) {
super(source, id, command);
this.codigoError = codigoError;
}
public int getCodigoError() {
return codigoError;
}
}
```

## EventoFechaListener

```
package gestioneventosfecha;
import java.awt.event.ActionListener;
public interface EventoFechaListener extends
ActionListener {
```

## GestorEventos1

```
package gestioneventosfecha;
import java.awt.event.ActionEvent;
public class GestorEventos1 implements
EventoFechaListener{
@Override
public void actionPerformed(ActionEvent e) {
System.out.println("-----
-----");
System.out.println("La instancia de la clase
"+this.getClass().getName()+" ha interceptado ");
System.out.println("el evento con actionCommand
<<"+e.getActionCommand()+">>");
System.out.println("Disparado desde la fuente
<<"+e.getSource().getClass().getName()+">>");
```

```

System.out.println("El filtro ha generado un código de
error "+((EventoFecha)e).getCodigoError());
System.out.println("Para un valor de fecha "+
((Fecha)e.getSource()).getFecha());
System.out.println("-----");
}
}

```

## GestorEventos2

```

package gestioneventosfecha;
import java.awt.event.ActionEvent;
public class GestorEventos2 implements
EventoFechaListener{
@Override
public void actionPerformed(ActionEvent e) {
System.out.println("-----");
System.out.println("La instancia de la clase
"+this.getClass().getName()+" ha interceptado ");
System.out.println("el evento con actionCommand
<<"+e.getActionCommand()+">>");
System.out.println("Disparado desde la fuente
<<"+e.getSource().getClass().getName()+">>");
System.out.println("El filtro ha generado un código de
error "+((EventoFecha)e).getCodigoError());
System.out.println("Para un valor de fecha "+
((Fecha)e.getSource()).getFecha());
System.out.println("-----");
}
}

```

Este ejercicio ejemplo, al igual que el siguiente, totalmente análogo a éste salvo leves diferencias que matizaremos en su momento, no están enfocados a mostrar una interfaz gráfica, sino

que la interacción con el usuario es mediante una interfaz texto que permite interacción con el usuario en consola de salida.

El objetivo didáctico a alcanzar con la aportación de este ejercicio ejemplo, y del siguiente, es mostrar al lector los mecanismos a utilizar en la implementación del lanzamiento explícito de eventos por parte de la aplicación, sin intervención del usuario. En los dos casos que estamos tratando, el evento se dispara al detectar una situación de error al aplicar un filtro de fecha:

```
if (codigoError > 0)
    fireEventoFecha(codigoError);
```

En la aplicación final **GestionLibros**, también se implementa un filtro de fecha, pero en esa aplicación, la detección de una situación de error asociada a una fecha conlleva el lanzamiento de una excepción:

```
if (codigoError > 0)
    throw new GenericaExcepcion(codigoError);
```

La utilización de un filtro de fecha como mecanismo de detección de una determinada situación susceptible de “lanzar una señal”, en este caso un evento, es en este ejemplo (y en el siguiente) meramente testimonial, concretamente, disponer de un escenario que permita mostrar al lector, de la forma más sencilla y clara posible, el mecanismo de lanzamiento explícito de eventos por parte de la aplicación. Para ello, el autor ha recreado todos los componentes y mecanismos intervenientes en un ámbito y contexto lo más reducido posible.

El mecanismo exclusivo a tratar en estos dos ejemplos, volverá a estar presente en la aplicación **GestionLibros**, pero en ese momento, con la intención de que se produzca una invocación automática de determinadas opciones de menú. En su momento se tratará con profundidad.

Procedamos ahora a analizar con detalle los componentes y mecanismos aplicados en estos ejemplos, que intervienen en el lanzamiento explícito y automático de eventos por parte de la aplicación.

## DEFINICIÓN DEL EVENTO

Partimos por definir una clase que nos permita modelar los eventos a gestionar. En este primer ejemplo:

```
public class EventoFecha extends ActionEvent
```

Necesariamente esta clase, aunque definida por el programador, debe contar con todas las funcionalidades propias de un evento. Optamos, pues, por heredar de una clase de evento predefinida, y añadirle los elementos que creamos oportunos. En este caso, el atributo:

```
private int codigoError
```

cuyo cometido no necesita aclaración dado que su nombre es suficientemente significativo. Y su correspondiente método “getter” **getCodigoError()**. El constructor de la clase se limita a recibir y encapsular el valor para dicho atributo, y otros parámetros que nos limitamos a transferir al constructor de la superclase en su invocación:

```
super(source, id, command)
```

## DEFINICIÓN DEL LISTENER

Nos hemos limitado a generar una réplica de *ActionListener*, sin ningún tipo de aportación adicional:

```
public interface EventoFechaListener extends  
ActionListener
```

Como ya habrá deducido el lector, en este primer ejemplo nos estamos limitando a personalizar eventos y escuchas *ActionListener*. Nos proporcionaría mucha más versatilidad y amplitud personalizar eventos y escuchas no tan específicos. El ejemplo siguiente nos proporcionará esa opción de mayor alcance y amplitud. Pero centrémonos, por el momento en esta opción, que nos puede dar la impresión de actuar contraproducentemente por su condición de restrictiva, es decir, por limitarse a ser una “variante” de *ActionListener*. Demostraremos que tiene su utilidad cuando pretendemos dar un tratamiento centralizado a la gestión de la escucha del evento, homogeneizándolo con la gestión de escucha de otros eventos lanzados por componentes de interfaz gráfica, tales como *JButton*, *JComboBox*, y *JMenuItem*. Tendremos oportunidad de ver todo ello reflejado en los ejemplos **PrediccionMeteorologicaJTree** y **GestionLibros**.

## DEFINICIÓN DE CLASES ESCUCHA

Podremos definir tantas clases escucha como se crea oportuno. En el ejercicio ejemplo hemos definido dos: *GestorEventos1* y *GestorEventos2*. Obviamente para poder erigirse en clases escucha del evento que estamos tratando, y ser parte integrante de los mecanismos que estamos describiendo, deben implementar la interface *EventoFechaListener*:

```
public class GestorEventos1 implements  
EventoFechaListener{  
public void actionPerformed(ActionEvent e) {  
    . . .  
}  
}
```

La implementación del método *actionPerformed()* deberá ser acorde con acciones a realizar ante el disparo del evento. En pro de la simplicidad didáctica de estos dos ejemplos, las acciones son meramente testimoniales.

## DEFINICIÓN DEL COMPONENTE LANZADOR DEL EVENTO

Le hemos dado dicho cometido en estos dos ejemplos análogos a la clase Fecha, que representa una clase encapsuladora, dotada con un constructor a tal efecto, y sus métodos “setters” y “getters”. Adicionalmente, le hemos aportando el correspondiente filtro, implementado por el método

```
public void filtrarFecha()
```

en cuya ejecución se lanza explícita y automáticamente el evento, caso de detectarse que la fecha encapsulada presenta una condición de error:

```
if (codigoError > 0)
fireEventoFecha(codigoError);
```

La ejecución de las actuaciones implementadas en el método *fireEventoFecha()* son lo que realmente suponen el lanzamiento explícito del evento. Para analizar dichas actuaciones, es necesario remitirnos previamente a la declaración en esta clase del atributo:

```
private List<EventoFechaListener> listaListeners = new
ArrayList();
```

que aglutinará todas las instancias de clases susceptibles de erigirse en escuchas del evento a tratar, y que serán aglutinadas en el *ArrayList* al ser recibidas las referencias a dichas instancias por el método:

```
public void addEventoFechaListener(EventoFechaListener  
eventoFechaListener) {  
listaListeners.add(eventoFechaListener);  
}
```

Volviendo al método

```
private void fireEventoFecha(int codigoError) {  
for (int i=0; i<listaListeners.size(); i++) {  
listaListeners.get(i).actionPerformed(new  
EventoFecha(this, 1, "fechaIncorrecta", codigoError));  
}  
}
```

observaremos que la actuaciones consisten en invocar al método escucha correspondiente, en este caso, *actionPerformed()* de todas las instancias previamente registradas, generando una instancia del evento para cada una de dichas invocaciones, cuya referencia es transferida en la citada invocación.

También podremos observar que la clase objeto de tratamiento en este apartado, también dispone del método antagónico al que acabamos de describir:

```
public void  
removeEventoFechaListener(EventoFechaListener  
eventoFechaListener) {  
listaListeners.remove(eventoFechaListener);  
}
```

## REGISTRO CON CLASE ESCUCHA

Desde el método *main()* de la clase **GestionEventosFecha1**:

```
fecha1.addEventoFechaListener(gestorEventos1);  
fecha1.addEventoFechaListener(gestorEventos2);
```

Dado que hemos definido dos clases escucha del evento diferentes, hemos registrado el componente lanzador del evento,

en este caso concreto la instancia de la clase *Fecha*, con cada una de las dos clases escucha previamente instanciadas, al transferir la referencia de cada instancia al método *addEventoFechaListener()* tratado con anterioridad.

## INVOCACIÓN A MÉTODO SUSCEPTIBLE DE LANZAR EL EVENTO

Desde el método *main()* de la clase **GestionEventosFecha1**:

```
fecha1.filtrarFecha();
```

Después de haber procedido a la descripción de los mecanismos que intervienen en la definición, registro, lanzamiento y tratamiento de eventos explícitamente desde la aplicación, estamos en condiciones de analizar la salida en consola producida por la ejecución de la aplicación:

**codigo error fecha 4**

La instancia de la clase  
gestioneventosfecha.GestorEventos1 ha interceptado  
el evento con actionPerformed <<fechaIncorrecta>>  
Disparado desde la fuente <<gestioneventosfecha.Fecha>>  
El filtro ha generado un código de error 4  
Para un valor de fecha 12-24-1968

---

---

---

---

La instancia de la clase  
gestioneventosfecha.GestorEventos2 ha interceptado  
el evento con actionPerformed <<fechaIncorrecta>>  
Disparado desde la fuente <<gestioneventosfecha.Fecha>>  
El filtro ha generado un código de error 4  
Para un valor de fecha 12-24-1968

**codigo error fecha 0**

Podremos comprobar, que en el método *main()* se instancian dos fechas, la primera, referenciada desde *fecha1* conlleva una situación de error, y la segunda es correcta. Dado que la instancia referenciada desde *fecha1*, se ha registrado con las dos clases escucha del evento, se ejecutará el método *actionPerformed()* de ambas clases. También podemos comprobar por la salida en consola que tenemos disponible el mecanismo asociado a *ActionCommand* utilizado y tratado en ejemplos anteriores.

# 23

## APLICACIÓN GESTIONEVENTOSFECHA2

Presenta exactamente la misma estructura que el ejemplo anterior:

- *package gestioneventosfecha* aglutina las clases:
  - GestionEventosFecha2
  - Fecha
  - EventoFecha
  - EventoFechaListener
  - GestorEventos1
  - GestorEventos2

### GestionEventosFecha2

```
package gestioneventosfecha;
public class GestionEventosFecha2 {
public static void main(String[] args) {
GestorEventos1 gestorEventos1 = new GestorEventos1();
GestorEventos2 gestorEventos2 = new GestorEventos2();
System.out.println("-----
-----");
Fecha fecha1 = new Fecha("12-24-1968"); // dispara el
evento
fecha1.addEventoFechaListener(gestorEventos1);
```

```

fecha1.addEventoFechaListener(gestorEventos2);
fecha1.filtrarFecha();
Fecha fecha2 = new Fecha("12-04-1968"); // no dispara
el evento
fecha2.addEventoFechaListener(gestorEventos1);
fecha1.addEventoFechaListener(gestorEventos2);
fecha2.filtrarFecha();
System.out.println("-----");
-----");
}
}

```

## Fecha

```

package gestioneventosfecha;
import javax.swing.event.EventListenerList;
public class Fecha {
private String fecha;
EventListenerList eventListenerList = new
EventListenerList();
public Fecha(String fecha) {
this.fecha = fecha;
}
public String getFecha() {
return fecha;
}
public void setFecha(String fecha) {
this.fecha = fecha;
}
public void filtrarFecha() {
int codigoError = 0;
int dia = 0;
int mes = 0;
int año = 0;
if (fecha.length() == 10 && fecha.charAt(2) == '-' &&
fecha.charAt(5) == '-')
{

```

```
try {
    dia = Integer.parseInt(fecha.substring(0, 2));
    mes = Integer.parseInt(fecha.substring(3, 5));
    año = Integer.parseInt(fecha.substring(6,
        fecha.length()));
} catch (NumberFormatException excepcion)
{ códigoError = códigoError | 1; } // dígito no
número en fecha
}
else
{
    códigoError = códigoError | 1; // formato fecha
    erróneo
}
if (códigoError == 0)
{
    if (Integer.toString(año).length() != 4) // año
    erróneo
    códigoError = códigoError | 2;
    if (mes < 1 || mes > 12) // mes erróneo
    códigoError = códigoError | 4;
    if (dia < 1 || dia > getDiaMaximoMes(mes, año)) // dia
    erróneo
    códigoError = códigoError | 8;
}
System.out.println("código error fecha "+códigoError);
if (códigoError > 0)
fireEventoFecha(códigoError);
}
private int getDiaMaximoMes(int mes, int año) {
int diaMaximoMes = 31;
switch(mes)
{ case 1:
case 3:
case 5:
case 7:
case 8:
```

```
case 10:  
case 12: diaMaximoMes = 31;  
break;  
case 4:  
case 6:  
case 9:  
case 11: diaMaximoMes = 30;  
break;  
case 2: if (((año % 4 == 0) && !(año % 100 == 0)) ||  
(año % 400 == 0))  
diaMaximoMes = 29;  
else  
diaMaximoMes = 28;  
break;  
}  
return diaMaximoMes;  
}  
public void addEventoFechaListener(EventoFechaListener  
eventoFechaListener) {  
eventListenerList.add(EventoFechaListener.class,  
eventoFechaListener);  
}  
public void  
removeEventoFechaListener(EventoFechaListener  
eventoFechaListener) {  
eventListenerList.remove(EventoFechaListener.class,  
eventoFechaListener);  
}  
private void fireEventoFecha(int codigoError) {  
Object[] eventoFechaListeners =  
eventListenerList.getListenerList();  
// Solamente registran referencia a objeto Listener  
los componentes con índice impar  
for (int i=1; i<eventoFechaListeners.length; i+=2) {  
((EventoFechaListener)eventoFechaListeners[i]).manejar  
EventoFecha(new EventoFecha(this, "fechaIncorrecta",  
codigoError));
```

```
}
```

```
}
```

```
}
```

## EventoFecha

```
package gestioneventosfecha;
import java.util.EventObject;
public class EventoFecha extends EventObject {
private String actionCommand;
private int codigoError;
public EventoFecha(Object source, String
actionCommand, int codigoError) {
super(source);
this.codigoError = codigoError;
this.actionCommand = actionCommand;
}
public String getActionCommand() {
return actionCommand;
}
public int getCodigoError() {
return codigoError;
}
}
```

## EventoFechaListener

```
package gestioneventosfecha;
import java.util.EventListener;
public interface EventoFechaListener extends
EventListener {
void manejarEventoFecha(EventoFecha eventoFecha);
}
```

## GestorEventos1

```
package gestioneventosfecha;
public class GestorEventos1 implements
```

```

EventoFechaListener{
@Override
public void manejarEventoFecha(EventoFecha e) {
System.out.println("-----");
System.out.println("La instancia de la clase
"+this.getClass().getName()+" ha interceptado ");
System.out.println("el evento con actionCommand
<<" + e.getActionCommand() + ">>");
System.out.println("Disparado desde la fuente
<<" + e.getSource().getClass().getName() + ">>");
System.out.println("El filtro ha generado un código de
error "+e.getCodigoError());
System.out.println("Para un valor de fecha "+
((Fecha)e.getSource()).getFecha());
System.out.println("-----");
}
}

```

## GestorEventos2

```

package gestioneventosfecha;
public class GestorEventos2 implements
EventoFechaListener{
@Override
public void manejarEventoFecha(EventoFecha e) {
System.out.println("-----");
System.out.println("La instancia de la clase
"+this.getClass().getName()+" ha interceptado ");
System.out.println("el evento con actionCommand
<<" + e.getActionCommand() + ">>");
System.out.println("Disparado desde la fuente
<<" + e.getSource().getClass().getName() + ">>");
System.out.println("El filtro ha generado un código de
error "+e.getCodigoError());
}
}

```

```
System.out.println("Para un valor de fecha "+  
((Fecha)e.getSource()).getFecha());  
System.out.println("-----");  
-----);  
}  
}
```

Observemos que la salida en consola correspondiente a la ejecución de la aplicación es exactamente la misma que la del ejercicio anterior:

### **codigo error fecha 4**

**La instancia de la clase  
gestioneventosfecha.GestorEventos1 ha interceptado  
el evento con actionPerformed <<fechaIncorrecta>>  
Disparado desde la fuente <<gestioneventosfecha.Fecha>>  
El filtro ha generado un código de error 4  
Para un valor de fecha 12-24-1968**

-----  
-----  
-----  
-----

**La instancia de la clase  
gestioneventosfecha.GestorEventos2 ha interceptado  
el evento con actionPerformed <<fechaIncorrecta>>  
Disparado desde la fuente <<gestioneventosfecha.Fecha>>  
El filtro ha generado un código de error 4  
Para un valor de fecha 12-24-1968**

### **codigo error fecha 0**

Existe una total analogía en los mecanismos contemplados en el ejercicio anterior, salvo algunos matices y diferencias que procedemos a comentar a continuación. Estableceremos dicha comparativa ciñéndonos a los puntos de discrepancia según

aparecen en el esquema de mecanismos utilizado para el ejercicio anterior.

## DEFINICIÓN DEL EVENTO

En este caso utilizamos:

```
public class EventoFecha extends EventObject
```

con lo que estamos personalizando, por herencia directa desde *EventObject*, un evento mucho más genérico que el del caso anterior, en que se trataba de un evento que heredaba de *ActionEvent*. Hemos de tener en cuenta que todos los eventos predefinidos heredan directa o indirectamente de *EventObject*.

En el caso anterior la gestión de *ActionCommand* estaba predefinida, no había más que transferirle el correspondiente valor al constructor de la superclase *ActionEvent*:

```
super(source, id, command);
```

y no existía necesidad de definir el método “getter” correspondiente a este atributo. Podemos decir que el *ActionCommand* “viene de serie” (permítanme el símil automovilístico). En cambio, ahora debemos definir el atributo “incorporando el accesorio” (permítanme seguir con el símil automovilístico) como atributo del evento

```
private String actionPerformed;
```

realizando la correspondiente asignación en el constructor

```
this.actionCommand = actionPerformed;
```

y definiendo necesariamente el método “getter” si se pretende acceder al atributo

```
public String getActionCommand() {  
    return actionPerformed;  
}
```

## DEFINICIÓN DEL LISTENER

Con total analogía a como hemos actuado con el evento, lo hacemos con el Listener al heredar directamente de *EventListener*, tratándose esta de una interface de la que heredan todos los Listeners. Circunstancia ésta que contribuye a movernos en un ámbito lo más genérico posible. Otra diferencia con el caso anterior es que no teníamos necesidad de establecer en el Listener que hemos definido el compromiso para las clases escucha de implementar el método *actionPerformed()*, ya que la definición de dicho compromiso es heredada de *ActionListener*. Ahora sí que debemos establecer dicha obligación para las clases escucha, dado que *EventListener* no establece ningún método a ser definido obligatoriamente que podamos heredar. Concretamente dicha obligación la establecemos con la definición en nuestra interface de **manejarEventoFecha()**.

```
public interface EventoFechaListener extends  
EventListener {  
void manejarEventoFecha(EventoFecha eventoFecha);  
}
```

## DEFINICIÓN DE CLASE ESCUCHA

Seguimos definiendo dos: **GestorEventos1** y **GestorEventos2**. Al igual que en el caso anterior, se erigen en clases escucha del evento que estamos tratando al implementar la interface *EventoFechaListener*, circunstancia que implica la obligación de implementar el nuevo método **manejarEventoFecha(EventoFecha e)** cuyo código se ejecutará ante el disparo del evento.

```
public class GestorEventos1 implements  
EventoFechaListener{  
public void manejarEventoFecha(EventoFecha e) {
```

```
...  
}  
}
```

## DEFINICIÓN DEL COMPONENTE LANZADOR DEL EVENTO

Es en este punto donde se acentúan las diferencias respecto al caso anterior. Destacamos en primer lugar que se aglutinarán las clases escucha que se vayan registrando en un *EventListenerList*, que realiza la misma función que el *ArrayList* del caso anterior, pero ahora se trata de una clase específica, expresamente diseñada para aglutinar clases escucha.

La tarea de añadir o eliminar clases escucha al elemento que actúa como lista, seguimos haciéndolo mediante métodos *add()* y *remove()*, pero ahora se tratan de métodos específicos de la clase *EventListenerList*, cuya invocación debe transferir dos parámetros en ambos casos:

```
eventListenerList.add(EventoFechaListener.class,  
eventoFechaListener);  
eventListenerList.remove(EventoFechaListener.class,  
eventoFechaListener);
```

Siendo el primero una instancia de la clase *Class* correspondiente al Listener, y el segundo se corresponde con el único que utilizábamos en el caso anterior. En el objeto aglutinador específico que actúa como lista, se van añadiendo, para cada registro que se produzca, los dos parámetros alternativamente, de tal forma que, en las posiciones con valor de índice par, se aglutan las instancias de la clase *Class*, y en las posiciones con valor de índice impar, lo hacen las referencias a los objetos a registrar como clase de escucha.

Análogamente a como hemos actuado en el ejercicio anterior, hemos de recorrer el elemento aglutinador de instancias de clases escucha registradas para invocar al correspondiente método en todas ellas, que en el caso anterior era *actionPerformed()*, y ahora se trata de **manejarEventoFecha()**. El tratamiento iterativo que hemos de aplicar será el propio de un array, que obtendremos

```
Object[] eventoFechaListeners =  
eventListenerList.getListenerList();
```

La peculiar distribución de los elementos aglutinados descrita previamente, nos fuerza a que solamente debamos acceder a los componentes con valor de índice impar:

```
for (int i=1; i<eventoFechaListeners.length; i+=2) {  
((EventoFechaListener)eventoFechaListeners[i]).manejar  
EventoFecha(new EventoFecha(this, "fechaIncorrecta",  
codigoError));
```

Podemos observar que, como consecuencia de haber obtenido un array de *Object*, nos vemos forzados a aplicar el correspondiente *cast*.

# 24

## APLICACIÓN PREDICCIONMETEOROLOGICA JTREE

La estructura que presenta esta aplicación corresponde a la establecida por el modelo de desarrollo de software arquitectura a tres capas:

- *package presentacion* aglutina las clases:
  - InterfazPrediccion
  - Controller
  - InvocacionCargaProvincias
  - EventoCargaProvincias
- *package negocio* aglutina las clases:
  - PrediccionMeteorologica
  - InformacionGeografica
- *package datos* aglutina las clases:
  - PrediccionMeteorologicaDATOS
  - ProvinciasDATOS
  - MunicipiosDATOS
- *package encapsuladores* aglutina las clases:

- MeteorologiaDiaria
- MeteorologiaMunicipio

## InterfazPrediccion

```

package presentacion;
import encapsuladores.MeteorologiaMunicipio;
import java.awt.Color;
import java.awt.EventQueue;
import java.util.List;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JScrollPane;
import javax.swing.JTree;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeModel;
import negocio.InformacionGeografica;
import negocio.PrediccionMeteorologica;
public class InterfazPrediccion extends JFrame {
private Controller controller;
private JTree jTree;
private DefaultTreeModel modelo;
private DefaultMutableTreeNode nodoRaiz,
nodoProductor, nodoPrediccionSemanal,
nodoPrediccionDiaria, nodoProbabilidadPrecipitacion,
nodoPeriodoProbabilidadPrecipitacion, nodoEstadoCielo,
nodoPeriodoEstadoCielo, nodoTemperatura,
nodoHumedadRelativa;
private JLabel jLabelSeleccionProvincia;
private JComboBox jComboBoxSeleccionProvincia;
private JLabel jLabelSeleccionMunicipio;
private JComboBox jComboBoxSeleccionMunicipio;
private JLabel jLabelAEMET;
public InterfazPrediccion() {
controller = new Controller(this);
setSize(1200,900);

```

```
setTitle("Predicción meteorológica");
ubicarComponentes();
setVisible(true);
}
private void ubicarComponentes() {
setLayout(null);
addWindowListener(controller); // REGISTRO DE ESCUCHA
DE EVENTOS DE VENTANA
nodoRaiz = new DefaultMutableTreeNode(" ");
jTree = new JTree( nodoRaiz );
modelo = (DefaultTreeModel)jTree.getModel();
JScrollPane jScrollPane = new JScrollPane(jTree);
jScrollPane.setBounds(20, 20, 1140, 450);
add(jScrollPane);
jLabelSeleccionProvincia = new JLabel("PROVINCIA :");
jLabelSeleccionProvincia.setBounds(80, 500, 120, 20);
add(jLabelSeleccionProvincia);
jComboBoxSeleccionProvincia = new JComboBox();
jComboBoxSeleccionProvincia.setBounds(210, 500, 370,
20);
jComboBoxSeleccionProvincia.setBackground(Color.white)
;
jComboBoxSeleccionProvincia.setActionCommand("comboProvincias");
add(jComboBoxSeleccionProvincia);
jLabelSeleccionMunicipio = new JLabel("MUNICIPIO :");
jLabelSeleccionMunicipio.setBounds(80, 670, 120, 20);
add(jLabelSeleccionMunicipio);
jComboBoxSeleccionMunicipio = new JComboBox();
jComboBoxSeleccionMunicipio.setBounds(210, 670, 370,
20);
jComboBoxSeleccionMunicipio.setBackground(Color.white)
;
jComboBoxSeleccionMunicipio.setActionCommand("comboMunicipios");
add(jComboBoxSeleccionMunicipio);
jLabelAEMET = new JLabel("© AEMET");
```

```
jLabelAEMET.setBounds(900, 500, 100, 20);
add(jLabelAEMET);
// LANZAMIENTO EXPLICITO EVENTO CARGA JComboBox
jComboBoxSeleccionProvincia
InvocacionCargaProvincias invocacionCargaProvincias =
new InvocacionCargaProvincias();
invocacionCargaProvincias.addEventoCargaProvinciasList
ener(controller);
invocacionCargaProvincias.fireEventoCargaProvincias("c
argarProvincias");
}
public void responderAController(String actionCommand)
throws Exception {
switch(actionCommand)
{
case "cargarProvincias" :
// La eliminación de registros en clases escucha
permite que no se dispare el correspondiente
// evento para cada uno de los items que se añaden al
JComboBox.
// Esta actuación no es necesaria para el JComboBox
correspondiente a la selección de provincia
// porque solamente se carga en una sola ocasión. Se
implementa testimonialmente.
if
(jComboBoxSeleccionProvincia.getActionListeners().leng
th > 0)
jComboBoxSeleccionProvincia.removeActionListener(controller);
List<String> listaProvincias = new
InformacionGeografica().leerProvincias();
for (int i=0; i<listaProvincias.size(); i++)
{
jComboBoxSeleccionProvincia.addItem(listaProvincias.ge
t(i));
}
jComboBoxSeleccionProvincia.addActionListener(controll
```

```
er);
break;
case "comboProvincias":
if
(jComboBoxSeleccionMunicipio.getActionListeners().length > 0)
jComboBoxSeleccionMunicipio.removeActionListener(controller);
jComboBoxSeleccionMunicipio.removeAllItems();
List<String> listaMunicipios = new
InformacionGeografica().leerMunicipios(jComboBoxSeleccionProvincia.getSelectedItem().toString().substring(0,
2));
for (int i=0; i<listaMunicipios.size(); i++)
{
jComboBoxSeleccionMunicipio.addItem(listaMunicipios.get(i));
}
jComboBoxSeleccionMunicipio.addActionListener(controller);
break;
case "comboMunicipios":
cargarArbol(new
PrediccionMeteorologica().obtenerPrediccionMeteorologica(jComboBoxSeleccionMunicipio.getSelectedItem().toString().substring(0, 5)));
break;
}
}
private void cargarArbol(MeteorologiaMunicipio
meteorologiaMunicipio) {
nodoRaiz.removeAllChildren();
nodoRaiz.setUserObject("Predicción municipio :
"+meteorologiaMunicipio.getNombreMunicipio());
modelo.reload();
nodoProductor = new DefaultMutableTreeNode("Productor
: "+meteorologiaMunicipio.getProductor());
```

```
nodoRaiz.add(nodoProductor);
nodoPrediccionSemanal = new
DefaultMutableTreeNode("Predicción semanal");
nodoRaiz.add(nodoPrediccionSemanal);
for (int i=0;
i<meteorologiaMunicipio.getMeteorologiaDiaria().length
; i++)
{
if (meteorologiaMunicipio.getMeteorologiaDiaria(i) != null)
{
nodoPrediccionDiaria = new DefaultMutableTreeNode(
meteorologiaMunicipio.getMeteorologiaDiaria(i).getFech
a().substring(8, 10) +"-"++
meteorologiaMunicipio.getMeteorologiaDiaria(i).getFech
a().substring(5, 7) +"-"++
meteorologiaMunicipio.getMeteorologiaDiaria(i).getFech
a().substring(0, 4)
);
nodoPrediccionSemanal.add(nodoPrediccionDiaria);
// PROBABILIDAD DE PRECIPITACION
nodoProbabilidadPrecipitacion = new
DefaultMutableTreeNode("Probabilidad precipitación");
for (int j=0;
j<meteorologiaMunicipio.getPeriodo().length; j++)
{
if
(meteorologiaMunicipio.getMeteorologiaDiaria(i).getPro
babilidadPrecipitacion(j) != null)
{
nodoPeriodoProbabilidadPrecipitacion = new
DefaultMutableTreeNode(meteorologiaMunicipio.getPeriod
o(j)+"
"+meteorologiaMunicipio.getMeteorologiaDiaria(i).getPr
obabilidadPrecipitacion(j));
nodoProbabilidadPrecipitacion.add(nodoPeriodoProbabili
dadPrecipitacion);
```

```
}

}

nodoPrediccionDiaria.add(nodoProbabilidadPrecipitacion
);

// ESTADO DEL CIELO
nodoEstadoCielo = new DefaultMutableTreeNode("Estado
del cielo");
for (int k=0;
k<meteorologiaMunicipio.getPeriodo().length; k++)
{
if
(meteorologiaMunicipio.getMeteorologiaDiaria(i).getEst
adoCielo(k) != null)
{
nodoPeriodoEstadoCielo = new
DefaultMutableTreeNode(meteorologiaMunicipio.getPeriod
o(k)+"
"+meteorologiaMunicipio.getMeteorologiaDiaria(i).getEs
tadoCielo(k));
nodoEstadoCielo.add(nodoPeriodoEstadoCielo);
}
}
nodoPrediccionDiaria.add(nodoEstadoCielo);
// TEMPERATURA
nodoTemperatura = new
DefaultMutableTreeNode("Temperatura");
nodoTemperatura.add(new DefaultMutableTreeNode("Máxima
:
"+meteorologiaMunicipio.getMeteorologiaDiaria(i).getTe
mperaturaMaxima()));
nodoTemperatura.add(new DefaultMutableTreeNode("Mínima
:
"+meteorologiaMunicipio.getMeteorologiaDiaria(i).getTe
mperaturaMinima()));
nodoPrediccionDiaria.add(nodoTemperatura);
// HUMEDAD RELATIVA
nodoHumedadRelativa = new
```

```

DefaultMutableTreeNode("Humedad Relativa");
nodoHumedadRelativa.add(new
DefaultMutableTreeNode("Máxima :
"+meteorologiaMunicipio.getMeteorologiaDiaria(i).getHu
medadRelativaMaxima()));
nodoHumedadRelativa.add(new
DefaultMutableTreeNode("Mínima :
"+meteorologiaMunicipio.getMeteorologiaDiaria(i).getHu
medadRelativaMinima()));
nodoPrediccionDiaria.add(nodoHumedadRelativa);
}
}
}

public static void main(String[] args) {
EventQueue.invokeLater(new Runnable() {
@Override
public void run() {
new InterfazPrediccion();
}
});
}
}
}

```

## Controller

```

package presentacion;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JOptionPane;
public class Controller extends WindowAdapter
implements ActionListener {
private InterfazPrediccion interfazPrediccion;
public Controller(InterfazPrediccion
interfazPrediccion) {
this.interfazPrediccion = interfazPrediccion;
}
}

```

```
}

// MÉTODO DE WindowAdapter
public void windowClosing(WindowEvent e) {
    centralizar("cerrarVentana");
}

// MÉTODO DE ActionListener
public void actionPerformed(ActionEvent e) {
    String componenteFuente = e.getActionCommand();
    switch(componenteFuente)
    {
        case "cargarProvincias" :
        case "comboProvincias":
        case "comboMunicipios":
            centralizar(componenteFuente);
            break;
    }
}

public void centralizar(String actionCommand) {
    try {
        switch (actionCommand)
        {
            case "cargarProvincias":
            case "comboProvincias":
            case "comboMunicipios":
                interfazPrediccion.responderAController(actionCommand);
                ;
                break;
            case "cerrarVentana":
                System.exit(0);
                break;
        }
    } catch (Exception exception)
    {
        JOptionPane.showMessageDialog(null,
        exception.getMessage(), "ERROR",
        JOptionPane.ERROR_MESSAGE); }
    }
}
```

## InvocacionCargaProvincias

```
package presentacion;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.List;
public class InvocacionCargaProvincias {
private List<ActionListener> listaListeners = new
ArrayList();
public InvocacionCargaProvincias() {
}
public void
addEventoCargaProvinciasListener(ActionListener
actionListener) {
listaListeners.add(actionListener);
}
public void
removeEventoCargaProvinciasListener(ActionListener
actionListener) {
listaListeners.remove(actionListener);
}
public void fireEventoCargaProvincias(String command)
{
for (int i=0; i<listaListeners.size(); i++) {
listaListeners.get(i).actionPerformed(new
EventoCargaProvincias(this, 1, command));
}
}
}
}
EventoCargaProvincias
package presentacion;
import java.awt.event.ActionEvent;
public class EventoCargaProvincias extends ActionEvent
{
public EventoCargaProvincias(Object source, int id,
String command) {
super(source, id, command);
}
```

```
}
```

## PrediccionMeteorologica

```
package negocio;
import datos.PrediccionMeteorologicaDATOS;
import encapsuladores.MeteorologiaMunicipio;
public class PrediccionMeteorologica {
    public MeteorologiaMunicipio
    obtenerPrediccionMeteorologica(String idMunicipio)
    throws Exception{
        return new
        PrediccionMeteorologicaDATOS().obtenerPrediccionMeteor
        ologica(idMunicipio);
    }
}
```

## InformacionGeografica

```
package negocio;
import datos.MunicipiosDATOS;
import datos.ProvinciasDATOS;
import java.util.List;
public class InformacionGeografica {
    public List<String> leerProvincias() throws Exception{
        return new ProvinciasDATOS().leerProvincias();
    }
    public List<String> leerMunicipios(String idProvincia)
    throws Exception{
        return new
        MunicipiosDATOS().leerMunicipios(idProvincia);
    }
}
```

## PrediccionMeteorologicaDATOS

```
package datos;
import encapsuladores.MeteorologiaDiaria;
```

```
import encapsuladores.MeteorologiaMunicipio;
import java.util.List;
import org.jdom2.Document;
import org.jdom2.Element;
import org.jdom2.input.SAXBuilder;
public class PrediccionMeteorologicaDATOS {
public MeteorologiaMunicipio
obtenerPrediccionMeteorologica(String idMunicipio)
throws Exception{
MeteorologiaMunicipio meteorologiaMunicipio = new
MeteorologiaMunicipio();
String[] periodo = {"00-24","00-12","12-24","00-
06","06-12","12-18","18-24"};
meteorologiaMunicipio.setPeriodo(periodo);
Document document = new
SAXBuilder().build("http://www.aemet.es/xml/municipios
/localidad_"+idMunicipio+".xml");
Element raiz = document.getRootElement();
Element origen = raiz.getChild("origen");
Element productor = origen.getChild("productor");
String nombreProductor = productor.getText();
meteorologiaMunicipio.setProductor(nombreProductor);
Element nombre = raiz.getChild("nombre");
String nombreMunicipio = nombre.getText();
meteorologiaMunicipio.setNombreMunicipio(nombreMunicipio);
Element prediccion = raiz.getChild("prediccion");
List<Element> listaDias =
prediccion.getChildren("dia");
for (int i=0; i<listaDias.size(); i++)
{
Element elementDia = listaDias.get(i);
meteorologiaMunicipio.setMeteorologiaDiaria(i, new
MeteorologiaDiaria());
meteorologiaMunicipio.getMeteorologiaDiaria(i).setFech
a(elementDia.getAttributeValue("fecha"));
List<Element> listaProbabilidadPrecipitacion =
```

```
elementDia.getChildren("prob_precipitacion");
for (int j=0; j<listaProbabilidadPrecipitacion.size(); j++)
{
Element probabilidadPrecipitacion =
listaProbabilidadPrecipitacion.get(j);
if (probabilidadPrecipitacion.getAttribute("periodo")
== null)
{
meteorologiaMunicipio.getMeteorologiaDiaria(i).setProb
abilidadPrecipitacion(j,
probabilidadPrecipitacion.getText());
}
else
{
if
(meteorologiaMunicipio.getPeriodo(j).compareTo(probabi
lidadPrecipitacion.getAttributeValue("periodo")) == 0)
{
meteorologiaMunicipio.getMeteorologiaDiaria(i).setProb
abilidadPrecipitacion(j,
probabilidadPrecipitacion.getText());
}
}
}
}

List<Element> listaEstadoCielo =
elementDia.getChildren("estado_cielo");
for (int k=0; k<listaEstadoCielo.size(); k++)
{
Element estadoCielo = listaEstadoCielo.get(k);
if (estadoCielo.getAttribute("periodo") == null)
{
meteorologiaMunicipio.getMeteorologiaDiaria(i).setEsta
doCielo(k,
estadoCielo.getAttributeValue("descripcion"));
}
else
```

```

{
if
(meteorologiaMunicipio.getPeriodo(k).compareTo(estadoCielo.getAttributeValue("periodo")) == 0)
{
meteorologiaMunicipio.getMeteorologiaDiaria(i).setEstadoCielo(k,
estadoCielo.getAttributeValue("descripcion"));
}
}
}

Element temperatura =
elementDia.getChild("temperatura");
Element temperaturaMaxima =
temperatura.getChild("maxima");
meteorologiaMunicipio.getMeteorologiaDiaria(i).setTemperaturaMaxima(temperaturaMaxima.getText());
Element temperaturaMinima =
temperatura.getChild("minima");
meteorologiaMunicipio.getMeteorologiaDiaria(i).setTemperaturaMinima(temperaturaMinima.getText());
Element humedadRelativa =
elementDia.getChild("humedad_relativa");
Element humedadRelativaMaxima =
humedadRelativa.getChild("maxima");
meteorologiaMunicipio.getMeteorologiaDiaria(i).setHumedadRelativaMaxima(humedadRelativaMaxima.getText());
Element humedadRelativaMinima =
humedadRelativa.getChild("minima");
meteorologiaMunicipio.getMeteorologiaDiaria(i).setHumedadRelativaMinima(humedadRelativaMinima.getText());
}
return meteorologiaMunicipio;
}
}

```

## ProvinciasDATOS

```
package datos;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStreamReader;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import org.jdom2.Document;
import org.jdom2.Element;
import org.jdom2.input.SAXBuilder;
public class ProvinciasDATOS {
    public List<String> leerProvincias() throws Exception
    {
        List<String> listaProvincias = new ArrayList();
        Document document = new SAXBuilder().build(new
InputStreamReader(new FileInputStream(new
File("xml\\municipios.xml")),
StandardCharsets.ISO_8859_1));
        // Document document = new
SAXBuilder().build("xml\\municipios.xml");
        Element raiz = document.getRootElement();
        List listaProvinciasXML =
raiz.getChildren("PROVINCIA");
        Iterator iteratorProvincias =
listaProvinciasXML.iterator();
        while (iteratorProvincias.hasNext())
        {
            Element provincia =
(Element)iteratorProvincias.next();
            listaProvincias.add(provincia.getAttributeValue("id_pr
ovincia") +
"+provincia.getAttributeValue("nombre"));
        }
        return listaProvincias;
    }
}
```

## MunicipiosDATOS

```
package datos;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import org.jdom2.Document;
import org.jdom2.Element;
import org.jdom2.JDOMException;
import org.jdom2.input.SAXBuilder;
public class MunicipiosDATOS {
    public List<String> leerMunicipios(String idProvincia)
        throws JDOMException, IOException {
        List<String> listaMunicipios = new ArrayList();
        Document document = new SAXBuilder().build(new
            InputStreamReader(new FileInputStream(new
                File("xml\\municipios.xml")),
                StandardCharsets.ISO_8859_1));
        // Document document = new
        // SAXBuilder().build("xml\\municipios.xml");
        Element raiz = document.getRootElement();
        boolean encontradaProvincia = false;
        List listaProvincias = raiz.getChildren("PROVINCIA");
        Iterator iteratorProvincias =
            listaProvincias.iterator();
        while (iteratorProvincias.hasNext() &&
            !encontradaProvincia)
        {
            Element provincia =
                (Element)iteratorProvincias.next();
            if
                (idProvincia.compareTo(provincia.getAttributeValue("id
                    _provincia")) == 0){
```

```

encontradaProvincia = true;
Element municipiosElement =
provincia.getChild("MUNICIPIOS");
List listaMunicipiosXML =
municipiosElement.getChildren("MUNICIPIO");
Iterator iteratorMunicipios =
listaMunicipiosXML.iterator();
while (iteratorMunicipios.hasNext()) {
Element municipio =
(Element)iteratorMunicipios.next();
listaMunicipios.add(municipio.getAttributeValue("id_municipio")+" - "+municipio.getText());
}
}
}
return listaMunicipios;
}
}

```

## MeteorologiaDiaria

```

package encapsuladores;
public class MeteorologiaDiaria {
private String fecha;
private String[] probabilidadPrecipitacion = new
String[7];
private String[] estadoCielo = new String[7];
private String temperaturaMaxima;
private String temperaturaMinima;
private String humedadRelativaMaxima;
private String humedadRelativaMinima;
public String getFecha() {
return fecha;
}
public void setFecha(String fecha) {
this.fecha = fecha;
}

```

```
public String getProbabilidadPrecipitacion(int indice)
{
    return probabilidadPrecipitacion[indice];
}
public void setProbabilidadPrecipitacion(int indice,
String probabilidadPrecipitacion) {
    this.probabilidadPrecipitacion[indice] =
probabilidadPrecipitacion;
}
public String getEstadoCielo(int indice) {
    return estadoCielo[indice];
}
public void setEstadoCielo(int indice, String
estadoCielo) {
    this.estadoCielo[indice] = estadoCielo;
}
public String getTemperaturaMaxima() {
    return temperaturaMaxima;
}
public void setTemperaturaMaxima(String
temperaturaMaxima) {
    this.temperaturaMaxima = temperaturaMaxima;
}
public String getTemperaturaMinima() {
    return temperaturaMinima;
}
public void setTemperaturaMinima(String
temperaturaMinima) {
    this.temperaturaMinima = temperaturaMinima;
}
public String getHumedadRelativaMaxima() {
    return humedadRelativaMaxima;
}
public void setHumedadRelativaMaxima(String
humedadRelativaMaxima) {
    this.humedadRelativaMaxima = humedadRelativaMaxima;
}
```

```
public String getHumedadRelativaMinima() {
    return humedadRelativaMinima;
}
public void setHumedadRelativaMinima(String
    humedadRelativaMinima) {
    this.humedadRelativaMinima = humedadRelativaMinima;
}
}
```

## MeteorologiaMunicipio

```
package encapsuladores;
public class MeteorologiaMunicipio {
    private String productor;
    private String nombreMunicipio;
    private MeteorologiaDiaria[] meteorologiaDiaria = new
        MeteorologiaDiaria[7];
    private String[] periodo = new String[7];
    public String getProductor() {
        return productor;
    }
    public void setProductor(String productor) {
        this.productor = productor;
    }
    public String getNombreMunicipio() {
        return nombreMunicipio;
    }
    public void setNombreMunicipio(String nombreMunicipio)
    {
        this.nombreMunicipio = nombreMunicipio;
    }
    public MeteorologiaDiaria[] getMeteorologiaDiaria() {
        return meteorologiaDiaria;
    }
    public void setMeteorologiaDiaria(MeteorologiaDiaria[]
        meteorologiaDiaria) {
        this.meteorologiaDiaria = meteorologiaDiaria;
    }
}
```

```
}

public MeteorologiaDiaria getMeteorologiaDiaria(int indice) {
    return meteorologiaDiaria[indice];
}

public void setMeteorologiaDiaria(int indice,
    MeteorologiaDiaria meteorologiaDiaria) {
    this.meteorologiaDiaria[indice] = meteorologiaDiaria;
}

public String getPeriodo(int indice) {
    return periodo[indice];
}

public void setPeriodo(int indice, String periodo) {
    this.periodo[indice] = periodo;
}

public String[] getPeriodo() {
    return periodo;
}

public void setPeriodo(String periodo[]) {
    this.periodo = periodo;
}
```

El presente ejercicio ejemplo es una réplica del presentado por el autor en su libro “Programación Orientada a Objetos en JAVA”, en que mediante una interfaz texto, en la consola de salida de le solicitaba al usuario él código de provincia. Con dicho valor introducido se presentaba al usuario la relación de municipios de la provincia seleccionada, y acto seguido se le solicitaba el código de municipio para el que se pretendía visualizar la predicción meteorológica tomando como fuente un fichero XML descargable desde la web de AEMET. Volvemos a recordar en este libro la advertencia legal de que se exige citar a AEMET como fuente de información. Nos remitimos al libro del autor

previamente citado en que los términos de dicha advertencia legal aparecen citados con mayor detalle.

Pretendemos, con la aportación de este ejercicio, los siguientes objetivos:

- Reforzar la utilización del *JComboBox* como componente de selección, en este ejemplo, aplicado a municipio y provincia.
- Presentar un nuevo componente: el *JTree*. En este ejemplo, nos limitamos a cargarlo con la información de la predicción meteorológica del municipio seleccionado, con el único fin de ser presentada al usuario organizada “en árbol”. En la aplicación final **GestionLibros** volverá a aparecer este componente, pero en ese momento, ofreciendo la posibilidad de interactuar con él aplicándole escucha de eventos.
- Presentar una aplicación práctica del mecanismo de lanzamiento explícito de eventos por el código de la aplicación, al que se dedicaron los dos ejercicios anteriores. Concretamente aplicado a la carga inicial del *JComboBox* utilizado para la selección de provincia. El autor ha tomado la decisión de plantear dicha implementación a efectos de que el lector se familiarice con dicho mecanismo, que volverá a aparecer en la aplicación **GestionLibros**, pero en ese momento aplicado a la invocación automática de determinadas opciones del menú. La variante aquí aplicada responde a la expuesta en el primer ejercicio, en que el evento utilizado hereda de *ActionEvent*. Como ya se comentó en su momento, la utilización de un evento con dicho nivel de concreción, nos permite homogeneizar su

tratamiento con otros predefinidos del mismo tipo, en este caso concreto, los lanzados al seleccionar un ítem en los *JComboBox*.

- Introducir a un nuevo modelo de configuración de la gestión de eventos. Constituye una evolución en relación a la que se ha venido aplicando hasta ahora en la mayor parte de aplicaciones con interfaz gráfico. Dicha evolución responde a un intento de acercamiento progresivo al utilizado en la aplicación final **GestionLibros**, en que la configuración allí aplicada pretende ser una aproximación al patrón de diseño Front Controller. En las aplicaciones anteriores, la clase **GestorEventos** asumía el tratamiento centralizado de todos los eventos que se producían en la aplicación, siendo el método específico de cada Listener el que implementaba las acciones a ejecutar en cada caso, finalizando en dicho método el tratamiento específico de cada evento. Ahora, la clase **GestorEventos** “ha evolucionado” a la clase **Controller**. El cambio más significativo consiste en que todos los métodos escucha de eventos, ya no asumen la implementación de las actuaciones finales de cada evento, sino que todos ellos derivan al método **centralizar()** de esta clase:

```
centralizar("cerrarVentana");  
centralizar(componenteFuente);
```

que su vez redirige el flujo de ejecución al método **responderAController()**:

```
interfazPrediccion.responderAController(actionComm  
and);
```

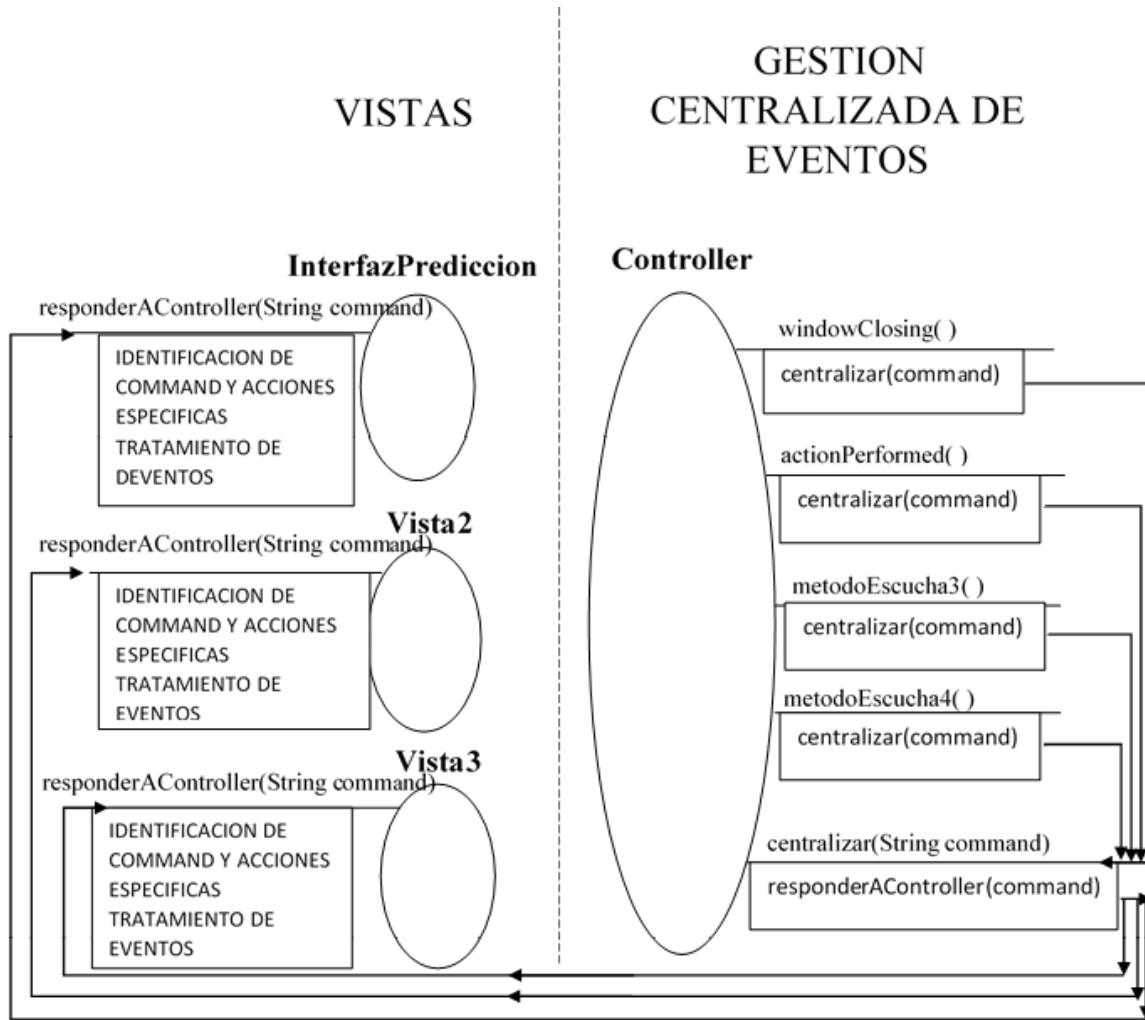
presente en la interfaz gráfica, y que contiene el componente que generó el evento. En este método, la

primera actuación es identificar el componente fuente del evento para proceder a continuación a un tratamiento diferenciado y específico para cada uno de ellos:

```
public void responderAController(String  
actionCommand) throws Exception {  
switch(actionCommand)  
{  
case "cargarProvincias" :  
    . . .  
break;  
case "comboProvincias":  
    . . .  
break;  
case "comboMunicipios":  
    . . .  
break;  
}  
}
```

En esta aplicación, el lector todavía no adquirirá perspectiva de las intenciones del autor, sugerimos que empiece ya a analizar este mecanismo en **GestionLibros** que dispone de un número considerable de vistas, y diversidad de métodos escucha de eventos, todos ellos presentes en la clase **Controller**. El *ActionCommand* adquiere total protagonismo con este modelo, actuando como “testigo” del evento, transfiriéndose entre los métodos intervinientes de la vista y del **Controller**, a efectos de que allá donde se requiera, actúe como elemento diferenciador e identificador del origen del evento. El evento **EventoCargaProvincias**, cuyo lanzamiento explícito por el programa determina la carga del *JComboBox* de selección de provincia, ha sido especialmente diseñado para poder encajar con la gestión conjunta del resto de eventos. Presentamos el siguiente

esquema para auxiliar al lector en la interpretación de lo detallado:



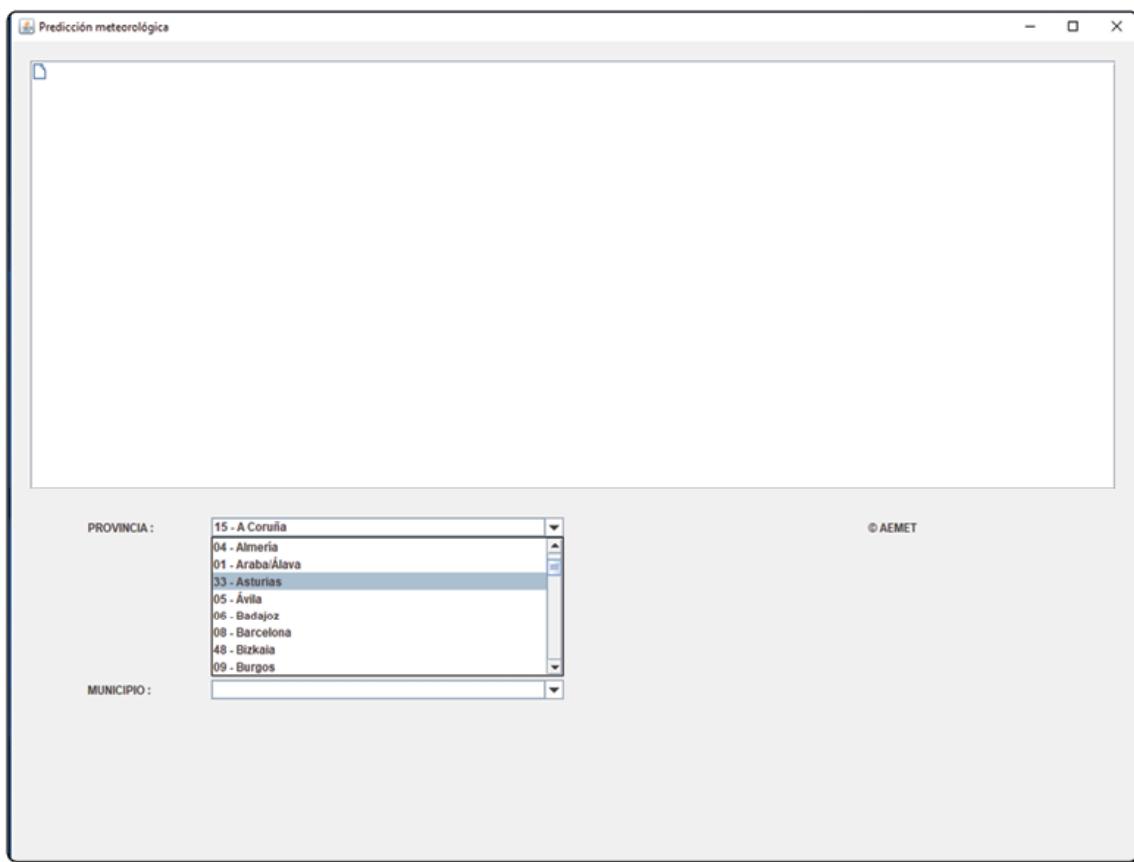
En el anterior esquema hemos ubicado **Vista2** y **Vista3**, que obviamente no existen en este ejemplo. También se han ubicado `metodoEscucha3` y `metodoEscucha4`. Se ha aplicado al esquema esta “extensión virtual” de vistas y de métodos escucha, porque esta aplicación ejemplo solamente dispone de una vista, **InterfazPrediccion**, y la configuración que estamos presentando empieza a rentabilizarse y a tener sentido con la presencia de varias vistas, y diversidad de métodos escucha de eventos en la clase **Controller**. Insistimos, es el modelo aplicable a

**GestionLibros**, que afrontamos aquí en preparación del estudio en detalle de la citada aplicación final. Todo ello, nos abrirá las puertas en dicha aplicación a un control centralizado y a una implementación polimórfica de las vistas.

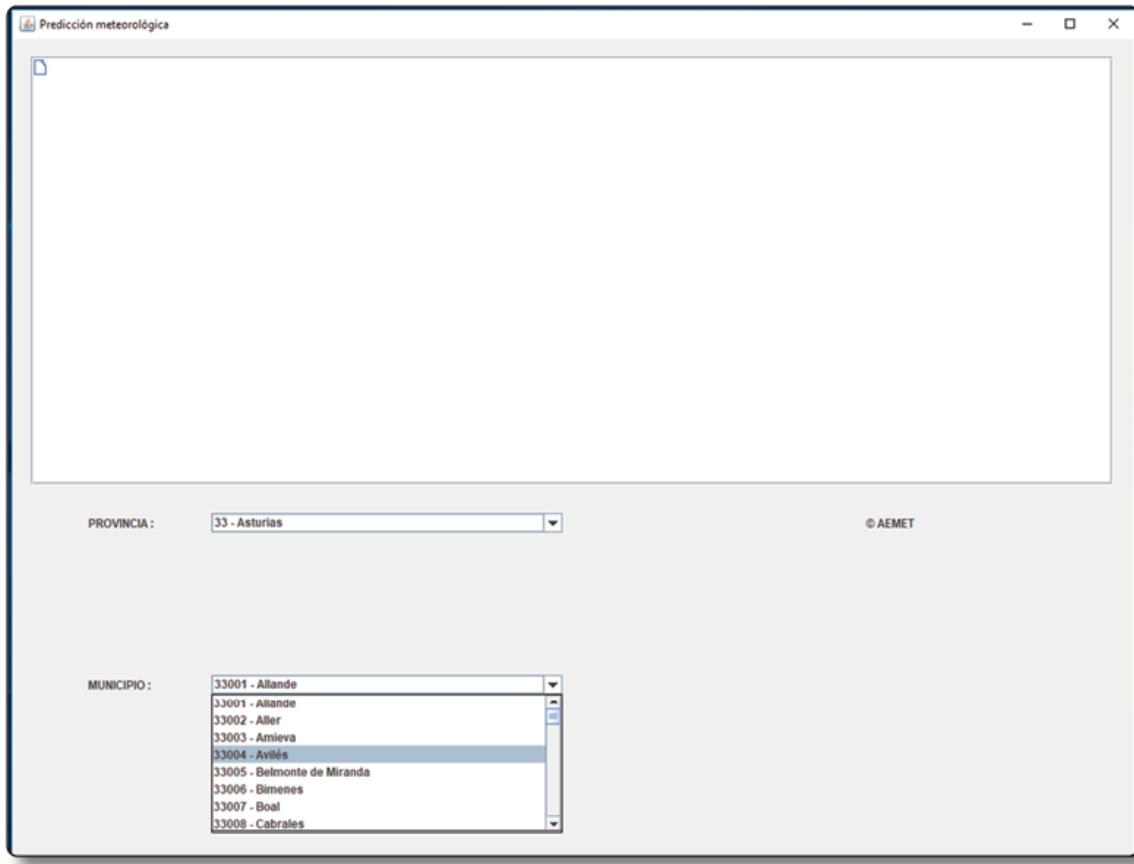
Como ya hemos mencionado, la información sobre la predicción meteorológica del municipio seleccionado se obtiene de la lectura del XML descargado desde la web de AEMET con dicha predicción meteorológica. Si el lector tiene inquietudes sobre los mecanismos relacionados con la lectura escritura de ficheros XML desde una aplicación Java, le remitimos al libro anteriormente mencionado del mismo autor, en que se destina un tema en exclusiva al tratamiento de XML desde Java. La carga de la relación de provincias y municipios se realiza desde el fichero presente en el proyecto **municipios.xml**, ubicado en el directorio **xml**. En el libro del autor que venimos citando, se trata dicho fichero, es más, se dedica una aplicación ejemplo en exclusiva para la obtención de dicho fichero. Si el lector tiene inquietudes sobre cómo se genera, le remitimos a dicho libro. En el presente, nos limitamos, simplemente a dejarlo disponible ya generado.

Lo que acabamos de mencionar suponen todos los detalles concernientes a las clases de la capa **datos**, y la capa **negocio** en esta aplicación, asume solamente la invocación al tratamiento XML implementado por los métodos de las clases de la capa **datos**. Las clases ubicadas en la capa **encapsuladores**, encapsulan (valga la redundancia) la información leída desde orígenes XML.

La ejecución de la aplicación presenta la siguiente ventana:

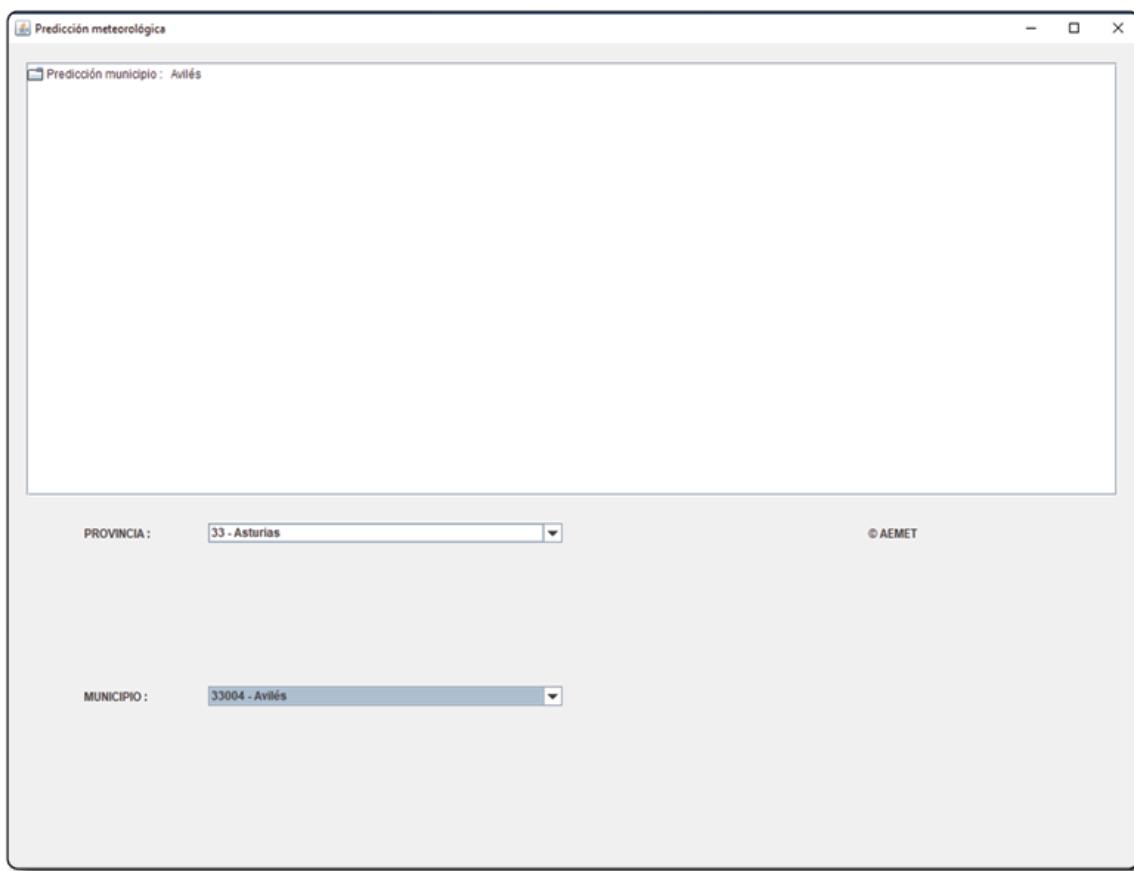


en que el *JComboBox* correspondiente a la selección de la provincia se presenta previamente cargado. Seleccionamos provincia e inmediatamente se carga el *JComboBox* de selección de municipio con los municipios de la provincia seleccionada:

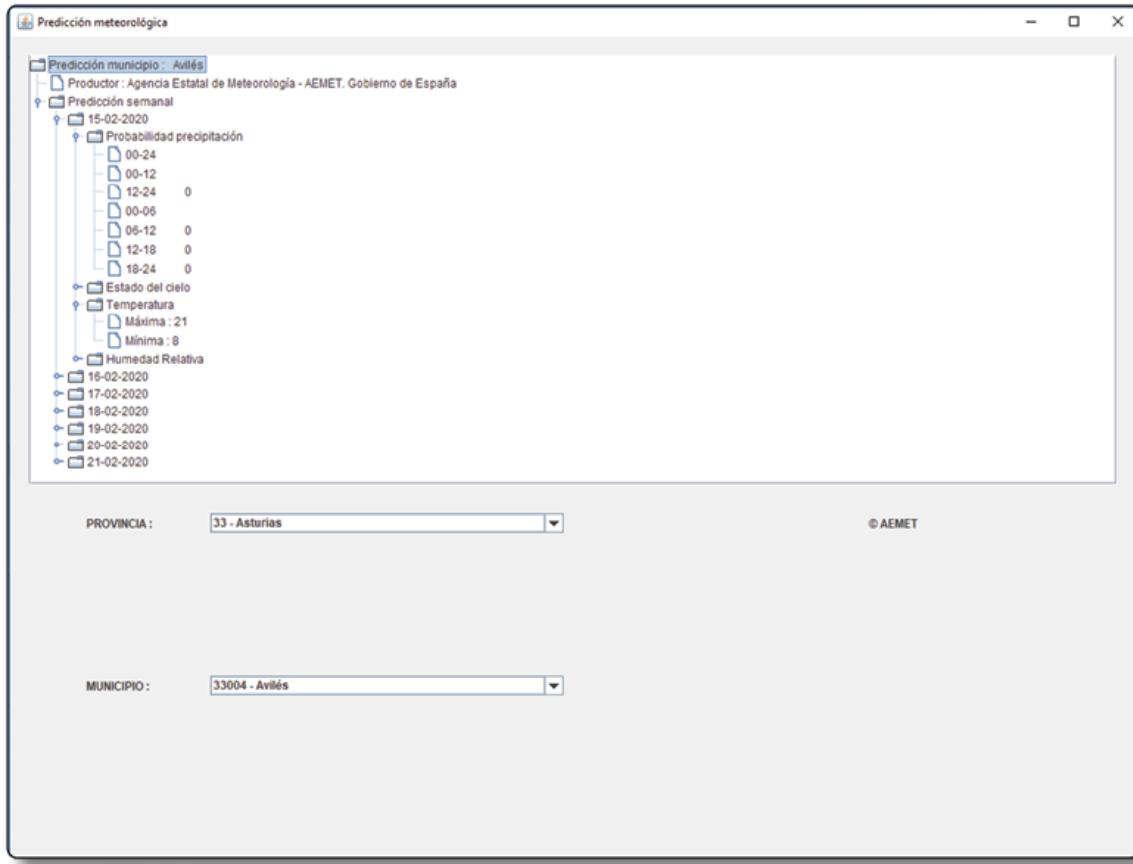


Seleccionamos municipio, y se carga el *JTree* con la predicción meteorológica del día actual y de los próximos días. En pro de minimizar la extensión del código a utilizar hemos contemplado, para cada día, solamente los parámetros meteorológicos más representativos.

El cambio de apariencia del nodo raíz y la aparición del texto:



Predicción municipio : NOMBRE MUNICIPIO nos indican que el *JTree* ya se ha cargado y que podemos proceder a su despliegue haciendo doble click en el icono representativo de aquellos nodos de los que se pretenda desplegar el subárbol correspondiente. Dicha operación en un nodo desplegado provoca el repliegue del mismo. Vemos que en la parte izquierda de todos los nodos a partir de los que se puede desplegar un subárbol, aparece un pequeño círculo. Hacer click en dicho círculo tiene el mismo efecto de apertura y cierre que doble click en el icono representativo del nodo. Dicho círculo es punto de origen de un pequeño segmento, que se posiciona verticalmente hacia abajo cuando el subárbol correspondiente está desplegado, y horizontal hacia la derecha cuando se encuentra replegado.



Para completar el análisis con detalle de la implementación presente en esta aplicación ejemplo, solamente nos resta ocuparnos de las clases ubicadas en la capa **presentacion**. Los aspectos más relevantes, ya se han comentado con anterioridad cuando hemos descrito el modelo de gestión de eventos, que de forma pionera empezamos a aplicar ya en esta aplicación. Tratar aquí los mecanismos aplicados a:

- El establecimiento del *ActionCommand* para cada *JComboBox*.
- La carga de los *JComboBox* de selección de provincia y municipio.

- Lectura del elemento seleccionado por el usuario en el *JComboBox*.

supondría caer en redundancia con las explicaciones minuciosas y detalladas que se han hecho al respecto en aplicaciones ejemplo anteriores. Sí que procede tratar con detalle los mecanismos asociados al *JTree* que de forma novedosa aparece en este ejemplo. Este componente aparece acompañado por elementos que ya se han tratado con anterioridad en otros componentes:

- *JScrollPane*, que nos aporta una barra de desplazamiento auxiliar para la visualización, caso de ser necesario en función del grado de despliegue que aplique el usuario a los subárboles del *JTree*, produciéndose el acoplamiento en:

```
JScrollPane jScrollPane = new JScrollPane(jTree);
```

- Un *Model*, en este caso *DefaultTreeModel*, que represente, almacene y permita la gestión de los datos que el componente se limita a visualizar. Lo obtenemos:

```
modelo = (DefaultTreeModel)jTree.getModel();
```

Todo nodo del *Jtree* es modelado por la clase *DefaultMutableTreeNode*, encontrándonos en esta aplicación con:

- nodoRaiz
- nodoProductor
- nodoPrediccionSemanal
- nodoPrediccionDiaria

- nodoProbabilidadPrecipitacion
- nodoPeriodoProbabilidadPrecipitacion
- nodoEstadoCielo
- nodoPeriodoEstadoCielo
- nodoTemperatura
- nodoHumedadRelativa

Viniendo determinado el destino de cada uno de ellos por el tipo de información a mostrar, así como de la jerarquía que ocupa dentro del árbol. Al constructor del *JTree* se le transfiere como argumento la referencia a la instancia del nodo que actuará como raíz en el árbol:

```
nodoRaiz = new DefaultMutableTreeNode(" ");
jTree = new JTree( nodoRaiz );
```

El árbol se construye añadiendo cada nodo al nodo padre correspondiente:

```
nodoRaiz.add(nodoPrediccionSemanal);
```

Se han utilizado dos formas diferentes de asignar contenido a un nodo:

- como argumento transferido al constructor del mismo:

```
nodoPrediccionSemanal = new
DefaultMutableTreeNode("Predicción semanal");
```

- mediante el método:

```
nodoRaiz.setUserObject("Predicción municipio :
"+meteorologiaMunicipio.getNombreMunicipio());
```

Para la inicialización del contenido del árbol hemos utilizado los métodos:

```
nodoRaiz.removeAllChildren();
modelo.reload();
```

El lanzamiento explícito de evento por parte del código de la aplicación se produce en **InterfazPrediccion**, mecanismo ya tratado amplia y exhaustivamente en los dos ejercicios ejemplo anteriores. En este caso:

- el evento es modelado por

```
class EventoCargaProvincias extends ActionEvent
```

- la clase escucha del evento es **Controller**, centralizadora de la gestión de eventos, que al asumir *implements ActionListener*, puede erigirse en clase oyente del evento que estamos tratando
- el papel de componente lanzador del evento es asumido, en este caso, por la clase **InvocacionCargaProvincias**, que implementa los métodos

```
addEventoCargaProvinciasListener(controller);
```

que se encarga del registro con la clase escucha e

```
fireEventoCargaProvincias("cargarProvincias");
```

que representa el lanzamiento efectivo del evento. Estos últimos métodos son invocados desde la vista **InterfazPrediccion**. Observemos que el argumento transferido a este último método actuará como *ActionCommand* del evento.

25

## APLICACIÓN VOTACIONPROPIUESTA

La estructura que presenta esta aplicación corresponde a la establecida por el modelo de desarrollo de software arquitectura a tres capas:

- *package presentacion* aglutina las clases:
  - PantallaJTable
  - ModeloDatos
  - Componentes
  - GestorEventos
  - GestorExcepciones
- *package negocio* aglutina las clases:
  - RepositorioNegocio
  - TitularesNegocio
  - VotacionesNegocio
  - IncidenciasNegocio
- *package datos* aglutina las clases:
  - RepositorioXML
  - ConexionBaseDatos

- TitularesDatos
- VotacionesDatos
- VotosEmitidosDatos
- IncidenciasDatos
- VotacionesPDF
- *package encapsuladores* aglutina las clases:
  - SistemaArchivos
  - Contexto
  - BaseDatos
  - Titular
  - Votacion
  - VotoEmitido
- *package excepciones* aglutina las clases:
  - GenericaExcepcion

## PantallaJTable

```
package presentacion;
import encapsuladores.BaseDatos;
import encapsuladores.Contexto;
import encapsuladores.Votacion;
import java.awt.Color;
import java.awt.EventQueue;
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.text.DecimalFormat;
import java.text.DecimalFormatSymbols;
import java.util.List;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JFrame;
```

```
import javax.swing.JLabel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.table.TableColumn;
import negocio.RepositorioNegocio;
import negocio.TitularesNegocio;
import negocio.VotacionesNegocio;
public class PantallaJTable extends JFrame {
private Contexto contexto = null;
public PantallaJTable() {
Object[] repositorio = null;
try {
contexto = new Contexto("usuario1", obtenerIP());
repositorio = new
RepositorioNegocio().cargarRepositorio();
setSize(1200,900);
setTitle("Votación propuesta");
Componentes componentes = new Componentes();
componentes.setPantallaJTable(this);
componentes.setRepositorio(repositorio);
componentes.setContexto(contexto);
ubicarComponentes(componentes);
componentes.getModeloDatos().cargarTitulares(new
TitularesNegocio().consultarTodos((BaseDatos)componentes.getRepositorio()[0]));
setVisible(true);
} catch (Exception exception)
{ new
GestorExcepciones().gestionarExpcion(exception,
contexto); }
}
private String obtenerIP() throws UnknownHostException
{
String[] cadenasIP =
InetAddress.getLocalHost().getHostAddress().split("\\.");
}
```

```
StringBuffer procesoIP = new StringBuffer(16);
for (int i=0; i<cadenasIP.length; i++)
{
if (i>0)
procesoIP.append(".");
procesoIP.append(String.format("%03d",
Integer.parseInt(cadenasIP[i])));
}
return new String(procesoIP);
}
private Componentes ubicarComponentes(Componentes componentes) throws Exception {
DecimalFormatSymbols decimalFormatSymbols = new
DecimalFormatSymbols();
DecimalFormat decimalFormat = new
DecimalFormat("##0.000", decimalFormatSymbols);
setLayout(null);
GestorEventos gestorEventos = new
GestorEventos(componentes);
addWindowListener(gestorEventos); // REGISTRO DE
ESCUCHA DE EVENTOS DE VENTANA
componentes.setModeloDatos(new
ModeloDatos(componentes, new
TitularesNegocio().consultarNumeroFilas((BaseDatos)com
ponentes.getRepositorio()[0])));
componentes.setjTable(new
JTable(componentes.getModeloDatos()));
componentes.setJScrollPaneTabla(new
JScrollPane(componentes.getjTable()));
componentes.getJScrollPaneTabla().setBounds(20,20,1140
,500);
add(componentes.getJScrollPaneTabla());
TableColumn columna[] = new TableColumn[7];
columna[0] =
componentes.getjTable().getColumnModel().getColumn(0);
columna[0].setPreferredWidth(30);
columna[1] =
```

```
componentes.getjTable().getColumnModel().getColumn(1);
columna[1].setPreferredWidth(350);
columna[2] =
componentes.getjTable().getColumnModel().getColumn(2);
columna[2].setPreferredWidth(50);
columna[3] =
componentes.getjTable().getColumnModel().getColumn(3);
columna[3].setPreferredWidth(50);
columna[4] =
componentes.getjTable().getColumnModel().getColumn(4);
columna[4].setPreferredWidth(50);
columna[5] =
componentes.getjTable().getColumnModel().getColumn(5);
columna[5].setPreferredWidth(50);
columna[6] =
componentes.getjTable().getColumnModel().getColumn(6);
columna[6].setPreferredWidth(50);
for (int i=0; i<4; i++)
{
componentes.setVisualizaNumVotos( new
JLabel(decimalFormat.format(componentes.getContadorVot
os(i))) ,i);
componentes.getVisualizaNumVotos(i).setBounds(730+
(i*120), 530, 100, 50);
add(componentes.getVisualizaNumVotos(i));
}
componentes.setBotonCargaVotacion(new JButton("Carga
votación"));
componentes.getBotonCargaVotacion().setBounds(100,
600, 150, 50);
componentes.getBotonCargaVotacion().addActionListener(
gestorEventos); // REGISTRO DE ESCUCHA DE EVENTO DE
BOTON
add(componentes.getBotonCargaVotacion());
componentes.setSeleccionIdVotacion(new JComboBox());
componentes.getSeleccionIdVotacion().setBounds(400,
620, 600, 20);
```

```
componentes.getSeleccionIdVotacion().setBackground(Color.white);
add(componentes.getSeleccionIdVotacion());
cargarComboVotaciones(componentes);
componentes.setBotonNuevaVotacion(new JButton("Nueva votación"));
componentes.getBotonNuevaVotacion().setBounds(100, 675, 150, 50);
componentes.getBotonNuevaVotacion().addActionListener(gestorEventos); // REGISTRO DE ESCUCHA DE EVENTO DE BOTON
add(componentes.getBotonNuevaVotacion());
componentes.setEtiquetaTemaVotado(new JLabel("Tema votado"));
componentes.getEtiquetaTemaVotado().setBounds(300, 675, 150, 50);
add(componentes.getEtiquetaTemaVotado());
componentes.setjTextFieldTemaVotado(new JTextField());
componentes.getjTextFieldTemaVotado().setBounds(400, 695, 600, 20);
add(componentes.getjTextFieldTemaVotado());
componentes.setBotonGuardaVotacion(new JButton("Guarda votación"));
componentes.getBotonGuardaVotacion().setBounds(100, 750, 150, 50);
componentes.getBotonGuardaVotacion().addActionListener(gestorEventos); // REGISTRO DE ESCUCHA DE EVENTO DE BOTON
add(componentes.getBotonGuardaVotacion());
return componentes;
}
public void cargarComboVotaciones(Componentes componentes) throws Exception {
componentes.getSeleccionIdVotacion().removeAllItems();
List<Votacion> listaVotaciones = new VotacionesNegocio().consultarTodasVotaciones((BaseDatos)componentes.getRepositorio())[0]);
```

```

for (int i=0; i<listaVotaciones.size(); i++)
{
    Votacion votacion = listaVotaciones.get(i);
    componentes.getSeleccionIdVotacion().addItem(votacion.
    getIdVotacion()+" "+votacion.getTemaVotado());
}
}

public static void main(String[] args) {
EventQueue.invokeLater(new Runnable() {
@Override
public void run() {
new PantallaJTable();
}
});
}
}
}

```

## ModeloDatos

```

package presentacion;
import encapsuladores.Titular;
import encapsuladores.VotoEmitido;
import java.text.DecimalFormat;
import java.text.DecimalFormatSymbols;
import java.util.List;
import javax.swing.table.AbstractTableModel;
public class ModeloDatos extends AbstractTableModel {
private Object datos[][][];
private String[] nombreColumnas =
{“Código”, “Nombre”, “Cuota
participación”, “AUSENTE”, “ABSTENCION”, “SI”, “NO”};
private Componentes componentes;
private int numeroFilas = 0;
public ModeloDatos(Componentes componentes, int
numeroFilas) {
this.componentes = componentes;
this.numeroFilas = numeroFilas;
}
}

```

```
datos = new Object[numeroFilas][7];
inicializarFilas(0, numeroFilas-1);
addTableModelListener(new GestorEventos(componentes));
}
private void inicializarFilas(int filaInicial, int
filaFinal) {
for (int i=filaInicial; i<=filaFinal; i++)
{ datos[i][0] = "";
datos[i][1] = "";
datos[i][2] = new Double(0);
datos[i][3] = new Boolean(false);
datos[i][4] = new Boolean(false);
datos[i][5] = new Boolean(false);
datos[i][6] = new Boolean(false);
}
}
public void cargarTitulares(List<Titular>
listaCopropietarios) {
DecimalFormatSymbols decimalFormatSymbols = new
DecimalFormatSymbols();
DecimalFormat decimalFormat = new
DecimalFormat("##0.000", decimalFormatSymbols);
for (int k=0; k<=3; k++)
componentes.setContadorVotos(0, k);
for (int i=0; i<listaCopropietarios.size(); i++)
{
Titular copropietario = listaCopropietarios.get(i);
datos[i][0] = copropietario.getCodigo();
datos[i][1] = copropietario.getNombre();
datos[i][2] = new
Double(copropietario.getCuotaParticipacion());
}
for (int k=0; k<=3; k++)
componentes.getVisualizaNumVotos(k).setText(decimalFor
mat.format(componentes.getContadorVotos(k)));
componentes.getjTable().repaint();
}
```

```
public void cargarVotacion(List<VotoEmitido>
listaVotosEmitidos) {
for (int k=0; k<=3; k++)
componentes.setContadorVotos(0, k);
for (int i=0; i<listaVotosEmitidos.size(); i++)
{
VotoEmitido votoEmitido = listaVotosEmitidos.get(i);
for (int k=0; k<=3; k++)
{
datos[i][k+3] =
votoEmitido.getComponenteOpcionesVotacion(k);
if
(((Boolean)votoEmitido.getComponenteOpcionesVotacion(k
)).booleanValue())
componentes.setContadorVotos(componentes.getContadorVo
tos(k) + ((Double)datos[i][2]).doubleValue(), k);
}
}
componentes.getjTable().repaint();
}
public void inicializarVotacion() {
for (int k=0; k<=3; k++)
componentes.setContadorVotos(0, k);
for (int i=0; i<datos.length; i++)
{
datos[i][3] = new Boolean(true);
datos[i][4] = new Boolean(false);
datos[i][5] = new Boolean(false);
datos[i][6] = new Boolean(false);
componentes.setContadorVotos(componentes.getContadorVo
tos(0) + ((Double)datos[i][2]).doubleValue(), 0);
}
componentes.getjTable().repaint();
}
public Object[][] getDatos() {
return datos;
}
```

```

public int getColumnCount() {
return(datos[0].length );
}
public int getRowCount() {
return(datos.length );
}
public Object getValueAt(int fila, int column) {
return datos[fila][column];
}
public void setValueAt(Object valor,int fila, int
columna ) {
datos[fila][columna] = valor;
// fireTableCellUpdated(fila, column);
componentes.setFilaActualizada(fila);
componentes.setColumnaActualizada(columna);
fireTableDataChanged();
}
public boolean isCellEditable(int fila, int column) {
if (columna < 3 || columna > 6)
return( false );
else
return( true );
}
public String getColumnName(int column) {
return nombreColumnas[column];
}
public Class getColumnClass(int column) {
return getValueAt(0,column).getClass();
}
}

```

## Componentes

```

package presentacion;
import encapsuladores.Contexto;
import javax.swing.JButton;

```

```
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextField;
public class Componentes {
private Object[] repositorio;
private Contexto contexto;
private PantallaJTable pantallaJTable;
private JTable jTable;
private ModeloDatos modeloDatos;
private JScrollPane JScrollPaneTabla;
private JLabel[] visualizaNumVotos = new JLabel[4];
private double[] contadorVotos = {0,0,0,0};
private JButton botonCargaVotacion;
private JButton botonNuevaVotacion;
private JButton botonGuardaVotacion;
private JComboBox seleccionIdVotacion;
private JLabel etiquetaTemaVotado;
private JTextField jTextFieldTemaVotado;
private int filaActualizada;
private int columnaActualizada;
public JLabel getVisualizaNumVotos(int componente) {
return visualizaNumVotos[componente];
}
public void setVisualizaNumVotos(JLabel
visualizaNumVotos, int componente) {
this.visualizaNumVotos[componente] =
visualizaNumVotos;
}
public double getContadorVotos(int componente) {
return contadorVotos[componente];
}
public void setContadorVotos(double contadorVotos, int
componente) {
this.contadorVotos[componente] = contadorVotos;
}
```

```
public Object[] getRepositorio() {
    return repositorio;
}
public void setRepositorio(Object[] repositorio) {
    this.repositorio = repositorio;
}
public Contexto getContexto() {
    return contexto;
}
public void setContexto(Contexto contexto) {
    this.contexto = contexto;
}
public PantallaJTable getPantallaJTable() {
    return pantallaJTable;
}
public void setPantallaJTable(PantallaJTable pantallaJTable) {
    this.pantallaJTable = pantallaJTable;
}
public JTable getjTable() {
    return jTable;
}
public void setjTable(JTable jTable) {
    this.jTable = jTable;
}
public ModeloDatos getModeloDatos() {
    return modeloDatos;
}
public void setModeloDatos(ModeloDatos modeloDatos) {
    this.modeloDatos = modeloDatos;
}
public JScrollPane getJScrollPaneTabla() {
    return JScrollPaneTabla;
}
public void setJScrollPaneTabla(JScrollPane JScrollPaneTabla) {
    this.JScrollPaneTabla = JScrollPaneTabla;
}
```

```
}

public JLabel[] getVisualizaNumVotos() {
    return visualizaNumVotos;
}

public void setVisualizaNumVotos(JLabel[] visualizaNumVotos) {
    this.visualizaNumVotos = visualizaNumVotos;
}

public double[] getContadorVotos() {
    return contadorVotos;
}

public void setContadorVotos(double[] contadorVotos) {
    this.contadorVotos = contadorVotos;
}

public JButton getBotonCargaVotacion() {
    return botonCargaVotacion;
}

public void setBotonCargaVotacion(JButton botonCargaVotacion) {
    this.botonCargaVotacion = botonCargaVotacion;
}

public JButton getBotonNuevaVotacion() {
    return botonNuevaVotacion;
}

public void setBotonNuevaVotacion(JButton botonNuevaVotacion) {
    this.botonNuevaVotacion = botonNuevaVotacion;
}

public JButton getBotonGuardaVotacion() {
    return botonGuardaVotacion;
}

public void setBotonGuardaVotacion(JButton botonGuardaVotacion) {
    this.botonGuardaVotacion = botonGuardaVotacion;
}

public JComboBox getSeleccionIdVotacion() {
    return seleccionIdVotacion;
}
```

```
}

public void setSeleccionIdVotacion(JComboBox seleccionIdVotacion) {
this.seleccionIdVotacion = seleccionIdVotacion;
}
public JLabel getEtiquetaTemaVotado() {
return etiquetaTemaVotado;
}
public void setEtiquetaTemaVotado(JLabel etiquetaTemaVotado) {
this.etiquetaTemaVotado = etiquetaTemaVotado;
}
public JTextField getjTextFieldTemaVotado() {
return jTextFieldTemaVotado;
}
public void setjTextFieldTemaVotado(JTextField jTextFieldTemaVotado) {
this.jTextFieldTemaVotado = jTextFieldTemaVotado;
}
public int getFilaActualizada() {
return filaActualizada;
}
public void setFilaActualizada(int filaActualizada) {
this.filaActualizada = filaActualizada;
}
public int getColumnaActualizada() {
return columnaActualizada;
}
public void setColumnaActualizada(int columnaActualizada) {
this.columnaActualizada = columnaActualizada;
}
}
```

## GestorEventos

```
package presentacion;
```

```
import encapsuladores.BaseDatos;
import encapsuladores.SistemaArchivos;
import encapsuladores.Votacion;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.text.DecimalFormat;
import java.text.DecimalFormatSymbols;
import java.text.SimpleDateFormat;
import javax.swing.JButton;
import javax.swing.event.TableModelEvent;
import javax.swing.event.TableModelListener;
import negocio.VotacionesNegocio;
public class GestorEventos extends WindowAdapter
implements ActionListener, TableModelListener {
private Componentes componentes;
private int filaSeleccionada = -1;
public GestorEventos(Componentes componentes) {
this.componentes = componentes;
}
// MÉTODO DE WindowAdapter
public void windowClosing(WindowEvent e) {
System.exit(0);
}
// MÉTODO DE ActionListener
public void actionPerformed(ActionEvent e) {
DecimalFormatSymbols decimalFormatSymbols = new
DecimalFormatSymbols();
DecimalFormat decimalFormat = new
DecimalFormat("##0.000", decimalFormatSymbols);
JButton jButton = (JButton) e.getSource();
if (jButton == componentes.getBotonCargaVotacion()) // // BOTON Carga Votacion
{
if
(componentes.getSeleccionIdVotacion().getSelectedItem(
```

```
) != null)
{
Votacion votacion = new Votacion();
votacion.setIdVotacion(componentes.getSeleccionIdVotacion().getSelectedItem().toString().substring(0, 19));
try {
componentes.getModeloDatos().cargarVotacion(new
VotacionesNegocio().consultarResultadoVotacion((BaseDa
tos)componentes.getRepository()[0], votacion));
} catch (Exception exception)
{ new
GestorExcepciones().gestionarExcepcion(exception,
componentes.getContexto()); }
for (int k=0; k<=3; k++)
componentes.getVisualizaNumVotos(k).setText(decimalFor
mat.format(componentes.getContadorVotos(k)));
componentes.getjTextFieldTemaVotado().setText(componentes
.getSeleccionIdVotacion().getSelectedItem().toStrin
g().substring(24));
}
}
else
if (jButton == componentes.getBotonNuevaVotacion()) // // BOTON Nueva Votacion
{
componentes.getModeloDatos().inicializarVotacion();
for (int k=0; k<=3; k++)
componentes.getVisualizaNumVotos(k).setText(decimalFor
mat.format(componentes.getContadorVotos(k)));
componentes.getjTextFieldTemaVotado().setText("");
}
else
if (jButton == componentes.getBotonGuardaVotacion())
// BOTON Guarda Votacion
{
Votacion votacion = new Votacion();
votacion.setIdVotacion(new SimpleDateFormat("yyyy-MM-
```

```
dd HH:mm:ss").format(new java.util.Date()));
votacion.setTemaVotado(componentes.getjTextFieldTemaVotado().getText());
try {
new
VotacionesNegocio().guardarVotacion((BaseDatos)componentes.getRepositorio()[0],
(SistemaArchivos)componentes.getRepositorio()[1],
votacion, componentes.getModeloDatos().getDatos(),
componentes.getContadorVotos());
componentes.getPantallaJTable().cargarComboVotaciones(
componentes);
} catch (Exception exception)
{ new
GestorExcepciones().gestionarExcepcion(exception,
componentes.getContexto()); }
}
}
// MÉTODO DE TableModelListener
public void tableChanged(TableModelEvent e )
{
// En esta aplicación, todas las columnas
actualizables son objetos Boolean. En el caso de que
otras columnas con otra clase de objetos
// asociados también fuesen actualizables, la
implementación de este método debería estar sometida a
la siguiente condición:
// if (componentes.getColumnaActualizada() >= 3 &&
componentes.getColumnaActualizada() <=6)
DecimalFormatSymbols decimalFormatSymbols = new
DecimalFormatSymbols();
DecimalFormat decimalFormat = new
DecimalFormat("##0.000", decimalFormatSymbols);
if (!((Boolean)componentes.getModeloDatos().getDatos()
[componentes.getFilaActualizada()])
[componentes.getColumnaActualizada()]).booleanValue()
{
```

```
componentes.getModeloDatos().getDatos()
[componentes.getFilaActualizada()]
[componentes.getColumnaActualizada()] = new
Boolean(true);
}
else
{
componentes.setContadorVotos(componentes.getContadorVo
tos(componentes.getColumnaActualizada()-3) + new
Double((Double)
(componentes.getModeloDatos().getDatos()
[componentes.getFilaActualizada()][2])).doubleValue(),
componentes.getColumnaActualizada()-3);
componentes.getVisualizaNumVotos(componentes.getColumn
aActualizada()-3).setText(decimalFormat.format(component
es.getContadorVotos(componentes.getColumnaActualizad
a()-3)));
for (byte i=3; i<=6; i++)
{
if (((Boolean)componentes.getModeloDatos().getDatos()
[componentes.getFilaActualizada()][i]).booleanValue()
&& i != componentes.getColumnaActualizada())
{
componentes.getModeloDatos().getDatos()
[componentes.getFilaActualizada()][i] = new
Boolean(false);
componentes.setContadorVotos(componentes.getContadorVo
tos(i-3) - new Double((Double)
(componentes.getModeloDatos().getDatos()
[componentes.getFilaActualizada()][2])).doubleValue(),
i-3);
componentes.getVisualizaNumVotos(i-
3).setText(decimalFormat.format(Math.abs(componentes.g
etContadorVotos(i-3))));
```

```
}\n}
```

## GestorExcepciones

```
package presentacion;\nimport encapsuladores.Contexto;\nimport excepciones.GenericaExcepcion;\nimport java.io.IOException;\nimport javax.swing.JOptionPane;\nimport negocio.IncidenciasNegocio;\npublic class GestorExcepciones {\n    public void gestionarExcepcion(Exception excepcion,\n        Contexto contexto) {\n        int codigoError = 0;\n        String mensajeError = "";\n        if (excepcion instanceof GenericaExcepcion)\n        {\n            GenericaExcepcion genericaExcepcion =\n                (GenericaExcepcion)excepcion;\n            codigoError = genericaExcepcion.getCodigoError();\n            switch (genericaExcepcion.getCodigoError())\n            {\n                case 20: mensajeError = "Se ha producido una situación\n                    de error como consecuencia de problemas con la\n                    conexión a la BD";\n                break;\n                case 30: mensajeError = "Se ha producido una situación\n                    de error en la lectura del fichero XML del\n                    repositorio";\n                break;\n                case 31: mensajeError = "Se ha producido una situación\n                    de error de E/S al intentar acceder al fichero XML del\n                    repositorio";\n                break;\n                case 40: mensajeError = "Se ha producido un error al\n                    intentar consultar la relación de titulares";\n            }\n        }\n    }\n}
```

```
break;
case 41: mensajeError = "Se ha producido un error al
intentar consultar el número de titulares existentes";
break;
case 50: mensajeError = "Se ha producido un error al
intentar insertar votación";
break;
case 51: mensajeError = "Se ha producido un error al
intentar consultar la relación de votaciones";
break;
case 60: mensajeError = "Se ha producido un error al
intentar insertar voto emitido";
break;
case 61: mensajeError = "Se ha producido un error al
intentar consultar la relación de votos emitidos en
una votación";
break;
}
}
else
{ if (excepcion instanceof IOException)
{
mensajeError = "Se ha producido una situación de error
de E/S";
}
else
{
mensajeError = excepcion.getMessage();
}
}
try {
contexto.setFechaHora(new java.util.Date());
new IncidenciasNegocio().escribirFichero(codigoError,
mensajeError, contexto);
} catch(IOException iOException) {
System.out.println("Se ha producido un error al
intentar escribir en fichero log de errores"); }
```

```
JOptionPane.showMessageDialog(null, mensajeError,  
"ERROR", JOptionPane.ERROR_MESSAGE);  
}  
}
```

## RepositorioNegocio

```
package negocio;  
import datos.RepositorioXML;  
public class RepositorioNegocio {  
public Object[] cargarRepositorio() throws Exception  
{  
return new RepositorioXML().cargar();  
}  
}
```

## TitularesNegocio

```
package negocio;  
import datos.ConexionBaseDatos;  
import datos.TitularesDatos;  
import encapsuladores.BaseDatos;  
import encapsuladores.Titular;  
import java.sql.Connection;  
import java.util.List;  
public class TitularesNegocio {  
public List<Titular> consultarTodos(BaseDatos  
baseDatos) throws Exception  
{  
Connection connection=null;  
ConexionBaseDatos conexionBaseDatos = new  
ConexionBaseDatos();  
List<Titular> listaCopropietarios = null;  
try {  
connection =  
conexionBaseDatos.abrirConexion(baseDatos);  
listaCopropietarios = new  
TitularesDatos().consultarTodos(connection);
```

```
        } catch (Exception excepcion)
        {
            throw excepcion;
        }
    finally
    {
        conexionBaseDatos.cerrarConexion(connection);
    }
    return listaCopropietarios;
}
public Integer consultarNumeroFilas(BaseDatos
baseDatos) throws Exception
{
    Connection connection=null;
    ConexionBaseDatos conexionBaseDatos = new
    ConexionBaseDatos();
    TitularesDatos copropietariosDatos = new
    TitularesDatos();
    Integer numFilas = null;
    try {
        connection =
        conexionBaseDatos.abrirConexion(baseDatos);
        numFilas =
        copropietariosDatos.consultarNumeroFilas(connection);
    } catch (Exception excepcion)
    {
        throw excepcion;
    }
    finally
    {
        conexionBaseDatos.cerrarConexion(connection);
    }
    return numFilas;
}
}
```

## VotacionesNegocio

```
package negocio;
import datos.ConexionBaseDatos;
import datos.VotacionesDatos;
import datos.VotacionesPDF;
import datos.VotosEmitidosDatos;
import encapsuladores.BaseDatos;
import encapsuladores.SistemaArchivos;
import encapsuladores.Votacion;
import encapsuladores.VotoEmitido;
import java.sql.Connection;
import java.util.List;
public class VotacionesNegocio {
    public void guardarVotacion(BaseDatos baseDatos,
        SistemaArchivos sistemaArchivos, Votacion votacion,
        Object datos[][][], double[] contadorVotos) throws
        Exception{
        Connection connection = null;
        ConexionBaseDatos conexionBaseDatos = new
        ConexionBaseDatos();
        try {
            connection =
            conexionBaseDatos.abrirConexion(baseDatos);
            connection.setAutoCommit(false);
            new VotacionesDatos().insertar(connection, votacion);
            new VotosEmitidosDatos().insertar(connection,
                votacion, datos);
            new VotacionesPDF().generarPDF(sistemaArchivos,
                votacion, contadorVotos);
            connection.commit();
        } catch (Exception excepcion)
        {
            connection.rollback();
            throw excepcion;
        }
        finally
        {
            conexionBaseDatos.cerrarConexion(connection);
        }
    }
}
```

```
}

}

public List<Votacion>
consultarTodasVotaciones(BaseDatos baseDatos) throws
Exception
{
Connection connection=null;
ConexionBaseDatos conexionBaseDatos = new
ConexionBaseDatos();
List<Votacion> listaVotaciones = null;
try {
connection =
conexionBaseDatos.abrirConexion(baseDatos);
listaVotaciones = new
VotacionesDatos().consultarTodas(connection);
} catch (Exception excepcion)
{
throw excepcion;
}
finally
{
conexionBaseDatos.cerrarConexion(connection);
}
return listaVotaciones;
}
public List<VotoEmitido>
consultarResultadoVotacion(BaseDatos baseDatos,
Votacion votacion) throws Exception
{
Connection connection=null;
ConexionBaseDatos conexionBaseDatos = new
ConexionBaseDatos();
List<VotoEmitido> listaVotosEmitidos = null;
try {
connection =
conexionBaseDatos.abrirConexion(baseDatos);
listaVotosEmitidos = new
```

```
VotosEmitidosDatos().consultarResultadoVotacion(connection, votacion);
} catch (Exception excepcion)
{
throw excepcion;
}
finally
{
conexionBaseDatos.cerrarConexion(connection);
}
return listaVotosEmitidos;
}
```

## IncidentesNegocio

```
package negocio;
import datos.IncidenciasDatos;
import encapsuladores.Contexto;
import java.io.IOException;
public class IncidentesNegocio {
public void escribirFichero(int codigoError, String mensajeError, Contexto contexto) throws IOException {
new
IncidenciasDatos().escribirEnFicheroIncidencias(codigo
Error, mensajeError, contexto);
}
}
```

## RepositorioXML

```
package datos;
import encapsuladores.BaseDatos;
import encapsuladores.SistemaArchivos;
import excepciones.GenericaExcepcion;
import java.io.IOException;
import java.util.Iterator;
import java.util.List;
```

```
import org.jdom2.Document;
import org.jdom2.Element;
import org.jdom2.JDOMException;
import org.jdom2.input.SAXBuilder;
public class RepositorioXML {
    public Object[] cargar() throws Exception {
        Object[] repositorio = new Object[2];
        BaseDatos baseDatos = null;
        try {
            Document document = new
            SAXBuilder().build("xml/repositorio.xml"); // xml es
            un directorio en el raiz del Proyecto
            Element raiz = document.getRootElement();
            // LEER BASE DE DATOS SELECCIONADA
            Element basesDeDatos =
            raiz.getChild("BASES_DE_DATOS");
            Element baseDatosSeleccionada =
            basesDeDatos.getChild("BASE_DATOS_SELECCIONADA");
            List listaXMLBD =
            basesDeDatos.getChildren("BASE_DATOS");
            boolean encontradaBD = false;
            Iterator iteratorBD = listaXMLBD.iterator();
            while (iteratorBD.hasNext() && !encontradaBD)
            {
                Element elementBD = (Element)iteratorBD.next();
                if
                (elementBD.getAttributeValue("nombre").compareTo(baseD
                    atosSeleccionada.getText()) == 0)
                { // ENCONTRADA BD SELECCIONADA Y CARGA XML EN OBJETO
                    ENCAPSULADOR
                    encontradaBD = true;
                    baseDatos = new BaseDatos();
                    baseDatos.setNombre(elementBD.getAttributeValue("nombr
                        e"));
                    baseDatos.setClassDriver(elementBD.getChild("CLASS_DRI
                        VER").getText());
                    baseDatos.setUrlConexion(elementBD.getChild("URL_CONEX
                        ")
                    )
                }
            }
        }
    }
}
```

```

    ION”).getText());
baseDatos.setUsuario(elementBD.getChild(“USUARIO”).getText());
baseDatos.setPassword(elementBD.getChild(“PASSWORD”).getText());
repositorio[0] = baseDatos;
}
}
// LEER SISTEMA ARCHIVOS
Element elementSistemaArchivos =
raiz.getChild(“SISTEMA_ARCHIVOS”);
SistemaArchivos sistemaArchivos = new
SistemaArchivos();
sistemaArchivos.setRutaCreacion(elementSistemaArchivos
.getChild(“RUTA_CREACION”).getText());
repositorio[1] = sistemaArchivos;
} catch(JDOMException excepcion)
{ throw new GenericaExcepcion(30); }
catch(IOException excepcion)
{ throw new GenericaExcepcion(31); }
return repositorio;
}
}

```

## ConexionBaseDatos

```

package datos;
import encapsuladores.BaseDatos;
import excepciones.GenericaExcepcion;
import java.sql.Connection;
import java.sql.SQLException;
public class ConexionBaseDatos {
public Connection abrirConexion(BaseDatos baseDatos)
throws Exception
{
Connection connection = null;
try {

```

```
Class.forName(baseDatos.getClassDriver());
connection =
java.sql.DriverManager.getConnection(baseDatos.getUrlConexion(),
baseDatos.getUsuario(),
baseDatos.getPassword());
} catch (SQLException excepcion) {
throw new GenericaExcepcion(20);
}
return connection;
}
public void cerrarConexion(Connection connection)
throws GenericaExcepcion
{
try {
if (connection!= null)
connection.close();
} catch (SQLException excepcion) {
throw new GenericaExcepcion(20);
}
}
```

## **TitularesDatos**

```
package datos;
import encapsuladores.Titular;
import excepciones.GenericaExcepcion;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
public class TitularesDatos {
public List<Titular> consultarTodos(Connection
connection) throws Exception
{
```

```
List<Titular> listaCopropietarios = new ArrayList();
ResultSet resultSet = null;
Statement statement = null;
try {
String sql = "SELECT * FROM titulares ORDER BY
codigo";
statement = connection.createStatement();
resultSet = statement.executeQuery(sql);
while (resultSet.next()) {
Titular copropietario = new Titular();
copropietario.setCodigo(resultSet.getString(1));
copropietario.setNombre(resultSet.getString(2));
copropietario.setCuotaParticipacion(new
Double(resultSet.getDouble(3)));
listaCopropietarios.add(copropietario);
}
} catch (SQLException excepcion) {
throw new GenericaExcepcion(40);
} finally
{
if (resultSet != null) resultSet.close();
if (statement != null) statement.close();
}
return listaCopropietarios;
}
public Integer consultarNumeroFilas(Connection
connection) throws Exception
{
Integer numFilas = null;
ResultSet resultSet = null;
Statement statement = null;
String sql = "SELECT COUNT(*) FROM titulares";
try {
statement = connection.createStatement();
resultSet = statement.executeQuery(sql);
if (resultSet.next()) {
numFilas = new Integer(resultSet.getInt(1));
}
```

```
}

} catch (SQLException excepcion) {
throw new GenericaExcepcion(41);
} finally
{
if (resultSet != null) resultSet.close();
if (statement != null) statement.close();
}
return numFilas;
}
}
```

## VotacionesDatos

```
package datos;
import encapsuladores.Votacion;
import excepciones.GenericaExcepcion;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
public class VotacionesDatos {
public void insertar(Connection connection, Votacion
votacion) throws Exception
{ PreparedStatement preparedStatement = null;
try {
String sql = “INSERT INTO votaciones VALUES(?,?)”;
preparedStatement = connection.prepareStatement(sql);
preparedStatement.setString(1,
votacion.getIdVotacion());
preparedStatement.setString(2,
votacion.getTemaVotado());
preparedStatement.executeUpdate();
} catch (SQLException excepcion) {
```

```
throw new GenericaExpcion(50);
} finally
{
if (preparedStatement != null)
preparedStatement.close();
}
}

public List<Votacion> consultarTodas(Connection
connection) throws Exception
{
List<Votacion> listaVotaciones = new ArrayList();
ResultSet resultSet = null;
Statement statement = null;
try {
String sql = "SELECT * FROM votaciones ORDER BY
tema_votado, id_votacion";
statement = connection.createStatement();
resultSet = statement.executeQuery(sql);
while (resultSet.next()) {
Votacion votacion = new Votacion();
votacion.setIdVotacion(resultSet.getString(1));
votacion.setTemaVotado(resultSet.getString(2));
listaVotaciones.add(votacion);
}
} catch (SQLException excepcion) {
throw new GenericaExpcion(51);
} finally
{
if (resultSet != null) resultSet.close();
if (statement != null) statement.close();
}
return listaVotaciones;
}
}
```

## VotosEmitidosDatos

```
package datos;
import encapsuladores.Votacion;
import encapsuladores.VotoEmitido;
import excepciones.GenericaExcepcion;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
public class VotosEmitidosDatos {
    public void insertar(Connection connection, Votacion votacion, Object datos[][])
        throws Exception {
        PreparedStatement preparedStatement = null;
        try {
            String sql = "INSERT INTO votos_emitidos VALUES(?,?,?,?,?,?)";
            preparedStatement = connection.prepareStatement(sql);
            for (int i=0; i<datos.length; i++) {
                preparedStatement.setString(1, (String)datos[i][0]);
                preparedStatement.setString(2, votacion.getIdVotacion());
                for (int k=3; k<=6; k++) {
                    byte voto = 0;
                    if (((Boolean)datos[i][k]).booleanValue())
                        voto = 1;
                    preparedStatement.setByte(k, voto);
                }
                preparedStatement.executeUpdate();
            }
        } catch (SQLException excepcion) {
            throw new GenericaExcepcion(60);
        } finally {
            if (preparedStatement != null)
```

```
preparedStatement.close();
}
}
public List<VotoEmitido>
consultarResultadoVotacion(Connection connection,
Votacion votacion) throws Exception
{
List<VotoEmitido> listaVotosEmitidos = new
ArrayList();
ResultSet resultSet = null;
PreparedStatement preparedStatement = null;
try {
String sql = "SELECT opcion_ausente,
opcion_abstencion, opcion_afirmativo,opcion_negativo
FROM votos_emitidos WHERE id_votacion=? ORDER BY
codigo_titular";
preparedStatement = connection.prepareStatement(sql);
preparedStatement.setString(1,
votacion.getIdVotacion());
resultSet = preparedStatement.executeQuery();
while (resultSet.next()) {
VotoEmitido votoEmitido = new VotoEmitido();
for (int i=1; i<=4; i++)
{
byte opcionVotacionBD = resultSet.getByte(i);
Boolean opcionVotacion;
if (opcionVotacionBD == 1)
votoEmitido.setComponenteOpcionesVotacion(new
Boolean(true), i-1);
else
votoEmitido.setComponenteOpcionesVotacion(new
Boolean(false), i-1);
}
listaVotosEmitidos.add(votoEmitido);
}
} catch (SQLException excepcion) {
throw new GenericaExcepcion(61);
```

```

    } finally
    {
        if (resultSet != null) resultSet.close();
        if (preparedStatement != null)
            preparedStatement.close();
    }
    return listaVotosEmitidos;
}
}

```

## IncidenciasDatos

```

package datos;
import encapsuladores.Contexto;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.text.SimpleDateFormat;
public class IncidenciasDatos {
    public void escribirEnFicheroIncidencias(int
        codigoError, String mensajeError, Contexto contexto)
        throws IOException {
        File file = new File("log/log.txt");
        if (!file.exists())
            file.createNewFile();
        FileWriter fileWriter = new FileWriter(file, true);
        PrintWriter printWriter = new PrintWriter(fileWriter);
        printWriter.println(new SimpleDateFormat("yyyy-MM-dd
            HH:mm:ss").format(contexto.getFechaHora()) +" "+
            String.format("%-20s",contexto.getIdentificadorUsuario
                ()) + String.format("%-18s",contexto.getIpCliente()) +
            String.format("%5s",codigoError)+ " "
            "+String.format("%-110s",mensajeError));
        if (printWriter != null)
            printWriter.close();
        if (fileWriter != null)

```

```
fileWriter.close();
}
}
```

## VotacionesPDF

```
package datos;
import encapsuladores.SistemaArchivos;
import encapsuladores.Votacion;
import java.io.IOException;
import java.text.DecimalFormat;
import java.text.DecimalFormatSymbols;
import org.apache.pdfbox.pdmodel.PDDocument;
import org.apache.pdfbox.pdmodel.PDPage;
import org.apache.pdfbox.pdmodel.PDPageContentStream;
import org.apache.pdfbox.pdmodel.font.PDFont;
import org.apache.pdfbox.pdmodel.font.PDType1Font;
public class VotacionesPDF {
public void generarPDF(SistemaArchivos
sistemaArchivos, Votacion votacion, double[]
contadorVotos) throws IOException{
String nombreFichero =
sistemaArchivos.getRutaCreacion()+"Votacion"+votacion.
getIdVotacion().substring(0,
4)+votacion.getIdVotacion().substring(5,
7)+votacion.getIdVotacion().substring(8,
10)+votacion.getIdVotacion().substring(11,
13)+votacion.getIdVotacion().substring(14,
16)+votacion.getIdVotacion().substring(17)+".pdf";
PDDocument pDDocument = new PDDocument();
PDPage pDPage = new PDPage();
pDDocument.addPage(pDPage);
PDPageContentStream pDPageContentStream = new
PDPageContentStream(pDDocument, pDPage);
escribirCuerpo(pDDocument, pDPageContentStream,
votacion, contadorVotos);
pDPageContentStream.close();
```

```
pDDocument.save(nombreFichero);
pDDocument.close();
}
private void escribirCuerpo(PDDocument pDDocument,
PDPageContentStream pDPageContentStream, Votacion
votacion, double[] contadorVotos) throws IOException{
DecimalFormatSymbols decimalFormatSymbols = new
DecimalFormatSymbols();
DecimalFormat decimalFormat = new
DecimalFormat("##0.000", decimalFormatSymbols);
escribirTexto(pDPageContentStream,
PDType1Font.TIMES_BOLD_ITALIC, 12, 50, 690, "Comunidad
de Propietarios Urbanización \"MONTES DE IBERIA\"");
escribirTexto(pDPageContentStream,
PDType1Font.TIMES_ITALIC, 12, 50, 670, "MADRID");
escribirLinea(pDPageContentStream, 30, 570, 580, 570);
escribirTexto(pDPageContentStream,
PDType1Font.TIMES_BOLD, 10, 180, 550, "VOTACIÓN POR
CUOTA DE PARTICIPACIÓN");
escribirLinea(pDPageContentStream, 30, 540, 580, 540);
escribirTexto(pDPageContentStream,
PDType1Font.TIMES_BOLD, 8, 50, 500, "Tema votado: ");
escribirTexto(pDPageContentStream,
PDType1Font.TIMES_ROMAN, 8, 130, 500,
votacion.getTemaVotado());
escribirTexto(pDPageContentStream,
PDType1Font.TIMES_BOLD, 8, 50, 480, "Fecha registro
:");
escribirTexto(pDPageContentStream,
PDType1Font.TIMES_ROMAN, 8, 130, 480,
votacion.getIdVotacion());
escribirTexto(pDPageContentStream,
PDType1Font.TIMES_BOLD, 8, 50, 460, "Total cuotas de
votos ausentes: ");
escribirTexto(pDPageContentStream,
PDType1Font.COURIER, 8, 170, 460,
String.format("%10s",
```

```
decimalFormat.format(contadorVotos[0])));  
escribirTexto(pDPageContentStream,  
PDTipo1Font.TIMES_BOLD, 8, 50, 440, "Total cuotas de  
abstenciones:");  
escribirTexto(pDPageContentStream,  
PDTipo1Font.COURIER, 8, 170, 440,  
String.format("%10s",  
decimalFormat.format(contadorVotos[1])));  
escribirTexto(pDPageContentStream,  
PDTipo1Font.TIMES_BOLD, 8, 50, 420, "Total cuotas de  
votos a favor:");  
escribirTexto(pDPageContentStream,  
PDTipo1Font.COURIER, 8, 170, 420,  
String.format("%10s",  
decimalFormat.format(contadorVotos[2])));  
escribirTexto(pDPageContentStream,  
PDTipo1Font.TIMES_BOLD, 8, 50, 400, "Total cuotas de  
votos en contra:");  
escribirTexto(pDPageContentStream,  
PDTipo1Font.COURIER, 8, 170, 400,  
String.format("%10s",  
decimalFormat.format(contadorVotos[3])));  
}  
private void escribirTexto(PDPageContentStream  
pDPageContentStream, PDFont pdFont, float sizeFuente,  
float inicioH, float inicioV, String texto) throws  
IOException {  
pDPageContentStream.beginText();  
pDPageContentStream.setFont(pdFont, sizeFuente);  
pDPageContentStream.newLineAtOffset(inicioH, inicioV);  
pDPageContentStream.showText(texto);  
pDPageContentStream.endText();  
}  
private void escribirLinea(PDPageContentStream  
pDPageContentStream, float inicioH, float inicioV,  
float finH, float finV) throws IOException {  
pDPageContentStream.moveTo(inicioH, inicioV);
```

```
pDPageContentStream.lineTo(finH, finV);
pDPageContentStream.stroke();
}
}
```

## SistemaArchivos

```
package encapsuladores;
public class SistemaArchivos {
private String rutaCreacion;
public String getRutaCreacion() {
return rutaCreacion;
}
public void setRutaCreacion(String rutaCreacion) {
this.rutaCreacion = rutaCreacion;
}
}
```

## Contexto

```
package encapsuladores;
import java.util.Date;
public class Contexto {
private String identificadorUsuario;
private Date fechaHora;
private String ipCliente;
public Contexto(String identificadorUsuario, String
ipCliente) {
this.identificadorUsuario = identificadorUsuario;
this.ipCliente = ipCliente;
fechaHora = new Date();
}
public String getIdentificadorUsuario() {
return identificadorUsuario;
}
public void setIdentificadorUsuario(String
identificadorUsuario) {
this.identificadorUsuario = identificadorUsuario;
```

```
}

public Date getFechaHora() {
    return fechaHora;
}

public void setFechaHora(Date fechaHora) {
    this.fechaHora = fechaHora;
}

public String getIpCliente() {
    return ipCliente;
}

public void setIpCliente(String ipCliente) {
    this.ipCliente = ipCliente;
}
```

## BaseDatos

```
package encapsuladores;
public class BaseDatos {
    private String nombre;
    private String classDriver;
    private String urlConexion;
    private String usuario;
    private String password;
    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getClassDriver() {
        return classDriver;
    }

    public void setClassDriver(String classDriver) {
        this.classDriver = classDriver;
    }

    public String getUrlConexion() {
```

```
return urlConexion;
}
public void setUrlConexion(String urlConexion) {
this.urlConexion = urlConexion;
}
public String getUsuario() {
return usuario;
}
public void setUsuario(String usuario) {
this.usuario = usuario;
}
public String getPassword() {
return password;
}
public void setPassword(String password) {
this.password = password;
}
}
```

## Titular

```
package encapsuladores;
public class Titular {
private String codigo;
private String nombre;
private double cuotaParticipacion;
public String getCodigo() {
return codigo;
}
public void setCodigo(String codigo) {
this.codigo = codigo;
}
public String getNombre() {
return nombre;
}
public void setNombre(String nombre) {
this.nombre = nombre;
}
```

```
}

public double getCuotaParticipacion() {
    return cuotaParticipacion;
}

public void setCuotaParticipacion(double cuotaParticipacion) {
    this.cuotaParticipacion = cuotaParticipacion;
}

}
```

## Votacion

```
package encapsuladores;
public class Votacion {
    private String idVotacion;
    private String temaVotado;
    public String getIdVotacion() {
        return idVotacion;
    }
    public void setIdVotacion(String idVotacion) {
        this.idVotacion = idVotacion;
    }
    public String getTemaVotado() {
        return temaVotado;
    }
    public void setTemaVotado(String temaVotado) {
        this.temaVotado = temaVotado;
    }
}
```

## VotoEmitido

```
package encapsuladores;
public class VotoEmitido {
    private Boolean[] opcionesVotacion = new Boolean[4];
    public Boolean[] getOpcionesVotacion() {
        return opcionesVotacion;
    }
}
```

```
public void setOpcionesVotacion(Boolean[] opcionesVotacion) {
    this.opcionesVotacion = opcionesVotacion;
}
public Boolean getComponenteOpcionesVotacion(int componente) {
    return opcionesVotacion[componente];
}
public void setComponenteOpcionesVotacion(Boolean opcionVotacion, int componente) {
    this.opcionesVotacion[componente] = opcionVotacion;
}
}
```

## GenericaExcepcion

```
package excepciones;
public class GenericaExcepcion extends Exception{
private int codigoError;
public GenericaExcepcion (int codigoError){
this.codigoError = codigoError;
}
public int getCodigoError(){
return codigoError;
}
}
```

Mostramos a continuación el contenido del fichero repositorio.xml, presente en el directorio xml del proyecto, y que actúa como repositorio centralizado de parámetros relevantes de la aplicación, tal y como hemos venido haciendo en ejercicios anteriores de este libro:

## Contenido del fichero repositorio.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<REPOSITORIO>
```

```
<BASES_DE_DATOS>
<BASE_DATOS_SELECCIONADA>MySQL</BASE_DATOS_SELECCIONADA>
<BASE_DATOS nombre="Oracle">
<CLASS_DRIVER>oracle.jdbc.driver.OracleDriver</CLASS_DRIVER>
<URL_CONEXION>jdbc:oracle:thin:@localhost:1521:ORCL</URL_CONEXION>
<USUARIO>scott</USUARIO>
<PASSWORD>tiger</PASSWORD>
</BASE_DATOS>
<BASE_DATOS nombre="MySQL">
<CLASS_DRIVER>com.mysql.jdbc.Driver</CLASS_DRIVER>
<URL_CONEXION>jdbc:mysql://localhost:3306/votaciones</URL_CONEXION>
<USUARIO>root</USUARIO>
<PASSWORD></PASSWORD>
</BASE_DATOS>
</BASES_DE_DATOS>
<SISTEMA_ARCHIVOS>
<RUTA_CREACION>D:\\PDFs_creados\\</RUTA_CREACION>
</SISTEMA_ARCHIVOS>
</REPOSITORIO>
```

## Instrucciones SQL

Procedemos a continuación a enumerar la relación de instrucciones SQL alojadas en esta aplicación, concretamente en métodos de las clases correspondientes de la capa de datos:

1. Consulta que devuelve la relación de titulares:

```
SELECT * FROM titulares ORDER BY código
```

2. Consulta que devuelve el número de titulares:

```
SELECT COUNT(*) FROM titulares
```

3. Insertar votación:

```
INSERT INTO votaciones VALUES(?,?)
```

4. Consulta que devuelve la relación de votaciones:

```
SELECT * FROM votaciones ORDER BY tema_votado,  
id_votacion
```

5. Insertar voto emitido:

```
INSERT INTO votos_emitidos VALUES(?,?,?,?,?,?)
```

6. Consulta que devuelve todos los votos emitidos en una votación:

```
SELECT opcion_ausente, opcion_abstencion,  
opcion_afirmativo,opcion_negativo  
FROM votos_emitidos  
WHERE id_votacion=?  
ORDER BY codigo_titular
```

Esta aplicación responde a una utilidad enmarcada en el ámbito de la gestión de sociedades, comunidades de propietarios, comunidades de vecinos, etc. En estos escenarios, la toma de decisiones corporativas, deben de someterse siempre a votación. En muchos de estos ámbitos corporativos, el voto de una persona no se corresponde a la unidad, es decir, cada persona no tiene un voto como ocurre en las elecciones de contexto político, sino que el peso específico del voto de una persona depende de la cuota de participación que esa persona tiene en la empresa, sociedad, comunidad de vecinos. Es fácil entender que, en empresas o sociedades, la cuota de participación dependa totalmente de las acciones que dicha persona posea en la empresa o sociedad. Y en escenarios asociados a comunidades de propietarios o vecinos, los estatutos de dicha comunidad pueden determinar la cuota de participación en base a un cálculo de diferentes parámetros, o de la combinación de algunos de ellos aplicando los porcentajes

pertinentes. Dichos parámetros pueden basarse en superficie en metros cuadrados de la propiedad, condición de estar edificada o no (determinante para el uso o desgaste que cada propiedad ejerce sobre los elementos comunes a mantener), metros lineales de fachada, etc. El uso de la utilidad que representa esta aplicación permite:

- Disponer automáticamente y en tiempo real del resultado de una votación conforme cada propietario va manifestando el sentido de su voto.
- Registrar el resultado de una votación ante la posible necesidad de tener que recuperarla.
- Obtener un documento PDF con el resultado de cada votación.

Al ejecutar la aplicación se muestra una ventana en que aparece un *JTable* con la relación de los nombres de las personas con derecho a participar en la votación, y el porcentaje correspondiente a su cuota de participación:

Votación propuesta

Código	Nombre	Čuota participación	AUSENTE	ABSTENCIÓN	SI	NO
001	ALARCON LAGUNA, CONCEPCION	1,143	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
002	ALPUENTE CARRASCO, VICENTA	1,48	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
003	APARISI SERRANO, FELIX	0,937	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
004	BARBERA SORIANO, RAMON	1,238	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
005	BLANCO ALAMAN, PILAR	1,012	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
006	BOLUMAR GASCO, ESTRELLA	1,239	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
007	BUIENAVENTURA SANCHEZ, ANTONIO	1,065	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
008	GALATRIVA GARCIA, TOMAS	0,871	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
009	CALVO SERRANO, ANGELES	1,52	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
010	CARABELLA CALVO, BLANCA	1,135	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
011	CARBO NAVARRO, SANTIAGO	1,305	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
012	CARRETERO RUS, ANTONIO	0,869	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
013	CATALA PARDÓ, CECILIA	1,315	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
014	DELGADO MARTIN, FRANCISCO MIGUEL	1,873	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
015	ESTEBAN MARTINEZ, MANUEL	0,968	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
016	FERRERO BOTELLA, FEDERICO	0,732	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
017	FONTALES ROMEU, INMACULADA	1,354	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
018	GALIANA MONSALVE, AGUSTIN	1,982	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
019	GALLEGO GONZALEZ, YOLANDA	2,105	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
020	GALLEGO LASTRA, FERNANDO	2,098	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
021	GARCIA JIMENEZ, DOLORES	1,129	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
022	GIMENEZ OLIVARES, EDUARDO	0,879	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
023	GOMEZ MARTINEZ, AURORA	1,93	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
024	GUTIERREZ SOTO, ALVARO	1,459	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
025	HERIRERA BALURTE, OSCAR	1,38	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
026	HUERTA CASTELLO, ARTURO	2,39	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
027	IBARRA PARDO, LUIS	1,814	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
028	ISERTE PARDO, BEATRIZ	1,59	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
029	JIMENEZ JIMENEZ, LUCAS	2,349	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
030	JULVE REVERTER, SANTIAGO	1,65	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

0,000      0,000      0,000      0,000

Tema votado:

Encontrándonos con la posibilidad de realizar tres posibles actuaciones.

## ABRIR NUEVA VOTACIÓN

Para ello pulsamos el botón correspondiente. Observaremos que por defecto aparecen marcadas todas las casillas de verificación correspondientes a la columna AUSENTE. Procedería que en este momento le diésemos nombre al tema que se va a someter a votación, con perspectiva de que una vez finalizada la votación, procedamos a registrarla pulsando el botón correspondiente:

Votación propuesta

Código	Nombre	Cuota participación	AUSENTE	ABSTENCION	SI	NO
001	ALARCON LAGUNA, CONCEPCION	1,143	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
002	ALPUENTE CARRASCO, VICENTA	1,46	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
003	APARISI SERRANO, FELIX	0,937	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
004	BARBERA SORIANO, RAMON	1,238	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
005	BLANCO ALAMAN, PILAR	1,012	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
006	BOLUMAR GASCO, ESTRELLA	1,239	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
007	BUENAVENTURA SANCHEZ, ANTONIO	1,085	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
008	CALATRAVA CARBONELL, TOMAS	0,871	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
009	CALVO SERRANO, ANGELES	1,52	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
010	CARABELLA CALVO, BLANCA	1,135	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
011	CARBO NAVARRO, SANTIAGO	1,305	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
012	CARRETERO RUS, ANTONIO	0,869	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
013	CATALA PARDO, CECILIA	1,315	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
014	DELGADO MARTIN, FRANCISCO MIGUEL	1,873	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
015	ESTEBAN MARTINEZ, MANUEL	0,968	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
016	FERRERO BOTELLA, FEDERICO	0,732	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
017	FONTALES ROMEU, INMACULADA	1,354	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
018	GALIANA MONSALVE, AGUSTIN	1,982	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
019	GALLEGO GONZALEZ, YOLANDA	2,105	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
020	GALLEGO LASTRA, FERNANDO	2,098	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
021	GARCIA JIMENEZ, DOLORES	1,129	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
022	GIMENEZ OLIVARES, EDUARDO	0,879	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
023	GOMEZ MARTINEZ, AURORA	1,93	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
024	GUTIERREZ SOTO, ALVARO	1,459	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
025	HERRERA BALUARTE, OSCAR	1,38	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
026	HUERTA CASTIELLO, ARTURO	2,39	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
027	IBARRA PARDO, LUIS	1,914	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
028	ISERTE PARDO, BEATRIZ	1,59	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
029	JIMENEZ JIMENEZ, LUCAS	2,349	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
030	JULVE REVERTER, SANTIAGO	1,65	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

100,000      0,000      0,000      0,000

Tema votado:

A partir de este instante, nos encontramos ya en condiciones de iniciar la votación. A medida cada una de las personas participantes vayan manifestando su opción (o bien se encuentran ausentes), vamos marcando la casilla de verificación correspondiente, produciéndose automáticamente dos actuaciones:

- Se desmarca la casilla de verificación correspondiente a esa persona que estuviese previamente marcada.
- Se recalculan al instante los porcentajes totales, visualizándose al pie de la columna correspondiente a cada opción.

Procedamos con la primera persona, imaginándonos que vota afirmativamente:

Votación propuesta

Código	Nombre	Cuota participación	AUSENTE	ABSTENCION	SI	NO
001	ALARCON LAGUNA, CONCEPCION	1,143	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
002	ALPUENTE CARRASCO, VICENTA	1,48	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
003	APARISI SERRANO, FELIX	0,937	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
004	BARBERA SORIANO, RAMON	1,238	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
005	BLANCO ALAMAN, PILAR	1,012	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
006	BOLUJAR GASCO, ESTRELLA	1,239	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
007	BUENAVENTURA SANCHEZ, ANTONIO	1,085	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
008	CALATRAVA CARBONELL, TOMAS	0,871	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
009	CALVO SERRANO, ANGELES	1,52	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
010	CARABELLA CALVO, BLANCA	1,136	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
011	CARBO NAVARRO, SANTIAGO	1,305	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
012	CARRETERO RUS, ANTONIO	0,869	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
013	CATALA PARDO, CECILIA	1,315	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
014	DELGADO MARTIN, FRANCISCO MIGUEL	1,873	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
015	ESTEBAN MARTINEZ, MANUEL	0,968	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
016	FERRERO BOTELLA, FEDERICO	0,732	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
017	FONTALES ROMEU, INMACULADA	1,354	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
018	GALIANA MONSALVE, AGUSTIN	1,082	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
019	GALLEGOS GONZALEZ, YOLANDA	2,105	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
020	GALLEGO LASTRA, FERNANDO	2,098	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
021	GARCIA JIMENEZ, DOLORES	1,129	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
022	GIMENEZ OLIVARES, EDUARDO	0,879	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
023	GOMEZ MARTINEZ, AURORA	1,93	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
024	GUTIERREZ SOTO, ALVARO	1,459	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
025	HERRERA BALUARTE, OSCAR	1,38	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
026	HUERTA CASTELLO, ARTURO	2,39	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
027	IBARRA PARDO, LUIS	1,814	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
028	ISERTE PARDO, BEATRIZ	1,59	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
029	JIMENEZ JIMENEZ, LUCAS	2,349	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
030	JULVE REVERTER, SANTIAGO	1,65	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

98,857      0,000      1,143      0,000

Tema votado: CONSTRUCCIÓN PISTA DE TENIS

Así proseguiríamos con el resto de personas que aparecen en el *JTable*, con posibilidad, por supuesto, de rectificar en cualquier momento y revertir un voto. Una vez se dé por finalizada la votación, ya podemos proceder a:

## REGISTRAR VOTACIÓN

Pulsando el botón **Guarda votación**.

Votación propuesta

Código	Nombre	Cuota participación	AUSENTE	ABSTENCION	SI	NO
033	LUJAN TORREGROSA, RAMON	1,27				
034	MAEZTU AMORES, JUAN	2,983	✓			
035	MARTINEZ CLIMENT, CESAR	1,579			✓	
036	MERLOS PASCUAL, ISABEL	1,809			✓	
037	MINGUEZ FONTALES, SERGIO	0,928			✓	
038	MONRRABAL ARIÑO, SANTIAGO	0,835			✓	
039	MORENO CASTILLO, MARCOS	2,558			✓	
040	MORILLAS SALVADOR, ANGEL	1,207			✓	
041	MUNOZ SANCHIS, VICTOR	0,893	✓			
042	ORDINAS SUAREZ, PEDRO	2,099		✓		
043	ORIOLA CASTELLO, ANA MARIA	1,901				
044	OSUNA PORTOLEZ, EUGENIO	2,254			✓	
045	PEREZ CRESPO, GONZALO	0,908			✓	
046	PEREZ RAMON, EUGENIO	0,945			✓	
047	PERIS ALBELDA, ALBERTO	1,703				
048	RAMIREZ SANCHEZ, ROSARIO	1,308			✓	
049	RAMS SANTAMARIA, GLORIA	0,85				
050	RAUSSELL MARTIN, VIRGINIA	1,589				
051	RODRIGO QUEVEDO, JULIAN	0,98				
052	RODRIGUEZ SANTOS, VALERIO	0,802				
053	ROIG GUILLOT, MARIA LUISA	2,19				
054	ROMERO TAMARIT, PAULA	1,578				
055	SANCHEZ PUERTAS, JOSE	1,759	✓			
056	SOLIVAS VALIENTE, MIGUEL ANGEL	0,845				
057	SORIANO LLOPIS, EMILIO	2,015				
058	SORIANO SAINZ, JOSE MANUEL	1,798				
069	TEJEDOR LAGUNA, VICENTE	1,89				
060	TELLO JOVER, TERESA	2,154	✓			
061	UMBERT MORENO, FELISA	1,35				
062	VELA ASENSI, SIXTO	2,057				

9,309      14,048      63,972      12,671

Carga votación      2020-02-16 20:40:01 CONSTRUCCION PISTA DE TENIS

Nueva votación      Tema votado CONSTRUCCIÓN PISTA DE TENIS

Guarda votación

Observamos que se añade al *JComboBox*, pasando a engrosar la relación de todos los temas votados registrados en la Base de Datos, la descripción que le hemos dado al tema votado, precedido del momento del tiempo en que se ha pulsado el botón. Simultáneamente, en la ruta a tal efecto establecida en fichero XML que actúa como repositorio centralizado, se ha generado un fichero PDF que plasma un resumen de la votación que acabamos de registrar, cuya visualización presentamos a continuación:

*Comunidad de Propietarios Urbanización "MONTES DE IBERIA"  
MADRID*

---

**VOTACIÓN POR CUOTA DE PARTICIPACIÓN**

---

Tema votado:	CONSTRUCCIÓN PISTA DE TENIS
Fecha registro :	2020-02-16 20:40:01
Total cuotas de votos ausentes:	9,309
Total cuotas de abstenciones:	14,048
Total cuotas de votos a favor:	63,972
Total cuotas de votos en contra:	12,671

## LEER RESULTADO DE VOTACIÓN PREVIAMENTE REGISTRADO EN LA BASE DE DATOS

Para ello, seleccionamos previamente en el *JComboBox*, en el que aparecen todas las votaciones registradas, aquella cuyo resultado pretendemos volver a visualizar en el *JTable*. A continuación, pulsamos el botón **Carga votación**, y el resultado de la votación pretendida vuelve a aparecer en el *JTable*, proceso cuya captura de pantalla no vamos a reproducir en pro de minimizar la extensión del libro.

Abordemos a continuación los aspectos meramente técnicos de esta aplicación ejemplo. Accesorialmente, vuelven a concurrir aquí aspectos ya tratados y abordados con anterioridad relacionados con la utilización de componentes de interfaz gráfica como el *JButton*, el *JComboBox*, y el *JTextField*. El *JTable* es un componente que aparece novedosamente en este ejemplo. Se trata de un componente infrautilizado. Suele utilizarse

habitualmente como elemento de salida de datos. Con esa proyección de componente de sólo salida, lo único que lo diferencia de otros componentes como el *JList*, el *JComboBox*, incluso el *JSpinner*, es que ofrece una perspectiva de los datos a dos dimensiones. En cambio, el verdadero potencial del *JTable* reside en la posibilidad de que también puede actuar como elemento de entrada de datos, diferencia fundamental en relación a todos los demás componentes enumerados. Ahora bien, explotar la vertiente del *JTable* como elemento de entrada, no es algo trivial, sino que requiere de un conocimiento profundo de los *Listeners*, y del *Model* asociados. Introducir al lector en los mecanismos inherentes a la utilización del *JTable* como componente que permite actividad interactiva, constituye el principal objetivo de esta aplicación ejemplo, que se verá culminado con el tratamiento que se le proporcionará a los dos *JTable* que integran la aplicación **GestionLibros**.

Nos centraremos a continuación en el estudio en detalle de los tres tipos de componentes mencionados, que aparecen novedosamente en esta aplicación, y que están íntimamente interrelacionados. Nos estamos refiriendo al *JTable*, al *Model*, y a los *Listeners* asociados que aparecen en este ejemplo.

## **JTable**

Este componente permite actividad interactiva con el usuario, circunstancia que le aplicamos en el presente ejemplo. Al igual que en casos anteriores, aparece acompañado de otros componentes:

- *JScrollPane*, que nos aporta una barra de desplazamiento auxiliar para acceder a la visualización de todas las filas contenidas en el *Model*, si el número de éstas supera las

que se pueden presentar en función del tamaño del marco que se le establezca. Se produce el acoplamiento en:

```
componentes.setJScrollPaneTabla(new  
JScrollPane(componentes.getjTable()));
```

- Un Model, en este caso por herencia de *AbstractTableModel*:

```
class ModeloDatos extends AbstractTableModel
```

que representa, se encarga del almacenamiento y permite la gestión de los datos que el componente se limita a visualizar, o a recoger en su interacción con el usuario. Lo asociamos al *JTable* como parámetro del constructor:

```
componentes.setjTable(new  
JTable(componentes.getModeloDatos()));
```

Podemos establecer una configuración individualizada asociada a cada columna, en el ejemplo, solamente hemos configurado el ancho. Para ello, instanciamos un array de *TableColumn*:

```
TableColumn columna[] = new TableColumn[7];
```

y procedemos a configurar cada componente, obteniendo previamente cada uno de ellos desde el *JTable*. Mostramos como actuamos con el primero de ellos:

```
columna[0] =  
componentes.getjTable().getColumnModel().getColumn(0);  
columna[0].setPreferredWidth(30);
```

Procedamos a continuación a estudiar en detalle el Model y el Listener con que se registra, que son donde se concentra toda la carga de la gestión de los datos del componente.

## Model

Declaramos el atributo:

```
private Object datos[][];
```

array de dos dimensiones dado que es el que debe almacenar los datos que se visualizan en el *JTable*, cuya disposición se contempla organizada en filas y columnas. Cada componente es un *Object* para proporcionar carácter genérico a objetos que pueden responder a clases diferentes en función de la columna. La concreción de la clase para cada columna se define en el momento de la asignación de los datos a los componentes del array. Lo podemos observar en el método:

```
private void inicializarFilas(int filaInicial, int
filaFinal) {
for (int i=filaInicial; i<=filaFinal; i++)
{ datos[i][0] = "";
datos[i][1] = "";
datos[i][2] = new Double(0);
datos[i][3] = new Boolean(false);
datos[i][4] = new Boolean(false);
datos[i][5] = new Boolean(false);
datos[i][6] = new Boolean(false);
}
}
```

El cometido de este método, invocado desde el constructor, dispone de un nombre lo suficientemente significativo como para que no haya necesidad de dotarlo de explicaciones accesorias.

Observaremos por las capturas de pantalla que para los datos de aquellas columnas a que se asignan objetos *Boolean*, se presenta para su edición una casilla de verificación. En la aplicación **GestionLibros** mostraremos cómo asociar la edición de un dato a la selección de un *JComboBox* asociado a la columna.

Siguiendo el recorrido por el Model, nos encontramos los métodos:

- cargarTitulares()
- cargarVotacion()
- inicializarVotacion()

cuyos nombres siguen siendo lo suficientemente significativos como para permitirnos obviar aclaraciones sobre su cometido. El primero de ellos es invocado desde el constructor del *JFrame*. Los dos siguientes son invocados en función de la interacción con el usuario mediante los *JButton* de la ventana. La ejecución de todos ellos supone modificar los datos de algunas columnas en todas las filas. La operación básica con que ello se consigue es almacenar en los componentes pertinentes del array *Object datos[][]* la información que se pretende visualizar en el *JTable*. En los dos primeros métodos, dichos datos le son transferidos al método mediante un *List*, y proceden del resultado de consultas SQL. Después de proceder a la modificación de los datos, la acción final que nos encontramos en todos ellos es:

```
componentes.getjTable().repaint();
```

El método

```
public Object[][] getDatos() {  
    return datos;  
}
```

constituye el método “getter” del array que da soporte a los datos almacenados, para dejarlo accesible desde otras clases de la aplicación para su manipulación, concretamente desde **GestorEventos**, clase que concentra todas las escuchas, tal y

como venimos haciendo en muchos ejercicios de esta parte del libro.

La presencia del resto de los métodos viene determinada por la herencia a que nos hemos sometido de *AbstractTableModel*, y esta clase abstracta implementa, a su vez, la *interface TableModel*. Mediante el código que implementemos en los mismos, estamos definiendo o sobrescribiendo, con unas actuaciones que determinan el comportamiento del *JTable* y su asociación con el Model. De la observación de los mismos, deducimos que la asociación entre el Model y el array *Object datos[][]*, viene definida por el código que implementamos en algunos de ellos. El comportamiento interactivo del *JTable* viene determinado por las escuchas de eventos que apliquemos, tanto del *JTable*, como del Model, aspecto que trataremos posteriormente, y del código que implementemos en los siguientes métodos:

```
public void setValueAt(Object valor, int fila, int
columna) {
    datos[fila][columna] = valor;
    // fireTableCellUpdated(fila, columna);
    componentes.setFilaActualizada(fila);
    componentes.setColumnaActualizada(columna);
    fireTableDataChanged();
}
```

Este método se ejecuta automáticamente cuando finaliza la introducción del dato de una celda del *JTable*. Recibe como parámetros el dato introducido, y la posición de la celda que se ha producido dicha introducción. Ofrece al programador la posibilidad de implementar el código asociado a las acciones que pretende se ejecuten cuando finalice dicha edición. En el caso del ejemplo:

- Almacenamos el valor introducido en el correspondiente componente del array de *Object*:

```
datos[fila][columna] = valor;
```

- Registraremos, para posteriores actuaciones, en los atributos correspondientes de la clase centralizadora Componentes, la posición de la celda actualizada:

```
componentes.setFilaActualizada(fila);
componentes.setColumnaActualizada(columna);
```

- Lanzamos explícitamente el evento *TableModelEvent*, lo cual permitirá desde la clase escucha correspondiente dar el tratamiento oportuno asociado a la introducción del dato:

```
fireTableDataChanged();
public boolean isCellEditable(int fila, int
columna ) {
if (columna < 3 || columna > 6)
return( false );
else
return( true );
}
```

Mediante el código que implementemos en este método establecemos qué columnas del *JTable* van a ser editables. En esta aplicación solamente permitimos modo edición a las columnas del *JTable* cuyo dato es editable (valga la redundancia) mediante casilla de verificación y que permite al usuario establecer el voto de una persona. Obviamente, impedimos al usuario la edición de las columnas asociadas, al **código, nombre, y cuota de participación** del votante.

## Listener

La escucha de eventos es el aspecto que en mayor grado confiere interactividad al componente. Pero, antes que nada, a efectos de proporcionar al lector una perspectiva más global sobre todo ello, vamos a aplicar un estudio comparativo entre los componentes tratados en este libro que llevan un Model asociado basado en lo que concierne a la escucha de eventos. Podemos clasificarlos en dos grupos en función de quien se registra con el Listener:

1. Es el propio elemento visualizador el que se registra con la escucha de eventos. Se encuentran en este grupo el *JList* (presentamos código extraído de la aplicación **InterfazGrafica9**):

```
componentes.setDefaultListModel(new  
DefaultListModel());  
componentes.setjList(new  
JList(componentes.getDefaultListModel()));  
componentes.getjList().addListSelectionListener(ge  
storEventos);
```

y el *JTree* (presentamos código extraído de la aplicación **GestionLibros**):

```
jTree = new JTree( raiz );  
modelo = (DefaultTreeModel)jTree.getModel();  
jTree.addMouseListener(controller);
```

2. Es el Model asociado al elemento visualizador el que se registra con la escucha de eventos el Model. Se encuentran en este grupo el *JSpinner* (presentamos código extraído de la aplicación **InterfazGrafica11**):

```
SpinnerListModel spinnerListModel = new  
SpinnerListModel(generos);  
spinnerListModel.addChangeListener(gestorEventos);  
componentes.setjSpinnerGeneros(new
```

```
JSpinner(spinnerListModel));
```

y el *JTable* (presentamos código extraído de la aplicación **VotacionPropuesta**):

```
componentes.setModeloDatos(new ModeloDatos( . . .  
));  
componentes.setjTable(new  
JTable(componentes.getModeloDatos())));
```

y el registro con la escucha de eventos lo hemos implementado en el constructor del Model:

```
public ModeloDatos(Componentes componentes, int  
numeroFilas) {  
. . .  
addTableModelListener(new  
GestorEventos(componentes));  
}
```

Hay ocasiones, tal y como podemos observar en los dos *JTable* que aparecen en la aplicación **GestionLibros**, que además de registrarse con escucha de eventos el Model asociado al *JTable*, también lo hace el *ListSelectionModel* que establecemos al *JTable*:

```
jTable = new JTable(modeloDatos);  
jTable.setSelectionMode(ListSelectionModel.SINGLE_  
SELECTION);  
listSelectionModel = jTable.getSelectionModel();  
listSelectionModel.addListSelectionListener(contro  
ller);
```

Concretamente el mismo Listener con que hemos registrado el *JList*.

Podemos concluir diciendo que existen Listeners asociados a componente, y Listeners asociados a Model.

En el caso del ejercicio que nos ocupa solamente se contempla la escucha de eventos *TableModelListener*, con que se registra el Model. Actuamos aquí, tal y como venimos haciendo en la mayor parte de ejercicios con interfaz gráfica presentados hasta ahora haciendo que la clase **GestorEventos** centralice todas las escuchas de eventos. En consecuencia, el registro de la escucha asociada a *TableModelListener* exige que **GestorEventos** implemente el método la citada escucha *tableChanged(TableModelEvent e)*. Dado que las columnas editables del *JTable* en cuestión son aquellas en que se produce la interacción con el usuario mediante casilla de verificación asociada a un *Boolean*, solamente se ejecutará el citado método cuando cambie el valor de alguna casilla de verificación. Lo más destacado a comentar de dicho método, es cómo se realiza el acceso desde este método a:

- array de *Object* asociado a los datos a gestionar por el Model:

```
componentes.getModeloDatos().getDatos()
```

donde **getDatos()** es el ya comentado método “getter” del array de *Object*

- fila de la celda afectada por la actualización del dato *Boolean*:

```
componentes.getFilaActualizada()
```

dato registrado en él método *setValueAt(Object valor, int fila, int columna )*, circunstancia ya comentada con anterioridad, y que aplicamos como expresión del primer índice de acceso al array

- columna de la celda afectada por la actualización en cuestión, pudiéndosele aplicar lo mismo que acabamos de

comentar, pero ahora para la expresión del segundo índice del array:

```
componentes.getColumnaActualizada()
```

Una vez comentado las expresiones utilizadas para acceder a la celda objeto de actualización, el lector no debe tener dificultad para entender la lógica del algoritmo aplicado, que tal y como se ha comentado al principio va únicamente enfocada a:

- Desmarcar la casilla de verificación correspondiente a esa persona que estuviese previamente marcada.
- Recalcular al instante los porcentajes totales, repercutiendo el resultado de dicho cálculo en los correspondientes contadores, y visualizando el valor de dichos contadores en los *JLabel* situados al pie de la columna correspondiente a cada opción.

Es necesario matizar que al “rescatar” un dato previamente introducido en el Model es necesario aplicar *cast* dada la declaración como *Object* con que se ha declarado el array **datos**:

```
((Boolean)componentes.getModeloDatos().getDatos()  
[componentes.getFilaActualizada()][i]).booleanValue()
```

En este ejemplo solamente se “rescatan” las columnas con datos *Boolean*, pero en otro caso se podrán rescatar objetos que respondan a cualquier otra clase.

Solamente nos queda tratar, una serie de aportaciones, que, con carácter novedoso, aparecen en esta aplicación, y que nos servirán de preludio para el estudio de la aplicación final **GestionLibros**.

## **Acceso a Base de Datos**

Es en esta aplicación, junto con la aplicación final **GestionLibros**, las únicas del libro en que aplicamos, de forma integral, en pro de los fines a contemplar por la aplicación, el acceso a Base de Datos acogiéndose a la estructura organizativa establecida por el modelo de desarrollo de software arquitectura a tres capas. Adicionalmente a las directrices que se proporcionaron en la parte del libro en que se trató este modelo de desarrollo, añadimos en la capa de datos, las clases

- IncidenciasDatos
- RepositorioXML
- VotacionesPDF

Estas clases no guardan ningún tipo de relación con el acceso a Bases de Datos desde una aplicación Java mediante el API JDBC, pero se han incluido en esta capa, porque suponen cada una de ellas la utilización de tecnologías específicas relacionadas con la E/S.

## **Fichero repositorio.xml**

Este fichero, presente en el directorio xml de la aplicación, actúa como repositorio centralizado de información externo al código del programa. La presencia de este fichero nos permite “extraer” del código del programa diferentes parámetros y datos susceptibles de cambiar en función de las circunstancias. En el caso de esta aplicación, solamente contemplamos:

- Los parámetros específicos necesarios para obtener una conexión con la Base de Datos, concretamente con Oracle y

MySQL. Este contexto podríamos hacerlo perfectamente extensivo a otros Gestores de Bases de Datos adicionalmente, de tal forma que, con total independencia de los Gestores de Bases de Datos representados en el fichero, la aplicación conectará con el Gestor establecido en:

```
<BASE_DATOS_SELECCIONADA>MySQL</BASE_DATOS_SELECCIONADA>
```

Observamos que con esta configuración, la aplicación conectará con MySQL.

- La ruta del sistema de archivos donde se crearán los ficheros PDF generados por la aplicación.

El fichero **repositorio.xml** correspondiente a la aplicación **GestionLibros** contempla, adicionalmente a los aquí expuestos, muchos más parámetros “externalizados” que ya se comentarán en su momento.

Las ventajas que nos reporta la “centralización externalizada” que representa este fichero son, entre otras, el incremento que supone en los aspectos organizativos y “documentativos”, pero sobre todo, la posibilidad de adaptar la aplicación a circunstancias que vengan impuestas externamente a la misma, sin necesidad de tener que recurrir a la modificación del código implementado. Tan sólo es necesario para ello, la edición del mencionado fichero mediante cualquier editor de texto. Es más, esta actuación podría correr a cargo de personal que no necesariamente deba disponer de la cualificación de programador.

## Gestión de incidencias

El tratamiento por parte del programa ante la intercepción de una excepción debe ir más allá de una mera visualización al usuario de que ha tenido lugar tal circunstancia. Tal vez, esta actuación, la única en muchas ocasiones, debería ser totalmente accesoria, y primar otras actuaciones de mayor relevancia y trascendencia, como podría ser dejar constancia de las mismas en un fichero de incidencias. En esta aplicación, como preludio al tratamiento que aplicaremos en el ejercicio final **GestionLibros**, le aplicamos una doble vertiente al tratamiento de las excepciones:

- Dejar constancia de las mismas en un fichero de incidencias. A tal efecto, se ha dispuesto en la capa **presentacion** la clase **GestorExcepciones**, a cuyo método **gestionarExpcion()** se canaliza finalmente el tratamiento de la excepción, en el que implementan las siguientes actuaciones:

- Encapsular en una instancia de la clase **Contexto** la información asociada al “escenario” en que se ha producido.
- Decodificar la información asociada a la naturaleza de la excepción que llega

```
genericaExpcion.getCodigoError()
```

y generar el *String mensajeError* con la descripción de la situación que generó la excepción.

- Invocar a capas inferiores

```
new  
IncidenciasNegocio().escribirFichero(codigoError,  
mensajeError,  
contexto);
```

a fin de que finalmente en el método **escribirEnFicheroIncidencias()** de la clase **IncidenciasDatos**, en la capa **datos**, se genere una fila en el fichero de caracteres **log.txt**, ubicado en el directorio **log** del proyecto, dispuesto a tal efecto.

- Visualizar al usuario la existencia de la excepción, mediante:

```
JOptionPane.showMessageDialog(null, mensajeError,  
    "ERROR",  
    JOptionPane.ERROR_MESSAGE);
```

## **Generación de ficheros PDF como documentos de salida**

Tal y como se ha mencionado cuando hemos descrito el funcionamiento de la aplicación, se genera un documento PDF en que queda constatado el resultado de una votación, paralelamente a cuando se produce el registro del resultado de la misma en la Base de Datos. No abordaremos en este libro el estudio en detalle concerniente a la generación del fichero PDF. Si el lector tiene inquietudes sobre los mecanismos relacionados con la gestión de este tipo de ficheros desde una aplicación Java, le remitimos al libro ya mencionado con anterioridad del mismo autor, en que se destina un tema en exclusiva a la gestión de ficheros PDF desde este lenguaje de programación.

# 26

## APLICACIÓN GESTIONLIBROS

La estructura que presenta esta aplicación corresponde a la establecida por el modelo de desarrollo de software arquitectura a tres capas:

- *package presentacion* aglutina las clases:
  - Menu
  - PantallaOpcion
  - Conexión
  - ConexionEfectuada
  - ConfigurarDocumento
  - Desconexion
  - EdicionNodo
  - EstadisticasActividad
  - VistaArbol
  - DefaultMutableTreeNodeHeredada
  - NodoRaiz
  - NodoGenero
  - NodoLibro
  - VistaFormulario
  - VistaPaginadaTabla

- VistaUnicaTabla
  - VolcadoaBD
  - EventoOpcionMenu
  - InvocacionAutomaticaMenu
  - ModeloDatos
  - Controller
  - GestorIncidencias
- *package negocio* aglutina las clases:
    - RepositorioNegocio
    - UsuariosBibliotecaNegocio
    - LibrosNegocio
    - IncidenciasNegocio
  - *package datos* aglutina las clases:
    - RepositorioXML
    - ConexionBaseDatos
    - UsuariosBibliotecaDatos
    - LibrosDatos
    - LibrosSecuencia
    - LibrosPDF
    - LibrosImpresora
    - ImpresorPagina
    - ActividadUsuariosDatos
    - IncidenciasDatos
  - *package encapsuladores* aglutina las clases:
    - SistemaArchivos
    - Contexto
    - BaseDatos

- Genero
- Libro
- ActividadUsuario
- LimitesListado
- ParametrosPaginacion
- ParametrosListado
- ParametrosSeleccionAgrupacion
- *package excepciones* aglutina las clases:
  - GenericaExcepcion
- *package utilities* aglutina las clases:
  - Filtros

## Menu

```
package presentacion;
import java.awt.CardLayout;
import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JPanel;
public class Menu extends JFrame {
private Controller controller;
private JPanel jPanelPrincipal;
private CardLayout cardLayout;
private JMenuBar jMenuBar = new JMenuBar();
private String[] opcionesMenu;
private String[][] opciones;
private JMenu[] jMenus;
private JMenuItem[][] jMenuItems;
public Menu() {
controller = new Controller();
```

```
controller.setMenu(this);
setSize(1200,900);
opcionesMenu = (String [])
(controller.getRepositorio())[2];
opciones = ((String [][][])
(controller.getRepositorio())[3]);
jMenus = new JMenu[opcionesMenu.length];
jMenuItem = new JMenuItem[opcionesMenu.length][];
ubicarComponentes();
setVisible(true);
// PRESENTA PANTALLA DE CONEXION
InvocacionAutomaticaMenu invocacionAutomaticaMenu =
new InvocacionAutomaticaMenu();
invocacionAutomaticaMenu.addEventoOpcionMenuListener(c
ontroller);
invocacionAutomaticaMenu.fireEventoOpcionMenu("Conexio
n");
}
private void ubicarComponentes() {
addWindowListener(controller); // REGISTRO DE ESCUCHA
DE EVENTOS DE VENTANA
for ( int i=0 ; i<opcionesMenu.length; i++)
{
jMenus[i] = new JMenu(opcionesMenu[i]);
jMenuItem[i] = new JMenuItem[opciones[i].length];
for ( int k=0 ; k<opciones[i].length; k++)
{
jMenuItem[i][k] = new JMenuItem(opciones[i][k]);
jMenus[i].add(jMenuItem[i][k]);
jMenuItem[i][k].addActionListener(controller);
}
jMenuBar.add(jMenus[i]);
}
setJMenuBar(jMenuBar);
}
public void responderAController(String actionCommand)
throws Exception {
```

```
String textoTitulo = "Gestión libros -  
"+actionCommand;  
if (actionCommand.compareTo("Conexion") != 0)  
textoTitulo+=" - Conectado como usuario  
"+controller.getUsuarioAutenticado().getIdentificadorU  
suario();  
setTitle(textoTitulo);  
boolean usuarioAutenticado = false;  
if  
(controller.getUsuarioAutenticado().getIdentificadorUs  
uario() != null)  
usuarioAutenticado = true;  
for ( int i=0 ; i<opcionesMenu.length; i++)  
{  
for ( int k=0 ; k<opciones[i].length; k++)  
{ if (jMenuItems[i]  
[k].getActionCommand().compareTo("Conexion") == 0 ||  
jMenuItems[i]  
[k].getActionCommand().compareTo(actionCommand) == 0)  
jMenuItems[i][k].setEnabled(false);  
else  
jMenuItems[i][k].setEnabled(usuarioAutenticado);  
}  
}  
if (jPanelPrincipal != null)  
this.getContentPane().remove(jPanelPrincipal);  
jPanelPrincipal = new JPanel(new CardLayout());  
PantallaOpcion pantallaOpcion =  
(PantallaOpcion)Class.forName("presentacion."+  
eliminarEspaciosEnBlanco(actionCommand)).newInstance()  
;  
controller.setPantallaOpcion(pantallaOpcion);  
pantallaOpcion.inicializarPostInstanciar(controller);  
pantallaOpcion.inicializarPantalla();  
jPanelPrincipal.add(eliminarEspaciosEnBlanco(actionCom  
mand), pantallaOpcion);  
this.getContentPane().add(jPanelPrincipal);
```

```

setVisible(true);
((CardLayout)jPanelPrincipal.getLayout()).show(jPanelPrincipal, eliminarEspaciosEnBlanco(actionCommand));
}
private String eliminarEspaciosEnBlanco(String cadenaRecibida) {
String nombreClase = "";
for ( int i=0 ; i<cadenaRecibida.length(); i++)
if (cadenaRecibida.charAt(i)!=' ')
nombreClase = nombreClase + cadenaRecibida.charAt(i);
return nombreClase;
}
public static void main(String[] args) {
EventQueue.invokeLater(new Runnable() {
@Override
public void run() {
new Menu();
}
});
}
}
}

```

## PantallaOpcion

```

package presentacion;
import javax.swing.JPanel;
public abstract class PantallaOpcion extends JPanel{
protected Controller controller;
protected Object[] componentesJPanel = new Object[20];
public abstract void
inicializarPostInstanciar(Controller controller)
throws Exception;
public abstract void responderAController(String
actionCommand) throws Exception;
public void inicializarPantalla() throws Exception {
}
public Object getComponenteJPanel(int componente) {

```

```
return componentesJPanel[componente];
}
public void setComponenteJPanel(Object object, int componente) {
componentesJPanel[componente] = object;
}
}
```

## Conexion

```
package presentacion;
import encapsuladores.BaseDatos;
import encapsuladores.Contexto;
import java.net.InetAddress;
import java.net.UnknownHostException;
import javax.swing.JLabel;
import javax.swing.JPasswordField;
import javax.swing.JTextField;
import negocio.UsuariosBibliotecaNegocio;
public class Conexion extends PantallaOpcion
{
private JLabel jLabelUsuario;
private JTextField jTextFieldUsuario;
private JLabel jLabelPassword;
private JPasswordField jPasswordFieldPassword;
public Conexion() {
ubicarComponentes();
}
private void ubicarComponentes() {
setLayout(null);
JLabel jLabelinstrucciones1 = new JLabel("Existen los
siguientes usuarios de prueba");
jLabelinstrucciones1.setBounds(400, 150, 600, 20);
add(jLabelinstrucciones1);
JLabel jLabelinstrucciones2 = new JLabel(" id_usuario
: usuario1 password : password1");
jLabelinstrucciones2.setBounds(400, 175, 600, 20);
}
```

```
add(jLabelinstrucciones2);
JLabel jLabelinstrucciones3 = new JLabel(" id_usuario
: usuario2 password : password2");
jLabelinstrucciones3.setBounds(400, 200, 600, 20);
add(jLabelinstrucciones3);
JLabel jLabelinstrucciones4 = new JLabel(" id_usuario
: usuario3 password : password4");
jLabelinstrucciones4.setBounds(400, 225, 600, 20);
add(jLabelinstrucciones4);
JLabel jLabelinstrucciones5 = new JLabel("Después de
introducir los datos requeridos pulse Intro en
password para proseguir con la autenticación");
jLabelinstrucciones5.setBounds(400, 275, 600, 20);
add(jLabelinstrucciones5);
jLabelUsuario = new JLabel("Usuario");
jLabelUsuario.setBounds(400, 370, 150, 20);
add(jLabelUsuario);
jTextFieldUsuario = new JTextField();
jTextFieldUsuario.setBounds(480, 370, 150, 20);
add(jTextFieldUsuario);
jLabelPassword = new JLabel("Password");
jLabelPassword.setBounds(400, 395, 150, 20);
add(jLabelPassword);
jPasswordFieldPassword = new JPasswordField();
jPasswordFieldPassword.setBounds(480, 395, 150, 20);
add(jPasswordFieldPassword);
jPasswordFieldPassword.addActionListener(controller);
}
public void inicializarPostInstanciar(Controller
controller) throws Exception {
this.controller = controller;
jPasswordFieldPassword.addActionListener(controller);
}
public void responderAController(String actionCommand)
throws Exception {
switch(actionCommand)
```

```

{
case "autenticacion" :
Contexto contexto = new
Contexto(jTextFieldUsuario.getText(), new
String(jPasswordFieldPassword.getPassword()), obtenerIP());
new
UsuariosBibliotecaNegocio().autenticar((BaseDatos)controller.getRepository()[0], contexto);
controller.setUsuarioAutenticado(contexto);
InvocacionAutomaticaMenu invocacionAutomaticaMenu =
new InvocacionAutomaticaMenu();
invocacionAutomaticaMenu.addEventoOpcionMenuListener(controller);
invocacionAutomaticaMenu.fireEventoOpcionMenu("ConexionEfectuada");
break;
}
}
private String obtenerIP() throws UnknownHostException
{
String[] cadenasIP =
InetAddress.getLocalHost().getHostAddress().split("\\.");
}
StringBuffer procesoIP = new StringBuffer(16);
for (int i=0; i<cadenasIP.length; i++)
{
if (i>0)
procesoIP.append(".");
procesoIP.append(String.format("%03d",
Integer.parseInt(cadenasIP[i]))));
}
return new String(procesoIP);
}
}

```

## ConexionEfectuada

```
package presentacion;
import java.awt.Font;
import javax.swing.JLabel;
public class ConexionEfectuada extends PantallaOpcion
{
private JLabel jLabelUsuarioConectado;
public ConexionEfectuada() {
ubicarComponentes();
}
private void ubicarComponentes() {
setLayout(null);
jLabelUsuarioConectado = new JLabel();
jLabelUsuarioConectado.setBounds(400, 300, 3000, 20);
jLabelUsuarioConectado.setFont(new Font("TimesRoman",
Font.BOLD, 20));
add(jLabelUsuarioConectado);
}
public void inicializarPostInstanciar(Controller
controller) throws Exception {
this.controller = controller;
jLabelUsuarioConectado.setText("Está conectado como
usuario
"+controller.getUsuarioAutenticado().getIdentificadorU
suario());
}
public void responderAController(String actionCommand)
throws Exception {
}
}
```

## ConfigurarDocumento

```
package presentacion;
import encapsuladores.BaseDatos;
import encapsuladores.Libro;
import encapsuladores.LmitesListado;
import encapsuladores.ParametrosListado;
```

```
import excepciones.GenericaExcepcion;
import java.awt.Color;
import java.util.List;
import javax.swing.ButtonGroup;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JFileChooser;
import javax.swing.JLabel;
import javax.swing.JRadioButton;
import negocio.LibrosNegocio;
public class ConfigurarDocumento extends
PantallaOpcion {
private JLabel jLabelCriterioOrdenacion;
private JRadioButton jRadioButtonIdLibro;
private JRadioButton jRadioButtonTitulo;
private JRadioButton jRadioButtonGenero;
private ButtonGroup buttonGroupCriteriosOrdenacion;
private JLabel jLabelSeleccionLmiteInferior;
private JComboBox jComboBoxSeleccionLmiteInferior;
private JLabel jLabelSeleccionLmiteSuperior;
private JComboBox jComboBoxSeleccionLmiteSuperior;
private JButton jButtonGenerarPDF;
private JButton jButtonImprimir;
public ConfigurarDocumento() {
componentes JPanel[10] = new Integer(1); // El valor
registrado determina el criterio de ordenación de las
filas en el documento PDF.
ubicarComponentes();
}
private void ubicarComponentes() {
setLayout(null);
jLabelCriterioOrdenacion = new JLabel("CRITERIO
ORDENACION :");
jLabelCriterioOrdenacion.setBounds(300, 100, 150, 25);
add(jLabelCriterioOrdenacion);
jRadioButtonIdLibro = new JRadioButton("identificador
libro", true);
```

```
jRadioButtonIdLibro.setBounds(300, 140, 150, 25);
add(jRadioButtonIdLibro);
jRadioButtonIdLibro.setActionCommand("ordenarPorIdentificador");
jRadioButtonTitulo = new JRadioButton("título",
false);
jRadioButtonTitulo.setBounds(300, 180, 150, 25);
add(jRadioButtonTitulo);
jRadioButtonTitulo.setActionCommand("ordenarPorTitulo");
);
jRadioButtonGenero = new JRadioButton("género",
false);
jRadioButtonGenero.setBounds(300, 220, 150, 25);
add(jRadioButtonGenero);
jRadioButtonGenero.setActionCommand("ordenarPorGenero");
);
buttonGroupCriteriosOrdenacion = new ButtonGroup();
buttonGroupCriteriosOrdenacion.add(jRadioButtonIdLibro);
buttonGroupCriteriosOrdenacion.add(jRadioButtonTitulo);
;
buttonGroupCriteriosOrdenacion.add(jRadioButtonGenero);
;
jLabelSeleccionLimiteInferior = new JLabel("LIMITE
INFERIOR :");
jLabelSeleccionLimiteInferior.setBounds(280, 300, 120,
20);
add(jLabelSeleccionLimiteInferior);
jComboBoxSeleccionLimiteInferior = new JComboBox();
jComboBoxSeleccionLimiteInferior.setBounds(410,300,370
,20);
jComboBoxSeleccionLimiteInferior.setBackground(Color.w
hite);
add(jComboBoxSeleccionLimiteInferior);
jLabelSeleccionLimiteSuperior = new JLabel("LIMITE
SUPERIOR :");
jLabelSeleccionLimiteSuperior.setBounds(280, 470, 120,
```

```
20);
add(jLabelSeleccionLimiteSuperior);
jComboBoxSeleccionLimiteSuperior = new JComboBox();
jComboBoxSeleccionLimiteSuperior.setBounds(410,470,370
,20);
jComboBoxSeleccionLimiteSuperior.setBackground(Color.w
hite);
add(jComboBoxSeleccionLimiteSuperior);
jButtonGenerarPDF = new JButton("Generar PDF");
jButtonGenerarPDF.setBounds(375,650,150,40);
jButtonGenerarPDF.setActionCommand("generarPDF");
add(jButtonGenerarPDF);
jButtonImprimir = new JButton("Imprimir");
jButtonImprimir.setBounds(575,650,150,40);
jButtonImprimir.setActionCommand("imprimir");
add(jButtonImprimir);
}
public void inicializarPostInstanciar(Controller
controller) throws Exception {
this.controller = controller;
jRadioButtonIdLibro.addActionListener(controller);
jRadioButtonTitulo.addActionListener(controller);
jRadioButtonGenero.addActionListener(controller);
jButtonGenerarPDF.addActionListener(controller);
jButtonImprimir.addActionListener(controller);
inicializarPantalla();
}
public void inicializarPantalla() throws Exception {
jComboBoxSeleccionLimiteInferior.removeAllItems();
jComboBoxSeleccionLimiteSuperior.removeAllItems();
List<Libro> listaLibros = null;
String[] listaGeneros = null;
int numeroItems = 0;
if (((Integer)componentesJPanel[10]).intValue() != 3)
{
listaLibros = new
LibrosNegocio().consultarTodos((BaseDatos)controller.g
```

```
etRepository()[0],  
((Integer)componentesJPanel[10]).intValue(), null,  
null);  
numeroItems = listaLibros.size();  
}  
else  
{ listaGeneros = new  
LibrosNegocio().consultarGeneros((BaseDatos)controller  
.getRepository()[0]);  
numeroItems = listaGeneros.length;  
}  
if (numeroItems > 0)  
{  
for (int i=0; i<numeroItems; i++)  
{  
if (((Integer)componentesJPanel[10]).intValue() != 3)  
{  
Libro libro = listaLibros.get(i);  
jComboBoxSeleccionLimiteInferior.addItem(libro.getIdLi  
bro()+" - "+libro.getTitulo());  
jComboBoxSeleccionLimiteSuperior.addItem(libro.getIdLi  
bro()+" - "+libro.getTitulo());  
}  
else  
{  
jComboBoxSeleccionLimiteInferior.addItem(listaGeneros[  
i]);  
jComboBoxSeleccionLimiteSuperior.addItem(listaGeneros[  
i]);  
}  
}  
jComboBoxSeleccionLimiteInferior.setSelectedIndex(0);  
jComboBoxSeleccionLimiteSuperior.setSelectedIndex(nume  
roItems-1);  
}  
}  
public void responderAController(String actionCommand)
```

```
throws Exception {
List<Libro> listaLibros;
switch(actionCommand)
{
case "reordenar" :
inicializarPantalla();
break;
case "generarPDF" :
listaLibros = obtenerListaLibros();
if (listaLibros.size() > 0)
new LibrosNegocio().generarPDF(listaLibros,
(ParametrosListado)controller.getRepositorio()[5],
seleccionDirectorioCreacionPDFs());
else
throw new GenericaExcepcion(70);
break;
case "imprimir" :
listaLibros = obtenerListaLibros();
if (listaLibros.size() > 0)
new LibrosNegocio().imprimir(listaLibros,
(ParametrosListado)controller.getRepositorio()[5]);
else
throw new GenericaExcepcion(70);
break;
}
}
private List<Libro> obtenerListaLibros() throws
Exception {
List<Libro> listaLibros;
LimitesListado limitesListado = new LimitesListado();
switch (((Integer)componentesJPanel[10]).intValue())
{
case 1:
limitesListado.setLimiteInferior(jComboBoxSeleccionLimiteInferior.getSelectedItem().toString().substring(0,
5));
limitesListado.setLimiteSuperior(jComboBoxSeleccionLimiteSuperior.getSelectedItem().toString().substring(0,
5));
}
```

```
iteSuperior.getSelectedItem().toString().substring(0,
5));
break;
case 2:
limitesListado.setLimiteInferior(jComboBoxSeleccionLim
iteInferior.getSelectedItem().toString().substring(10)
);
limitesListado.setLimiteSuperior(jComboBoxSeleccionLim
iteSuperior.getSelectedItem().toString().substring(10)
);
break;
case 3:
limitesListado.setLimiteInferior(jComboBoxSeleccionLim
iteInferior.getSelectedItem().toString().substring(0,
1));
limitesListado.setLimiteSuperior(jComboBoxSeleccionLim
iteSuperior.getSelectedItem().toString().substring(0,
1));
break;
}
listaLibros = new
LibrosNegocio().consultarTodos((BaseDatos)controller.g
etRepository())[0],
((Integer)componentesJPanel[10]).intValue(), null,
limitesListado);
return listaLibros;
}
private String seleccionDirectorioCreacionPDFs() {
JFileChooser jFileChooser = new
JFileChooser("D:\\PDFs_creados");
jFileChooser.setFileSelectionMode(JFileChooser.DIRECTO
RIES_ONLY);
jFileChooser.showDialog(this, "Selección directorio
creación PDFs");
return
jFileChooser.getSelectedFile().getAbsolutePath();
}
```

```
}
```

## Desconexion

```
package presentacion;
import javax.swing.JButton;
// Esta clase no se instancia en ningún momento.
// El mecanismo de la desconexión se implementa en el
método centralizar de la class Controller.
// La presenta clase podría eliminarse del proyecto.
public class Desconexion extends PantallaOpcion {
private JButton jButtonDesconectar;
public Desconexion() {
ubicarComponentes();
}
private void ubicarComponentes() {
setLayout(null);
jButtonDesconectar = new JButton("Desconectar");
jButtonDesconectar.setBounds(340,280,120,40);
jButtonDesconectar.setActionCommand("desconectar");
add(jButtonDesconectar);
}
public void inicializarPostInstanciar(Controller
controller) throws Exception {
this.controller = controller;
jButtonDesconectar.addActionListener(controller);
}
public void responderAController(String actionCommand)
throws Exception {
switch(actionCommand)
{
case "desconectar" :
controller.getUsuarioAutenticado().setIdentificadorUsu
ario(null);
controller.getUsuarioAutenticado().setPassword(null);
InvocacionAutomaticaMenu invocacionAutomaticaMenu =
new InvocacionAutomaticaMenu();
```

```
invocacionAutomaticaMenu.addEventoOpcionMenuListener(controller);
invocacionAutomaticaMenu.fireEventoOpcionMenu("Conexion");
break;
}
}
}
```

## EdicionNodo

```
package presentacion;
import encapsuladores.BaseDatos;
import encapsuladores.Libro;
import java.text.SimpleDateFormat;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
import negocio.LibrosNegocio;
import utilities.Filtros;
public class EdicionNodo extends PantallaOpcion {
private VistaArbol vistaArbol;
private JPanel panelInterior;
private JLabel jLabelCodigo;
private JTextField jTextFieldCodigo;
private JLabel jLabelTitulo;
private JTextField jTextFieldTitulo;
private JLabel jLabelFechaEdicion;
private JTextField jTextFieldFechaEdicion;
private JCheckBox jCheckBoxPremiado;
private JLabel jLabelNumeroPaginas;
private JTextField jTextFieldNumeroPaginas;
private JButton jButtonAplicarCambios;
private JButton jButtonCancelarEdicion;
private DefaultMutableTreeNodeHeredada
```

```
nodoGeneroSeleccion;
private DefaultMutableTreeNodeHeredada
nodoLibroSeleccion;
private int actualizaciones = 0;
public EdicionNodo() {
ubicarComponentes();
}
private void ubicarComponentes() {
setLayout(null);
panelInterior = new JPanel();
panelInterior.setLayout(null);
jLabelCodigo = new JLabel("Código");
jLabelCodigo.setBounds(20, 20, 45, 20);
panelInterior.add(jLabelCodigo);
jTextFieldCodigo = new JTextField();
jTextFieldCodigo.setBounds(65, 20, 48, 20);;
jTextFieldCodigo.setEditable(false);
panelInterior.add(jTextFieldCodigo);
jLabelTitulo = new JLabel("Título");
jLabelTitulo.setBounds(20, 60, 43, 20);
panelInterior.add(jLabelTitulo);
jTextFieldTitulo = new JTextField();
jTextFieldTitulo.setBounds(65, 60 ,300, 20);
panelInterior.add(jTextFieldTitulo);
componentesJPanel[2] = jTextFieldTitulo;
jLabelFechaEdicion = new JLabel("Fecha edición (dd-mm-
aaaa)");
jLabelFechaEdicion.setBounds(20, 100 ,170, 20);
panelInterior.add(jLabelFechaEdicion);
jTextFieldFechaEdicion = new JTextField();
jTextFieldFechaEdicion.setBounds(190, 100, 80, 20);
panelInterior.add(jTextFieldFechaEdicion);
componentesJPanel[3] = jTextFieldFechaEdicion;
jCheckBoxPremiado = new JCheckBox("Premiado");
jCheckBoxPremiado.setBounds(20, 140 ,150, 20);
panelInterior.add(jCheckBoxPremiado);
componentesJPanel[4] = jCheckBoxPremiado;
```

```
jLabelNumeroPaginas = new JLabel("Número páginas");
jLabelNumeroPaginas.setBounds(20, 180 ,100, 20);
panelInterior.add(jLabelNumeroPaginas);
jTextFieldNumeroPaginas = new JTextField();
jTextFieldNumeroPaginas.setBounds(124, 180, 43, 20);
panelInterior.add(jTextFieldNumeroPaginas);
componentesJPanel[6] = jTextFieldNumeroPaginas;
jButtonAplicarCambios = new JButton("Aplicar
cambios");
jButtonAplicarCambios.setBounds(125, 340, 130, 40);
jButtonAplicarCambios.setActionCommand("aplicarCambios
");
panelInterior.add(jButtonAplicarCambios);
jButtonCancelarEdicion = new JButton("Cancelar");
jButtonCancelarEdicion.setBounds(125, 400, 130, 40);
jButtonCancelarEdicion.setActionCommand("cancelarEdici
onNodo");
panelInterior.add(jButtonCancelarEdicion);
panelInterior.setBounds(1, 1, 760, 450);
add(panelInterior);
}
public void inicializarPostInstanciar(Controller
controller) throws Exception {
this.controller = controller;
jTextFieldTitulo.addCaretListener(controller);
jTextFieldFechaEdicion.addCaretListener(controller);
jCheckBoxPremiado.addItemListener(controller);
jTextFieldNumeroPaginas.addCaretListener(controller);
jButtonAplicarCambios.addActionListener(controller);
jButtonCancelarEdicion.addActionListener(controller);
}
public void setVistaArbol(VistaArbol vistaArbol) {
this.vistaArbol = vistaArbol;
}
public JPanel getPanelInterior(){
return panelInterior;
}
```

```
public void
inicializarPantalla(DefaultMutableTreeNodeHeredada
nodoGeneroSeleccion, DefaultMutableTreeNodeHeredada
nodoLibroSeleccion) throws Exception {
this.nodoGeneroSeleccion = nodoGeneroSeleccion;
this.nodoLibroSeleccion = nodoLibroSeleccion;
if (nodoLibroSeleccion != null)
{ // EDITAR NODO YA EXISTENTE
visualizarLibroSeleccionado(nodoLibroSeleccion.getIden
tificativo());
actualizaciones = 0;
}
else
{ // EDITAR NUEVO NODO
inicializarComponentes();
}
}
private void visualizarLibroSeleccionado(String
idLibro) throws Exception {
Libro libro = new Libro();
libro.setIdLibro(idLibro);
Libro libroObtenido = new
LibrosNegocio().consultarPorIdLibro((BaseDatos)control
ler.getRepository()[0], libro);
jTextFieldCodigo.setText(libroObtenido.getIdLibro());
jTextFieldTitulo.setText(libroObtenido.getTitulo());
jTextFieldFechaEdicion.setText(new
SimpleDateFormat("dd-MM-
yyyy").format(libroObtenido.getFechaEdicion()));
jCheckBoxPremiado.setSelected(libroObtenido.isPremiado
());
jTextFieldNumeroPaginas.setText(new
Integer(libroObtenido.getNumeroPaginas()).toString());
}
private void inicializarComponentes() {
jTextFieldCodigo.setText("");
jTextFieldTitulo.setText("");
}
```

```
jTextFieldFechaEdicion.setText("");
jCheckBoxPremiado.setSelected(false);
jTextFieldNumeroPaginas.setText("");
}
public void responderAController(String actionCommand)
throws Exception {
switch(actionCommand)
{
case "actualizadoTitulo" :
actualizaciones|=1;
break;
case "actualizadoFechaEdicion" :
actualizaciones|=4;
break;
case "actualizadoNumeroPaginas" :
actualizaciones|=8;
break;
case "actualizadoPremiado" :
actualizaciones|=16;
break;
case "aplicarCambios" :
Filtros.filtrarDatosNulos( new String[]{
jTextFieldTitulo.getText(),
jTextFieldFechaEdicion.getText(),
jTextFieldNumeroPaginas.getText()
})
;
Filtros.filtrarFecha(jTextFieldFechaEdicion.getText())
;
Filtros.filtrarNumeroPaginasLibro(Integer.parseInt(jTextFieldNumeroPaginas.getText()));
Libro libro = new Libro();
libro.setTitulo(jTextFieldTitulo.getText());
libro.setGenero(nodoGeneroSeleccion.getIdentificativo()
.substring(0, 1));
libro.setFechaEdicion(java.sql.Date.valueOf(jTextField
FechaEdicion.getText().substring(6, 10) +"-"+
```

```

jTextFieldFechaEdicion.getText().substring(3, 5) +”-”+
jTextFieldFechaEdicion.getText().substring(0, 2)
));
libro.setNumeroPaginas(Integer.parseInt(jTextFieldNumeroPaginas.getText()));
libro.setPremiado(jCheckBoxPremiado.isSelected());
if (jTextFieldCodigo.getText().compareTo("") == 0)
{
libro.setIdLibro(new
LibrosNegocio().insertar((BaseDatos)controller.getRepositorio()[0], libro));
vistaArbol.insertarNodoLibro(libro);
}
else
{
if (actualizaciones > 0)
{
libro.setIdLibro(jTextFieldCodigo.getText());
new
LibrosNegocio().actualizar((BaseDatos)controller.getRepositorio()[0], libro, actualizaciones);
vistaArbol.modificarNodoLibro(libro, actualizaciones);
}
}
vistaArbol.ocultarEdicionNodo();
break;
case “cancelarEdicionNodo” :
vistaArbol.ocultarEdicionNodo();
break;
}
}
}

```

## **EstadisticasActividad**

```

package presentacion;
import encapsuladores.ActividadUsuario;
```

```
import encapsuladores.BaseDatos;
import encapsuladores.ParametrosSeleccionAgrupacion;
import java.awt.Color;
import java.awt.Font;
import java.awt.datatransfer.Clipboard;
import java.awt.datatransfer.StringSelection;
import java.util.List;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import negocio.IncidenciasNegocio;
public class EstadisticasActividad extends
PantallaOpcion {
private JLabel jLabelSeleccionAgrupaciones;
private JCheckBox jCheckBoxSeleccionUsuario;
private JCheckBox jCheckBoxSeleccionFecha;
private JCheckBox jCheckBoxSeleccionActividad;
private JButton jButtonConsultar;
private JButton jButtonCopiarAlPortapapeles;
private JLabel jLabelSeleccionUsuario;
private JComboBox jComboBoxSeleccionUsuario;
private JLabel jLabelSeleccionFechaInferior;
private JComboBox jComboBoxSeleccionFechaInferior;
private JLabel jLabelSeleccionFechaSuperior;
private JComboBox jComboBoxSeleccionFechaSuperior;
private JTextArea jTextArea;
private JScrollPane jScrollPaneTextarea;
private Clipboard clipboard;
public EstadisticasActividad() {
ubicarComponentes();
}
private void ubicarComponentes() {
setLayout(null);
jLabelSeleccionAgrupaciones = new JLabel("SELECCIONE
```

```
AGRUPACIONES :”);
jLabelSeleccionAgrupaciones.setBounds(80, 50, 200,
20);
add(jLabelSeleccionAgrupaciones);
jCheckBoxSeleccionUsuario = new JCheckBox(“Usuario”);
jCheckBoxSeleccionUsuario.setBounds(100,100,150,20);
add(jCheckBoxSeleccionUsuario);
componentesJPanel[0] = jCheckBoxSeleccionUsuario;
jCheckBoxSeleccionFecha = new JCheckBox(“Fecha”);
jCheckBoxSeleccionFecha.setBounds(100,150,150,20);
add(jCheckBoxSeleccionFecha);
componentesJPanel[1] = jCheckBoxSeleccionFecha;
jCheckBoxSeleccionActividad = new
JCheckBox(“Actividad”);
jCheckBoxSeleccionActividad.setBounds(100,200,150,20);
add(jCheckBoxSeleccionActividad);
componentesJPanel[2] = jCheckBoxSeleccionActividad;
jLabelSeleccionUsuario = new JLabel(“USUARIO :”);
jLabelSeleccionUsuario.setBounds(80, 350, 120, 20);
add(jLabelSeleccionUsuario);
jComboBoxSeleccionUsuario = new JComboBox();
jComboBoxSeleccionUsuario.setBounds(210,350,370,20);
jComboBoxSeleccionUsuario.setBackground(Color.white);
add(jComboBoxSeleccionUsuario);
componentesJPanel[3] = jComboBoxSeleccionUsuario;
jLabelSeleccionFechaInferior = new JLabel(“FECHA
INFERIOR :”);
jLabelSeleccionFechaInferior.setBounds(80, 400, 120,
20);
add(jLabelSeleccionFechaInferior);
jComboBoxSeleccionFechaInferior = new JComboBox();
jComboBoxSeleccionFechaInferior.setBounds(210,400,370,
20);
jComboBoxSeleccionFechaInferior.setBackground(Color.wh
ite);
add(jComboBoxSeleccionFechaInferior);
componentesJPanel[4] =
```

```
jComboBoxSeleccionFechaInferior;
jLabelSeleccionFechaSuperior = new JLabel("FECHA
SUPERIOR :");
jLabelSeleccionFechaSuperior.setBounds(80, 450, 120,
20);
add(jLabelSeleccionFechaSuperior);
jComboBoxSeleccionFechaSuperior = new JComboBox();
jComboBoxSeleccionFechaSuperior.setBounds(210,450,370,
20);
jComboBoxSeleccionFechaSuperior.setBackground(Color.wh
ite);
add(jComboBoxSeleccionFechaSuperior);
componentesJPanel[5] =
jComboBoxSeleccionFechaSuperior;
jTextArea = new JTextArea();
jScrollPaneTextarea=new JScrollPane(jTextArea);
jScrollPaneTextarea.setBounds(600, 50, 550, 730);
add(jScrollPaneTextarea);
jTextArea.setEditable(false);
jTextArea.setFont(new
Font("TimesRoman",Font.PLAIN,11));
jButtonConsultar = new JButton("Consultar
estadística");
jButtonConsultar.setBounds(200,580,180,40);
jButtonConsultar.setActionCommand("consultarEstadistic
a");
add(jButtonConsultar);
jButtonCopiarAlPortapapeles = new JButton("Copiar al
portapapeles");
jButtonCopiarAlPortapapeles.setBounds(200,640,180,40);
jButtonCopiarAlPortapapeles.setActionCommand("copiarAl
Portapapeles");
add(jButtonCopiarAlPortapapeles);
clipboard = getToolkit().getSystemClipboard();
}
public void inicializarPostInstanciar(Controller
controller) throws Exception {
```

```
this.controller = controller;
jComboBoxSeleccionUsuario.removeAllItems();
jComboBoxSeleccionUsuario.addItem(" ");
List<String> listaUsuariosConActividad = new
IncidenciasNegocio().consultarUsuariosConActividad((Ba
seDatos)controller.getRepositorio()[0]);
for (int i=0; i<listaUsuariosConActividad.size(); i++)
jComboBoxSeleccionUsuario.addItem(listaUsuariosConActi
vidad.get(i));
jComboBoxSeleccionUsuario.setSelectedIndex(0);
List<String> listaFechasConActividad;
jComboBoxSeleccionFechaInferior.removeAllItems();
jComboBoxSeleccionFechaInferior.addItem(" ");
listaFechasConActividad = new
IncidenciasNegocio().consultarFechasConActividad((Base
Datos)controller.getRepositorio()[0]);
for (int i=0; i<listaFechasConActividad.size(); i++)
jComboBoxSeleccionFechaInferior.addItem(listaFechasCon
Actividad.get(i));
jComboBoxSeleccionFechaInferior.setSelectedIndex(0);
jComboBoxSeleccionFechaSuperior.removeAllItems();
jComboBoxSeleccionFechaSuperior.addItem(" ");
listaFechasConActividad = new
IncidenciasNegocio().consultarFechasConActividad((Base
Datos)controller.getRepositorio()[0]);
for (int i=0; i<listaFechasConActividad.size(); i++)
jComboBoxSeleccionFechaSuperior.addItem(listaFechasCon
Actividad.get(i));
jComboBoxSeleccionFechaSuperior.setSelectedIndex(0);
jButtonCopiarAlPortapapeles.setEnabled(false);
jButtonConsultar.addActionListener(controller);
jButtonCopiarAlPortapapeles.addActionListener(controll
er);
}
public void responderAController(String actionCommand)
throws Exception {
switch(actionCommand)
```

```
{  
case "consultarEstadistica" :  
// RECOGER PARAMETROS DE AGRUPACION Y SELECCION DESDEL  
EL JPanel  
ParametrosSeleccionAgrupacion  
parametrosSeleccionAgrupacion = new  
ParametrosSeleccionAgrupacion();  
// Recoger parámetros de agrupación  
for (int i=0;  
i<parametrosSeleccionAgrupacion.getCriteriaosAgrupacion()  
.length; i++)  
{  
parametrosSeleccionAgrupacion.setCriteriaoAgrupacion(((  
JCheckBox)componentesJPanel[i]).isSelected(), i);  
}  
// Recoger parámetros de selección  
for (int i=0;  
i<parametrosSeleccionAgrupacion.getCriteriaosSeleccion()  
.length; i++)  
{  
if  
(((JComboBox)componentesJPanel[i+3]).getSelectedItem()  
.toString().compareTo(" ") == 0)  
parametrosSeleccionAgrupacion.setCriteriaoSeleccion(null,  
i);  
else  
parametrosSeleccionAgrupacion.setCriteriaoSeleccion(((J  
ComboBox)componentesJPanel[i+3]).getSelectedItem().toS  
tring(), i);  
}  
visualizarConsultaEnJTextArea(new  
IncidenciasNegocio().consultarAplicandoCriteriaosSelecc  
ion((BaseDatos)controller.getRepositorio()[0],  
parametrosSeleccionAgrupacion),  
parametrosSeleccionAgrupacion);  
jButtonCopiarAlPortapapeles.setEnabled(true);  
break;
```

```
case "copiarAlPortapapeles" :  
    clipboard.setContents(new  
StringSelection(jTextArea.getText()), controller);  
    break;  
}  
}  
private void  
visualizarConsultaEnJTextArea(List<ActividadUsuario>  
listaActividadUsuario, ParametrosSeleccionAgrupacion  
parametrosSeleccionAgrupacion) {  
    boolean conAgrupaciones = false;  
    for (int i=0;  
i<parametrosSeleccionAgrupacion.getCriteriaosAgrupacion  
().length; i++)  
{  
    if  
(parametrosSeleccionAgrupacion.isCriteriaoAgrupacion(i)  
)  
        conAgrupaciones = true;  
    }  
    jTextArea.setText("");  
    for (int i=0; i<listaActividadUsuario.size(); i++)  
{  
        ActividadUsuario actividadUsuario =  
        listaActividadUsuario.get(i);  
        int contadorCriteriaosAgrupacion = 0;  
        if  
(parametrosSeleccionAgrupacion.isCriteriaoAgrupacion(co  
ntadorCriteriaosAgrupacion) || !conAgrupaciones)  
{  
            jTextArea.append("USUARIO:  
"+String.format("%-20s",actividadUsuario.getIdentifica  
dorUsuario())+"\n");  
        }  
        contadorCriteriaosAgrupacion ++;  
        if  
(parametrosSeleccionAgrupacion.isCriteriaoAgrupacion(co
```

```

ntadorCriteriosAgrupacion) || !conAgrupaciones)
{
jTextArea.append("FECHA:
"+String.format("%-20s", actividadUsuario.getCadenaFechaHora())+"\n");
}
contadorCriteriosAgrupacion++;
if
(parametrosSeleccionAgrupacion.isCriterioAgrupacion(co
ntadorCriteriosAgrupacion) || !conAgrupaciones)
{
jTextArea.append("CODIGO ACTIVIDAD:
"+String.format("%10s", actividadUsuario.getCodigoActividad())+"\n");
jTextArea.append("DESCRIPCION:
"+String.format("%-52s", actividadUsuario.getDescripcion())+"\n");
}
if (conAgrupaciones)
{
jTextArea.append("NUMERO DE ACTUACIONES:
"+String.format("%10s", actividadUsuario.getNumeroIncidentes())+"\n");
}
jTextArea.append("-----\n");
}
}
}
}

```

## VistaArbol

```

package presentacion;
import encapsuladores.BaseDatos;
import encapsuladores.Libro;
import java.awt.Component;
import java.awt.event.MouseEvent;

```

```
import java.util.List;
import javax.swing.JMenuItem;
import javax.swing.JPopupMenu;
import javax.swing.JScrollPane;
import javax.swing.JSplitPane;
import javax.swing.JTree;
import javax.swing.SwingUtilities;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeModel;
import javax.swing.tree.TreePath;
import negocio.LibrosNegocio;
public class VistaArbol extends PantallaOpcion {
private JScrollPane jScrollPane;
private JTree jTree;
private DefaultTreeModel modelo;
private NodoRaiz raiz;
private JSplitPane splitPane;
private EdicionNodo edicionNodo;
private JPopupMenu popupMenu1;
private JPopupMenu popupMenu2;
private JMenuItem nuevoNodo;
private JMenuItem editarNodo;
private JMenuItem eliminarNodo;
private String[] generos;
private DefaultMutableTreeNodeHeredada
nodoGeneroSeleccion;
private DefaultMutableTreeNodeHeredada
nodoLibroSeleccion;
public VistaArbol() {
componentesJPanel[13] = new Integer(-1); // Fila de la
posición en que se ha pulsado botón de ratón
componentesJPanel[14] = new Integer(-1); // Columna de
la posición en que se ha pulsado botón de ratón
componentesJPanel[15] = “”; // Componente sobre el que
se hace click con el ratón
componentesJPanel[16] = “”; // MouseEvent
ubicarComponentes();
```

```
}

private void ubicarComponentes() {
setLayout(null);
raiz = new NodoRaiz("Libros");
jTree = new JTree( raiz );
modelo = (DefaultTreeModel)jTree.getModel();
popupMenu1 = new JPopupMenu();
nuevoNodo = new JMenuItem("Nuevo nodo");
popupMenu1.add(nuevoNodo);
popupMenu2 = new JPopupMenu();
editarNodo = new JMenuItem("Editar nodo");
popupMenu2.add(editarNodo);
eliminarNodo = new JMenuItem("Eliminar nodo");
popupMenu2.add(eliminarNodo);
}
public void inicializarPostInstanciar(Controller controller) throws Exception {
this.controller = controller;
edicionNodo = new EdicionNodo();
edicionNodo.inicializarPostInstanciar(controller);
edicionNodo.setVistaArbol(this);
jScrollPane = new JScrollPane(jTree);
splitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, jScrollPane,
edicionNodo);
splitPane.setOneTouchExpandable(true);
splitPane.setDividerLocation(385);
splitPane.setBounds(1,1,775,460);
add(splitPane);
jTree.addMouseListener(controller);
nuevoNodo.addActionListener(controller);
editarNodo.addActionListener(controller);
eliminarNodo.addActionListener(controller);
ocultarEdicionNodo();
cargarArbol();
}
private void mostrarEdicionNodo() throws Exception {
```

```
controller.setPantallaOpcion(edicionNodo);
edicionNodo.inicializarPantalla(nodoGeneroSeleccion,
nodoLibroSeleccion);
edicionNodo.getPanelInterior().setVisible(true);
}
public void ocultarEdicionNodo() {
controller.setPantallaOpcion(this);
edicionNodo.getPanelInterior().setVisible(false);
}
private void cargarArbol() throws Exception {
while ( modelo.getChildCount(raiz) > 0 )
modelo.removeNodeFromParent((DefaultMutableTreeNode)mo
delo.getChildAt(raiz,0));
generos = new
LibrosNegocio().consultarGeneros((BaseDatos)controller
.getRepository()[0]);
for ( int i=0 ; i<generos.length ; i++)
{
NodoGenero nodoGenero = new NodoGenero( generos[i] );
nodoGenero.setIdentificativo(generos[i]);
modelo.insertNodeInto( nodoGenero, raiz,
modelo.getChildCount(raiz) );
List<Libro> listaLibros = new
LibrosNegocio().consultarTodos((BaseDatos)controller.g
etRepository()[0], 2, generos[i].substring(0, 1),
null);
for (int k=0; k<listaLibros.size(); k++)
{
Libro libro = listaLibros.get(k);
NodoLibro nodoLibro = new
NodoLibro(libro.getIdLibro()+" - "+libro.getTitulo());
nodoLibro.setIdentificativo(libro.getIdLibro());
nodoLibro.setTitulo(libro.getTitulo());
nodoGenero.add(nodoLibro);
}
}
}
```

```
public void insertarNodoLibro(Libro libro) {
int i;
boolean encontradaPosicion;
NodoLibro nodoLibro = new
NodoLibro(libro.getIdLibro()+" - "+libro.getTitulo())
);
nodoLibro.setIdentificativo(libro.getIdLibro());
nodoLibro.setTitulo(libro.getTitulo());
encontradaPosicion = false;
i = 0;
while ( !encontradaPosicion && i <
modelo.getChildCount( nodoGeneroSeleccion ) )
{
if (
libro.getTitulo().compareTo(((NodoLibro)modelo.getChildAt(nodoGeneroSeleccion, i)).getTitulo()) < 0)
encontradaPosicion = true;
else
i++;
}
modelo.insertNodeInto(nodoLibro, nodoGeneroSeleccion,
i);
}
private void eliminarNodoLibro() {
modelo.removeNodeFromParent((DefaultMutableTreeNode)mo
delo.getChildAt(nodoGeneroSeleccion,
modelo.getIndexofChild(nodoGeneroSeleccion,
nodoLibroSeleccion)));
}
public void modificarNodoLibro(Libro libro, int
actualizaciones) {
if (((int)actualizaciones & 1) > 0) // PROCEDEMOS A
ELIMINAR Y A INSERTAR NUEVAMENTE EL NODO SI SE HA
MODIFICADO EL TITULO
{
eliminarNodoLibro();
insertarNodoLibro(libro);
```

```
}

}

public void responderAController(String actionCommand)
throws Exception {
switch(actionCommand)
{
case "Nuevo nodo" :
case "Editar nodo" :
mostrarEdicionNodo();
break;
case "Eliminar nodo" :
Libro libro = new Libro();
libro.setIdLibro(nodoLibroSeleccion.getIdentificativo());
new
LibrosNegocio().eliminar((BaseDatos)controller.getRepo
sitorio()[0], libro);
eliminarNodoLibro();
break;
case "ratonClicked" :
nodoGeneroSeleccion = null;
nodoLibroSeleccion = null;
TreePath seleccionEvento =
jTree.getPathForLocation(((Integer)componentesJPanel[1
3]).intValue(),
((Integer)componentesJPanel[14]).intValue());
if (seleccionEvento != null)
{
DefaultMutableTreeNodeHeredada nodoSeleccionado =
(DefaultMutableTreeNodeHeredada)jTree.getLastSelectedP
athComponent();
DefaultMutableTreeNodeHeredada nodoSeleccionEvento =
(DefaultMutableTreeNodeHeredada)seleccionEvento.getLas
tPathComponent();
if ( nodoSeleccionado != null )
{
if
```

```
(seleccionEvento.getLastPathComponent().getClass().get
Name().compareTo("presentacion.NodoGenero")==0 &&
nodoSeleccionado.getClass().getName().compareTo("prese
ntacion.NodoGenero")==0 &&
nodoSeleccionado.getIdentificativo().compareTo(nodoSel
ecionEvento.getIdentificativo())==0 &&
SwingUtilities.isRightMouseButton((MouseEvent)componen
tes JPanel[16]) )
{
nodoGeneroSeleccion = nodoSeleccionado;
popupMenu1.show( (Component)componentes JPanel[15],
((Integer)componentes JPanel[13]).intValue(),
((Integer)componentes JPanel[14]).intValue() );
}
else
{
if
(seleccionEvento.getLastPathComponent().getClass().get
Name().compareTo("presentacion.NodoLibro")==0 &&
nodoSeleccionado.getClass().getName().compareTo("prese
ntacion.NodoLibro")==0 &&
nodoSeleccionado.getIdentificativo().compareTo(nodoSel
ecionEvento.getIdentificativo())==0 &&
SwingUtilities.isRightMouseButton((MouseEvent)componen
tes JPanel[16]) )
{
nodoGeneroSeleccion =
(DefaultMutableTreeNodeHeredada)nodoSeleccionado.getParent();
nodoLibroSeleccion = nodoSeleccionado ;
popupMenu2.show( (Component)componentes JPanel[15],
((Integer)componentes JPanel[13]).intValue(),
((Integer)componentes JPanel[14]).intValue() );
}
}
}
}
}
```

```
break;  
}  
}  
}
```

## DefaultMutableTreeNodeHeredada

```
package presentacion;  
import javax.swing.tree.DefaultMutableTreeNode;  
public abstract class DefaultMutableTreeNodeHeredada  
extends DefaultMutableTreeNode {  
DefaultMutableTreeNodeHeredada(String  
cadenaVisualizada)  
{  
super(cadenaVisualizada);  
}  
abstract void setIdentificativo(String  
identificativo);  
abstract String getIdentificativo();  
}
```

## NodoRaiz

```
package presentacion;  
public class NodoRaiz extends  
DefaultMutableTreeNodeHeredada {  
private String identificativo = “”;  
public NodoRaiz(String cadenaVisualizada)  
{  
super(cadenaVisualizada);  
}  
public void setIdentificativo(String identificativo)  
{  
this.identificativo = identificativo;  
}  
public String getIdentificativo()  
{  
return identificativo;
```

```
}
```

## NodoGenero

```
package presentacion;
public class NodoGenero extends
DefaultMutableTreeNodeHeredada {
private String identificativo = "";
public NodoGenero(String cadenaVisualizada)
{
super(cadenaVisualizada);
}
public void setIdentificativo(String identificativo)
{
this.identificativo = identificativo;
}
public String getIdentificativo()
{
return identificativo;
}
}
```

## NodoLibro

```
package presentacion;
public class NodoLibro extends
DefaultMutableTreeNodeHeredada {
private String identificativo = "";
private String titulo = "";
public NodoLibro(String cadenaVisualizada)
{
super(cadenaVisualizada);
}
public void setIdentificativo(String identificativo)
{
this.identificativo = identificativo;
}
```

```
public String getIdentificativo()
{
    return identificativo;
}
public void setTitulo(String titulo)
{
    this.titulo = titulo;
}
public String getTitulo()
{
    return titulo;
}
```

## VistaFormulario

```
package presentacion;
import encapsuladores.BaseDatos;
import encapsuladores.Libro;
import java.awt.Color;
import java.text.SimpleDateFormat;
import java.util.List;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JScrollPane;
import javax.swing.JSlider;
import javax.swing.JTextField;
import javax.swing.ListSelectionModel;
import javax.swing.border.TitledBorder;
import negocio.LibrosNegocio;
import utilities.Filtros;
public class VistaFormulario extends PantallaOpcion {
    private ListSelectionModel listSelectionModel;
    private JComboBox jComboBoxLibros;
```

```
private JLabel jLabelCodigo;
private JTextField jTextFieldCodigo;
private JLabel jLabelTitulo;
private JTextField jTextFieldTitulo;
private JLabel jLabelFechaEdicion;
private JTextField jTextFieldFechaEdicion;
private JCheckBox jCheckBoxPremiado;
private JLabel jLabelGenero;
private JList jListGeneros;
private JSlider jSliderNumeroPaginas;
private JLabel jLabelNumeroPaginas;
private JButton jButtonNuevoLibro;
private JButton jButtonAplicarCambios;
private JButton jButtonEliminar;
private int actualizaciones;
public VistaFormulario() {
    ubicarComponentes();
}
private void ubicarComponentes() {
    setLayout(null);
    jComboBoxLibros = new JComboBox();
    jComboBoxLibros.setBounds(20, 100, 450, 20);
    jComboBoxLibros.setBackground(Color.white);
    jComboBoxLibros.setActionCommand("comboLibros");
    add(jComboBoxLibros);
    jLabelCodigo = new JLabel("Código");
    jLabelCodigo.setBounds(550, 100, 45, 20);
    add(jLabelCodigo);
    jTextFieldCodigo = new JTextField();
    jTextFieldCodigo.setBounds(600, 100, 50, 20);
    jTextFieldCodigo.setEditable(false);
    add(jTextFieldCodigo);
    jLabelTitulo = new JLabel("Título");
    jLabelTitulo.setBounds(550, 1400, 45, 20);
    add(jLabelTitulo);
    jTextFieldTitulo = new JTextField();
    jTextFieldTitulo.setBounds(600, 140, 400, 20);
```

```
add(jTextFieldTitulo);
componentesJPanel[2] = jTextFieldTitulo;
jLabelFechaEdicion = new JLabel("Fecha edición (dd-mm-aaaa)");
jLabelFechaEdicion.setBounds(550, 180, 170, 20);
add(jLabelFechaEdicion);
jTextFieldFechaEdicion = new JTextField();
jTextFieldFechaEdicion.setBounds(730, 180, 80, 20);
add(jTextFieldFechaEdicion);
componentesJPanel[3] = jTextFieldFechaEdicion;
jCheckBoxPremiado = new JCheckBox("Premiado");
jCheckBoxPremiado.setBounds(550, 220, 150, 20);
add(jCheckBoxPremiado);
componentesJPanel[4] = jCheckBoxPremiado;
jLabelGenero = new JLabel("Genero");
jLabelGenero.setBounds(550, 260, 50, 20);
add(jLabelGenero);
jSliderNumeroPaginas = new
JSlider(JSlider.HORIZONTAL,1,2000,300);
jSliderNumeroPaginas.setBounds(20, 430, 1070, 48);
jSliderNumeroPaginas.setBorder(new
TitledBorder("Número páginas"));
jSliderNumeroPaginas.setPaintTicks(true);
jSliderNumeroPaginas.setMajorTickSpacing(200);
jSliderNumeroPaginas.setMinorTickSpacing(10);
add(jSliderNumeroPaginas);
componentesJPanel[6] = jSliderNumeroPaginas;
jLabelNumeroPaginas = new JLabel();
jLabelNumeroPaginas.setBounds(1095, 450, 80, 20);
add(jLabelNumeroPaginas);
jButtonNuevoLibro = new JButton("Nuevo libro");
jButtonNuevoLibro.setBounds(475, 550, 200, 40);
jButtonNuevoLibro.setActionCommand("nuevoLibro");
add(jButtonNuevoLibro);
jButtonAplicarCambios = new JButton("Aplicar
cambios");
jButtonAplicarCambios.setBounds(475, 630, 200, 40);
```

```
jButtonAplicarCambios.setActionCommand("aplicarCambios");
");
add(jButtonAplicarCambios);
jButtonEliminar = new JButton("Eliminar");
jButtonEliminar.setBounds(475, 710, 200, 40);
jButtonEliminar.setActionCommand("eliminarLibro");
add(jButtonEliminar);
}
public void inicializarPostInstanciar(Controller controller) throws Exception {
this.controller = controller;
jListGeneros = new JList(new
LibrosNegocio().consultarGeneros((BaseDatos)controller
.getRepositorio()[0]));
JScrollPane jScrollPaneGenero = new
JScrollPane(jListGeneros);
jScrollPaneGenero.setBounds(550, 290, 250, 115);
add(jScrollPaneGenero);
jListGeneros.setSelectionMode(ListSelectionModel.SINGL
E_SELECTION);
listSelectionModel = jListGeneros.getSelectionModel();
listSelectionModel.addListSelectionListener(controller
);
componentes JPanel[12] = listSelectionModel;
componentes JPanel[5] = jListGeneros;
jTextFieldTitulo.addCaretListener(controller);
jTextFieldFechaEdicion.addCaretListener(controller);
jCheckBoxPremiado.addItemListener(controller);
jSliderNumeroPaginas.addChangeListener(controller);
jButtonNuevoLibro.addActionListener(controller);
jButtonAplicarCambios.addActionListener(controller);
jButtonEliminar.addActionListener(controller);
}
public void inicializarPantalla() throws Exception {
jTextFieldCodigo.setText("");
jTextFieldTitulo.setText("");
jTextFieldFechaEdicion.setText("");
}
```

```
jCheckBoxPremiado.setSelected(false);
jSLiderNumeroPaginas.setValue(1);
jListGeneros.removeSelectionInterval(jListGeneros.getSelectedIndex(), jListGeneros.getSelectedIndex());
if (jComboBoxLibros.getActionListeners().length > 0)
jComboBoxLibros.removeActionListener(controller);
jComboBoxLibros.removeAllItems();
List<Libro> listaLibros = new
LibrosNegocio().consultarTodos((BaseDatos)controller.getRepository()[0], 2, null, null);
for (int i=0; i<listaLibros.size(); i++)
{
Libro libro = listaLibros.get(i);
jComboBoxLibros.addItem(libro.getIdLibro()+" -
"+libro.getTitulo());
}
jComboBoxLibros.addActionListener(controller);
actualizaciones = 0;
}
public void responderAController(String actionCommand)
throws Exception {
Libro libro;
switch(actionCommand)
{
case "comboLibros" :
visualizarLibroSeleccionado(jComboBoxLibros.getSelectedItem().toString().substring(0, 5));
actualizaciones = 0;
break;
case "nuevoLibro" :
inicializarPantalla();
break;
case "aplicarCambios" :
Filtros.filtrarDatosNulos( new String[]{
jTextFieldTitulo.getText(),
jTextFieldFechaEdicion.getText()
}
}
```

```
);

Filtros.filtrarFecha(jTextFieldFechaEdicion.getText())
;
// 
Filtros.filtrarNumeroPaginasLibro(jSliderNumeroPaginas
.getValue()); // Innecesario porque el JSlider tiene
el rango de valores limitado
Filtros.filtrarSeleccionJList((jListGeneros));
libro = new Libro();
libro.setTitulo(jTextFieldTitulo.getText());
libro.setGenero(((String)
(jListGeneros.getSelectedValue())).substring(0, 1));
libro.setFechaEdicion(java.sql.Date.valueOf(jTextField
FechaEdicion.getText().substring(6, 10) +”-”+
jTextFieldFechaEdicion.getText().substring(3, 5) +”-”+
jTextFieldFechaEdicion.getText().substring(0, 2)
));
libro.setNumeroPaginas(jSliderNumeroPaginas.getValue()
);
libro.setPremiado(jCheckBoxPremiado.isSelected());
if (jTextFieldCodigo.getText().compareTo("") == 0)
{
new
LibrosNegocio().insertar((BaseDatos)controller.getRepo
sitorio()[0], libro);
}
else
{
if (actualizaciones > 0)
{
libro.setIdLibro(jTextFieldCodigo.getText());
new
LibrosNegocio().actualizar((BaseDatos)controller.getRe
positorio()[0], libro, actualizaciones);
}
}
inicializarPantalla();
```

```
break;
case "eliminarLibro" :
if (jTextFieldCodigo.getText().compareTo("") != 0)
{
libro = new Libro();
libro.setIdLibro(jTextFieldCodigo.getText());
new
LibrosNegocio().eliminar((BaseDatos)controller.getRepo
sitorio()[0], libro);
inicializarPantalla();
}
break;
case "actualizadoTitulo" :
actualizaciones|=1;
break;
case "actualizadoGenero" :
actualizaciones|=2;
break;
case "actualizadoFechaEdicion" :
actualizaciones|=4;
break;
case "actualizadoNumeroPaginas" :
actualizaciones|=8;
jLabelNumeroPaginas.setText(Integer.toString(jSLiderNu
meroPaginas.getValue())+" páginas");
break;
case "actualizadoPremiado" :
actualizaciones|=16;
break;
}
System.out.println("Valor acumulado en actualizaciones
: "+actualizaciones);
}
private void visualizarLibroSeleccionado(String
idLibro) throws Exception {
Libro libro = new Libro();
libro.setIdLibro(idLibro);
```

```

Libro libroObtenido = new
LibrosNegocio().consultarPorIdLibro((BaseDatos)control
ller.getRepository()[0], libro);
jTextFieldCodigo.setText(libroObtenido.getIdLibro());
jTextFieldTitulo.setText(libroObtenido.getTitulo());
jTextFieldFechaEdicion.setText(new
SimpleDateFormat("dd-MM-
yyyy").format(libroObtenido.getFechaEdicion())));
jCheckBoxPremiado.setSelected(libroObtenido.isPremiado
());
jListGeneros.setSelectedValue(libroObtenido.getGenero(
)+" - "+libroObtenido.getDescripcion(), true);
jSliderNumeroPaginas.setValue(libroObtenido.getNumeroP
aginas());
}
}

```

## VistaPaginadaTabla

```

package presentacion;
import encapsuladores.BaseDatos;
import encapsuladores.Libro;
import encapsuladores.ParametrosPaginacion;
import java.awt.Font;
import javax.swing.ButtonGroup;
import javax.swing.DefaultCellEditor;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JRadioButton;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.ListSelectionModel;
import javax.swing.table.DefaultTableCellRenderer;
import javax.swing.table.TableColumn;
import negocio.LibrosNegocio;
public class VistaPaginadaTabla extends PantallaOpcion

```

```
{  
private ModeloDatos modeloDatos;  
private JTable jTable;  
private ListSelectionModel listSelectionModel;  
private JButton[] botonesPaginacion = {null, null,  
null, null, null, null, null, null, null, null,  
null, null, null, null, null, null, null, null, null};  
private JLabel[] etiquetasPuntosSuspensivos = {null,  
null};  
private ParametrosPaginacion parametrosPaginacion;  
private int numeroTotalPaginas = 0;  
public VistaPaginadaTabla() {  
componentesJPanel[2] = new Integer(-1); // Fila  
seleccionada en JTable  
componentesJPanel[10] = new Integer(1); // Criterio de  
ordenación de las filas del JTable.  
componentesJPanel[11] = new Integer(1); // Número de  
página seleccionada para ser visualizada  
componentesJPanel[16] = new Integer(-1); // En botones  
de paginación numérica, número de página que consta en  
el text del botón  
ubicarComponentes();  
}  
private void ubicarComponentes() {  
setLayout(null);  
}  
public void inicializarPostInstanciar(Controller  
controller) throws Exception {  
this.controller = controller;  
parametrosPaginacion =  
(ParametrosPaginacion)controller.getRepositorio()[4];  
modeloDatos = new ModeloDatos(controller,  
((ParametrosPaginacion)controller.getRepositorio()  
[4]).getNumeroFilasPagina());  
//componentesJPanel[15] = modeloDatos;  
jTable = new JTable(modeloDatos);  
JScrollPane jScrollPaneTabla = new
```

```
JScrollPane(jTable);
jScrollPaneTabla.setBounds(20, 20, 1140, 20+16*
((ParametrosPaginacion)controller.getRepositorio()
[4]).getNumeroFilasPagina());
add(jScrollPaneTabla);
componentesJPanel[0] = jTable;
jTable.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
listSelectionModel = jTable.getSelectionModel();
listSelectionModel.addListSelectionListener(controller);
);
componentesJPanel[1] = listSelectionModel;
TableColumn columna[] = new TableColumn[6];
columna[0] = jTable.getColumnModel().getColumn(0);
columna[0].setPreferredWidth(30);
columna[1] = jTable.getColumnModel().getColumn(1);
columna[1].setPreferredWidth(450);
columna[2] = jTable.getColumnModel().getColumn(2);
columna[2].setPreferredWidth(180);
JComboBox jComboBoxGenero = new JComboBox(new
LibrosNegocio().consultarGeneros((BaseDatos)controller
.getRepositorio()[0]));
jComboBoxGenero.setFont(new Font("TimesRoman",
Font.BOLD, 10));
columna[2].setCellEditor(new
DefaultCellEditor(jComboBoxGenero));
DefaultTableCellRenderer renderer = new
DefaultTableCellRenderer();
renderer.setToolTipText("Click para seleccionar
genero");
columna[2].setCellRenderer(renderer);
columna[3] = jTable.getColumnModel().getColumn(3);
columna[3].setPreferredWidth(40);
columna[4] = jTable.getColumnModel().getColumn(4);
columna[4].setPreferredWidth(45);
columna[5] = jTable.getColumnModel().getColumn(5);
columna[5].setPreferredWidth(35);
```

```
JLabel jLabelCriterioOrdenacion = new JLabel("CRITERIO  
ORDENACION :");  
jLabelCriterioOrdenacion.setBounds(100, 180+16*  
((ParametrosPaginacion)controller.getRepositorio()  
[4]).getNumeroFilasPagina(), 150, 25);  
add(jLabelCriterioOrdenacion);  
JRadioButton jRadioButtonIdLibro = new  
JRadioButton("identificador libro", true);  
jRadioButtonIdLibro.setBounds(100, 220+16*  
((ParametrosPaginacion)controller.getRepositorio()  
[4]).getNumeroFilasPagina(), 150, 25);  
add(jRadioButtonIdLibro);  
jRadioButtonIdLibro.setActionCommand("ordenarPorIdentifi-  
cador");  
jRadioButtonIdLibro.addActionListener(controller);  
JRadioButton jRadioButtonTitulo = new  
JRadioButton("título", false);  
jRadioButtonTitulo.setBounds(100, 260+16*  
((ParametrosPaginacion)controller.getRepositorio()  
[4]).getNumeroFilasPagina(), 150, 25);  
add(jRadioButtonTitulo);  
jRadioButtonTitulo.setActionCommand("ordenarPorTitulo");  
jRadioButtonTitulo.addActionListener(controller);  
JRadioButton jRadioButtonGenero = new  
JRadioButton("género", false);  
jRadioButtonGenero.setBounds(100, 300+16*  
((ParametrosPaginacion)controller.getRepositorio()  
[4]).getNumeroFilasPagina(), 150, 25);  
add(jRadioButtonGenero);  
jRadioButtonGenero.setActionCommand("ordenarPorGenero");  
jRadioButtonGenero.addActionListener(controller);  
ButtonGroup buttonGroupCriteriosOrdenacion = new  
ButtonGroup();  
buttonGroupCriteriosOrdenacion.add(jRadioButtonIdLibro);
```

```
buttonGroupCriteriosOrdenacion.add(jRadioButtonTitulo)
;
buttonGroupCriteriosOrdenacion.add(jRadioButtonGenero)
;
JButton jButtonEliminarFilaSeleccionada = new
JButton("Eliminar fila seleccionada");
jButtonEliminarFilaSeleccionada.setBounds(400, 260+16*
((ParametrosPaginacion)controller.getRepositorio()
[4]).getNumeroFilasPagina(), 200, 40);
jButtonEliminarFilaSeleccionada.setActionCommand("eliminarFilaSeleccionada");
add(jButtonEliminarFilaSeleccionada);
jButtonEliminarFilaSeleccionada.addActionListener(controller);
}
private void instanciarBotonesPaginacion() {
int numeroBotonesNumericos =
((ParametrosPaginacion)controller.getRepositorio()
[4]).getNumeroBotonesNumericos();
int paginaActual =
((Integer)componentesJPanel[11]).intValue();
// Eliminar de la Vista los botones de paginación de
la visualización anterior
for (int i=0; i<botonesPaginacion.length; i++)
{
if (botonesPaginacion[i] != null)
{ botonesPaginacion[i].setVisible(false);
remove(botonesPaginacion[i]);
botonesPaginacion[i] = null;
}
}
// Eliminar de la Vista las etiquetas de puntos
suspensivos de la visualización anterior
for (int i=0; i<etiquetasPuntosSuspensivos.length;
i++)
{
if (etiquetasPuntosSuspensivos[i] != null)
```

```
{ etiquetasPuntosSuspensivos[i].setVisible(false);
remove(etiquetasPuntosSuspensivos[i]);
etiquetasPuntosSuspensivos[i] = null;
}
}
int contadorBotones = 0;
if (paginaActual > 1)
{
botonesPaginacion[contadorBotones] = new JButton();
botonesPaginacion[contadorBotones].setBounds(50+
(80*contadorBotones), 80+16*
((ParametrosPaginacion)controller.getRepositorio()
[4]).getNumeroFilasPagina(), 70, 50);
botonesPaginacion[contadorBotones].setActionCommand("b
otonPaginacionAnterior");
botonesPaginacion[contadorBotones].setText("<");
add(botonesPaginacion[contadorBotones]);
botonesPaginacion[contadorBotones].addActionListener(c
ontroller);
contadorBotones++;
}
int paginaInicio = paginaActual;
if (paginaInicio-(numeroBotonesNumericos/2) > 0 &&
(numeroTotalPaginas-numeroBotonesNumericos) >= 0)
paginaInicio = paginaInicio-
(numeroBotonesNumericos/2);
else
paginaInicio = 1;
if ((numeroTotalPaginas-paginaInicio) <
(numeroBotonesNumericos-1) && (numeroTotalPaginas-
(numeroBotonesNumericos-1))>0)
paginaInicio = numeroTotalPaginas-
(numeroBotonesNumericos-1);
boolean esElPrimero = true;
int ultimoValorVariableControl = 0;
for (int i=paginaInicio; i <
paginaInicio+numeroBotonesNumericos &&
```

```
i<=numeroTotalPaginas; i++)
{
if (esElPrimero)
{
if (i > 1)
{
botonesPaginacion[contadorBotones] = new JButton();
botonesPaginacion[contadorBotones].setBounds(50+
(80*contadorBotones), 80+16*
((ParametrosPaginacion)controller.getRepositorio()
[4]).getNumeroFilasPagina(), 70, 50);
botonesPaginacion[contadorBotones].setActionCommand("b
otonPaginacionNumerica");
botonesPaginacion[contadorBotones].setText("1");
add(botonesPaginacion[contadorBotones]);
botonesPaginacion[contadorBotones].addActionListener(c
ontroller);
contadorBotones++;
}
if (i > 2)
{
etiquetasPuntosSuspensivos[0] = new JLabel(" . . .");
etiquetasPuntosSuspensivos[0].setBounds(50+
(80*contadorBotones), 80+16*
((ParametrosPaginacion)controller.getRepositorio()
[4]).getNumeroFilasPagina(), 70, 50);
add(etiquetasPuntosSuspensivos[0]);
contadorBotones++;
}
esElPrimero = false;
}
botonesPaginacion[contadorBotones] = new JButton();
botonesPaginacion[contadorBotones].setBounds(50+
(80*contadorBotones), 80+16*
((ParametrosPaginacion)controller.getRepositorio()
[4]).getNumeroFilasPagina(), 70, 50);
botonesPaginacion[contadorBotones].setActionCommand("b
```

```
otonPaginacionNumerica");
botonesPaginacion[contadorBotones].setText(new
Integer(i+1-1).toString());
add(botonesPaginacion[contadorBotones]);
botonesPaginacion[contadorBotones].addActionListener(c
ontroller);
contadorBotones++;
ultimoValorVariableControl = i;
}
if (ultimoValorVariableControl <= numeroTotalPaginas-
2)
{
etiquetasPuntosSuspensivos[1] = new JLabel(" . . .");
etiquetasPuntosSuspensivos[1].setBounds(50+
(80*contadorBotones), 80+16*
((ParametrosPaginacion)controller.getRepositorio()
[4]).getNumeroFilasPagina(), 70, 50);
add(etiquetasPuntosSuspensivos[1]);
contadorBotones++;
}
if (ultimoValorVariableControl <= numeroTotalPaginas-
1)
{
botonesPaginacion[contadorBotones] = new JButton();
botonesPaginacion[contadorBotones].setBounds(50+
(80*contadorBotones), 80+16*
((ParametrosPaginacion)controller.getRepositorio()
[4]).getNumeroFilasPagina(), 70, 50);
botonesPaginacion[contadorBotones].setActionCommand("b
otonPaginacionNumerica");
botonesPaginacion[contadorBotones].setText(new
Integer(numeroTotalPaginas).toString());
add(botonesPaginacion[contadorBotones]);
botonesPaginacion[contadorBotones].addActionListener(c
ontroller);
contadorBotones++;
}
```

```
if (paginaActual < numeroTotalPaginas)
{
botonesPaginacion[contadorBotones] = new JButton();
botonesPaginacion[contadorBotones].setBounds(50+
(80*contadorBotones), 80+16*
((ParametrosPaginacion)controller.getRepositorio()
[4]).getNumeroFilasPagina(), 70, 50);
botonesPaginacion[contadorBotones].setActionCommand("b
otonPaginacionSiguiente");
botonesPaginacion[contadorBotones].setText(">");
add(botonesPaginacion[contadorBotones]);
botonesPaginacion[contadorBotones].addActionListener(c
ontroller);
}
for (int i=0; i<botonesPaginacion.length; i++)
{
if (botonesPaginacion[i] != null)
if
(botonesPaginacion[i].getActionCommand().compareTo("bo
tonPaginacionNumerica") == 0 )
if (Integer.parseInt(botonesPaginacion[i].getText())
== paginaActual )
botonesPaginacion[i].setForeground(java.awt.Color.LIGH
T_GRAY);
}
}
private int calcularNumeroPaginas(Integer numeroFilas,
ParametrosPaginacion parametrosPaginacion) {
int numeroPaginas = numeroFilas.intValue() /
parametrosPaginacion.getNumeroFilasPagina();
if (numeroFilas.intValue() %
parametrosPaginacion.getNumeroFilasPagina() > 0)
numeroPaginas++;
return numeroPaginas;
}
public void inicializarPantalla() throws Exception {
parametrosPaginacion.setNumeroPagina(((Integer)componer
```

```
ntes JPanel[11]).intValue());
modeloDatos.cargar(new
LibrosNegocio().consultarPagina((BaseDatos)controller.
getRepositorio())[0],
((Integer)componentes JPanel[10]).intValue(),
parametrosPaginacion));
numeroTotalPaginas = calcularNumeroPaginas(new
LibrosNegocio().consultarNumeroFilas((BaseDatos)contro
ller.getRepositorio())[0]), parametrosPaginacion); // Es necesario tener en cuenta que actualizaciones realizadas por otras sesiones podrían repercutir en el número de filas de la tabla, y en consecuencia, en el número de páginas.
instanciarBotonesPaginacion();
}
public void responderAController(String actionCommand)
throws Exception {
switch(actionCommand)
{
case "reordenar" :
componentes JPanel[11] = new Integer(1);
break;
case "botonPaginacionAnterior" :
componentes JPanel[11] = new
Integer(((Integer)componentes JPanel[11]).intValue() -
1);
break;
case "botonPaginacionSiguiente" :
componentes JPanel[11] = new
Integer(((Integer)componentes JPanel[11]).intValue() +
1);
break;
case "botonPaginacionNumerica" :
componentes JPanel[11] = new
Integer(((Integer)componentes JPanel[16]).intValue());
break;
case "eliminarFilaSeleccionada" :
```

```

if (((Integer)componentes JPanel[2]).intValue() != -1)
{
Libro libro = new Libro();
libro.setIdLibro((String)modeloDatos.getDatos()
[((Integer)componentes JPanel[2]).intValue()][0]);
new
LibrosNegocio().eliminar((BaseDatos)controller.getRepo
sitorio()[0], libro);
componentes JPanel[11] = new Integer(1);
listSelectionModel.removeSelectionInterval(((Integer)c
omponentes JPanel[2]).intValue(),
((Integer)componentes JPanel[2]).intValue());
componentes JPanel[2] = new Integer(-1);
}
break;
}
inicializarPantalla();
}
}

```

## VistaUnicaTabla

```

package presentacion;
import encapsuladores.BaseDatos;
import encapsuladores.Libro;
import java.awt.Font;
import javax.swing.ButtonGroup;
import javax.swing.DefaultCellEditor;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JRadioButton;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.ListSelectionModel;
import javax.swing.table.DefaultTableCellRenderer;
import javax.swing.table.TableColumn;

```

```
import negocio.LibrosNegocio;
import utilities.Filtros;
public class VistaUnicaTabla extends PantallaOpcion {
private ModeloDatos modeloDatos;
private JTable jTable;
private ListSelectionModel listSelectionModel;
private JLabel jLabelCriterioOrdenacion;
private JRadioButton jRadioButtonIdLibro;
private JRadioButton jRadioButtonTitulo;
private JRadioButton jRadioButtonGenero;
private ButtonGroup buttonGroupCriteriosOrdenacion;
private JButton jButtonInsertarFila;
private JButton jButtonCancelarInsercionFila;
private JButton jButtonGuardarFilaInsertada;
private JButton jButtonEliminarFilaSeleccionada;
private JLabel jLabelInformaFilaInsertada;
private JLabel jLabelInformaGuardarFilaInsertada;
private int filaAInsertarJTable = -1;
public VistaUnicaTabla() {
componentes JPanel[2] = new Integer(-1); // Fila
seleccionada en JTable
componentes JPanel[10] = new Integer(1); // El valor
registrado determina el criterio de ordenación de las
filas del JTable.
componentes JPanel[13] = new Integer(1); // Fila de
celda actualizada en JTable.
componentes JPanel[14] = new Integer(1); // Columna de
celda actualizada en JTable.
ubicarComponentes();
}
private void ubicarComponentes() {
setLayout(null);
jLabelCriterioOrdenacion = new JLabel("CRITERIO
ORDENACION :");
jLabelCriterioOrdenacion.setBounds(100, 550, 150, 25);
add(jLabelCriterioOrdenacion);
jRadioButtonIdLibro = new JRadioButton("identificador
```

```
libro”, true);
jRadioButtonIdLibro.setBounds(100, 590, 150, 25);
add(jRadioButtonIdLibro);
jRadioButtonIdLibro.setActionCommand(“ordenarPorIdentificador”);
jRadioButtonTitulo = new JRadioButton(“título”, false);
jRadioButtonTitulo.setBounds(100, 630, 150, 25);
add(jRadioButtonTitulo);
jRadioButtonTitulo.setActionCommand(“ordenarPorTitulo”);
jRadioButtonGenero = new JRadioButton(“género”, false);
jRadioButtonGenero.setBounds(100, 670, 150, 25);
add(jRadioButtonGenero);
jRadioButtonGenero.setActionCommand(“ordenarPorGenero”);
buttonGroupCriteriosOrdenacion = new ButtonGroup();
buttonGroupCriteriosOrdenacion.add(jRadioButtonIdLibro);
buttonGroupCriteriosOrdenacion.add(jRadioButtonTitulo);
buttonGroupCriteriosOrdenacion.add(jRadioButtonGenero);
jButtonInsertarFila = new JButton(“Insertar fila al final”);
jButtonInsertarFila.setBounds(400,510,200,40);
jButtonInsertarFila.setActionCommand(“insertarFila”);
add(jButtonInsertarFila);
componentesJPanel[3] = jButtonInsertarFila;
jButtonCancelarInsercionFila = new JButton(“Cancelar inserción fila”);
jButtonCancelarInsercionFila.setBounds(400,580,200,40);
jButtonCancelarInsercionFila.setActionCommand(“cancelarInsercionFila”);
add(jButtonCancelarInsercionFila);
```

```
componentesJPanel[4] = jButtonCancelarInsercionFila;
jButtonGuardarFilaInsertada = new JButton("Guardar
fila insertada");
jButtonGuardarFilaInsertada.setBounds(400,650,200,40);
jButtonGuardarFilaInsertada.setActionCommand("guardarF
ilaInsertada");
add(jButtonGuardarFilaInsertada);
componentesJPanel[5] = jButtonGuardarFilaInsertada;
jButtonEliminarFilaSeleccionada = new
JButton("Eliminar fila seleccionada");
jButtonEliminarFilaSeleccionada.setBounds(400,720,200,
40);
jButtonEliminarFilaSeleccionada.setActionCommand("elim
inarFilaSeleccionada");
add(jButtonEliminarFilaSeleccionada);
componentesJPanel[6] =
jButtonEliminarFilaSeleccionada;
jLabelInformaFilaInsertada = new JLabel();
jLabelInformaFilaInsertada.setBounds(650,510,200,40);
add(jLabelInformaFilaInsertada);
jLabelInformaFilaInsertada.setText(" ");
jLabelInformaGuardarFilaInsertada = new JLabel();
jLabelInformaGuardarFilaInsertada.setBounds(650,650,50
0,40);
add(jLabelInformaGuardarFilaInsertada);
jLabelInformaGuardarFilaInsertada.setText(" ");
}
public void inicializarPostInstanciar(Controller
controller) throws Exception {
this.controller = controller;
modeloDatos = new ModeloDatos(controller, new
LibrosNegocio().consultarNumeroFilas((BaseDatos)contro
ller.getRepository()[0]));
componentesJPanel[15] = modeloDatos;
jTable = new JTable(modeloDatos);
JScrollPane jScrollPaneTabla = new
JScrollPane(jTable);
```

```
jScrollPaneTabla.setBounds(20,20,1140,450);
add(jScrollPaneTabla);
componentesJPanel[0] = jTable;
jTable.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
listSelectionModel = jTable.getSelectionModel();
listSelectionModel.addListSelectionListener(controller);
);
componentesJPanel[1] = listSelectionModel;
TableColumn columnas[] = new TableColumn[6];
columnas[0] = jTable.getColumnModel().getColumn(0);
columnas[0].setPreferredWidth(30);
columnas[1] = jTable.getColumnModel().getColumn(1);
columnas[1].setPreferredWidth(450);
columnas[2] = jTable.getColumnModel().getColumn(2);
columnas[2].setPreferredWidth(180);
JComboBox jComboBoxGenero = new JComboBox(new
LibrosNegocio().consultarGeneros((BaseDatos)controller
.getRepositorio()[0]));
jComboBoxGenero.setFont(new Font("TimesRoman",
Font.BOLD, 10));
columnas[2].setCellEditor(new
DefaultCellEditor(jComboBoxGenero));
DefaultTableCellRenderer renderer = new
DefaultTableCellRenderer();
renderer.setToolTipText("Click para seleccionar
genero");
columnas[2].setCellRenderer(renderer);
columnas[3] = jTable.getColumnModel().getColumn(3);
columnas[3].setPreferredWidth(40);
columnas[4] = jTable.getColumnModel().getColumn(4);
columnas[4].setPreferredWidth(45);
columnas[5] = jTable.getColumnModel().getColumn(5);
columnas[5].setPreferredWidth(35);
jRadioButtonIdLibro.addActionListener(controller);
jRadioButtonTitulo.addActionListener(controller);
jRadioButtonGenero.addActionListener(controller);
```

```
jButtonInsertarFila.addActionListener(controller);
jButtonCancelarInsercionFila.addActionListener(controller);
jButtonGuardarFilaInsertada.addActionListener(controller);
jButtonEliminarFilaSeleccionada.addActionListener(controller);
}
public void inicializarPantalla() throws Exception {
activarDesactivarJButtonInsercionFila(new boolean[]
{true, false, false, true});
modeloDatos.cargar(new
LibrosNegocio().consultarTodos((BaseDatos)controller.getRepository())[0],
((Integer)componentesJPanel[10]).intValue(), null,
null));
}
private void
activarDesactivarJButtonInsercionFila(boolean[]
valoresActivacionDesactivacion) {
for (int i=3; i<=6; i++)
((JButton)componentesJPanel[i]).setEnabled(valoresActivacionDesactivacion[i-3]);
}
public void responderAController(String actionCommand)
throws Exception {
Libro libro;
switch(actionCommand)
{
case "reordenar" :
inicializarPantalla();
break;
case "insertarFila" :
modeloDatos.insertarFila();
listSelectionModel.setSelectionInterval((modeloDatos.getDatos().length)-1,
(modeloDatos.getDatos().length)-1);
```

```
componentes JPanel[2] = new
Integer((modeloDatos.getDatos().length)-1); //
Modifica valor de fila seleccionada en JTable
filaAInsertarJTable =
(modeloDatos.getDatos().length)-1;
activarDesactivarJButtonInsercionFila(new boolean[]
{false, true, true, false});
jLabelInformaFilaInsertada.setText("Fila insertada al
final de la tabla");
jLabelInformaGuardarFilaInsertada.setText("Antes de
pulsar el botón Guardar situe el cursor en columna
Código de la fila");
break;
case "cancelarInsercionFila" :
modeloDatos.eliminarFilaInsertada();
listSelectionModel.removeSelectionInterval(((Integer)c
omponentes JPanel[2]).intValue(),
((Integer)componentes JPanel[2]).intValue());
componentes JPanel[2] = new Integer(-1); // Modifica
valor de fila seleccionada en JTable
filaAInsertarJTable = -1;
activarDesactivarJButtonInsercionFila(new boolean[]
{true, false, false, true});
jLabelInformaFilaInsertada.setText(" ");
jLabelInformaGuardarFilaInsertada.setText(" ");
break;
case "guardarFilaInsertada" :
Filtros.filtrarDatosNulos( new String[]{ (String)
(modeloDatos.getDatos())[filaAInsertarJTable][1]),
(String)(modeloDatos.getDatos())[filaAInsertarJTable]
[2]),
(String)(modeloDatos.getDatos())[filaAInsertarJTable]
[3])
}
);
Filtros.filtrarFecha((String) modeloDatos.getDatos()
[filaAInsertarJTable][3]);
```

```
Filtros.filtrarNumeroPaginasLibro(((Integer)modeloDatos.getDatos()[filaAInsertarJTable][4]).intValue());
libro = new Libro();
libro.setTitulo((String)(modeloDatos.getDatos()[filaAInsertarJTable][1]));
libro.setGenero(((String)(modeloDatos.getDatos()[filaAInsertarJTable][2])).substring(0, 1));
libro.setFechaEdicion(java.sql.Date.valueOf(((String)(modeloDatos.getDatos()[filaAInsertarJTable][3])).substring(6, 10) +"-"+
((String)(modeloDatos.getDatos()[filaAInsertarJTable][3])).substring(3, 5) +"-"+
((String)(modeloDatos.getDatos()[filaAInsertarJTable][3])).substring(0, 2)));
libro.setNumeroPaginas(((Integer)(modeloDatos.getDatos()[filaAInsertarJTable][4])).intValue());
libro.setPremiado(((Boolean)(modeloDatos.getDatos()[filaAInsertarJTable][5])).booleanValue());
new
LibrosNegocio().insertar((BaseDatos)controller.getRepositorio()[0], libro);
inicializarPantalla();
listSelectionModel.removeSelectionInterval(((Integer)componentes JPanel[2]).intValue(),
((Integer)componentes JPanel[2]).intValue());
componentes JPanel[2] = new Integer(-1); // Modifica
valor de fila seleccionada en JTable
filaAInsertarJTable = -1;
activarDesactivar JButtonInsercionFila(new boolean[] {true, false, false, true});
jLabelInformaFilaInsertada.setText(" ");
jLabelInformaGuardarFilaInsertada.setText(" ");
break;
case "eliminarFilaSeleccionada" :
if (((Integer)componentes JPanel[2]).intValue() != -1)
```

```
{  
libro = new Libro();  
libro.setIdLibro((String)modeloDatos.getDatos()  
[((Integer)componentesJPanel[2]).intValue()][0]);  
new  
LibrosNegocio().eliminar((BaseDatos)controller.getRepo  
sitorio()[0], libro);  
inicializarPantalla();  
listSelectionModel.removeSelectionInterval(((Integer)c  
omponentesJPanel[2]).intValue(),  
((Integer)componentesJPanel[2]).intValue());  
componentesJPanel[2] = new Integer(-1);  
}  
break;  
case "actualizadaColumnaJTable" :  
if (((Integer)componentesJPanel[14]).intValue() == 3)  
Filtros.filtrarFecha((String) modeloDatos.getDatos()  
[((Integer)componentesJPanel[13]).intValue()]  
[((Integer)componentesJPanel[14]).intValue()]);  
if (((Integer)componentesJPanel[14]).intValue() == 4)  
Filtros.filtrarNumeroPaginasLibro(((Integer)modeloDato  
s.getDatos()  
[((Integer)componentesJPanel[13]).intValue()]  
[((Integer)componentesJPanel[14]).intValue()]).intValu  
e());  
if (((String)modeloDatos.getDatos())  
[((Integer)componentesJPanel[13]).intValue()][  
[0]).compareTo("") != 0)  
{  
libro = new Libro();  
libro.setIdLibro((String)modeloDatos.getDatos()  
[((Integer)componentesJPanel[13]).intValue()][0]);  
if (((Integer)componentesJPanel[14]).intValue() == 2)  
libro.setdatoActualizado(((String)modeloDatos.getDatos  
())[((Integer)componentesJPanel[13]).intValue()]  
[((Integer)componentesJPanel[14]).intValue()]).substri  
ng(0, 1));
```

```

else
libro.setDatoActualizado(modeloDatos.getDatos()
[((Integer)componentesJPanel[13]).intValue()]
[((Integer)componentesJPanel[14]).intValue()]);
libro.setColumnaActualizada(((Integer)componentesJPanel
[14]).intValue());
new
LibrosNegocio().actualizar((BaseDatos)controller.getRe
positorio()[0], libro, -1);
}
break;
}
}
}

```

## VolcadoaBD

```

package presentacion;
import encapsuladores.BaseDatos;
import javax.swing.JButton;
import negocio.IncidenciasNegocio;
public class VolcadoaBD extends PantallaOpcion {
private JButton jButtonVolcar;
public VolcadoaBD() {
ubicarComponentes();
}
private void ubicarComponentes() {
setLayout(null);
jButtonVolcar = new JButton("Volcar a BD");
jButtonVolcar.setBounds(340,280,120,40);
jButtonVolcar.setActionCommand("volcarIncidencias");
add(jButtonVolcar);
}
public void inicializarPostInstanciar(Controller
controller) throws Exception {
this.controller = controller;
jButtonVolcar.addActionListener(controller);
}

```

```
}

public void responderAController(String actionCommand)
throws Exception {
switch(actionCommand)
{
case "volcarIncidencias" :
new
IncidenciasNegocio().volcarFichero((BaseDatos)controller.getRepository()[0]);
break;
}
}
}
```

## EventoOpcionMenu

```
package presentacion;
import java.awt.event.ActionEvent;
public class EventoOpcionMenu extends ActionEvent {
public EventoOpcionMenu(Object source, int id, String
command) {
super(source, id, command);
}
}
```

## InvocacionAutomaticaMenu

```
package presentacion;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.List;
public class InvocacionAutomaticaMenu {
private List<ActionListener> listaListeners = new
ArrayList();
public InvocacionAutomaticaMenu() {
}
public void addEventoOpcionMenuListener(ActionListener
actionListener) {
```

```
listaListeners.add(actionListener);
}
public void
removeEventoOpcionMenuListener(ActionListener
actionListener) {
listaListeners.add(actionListener);
}
public void fireEventoOpcionMenu(String command) {
for (int i=0; i<listaListeners.size(); i++) {
listaListeners.get(i).actionPerformed(new
EventoOpcionMenu(this, 1, command));
}
}
}
```

## ModeloDatos

```
package presentacion;
import encapsuladores.Libro;
import encapsuladores.ParametrosPaginacion;
import java.text.SimpleDateFormat;
import java.util.List;
import javax.swing.JTable;
import javax.swing.table.AbstractTableModel;
public class ModeloDatos extends AbstractTableModel {
private Object datos[][];
private String[] nombreColumnas =
{“Código”, “Título”, “Género”, “Fecha edición”, “Número
páginas”, “Premiado”};
private Controller controller;
private int numeroFilas = 0;
private Object copiaReservaDato;
private int numeroFilasRecibidas = 5;
public ModeloDatos(Controller controller, int
numeroFilas) throws Exception {
this.controller = controller;
this.numeroFilas = numeroFilas;
```

```
instanciarArrayDatos();
addTableModelListener(controller);
}
private void instanciarArrayDatos() throws Exception {
if (numeroFilas == 0)
numeroFilas=1;
datos = new Object[numeroFilas][6];
}
private void inicializarFilas(int filaInicial, int
filaFinal) {
for (int i=filaInicial; i<=filaFinal; i++)
{ datos[i][0] = "";
datos[i][1] = "";
datos[i][2] = "";
datos[i][3] = "";
datos[i][4] = new Integer(0);
datos[i][5] = new Boolean(false);
}
}
public void insertarFila() {
Object datosAuxiliar[][] = datos;
numeroFilas++;
datos = new Object[numeroFilas][6];
for (int i=0; i<(datos.length)-1; i++)
{ datos[i][0] = datosAuxiliar[i][0];
datos[i][1] = datosAuxiliar[i][1];
datos[i][2] = datosAuxiliar[i][2];
datos[i][3] = datosAuxiliar[i][3];
datos[i][4] = datosAuxiliar[i][4];
datos[i][5] = datosAuxiliar[i][5];
}
inicializarFilas((datos.length)-1, (datos.length)-1);
((JTable)controller.getPantallaOpcion().getComponenteJ
Panel(0)).repaint();
}
public void eliminarFilaInsertada() {
Object datosAuxiliar[][] = datos;
```

```
numeroFilas--;
datos = new Object[numeroFilas][6];
for (int i=0; i<datos.length; i++)
{ datos[i][0] = datosAuxiliar[i][0];
datos[i][1] = datosAuxiliar[i][1];
datos[i][2] = datosAuxiliar[i][2];
datos[i][3] = datosAuxiliar[i][3];
datos[i][4] = datosAuxiliar[i][4];
datos[i][5] = datosAuxiliar[i][5];
}
((JTable)controller.getPantallaOpcion().getComponenteJ
Panel(0)).repaint();
}
public void cargar(List<Libro> listaLibros) throws
Exception {
switch(controller.getPantallaOpcion().getClass().getNa
me())
{ case "presentacion.VistaUnicaTabla": numeroFilas =
listaLibros.size();
break;
case "presentacion.VistaPaginadaTabla": numeroFilas =
((ParametrosPaginacion)controller.getRepositorio()
[4]).getNumeroFilasPagina();
break;
}
numeroFilasRecibidas = listaLibros.size();
instanciarArrayDatos();
if (listaLibros.size() > 0)
{
for (int i=0; i<listaLibros.size(); i++)
{
Libro libro = listaLibros.get(i);
datos[i][0] = libro.getIdLibro();
datos[i][1] = libro.getTitulo();
datos[i][2] = libro.getGenero()+" -
"+libro.getDescripcion();
datos[i][3] = new SimpleDateFormat("dd-MM-
```

```
yyyy").format(libro.getFechaEdicion()));
datos[i][4] = new Integer(libro.getNumeroPaginas());
if (libro.isPremiado())
datos[i][5] = new Boolean(true);
else
datos[i][5] = new Boolean(false);
}
}
((JTable)controller.getPantallaOpcion().getComponenteJ
Panel(0)).repaint();
}
public Object[][] getDatos() {
return datos;
}
public Object getCopiaReservaDato() {
return copiaReservaDato;
}
public int getColumnCount() {
return(datos[0].length );
}
public int getRowCount() {
return(datos.length );
}
public Object getValueAt(int fila, int column) {
Object datoADevolver = null;
if (fila >= 0 && fila <= numeroFilas-1)
{
datoADevolver = datos[fila][column];
}
return datoADevolver;
}
public void setValueAt(Object valor,int fila, int
columna ) {
if (fila >= 0 && fila <= numeroFilas-1)
{ copiaReservaDato = datos[fila][columna];
datos[fila][columna] = valor;
fireTableCellUpdated(fila, columna);
```

```
// fireTableDataChanged();
}
}
public boolean isCellEditable(int fila, int columnna )
{
boolean editable = false;
switch(controller.getPantallaOpcion().getClass().getName())
{ case "presentacion.VistaUnicaTabla": if (columnna >0)
editable = true;
break;
case "presentacion.VistaPaginadaTabla": if (fila <
numeroFilasRecibidas && columnna >0)
editable = true;
break;
}
return editable;
}
public String getColumnName(int columnna) {
return nombreColumnas[columnna];
}
public Class getColumnClass(int columnna) {
Class classADevolver = null;
try {
switch (columnna)
{
case 0: classADevolver =
Class.forName("java.lang.String");
case 1: classADevolver =
Class.forName("java.lang.String");
case 2: classADevolver =
Class.forName("java.lang.String");
case 3: classADevolver =
Class.forName("java.lang.String");
break;
case 4: classADevolver =
Class.forName("java.lang.Integer");
}
```

```

break;
case 5: classADevolver =
Class.forName("java.lang.Boolean");
break;
}
} catch (ClassNotFoundException ex) {
System.out.println("Error debido a clase no encontrada
en obtener Class");
}
// classADevolver = getValueAt(0,columna).getClass();
Se podría devolver directamente en el caso de que el
JTable nunca estuviese sin filas.
return classADevolver;
}
}

```

## Controller

```

package presentacion;
import encapsuladores.BaseDatos;
import encapsuladores.Contexto;
import excepciones.GenericaExcepcion;
import java.awt.datatransfer.Clipboard;
import java.awt.datatransfer.ClipboardOwner;
import java.awt.datatransfer.Transferable;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.awt.event.MouseEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.util.HashMap;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JList;

```

```
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JSlider;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.ListSelectionModel;
import javax.swing.event.CaretEvent;
import javax.swing.event.CaretListener;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
import javax.swing.event.MouseInputListener;
import javax.swing.event.TableModelEvent;
import javax.swing.event.TableModelListener;
import negocio.IncidenciasNegocio;
import negocio.RepositorioNegocio;
public class Controller extends WindowAdapter
implements ActionListener, TableModelListener,
ListSelectionListener, CaretListener, ItemListener,
ChangeListener, MouseInputListener, ClipboardOwner {
private Object[] repositorio;
private Menu menu;
private PantallaOpcion pantallaOpcion;
private Contexto usuarioAutenticado = null;
HashMap<String, Integer> actividadesCodificadas;
private boolean registrarActualizadoNumeroPaginas =
true;
public Controller() {
try {
usuarioAutenticado = new Contexto(null, null,
obtenerIP());
repositorio = new
RepositorioNegocio().cargarRepositorio();
actividadesCodificadas = new
IncidenciasNegocio().consultarCodificacionActividades(
(BaseDatos)repositorio[0]);
```

```
        } catch (Exception exception)
        { new
GestorIncidencias().gestionarExcepcion(exception,
usuarioAutenticado); }
}
private String obtenerIP() throws UnknownHostException
{
String[] cadenasIP =
InetAddress.getLocalHost().getHostAddress().split("\\.");
}
StringBuffer procesoIP = new StringBuffer(16);
for (int i=0; i<cadenasIP.length; i++)
{
if (i>0)
procesoIP.append(".");
procesoIP.append(String.format("%03d",
Integer.parseInt(cadenasIP[i])));
}
return new String(procesoIP);
}
// MÉTODO DE WindowAdapter
public void windowClosing(WindowEvent e) {
centralizar("op_menu - CierreVentana");
}
// MÉTODO DE ActionListener
public void actionPerformed(ActionEvent e) {
// Los actionCommand han de tener un mínimo de ocho
posiciones para tener mayor longitud que "op_menu".
String componenteFuente = e.getActionCommand();
if (e.getSource() instanceof JMenuItem)
componenteFuente = ((JMenuItem)
e.getSource()).getActionCommand();
switch(componenteFuente)
{
case "Conexion" :
case "ConexionEfectuada" :
case "Volcado a BD":
```

```
case "Vista Formulario":  
case "Estadisticas Actividad":  
case "Vista Unica Tabla":  
case "Vista Paginada Tabla":  
case "Vista Arbol":  
case "Configurar Documento":  
case "Desconexion":  
case "CierreVentana":  
    centralizar("op_menu - "+componenteFuente);  
break;  
case "autenticacion" :  
case "volcarIncidencias" :  
case "consultarEstadistica" :  
case "copiarAlPortapapeles":  
case "comboLibros" :  
case "nuevoLibro" :  
case "aplicarCambios" :  
case "eliminarLibro" :  
case "insertarFila" :  
case "cancelarInsercionFila" :  
case "guardarFilaInsertada" :  
case "eliminarFilaSeleccionada" :  
case "Nuevo nodo" :  
case "Editar nodo" :  
case "Eliminar nodo" :  
case "aplicarCambiosNodo" :  
case "cancelarEdicionNodo" :  
case "generarPDF" :  
case "imprimir" :  
    centralizar(componenteFuente);  
break;  
case "ordenarPorIdentificador" :  
    pantallaOpcion.setComponenteJPanel(new Integer(1),  
    10);  
    centralizar("reordenar");  
break;  
case "ordenarPorTitulo" :  
    centralizar("ordenarPorTitulo");  
break;
```

```
pantallaOpcion.setComponenteJPanel(new Integer(2),
10);
centralizar("reordenar");
break;
case "ordenarPorGenero" :
pantallaOpcion.setComponenteJPanel(new Integer(3),
10);
centralizar("reordenar");
break;
case "botonPaginacionAnterior" :
centralizar(componenteFuente);
break;
case "botonPaginacionSiguiente" :
centralizar(componenteFuente);
break;
case "botonPaginacionNumerica" :
pantallaOpcion.setComponenteJPanel(new
Integer(((JButton)e.getSource()).getText()), 16);
centralizar(componenteFuente);
break;
}
}
// MÉTODO DE TableModelListener
public void tableChanged(TableModelEvent e )
{
// La invocación a estos métodos es efectiva cuando en
el método setValueAt(Object valor,int fila, int
columna )
// de ModeloDatos se activa el evento mediante la
invocación a fireTableCellUpdated(fila, columna);
pantallaOpcion.setComponenteJPanel(new
Integer(e.getFirstRow()), 13); // Fila de celda
actualizada en JTable.
pantallaOpcion.setComponenteJPanel(new
Integer(e.getColumn()), 14); // Columna de celda
actualizada en JTable.
centralizar("actualizadaColumnaJTable");
```

```
}

// MÉTODO DE ListSelectionListener
public void valueChanged(ListSelectionEvent e)
{
    // Cuando se efectúa botón del ratón sobre una fila se
    // intercepta el evento. Y cuando se suelta dicho botón,
    // vuelve a interceptarse el evento.
    ListSelectionModel listSelectionModel =
        (ListSelectionModel)e.getSource();
    switch(pantallaOpcion.getClass().getName())
    { case "presentacion.VistaUnicaTabla":
    case "presentacion.VistaPaginadaTabla":
        if (!(e.getValueIsAdjusting()))
        {
            pantallaOpcion.setComponent JPanel(new
                Integer(listSelectionModel.getMinSelectionIndex()),
                2); // Modifica valor de fila seleccionada en JTable
            System.out.println("fila seleccionada en JTable
                "+pantallaOpcion.getComponent JPanel(2));
        }
        break;
    case "presentacion.VistaFormulario":
        if (e.getSource() ==
            (pantallaOpcion.getComponent JPanel(12)))
        {
            centralizar("actualizadoGenero");
        }
        System.out.println("fila seleccionada en JList
            "+listSelectionModel.getMinSelectionIndex());
        System.out.println("dato actualizado "+((String)
            ((JList)pantallaOpcion.getComponent JPanel(5)).getSe
            lectedValue()));
        break;
    }
}

// MÉTODO DE CaretListener
public void caretUpdate(CaretEvent e)
```

```
{  
if (e.getSource() ==  
(pantallaOpcion.getComponenteJPanel(2)))  
{  
centralizar("actualizadoTitulo");  
}  
else  
if (e.getSource() ==  
(pantallaOpcion.getComponenteJPanel(3)))  
{  
centralizar("actualizadoFechaEdicion");  
}  
else  
if (e.getSource() ==  
(pantallaOpcion.getComponenteJPanel(6)))  
{  
centralizar("actualizadoNumeroPaginas");  
}  
System.out.println("Dato actualizado : "+  
((JTextField)e.getSource()).getText());  
}  
// MÉTODO DE ItemListener  
public void itemStateChanged(ItemEvent e)  
{  
if (e.getItemSelectable() ==  
(pantallaOpcion.getComponenteJPanel(4)))  
{  
{  
centralizar("actualizadoPremiado");  
}  
}  
}  
System.out.println("Dato actualizado : "+  
((JCheckBox)pantallaOpcion.getComponenteJPanel(4)).isSelected());  
}  
// MÉTODO DE ChangeListener  
public void stateChanged(ChangeEvent e)
```

```
{  
if (e.getSource() ==  
(pantallaOpcion.getComponenteJPanel(6)))  
{  
centralizar("actualizadoNumeroPaginas");  
}  
System.out.println("Dato actualizado : "+  
(JSlider)pantallaOpcion.getComponenteJPanel(6)).getVa  
lue());  
}  
// MÉTODO DE ClipboardOwner  
public void lostOwnership(Clipboard clipb,  
Transferable transferable) {  
JOptionPane.showMessageDialog(null, "Otra aplicación  
ha reemplazado el contenido del portapapeles que había  
sido depositado al pulsar el Botón destinado a ello de  
esta aplicación", "AVISO",  
JOptionPane.INFORMATION_MESSAGE);  
}  
// MÉTODOS DE MouseInputListener  
public void mouseClicked(MouseEvent e) {  
pantallaOpcion.setComponenteJPanel(new  
Integer(e.getX()), 13); // Fila de la posición en que  
se ha pulsado botón de ratón  
pantallaOpcion.setComponenteJPanel(new  
Integer(e.getY()), 14); // Columna de la posición en  
que se ha pulsado botón de ratón  
pantallaOpcion.setComponenteJPanel(e.getComponent(),  
15); // Componente sobre el que se hace click con el  
ratón  
pantallaOpcion.setComponenteJPanel(e, 16); //  
MouseEvent  
centralizar("ratonClicked");  
}  
public void mouseMoved(MouseEvent e) {}  
public void mousePressed(MouseEvent e) {
```

```
}

public void mouseReleased(MouseEvent e) {
}

public void mouseEntered(MouseEvent e) {
}

public void mouseExited(MouseEvent e) {
}

public void mouseDragged(MouseEvent e) {
}

public void centralizar(String actionCommand) {
try {
// El objetivo de la siguiente implementación es que
el movimiento del JSlider solamente genere un registro
en incidencias.
if
(actionCommand.compareTo("actualizadoNumeroPaginas")
!= 0 || registrarActualizadoNumeroPaginas)
{
usuarioAutenticado.setFechaHora(new java.util.Date());
new
GestorIncidencias().gestionarActividad(usuarioAutentic
ado,
actividadesCodificadas.get(actionCommand).intValue(),
actionCommand);
}
if
(actionCommand.compareTo("actualizadoNumeroPaginas")
!= 0)
registrarActualizadoNumeroPaginas = true;
else
registrarActualizadoNumeroPaginas = false;
if (actionCommand.substring(0, 7).compareTo("op_menu")
== 0) // OPCION DE MENU
{
switch (actionCommand)
{
case "op_menu - Conexion":
```

```
case "op_menu - ConexionEfectuada":  
case "op_menu - Volcado a BD":  
case "op_menu - Vista Formulario":  
case "op_menu - Estadisticas Actividad":  
case "op_menu - Vista Unica Tabla":  
case "op_menu - Vista Paginada Tabla":  
case "op_menu - Vista Arbol":  
case "op_menu - Configurar Documento":  
actionCommand = actionCommand.substring(10);  
menu.responderAController(actionCommand);  
break;  
case "op_menu - Desconexion":  
// DESCONECTAR SESION DE USUARIO Y PRESENTAR PANTALLA  
DE CONEXION  
usuarioAutenticado.setIdentificadorUsuario(null);  
usuarioAutenticado.setPassword(null);  
InvocacionAutomaticaMenu invocacionAutomaticaMenu =  
new InvocacionAutomaticaMenu();  
invocacionAutomaticaMenu.addEventoOpcionMenuListener(t  
his);  
invocacionAutomaticaMenu.fireEventoOpcionMenu("Conexio  
n");  
break;  
case "op_menu - CierreVentana":  
System.exit(0);  
break;  
}  
}  
else // RESPUESTA A EVENTO DISPARADO POR COMPONENTE U  
OPCION DE MENU DE Nodo DE VistaArbol  
{  
pantallaOpcion.responderAController(actionCommand);  
}  
} catch (Exception exception)  
{  
usuarioAutenticado.setFechaHora(new java.util.Date());  
new GestorIncidencias().gestionarExcepcion(exception,
```

```
usuarioAutenticado);
if
(actionCommand.compareTo("actualizadaColumnaJTable")
== 0)
{
if (exception instanceof GenericaExcepcion)
{
GenericaExcepcion genericaExcepcion =
(GenericExcepcion)exception;
if (genericaExcepcion.getCodigoError() <= 16) //  
CODIGOS DE ERROS DE GenericaExcepcion GENERADOS POR  
LOS METODOS filtrarFecha() O  
filtrarNumeroPaginasLibro() QUE IMPLICAN RESTAURAR A  
LA CELDA EL VALOR ANTERIOR
{
((ModeloDatos)pantallaOpcion.getComponenteJPanel(15)).  
getDatos()
[((Integer)pantallaOpcion.getComponenteJPanel(13)).int  
Value()]
[((Integer)pantallaOpcion.getComponenteJPanel(14)).int  
Value()] =
((ModeloDatos)pantallaOpcion.getComponenteJPanel(15)).  
getCopiaReservaDato();
((JTable)pantallaOpcion.getComponenteJPanel(0)).repain  
t();
}
}
}
}
}
}

public Object[] getRepositorio() {
return repositorio;
}
public void setMenu(Menu menu) {
this.menu = menu;
}
public PantallaOpcion getPantallaOpcion() {
```

```
return pantallaOpcion;
}
public void setPantallaOpcion(PantallaOpcion
pantallaOpcion) {
this.pantallaOpcion = pantallaOpcion;
}
public Contexto getUsuarioAutenticado() {
return usuarioAutenticado;
}
public void setUsuarioAutenticado(Contexto
usuarioAutenticado) {
this.usuarioAutenticado = usuarioAutenticado;
}
}
```

## GestorIncidentes

```
package presentacion;
import encapsuladores.Contexto;
import excepciones.GenericaExcepcion;
import java.io.IOException;
import javax.swing.JOptionPane;
import negocio.IncidenciasNegocio;
public class GestorIncidentes {
public void gestionarExcepcion(Exception excepcion,
Contexto contexto) {
int codigoError = 0;
String mensajeError = "";
if (excepcion instanceof GenericaExcepcion)
{
GenericaExcepcion genericaExcepcion =
(GenericExcepcion)excepcion;
codigoError = genericaExcepcion.getCodigoError();
String mensajesError[] = {"formato fecha erróneo",
" año erróneo", "mes erróneo", "día erróneo", "número
de páginas erróneo"};
for (int i=1, contadorMensajes=0; i<=16; i*=2,
```

```
contadorMensajes++)
{ int codigoErrorFechaParcial = codigoError & i;
if (codigoErrorFechaParcial > 0)
mensajeError += mensajesError[contadorMensajes]+” “;
}
switch (genericaExcepcion.getCodigoError())
{
case 17: mensajeError = “No puede dejar ningún dato
nulo”;
break;
case 18: mensajeError = “Debe seleccionar género”;
break;
case 20: mensajeError = “No se ha podido conectar: No
existe un usuario con este identificador”;
break;
case 21: mensajeError = “No se ha podido conectar:
Password introducido incorrecto”;
break;
case 25: mensajeError = “Se ha producido una situación
de error como consecuencia de problemas con la
conexión a la BD”;
break;
case 30: mensajeError = “Se ha producido una situación
de error en la lectura del fichero XML del
repositorio”;
break;
case 31: mensajeError = “Se ha producido una situación
de error de E/S al intentar acceder al fichero XML del
repositorio”;
break;
case 40: mensajeError = “Se ha producido un error al
intentar insertar actividad de usuario”;
break;
case 41: mensajeError = “Se ha producido un error al
intentar consultar relación de usuarios con
actividad”;
break;
```

```
case 42: mensajeError = "Se ha producido un error al intentar consultar relación de fechas en que se ha producido actividad";
break;
case 43: mensajeError = "Se ha producido un error al intentar consultar estadística de actividad";
break;
case 50: mensajeError = "Se ha producido un error al intentar insertar libro";
break;
case 51: mensajeError = "Se ha producido un error al intentar eliminar libro";
break;
case 52: mensajeError = "Se ha producido un error al intentar actualizar un dato de libro";
break;
case 53: mensajeError = "Se ha producido un error al intentar actualizar datos de libro";
break;
case 54: mensajeError = "Se ha producido un error al intentar consultar por identificador de libro";
break;
case 55: mensajeError = "Se ha producido un error al intentar consultar listado de libros";
break;
case 56: mensajeError = "Se ha producido un error al intentar página de listado de libros";
break;
case 57: mensajeError = "Se ha producido un error al intentar consultar número de libros registrados";
break;
case 58: mensajeError = "Se ha producido un error al intentar consultar géneros";
break;
case 60: mensajeError = "Se ha producido un error al intentar consultar por identificador de usuario";
break;
```

```
case 70: mensajeError = “No se puede generar el
documento por ausencia de filas dados los límites
establecidos”;
break;
case 100: mensajeError = “Se ha producido una
situación de error en la BD al intentar obtener valor
de secuencia libros”;
break;
}
}
else
{ if (excepcion instanceof IOException)
{
mensajeError = “Se ha producido una situación de error
de E/S”;
}
else
{
mensajeError = excepcion.getMessage();
}
}
try {
new IncidenciasNegocio().escribirFichero(codigoError,
mensajeError, “errores”, contexto);
} catch(IOException iOException) {
System.out.println(“Se ha producido un error al
intentar escribir en fichero log de errores”); }
JOptionPane.showMessageDialog(null, mensajeError,
“ERROR”, JOptionPane.ERROR_MESSAGE);
}
public void gestionarActividad(Contexto contexto, int
codigo, String mensaje) {
try {
new IncidenciasNegocio().escribirFichero(codigo,
mensaje, “actividad”, contexto);
} catch(IOException iOException) {
System.out.println(“Se ha producido un error al
```

```
intentar escribir en fichero log de actividad"); }  
}  
}
```

## RepositorioNegocio

```
package negocio;  
import datos.RepositorioXML;  
public class RepositorioNegocio {  
public Object[] cargarRepositorio() throws Exception  
{  
return new RepositorioXML().cargar();  
}  
}
```

## UsuariosBibliotecaNegocio

```
package negocio;  
import datos.ConexionBaseDatos;  
import datos.UsuariosBibliotecaDatos;  
import encapsuladores.BaseDatos;  
import encapsuladores.Contexto;  
import excepciones.GenericaExcepcion;  
import java.sql.Connection;  
public class UsuariosBibliotecaNegocio {  
public Contexto autenticar(BaseDatos baseDatos,  
Contexto usuarioIntroducido) throws Exception  
{  
Connection connection = null;  
Contexto usuarioAutenticado = null;  
ConexionBaseDatos conexionBaseDatos = new  
ConexionBaseDatos();  
try {  
connection =  
conexionBaseDatos.abrirConexion(baseDatos);  
usuarioAutenticado = new  
UsuariosBibliotecaDatos().buscarPorIdUsuario(connectio  
n, usuarioIntroducido);
```

```
if (usuarioAutenticado == null)
throw new GenericaExcepcion(20);
if
(usuarioAutenticado.getPassword().compareTo(usuarioIntroducido.getPassword()) != 0)
throw new GenericaExcepcion(21);
usuarioAutenticado.setIpCliente(usuarioIntroducido.getIpCliente());
} catch (Exception excepcion)
{
throw excepcion;
}
finally
{
conexionBaseDatos.cerrarConexion(connection);
}
return usuarioAutenticado;
}
```

## LibrosNegocio

```
package negocio;
import datos.ConexionBaseDatos;
import datos.LibrosDatos;
import datos.LibrosImpresora;
import datos.LibrosPDF;
import datos.LibrosSecuencia;
import encapsuladores.BaseDatos;
import encapsuladores.Genero;
import encapsuladores.Libro;
import encapsuladores.LmitesListado;
import encapsuladores.ParametrosListado;
import encapsuladores.ParametrosPaginacion;
import java.sql.Connection;
import java.util.List;
public class LibrosNegocio {
```

```
public String insertar(BaseDatos baseDatos, Libro libro) throws Exception{
    Connection connection = null;
    ConexionBaseDatos conexionBaseDatos = new
    ConexionBaseDatos();
    try {
        connection =
        conexionBaseDatos.abrirConexion(baseDatos);
        libro.setIdLibro(new
        LibrosSecuencia().consultarValorSecuencia(connection,
        baseDatos));
        new LibrosDatos().insertar(connection, libro);
    } catch (Exception excepcion)
    {
        throw excepcion;
    }
    finally
    {
        conexionBaseDatos.cerrarConexion(connection);
    }
    return libro.getIdLibro();
}
public void eliminar(BaseDatos baseDatos, Libro libro) throws Exception{
    Connection connection = null;
    ConexionBaseDatos conexionBaseDatos = new
    ConexionBaseDatos();
    try {
        connection =
        conexionBaseDatos.abrirConexion(baseDatos);
        new LibrosDatos().eliminar(connection, libro);
    } catch (Exception excepcion)
    {
        throw excepcion;
    }
    finally
    {
```

```
conexionBaseDatos.cerrarConexion(connection);
}
}
public void actualizar(BaseDatos baseDatos, Libro
libro, int actualizaciones) throws Exception{
Connection connection = null;
ConexionBaseDatos conexionBaseDatos = new
ConexionBaseDatos();
try {
connection =
conexionBaseDatos.abrirConexion(baseDatos);
if (actualizaciones == -1)
new LibrosDatos().actualizarColumna(connection,
libro);
else
new LibrosDatos().actualizar(connection, libro,
actualizaciones);
} catch (Exception excepcion)
{
throw excepcion;
}
finally
{
conexionBaseDatos.cerrarConexion(connection);
}
}
public Libro consultarPorIdLibro(BaseDatos baseDatos,
Libro libro) throws Exception
{
Connection connection = null;
Libro libroObtenido = null;
ConexionBaseDatos conexionBaseDatos = new
ConexionBaseDatos();
try {
connection =
conexionBaseDatos.abrirConexion(baseDatos);
libroObtenido = new
```

```
LibrosDatos().consultarPorIdLibro(connection, libro);
} catch (Exception excepcion)
{
throw excepcion;
}
finally
{
conexionBaseDatos.cerrarConexion(connection);
}
return libroObtenido;
}
public List<Libro> consultarTodos(BaseDatos baseDatos,
int criterioOrdenacion, String genero, LimitesListado
limitesListado) throws Exception
{
Connection connection=null;
ConexionBaseDatos conexionBaseDatos = new
ConexionBaseDatos();
List<Libro> listaLibros = null;
try {
connection =
conexionBaseDatos.abrirConexion(baseDatos);
listaLibros = new
LibrosDatos().consultarTodos(connection,
criterioOrdenacion, genero, limitesListado);
} catch (Exception excepcion)
{
throw excepcion;
}
finally
{
conexionBaseDatos.cerrarConexion(connection);
}
return listaLibros;
}
public List<Libro> consultarPagina(BaseDatos
baseDatos, int criterioOrdenacion,
```

```
ParametrosPaginacion parametrosPaginacion) throws
Exception
{
Connection connection=null;
ConexionBaseDatos conexionBaseDatos = new
ConexionBaseDatos();
List<Libro> listaLibros = null;
try {
connection =
conexionBaseDatos.abrirConexion(baseDatos);
listaLibros = new
LibrosDatos().consultarPagina(connection,
criterioOrdenacion, parametrosPaginacion, baseDatos);
} catch (Exception excepcion)
{
throw excepcion;
}
finally
{
conexionBaseDatos.cerrarConexion(connection);
}
return listaLibros;
}
public Integer consultarNumeroFilas(BaseDatos
baseDatos) throws Exception
{
Connection connection=null;
ConexionBaseDatos conexionBaseDatos = new
ConexionBaseDatos();
Integer numFilas = null;
try {
connection =
conexionBaseDatos.abrirConexion(baseDatos);
numFilas = new
LibrosDatos().consultarNumeroFilas(connection);
} catch (Exception excepcion)
{
```

```
throw excepcion;
}
finally
{
conexionBaseDatos.cerrarConexion(connection);
}
return numFilas;
}
public String[] consultarGeneros(BaseDatos baseDatos)
throws Exception
{
Connection connection=null;
ConexionBaseDatos conexionBaseDatos = new
ConexionBaseDatos();
List<Genero> listaGeneros = null;
String[] generos = null;
try {
connection =
conexionBaseDatos.abrirConexion(baseDatos);
listaGeneros = new
LibrosDatos().consultarGeneros(connection);
generos = new String[listaGeneros.size()];
for (int i=0; i<listaGeneros.size(); i++)
{
Genero genero = listaGeneros.get(i);
generos[i] = genero.getCodigo()+" -
"+genero.getDescripcion();
}
} catch (Exception excepcion)
{
throw excepcion;
}
finally
{
conexionBaseDatos.cerrarConexion(connection);
}
return generos;
```

```
}

public void generarPDF(List<Libro> listaLibros,
ParametrosListado parametrosListado, String
directorioCreacionPDFs) throws Exception
{
new LibrosPDF().generarPDF(listaLibros,
parametrosListado, directorioCreacionPDFs);
}

public void imprimir(List<Libro> listaLibros,
ParametrosListado parametrosListado) throws Exception
{
new LibrosImpresora().imprimir(listaLibros,
parametrosListado);
}

}
```

## IncidenciasNegocio

```
package negocio;
import datos.ActividadUsuariosDatos;
import datos.ConexionBaseDatos;
import datos.IncidenciasDatos;
import encapsuladores.ActividadUsuario;
import encapsuladores.BaseDatos;
import encapsuladores.Contexto;
import encapsuladores.ParametrosSeleccionAgrupacion;
import java.io.IOException;
import java.sql.Connection;
import java.util.HashMap;
import java.util.List;
public class IncidenciasNegocio {
public void escribirFichero(int codigo, String
mensaje, String nombreFichero, Contexto contexto)
throws IOException {
new IncidenciasDatos().escribirFichero(codigo,
mensaje, nombreFichero, contexto);
}
```

```
public void volcarFichero(BaseDatos baseDatos) throws
Exception {
IncidenciasDatos incidenciasDatos = new
IncidenciasDatos();
Connection connection = null;
ConexionBaseDatos conexionBaseDatos = new
ConexionBaseDatos();
ActividadUsuariosDatos actividadUsuariosDatos = new
ActividadUsuariosDatos();
try {
connection =
conexionBaseDatos.abrirConexion(baseDatos);
if (incidenciasDatos.comprobarExistenciaFichero())
{
List<String> listaLineas =
incidenciasDatos.leerFichero();
for (int i=0; i<listaLineas.size(); i++)
{
String lineaLeida = listaLineas.get(i);
ActividadUsuario actividadUsuario = new
ActividadUsuario();
actividadUsuario.setFechaHora(java.sql.Timestamp.value
Of(lineaLeida.substring(0, 19)));
actividadUsuario.setIdentificadorUsuario(lineaLeida.su
bstring(21, 36));
actividadUsuario.setIpCliente(lineaLeida.substring(39,
55));
// Acumulación solamente de caracteres numéricos desde
la subcadena correspondiente al código
StringBuffer stringBufferCodigo = new StringBuffer();
for (int k=0; k<lineaLeida.substring(59, 62).length(); k++)
{
if (lineaLeida.substring(59, 62).charAt(k) != ' ')
stringBufferCodigo.append(lineaLeida.substring(59,
62).charAt(k));
}
}
```

```
actividadUsuario.setCodigoActividad(Integer.parseInt(new String(stringBufferCodigo)));
actividadUsuariosDatos.insertar(connection,
actividadUsuario);
}
incidenciasDatos.eliminarFichero();
}
} catch (Exception excepcion)
{
throw excepcion;
}
finally
{
conexionBaseDatos.cerrarConexion(connection);
}
}
public HashMap<String, Integer>
consultarCodificacionActividades(BaseDatos baseDatos)
throws Exception
{
Connection connection=null;
ConexionBaseDatos conexionBaseDatos = new
ConexionBaseDatos();
HashMap<String, Integer> actividadesCodificadas =
null;
try {
connection =
conexionBaseDatos.abrirConexion(baseDatos);
actividadesCodificadas = new
ActividadUsuariosDatos().consultarCodificacionActivida
des(connection);
} catch (Exception excepcion)
{
throw excepcion;
}
finally
{
```

```
conexionBaseDatos.cerrarConexion(connection);
}
return actividadesCodificadas;
}
public List<ActividadUsuario>
consultarAplicandoCriteriosSeleccion(BaseDatos
baseDatos, ParametrosSeleccionAgrupacion
parametrosSeleccionAgrupacion) throws Exception
{
Connection connection=null;
ConexionBaseDatos conexionBaseDatos = new
ConexionBaseDatos();
List<ActividadUsuario> listaActividadUsuario = null;
try {
connection =
conexionBaseDatos.abrirConexion(baseDatos);
listaActividadUsuario = new
ActividadUsuariosDatos().consultarConCriteriosSeleccio
nAgrupacion(connection, parametrosSeleccionAgrupacion,
baseDatos);
} catch (Exception excepcion)
{
throw excepcion;
}
finally
{
conexionBaseDatos.cerrarConexion(connection);
}
return listaActividadUsuario;
}
public List<String>
consultarUsuariosConActividad(BaseDatos baseDatos)
throws Exception
{
Connection connection=null;
ConexionBaseDatos conexionBaseDatos = new
ConexionBaseDatos();
```

```
List<String> listaUsuarios = null;
try {
connection =
conexionBaseDatos.abrirConexion(baseDatos);
listaUsuarios = new
ActividadUsuariosDatos().consultarUsuariosConActividad
(connection);
} catch (Exception excepcion)
{
throw excepcion;
}
finally
{
conexionBaseDatos.cerrarConexion(connection);
}
return listaUsuarios;
}
public List<String>
consultarFechasConActividad(BaseDatos baseDatos)
throws Exception
{
Connection connection=null;
ConexionBaseDatos conexionBaseDatos = new
ConexionBaseDatos();
List<String> listaFechas = null;
try {
connection =
conexionBaseDatos.abrirConexion(baseDatos);
listaFechas = new
ActividadUsuariosDatos().consultarFechasConActividad(c
onnection, baseDatos);
} catch (Exception excepcion)
{
throw excepcion;
}
finally
{
```

```
conexionBaseDatos.cerrarConexion(connection);
}
return listaFechas;
}
}
```

## RepositorioXML

```
package datos;
import encapsuladores.BaseDatos;
import encapsuladores.ParametrosListado;
import encapsuladores.ParametrosPaginacion;
import encapsuladores.SistemaArchivos;
import excepciones.GenericaExcepcion;
import java.io.IOException;
import java.util.Iterator;
import java.util.List;
import org.jdom2.Document;
import org.jdom2.Element;
import org.jdom2.JDOMException;
import org.jdom2.input.SAXBuilder;
public class RepositorioXML {
public Object[] cargar() throws Exception {
Object[] repositorio = new Object[6];
BaseDatos baseDatos = null;
try {
Document document = new
SAXBuilder().build("xml/repositorio.xml"); // xml es
un directorio en el raiz del Proyecto
Element raiz = document.getRootElement();
// LEER BASE DE DATOS SELECCIONADA
Element basesDeDatos =
raiz.getChild("BASES_DE_DATOS");
Element baseDatosSeleccionada =
basesDeDatos.getChild("BASE_DATOS_SELECCIONADA");
List listaXMLBD =
basesDeDatos.getChildren("BASE_DATOS");

```

```
boolean encontradaBD = false;
Iterator iteratorBD = listaXMLBD.iterator();
while (iteratorBD.hasNext() && !encontradaBD)
{
Element elementBD = (Element)iteratorBD.next();
if
(elementBD.getAttributeValue("nombre").compareTo(baseD
atosSeleccionada.getText()) == 0)
{ // ENCONTRADA BD SELECCIONADA Y CARGA XML EN OBJETO
ENCAPSULADOR
encontradaBD = true;
baseDatos = new BaseDatos();
baseDatos.setNombre(elementBD.getAttributeValue("nombr
e"));
baseDatos.setClassDriver(elementBD.getChild("CLASS_DRI
VER").getText());
baseDatos.setUrlConexion(elementBD.getChild("URL_CONEX
ION").getText());
baseDatos.setUsuario(elementBD.getChild("USUARIO").get
Text());
baseDatos.setPassword(elementBD.getChild("PASSWORD").g
etText());
repositorio[0] = baseDatos;
}
}
// LEER SISTEMA ARCHIVOS
Element elementSistemaArchivos =
raiz.getChild("SISTEMA_ARCHIVOS");
SistemaArchivos sistemaArchivos = new
SistemaArchivos();
sistemaArchivos.setRutaCreacion(elementSistemaArchivos
.getChild("RUTA_CREACION").getText());
repositorio[1] = sistemaArchivos;
// LEER OPCIONES MENU
Element opcionesMenu = raiz.getChild("OPCIONES_MENU");
List listaGruposOpciones =
opcionesMenu.getChildren("GRUPO_OPCIONES");
```

```
repositorio[2] = new
String[listaGruposOpciones.size()];
repositorio[3] = new
String[listaGruposOpciones.size()][];
for (int i=0; i<listaGruposOpciones.size(); i++)
{
Element grupoOpciones = ((Element)
listaGruposOpciones.get(i));
((String [])repositorio[2])[i] =
grupoOpciones.getAttributeValue("nombre_grupo");
List listaOpciones =
grupoOpciones.getChildren("OPCION");
String[] opciones = new String[listaOpciones.size()];
for (int k=0; k<listaOpciones.size(); k++)
{
opciones[k] = ((Element)
listaOpciones.get(k)).getText();
}
((String [][])) repositorio[3])[i] = opciones;
}
// LEER PARAMETROS PAGINACIÓN
Element elementParametrosPaginacion =
raiz.getChild("PARAMETROS_PAGINACION");
ParametrosPaginacion parametrosPaginacion = new
ParametrosPaginacion();
parametrosPaginacion.setNumeroFilasPagina(Integer.pars
eInt(elementParametrosPaginacion.getChild("NUMERO_FILA
S_PAGINA").getText()));
parametrosPaginacion.setNumeroBotonesNumericos(Integer
.parseInt(elementParametrosPaginacion.getChild("NUMERO
_BOTONES_NUMERICOS").getText()));
repositorio[4] = parametrosPaginacion;
// LEER PARAMETROS LISTADO
Element elementParametrosListado =
raiz.getChild("PARAMETROS_LISTADO");
ParametrosListado parametrosListado = new
ParametrosListado();
```

```
parametrosListado.setNumeroFilasPagina(Integer.parseInt(elementParametrosListado.getChild("NUMERO_FILAS_POR_PAGINA").getText())));
repositorio[5] = parametrosListado;
} catch(JDOMException excepcion)
{ throw new GenericaExcepcion(30); }
catch(IOException excepcion)
{ throw new GenericaExcepcion(31); }
return repositorio;
}
}
```

## ConexionBaseDatos

```
package datos;
import encapsuladores.BaseDatos;
import excepciones.GenericaExcepcion;
import java.sql.Connection;
import java.sql.SQLException;
public class ConexionBaseDatos {
public Connection abrirConexion(BaseDatos baseDatos)
throws Exception
{
Connection connection = null;
try {
Class.forName(baseDatos.getClassDriver());
connection =
java.sql.DriverManager.getConnection(baseDatos.getUrlConexion(),
baseDatos.getUsuario(),
baseDatos.getPassword());
} catch (SQLException excepcion) {
throw new GenericaExcepcion(25);
}
return connection;
}
public void cerrarConexion(Connection connection)
throws GenericaExcepcion
```

```
{  
try {  
if (connection!= null)  
connection.close();  
} catch (SQLException excepcion) {  
throw new GenericaExcepcion(25);  
}  
}  
}  
}
```

## UsuariosBibliotecaDatos

```
package datos;  
import encapsuladores.Contexto;  
import excepciones.GenericaExcepcion;  
import java.sql.Connection;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
public class UsuariosBibliotecaDatos {  
public Contexto buscarPorIdUsuario(Connection  
connection, Contexto contexto) throws Exception  
{  
Contexto usuarioObtenido = null;  
ResultSet resultSet = null;  
PreparedStatement preparedStatement = null;  
try {  
String sql = “SELECT * FROM usuarios_biblioteca WHERE  
id_usuario = CAST(? AS CHAR(15));”;  
preparedStatement = connection.prepareStatement(sql);  
preparedStatement.setString(1,  
contexto.getIdentificadorUsuario());  
resultSet = preparedStatement.executeQuery();  
if (resultSet.next()) {  
usuarioObtenido = new Contexto();  
usuarioObtenido.setIdentificadorUsuario(resultSet.getString(1));  
}
```

```
usuarioObtenido.setPassword(resultSet.getString(2));
}
} catch (SQLException excepcion) {
throw new GenericaExcepcion(60);
} finally
{
if (resultSet != null) resultSet.close();
if (preparedStatement != null)
preparedStatement.close();
}
return usuarioObtenido;
}
}
```

## LibrosDatos

```
package datos;
import encapsuladores.BaseDatos;
import encapsuladores.Genero;
import encapsuladores.Libro;
import encapsuladores.LmitesListado;
import encapsuladores.ParametrosPaginacion;
import excepciones.GenericaExcepcion;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
public class LibrosDatos {
public void insertar(Connection connection, Libro
libro) throws Exception
{ PreparedStatement preparedStatement = null;
try {
String sql = "INSERT INTO libros VALUES(?,?,?,?,?,?)";
preparedStatement = connection.prepareStatement(sql);
```

```
preparedStatement.setString(1, libro.getIdLibro());
preparedStatement.setString(2, libro.getTitulo());
preparedStatement.setString(3, libro.getGenero());
preparedStatement.setDate(4, libro.getFechaEdicion());
preparedStatement.setInt(5, libro.getNumeroPaginas());
byte premiado = 0;
if (libro.isPremiado())
premiado = 1;
preparedStatement.setByte(6, premiado);
preparedStatement.executeUpdate();
} catch (SQLException excepcion) {
throw new GenericaExcepcion(50);
} finally
{
if (preparedStatement != null)
preparedStatement.close();
}
}

public void eliminar(Connection connection, Libro
libro) throws Exception
{ PreparedStatement preparedStatement = null;
try {
String sql = "DELETE FROM libros WHERE id_libro =
CAST(? AS CHAR(5))";
preparedStatement = connection.prepareStatement(sql);
preparedStatement.setString(1, libro.getIdLibro());
preparedStatement.executeUpdate();
} catch (SQLException excepcion) {
throw new GenericaExcepcion(51);
} finally
{
if (preparedStatement != null)
preparedStatement.close();
}
}

public void actualizarColumna(Connection connection,
Libro libro) throws Exception
```

```
{ PreparedStatement preparedStatement = null;
try {
String sql = "UPDATE libros SET ";
switch(libro.getColumnaActualizada())
{ case 1: sql += "titulo = ?";
break;
case 2: sql += "genero = ?";
break;
case 3: sql += "fecha_edicion = ?";
break;
case 4: sql += "numero_paginas = ?";
break;
case 5: sql += "premiado = ?";
break;
}
sql += " WHERE id_libro = CAST(? AS CHAR(5))";
PreparedStatement = connection.prepareStatement(sql);
switch(libro.getColumnaActualizada())
{ case 1: preparedStatement.setString(1,
(String)libro.getDateActualizado());
break;
case 2: preparedStatement.setString(1,
(String)libro.getDateActualizado());
break;
case 3: preparedStatement.setDate(1,
java.sql.Date.valueOf(((String)libro.getDateActualizado()).substring(6, 10) +"-"+
((String)libro.getDateActualizado()).substring(3, 5)
+"-"+
((String)libro.getDateActualizado()).substring(0,
2)));
break;
case 4: preparedStatement.setInt(1,
(Integer)libro.getDateActualizado());
break;
case 5: byte premiado = 0;
if
```

```
((Boolean)libro.getDateActualizado()).booleanValue())
preiado = 1;
PreparedStatement.setByte(1, premiado);
break;
}
PreparedStatement.setString(2, libro.getIdLibro());
PreparedStatement.executeUpdate();
} catch (SQLException excepcion) {
throw new GenericaExcepcion(52);
} finally
{
if (PreparedStatement != null)
PreparedStatement.close();
}
}

public String concatenarSeparador(boolean iniciado) {
String separador = "";
if (iniciado)
separador = ", ";
return separador;
}
public void actualizar(Connection connection, Libro
libro, int actualizaciones) throws Exception
{ PreparedStatement preparedStatement = null;
try {
boolean iniciado = false;
String sql = "UPDATE libros SET ";
for (int i=1; i<=16384; i*=2)
{ int pesoDelBit = (int) actualizaciones & i;
switch (i)
{ case 1 : if (pesoDelBit > 0)
{ sql += concatenarSeparador(iniciado);
sql += "titulo = ?";
iniciado = true;
}
break;
case 2 : if (pesoDelBit > 0)
```

```
{ sql += concatenarSeparador(iniciado);
sql += "genero = ?";
iniciado = true;
}
break;
case 4 : if (pesoDelBit > 0)
{ sql += concatenarSeparador(iniciado);
sql += "fecha_edicion = ?";
iniciado = true;
}
break;
case 8 : if (pesoDelBit > 0)
{ sql += concatenarSeparador(iniciado);
sql += "numero_paginas = ?";
iniciado = true;
}
break;
case 16 : if (pesoDelBit > 0)
{ sql += concatenarSeparador(iniciado);
sql += "premiado = ?";
iniciado = true;
}
break;
}
}
sql += " WHERE id_libro = CAST(?) AS CHAR(5))";
System.out.println(sql);
preparedStatement = connection.prepareStatement(sql);
int contadorActualizaciones = 1;
for (int i=1; i<=16384; i*=2)
{ int pesoDelBit = (int) actualizaciones & i;
switch (i)
{ case 1 : if (pesoDelBit > 0)
{ preparedStatement.setString(contadorActualizaciones,
libro.getTitulo());
contadorActualizaciones++;
}
```

```
break;
case 2 : if (pesoDelBit > 0)
{ preparedStatement.setString(contadorActualizaciones,
libro.getGenero());
contadorActualizaciones++;
}
break;
case 4 : if (pesoDelBit > 0)
{ preparedStatement.setDate(contadorActualizaciones,
libro.getFechaEdicion());
contadorActualizaciones++;
}
break;
case 8 : if (pesoDelBit > 0)
{ preparedStatement.setInt(contadorActualizaciones,
libro.getNumeroPaginas());
contadorActualizaciones++;
}
break;
case 16 : if (pesoDelBit > 0)
{ byte premiado = 0;
if (libro.isPremiado())
premiado = 1;
preparedStatement.setByte(contadorActualizaciones,
premiado);
contadorActualizaciones++;
}
break;
}
}
preparedStatement.setString(contadorActualizaciones,
libro.getIdLibro());
preparedStatement.executeUpdate();
} catch (SQLException excepcion) {
throw new GenericaExcepcion(53);
} finally
{
```

```
if (preparedStatement != null)
preparedStatement.close();
}
}
public Libro consultarPorIdLibro(Connection
connection, Libro libro) throws Exception
{
Libro libroObtenido = null;
ResultSet resultSet = null;
PreparedStatement preparedStatement = null;
try {
String sql = "SELECT id_libro, titulo, generos.codigo,
generos.descripcion, fecha_edicion, numero_paginas,
premiado FROM generos INNER JOIN libros ON
generos.codigo = libros.genero WHERE id_libro = CAST(? AS CHAR(5))";
preparedStatement = connection.prepareStatement(sql);
preparedStatement.setString(1, libro.getIdLibro());
resultSet = preparedStatement.executeQuery();
if (resultSet.next()) {
libroObtenido = new Libro();
libroObtenido.setIdLibro(resultSet.getString(1));
libroObtenido.setTitulo(resultSet.getString(2));
libroObtenido.setGenero(resultSet.getString(3));
libroObtenido.setDescripcion(resultSet.getString(4));
libroObtenido.setFechaEdicion(resultSet.getDate(5));
libroObtenido.setNumeroPaginas(resultSet.getInt(6));
byte premiado = resultSet.getByte(7);
if (premiado == 1)
libroObtenido.setPremiado(true);
else
libroObtenido.setPremiado(false);
}
} catch (SQLException excepcion) {
throw new GenericaExcepcion(54);
} finally
{
```

```
if (resultSet != null) resultSet.close();
if (preparedStatement != null)
preparedStatement.close();
}
return libroObtenido;
}
public List<Libro> consultarTodos(Connection
connection, int criterioOrdenacion, String genero,
LimitesListado limitesListado) throws Exception
{
List<Libro> listaLibros = new ArrayList();
ResultSet resultSet = null;
PreparedStatement preparedStatement = null;
try {
String sql = "SELECT id_libro, titulo, generos.codigo,
generos.descripcion, fecha_edicion, numero_paginas,
premiado FROM generos INNER JOIN libros ON
generos.codigo = libros.genero ";
if (genero != null)
sql += "WHERE generos.codigo = ? ";
if (limitesListado != null)
{
if (genero == null)
sql += "WHERE ";
else
sql += "AND ";
switch (criterioOrdenacion)
{
case 1 : sql += "id_libro >= ? AND id_libro <= ? ";
break;
case 2 : sql += "titulo >= ? AND titulo <= ? ";
break;
case 3 : sql += "genero >= ? AND genero <= ? ";
break;
}
}
switch (criterioOrdenacion)
```

```
{  
case 1 : sql += "ORDER BY id_libro";  
break;  
case 2 : sql += "ORDER BY titulo";  
break;  
case 3 : sql += "ORDER BY genero, titulo";  
break;  
}  
PreparedStatement = connection.prepareStatement(sql);  
int contadorParametros = 1;  
if (genero != null)  
{  
PreparedStatement.setString(contadorParametros,  
genero);  
contadorParametros++;  
}  
if (limitesListado != null)  
{  
PreparedStatement.setString(contadorParametros,  
limitesListado.getLimiteInferior());  
contadorParametros++;  
PreparedStatement.setString(contadorParametros,  
limitesListado.getLimiteSuperior());  
// contadorParametros++;  
}  
resultSet = preparedStatement.executeQuery();  
while (resultSet.next()) {  
Libro libro = new Libro();  
libro.setIdLibro(resultSet.getString(1));  
libro.setTitulo(resultSet.getString(2));  
libro.setGenero(resultSet.getString(3));  
libro.setDescripcion(resultSet.getString(4));  
libro.setFechaEdicion(resultSet.getDate(5));  
libro.setNumeroPaginas(resultSet.getInt(6));  
if (resultSet.getInt(7) == 1)  
libro.setPremiado(true);  
else
```

```
libro.setPremiado(false);
listaLibros.add(libro);
}
} catch (SQLException excepcion) {
throw new GenericaExcepcion(55);
} finally
{
if (resultSet != null) resultSet.close();
if (preparedStatement != null)
preparedStatement.close();
}
return listaLibros;
}

public List<Libro> consultarPagina(Connection
connection, int criterioOrdenacion,
ParametrosPaginacion parametrosPaginacion, BaseDatos
baseDatos) throws Exception
{
List<Libro> listaLibros = new ArrayList();
ResultSet resultSet = null;
PreparedStatement preparedStatement = null;
try {
String clausulaORDERBY = "";
switch (criterioOrdenacion)
{
case 1 : clausulaORDERBY = "ORDER BY id_libro";
break;
case 2 : clausulaORDERBY = "ORDER BY titulo";
break;
case 3 : clausulaORDERBY = "ORDER BY genero";
break;
}
String sql = "";
switch(baseDatos.getNombre())
{
case "Oracle": sql = "SELECT id_libro, titulo,
generos_codigo, generos_descripcion, fecha_edicion,
```

```
numero_paginas, premiado, num_fila FROM (SELECT
id_libro, titulo, generos_codigo, generos_descripcion,
fecha_edicion, numero_paginas, premiado, rownum AS
num_fila FROM (SELECT id_libro, titulo, generos.codigo
AS generos_codigo, generos.descripcion AS
generos_descripcion, fecha_edicion, numero_paginas,
premiado FROM generos INNER JOIN libros ON
generos.codigo = libros.genero “+ clausulaORDERBY +”
)) WHERE num_fila>=((?-1)*?)+1 AND num_fila<=(?*?);
break;
case “MySQL”: sql = “SELECT id_libro, titulo,
generos.codigo, generos.descripcion, fecha_edicion,
numero_paginas, premiado FROM generos INNER JOIN
libros ON generos.codigo = libros.genero “+
clausulaORDERBY +” LIMIT ?,?”;
break;
}
PreparedStatement = connection.prepareStatement(sql);
switch(baseDatos.getNombre())
{
case “Oracle”: preparedStatement.setInt(1,
parametrosPaginacion.getNumeroPagina());
preparedStatement.setInt(2,
parametrosPaginacion.getNumeroFilasPagina());
preparedStatement.setInt(3,
parametrosPaginacion.getNumeroPagina());
preparedStatement.setInt(4,
parametrosPaginacion.getNumeroFilasPagina());
break;
case “MySQL”: preparedStatement.setInt(1,
(parametrosPaginacion.getNumeroPagina() - 1) *
(parametrosPaginacion.getNumeroFilasPagina()));
preparedStatement.setInt(2,
parametrosPaginacion.getNumeroFilasPagina());
break;
}
resultSet = preparedStatement.executeQuery();
```

```
while (resultSet.next()) {
    Libro libro = new Libro();
    libro.setIdLibro(resultSet.getString(1));
    libro.setTitulo(resultSet.getString(2));
    libro.setGenero(resultSet.getString(3));
    libro.setDescripcion(resultSet.getString(4));
    libro.setFechaEdicion(resultSet.getDate(5));
    libro.setNumeroPaginas(resultSet.getInt(6));
    if (resultSet.getInt(7) == 1)
        libro.setPremiado(true);
    else
        libro.setPremiado(false);
    listaLibros.add(libro);
}
} catch (SQLException excepcion) {
    throw new GenericaExcepcion(56);
} finally
{
    if (resultSet != null) resultSet.close();
    if (preparedStatement != null)
        preparedStatement.close();
}
return listaLibros;
}

public Integer consultarNumeroFilas(Connection connection) throws Exception
{
    Integer numFilas = null;
    ResultSet resultSet = null;
    Statement statement = null;
    String sql = "SELECT COUNT(*) FROM libros";
    try {
        statement = connection.createStatement();
        resultSet = statement.executeQuery(sql);
        if (resultSet.next()) {
            numFilas = new Integer(resultSet.getInt(1));
        }
    }
```

```
        } catch (SQLException excepcion) {
        throw new GenericaExcepcion(57);
    } finally
    {
        if (resultSet != null) resultSet.close();
        if (statement != null) statement.close();
    }
    return numFilas;
}
public List<Genero> consultarGeneros(Connection
connection) throws Exception
{
List<Genero> listaGeneros = new ArrayList();
ResultSet resultSet = null;
Statement statement = null;
try {
String sql = "SELECT * FROM generos ORDER BY codigo ";
statement = connection.createStatement();
resultSet = statement.executeQuery(sql);
while (resultSet.next()) {
Genero genero = new Genero();
genero.setCodigo(resultSet.getString(1));
genero.setDescripcion(resultSet.getString(2));
listaGeneros.add(genero);
}
} catch (SQLException excepcion) {
throw new GenericaExcepcion(58);
} finally
{
        if (resultSet != null) resultSet.close();
        if (statement != null) statement.close();
}
return listaGeneros;
}
}
```

## LibrosSecuencia

```
package datos;
import encapsuladores.BaseDatos;
import excepciones.GenericaExcepcion;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
public class LibrosSecuencia {
public String consultarValorSecuencia(Connection
connection, BaseDatos baseDatos) throws Exception
{
String valorSecuencia = null;
ResultSet resultSet = null;
Statement statement = null;
String sql = "";
switch(baseDatos.getNombre())
{
case "Oracle": sql = "SELECT
REPLACE(TO_CHAR(secuencia_libros.NEXTVAL,'09999'),' ')
FROM DUAL";
break;
case "MySQL": sql = "SELECT
LPAD(FORMAT(secuencia_next_valor(\"secuencia_libros\")
,0),5,'0')";
break;
}
try {
statement = connection.createStatement();
resultSet = statement.executeQuery(sql);
if (resultSet.next()) {
valorSecuencia = resultSet.getString(1);
}
} catch (SQLException excepcion) {
throw new GenericaExcepcion(100);
} finally
{
if (resultSet != null) resultSet.close();
```

```
if (statement != null) statement.close();
}
return valorSecuencia;
}
}
```

## LibrosPDF

```
package datos;
import encapsuladores.Libro;
import encapsuladores.ParametrosListado;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.List;
import org.apache.pdfbox.pdmodel.PDDocument;
import org.apache.pdfbox.pdmodel.PDPage;
import org.apache.pdfbox.pdmodel.PDPageContentStream;
import
org.apache.pdfbox.pdmodel.encryption.AccessPermission;
import
org.apache.pdfbox.pdmodel.encryption.StandardProtectio
nPolicy;
import org.apache.pdfbox.pdmodel.font.PDFont;
import org.apache.pdfbox.pdmodel.font.PDType1Font;
import
org.apache.pdfbox.pdmodel.graphics.image.PDImageXObject;
public class LibrosPDF {
public void generarPDF(List<Libro> listaLibros,
ParametrosListado parametrosListadoPDF, String
directorioCreacionPDFs) throws IOException{
PDDocument pDDocument = new PDDocument();
Libro[] lineasPagina = new
Libro[parametrosListadoPDF.getNumeroFilasPagina()];
int numeroPagina = 1;
int contadorLineas = 0;
inicializarLineasPagina(lineasPagina);
```

```
for (int i=0; i<listaLibros.size(); i++)
{
lineasPagina[contadorLineas] = listaLibros.get(i);
contadorLineas++;
if (contadorLineas ==
parametrosListadoPDF.getNumeroFilasPagina())
{
escribirPagina(pDDocument, lineasPagina,
numeroPagina++);
inicializarLineasPagina(lineasPagina);
contadorLineas = 0;
}
}
if (contadorLineas > 0)
escribirPagina(pDDocument, lineasPagina,
numeroPagina););
String nombreArchivoPDF = directorioCreacionPDFs +
“\\Listado_libros”+ new
SimpleDateFormat(“yyyyMMddHHmmss”).format(new
java.util.Date()) + “.pdf”;
pDDocument.save(nombreArchivoPDF);
pDDocument.close();
}
private void inicializarLineasPagina(Libro[ ]
lineasPagina) {
for (int i=0; i<lineasPagina.length; i++)
lineasPagina[i] = null;
}
private void escribirPagina(PDDocument pDDocument,
Libro[ ] lineasPagina, int numeroPagina) throws
IOException {
PDPAGE pDPage = new PDPAGE();
pDDocument.addPage(pDPage);
PDPageContentStream pDPageContentStream = new
PDPageContentStream(pDDocument, pDPage);
escribirCabecera(pDDocument, pDPageContentStream);
int contadorLineas =
```

```
escribirCuerpo(pDPageContentStream, lineasPagina);
escribirPie(pDPageContentStream, numeroPagina,
contadorLineas);
pDPageContentStream.close();
}
private void escribirCabecera(PDDocument pDDocument,
PDPAGEContentStream pDPageContentStream) throws
IOException {
// LOGO
PDImageXObject imagen =
PDImageXObject.createFromFile("imgs/logo1.jpg",
pDDocument);
pDPageContentStream.drawImage(imagen, 50, 690);
escribirTexto(pDPageContentStream,
PDType1Font.TIMES_BOLD, 12, 150, 730, "BIBLIOTECA
MUNICIPAL DE VILLAR DEL MONTE");
escribirTexto(pDPageContentStream,
PDType1Font.TIMES_BOLD, 10, 460, 705, "Fecha :");
escribirTexto(pDPageContentStream,
PDType1Font.TIMES_ROMAN, 10, 500, 705, new
java.text.SimpleDateFormat("dd-MM-yyyy").format(new
java.util.Date()));
escribirLinea(pDPageContentStream, 30, 680, 580, 680);
escribirTexto(pDPageContentStream,
PDType1Font.TIMES_BOLD, 10, 52, 662, "Código");
escribirTexto(pDPageContentStream,
PDType1Font.TIMES_BOLD, 10, 93, 662, "Título");
escribirTexto(pDPageContentStream,
PDType1Font.TIMES_BOLD, 10, 382, 662, "Género");
escribirLinea(pDPageContentStream, 30, 650, 580, 650);
}
private int escribirCuerpo(PDPAGEContentStream
pDPageContentStream, Libro[] lineasPagina) throws
IOException {
int i;
for (i=0; i<lineasPagina.length; i++)
if (lineasPagina[i] != null)
```

```
escribirTexto(pDPageContentStream,
PDType1Font.COURIER, 8, 55, 635-(15*(i)),
String.format("%-8s", lineasPagina[i].getIdLibro()) +
String.format("%-60s", lineasPagina[i].getTitulo()) +
lineasPagina[i].getGenero() + " - " +
String.format("%-32s",
lineasPagina[i].getDescripcion()));
return i;
}
private void escribirPie(PDPageContentStream
pDPageContentStream, int numeroPagina, int
contadorLineas) throws IOException {
escribirLinea(pDPageContentStream, 30, 640-(15*
(contadorLineas)), 580, 640-(15*(contadorLineas)));
escribirTexto(pDPageContentStream,
PDType1Font.TIMES_ROMAN, 8, 520, 625-(15*
(contadorLineas)), "pág. "+numeroPagina);
}
private void escribirTexto(PDPageContentStream
pDPageContentStream, PDFont pdFont, float sizeFuente,
float inicioH, float inicioV, String texto) throws
IOException {
pDPageContentStream.beginText();
pDPageContentStream.setFont(pdFont, sizeFuente);
pDPageContentStream.newLineAtOffset(inicioH, inicioV);
pDPageContentStream.showText(texto);
pDPageContentStream.endText();
}
private void escribirLinea(PDPageContentStream
pDPageContentStream, float inicioH, float inicioV,
float finH, float finV) throws IOException {
pDPageContentStream.moveTo(inicioH, inicioV);
pDPageContentStream.lineTo(finH, finV);
pDPageContentStream.stroke();
}
private void protegerDocumento(PDDocument pDDocument)
throws IOException{
```

```
AccessPermission accessPermission = new
AccessPermission();
accessPermission.setCanPrint(false);
accessPermission.setCanPrintDegraded(false);
accessPermission.setCanExtractContent(false);
accessPermission.setCanExtractForAccessibility(false);
accessPermission.setCanFillInForm(true);
accessPermission.setCanModify(false);
accessPermission.setReadOnly();
// PASSWORD PASSWORD
// PROPIETARIO USUARIO
StandardProtectionPolicy standardProtectionPolicy =
new StandardProtectionPolicy("123", "12345",
accessPermission);
standardProtectionPolicy.setEncryptionKeyLength(128);
standardProtectionPolicy.setPreferAES(true);
standardProtectionPolicy.setPermissions(accessPermission);
pDDocument.protect(standardProtectionPolicy);
}
}
```

## LibrosImpresora

```
package datos;
import encapsuladores.Libro;
import encapsuladores.ParametrosListado;
import java.awt.print.Book;
import java.awt.print.PageFormat;
import java.awt.print.Paper;
import java.awt.print.PrinterException;
import java.awt.print.PrinterJob;
import java.util.List;
public class LibrosImpresora {
public void imprimir(List<Libro> listaLibros,
ParametrosListado parametrosListado) throws
PrinterException {
```

```
PrinterJob printerJob = PrinterJob.getPrinterJob();
Book book = new Book();
PageFormat pageFormat = new PageFormat();
// pageFormat.setOrientation(PageFormat.LANDSCAPE); // CASO DE REQUERIRSE FORMATO HORIZONTAL
Paper paper = new Paper();
// SI SE REQUIRIESE UN MAYOR TAMAÑO DE PAPEL, APLICARIAMOS:
// paper.setSize(750.0,850.0);
paper.setImageableArea(1, 1, 610, 790); // ESTABLECEMOS ESTOS VALORES DADO QUE EL TAMAÑO POR DEFECTO DEL PAPEL ES: 612, 792
pageFormat.setPaper(paper);
Libro[] lineasPagina = new Libro[parametrosListado.getNumeroFilasPagina()];
int numeroPagina = 1;
int contadorLineas = 0;
inicializarLineasPagina(lineasPagina);
for (int i=0; i<listaLibros.size(); i++)
{
    lineasPagina[contadorLineas] = listaLibros.get(i);
    contadorLineas++;
    if (contadorLineas == parametrosListado.getNumeroFilasPagina())
    {
        book.append(new ImpresorPagina(lineasPagina,
            numeroPagina++), pageFormat);
        inicializarLineasPagina(lineasPagina);
        contadorLineas = 0;
    }
}
if (contadorLineas > 0)
    book.append(new ImpresorPagina(lineasPagina,
        numeroPagina++), pageFormat);
printerJob.setPageable(book);
if (printerJob.printDialog())
    printerJob.print();
```

```
}

private void inicializarLineasPagina(Libro[] lineasPagina) {
    for (int i=0; i<lineasPagina.length; i++)
        lineasPagina[i] = null;
}
```

## ImpresorPagina

```
package datos;
import encapsuladores.Libro;
import java.awt.Component;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Image;
import java.awt.MediaTracker;
import java.awt.Toolkit;
import java.awt.print.PageFormat;
import java.awt.print.Printable;
public class ImpresorPagina extends Component
implements Printable {
private Libro[] lineasPagina;
private int numeroPagina;
public ImpresorPagina(Libro[] lineasPaginaTransferido,
int numeroPagina) {
this.numeroPagina = numeroPagina;
lineasPagina = new
Libro[lineasPaginaTransferido.length];
for (int i=0; i<lineasPagina.length; i++)
lineasPagina[i] = lineasPaginaTransferido[i];
}
public int print(Graphics g, PageFormat pageFormat,
int n) {
// CABECERA DEL LISTADO
MediaTracker mediaTracker = new MediaTracker(this);
Image image =
```

```
Toolkit.getDefaultToolkit().getImage("imgs/logo1.jpg")
;
mediaTracker.addImage(image,0);
try{
mediaTracker.waitForAll();
} catch (Exception exception)
{ System.out.println("Error en metodo print de
ImpresorPagina "+exception.getMessage()); }
g.drawImage(image, 70, 30, this);
g.setFont(new Font("TimesRoman", Font.PLAIN +
Font.BOLD, 15));
g.drawString("BIBLIOTECA MUNICIPAL DE VILLAR DEL
MONTE", 160, 55);
g.setFont(new Font("TimesRoman", Font.PLAIN +
Font.BOLD, 10));
g.drawString("Fecha : ", 455, 90);
g.setFont(new Font("TimesRoman", Font.PLAIN, 10));
g.drawString(new java.text.SimpleDateFormat("dd-MM-
yyyy").format(new java.util.Date()), 495, 90);
g.drawLine(50, 120, 580, 120);
g.setFont(new Font("TimesRoman",Font.BOLD,12));
g.drawString("Código", 50, 135);
g.drawString("Título", 105, 135);
g.drawString("Género", 415, 135);
g.drawLine(50, 140, 580, 140);
// CUERPO DEL LISTADO
g.setFont(new Font("TimesRoman",Font.PLAIN,10));
int i;
for (i=0; i<lineasPagina.length; i++)
if (lineasPagina[i] != null)
{
g.drawString(lineasPagina[i].getIdLibro(), 50, 160+
(i*15));
g.drawString(lineasPagina[i].getTitulo(), 105, 160+
(i*15));
g.drawString(lineasPagina[i].getGenero() + " - " +
lineasPagina[i].getDescripcion(), 415, 160+(i*15));
```

```

}
// PIE DEL LISTADO
g.setFont(new Font("TimesRoman", Font.BOLD,12));
g.drawLine(50, 160+(i*15) ,580, 160+(i*15));
g.setFont(new Font("Dialog", Font.ITALIC,10));
g.drawString("pág. "+numeroPagina, 520, 180+(i*15));
return (PAGE_EXISTS);
}
}

```

## ActividadUsuariosDatos

```

package datos;
import encapsuladores.ActividadUsuario;
import encapsuladores.BaseDatos;
import encapsuladores.ParametrosSeleccionAgrupacion;
import excepciones.GenericaExcepcion;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
public class ActividadUsuariosDatos {
public void insertar(Connection connection,
ActividadUsuario actividadUsuario) throws Exception
{ PreparedStatement preparedStatement = null;
try {
String sql = "INSERT INTO actividad_usuarios
VALUES(?, ?, ?, ?)";
preparedStatement = connection.prepareStatement(sql);
preparedStatement.setTimestamp(1,
actividadUsuario.getFechaHora());
preparedStatement.setString(2,
actividadUsuario.getIdentificadorUsuario());

```

```
preparedStatement.setString(3,
actividadUsuario.getIpCliente());
preparedStatement.setInt(4,
actividadUsuario.getCodigoActividad());
preparedStatement.executeUpdate();
} catch (SQLException excepcion) {
throw new GenericaExcepcion(40);
} finally
{
if (preparedStatement != null)
preparedStatement.close();
}
}

public HashMap<String, Integer>
consultarCodificacionActividades(Connection
connection) throws Exception
{
HashMap<String, Integer> actividadesCodificadas = new
HashMap();
ResultSet resultSet = null;
Statement statement = null;
try {
String sql = "SELECT descripcion, codigo FROM
codificacion_actividades";
statement = connection.createStatement();
resultSet = statement.executeQuery(sql);
while (resultSet.next()) {
actividadesCodificadas.put(resultSet.getString(1), new
Integer(resultSet.getInt(2)));
}
} catch (SQLException excepcion) {
throw new GenericaExcepcion(55);
} finally
{
if (resultSet != null) resultSet.close();
if (statement != null) statement.close();
}
```

```
return actividadesCodificadas;
}
public List<String>
consultarUsuariosConActividad(Connection connection)
throws Exception
{
List<String> listaUsuarios = new ArrayList();
ResultSet resultSet = null;
Statement statement = null;
try {
String sql = "SELECT DISTINCT id_usuario FROM
actividad_usuarios ORDER BY id_usuario";
statement = connection.createStatement();
resultSet = statement.executeQuery(sql);
while (resultSet.next()) {
listaUsuarios.add(new String(resultSet.getString(1)));
}
} catch (SQLException excepcion) {
throw new GenericaExcepcion(41);
} finally
{
if (resultSet != null) resultSet.close();
if (statement != null) statement.close();
}
return listaUsuarios;
}
public List<String>
consultarFechasConActividad(Connection connection,
BaseDatos baseDatos) throws Exception
{
List<String> listaFechas = new ArrayList();
ResultSet resultSet = null;
Statement statement = null;
try {
String sql = "";
switch(baseDatos.getNombre())
{
```

```
case "Oracle": sql = "SELECT DISTINCT
TO_CHAR(fecha_hora,'YYYY-MM-DD') FROM
actividad_usuarios ORDER BY TO_CHAR(fecha_hora,'YYYY-
MM-DD')";
break;
case "MySQL": sql = "SELECT DISTINCT
DATE_FORMAT(fecha_hora, '%Y-%m-%d') FROM
actividad_usuarios ORDER BY DATE_FORMAT(fecha_hora,
'%Y-%m-%d')";
break;
}
statement = connection.createStatement();
resultSet = statement.executeQuery(sql);
while (resultSet.next()) {
listaFechas.add(new String(resultSet.getString(1)));
}
} catch (SQLException excepcion) {
throw new GenericaExcepcion(42);
} finally
{
if (resultSet != null) resultSet.close();
if (statement != null) statement.close();
}
return listaFechas;
}
private boolean
detectarExistenciaAgrupaciones(ParametrosSeleccionAgrupacion parametrosSeleccionAgrupacion) {
boolean conAgrupaciones = false;
for (int i=0;
i<parametrosSeleccionAgrupacion.getCriteriaosAgrupacion().length; i++)
{
if
(parametrosSeleccionAgrupacion.isCriteriaoAgrupacion(i)
)
conAgrupaciones = true;
```

```
}

return conAgrupaciones;
}
private String
construirExpresionesADevolver(ParametrosSeleccionAgrupacion parametrosSeleccionAgrupacion, BaseDatos
baseDatos)
{
String[][] expresionesADevolver = {{“id_usuario”,
“TO_CHAR(fecha_hora,’YYYY-MM-DD’)”, “codificacion_actividades.codigo”, “descripcion”},
{“id_usuario”, “DATE_FORMAT(fecha_hora, ‘%Y-%m-%d’)”, “codificacion_actividades.codigo”, “descripcion”}}
};

int indiceBD = 0;
switch(baseDatos.getNombre())
{
case “Oracle”: indiceBD = 0;
break;
case “MySQL”: indiceBD = 1;
break;
}
StringBuffer construccionSQL = new StringBuffer(120);
boolean conAgrupaciones =
detectarExistenciaAgrupaciones(parametrosSeleccionAgrupacion);
int contadorExpresiones = 0;
for (int i=0;
i<(expresionesADevolver[indiceBD].length-1); i++ )
{
if
(parametrosSeleccionAgrupacion.isCriterioAgrupacion(i)
|| !conAgrupaciones)
{
if (contadorExpresiones > 0)
construccionSQL.append(“,”);
construccionSQL.append(expresionesADevolver[indiceBD]
```

```
[i]);
contadorExpresiones++;
}
}
if
(parametrosSeleccionAgrupacion.isCriterioAgrupacion((parametrosSeleccionAgrupacion.getCriteria
```

```
break;
}
StringBuffer construccionSQL = new StringBuffer(120);
int contadorSelecciones = 0;
for (int i=0; i<condicionesConsulta[indiceBD].length;
i++)
{
if
(parametrosSeleccionAgrupacion.getCriteriaSeleccion(i)
!= null)
{
construccionSQL.append(obtenerOperador(contadorSelecciones));
construccionSQL.append(condicionesConsulta[indiceBD]
[i]);
contadorSelecciones++;
}
}
return new String(construccionSQL);
}
private String obtenerOperador(int
contadorSelecciones)
{
String operadorDevuelto="";
if (contadorSelecciones == 0)
operadorDevuelto = " WHERE ";
else
operadorDevuelto = " AND ";
return operadorDevuelto;
}
private String
construirClausulaGROUPBY(ParametrosSeleccionAgrupacion
parametrosSeleccionAgrupacion, BaseDatos baseDatos)
{
String[][] expresionesADevolver = {{“id_usuario”,
“TO_CHAR(fecha_hora,’YYYY-MM-DD’)”,
“codificacion_actividades.codigo”, “descripcion”},
```

```
{"id_usuario", "DATE_FORMAT(fecha_hora, '%Y-%m-%d')",
"codificacion_actividades.codigo", "descripcion"}
};

int indiceBD = 0;
switch(baseDatos.getNombre())
{
case "Oracle": indiceBD = 0;
break;
case "MySQL": indiceBD = 1;
break;
}
StringBuffer construccionSQL = new StringBuffer(120);
boolean conAgrupaciones =
detectarExistenciaAgrupaciones(parametrosSeleccionAgrupacion);
if (conAgrupaciones)
{
boolean ubicarSeparador = false;
construccionSQL.append(" GROUP BY ");
for (int i=0;
i<(expresionesADevolver[indiceBD].length-1); i++ )
{
if
(parametrosSeleccionAgrupacion.isCriterioAgrupacion(i))
{
if (i>0 && ubicarSeparador)
construccionSQL.append(",");
construccionSQL.append(expresionesADevolver[indiceBD]
[i]);
ubicarSeparador = true;
}
}
if
(parametrosSeleccionAgrupacion.isCriterioAgrupacion((parametrosSeleccionAgrupacion.getCriteriaosAgrupacion().length-1)))
```

```
{  
    construccionSQL.append(",");  
    construccionSQL.append(expresionesADevolver[indiceBD]  
        [(expresionesADevolver[indiceBD].length-1)]);  
};  
}  
return new String(construccionSQL);  
}  
private String  
construirClausulaORDERBY(ParametrosSeleccionAgrupacion  
parametrosSeleccionAgrupacion, BaseDatos baseDatos)  
{  
    String[][] expresionesADevolver = {{“id_usuario”,  
“TO_CHAR(fecha_hora,’YYYY-MM-DD’)”,  
“codificacion_actividades.codigo”},  
    {“id_usuario”, “DATE_FORMAT(fecha_hora, ‘%Y-%m-%d’)”,  
“codificacion_actividades.codigo”}}  
};  
int indiceBD = 0;  
switch(baseDatos.getNombre())  
{  
    case “Oracle”: indiceBD = 0;  
    break;  
    case “MySQL”: indiceBD = 1;  
    break;  
}  
StringBuffer construccionSQL = new StringBuffer(120);  
construccionSQL.append(“ ORDER BY ”);  
boolean conAgrupaciones =  
detectarExistenciaAgrupaciones(parametrosSeleccionAgru  
pacion);  
int contadorExpresiones = 0;  
for (int i=0;  
i<(expresionesADevolver[indiceBD].length); i++ )  
{  
if  
(parametrosSeleccionAgrupacion.isCriteriaoAgrupacion(i)
```

```
|| !conAgrupaciones)
{
if (contadorExpresiones > 0)
construccionSQL.append(",");
construccionSQL.append(expresionesADevolver[indiceBD]
[i]);
contadorExpresiones++;
}
}
return new String(construccionSQL);
}
public List<ActividadUsuario>
consultarConCriteriosSeleccionAgrupacion(Connection
connection, ParametrosSeleccionAgrupacion
parametrosSeleccionAgrupacion, BaseDatos baseDatos)
throws Exception
{
List<ActividadUsuario> listaActividadUsuario = new
ArrayList();
ResultSet resultSet = null;
PreparedStatement preparedStatement = null;
String sql = "SELECT
"+construirExpresionesADevolver(parametrosSeleccionAgr
upacion, baseDatos)+" FROM codificacion_actividades
INNER JOIN actividad_usuarios ON
codificacion_actividades.codigo =
actividad_usuarios.codigo " +
construirClausulaWHERE(parametrosSeleccionAgrupacion,
baseDatos) +
construirClausulaGROUPBY(parametrosSeleccionAgrupacion
, baseDatos) +
construirClausulaORDERBY(parametrosSeleccionAgrupacion
, baseDatos) ;
System.out.println(sql);
try {
preparedStatement = connection.prepareStatement(sql);
int contadorSecciones = 1;
```

```
for (int i=0;
i<parametrosSeleccionAgrupacion.getCriteriaosSeleccion(
).length; i++)
{
if
(parametrosSeleccionAgrupacion.getCriteriaoSeleccion(i)
!= null)
{
PreparedStatement.setString(contadorSelecciones,
parametrosSeleccionAgrupacion.getCriteriaoSeleccion(i))
;
contadorSelecciones++;
}
}
resultSet = preparedStatement.executeQuery();
boolean conAgrupaciones = false;
for (int i=0;
i<parametrosSeleccionAgrupacion.getCriteriaosAgrupacion
().length; i++)
{
if
(parametrosSeleccionAgrupacion.isCriteriaoAgrupacion(i)
)
conAgrupaciones = true;
}
ActividadUsuario actividadUsuario = null;
while (resultSet.next()) {
int contadorCriteriaosAgrupacion = 0;
int contadorColumnasDevueltas = 1;
actividadUsuario = new ActividadUsuario();
if
(parametrosSeleccionAgrupacion.isCriteriaoAgrupacion(co
ntadorCriteriaosAgrupacion) || !conAgrupaciones)
{
actividadUsuario.setIdentificadorUsuario(resultSet.get
String(contadorColumnasDevueltas));
contadorColumnasDevueltas++;
}
```

```
}

contadorCriteriosAgrupacion++;
if
(parametrosSeleccionAgrupacion.isCriterioAgrupacion(co
ntadorCriteriosAgrupacion) || !conAgrupaciones)
{
actividadUsuario.setCadenaFechaHora(resultSet.getString(contadorColumnasDevueltas));
contadorColumnasDevueltas++;
}
contadorCriteriosAgrupacion++;
if
(parametrosSeleccionAgrupacion.isCriterioAgrupacion(co
ntadorCriteriosAgrupacion) || !conAgrupaciones)
{
actividadUsuario.setCodigoActividad(resultSet.getInt(c
ontadorColumnasDevueltas));
contadorColumnasDevueltas++;
actividadUsuario.setDescripcion(resultSet.getString(co
ntadorColumnasDevueltas));
contadorColumnasDevueltas++;
}
if (conAgrupaciones)
actividadUsuario.setNumeroIncidencias(resultSet.getInt
(contadorColumnasDevueltas));
listaActividadUsuario.add(actividadUsuario);
}
} catch (SQLException excepcion) {
throw new GenericaExcepcion(43);
} finally
{
if (resultSet != null) resultSet.close();
if (preparedStatement != null)
preparedStatement.close();
}
return listaActividadUsuario;
}
```

```
}
```

## IncidenciasDatos

```
package datos;
import encapsuladores.Contexto;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.List;
public class IncidenciasDatos {
    public void escribirFichero(int codigo, String
        mensaje, String nombreFichero, Contexto contexto)
        throws IOException {
        File file = new File("log/" + nombreFichero + ".txt");
        if (!file.exists())
            file.createNewFile();
        FileWriter fileWriter = new FileWriter(file, true);
        PrintWriter printWriter = new PrintWriter(fileWriter);
        String identificadorUsuario = " ";
        if (contexto.getIdentificadorUsuario() != null)
            identificadorUsuario =
                contexto.getIdentificadorUsuario();
        printWriter.println(new SimpleDateFormat("yyyy-MM-dd
            HH:mm:ss").format(contexto.getFechaHora()) +
            String.format("%-18s", identificadorUsuario) +
            String.format("%-18s", contexto.getIpCliente()) +
            String.format("%5s", codigo) +
            "+String.format("%-110s",mensaje));
        if (printWriter != null)
            printWriter.close();
        if (fileWriter != null)
```

```
fileWriter.close();
}
public List<String> leerFichero() throws IOException {
List<String> listaLineas = new ArrayList();
String cadenaLeida;
FileReader fileReader = new FileReader(new
File("log/actividad.txt"));
BufferedReader bufferedReader = new
BufferedReader(fileReader);
while ((cadenaLeida = bufferedReader.readLine()) !=
null) {
listaLineas.add(cadenaLeida);
}
if (fileReader != null)
fileReader.close();
return listaLineas;
}
public boolean comprobarExistenciaFichero() {
boolean existeFichero = false;
File file = new File("log/actividad.txt");
if (file.exists())
existeFichero = true;
return existeFichero;
}
public void eliminarFichero() throws IOException {
new File("log/actividad.txt").delete();
}
}
```

## SistemaArchivos

```
package encapsuladores;
public class SistemaArchivos {
private String rutaCreacion;
public String getRutaCreacion() {
return rutaCreacion;
}
```

```
public void setRutaCreacion(String rutaCreacion) {  
    this.rutaCreacion = rutaCreacion;  
}  
}
```

## Contexto

```
package encapsuladores;  
import java.util.Date;  
public class Contexto {  
    private String identificadorUsuario;  
    private String password;  
    private Date fechaHora;  
    private String ipCliente;  
    private int codigoActividad;  
    public Contexto() {  
        fechaHora = new Date();  
    }  
    public Contexto(String identificadorUsuario, String  
    password, String ipCliente) {  
        this.identificadorUsuario = identificadorUsuario;  
        this.password = password;  
        this.ipCliente = ipCliente;  
        fechaHora = new Date();  
    }  
    public String getIdentificadorUsuario() {  
        return identificadorUsuario;  
    }  
    public void setIdentificadorUsuario(String  
    identificadorUsuario) {  
        this.identificadorUsuario = identificadorUsuario;  
    }  
    public String getPassword() {  
        return password;  
    }  
    public void setPassword(String password) {  
        this.password = password;
```

```
}

public Date getFechaHora() {
    return fechaHora;
}

public void setFechaHora(Date fechaHora) {
    this.fechaHora = fechaHora;
}

public String getIpCliente() {
    return ipCliente;
}

public void setIpCliente(String ipCliente) {
    this.ipCliente = ipCliente;
}
```

## BaseDatos

```
package encapsuladores;
public class BaseDatos {
    private String nombre;
    private String classDriver;
    private String urlConexion;
    private String usuario;
    private String password;
    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getClassDriver() {
        return classDriver;
    }

    public void setClassDriver(String classDriver) {
        this.classDriver = classDriver;
    }

    public String getUrlConexion() {
```

```
return urlConexion;
}
public void setUrlConexion(String urlConexion) {
this.urlConexion = urlConexion;
}
public String getUsuario() {
return usuario;
}
public void setUsuario(String usuario) {
this.usuario = usuario;
}
public String getPassword() {
return password;
}
public void setPassword(String password) {
this.password = password;
}
}
```

## Genero

```
package encapsuladores;
public class Genero {
private String codigo;
private String descripcion;
public String getCodigo() {
return codigo;
}
public void setCodigo(String codigo) {
this.codigo = codigo;
}
public String getDescripcion() {
return descripcion;
}
public void setDescripcion(String descripcion) {
this.descripcion = descripcion;
}
}
```

```
}
```

## Libro

```
package encapsuladores;
import java.sql.Date;
public class Libro {
private String idLibro;
private String titulo;
private String genero;
private String descripcion;
private Date fechaEdicion;
private int numeroPaginas;
private boolean premiado;
private Object datoActualizado;
private int columnaActualizada;
public String getIdLibro() {
return idLibro;
}
public void setIdLibro(String idLibro) {
this.idLibro = idLibro;
}
public String getTitulo() {
return titulo;
}
public void setTitulo(String titulo) {
this.titulo = titulo;
}
public String getGenero() {
return genero;
}
public void setGenero(String genero) {
this.genero = genero;
}
public String getDescripcion() {
return descripcion;
}
```

```
public void setDescripcion(String descripcion) {  
    this.descripcion = descripcion;  
}  
public Date getFechaEdicion() {  
    return fechaEdicion;  
}  
public void setFechaEdicion(Date fechaEdicion) {  
    this.fechaEdicion = fechaEdicion;  
}  
public int getNumeroPaginas() {  
    return numeroPaginas;  
}  
public void setNumeroPaginas(int numeroPaginas) {  
    this.numeroPaginas = numeroPaginas;  
}  
public boolean isPremiado() {  
    return premiado;  
}  
public void setPremiado(boolean premiado) {  
    this.premiado = premiado;  
}  
public Object datoActualizado() {  
    return datoActualizado;  
}  
public void setDatoActualizado(Object datoActualizado)  
{  
    this.datoActualizado = datoActualizado;  
}  
public int columnaActualizada() {  
    return columnaActualizada;  
}  
public void setColumnaActualizada(int  
    columnaActualizada) {  
    this.columnaActualizada = columnaActualizada;  
}  
}
```

## ActividadUsuario

```
package encapsuladores;
import java.sql.Timestamp;
public class ActividadUsuario {
private Timestamp fechaHora;
private String cadenaFechaHora;
private String identificadorUsuario;
private String ipCliente;
private int codigoActividad;
private String descripcion;
private int numeroIncidencias;
public Timestamp getFechaHora() {
return fechaHora;
}
public void setFechaHora(Timestamp fechaHora) {
this.fechaHora = fechaHora;
}
public String getCadenaFechaHora() {
return cadenaFechaHora;
}
public void setCadenaFechaHora(String cadenaFechaHora)
{
this.cadenaFechaHora = cadenaFechaHora;
}
public String getIdentificadorUsuario() {
return identificadorUsuario;
}
public void setIdentificadorUsuario(String
identificadorUsuario) {
this.identificadorUsuario = identificadorUsuario;
}
public String getIpCliente() {
return ipCliente;
}
public void setIpCliente(String ipCliente) {
this.ipCliente = ipCliente;
}
```

```
public int getCodigoActividad() {
    return codigoActividad;
}
public void setCodigoActividad(int codigoActividad) {
    this.codigoActividad = codigoActividad;
}
public String getDescripcion() {
    return descripcion;
}
public void setDescripcion(String descripcion) {
    this.descripcion = descripcion;
}
public int getNumeroIncidentes() {
    return numeroIncidentes;
}
public void setNumeroIncidentes(int
    numeroIncidentes) {
    this.numeroIncidentes = numeroIncidentes;
}
```

## LímitesListado

```
package encapsuladores;
public class LímitesListado {
    private String límiteInferior;
    private String límiteSuperior;
    public String getLímiteInferior() {
        return límiteInferior;
    }
    public void setLímiteInferior(String límiteInferior) {
        this.límiteInferior = límiteInferior;
    }
    public String getLímiteSuperior() {
        return límiteSuperior;
    }
    public void setLímiteSuperior(String límiteSuperior) {
```

```
this.limiteSuperior = limiteSuperior;
}
}
```

## ParametrosPaginacion

```
package encapsuladores;
public class ParametrosPaginacion {
private int numeroFilasPagina;
private int numeroBotonesNumericos;
private int numeroPagina;
public int getNumeroFilasPagina() {
return numeroFilasPagina;
}
public void setNumeroFilasPagina(int
numeroFilasPagina) {
this.numeroFilasPagina = numeroFilasPagina;
}
public int getNumeroBotonesNumericos() {
return numeroBotonesNumericos;
}
public void setNumeroBotonesNumericos(int
numeroBotonesNumericos) {
this.numeroBotonesNumericos = numeroBotonesNumericos;
}
public int getNumeroPagina() {
return numeroPagina;
}
public void setNumeroPagina(int numeroPagina) {
this.numeroPagina = numeroPagina;
}
}
```

## ParametrosListado

```
package encapsuladores;
public class ParametrosListado {
private int numeroFilasPagina;
```

```
public int getNumeroFilasPagina() {
    return numeroFilasPagina;
}
public void setNumeroFilasPagina(int
    numeroFilasPagina) {
    this.numeroFilasPagina = numeroFilasPagina;
}
}
```

## ParametrosSeleccionAgrupacion

```
package encapsuladores;
public class ParametrosSeleccionAgrupacion {
    private String[] criteriosSeleccion = {null, null,
        null};
    private boolean[] criteriosAgrupacion = new
        boolean[3];
    public String getCriterioSeleccion(int componente) {
        return criteriosSeleccion[componente];
    }
    public void setCriterioSeleccion(String
        criterioSeleccion, int componente) {
        this.criteriosSeleccion[componente] =
            criterioSeleccion;
    }
    public String[] getCriteriosSeleccion() {
        return criteriosSeleccion;
    }
    public void setCriteriosSeleccion(String[]
        criteriosSeleccion) {
        this.criteriosSeleccion = criteriosSeleccion;
    }
    public boolean isCriterioAgrupacion(int componente) {
        return criteriosAgrupacion[componente];
    }
    public void setCriterioAgrupacion(boolean
        criterioAgrupacion, int componente) {
```

```
        this.criteriosAgrupacion[componente] =  
        criterioAgrupacion;  
    }  
    public boolean[] getCriteriosAgrupacion() {  
        return criteriosAgrupacion;  
    }  
    public void setCriteriosAgrupacion(boolean[]  
        criteriosAgrupacion) {  
        this.criteriosAgrupacion = criteriosAgrupacion;  
    }  
}
```

## GenericaExcepcion

```
package excepciones;  
public class GenericaExcepcion extends Exception{  
    private int codigoError;  
    public GenericaExcepcion (int codigoError){  
        this.codigoError = codigoError;  
    }  
    public int getCodigoError(){  
        return codigoError;  
    }  
}
```

## Filtros

```
package utilities;  
import excepciones.GenericaExcepcion;  
import javax.swing.JList;  
public class Filtros {  
    public static void filtrarNumeroPaginasLibro(int  
        numeroPaginas) throws GenericaExcepcion {  
        if (numeroPaginas < 1 || numeroPaginas > 2000)  
            throw new GenericaExcepcion(16);  
    }  
    public static void filtrarFecha(String cadenaFecha)  
        throws GenericaExcepcion {
```

```
int codigoError = 0;
int dia = 0;
int mes = 0;
int año = 0;
if (cadenaFecha.length() == 10 &&
cadenaFecha.charAt(2) == '-' && cadenaFecha.charAt(5)
== '-')
{
try {
dia = Integer.parseInt(cadenaFecha.substring(0, 2));
mes = Integer.parseInt(cadenaFecha.substring(3, 5));
año = Integer.parseInt(cadenaFecha.substring(6,
cadenaFecha.length())));
} catch (NumberFormatException excepcion)
{ códigoError |= 1; } // dígito no numérico en fecha
}
else
{
códigoError |= 1; // formato fecha erróneo
}
if (códigoError == 0)
{
if (Integer.toString(año).length() != 4) // año
erróneo
códigoError |= 2;
if (mes < 1 || mes > 12) // mes erróneo
códigoError |= 4;
if (dia < 1 || dia > getDiaMaximoMes(mes, año)) // dia
erróneo
códigoError |= 8;
}
System.out.println("código error fecha "+códigoError);
if (códigoError > 0)
throw new GenericaExcepcion(códigoError);
}
private static int getDiaMaximoMes(int mes, int año) {
int diaMaximoMes = 31;
```

```
switch(mes)
{ case 1:
case 3:
case 5:
case 7:
case 8:
case 10:
case 12: diaMaximoMes = 31;
break;
case 4:
case 6:
case 9:
case 11: diaMaximoMes = 30;
break;
case 2: if (((año % 4 == 0) && !(año % 100 == 0)) ||
(año % 400 == 0))
diaMaximoMes = 29;
else
diaMaximoMes = 28;
break;
}
return diaMaximoMes;
}

public static void filtrarDatosNulos(String[]
datosAFiltrar) throws GenericaExcepcion {
boolean existenDatosNulos = false;
for (int i=0; i<datosAFiltrar.length; i++)
{
if (datosAFiltrar[i].compareTo("") == 0)
existenDatosNulos = true;
}
if (existenDatosNulos)
throw new GenericaExcepcion(17);
}

public static void filtrarSeleccionJList(JList jList)
throws GenericaExcepcion {
boolean noExisteFilaSeleccionada = false;
```

```
if (jList.isEmpty())
noExisteFilaSeleccionada = true;
if (noExisteFilaSeleccionada)
throw new GenericaExcepcion(18);
}
}
```

El contenido del fichero repositorio.xml, presente en el directorio xml del proyecto, y que actúa como repositorio centralizado de parámetros relevantes de la aplicación es:

### Contenido del fichero repositorio.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<REPOSITORIO>
<BASES_DE_DATOS>
<BASE_DATOS_SELECCIONADA>MySQL</BASE_DATOS_SELECCIONADA>
<BASE_DATOS nombre="Oracle">
<CLASS_DRIVER>oracle.jdbc.driver.OracleDriver</CLASS_DRIVER>
<URL_CONEXION>jdbc:oracle:thin:@localhost:1521:ORCL</URL_CONEXION>
<USUARIO>scott</USUARIO>
<PASSWORD>tiger</PASSWORD>
</BASE_DATOS>
<BASE_DATOS nombre="MySQL">
<CLASS_DRIVER>com.mysql.jdbc.Driver</CLASS_DRIVER>
<URL_CONEXION>jdbc:mysql://localhost:3306/biblioteca</URL_CONEXION>
<USUARIO>root</USUARIO>
<PASSWORD></PASSWORD>
</BASE_DATOS>
</BASES_DE_DATOS>
<SISTEMA_ARCHIVOS>
<RUTA_CREACION>D:\\PDFs_creados\\</RUTA_CREACION>
</SISTEMA_ARCHIVOS>
```

```

<OPCIONES_MENU>
<GRUPO_OPCIONES nombre_grupo="Conexión-Desconexión">
<OPCION>Conexion</OPCION>
<OPCION>Desconexion</OPCION>
</GRUPO_OPCIONES>
<GRUPO_OPCIONES nombre_grupo="Mantenimiento">
<OPCION>Vista Formulario</OPCION>
<OPCION>Vista Unica Tabla</OPCION>
<OPCION>Vista Paginada Tabla</OPCION>
<OPCION>Vista Arbol</OPCION>
</GRUPO_OPCIONES>
<GRUPO_OPCIONES nombre_grupo="Actividad">
<OPCION>Estadisticas Actividad</OPCION>
<OPCION>Volcado a BD</OPCION>
</GRUPO_OPCIONES>
<GRUPO_OPCIONES nombre_grupo="Generar documentos">
<OPCION>Configurar Documento</OPCION>
</GRUPO_OPCIONES>
</OPCIONES_MENU>
<PARAMETROS_PAGINACION>
<NUMERO_FILAS_PAGINA>5</NUMERO_FILAS_PAGINA>
<NUMERO_BOTONES_NUMERICOS>3</NUMERO_BOTONES_NUMERICOS>
</PARAMETROS_PAGINACION>
<PARAMETROS_LISTADO>
<NUMERO_FILAS POR_PAGINA>10</NUMERO_FILAS POR_PAGINA>
</PARAMETROS_LISTADO>
</REPOSITORIO>

```

## Instrucciones SQL

Enumerales, a continuación, la relación de instrucciones SQL alojadas en esta aplicación, concretamente en métodos de las clases correspondientes de la capa de datos:

1. Consulta que devuelve un usuario aplicando criterio de búsqueda por clave primaria (id\_usuario):

```
SELECT * FROM usuarios_biblioteca  
WHERE id_usuario = CAST(?) AS CHAR(15)
```

2. Insertar libro:

```
INSERT INTO libros VALUES(?, ?, ?, ?, ?, ?, ?)
```

3. Eliminar libro:

```
DELETE FROM libros WHERE id_libro = CAST(?) AS  
CHAR(5)
```

4. Actualizar columnas. Se trata de una SQL presente en dos métodos diferentes, a invocar en situaciones diferentes. La SQL se compone en tiempo de ejecución, en función de la interacción que se produzca con el usuario previa a lanzar la consulta. Por tanto, solamente reproducimos aquí los String que pueden formar parte de dicha composición. Esta misma circunstancia es aplicable a todas las SQL que aparecerán en lo sucesivo que también se compondrán en tiempo de ejecución:

```
UPDATE libros SET  
titulo = ?  
genero = ?  
fecha_edicion = ?  
numero_paginas = ?  
premiado = ?  
WHERE id_libro = CAST(?) AS CHAR(5))
```

5. Consultar libro aplicando el criterio de búsqueda por clave primaria (id\_libro):

```
SELECT id_libro, titulo, generos.codigo,  
generos.descripcion, fecha_edicion,  
numero_paginas, premiado  
FROM generos INNER JOIN libros  
ON generos.codigo = libros.genero  
WHERE id_libro = CAST(?) AS CHAR(5))
```

6. Consultar una relación de libros susceptible de aplicar diferentes criterios de selección y de ordenación. La SQL se compondrá en tiempo de ejecución:

```
SELECT id_libro, titulo, generos.codigo,
generos.descripcion, fecha_edicion,
numero_paginas, premiado
FROM generos INNER JOIN libros
ON generos.codigo = libros.genero
WHERE generos.codigo = ?
id_libro >= ? AND id_libro <= ?
titulo >= ? AND titulo <= ?
genero >= ? AND genero <= ?
ORDER BY id_libro
ORDER BY titulo
ORDER BY genero, titulo
```

7. La SQL que exponemos a continuación, devolverá las filas correspondientes a la consulta paginada a mostrar en el JTable integrado en la vista VistaPaginadaTabla. La SQL también se compone en tiempo de ejecución. En este caso, nos encontramos en la necesidad de personalizarla adecuándola al Gestor de Base de Datos que haya que lanzarla. No entraremos en detalles al respecto, dado que se extralimita a la intención, propósitos, y extensión de este libro profundizar en temas relacionados con Bases de Datos.

◦ *Oracle:*

```
SELECT id_libro, titulo, generos_codigo,
generos_descripcion, fecha_edicion,
numero_paginas, premiado, num_fila
FROM
(SELECT id_libro, titulo, generos_codigo,
generos_descripcion,
```

```

fecha_edicion, numero_paginas, premiado, rownum AS
num_fila
FROM
(SELECT id_libro, titulo, generos.codigo AS
generos_codigo,
generos.descripcion AS generos_descripcion,
fecha_edicion, numero_paginas, premiado
FROM generos INNER JOIN libros
ON generos.codigo = libros.genero
*** clausulaORDERBY ***
)
)
WHERE num_fila>=((?-1)*?)+1
AND num_fila<=(?*?)

```

○ *MySql:*

```

SELECT id_libro, titulo, generos.codigo,
generos.descripcion, fecha_edicion,
numero_paginas, premiado
FROM generos INNER JOIN libros
ON generos.codigo = libros.genero
*** clausulaORDERBY ***
LIMIT ?,?

```

Posibles criterios de ordenación:

\*\*\* clausulaORDERBY \*\*\* **ORDER BY id\_libro**  
 \*\*\* clausulaORDERBY \*\*\* **ORDER BY titulo**  
 \*\*\* clausulaORDERBY \*\*\* **ORDER BY genero**

8. Consulta que devuelve el número de libros:

```
SELECT COUNT(*) FROM libros
```

9. Consulta que devuelve la relación de géneros:

```
SELECT * FROM generos ORDER BY código
```

10. Consulta que devuelve el siguiente valor de secuencia\_libros convertido a cadena, justificado a la derecha, y llenando el espacio sobrante a la izquierda con '0'. Implementa la obtención del autonumérico a aplicar a la columna id\_libro:

◦ *Oracle:*

```
SELECT
REPLACE(TO_CHAR(secuencia_libros.NEXTVAL,'09999'),
' ')
FROM DUAL
```

◦ *MySQL:*

```
SELECT
LPAD(FORMAT(secuencia_next_valor(\"secuencia_libro
s\"),0),5,'0')
```

Como ya hemos señalado con anterioridad, no pretendemos entrar en conceptos sobre Bases de Datos, pero en relación a esta SQL, sí que conviene matizar que en Oracle podemos implementar la gestión de un autonumérico mediante el objeto Sequence. Dicho objeto no se encuentra disponible en MySQL, Suplimos este inconveniente mediante la implementación en base a procedimientos almacenados que podrá observar el lector en el correspondiente script de recreación de la Base de Datos disponible en el material descargable del libro.

11. Inserta registro de actividad de usuario:

```
INSERT INTO actividad_usuarios VALUES(?, ?, ?, ?)
```

12. Consultar la relación de tipos de actividad de usuario contemplados y su correspondiente codificación. El destino de esta consulta es cargar un HashMap.

```
SELECT descripcion, codigo FROM
```

## **codificacion\_actividades**

13. Consultar la relación de usuarios en los que se haya registrado actividad:

```
SELECT DISTINCT id_usuario FROM actividad_usuarios  
ORDER BY id_usuario
```

14. Consultar la relación de fechas en que se haya registrado actividad. Es otro caso en que nos vemos obligados a personalizar para cada Gestor de Base de Datos:

◦ *Oracle:*

```
SELECT DISTINCT TO_CHAR(fecha_hora, 'YYYY-MM-DD')  
FROM actividad_usuarios  
ORDER BY TO_CHAR(fecha_hora, 'YYYY-MM-DD')
```

◦ *MySQL:*

```
SELECT DISTINCT DATE_FORMAT(fecha_hora, '%Y-%m-%d')  
FROM actividad_usuarios  
ORDER BY DATE_FORMAT(fecha_hora, '%Y-%m-%d')
```

15. Consulta que devuelve el resultado de una estadística de actividad de usuarios solicitada en la vista EstadisticasActividad. Se trata del caso más acentuado que aparece en este libro, en lo que a diversidad se refiere, de instrucción SQL que se conforma en tiempo de ejecución. Es poco habitual el que aparezcan en una aplicación instrucciones SQL a conformar en tiempo de ejecución con tan elevado grado de versatilidad. Sugerimos al lector proceda a un análisis en profundidad de la lógica algorítmica aplicada cuya codificación encontrará distribuida en diversos métodos de la clase ActividadUsuariosDatos, siendo invocada la utilidad desde la capa de negocio:

```
new
ActividadUsuariosDatos().consultarConCriteriosSele
cciónAgrupacion
(connection, parametrosSelecciónAgrupacion,
baseDatos)
```

A efectos didácticos, cuando en tiempo de ejecución de la aplicación se lanza esta SQL, es mostrada la instrucción concreta conformada en cada caso, en la consola de salida. Para mayor claridad para el lector, vamos a mostrar algunas SQL concretas obtenidas en la consola de salida al ejecutarse esta opción conectando con MySQL:

```
SELECT id_usuario,DATE_FORMAT(fecha_hora, '%Y-%m-
%d'),
codificacion_actividades.codigo,descripcion
FROM codificacion_actividades INNER JOIN
actividad_usuarios
ON codificacion_actividades.codigo =
actividad_usuarios.codigo
ORDER BY id_usuario, DATE_FORMAT(fecha_hora, '%Y-
%m-%d'),
codificacion_actividades.codigo
SELECT id_usuario,DATE_FORMAT(fecha_hora, '%Y-%m-
%d'),
codificacion_actividades.codigo,descripcion
FROM codificacion_actividades INNER JOIN
actividad_usuarios
ON codificacion_actividades.codigo =
actividad_usuarios.codigo
WHERE id_usuario = CAST(? AS CHAR(15))
ORDER BY id_usuario,DATE_FORMAT(fecha_hora, '%Y-
%m-%d'),
codificacion_actividades.codigo
```

```
SELECT id_usuario,COUNT(*)
```

```
FROM codificacion_actividades INNER JOIN  
actividad_usuarios  
ON codificacion_actividades.codigo =  
actividad_usuarios.codigo  
WHERE DATE_FORMAT(fecha_hora, '%Y-%m-%d') >= ?  
AND DATE_FORMAT(fecha_hora, '%Y-%m-%d') <= ?  
GROUP BY id_usuario  
ORDER BY id_usuario
```

Son múltiples las combinaciones de cláusulas y de columnas aplicadas a las mismas que se pueden obtener estableciendo diferentes configuraciones de criterios de selección y agrupación para la misma SQL base. Invitamos al lector a que ejecute la aplicación y analice las diversas SQL generadas y que se muestran en consola de salida texto.

Para ejecutar la aplicación son necesarias las siguientes actuaciones previas:

- Establecer en la línea

```
<BASE_DATOS_SELECCIONADA>MySQL</BASE_DATOS_SELECCIONADA>
```

si se pretende que la aplicación conecte con el Gestor de Bases de Datos MySQL, ó

```
<BASE_DATOS_SELECCIONADA>Oracle</BASE_DATOS_SELECCIONADA>
```

para el caso de pretender que la aplicación conecte con el Gestor de Bases de Datos Oracle del fichero repositorio.xml, presente en el directorio xml del proyecto.

- Iniciar los servicios correspondientes al Gestor de Bases de Datos seleccionado.
- Para la primera ejecución de la aplicación deben haberse lanzado las instrucciones SQL de:

- Creación de tablas
- Creación de secuencia o de procedimientos almacenados, según el gestor de que se trate
- Inserción de datos de prueba que aparecen en los ficheros “script de recreación” de la Base de Datos correspondientes al Gestor de Bases de Datos seleccionado. Dichos ficheros se encuentran disponibles en el material descargable del libro.

# 27

## ASPECTOS GLOBALES DE LA APLICACIÓN

La lógica de negocio asociada a esta aplicación está basada en las funcionalidades a implementar asociadas al mantenimiento de una tabla de una Base de Datos. A primera vista, el lector, puede tener la impresión de que las subopciones de la opción **Mantenimiento** del menú son redundantes; pero pensemos que la aplicación no es un fin en sí misma, sino un medio para poder disponer de un escenario en que poder aplicar de forma integral todos los contenidos, mecanismos y estrategias objeto de estudio y tratamiento en este libro.

En el análisis detallado de esta aplicación, abordaremos en primer lugar los aspectos globales de la misma, para después proseguir en un análisis exhaustivo y minucioso de cada una de las opciones que componen el menú. La mayor parte de componentes y mecanismos integrados en esta aplicación ya han sido abordados con anterioridad en los ejercicios que se han venido planteando a lo largo del libro. Este ejercicio pretende ser un escenario de integración de todos ellos, en que se aportan nuevos modelos de organización. Vayámonos adentrándonos progresivamente en todos los detalles a analizar.

La aplicación presenta un menú en que se utilizan las clases y mecanismos que se abordaron en la aplicación

**MenuJMenuBar.** Acorde con ello, cada una de las opciones accesibles por selección del correspondiente *JMenuItem* son implementadas por una clase que “hereda” de **PantallaOpcion**, clase que a su vez “hereda” de *JPanel*.

En la aplicación **MenuJMenuBar** se aplicaba una definición explícita de los *JMenu*:

```
JMenu jMenuOpciones = new JMenu("Opciones con  
funcionalidad");
```

y de los *JMenuItem*:

```
JMenuItem jMenuItemOPcion1 = new JMenuItem("Opcion  
1");
```

En la aplicación **GestiónLibros** definimos las opciones de menú de ambos niveles en el fichero repositorio.xml, presente en el directorio xml del proyecto, concretamente en el subárbol:

```
<OPCIONES_MENU>  
<GRUPO_OPCIONES nombre_grupo="Conexión-Desconexión">  
<OPCION>Conexion</OPCION>  
<OPCION>Desconexion</OPCION>  
</GRUPO_OPCIONES>  
<GRUPO_OPCIONES nombre_grupo="Mantenimiento">  
<OPCION>Vista Formulario</OPCION>  
<OPCION>Vista Unica Tabla</OPCION>  
<OPCION>Vista Paginada Tabla</OPCION>  
<OPCION>Vista Arbol</OPCION>  
</GRUPO_OPCIONES>  
<GRUPO_OPCIONES nombre_grupo="Actividad">  
<OPCION>Estadisticas Actividad</OPCION>  
<OPCION>Volcado a BD</OPCION>  
</GRUPO_OPCIONES>  
<GRUPO_OPCIONES nombre_grupo="Generar documentos">  
<OPCION>Configurar Documento</OPCION>  
</GRUPO_OPCIONES>  
</OPCIONES_MENU>
```

De tal modo que la configuración que se establezca en este fichero que actúa como repositorio centralizado, determinará las opciones de menú de ambos niveles aparecerán en la ventana, sin más condición que cada una de las opciones de segundo nivel se encuentren “respaldadas” por la existencia de una clase con el mismo nombre, pero al que se le han eliminado los posibles espacios en blanco, por ejemplo:

```
<OPCION>Vista Paginada Tabla</OPCION>
```

Debe estar respaldada por la existencia en el *package* presentación de la clase:

```
public class VistaPaginadaTabla extends PantallaOpcion
```

La carga en arrays de la lectura del fichero repositorio.xml, se produce en el método cargar() de la clase **RespositorioXML**, invocado desde el constructor de la clase **Controller**, de la que hablaremos más adelante en profundidad. Como ya hemos mencionado, el fichero repositorio.xml actúa como repositorio centralizado de parámetros de naturaleza muy diversa que utiliza la aplicación, circunstancia que podrá deducir el lector de la observación del contenido de dicho fichero, anteriormente expuesto. Con posterioridad a la carga en arrays de la información que consta en el “repositorio”, ya podemos en el constructor de la clase **Menu** asociar los componentes de dicho array que proceda a:

```
private String[] opcionesMenu;
opcionesMenu = (String [])
(controller.getRepositorio())[2];
private String[][] opciones;
opciones = ((String [][][])
(controller.getRepositorio())[3]);
```

el tamaño y contenido de estos arrays permitirá la instanciación y generación de los correspondientes componentes que quedarán aglutinados en los arrays

```
private JMenu[] jMenus;
private JMenuItem[][] jMenuItems;
```

Todo ello tendrá lugar en el método **ubicarComponentes()** de la clase Menú, así como asociar cada *JMenuItem* al *JMenu* que le corresponde por grupo. Sugerimos al lector que se detenga en la observación del código implementado en el citado método.

Nos falta todavía la segunda parte de la configuración del menú, la que ataña a la instanciación de los *JPanel* que corresponden a cada una de las vistas opciones del menú, y a la invocación de sus métodos. Pero antes de proseguir el autor cree oportuno hacer “un alto en el camino”, dado de que es consciente que el lector puede tener la sensación de que toda esta implementación es retorcida, rebuscada, tal vez innecesaria, pero sobre todo compleja. A efectos de disuadir al lector de que abandone y motivarle para que prosiga adentrándose en los mecanismos presentes en esta aplicación, hay que señalar dos cuestiones:

- El coste que le pueda suponer al lector entender los mecanismos inherentes a la parametrización subyacente en un fichero externo de configuración (en este caso, repositorio.xml), lo rentabilizará con creces posteriormente, pues le proporcionará una mayor versatilidad y automatización en el desarrollo de aplicaciones, al menos en lo que a la configuración del menú se refiere.
- La parametrización subyacente en ficheros de configuración externos, que de otra manera debe definir explícitamente el programador en su código, suponen el punto de partida del

desarrollo de aplicaciones parametrizables por el programador externamente al código, y del principio del desarrollo de “frameworks”. Invitamos al lector a que edite y configure el citado fichero repositorio.xml mediante cualquier editor de texto, definiendo nuevas opciones, y definiendo, paralelamente, los *JPanel* que actúan como vistas de dichas opciones. Obviamente el lector podrá afrontar este reto cuando ya esté familiarizado con los mecanismos que todo ello comporta. El siguiente nivel consistiría en desarrollar interfaces gráficas cuya única finalidad sea proporcionar un edición mucho más amigable, sugerente e intuitiva de estos ficheros de configuración externos, materializado en nuestro caso por repositorio.xml, circunstancia que se extralimita a los propósitos de este libro. Tampoco debe descartar el lector la posibilidad de desarrollar las susodichas interfaces gráficas que permiten la edición automática de ficheros de configuración. En muchas ocasiones, los seres humanos tendemos a crear mitos asociados a proyectos que pueden estar perfectamente a nuestro alcance y ser asumidos por nosotros.

Confiando el autor en que, con estas reflexiones, el lector se encuentre en mejores condiciones anímicas para proseguir, volvamos al “terreno” del código. Así como en el ejemplo **MenuJMenuBar** todos los *JPanel* “vistas” habían de ser instanciados en la carga del menú, y los correspondientes objetos presentes en memoria en todo momento, en la nueva organización planteada en **GestionLibros**, cada *JPanel* solamente es instanciado cuando va a ser presentado como “vista”, siendo eliminado posteriormente por el Garbage collector cuando sea

reemplazado por otro. En conclusión, solamente tenemos cargada en memoria en cada momento, la instancia correspondiente al JPanel presentado como “vista”. Instanciamos el JPanel a visualizar en cada momento:

```
PantallaOpcion pantallaOpcion =  
(PantallaOpcion)Class.forName("presentacion."+  
eliminarEspaciosEnBlanco(actionCommand)).newInstance()  
;
```

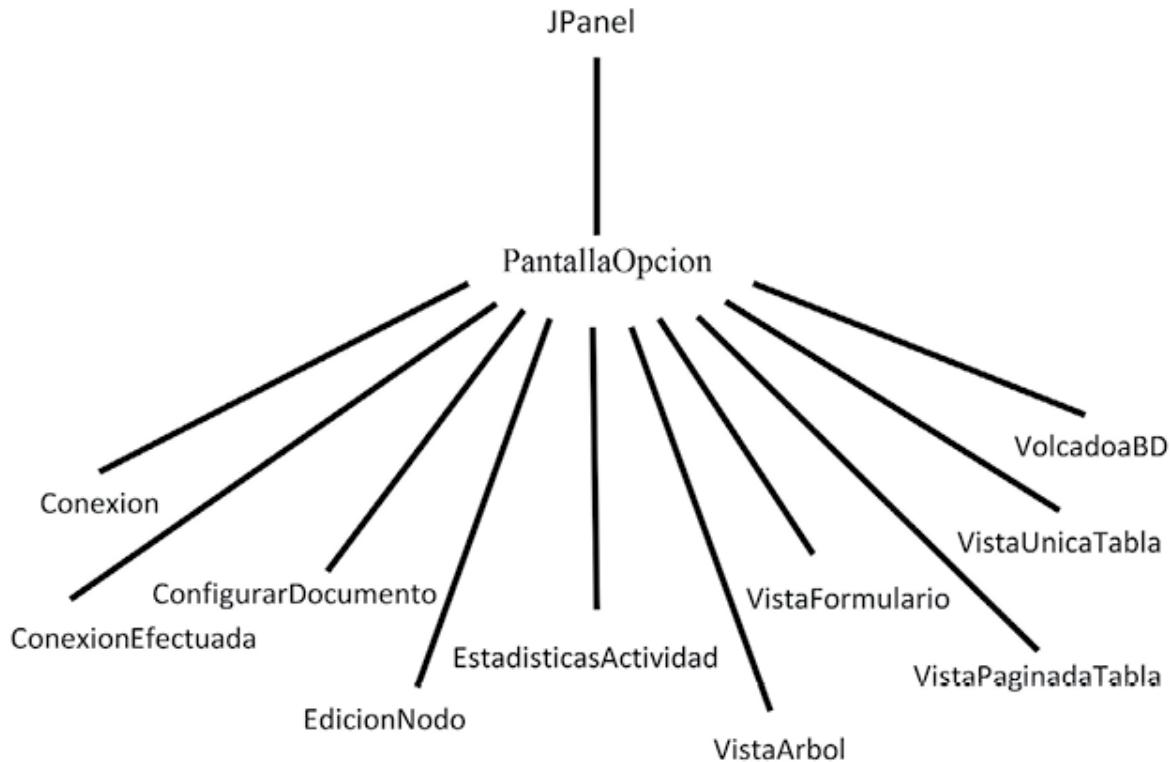
donde actionCommand es definido en:

```
((JMenuItem) e.getSource()).getActionCommand()
```

en el método *actionPerformed()* de **Controller** (clase centralizadora de todas las escuchas), al ser seleccionado el correspondiente JMenuItem y dispararse el evento asociado. Y como ya hemos comentado con anterioridad debe existir una asociación entre las opciones de menú materializadas en los JMenuItem, y el nombre de las clases (eliminando los espacios en blanco) de los JPanel que se corresponden con las vistas-opciones del menú.

Si el lector no está familiarizado con la clase *Class* y sus métodos *forName()* y *newInstance()*, le remitimos al libro “Programación Orientada a Objetos en Java”, también del mismo autor, todo ello tratado con amplitud en el capítulo correspondiente a **Polimorfismo.Reflection.Genéricos**.

Con los pocos indicios que hemos proporcionado hasta ahora, la sagacidad del lector le habrá llevado a concluir que el conjunto de la clase **PantallaOpcion**, y todas las vistas que conforman el menú, “herederas” de la primera, van a ser el escenario de una implementación polimórfica:



Dicha implementación polimórfica nos va a permitir que en todo momento la referencia:

#### **PantallaOpcion pantallaOpcion**

“apunte” al *JPanel* correspondiente a la vista concreta que en cada momento va a ser visualizada, y cuyos métodos vamos a invocar, previamente a la visualización, y cuando ya aparezca como visualizada, tanto desde el **Menú**:

```
pantallaOpcion.inicializarPostInstanciar(controller);
pantallaOpcion.inicializarPantalla();
```

como desde el método **centralizar()** de la clase **Controller**:

```
pantallaOpcion.responderAController(actionCommand)
```

Los tres métodos citados existen en todos los *JPanel* vistas-opciones del menú, bien porque son heredados de **PantallaOpcion**, ofreciendo una implementación vacía:

```
public void inicializarPantalla() throws Exception { }
```

O porque son de obligatoria implementación en las clases “herederas” dado el modificador *abstract*:

```
public abstract void  
inicializarPostInstanciar(Controller controller)  
throws Exception;  
public abstract void responderAController(String  
actionCommand) throws Exception;
```

La dinámica del flujo de invocaciones a métodos provocadas por el lanzamiento de cualquier evento, se acoge totalmente al ya utilizado en la aplicación **PrediccionMeteorologicaJTree**. De hecho, se aplicó en su momento en el citado ejercicio, única y exclusivamente, con carácter preparatorio de poder entender las estrategias y mecanismos utilizados en la aplicación final **GestionLibros** que estamos tratando en estos momentos. Para mejor comprensión del lector, le remitimos al esquema de clases-métodos mostrado en el análisis de la aplicación **PrediccionMeteorologicaJTree**. Como ya se comentó cuando se estudió esta aplicación, dichos mecanismos y estrategias eran excesivos para las necesidades organizativas que allí se planteaban, en términos coloquiales, era como “matar moscas a cañonazos”. Pero insistimos, supusieron allí un preliminar para su implementación en **GestionLibros**, escenario en que sí que rentabilizamos la organización aportada. Como ya se comentó en su momento, el autor pretende con todo ello aplicar criterios organizativos basados en el patrón de diseño Front Controller, cuyo principal beneficio en esta aplicación queda plasmado, entre otras circunstancias en la posibilidad de disponer de “un punto de paso obligatorio” centralizado de cualquier flujo de ejecución en el tratamiento de eventos que se produzca en la aplicación. Una

clara manifestación de lo que estamos comentando es que en la ejecución de la aplicación se produce:

- Registro de la actividad del usuario:

```
new  
GestorIncidentes().gestionarActividad(usuarioAutenticado,  
actividadesCodificadas.get(actionCommand).intValue()  
(), actionCommand);
```

que permitirá una posterior aplicación de estadísticas, circunstancia que trataremos con detalle al estudiar en profundidad las opciones de menú **Volvado a BD** y **Estadísticas Actividad**.

- Registro y tratamiento de las excepciones lanzadas desde cualquier parte de la aplicación:

```
new  
GestorIncidentes().gestionarExcepcion(exception,  
usuarioAutenticado);
```

Ambos actuaciones tienen lugar en el método **centralizar()**, que representa el punto de paso obligatorio y centralizado a que hemos hecho alusión. Lo que acabamos de comentar no es más que una muestra de los beneficios de centralizar flujos de ejecución, uno de los beneficios obtenidos consecuencia de la aplicación del Front Controller. El estudio de este patrón de diseño queda fuera de los contenidos y objetivos planteados en este libro, es más, siendo rigurosos en el seguimiento de las directrices marcadas por el Front Controller, también deberíamos implementado en el método **centralizar()**, la instanciación y visualización del *JPanel* vista-opción seleccionado por el usuario, y el control de autenticación asociado a sesión de

usuario, del que hablaremos más adelante. Decimos “deberíamos” y no “hemos hecho” porque estas actuaciones citadas se han desviado, la instanciación y visualización del *JPanel* vista-opción a la clase **Menu**, y la autenticación asociado a sesión de usuario a la clase *JPanel Conexión*. Nos hemos “apartado” del rigor establecido por el Front Controller en pro de disminuir la complejidad de la lógica-algorítmica en aras de facilitar la comprensión inicial. No obstante, invitamos al lector, a que, una vez totalmente asumidas la lógica asociada a los mecanismos y estrategias aplicados en este escenario organizativo, derive al método **centralizar()** las actuaciones citadas, que es donde deberían ser ubicas, hablando en términos de rigor y purismo.

Otro aspecto del esquema organizativo que estamos presentando en que nos hemos apartado del rigor exigido por el Front Controller, siempre en pro de facilitar la comprensión por parte del lector, estriba en la existencia del método **responderAController()**, presente obligatoriamente en todas las vistas. Utilizamos este método para implementar las actuaciones concretas a realizar asociadas al tratamiento de un evento. Supone un “retorno a la vista” desde el enclave centralizador concretado en el método **centralizar()** de la clase **Controller**. La alternativa más purista a ello consistiría en que el método **responderAController()**, o su equivalente, estuviese presente en otra clase específica y únicamente destinada a implementar las actuaciones de respuesta al evento. Deberíamos disponer de una clase específica para cada una de las opciones. El equivalente al método **responderAController()** en la nueva clase que estamos apuntando, devolvería un *String* con el nombre de la nueva vista a mostrar en los casos en que ello fuera procedente. Con el *String* “retornado”, se tomarían las medidas oportunas para

visualizar la nueva vista. Animamos al lector para que acometa estos cambios en su momento.

En lo que a aspectos globales se refiere, nos falta tratar los mecanismos utilizados para permitir un acceso centralizado a determinados componentes y datos cuando se es requerido, aspecto novedoso en esta aplicación, y que representa una variante en relación a estrategias al respecto previamente utilizadas. Para ello, centramos nuestra atención en el atributo de la superclase **PantallaOpcion**:

```
protected Object[] componentesJPanel = new Object[20];
```

que dispone de su “setter” y “getter”:

```
public Object getComponenteJPanel(int componente) {  
    return componentesJPanel[componente];  
}  
public void setComponenteJPanel(Object object, int  
    componente) {  
    componentesJPanel[componente] = object;  
}
```

Obsérvese que dichos métodos no acceden a la estructura de datos en su conjunto, sino que permiten el acceso individualizado a un componente en concreto, determinado por uno de los parámetros transferidos al método:

```
int componente
```

Apelamos nuevamente a la sagacidad del lector, para que previamente a la explicación detallada, empiece a intuir, que definir como *Object* los componentes del array, responde a una intención de tratamiento genérico. Entremos directamente a la cuestión. En las aplicaciones tratadas hasta ahora, la necesidad de poder acceder desde diferentes lugares (obviamente, clases) de la aplicación a un componente o dato, se ha conseguido

estableciendo una clase centralizadora de referencias a instancias, dicho papel ha sido ejercido por la clase Componentes. La solución empleada hasta ahora ha sido viable dado que el número de componentes a acceder de forma centralizada era reducido y todas las instancias de todas las clases intervenientes estaban permanentemente presentes en memoria. Pero en la aplicación **GestionLibros**, esta solución no es factible. La principal razón de la inviabilidad de la solución aplicada hasta ahora es que la instancia de los *JPanel* vistas-opción de menú solamente existe mientras se encuentra en estado de visualización. Al ser reemplazada por otra, deja de ser utilizada, y es eliminada por el Garbage collector, y en consecuencia también son eliminados todos sus componentes y datos asociados. La solución organizativa que estamos comentando del acceso “distribuido” a componentes y datos asociados, utilizada novedosamente en esta aplicación, consiste en que cada *JPanel*, cuando es instanciado, registre en los componentes del array

#### **Object[] componentesJPanel**

la referencia de los componentes y datos que vayan a requerir el acceso desde otra clase diferente al *JPanel* “al que pertenecen”. Dicho registro no se aplicará a todos los componentes, sino solamente a aquellos en que exista la necesidad de un acceso “distribuido”.

El hecho de encontrarnos en un escenario polimórfico, nos fuerza a proporcionar un carácter genérico a todas las actuaciones y elementos intervenientes, que conseguimos aplicando las siguientes medidas:

1. Definiendo como *Object* los componentes del array **componentesJPanel**.

2. Declarando dicho array en la superclase, estando, en consecuencia, disponible por herencia en todos los *JPanel* vista-opción.
3. Aplicando la medida anterior a los métodos “setter” y “getter” de acceso a los componentes del array anteriormente mencionados.

Mostramos a continuación una concreción de aplicación de los mecanismos que estamos describiendo. Hemos escogido para ello la clase **VistaUnicaTabla**, por ser el caso donde todo lo que estamos comentando se manifiesta, tal vez, de forma más acentuada. Detallamos el tratamiento que aplicamos a los componentes a almacenar en el array **componentesJPanel** de este *JPanel* en cada una de las clases:

### **En el propio JPanel VistaUnicaTabla**

Registraremos los componentes:

```
componentesJPanel[0] = jTable;
componentesJPanel[3] = jButtonInsertarFila;
componentesJPanel[4] = jButtonCancelarInsercionFila;
componentesJPanel[5] = jButtonGuardarFilaInsertada;
componentesJPanel[6] =
jButtonEliminarFilaSeleccionada;
```

los Model:

```
componentesJPanel[1] = listSelectionModel;
componentesJPanel[15] = modeloDatos;
```

y los datos:

```
componentesJPanel[2] = new Integer(-1); // Fila
seleccionada en JTable
componentesJPanel[10] = new Integer(1); // El valor
registrado determina el criterio
```

```
de ordenación de las filas del JTable.  
componentesJPanel[13] = new Integer(1); // Fila de  
celda actualizada en JTable.  
componentesJPanel[14] = new Integer(1); // Columna de  
celda actualizada en JTable.
```

Tratándose estos últimos de datos enteros, que debemos encapsular en su correspondiente wrapper para poder ser registrados en el array de *Object*. Inicialmente, el dato encapsulado es inicializado, pero es susceptible de variar como consecuencia de la interacción con el usuario.

Detallamos a continuación las otras clases y métodos de las mismas desde las que es necesario que se tenga acceso a los componentes del array que estamos tratando. Obsérvese que el acceso se realiza mediante el “setter” y “getter” implementados a tal efecto, y que el uso del “getter” comporta siempre aplicar *cast* dada la naturaleza genérica *Object* con que se ha declarado el array.

## Desde ModeloDatos

- En varios métodos:

```
((JTable)controller.getPantallaOpcion().getCompon  
enteJPanel(0)).repaint();
```

## Desde Controller

- En **actionPerformed()**, asociado a los *actionCommand* de los *JRadioButton* que provocan reordenación por otro criterio:

```
pantallaOpcion.setComponenteJPanel(new Integer(1),  
10);
```

En que el valor encapsulado determina el criterio de ordenación a aplicar en la consulta a la Base de Datos. En el *JPanel* correspondiente a **VistaPaginadaTabla** también se han integrado los mismos *JRadioButton* cuya selección provoca reordenación de las filas de la tabla. Ello permite “reutilizar” los mismos *actionCommand* para ambas vistas, lo cual determina que el componente del array **componentesJPanel** que registra el valor asociado al criterio de ordenación a aplicar, debe ocupar la misma posición en dicha estructura de ambos *JPanel*.

- En **tablaChanged()**:

```
pantallaOpcion.setComponenteJPanel(new
Integer(e.getFirstRow()), 13);
pantallaOpcion.setComponenteJPanel(new
Integer(e.getColumn()), 14);
```

- En **valueChanged()**:

```
pantallaOpcion.setComponenteJPanel(
new
Integer(listSelectionModel.getMinSelectionIndex())
, 2);
```

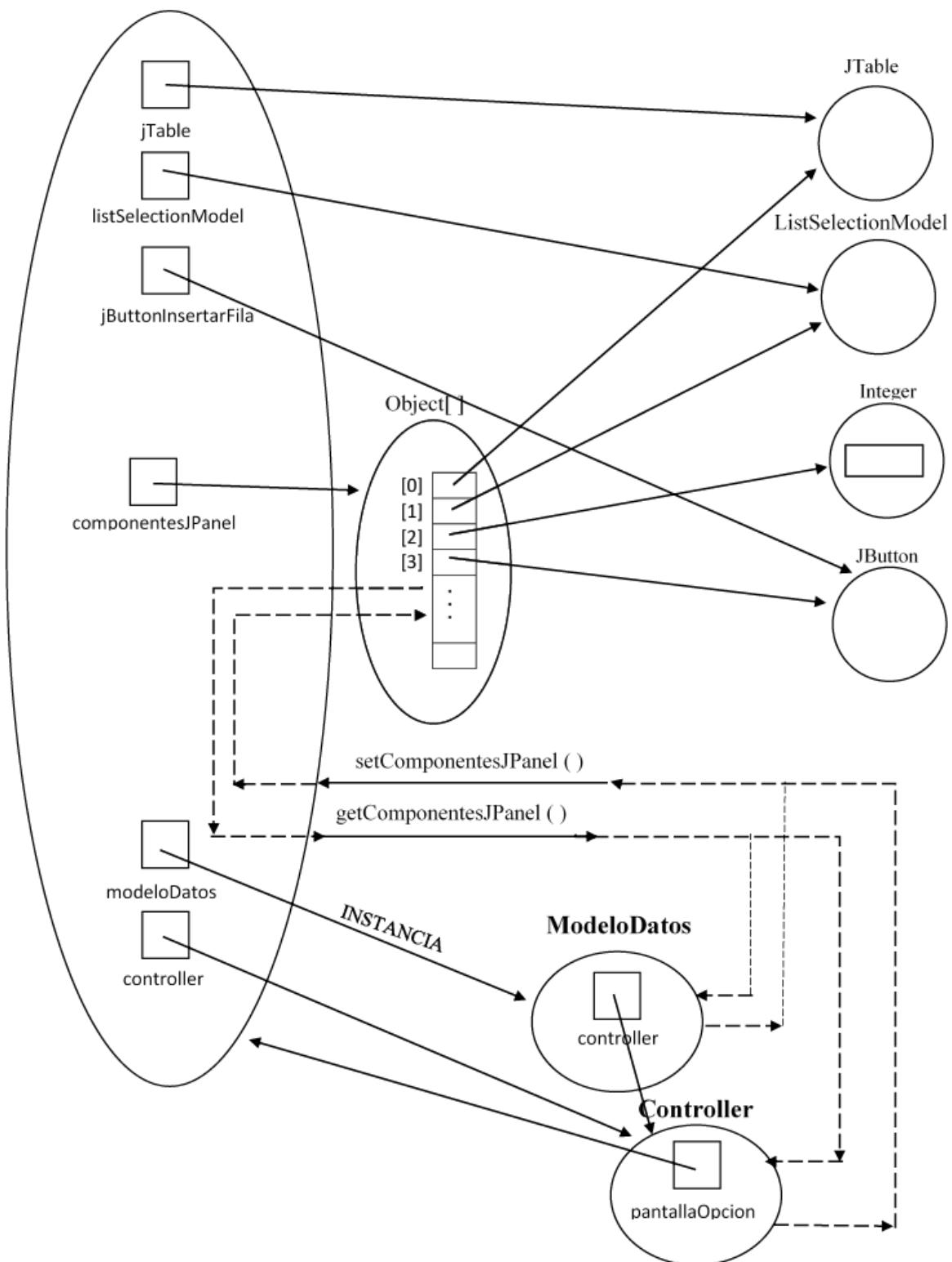
- En **centralizar()**:

```
((ModeloDatos)pantallaOpcion.getComponenteJPanel(1
5)).
getDatos()
[((Integer)pantallaOpcion.getComponenteJPanel(13))
.
intValue()]
[((Integer)pantallaOpcion.getComponenteJPanel(14))
.
intValue()] = ((ModeloDatos)pantallaOpcion.
getComponenteJPanel(15)).getCopiaReservaDato();
((JTable)pantallaOpcion.getComponenteJPanel(0)).re
```

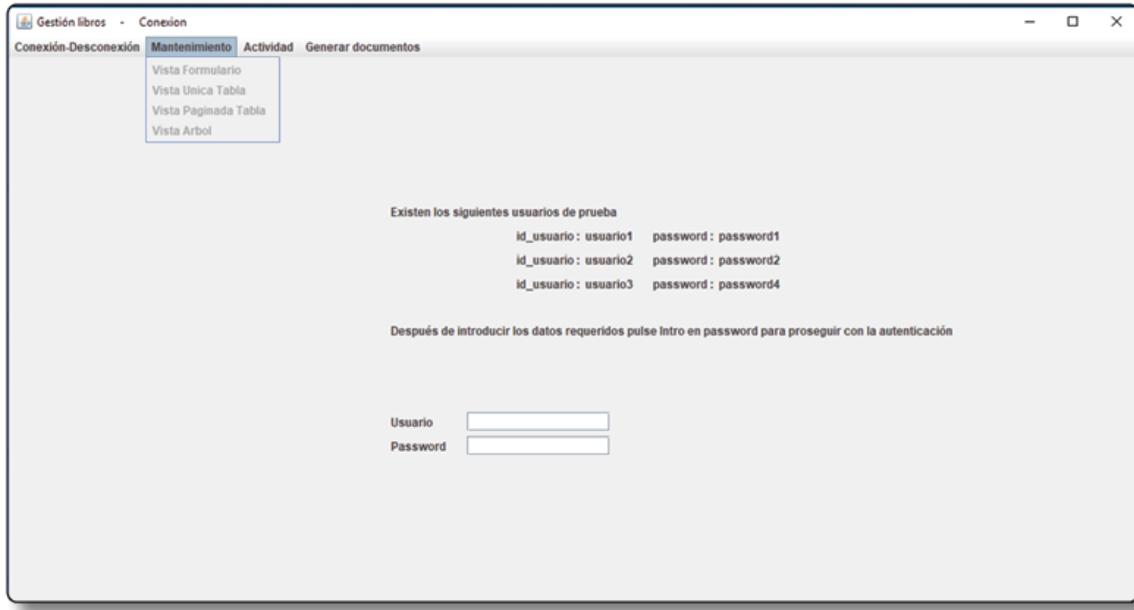
```
paint();
```

Mostramos a continuación un esquema con que pretendemos representar los mecanismos descritos aplicados a la clase **VistaUnicaTabla**. Los mecanismos descritos, así como el siguiente esquema, lo aplicaremos por extensión al resto de *JPanel* vista-opción del menú.

### VistaUnicaTabla



La ejecución de la aplicación visualiza la siguiente ventana:



Apreciamos que en la parte superior tenemos el menú, y que cada opción tiene su correspondiente submenú. El *JPanel* que aparece visualizado por defecto es el correspondiente a la clase Conexión, que procederemos a estudiar con detalle en primer lugar. Deducimos inmediatamente que este primer *JPanel* es el que permite la conexión con sesión de usuario. Observando el submenú desplegado, comprobamos que se encuentran desactivadas todas las opciones de dicho submenú, cuya selección daría paso a la visualización del *JPanel* correspondiente en cada caso. La circunstancia de estar todas desactivadas es extensiva al resto de opciones de submenú. Todo ello consecuencia de no estar conectado a la aplicación con sesión de usuario.

## OPCIÓN: CONEXION

La vista correspondiente a este *JPanel* ha sido ya mostrada con anterioridad. Todas las opciones de menú disponen de su *JPanel* correspondiente, de tal modo que la selección del *JMenuItem*

asociado lanza un evento, cuyo tratamiento provoca la visualización de la vista asociada, a excepción de los *JPanel Conexion* y **ConexionEfectuada**. La visualización de ambos *JPanel* es provocada un lanzamiento explícito de evento. Este mecanismo ya ha sido utilizado con anterioridad en la aplicación **PrediccionMeteorologicaJTree**, y tratado en otros dos ejercicios dedicados en exclusiva. Para la implementación de este mecanismo, se han dispuesto las clases **InvocacionAutomaticaMenu** y **EventoOpcionMenu**. En esta última clase se define el evento a lanzar. Este “hereda” de *ActionEvent*, con la intención de poder homogeneizar su tratamiento con todo el resto de eventos de la misma clase lanzados por componentes de la aplicación como consecuencia de la interacción con el usuario. Los aspectos relacionados con la homogeneización del evento de lanzamiento explícito con el resto, ya se ha comentado en anteriores ocasiones. La implementación del lanzamiento explícito del evento que desencadena la visualización del *JPanel Conexion* :

```
InvocacionAutomaticaMenu invocacionAutomaticaMenu =  
new InvocacionAutomaticaMenu();  
invocacionAutomaticaMenu.addEventoOpcionMenuListener(c  
ontroller);  
invocacionAutomaticaMenu.fireEventoOpcionMenu("Conexio  
n");
```

la aplicamos en:

- El constructor de la clase **Menu**.
- En el método **responderAController()** del *JPanel Desconexion*, como tratamiento al evento lanzado al pulsar el *JButton* “desconectar”.

Los componentes de interacción con el usuario son:

- *JTextField* **jTextFieldUsuario**, que nos permite introducir usuario.
- *JPasswordField* **jPasswordFieldPassword**, para la introducción del password, que como en cualquier otro componente análogo, el usuario percibe que está introduciendo caracteres, pero sin quedar evidencia de cuáles son exactamente.

Una pantalla de estas características está presente en todas las aplicaciones en que se requiera una autenticación para obtener una conexión con sesión de usuario, pero habitualmente, se ofrece un botón para, una vez introducidos usuario y password, dar opción a aceptar y proceder al proceso de autenticación. En este ejemplo hemos prescindido de dicho botón, y se procede a dar paso a la autenticación pulsando Intro en el *JPasswordField* después de haberlo cumplimentado. Para ello, se ha recurrido a la posibilidad que ofrecen estos componentes de ser registrados con la escucha *ActionListener*:

```
jPasswordFieldPassword.addActionListener(controller);
```

y se le ha asignado un *ActionCommand* con que detectar dicho lanzamiento de evento:

```
jPasswordFieldPassword.setActionCommand("autenticacion");
```

La utilización de esta forma tan insólita de salir de una pantalla de autenticación, responde a la intención de poder disponer en algún ejercicio del libro de un escenario que ofrezca la posibilidad de registrar este tipo de componentes con una escucha *ActionListener*.

Finalmente, si el proceso de autenticación ha sido exitoso, se procede crear la sesión de usuario, materializada en la existencia de una instancia de la clase Contexto, que encapsula la información inherente a la sesión de usuario. Para proceder al cambio de *JPanel* visualizado, recurrimos a otro lanzamiento explícito de evento, cuyo tratamiento permitirá la visualización del *JPanel ConexionEfectuada*, representando esta actuación la otra excepción a que hemos hecho alusión. Queda implementado en lanzamiento explícito del evento en el método **responderAController()** del *JPanel Conexion*:

```
InvocacionAutomaticaMenu invocacionAutomaticaMenu =  
new InvocacionAutomaticaMenu();  
invocacionAutomaticaMenu.addEventoOpcionMenuListener(c  
ontroller);  
invocacionAutomaticaMenu.fireEventoOpcionMenu("Conexio  
nEfectuada");
```

Mostramos a continuación la pantalla que ofrece la aplicación al lograr una conexión de usuario exitosa:

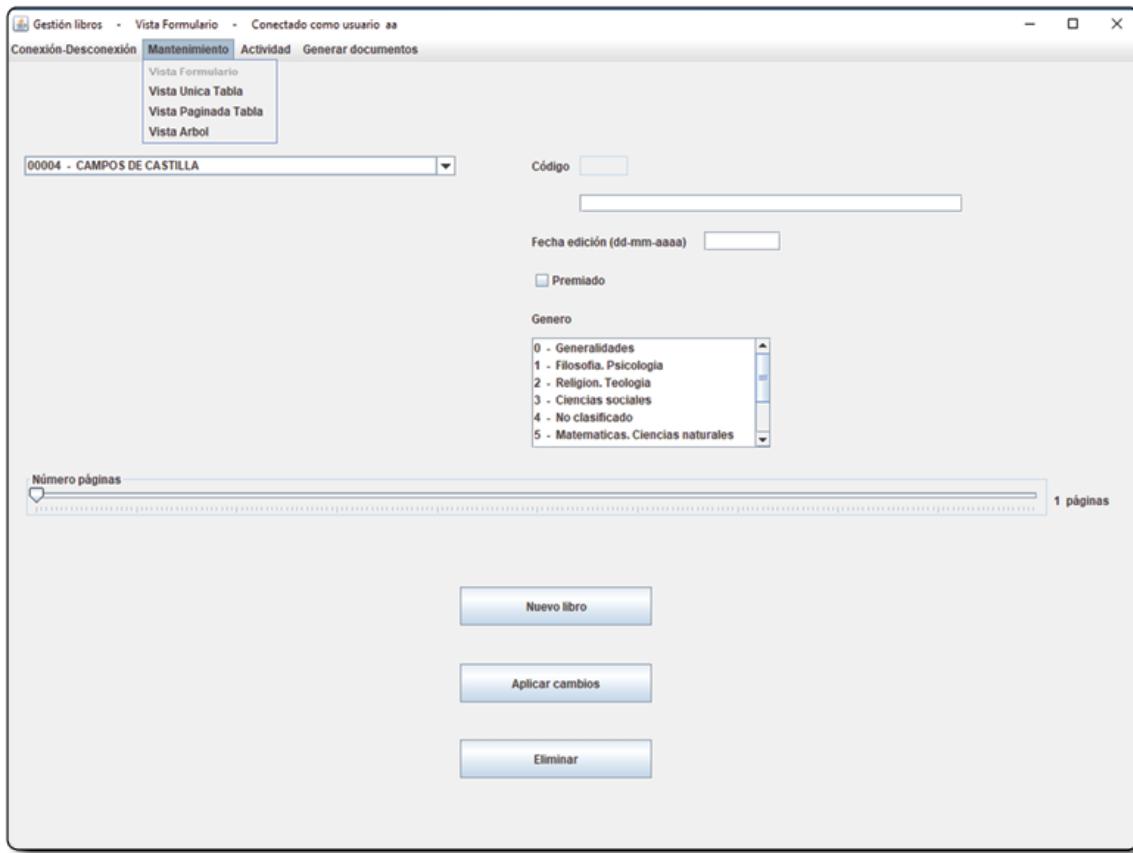


Podemos observar que, ya se ha creado una sesión de usuario. En consecuencia, al desplegar la opción de menú, las subopciones que integra están activadas y en condiciones de ser seleccionadas. Hemos optado por utilizar el *JPanel ConexionEfectuada* como pantalla “de bienvenida” a la sesión de usuario. Perfectamente podríamos haber dispuesto a tal efecto cualquier otro *JPanel* que permita operatividad.

Al proceso antagónico, la desconexión, se accede por selección de dicha subopción en el grupo de opciones Conexión-Desconexión del menú. Las actuaciones derivadas de la desconexión son establecer a *null* los atributos **identificadorUsuario** y **password** de la instancia de la clase Contexto, referenciada desde **usuarioAutenticado**; y el lanzamiento explícito del evento cuyo tratamiento provocará la visualización del *JPanel Conexion*.

## OPCIÓN: VISTA FORMULARIO

La selección de esta opción visualiza el *JPanel* asociado, que presenta la siguiente apariencia:

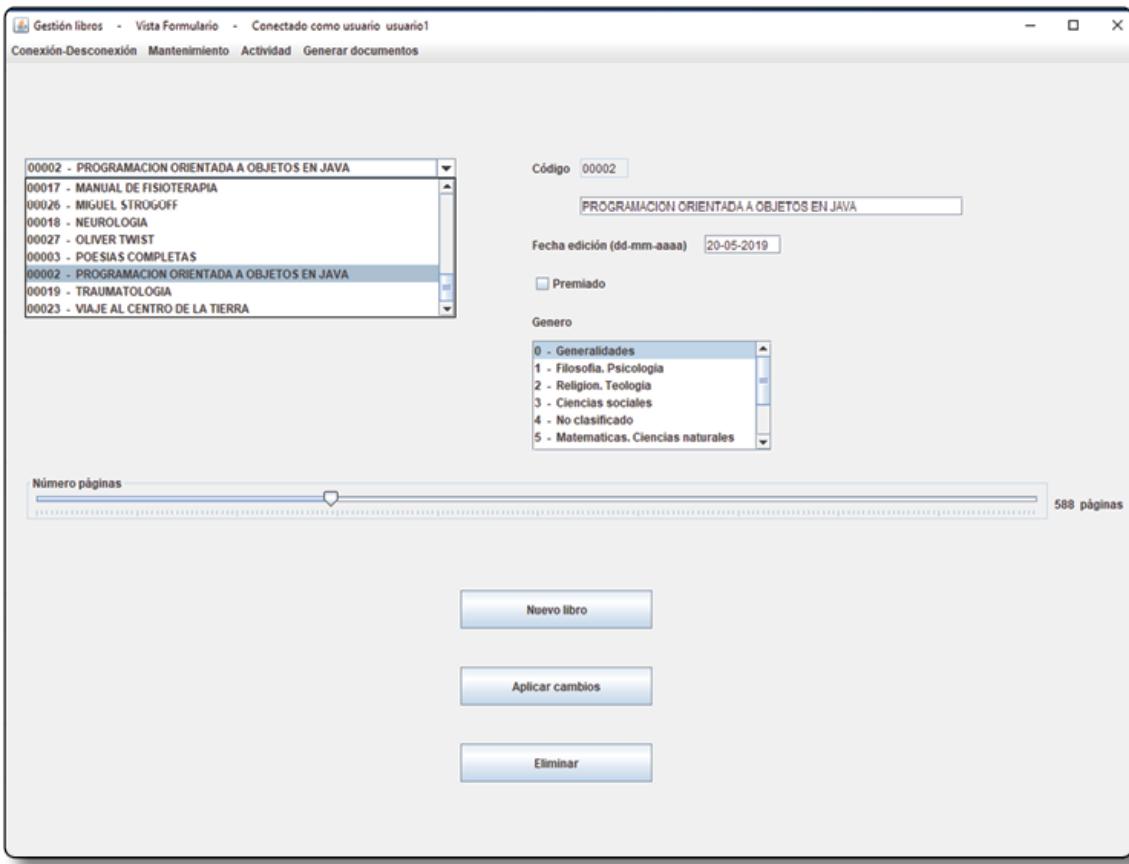


Antes de entrar en cuestiones específicas de este *JPanel*, queremos fijar la atención del lector en que en la pantalla anterior aparece desplegado el grupo de opciones donde está integrada la subopción asociada al *JPanel* visualizado. Se observará que aparece desactivada dicha subopción, pero activadas todo el resto, en condiciones de ser seleccionadas y proceder a un cambio de *JPanel* visualizado.

Mediante la inclusión de este *JPanel* en la aplicación, pretendemos recopilar la utilización de la mayoría de los componentes de interfaz gráfica de usuario que se han venido presentando y tratando a lo largo de los ejercicios del libro. Responde al formulario clásico de edición de datos de cualquier aplicación, aglutinando todas las funciones de un CRUD:

- **NUEVO REGISTRO:** El usuario procede a cumplimentar los valores de todos los datos contemplados por el formulario. Caso de no encontrarse previamente con todos los componentes inicializados, puede pulsar el *JButton Nuevo Libro* para dejar el formulario en estas condiciones. Una vez cumplimentados todos los datos, queda registrada el alta en Base de Datos al pulsar el *JButton Aplicar cambios*.
- **CONSULTAR REGISTRO:** La lectura de un registro existente en la Base de Datos se consigue mediante la selección de un ítem del *JComboBox*, que conduce a la visualización de los datos del registro en los componentes de visualización respectivos.
- **ACTUALIZAR REGISTRO:** Visualizado un registro, podemos proceder a la modificación de los datos que se crean oportunos, registrándose dichos cambios en la Base de Datos al pulsar el *JButton Aplicar cambios*.
- **ELIMINAR REGISTRO:** Pulsar el *JButton Eliminar* estando visualizado un registro, comporta su eliminación de la Base de Datos.

Mostramos nuevamente la vista correspondiente a la opción que estamos analizando, pero estando desplegado el *JComboBox*, y visualizado el registro seleccionado, que da opción a actualizarlo o a eliminarlo, en su caso, siguiendo el procedimiento descrito:



Obsérvese que el *JTextField* asociado al dato Código está establecido como no editable, dado que el dato asociado se genera automáticamente mediante un autonumérico.

El *JList* integrado en esta vista es gestionado de forma diferentes a como se hizo con el que aparece en la aplicación **InterfazGrafica9**. En aquella aplicación para la gestión de los ítems se utiliza un *DefaultListModel*, y en ésta se utiliza un *ListSelectionModel*. Nos encontramos ante dos dinámicas diferentes de gestión de un mismo componente. En la vista que estamos tratando, el componente presenta un contenido estático, que se le transfiere en el constructor:

```
new JList( String[] )
```

Una vez instanciado el componente, y cargado con el contenido, la gestión de la aplicación se limita a leer el ítem seleccionado mediante el *ListSelectionModel*:

```
listSelectionModel.getMinSelectionIndex()
```

La gestión planteada en InterfazGrafica9 supone una dinámica cambiante del contenido, circunstancia que requiere la utilización del *DefaultListModel*. Esta circunstancia nos ha permitido exponer al lector dos formas diferentes de gestionar un mismo componente.

Entrando “en el terreno” de los Listeners, el presente *JPanel*, aporta dos nuevas escuchas no tratadas hasta ahora:

- **CaretListener.** Con esta escucha registramos los *JTextField* que intervienen en la vista. En consecuencia el método *caretUpdate(CaretEvent e)* asociado a esta escucha, se ejecutará cuando se registre cualquier actividad en los mencionados *JTextField*.
- **ItemListener.** El componente registrado con esta escucha ha sido el *JCheckBox jCheckBoxPremiado*, de tal modo que cuando cambie el valor del componente se ejecutará el método *itemStateChanged(ItemEvent e)* asociado a esta escucha. Recordemos que en la aplicación **VotacionPropuesta** también aparecía este tipo de componente, pero los cambios de la selección del mismo eran detectados por la escucha *TableModelListener*, la cual detectaba las actualizaciones de cualquier columna del *JTable*.

Hemos implementado, novedosamente, un mecanismo que permite, ante la actualización de un registro, la detección en la

capa de datos de las columnas que han sido objeto de modificación, y en consecuencia de “engrosar” las columnas a aplicar a la cláusula SET de la instrucción SQL UPDATE. Instrumentalizamos el atributo

```
int actualizaciones
```

a tal efecto. Procedamos a analizar dicho mecanismo. Este atributo es inicializado en el método **inicializarPantalla()**, y al visualizar registro. Es sometido a una actualización a nivel de bit cuando se detecta actualización en algún componente de la vista, de tal modo que a cada uno de dichos componentes se le asocia un bit del dato. Conseguimos con todo ello, codificar en una sola variable la coyuntura asociada al estado de actualización de todas las columnas.

Al ser inicializado el atributo, todos los bits se encuentran a “0”, y cuando se detecta actualización en algún componente, pasa a valor “1” el bit asociado al mismo. Esta actuación es implementada en el método **responderAController()** de la vista:

```
switch(actionCommand)
{
    . . .
    case "actualizadoTitulo" :
        actualizaciones |=1;
        break;
    case "actualizadoGenero" :
        actualizaciones |=2;
        break;
    case "actualizadoFechaEdicion" :
        actualizaciones |=4;
        break;
    case "actualizadoNumeroPaginas" :
        actualizaciones |=8;
        jLabelNumeroPaginas.setText(Integer.toString(jSLiderNumeroPaginas.getValue())+" páginas");
        break;
```

```
case "actualizadoPremiado" :  
actualizaciones|=16;  
break;  
}
```

Con intención didáctica, se procede a visualizar en consola de salida, los valores que va tomando el atributo en cuestión que codifica la identificación de las columnas que van siendo actualizadas.

El atributo **actualizaciones** interviene en:

- Detección de si, al pulsar el *JButton Aplicar cambios* ante un registro visualizado, procede o no invocar a los métodos de las capas de negocio y datos para dar que se produzca la actualización de dicho registro en la Base de Datos mediante la instrucción SQL UPDATE, en método **responderAController()** de la vista:

```
if (actualizaciones > 0)  
{  
libro.setIdLibro(jTextFieldCodigo.getText());  
new  
LibrosNegocio().actualizar((BaseDatos)controller.g  
etRepositorio()[0], libro,  
actualizaciones);  
}
```

Obviamente, si no se ha producido ninguna actualización en ningún componente, es improcedente activar las invocaciones que conllevan la ejecución del UPDATE. Lo que ha ocurrido, simplemente, es que el usuario ha pulsado el *JButton Aplicar cambios* innecesariamente.

- Identificación de las columnas a aplicar a la cláusula **SET** de la instrucción **UPDATE** en el método **actualizar()** de la

### clase LibrosDatos:

```
for (int i=1; i<=16384; i*=2)
{ int pesoDelBit = (int) actualizaciones & i;
switch (i)
{ case 1 : if (pesoDelBit > 0)
{ ACTUACIONES ASOCIADAS A COLUMNA titulo
}
break;
case 2 : if (pesoDelBit > 0)
{ ACTUACIONES ASOCIADAS A COLUMNA genero
}
break;
case 4 : if (pesoDelBit > 0)
{ ACTUACIONES ASOCIADAS A COLUMNA fecha_edicion
}
break;
case 8 : if (pesoDelBit > 0)
{ ACTUACIONES ASOCIADAS A COLUMNA numero_paginas
}
break;
case 16 : if (pesoDelBit > 0)
{ ACTUACIONES ASOCIADAS A COLUMNA premiado
}
break;
}
}
```

Es necesario repetir el procedimiento en el mismo método, pero ahora para “settear”, los valores a los parámetros que corresponda.

Como ya comentamos con anterioridad, el método **actualizar()** que estamos tratando, es uno de los casos en que se procede a conformar una SQL en tiempo de ejecución.

Si el lector no está familiarizado con la utilización de operadores a nivel de bit, le remitimos al libro Programación Orientada a Objetos en Java, también del mismo autor, todo ello tratado extensamente con los debidos ejemplos en el Capítulo 1, en el apartado correspondiente a **Operadores**.

Se puede observar en el tratamiento aplicado al evento asociado al botón **Aplicar Cambios**, en el método **responderAController()** de la vista, la aplicación de una serie de filtros enfocados a:

- La detección de ausencia de dato en los *JTextField*, a excepción del asociado al **código**.
- Valor coherente de fecha.
- Item seleccionado en *JList* ante nuevo registro.

Todos ellos implementados en la clase **Filtros** del package **utilites**. Cuando un proceso de filtrado detecta una situación anómala, lanza una excepción, que es propagada hasta el método **centralizar()** de la clase **Controller**, en que se procede a su tratamiento, del que hablaremos más adelante.

Centremos la atención en el filtro asociado a la fecha. Hemos aplicado el mecanismo de manipulación de bits antes descrito para disponer en la variable *int codigoError* de la codificación de las unidades de fecha incoherentes o anómalas. Para ello hemos recurrido análogamente al caso anteriormente descrito, asociando un bit a cada una de las unidades de tiempo contempladas (año, mes, día), y a la condición de formato de fecha erróneo, donde un valor de “1” en un bit codifica estado de incoherencia. Al igual que en el caso análogo, inicializamos la variable:

```
int codigoError = 0;
```

y establecemos a “1” el bit correspondiente en la medida vamos detectando la incoherencia asociada:

```
try { . . .
} catch (NumberFormatException excepcion)
{ codigoError |= 1; } // dígito no numérico en fecha
codigoError |= 1; // formato fecha erróneo
if (Integer.toString(año).length() != 4) // año
erróneo
codigoError |= 2;
if (mes < 1 || mes > 12) // mes erróneo
codigoError |= 4;
if (dia < 1 || dia > getDiaMaximoMes(mes, año)) // dia
erróneo
codigoError |= 8;
```

Procediendo antagónicamente, es decir, a la decodificación, y en consecuencia, detección del tipo de anomalía o incoherencia producida en el método **gestionarExcepcion()** de la clase **GestorIncidencias**, siguiendo el mismo procedimiento del caso anterior:

```
String mensajesError[] = {"formato fecha erróneo",
"año erróneo", "mes erróneo", "día erróneo", "número
de páginas erróneo"};
for (int i=1, contadorMensajes=0; i<=16; i*=2,
contadorMensajes++)
{ int codigoErrorFechaParcial = codigoError & i;
if (codigoErrorFechaParcial > 0)
mensajeError += mensajesError[contadorMensajes]+";"
}
```

## OPCIÓN: VISTAUNICATABLA

La vista asociada a la selección de esta opción es:

Gestión libros - Vista Única Tabla - Conectado como usuario usuario1

Conexión-Desconexión Mantenimiento Actividad Generar documentos

Código	Título	Género	Fecha edición	Número páginas	Premiado
00001	EL QUIJOTE	8 - Literatura	12-04-1987	890	<input checked="" type="checkbox"/>
00002	PROGRAMACIÓN ORIENTADA A OBJETOS EN JAVA	0 - Generalidades	20-05-2019	588	<input type="checkbox"/>
00003	POESIAS COMPLETAS	8 - Literatura	18-05-2006	540	<input checked="" type="checkbox"/>
00004	CAMPOS DE CASTILLA	8 - Literatura	18-05-2006	289	<input checked="" type="checkbox"/>
00005	HAMLET	8 - Literatura	01-09-1987	452	<input checked="" type="checkbox"/>
00006	FUENTE OVEJUNA	8 - Literatura	12-11-1973	650	<input checked="" type="checkbox"/>
00007	CUENTOS DE LOS HERMANOS GRIMM	8 - Literatura	22-04-1990	357	<input type="checkbox"/>
00008	LOS IBEROS	9 - Geografía. Historia	05-08-1984	259	<input type="checkbox"/>
00009	HISTORIA DE ROMA	9 - Geografía. Historia	12-09-1988	536	<input type="checkbox"/>
00010	JULIO CESAR	9 - Geografía. Historia	22-11-2001	380	<input type="checkbox"/>
00011	MANUAL DE AUTOAYUDA	1 - Filosofía. Psicología	22-04-2018	540	<input type="checkbox"/>
00012	LA FOTOSÍNTESIS	5 - Matemáticas. Ciencias naturales	30-09-2005	309	<input type="checkbox"/>
00013	GENÉTICA	5 - Matemáticas. Ciencias naturales	02-12-2007	519	<input type="checkbox"/>
00014	LA CELULA	5 - Matemáticas. Ciencias naturales	12-11-2001	648	<input type="checkbox"/>
00015	EL SISTEMA LINFÁTICO	6 - Ciencias aplicadas	15-04-2005	719	<input type="checkbox"/>
00016	EL APAGA TU LIGEREDAD	6 - Ciencias aplicadas	12-08-1998	817	<input type="checkbox"/>
00017	MANUAL DE FISIOTERAPIA	6 - Ciencias aplicadas	29-09-2014	790	<input type="checkbox"/>
00018	NEUROLOGÍA	6 - Ciencias aplicadas	19-12-2010	583	<input type="checkbox"/>
00019	TRAUMATOLOGÍA	6 - Ciencias aplicadas	23-10-2009	490	<input type="checkbox"/>
00020	LOS MISERABLES	8 - Literatura	09-12-1989	590	<input type="checkbox"/>
00021	LA VUELTA AL MUNDO EN 80 DÍAS	8 - Literatura	10-04-1985	457	<input type="checkbox"/>
00022	LOS TRES MOSQUETEROS	8 - Literatura	03-04-1981	615	<input type="checkbox"/>
00023	VIAJE AL CENTRO DE LA TIERRA	8 - Literatura	30-04-2000	403	<input type="checkbox"/>
00024	HISTORIA DE DOS CIUDADES	8 - Literatura	12-11-1980	369	<input type="checkbox"/>
00025	LA ILIADA	8 - Literatura	12-04-1987	245	<input type="checkbox"/>
00026	MIGUEL STROGOFF	8 - Literatura	25-04-2010	481	<input type="checkbox"/>
00027	OLIVER TWIST	8 - Literatura	10-08-1987	340	<input type="checkbox"/>

[Insertar fila al final](#)

CRITERIO ORDENACIÓN :

identificador libro      [Cancelar inserción fila](#)

título      [Guardar fila insertada](#)

género      [Eliminar fila seleccionada](#)

La existencia de este *JPanel* responde a la intención didáctica de mostrar cómo implementar un CRUD mediante un *JTable*. Veamos a continuación el procedimiento a seguir en esta vista para cada una de las opciones:

- **NUEVO REGISTRO:** En primer lugar, el usuario debe pulsar el botón **Insertar fila al final**. A continuación, ha de posicionarse al final del *JTable* mediante la barra de desplazamiento, si ello fuese necesario, y se encontrará con una nueva fila en que cumplimentar los valores para las columnas del nuevo registro. Podemos observar que al igual que en el *JTable* de la aplicación **VotacionPropuesta**, hemos utilizado un *JCheckBox* asociado a la columna **Premiado**:

Gestión libros - Vista Única Tabla - Conectado como usuario usuario

Conexión-Desconexión Mantenimiento Actividad Generar documentos

Código	Título	Género	Fecha edición	Número páginas	Premiado
UUU05	FUENTE OVEJUNA	8 - Literatura	12-11-1973	650	<input checked="" type="checkbox"/>
00007	CUENTOS DE LOS HERMANOS GRIMM	8 - Literatura	22-04-1990	357	<input type="checkbox"/>
00008	LOS IBEROS	9 - Geografía, Historia	05-08-1984	259	<input type="checkbox"/>
00009	HISTORIA DE ROMA	9 - Geografía, Historia	12-09-1968	536	<input type="checkbox"/>
00010	JULIO CESAR	9 - Geografía, Historia	22-11-2001	380	<input type="checkbox"/>
00011	MANUAL DE AUTOAYUDA	1 - Filosofía, Psicología	22-04-2018	540	<input type="checkbox"/>
00012	LA FOTOSÍNTESIS	5 - Matemáticas, Ciencias naturales	30-09-2005	309	<input type="checkbox"/>
00013	GENÉTICA	5 - Matemáticas, Ciencias naturales	02-12-2007	519	<input type="checkbox"/>
00014	LA CELULA	5 - Matemáticas, Ciencias naturales	12-11-2001	648	<input type="checkbox"/>
00015	EL SISTEMA LINFÁTICO	6 - Ciencias aplicadas	15-04-2005	719	<input type="checkbox"/>
00016	EL APARATO DIGESTIVO	6 - Ciencias aplicadas	12-08-1998	817	<input type="checkbox"/>
00017	MANUAL DE FISIOTERAPIA	6 - Ciencias aplicadas	29-08-2014	790	<input type="checkbox"/>
00018	NEUROLOGÍA	6 - Ciencias aplicadas	19-12-2010	583	<input type="checkbox"/>
00019	TRAUMATOLÓGIA	6 - Ciencias aplicadas	23-10-2009	490	<input type="checkbox"/>
00020	LOS MISERABLES	8 - Literatura	09-12-1989	590	<input type="checkbox"/>
00021	LA VUELTA AL MUNDO EN 80 DÍAS	8 - Literatura	10-04-1985	457	<input type="checkbox"/>
00022	LOS TRES MOSQUETEROS	8 - Literatura	03-04-1981	615	<input type="checkbox"/>
00023	VIAJE AL CENTRO DE LA TIERRA	8 - Literatura	30-04-2000	403	<input type="checkbox"/>
00024	HISTORIA DE DOS CIUDADES	8 - Literatura	12-11-1980	369	<input type="checkbox"/>
00025	LA ILIADA	8 - Literatura	12-04-1987	245	<input type="checkbox"/>
00026	MIGUEL STROGOFF	8 - Literatura	25-04-2010	481	<input type="checkbox"/>
00027	OLIVER TWIST	8 - Literatura	10-08-1987	340	<input type="checkbox"/>
00028	LA ODISÉA	8 - Literatura	03-08-1986	290	<input type="checkbox"/>
00029	GUERRA Y PAZ	8 - Literatura	12-11-1987	810	<input type="checkbox"/>
00030	EL LAZARILLO DE TORMES	8 - Literatura	22-04-1983	356	<input type="checkbox"/>
00031	LA TIA TULA	8 - Literatura	12-11-1968	656	<input checked="" type="checkbox"/>

0

     Fila insertada al final de la tabla
  
**CRITERIO ORDENACION :**
  
 **Identificador libro**      
  
 **título**      
  
 **género**

Eliminar fila seleccionada
  
Antes de pulsar el botón Guardar sitúe el cursor en columna Código de la fila

El usuario procede a cumplimentar los valores en todas las columnas para la nueva fila. Observemos que la edición de la columna **Género** se efectúa por selección de un ítem del *JComboBox* asociado a dicha columna. Observaremos también un *ToolTipText* que aparece al posicionarse sobre dicha columna.

Gestión libros • Vista Única Tabla • Conectado como usuario: usuario

Conexión-Desconexión Mantenimiento Actividad Generar documentos

Código	Título	Género	Fecha edición	Número páginas	Premiado
UUU005	FUENTE OVEJUNA	8 - Literatura	12-11-1973	650	<input checked="" type="checkbox"/>
00007	CUENTOS DE LOS HERMANOS GRIMM	8 - Literatura	22-04-1990	357	<input type="checkbox"/>
00008	LOS IBEROS	9 - Geografía. Historia	05-08-1964	259	<input type="checkbox"/>
00009	HISTORIA DE ROMA	9 - Geografía. Historia	12-09-1968	536	<input type="checkbox"/>
00010	JULIO CESAR	9 - Geografía. Historia	22-11-2001	380	<input type="checkbox"/>
00011	MANUAL DE AUTOAYUDA	1 - Filosofía. Psicología	22-04-2018	540	<input type="checkbox"/>
00012	LA FOTOSÍNTESIS	5 - Matemáticas. Ciencias naturales	30-09-2005	309	<input type="checkbox"/>
00013	GENÉTICA	5 - Matemáticas. Ciencias naturales	02-12-2007	519	<input type="checkbox"/>
00014	LA CELULA	5 - Matemáticas. Ciencias naturales	12-11-2001	648	<input type="checkbox"/>
00015	EL SISTEMA LINFÁTICO	6 - Ciencias aplicadas	15-04-2005	719	<input type="checkbox"/>
00016	EL APARATO DIGESTIVO	6 - Ciencias aplicadas	12-08-1998	817	<input type="checkbox"/>
00017	MANUAL DE FISIOTERAPIA	6 - Ciencias aplicadas	29-08-2014	790	<input type="checkbox"/>
00018	NEUROLOGÍA	6 - Ciencias aplicadas	19-12-2010	583	<input type="checkbox"/>
00019	TRAUMATOLOGÍA	6 - Ciencias aplicadas	23-10-2009	490	<input type="checkbox"/>
00020	LOS MISERABLES	8 - Literatura	09-12-1969	590	<input type="checkbox"/>
00021	LA VUELTA AL MUNDO EN 80 DÍAS	8 - Literatura	10-04-1065	457	<input type="checkbox"/>
00022	LOS TRES MOSQUETEROS	8 - Literatura	03-04-1981	615	<input type="checkbox"/>
00023	VIAJE AL CENTRO DE LA TIERRA	8 - Literatura	30-04-2000	403	<input type="checkbox"/>
00024	HISTORIA DE DOS CIUDADES	8 - Literatura	12-11-1980	369	<input type="checkbox"/>
00025	LA ILIADA	8 - Literatura	12-04-1987	245	<input type="checkbox"/>
00026	MIGUEL STROGOFF	8 - Literatura	25-04-2010	481	<input type="checkbox"/>
00027	OLIVER TWIST	8 - Literatura	10-08-1987	340	<input type="checkbox"/>
00028	LA ODISEA	8 - Literatura	03-08-1986	290	<input type="checkbox"/>
00029	GUERRA Y PAZ	8 - Literatura	12-11-1987	810	<input type="checkbox"/>
00030	EL LAZARILLO DE TORMES	8 - Literatura	22-04-1983	358	<input type="checkbox"/>
00031	LA TIA TULA	8 - Literatura	12-11-1968	658	<input checked="" type="checkbox"/>
	LA SOMBRA DEL VIENTO	8 - Literatura	15-10-2001	582	<input type="checkbox"/>
		2 - Religión. Teología			
		3 - Ciencias sociales			
		4 - No clasificado			
		5 - Matemáticas. Ciencias naturales			
		6 - Ciencias aplicadas			
		7 - Bellas artes. Deportes			
		8 - Literatura			
		9 - Geografía. Historia			

Insertar fila al final

CRITERIO ORDENACIÓN :

identificador libro

título

género

Cancelar inserción fila

Guardar fila insertada

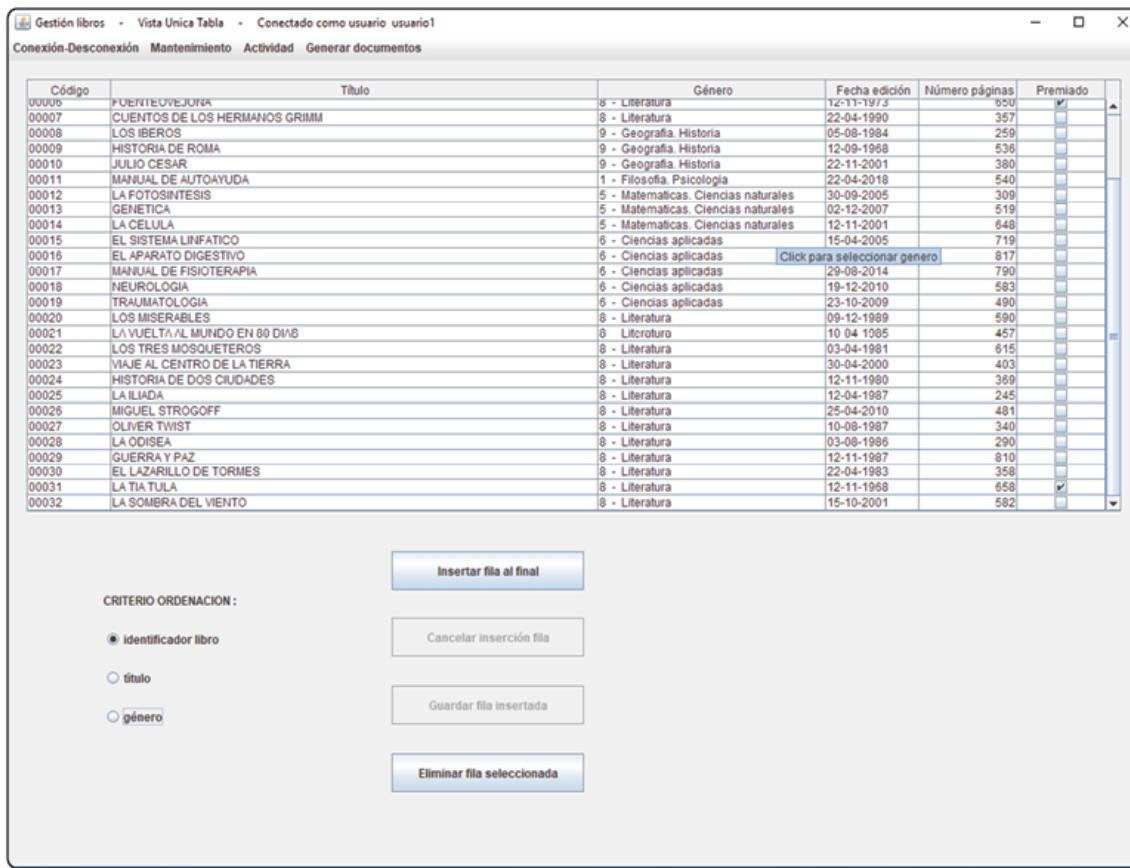
Antes de pulsar el botón Guardar sitúe el cursor en columna Código de la fila

Eliminar fila seleccionada

Llamamos la atención del lector a observar que ahora solamente se encuentran activos los botones que permiten registrar la fila en la base de datos, o cancelar el proceso de edición de la nueva fila.

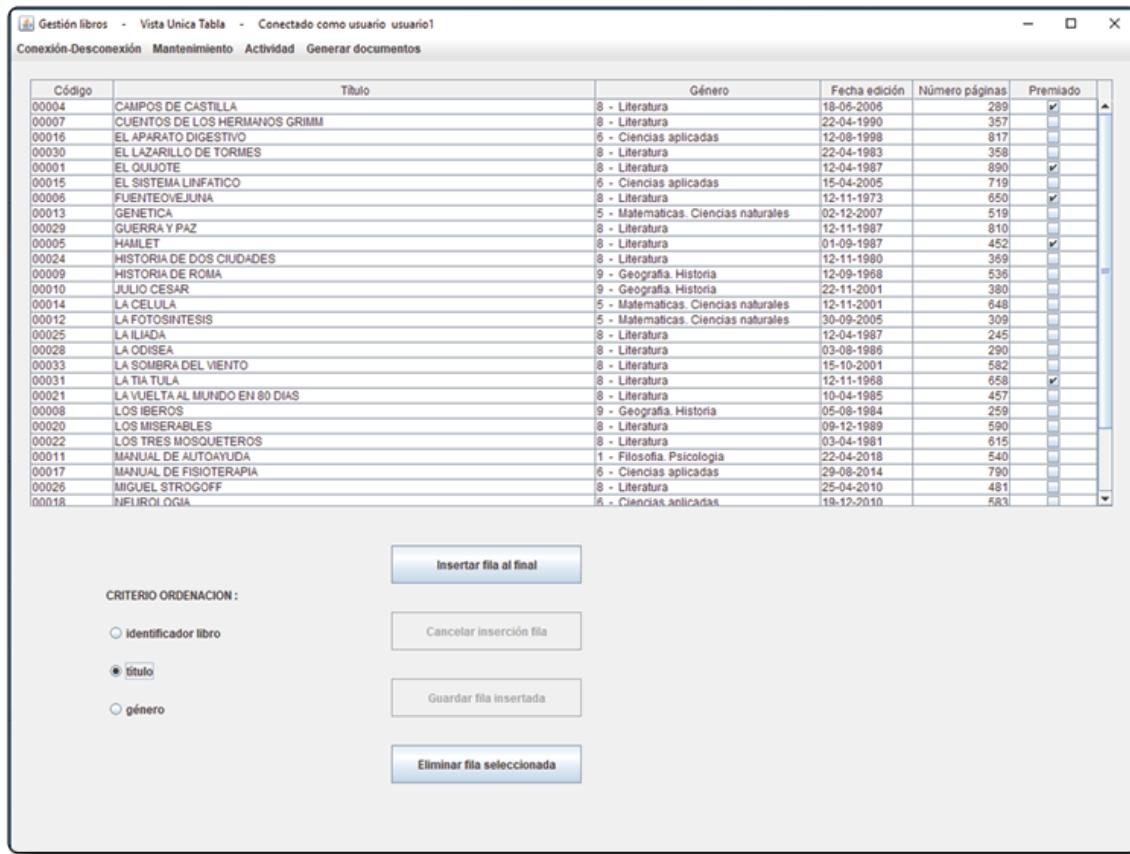
Una vez cumplimentada la información para todas las columnas, a excepción de **Código**, que no es editable, por tratarse de un dato generado mediante un autonumérico, ya se encuentra el usuario en condiciones de pulsar el botón **Guardar fila insertada**, actuación que comportará la inserción en la Base de Datos. Para que la operación se realice exitosamente, al pulsar el botón, el cursor ha de estar situado en cualquier celda que no sea una de las que hemos editado. Ello obedece a que el proceso de filtrado, ya mencionado en el

*JPanel VistaFormulario*, necesita, para detectar que todas las celdas de la nueva fila han sido cumplimentadas, que el cursor se encuentre fuera de ellas. Una de las posibilidades es que se encuentre ubicado en la celda correspondiente a la columna **Código**, que, como ya hemos comentado, no es editable. Si la operación ha resultado exitosa, se restaura el estado de activación de botones al que había con anterioridad al inicio del proceso de edición de la nueva fila, quedando “refrescado” el *JTable*, apareciendo la nueva fila como una más:



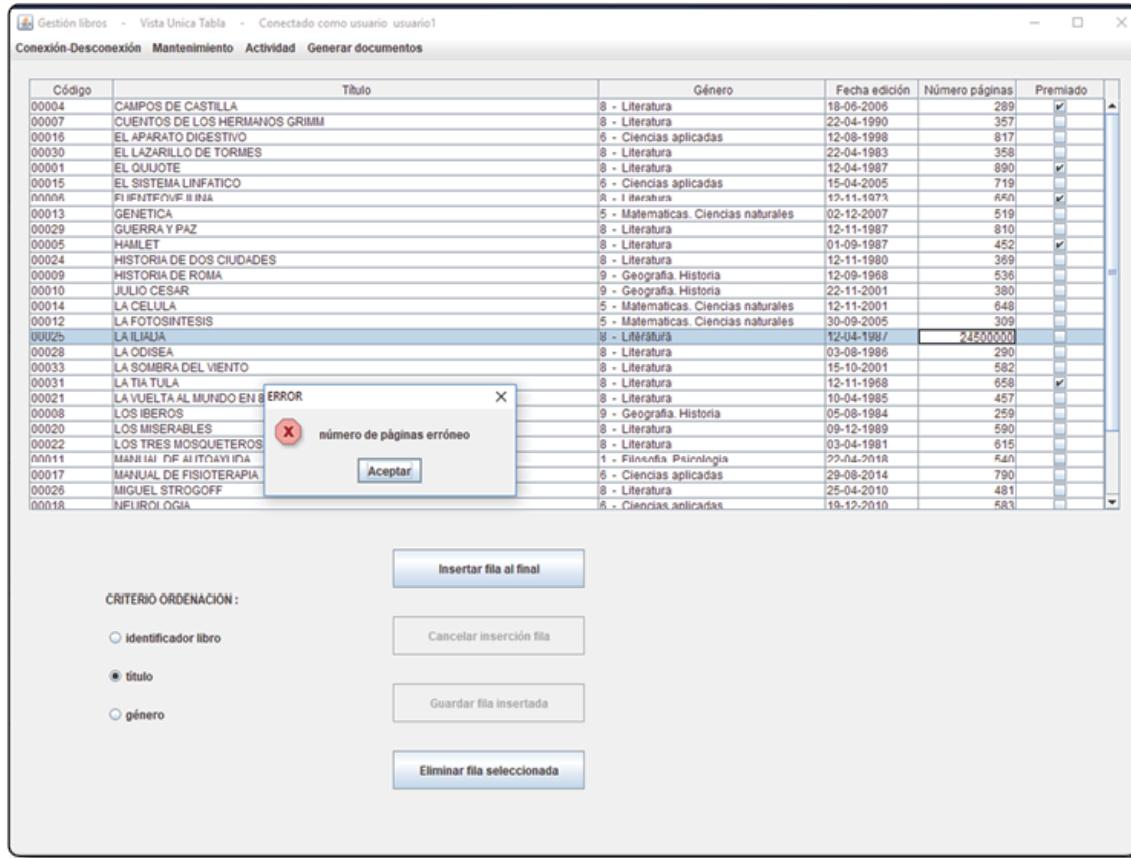
- **CONSULTAR REGISTRO:** Esta operación es inherente a la presencia de todas las filas visualizadas en el *JTable*. El criterio de ordenación aplicado por defecto al orden de aparición de las filas es por la columna Código. A efectos

de poder cambiar dicho criterio de ordenación, la pantalla presenta el grupo de *JRadioButton*. Si cambiamos la selección al *JRadioButton* **jRadioButtonTitulo**, comprobaremos que aparecen nuevamente visualizadas las filas en el *JTable* pero ordenadas por la columna **título**:



- **ACTUALIZAR REGISTRO:** Para modificar el valor de una celda, hay que posicionarse en ella, modificar el dato, y con sólo retirar el posicionamiento del cursor de dicha celda, ya se produce la actualización de la Base de Datos con el nuevo dato. Caso de tratarse de una columna sujeta a un proceso de filtrado, si el nuevo dato no cumpliese con los requerimientos del filtro, al retirar el cursor de dicha

celda, aparecería una ventana que indica al usuario la condición de error. Al cerrarla el valor anterior es automáticamente restaurado.



- **ELIMINAR REGISTRO:** En primer lugar, se selecciona la fila que pretendemos eliminar, y ya estamos en condiciones de pulsar el *JButton Eliminar fila seleccionada*, actuación que dará lugar a su eliminación de la Base de Datos.

Una parte considerable de los mecanismos utilizados en esta opción, ya aparecieron y fueron tratados en el estudio del *JTable* núcleo de la ventana correspondiente a la aplicación

**VotacionPropuesta.** El método **cargar()** tiene la misma función que sus homólogos **cargarTitulares()** y **cargarVotacion()** de la aplicación **VotacionPropuesta**, es decir, almacenar en el array

```
Object datos[][][]
```

del Model los datos que serán visualizados en el *JTable*, y que se le transfieren al método mediante un *List*, procedentes en ambas aplicaciones, del resultado de consultas SQL.

Abordaremos solamente las aportaciones novedosas incorporadas a esta opción. Para la comprensión del código implementado que ocasiona el comportamiento comentado del *JTable*, es necesario iniciar el enfoque desde el tratamiento aplicado a las escuchas:

- **TableModelListener.** Esta escucha interviene en el proceso de modificación del valor de una celda, tanto cuando se trata de una nueva fila a insertar, como cuando se pretende actualizar el valor de una celda en una fila previamente existente. En el método *tableChanged()* nos limitamos a registrar la fila y columna en que se ha producido la actualización. Utilizamos para ello dos componentes del array **componentesJPanel**:

```
pantallaOpcion.setComponenteJPanel(new
Integer(e.getFirstRow()), 13);
pantallaOpcion.setComponenteJPanel(new
Integer(e.getColumn()), 14);
```

El registro de esta información nos va a permitir acceder al nuevo dato que participa en la actualización de la celda, desde el método **responderAController()** del *JPanel*, para un valor de *actionCommand*

**“actualizadaColumnaJTable”.** Con dicho dato procedemos como a continuación se describe:

- Aplicar el correspondiente filtro en el caso de que la información de la columna sea susceptible de aplicársele un proceso de filtrado. Si el nuevo dato introducido no superó el proceso de filtrado, se lanza una excepción que es propagada hasta el método **centralizar()** de la clase **Controller**, en que se le da un tratamiento centralizado que describiremos más adelante. Solamente señalar cuando la situación de “rechazo” de un dato por el filtro se produce en el *JTable*, adicionalmente al tratamiento que se le da a cualquier otra excepción, se restaura el valor anterior. Para la implementación de esta actuación, instrumentalizamos el atributo

Object copiaReservaDato  
del Model:

- En el método *setValueAt()* del Model “guardamos” siempre una copia de reserva del dato que va a ser reemplazado, por si procediese restaurarlo:

copiaReservaDato = datos[fila][columna];

- Caso de proceder la restauración del dato anterior, en el tratamiento centralizado de la excepción implementamos

```
((ModeloDatos)pantallaOpcion.getComponente JPanel(15)).getDatos()  
[((Integer)pantallaOpcion.getComponente JPanel(13))  
.intValue()]  
[((Integer)pantallaOpcion.getComponente JPanel(14))
```

```
.intValue()]) =  
((ModeloDatos)pantallaOpcion.getComponenteJPanel(1  
5)).  
getCopiarReservaDato();
```

- Las siguientes actuaciones están supeditadas a que se trate de la actualización de una celda de una columna previamente existente. Tal circunstancia es detectada al comprobar la existencia de dato en la primera columna (**Código**), que de cumplirse:
  - Instanciamos un objeto de la clase Libro, en el que procedemos a encapsular, además del identificador, información en los siguientes atributos:
    - Object datoActualizado
    - int columnaActualizada
  - Procedemos a la actualización en la Base de Datos transfiriendo a la capa de negocio la información que hemos encapsulado en la instancia **Libro**:

```
new LibrosNegocio().  
actualizar((BaseDatos)controller.getRepositorio()  
[0], libro, -1);
```

El valor -1 del último parámetro determina la invocación del método:

```
new LibrosDatos().actualizarColumna(connection,  
libro);
```

en que se produce la actualización de la columna con el nuevo valor.

- **ListSelectionListener**. Utilizado hasta ahora para la escucha de eventos del *JList*, novedosamente lo aplicamos en esta aplicación a la escucha de eventos del *JTable*,

además del *JList*. De tal modo que en el método *valueChanged()*, la actuación más inmediata es discernir en qué *JPanel* se ha lanzado el evento:

```
switch(pantallaOpcion.getClass().getName())
{ case "presentacion.VistaUnicaTabla":
case "presentacion.VistaPaginadaTabla":
    . . .
}
break;
case "presentacion.VistaFormulario":
    . . .
break;
}
```

Si el *JPanel* origen ha sido **VistaUnicaTabla** o **VistaPaginadaTabla**, el componente origen habrá sido un *JTable*, y la causa habrá sido la selección de una fila. En ambos casos procedemos a registrar en un componente del array **componentesJPanel** el número de la fila seleccionada:

```
pantallaOpcion.setComponenteJPanel(new Integer
(listSelectionModel.getMinSelectionIndex()), 2);
```

Valor que nos permitirá proceder a la eliminación de dicha fila, si se pulsa el botón **Eliminar fila seleccionada**.

Y si el *JPanel* origen ha sido *VistaFormulario*, el evento se habrá lanzado al seleccionar el usuario un ítem del *JList*.

Para la inserción de nueva fila, se invoca, en primer lugar, al método:

```
modeloDatos.insertarFila();
```

del Model. Observando la implementación de dicho método, deducimos que realmente actuamos sobre el array *Object datos[]*

[], produciendo un reajuste de los componentes, añadiendo una fila más. Aunque, hablando en propiedad, dado que se trata de arrays, deberíamos decir que se ha añadido la posibilidad de disponer de un valor más en primera dimensión, y en consecuencia un conjunto más de componentes disponibles en segunda dimensión. A continuación, establecemos como seleccionada la última fila del nuevo conjunto obtenido del *JTable*:

```
listSelectionModel.setSelectionInterval((modeloDatos.getDatos().length)-1,  
(modeloDatos.getDatos().length)-1);
```

y registramos su valor en el componente del array **componentesJPanel** a que hemos hecho alusión anteriormente:

```
componentesJPanel[2] = new  
Integer((modeloDatos.getDatos().length)-1);
```

Si durante el proceso de inserción de nueva fila, el usuario decide no proseguir, pulsaría el botón **Cancelar inserción fila**, que revertiría el proceso. Para ello, se invoca, en primer lugar, al método el método:

```
modeloDatos.eliminarFilaInsertada();
```

en que actuamos análogamente como lo hemos hecho en el método **insertarFila()**, reajustando el número de componentes, pero obteniendo el resultado inverso, es decir, disponiendo de un valor menos en primera dimensión.

Y si el usuario se decantase por la inserción del nuevo registro en la Base de Datos al pulsar el botón **Guardar fila insertada**, el proceso es análogo al seguido al pulsar el botón **Aplicar cambios** del *JPanel VistaFormulario*, procediendo en primer lugar a la aplicación del proceso de filtrado pertinente. Superado el proceso de filtrado, se produciría la inserción en la Base de Datos, además

de otros efectos visuales como la eliminación de la selección de fila, y el cambio de activación en botones.

## OPCIÓN: VISTAPAGINADATABLA

La vista asociada a la selección de esta opción es:

Código	Título	Género	Fecha edición	Número páginas	Premiado
00001	EL QUIJOTE	B - Literatura	12-04-1907	890	<input checked="" type="checkbox"/>
00002	PROGRAMACIÓN ORIENTADA A OBJETOS EN JAVA	B - Generalidades	20-05-2019	588	<input type="checkbox"/>
00003	POESÍAS COMPLETAS	B - Literatura	18-05-2006	540	<input checked="" type="checkbox"/>
00004	CAMPÓS DE CASTILLA	B - Literatura	18-05-2006	289	<input checked="" type="checkbox"/>
00005	HAMLET	B - Literatura	01-09-1987	450	<input checked="" type="checkbox"/>

Mediante esta opción, pretendemos recrear un escenario de “paginación”. En realidad, dicho escenario solamente es habitual en aplicaciones web. A pesar de que es una circunstancia insólita contemplar esta solución en aplicaciones con interfaz gráfica de escritorio, ofrece indudables ventajas en relación con la anterior, la aplicada a **VistaUnicaTabla**. En la anterior, el componente, en este caso el *JTable*, mantiene la totalidad de las filas a ser mostradas. Es una solución aceptable mientras nos movamos en rango “moderado” de filas, en que, con un leve movimiento de la barra de desplazamiento, tenemos rápidamente acceso a cualquier parte del conjunto. Pero en la medida crece el número de filas del conjunto a mostrar, empieza a volverse impracticable. Con la solución que ofrecemos en esta opción de dar acceso

“paginadamente” a las filas proporcionamos un acceso mucho más rápido a cualquier fila del conjunto, por distantes que estén respecto a la presentada, y al mismo tiempo proporciona noción al usuario de la cantidad de filas accesibles al mostrar el número de la última página. En el caso de las pantallas que estamos mostrando, solamente aparecen cinco filas por página, pero pensemos que una página podría contemplar muchas más, que, aunque no fuesen presentadas visualmente en su totalidad, se podría recurrir a la utilización de la barra de desplazamiento, tal y como se hace en el caso anterior. Las ventajas de la solución paginada en relación a la solución de presentación monolítica aumentan proporcionalmente al número total de filas a dejar accesibles al usuario.

Se ha incluido en este libro con el fin de proporcionar perspectiva al lector en la lógica algorítmica asociada al proceso, que viene a ser la misma que cuando el proceso de paginación es utilizado en aplicaciones web. Comparte con el *JPanel VistaUnicaTabla* las funcionalidades de eliminar fila seleccionada, y la aplicación de tres criterios diferentes en el orden de aparición de las filas.

Volviendo a la pantalla que hemos mostrado comprobaremos que “el juego” aplicado de botones para la paginación es:

- Botón de acceso a página anterior (“<”), que aparece solamente si la página mostrada no es la primera.
- Botón que da acceso a página 1.
- Puntos suspensivos a mostrar solamente en el caso de que el número que aparece como texto en el primer botón del bloque siguiente de botones numéricos, no sea

inmediatamente posterior al “1”, es decir, que no se trate del “2”.

- Bloque de botones numéricos cuyo número viene determinado por el parámetro

`<NUMERO_BOTONES_NUMERICOS>`

Los valores numéricos de los botones se ajustan para que, en la medida de lo posible, el número de la página activa ocupe la posición central.

- Puntos suspensivos a mostrar solamente en el caso de que el número que aparece como texto en el último botón del bloque de botones numéricos consecutivos, no sea inmediatamente anterior al último.
- Botón que da acceso a la última página, en que aparece como texto el número de dicha última página, circunstancia que proporciona noción al usuario del número de páginas totales existentes.
- Botón de acceso a página siguiente (“<”), que aparece solamente si la página mostrada no es la última.

Observaremos que el texto que aparece en el botón cuyo valor numérico coincide con el de la página activa, se muestra con una apariencia diferente a los demás, a efectos de que el usuario pueda identificar el número de página que está siendo mostrada en esos momentos. El valor del número de página correspondiente a la mostrada en la siguiente pantalla permite que con la configuración de botones asociada aparezcan todos los botones y elementos contemplados:

Código	Título	Género	Fecha edición	Número páginas	Premiado
00016	EL APARATO DIGESTIVO	6 - Ciencias aplicadas	12-08-1998	817	
00017	MANUAL DE FISIOTERAPIA	6 - Ciencias aplicadas	29-08-2014	790	
00018	NEUROLOGIA	6 - Ciencias aplicadas	19-12-2010	583	
00019	TRAUMATOLOGIA	6 - Ciencias aplicadas	23-10-2009	490	
00020	LOS MISERABLES	9 - Literatura	09-12-1989	590	

En línea con el propósito que venimos manteniendo de facilitar al máximo la modificabilidad y la escalabilidad de cualquier aplicación, hemos desviado al fichero externo de configuración repositorio.xml la parametrización de los valores que intervienen en cualquier proceso de paginación:

```
<PARAMETROS_PAGINACION>
<NUMERO_FILAS_PAGINA>5</NUMERO_FILAS_PAGINA>
<NUMERO_BOTONES_NUMERICOS>3</NUMERO_BOTONES_NUMERICOS>
</PARAMETROS_PAGINACION>
```

Mostramos a continuación pantalla en que hemos establecido a “25” el parámetro

```
<NUMERO_FILAS_PAGINA>25</NUMERO_FILAS_PAGINA>
```

dejando inalterado el otro:

Gestión libros - Vista Página Tabla - Conectado como usuario usuario1

Conexión Desconexión Mantenimiento Actividad Generar documentos

Código	Título	Género	Fecha edición	Número páginas	Premiado
00001	EL QUIJOTE	8 - Literatura	12-04-1987	890	<input checked="" type="checkbox"/>
00002	PROGRAMACIÓN ORIENTADA A OBJETOS EN JAVA	0 - Generalidades	20-05-2019	568	<input type="checkbox"/>
00003	POESÍAS COMPLETAS	8 - Literatura	18-05-2005	540	<input checked="" type="checkbox"/>
00004	CAMPLOS DE CASTILLA	8 - Literatura	18-05-2006	289	<input checked="" type="checkbox"/>
00005	HAMLET	8 - Literatura	01-09-1987	452	<input checked="" type="checkbox"/>
00006	FUENTEVIEJUNA	8 - Literatura	12-11-1973	650	<input checked="" type="checkbox"/>
00007	CUENTOS DE LOS HERMANOS GRIMM	8 - Literatura	22-04-1990	357	<input type="checkbox"/>
00008	LOS IBEROS	9 - Geografía. Historia	05-08-1984	259	<input type="checkbox"/>
00009	HISTORIA DE ROMA	9 - Geografía. Historia	12-09-1968	536	<input type="checkbox"/>
00010	JULIO CESAR	9 - Geografía. Historia	22-11-2001	380	<input type="checkbox"/>
00011	MANUAL DE AUTOAYUDA	1 - Filosofía. Psicología	22-04-2018	540	<input type="checkbox"/>
00012	LA FOTOSINTESIS	5 - Matemáticas. Ciencias naturales	30-09-2005	309	<input type="checkbox"/>
00013	GENÉTICA	5 - Matemáticas. Ciencias naturales	02-12-2007	519	<input type="checkbox"/>
00014	LA CELULA	5 - Matemáticas. Ciencias naturales	12-11-2001	648	<input type="checkbox"/>
00015	EL SISTEMA LINFÁTICO	6 - Ciencias aplicadas	15-04-2005	719	<input type="checkbox"/>
00016	EL APARATO DIGESTIVO	6 - Ciencias aplicadas	12-08-1998	817	<input type="checkbox"/>
00017	MANUAL DE FISIOTERAPIA	6 - Ciencias aplicadas	29-08-2014	790	<input type="checkbox"/>
00018	NEUROLOGÍA	6 - Ciencias aplicadas	19-12-2010	583	<input type="checkbox"/>
00019	TRAUMATOLÓGIA	6 - Ciencias aplicadas	23-10-2009	490	<input type="checkbox"/>
00020	LOS MISERABLES	8 - Literatura	09-12-1989	590	<input type="checkbox"/>
00021	LA VUELTA AL MUNDO EN 80 DÍAS	8 - Literatura	10-04-1985	457	<input type="checkbox"/>
00022	LOS TRES MOSQUETEROS	8 - Literatura	03-04-1981	615	<input type="checkbox"/>
00023	VIAJE AL CENTRO DE LA TIERRA	8 - Literatura	30-04-2000	403	<input type="checkbox"/>
00024	HISTORIA DE DOS CIUDADES	8 - Literatura	12-11-1980	372	<input checked="" type="checkbox"/>
00025	LA ILIADA	8 - Literatura	12-04-1987	245	<input type="checkbox"/>

1    2    >

CRITERIO ORDENACION :

Identificador libro

título

género

Eliminar fila seleccionada

Comprobamos que, con tan solo modificar dicho valor en este fichero de configuración, el número de filas de cada “página” trasladada al *JTable*, ya se acoge al nuevo valor. Análogamente responderá al nuevo valor parametrizado en el fichero, el número de botones de paginación numérica que aparecen, además de los correspondientes a la primera y a la última página. También se observa que, al aplicar el cambio, el incremento del tamaño del *JTable* no hay riesgo en que se superponga sobre los elementos que aparecen en la parte inferior (botones de paginación, grupo de *JRadioButton* de selección de criterio de ordenación, etc.), pues el código del programa ajusta la posición de estos en la parte inferior del *JTable*. Invitamos al lector a que haga diversas pruebas alterando el valor de los dos parámetros. Para comprobar la rentabilidad de los cambios aplicados, recomendamos que se incremente el número de filas de la tabla **libros**.

Centrándonos en las cuestiones meramente técnicas de esta opción, la gestión de este JPanel no supone la necesidad de incorporar nuevas estrategias relacionadas con el *JTable*, los Listeners, o el Model, en relación a las que ya se trataron en el estudio de **VistaUnicaTabla**. Es más, hay que señalar que **VistaUnicaTabla** y **VistaPaginadaTabla** utilizan como Model instancias de la misma clase **ModeloDatos**. Esta compartición de Model por ambos *JPanel* obliga a que en algunas ocasiones nos veamos obligados a proporcionar un trato diferenciado al tratamiento en función de la clase de que se trate. Esta circunstancia queda reflejada en el método **cargar()**:

```
switch(controller.getPantallaOpcion().getClass().getName())
{ case "presentacion.VistaUnicaTabla": numeroFilas =
listaLibros.size();
break;
case "presentacion.VistaPaginadaTabla": numeroFilas =
((ParametrosPaginacion)controller.getRepositorio()
[4]).getNumeroFilasPagina();
break;
}
```

y en el método **isCellEditable()**:

```
switch(controller.getPantallaOpcion().getClass().getName())
{ case "presentacion.VistaUnicaTabla": if (columna >0)
editable = true;
break;
case "presentacion.VistaPaginadaTabla":
if (fila < numeroFilasRecibidas && columna >0)
editable = true;
break;
}
```

La expresión adicional

```
fila < numeroFilasRecibidas
```

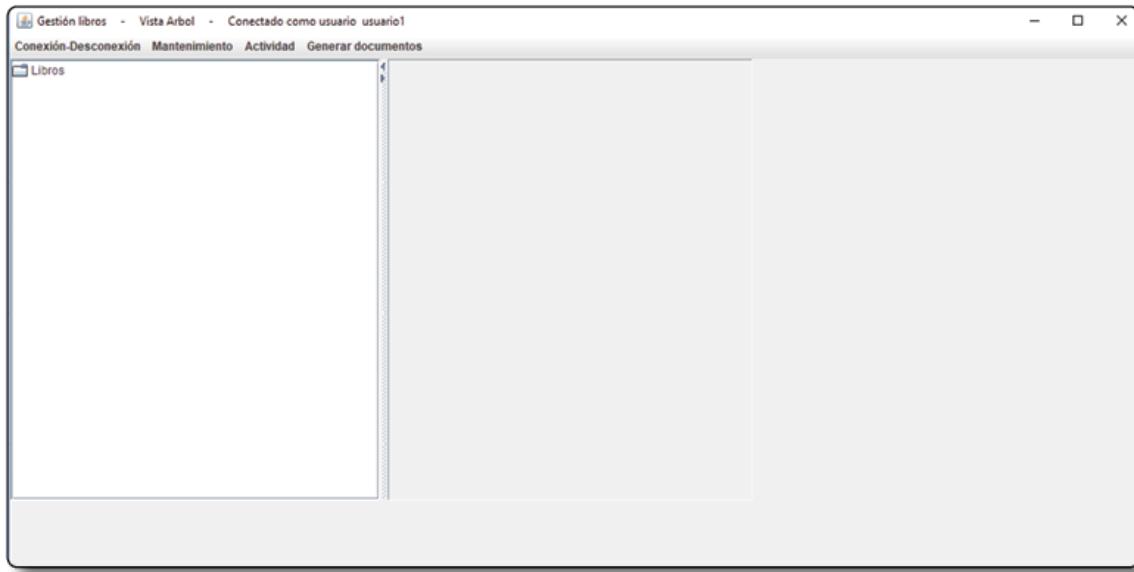
en el caso de **VistaPaginadaTabla**, determina que además de impedir la edición de la primera columna (“**Código**”), también impide la edición de las filas que aparecen vacías en la última página si el número total de filas no es múltiplo exacto del número de filas por página.

Faltaría por dedicar un estudio exhaustivo a la lógica algorítmica asociada al proceso de “paginación”. Para ello no sería necesario aportar nuevos conceptos, contenidos, técnicas o estrategias asociadas a Swing adicionales a las tratadas hasta ahora, que suponen realmente el objeto de estudio en este libro. Dado el nivel avanzado de este libro, no dudamos de que el lector no encontrará problemas en la comprensión del código utilizado para implementar la paginación.

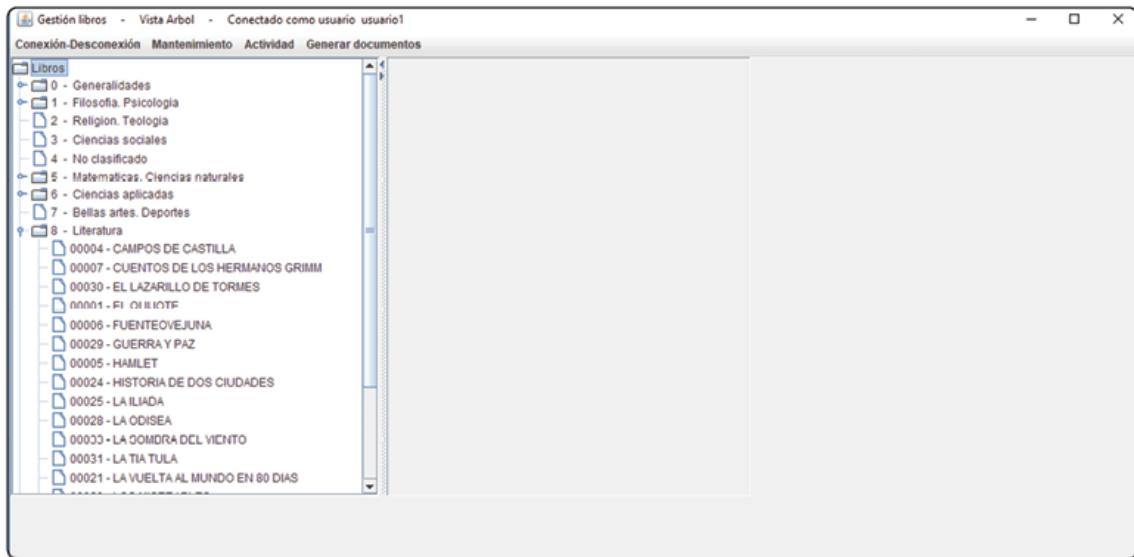
La instrucción SQL encargada de devolver las filas de la siguiente “página” a mostrar en el proceso de paginación es la presentada en séptimo lugar en la relación correspondiente que hemos presentado con anterioridad.

## OPCIÓN: VISTAARBOL

La principal pretensión del autor con la inclusión de esta opción en la aplicación, es presentar un *JTree* “dotado” de actividad interactiva, hasta el punto de que nos encontraremos nuevamente con la implementación de un CRUD. Conforme vayamos avanzando, auxiliados por las respectivas pantallas, en la observación del comportamiento de la aplicación, iremos comprobando que esta opción reúne todas las funciones de CRUD. Al seleccionar esta opción, la vista que se presenta al usuario es:



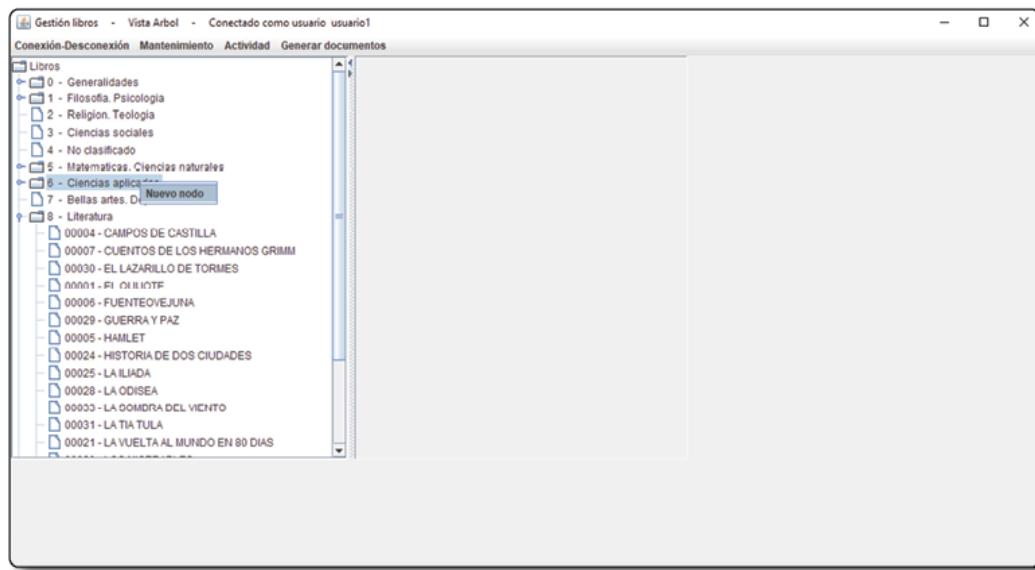
Observando la apariencia del nodo raíz, se deduce, que, aunque replegado, el *JTree* está cargado con información. Si procedemos a su despliegue:



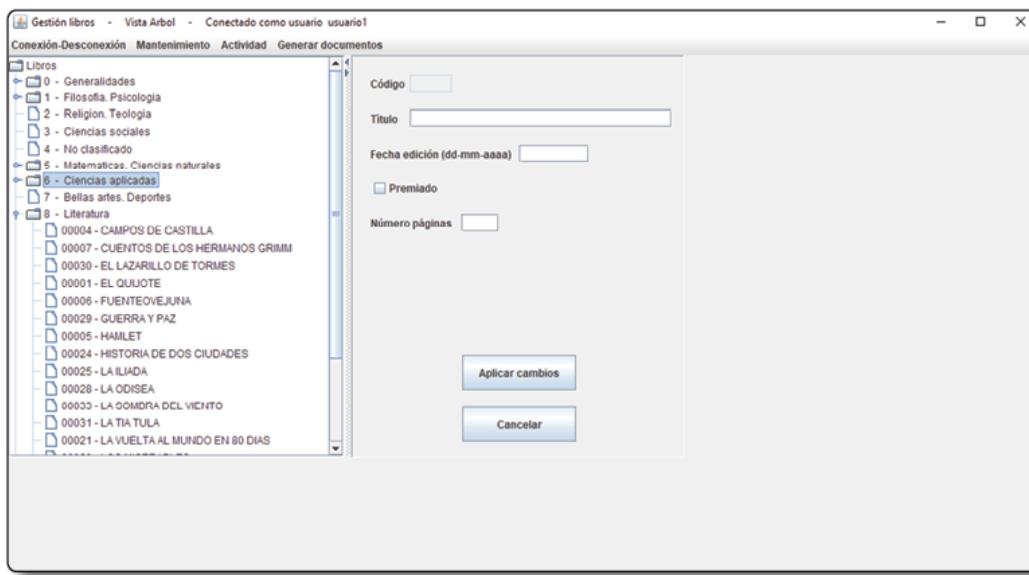
Comprobamos, que además del nodo raíz, nos encontramos con dos niveles adicionales de nodos: uno asociado a **géneros**, permitiendo agrupar los libros por esta característica, y otro

asociado a **libro**. La actividad interactiva con que hemos dotado al *JTree* se manifiesta en:

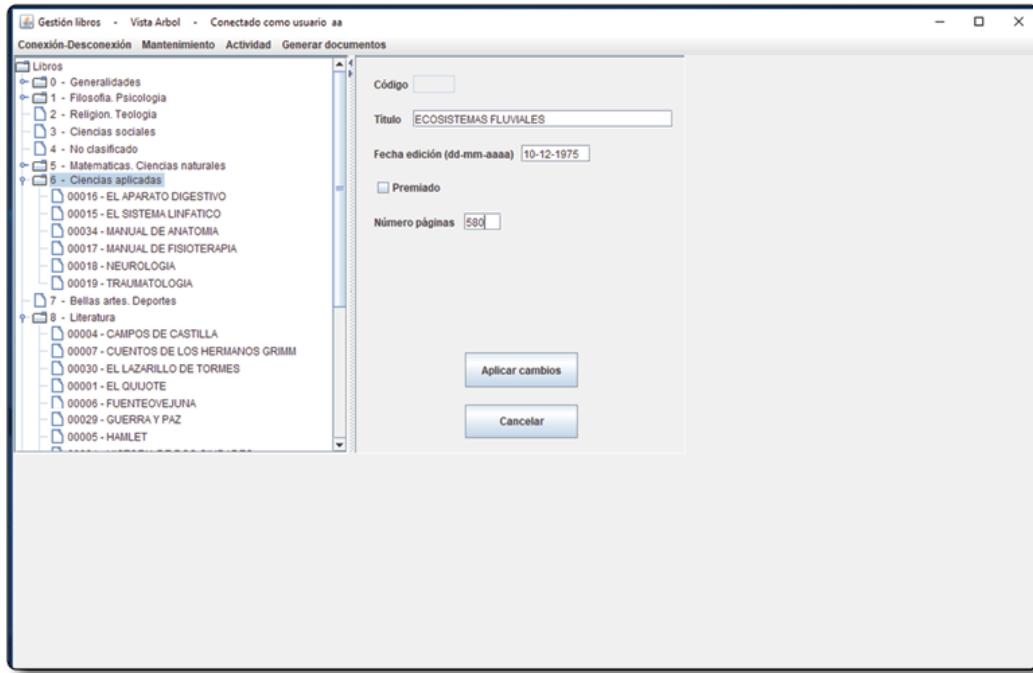
1. Si estando seleccionado un nodo tipo **género**, hacemos click con el botón derecho del ratón, se despliega un menú contextual, tal y como se muestra en:



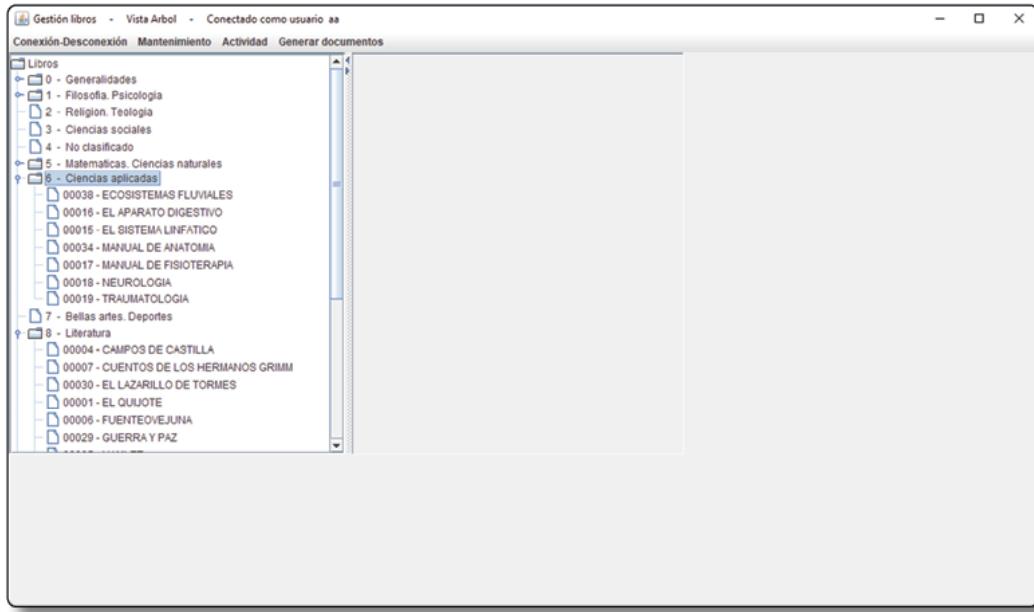
Con una sola opción **Nuevo nodo**, que permite insertar un nuevo nodo de tipo libro con el género del nodo seleccionado. Si seleccionamos la opción:



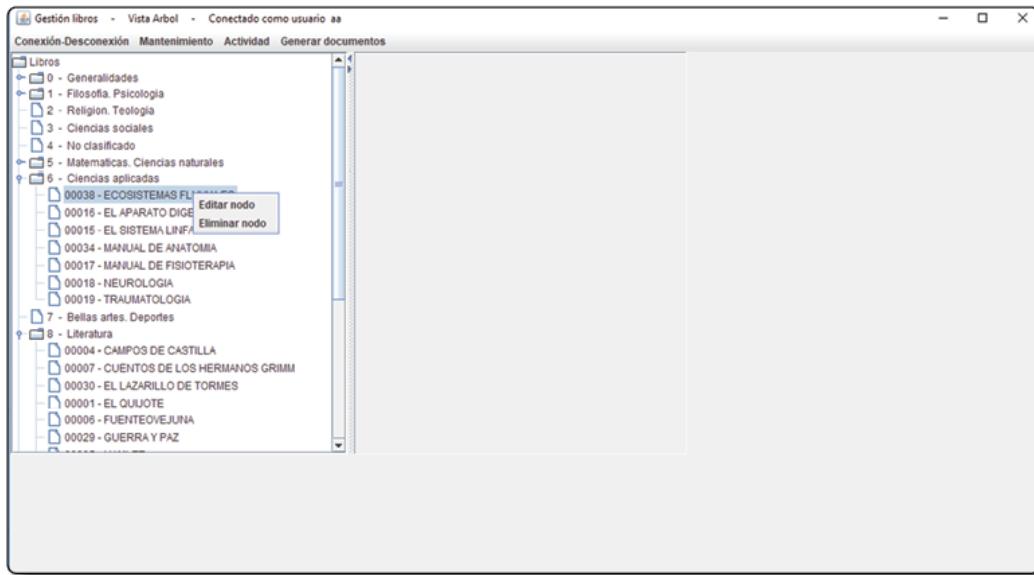
surge **EdicionNodo** un nuevo *JPanel*, “heredero” de **PantallaOpcion**, al igual que el resto de las vistas, que tendrá exactamente las mismas características y responderá al mismo comportamiento, pero con la diferencia, de que no es aglutinado por el *CardLayout*, como el resto, sino que está circunscrito a una de las áreas del *JSplitPane*, componente que comentaremos posteriormente. Este nuevo *Jpanel* **EdicionNodo** será visualizado en diferentes circunstancias, desde la que estamos exponiendo en estos momentos, nos permite solicitar información al usuario



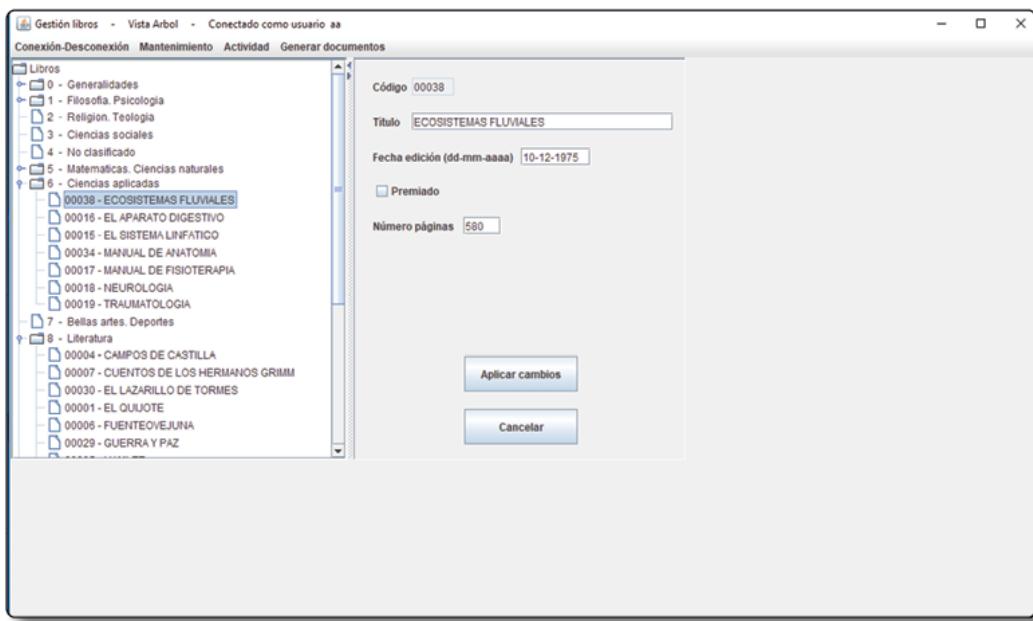
que será insertada como nueva fila en la tabla correspondiente de la Base de Datos al pulsar el botón **Aplicar cambios**. También podemos observar que el *JTextField* correspondiente al dato **Código** sigue siendo no editable, análogamente al resto de vistas. El *JPanel* que ha permitido la edición vuelve a ocultarse al pulsar el mencionado botón. Comprobamos en la siguiente pantalla que se ha creado un nuevo registro con el género correspondiente al nodo desde el que hemos iniciado la operación:



2. Si estando seleccionado un nodo de tipo **libro**, hacemos click con el botón derecho del ratón, se despliega un menú contextual:



que permite la eliminación del nodo, o la visualización de la información completa del libro asociado al nodo al hacerse visible el JPanel **EdicionNodo**:



permitiendo la modificación de los datos que se crean pertinentes, y su actualización al pulsar el botón **Aplicar cambios**. Observemos que ahora aparece el valor del campo **Código** en el *JTextField* no editable. Hemos demostrado que hemos conseguido implementar un CRUD completo en esta opción, tal y como hemos comentado al principio.

Procedemos a continuación al análisis exhaustivo del código aplicado. En primer lugar, comentar que el *JPanel EdicionNodo* responde a una réplica del ya tratado **VistaFormulario**. El aspecto más relevante es que volvemos a utilizar el mismo mecanismo que en su momento se expuso de detección de los campos que han sido actualizados mediante una instrumentalización del atributo **actualizaciones**. Abordemos a continuación las aportaciones a la aplicación de carácter novedoso:

## JSplitPane

Se trata de un componente que nos permite añadir conjuntamente dos elementos:

```
splitPane = new  
JSplitPane(JSplitPane.HORIZONTAL_SPLIT,  
jScrollPane, edicionNodo);
```

El valor del primer parámetro determina la orientación de la separación. Los otros dos parámetros, son las referencias a cada uno de los objetos que van a quedar circunscritos por el *JSplitPane*, en el caso del ejemplo:

- El *JTree* acoplado a un *JScrollPane*.
- El *JPanel EdicionNodo*.

Ambos elementos se presentan separados mediante una barra divisoria que podemos fijar en una posición de determinada:

```
splitPane.setDividerLocation(385);
```

Cuando el usuario desplaza dicha barra, un elemento gana espacio dentro del área delimitada por el *JSplitPane*, en detrimento del que ocupa el otro.

## MENÚS CONTEXTUALES

Este tipo de menús son modelados por la clase *JPopupMenu*, y las opciones siguen siendo modeladas por la clase *JMenuItem*, tal como hemos visto en los menús *JMenu*. Les aplicaremos los mismos registros de escucha y tratamiento de eventos que al resto de *JMenuItem*. Mostramos a continuación la implementación de uno de dichos menús contextuales:

```
private JPopupMenu popupMenu2;  
popupMenu2 = new JPopupMenu();  
editarNodo = new JMenuItem("Editar nodo");  
popupMenu2.add(editarNodo);
```

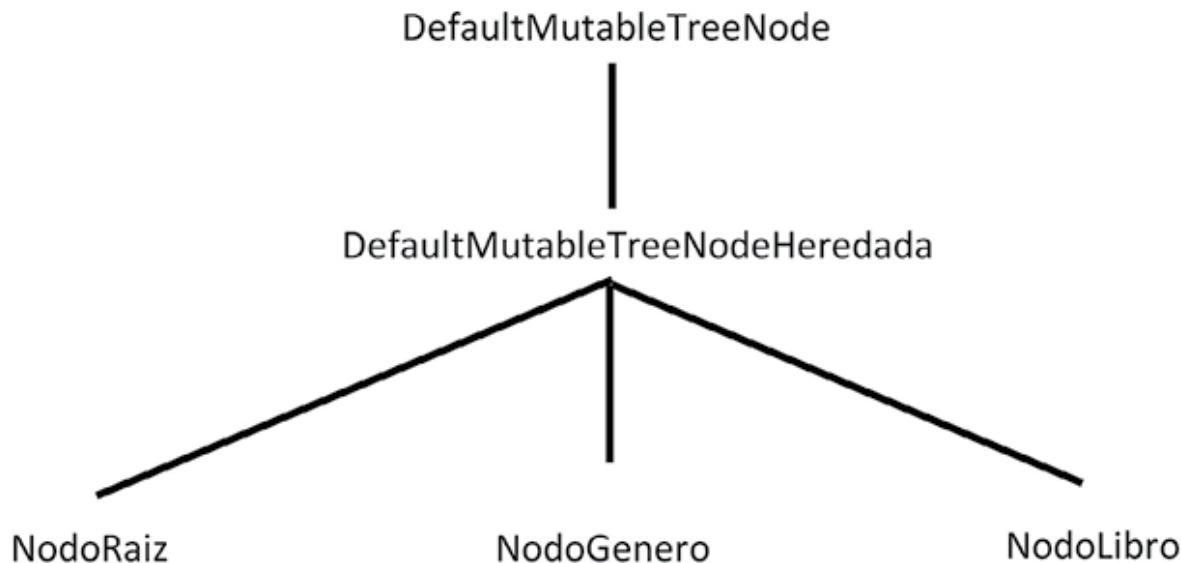
```
eliminarNodo = new JMenuItem("Eliminar nodo");
popupMenu2.add(eliminarNodo);
```

El tratamiento del evento asociado a la selección de los *JMenuItem* **Nuevo nodo** y **Editar nodo** provocará la visualización del *JPanel* **EdicionNodo**. Y la selección de Eliminar nodo, la invocación a la capa de negocio para la eliminación del registro en la Base de Datos.

## LOS NODOS

El *JTree* ya fue utilizado en la aplicación **PrediccionMeteorologicaJTree** como preludio a la gestión ampliada que vamos a aplicar en esta aplicación a dicho componente. Los aspectos afectados por la ampliación son los nodos y el tratamiento de eventos.

En la versión simplificada de *JTree* aplicada a **PrediccionMeteorologicaJTree**, los nodos eran todos instancias de la clase *DefaultMutableTreeNode*, y la información a mostrar asociada a cada nodo era transferida al mismo en el constructor. Aquí contemplamos una jerarquía polimórfica:



a partir de *DefaultMutableTreeNode*, en que los nodos instanciables (de los tipos raíz, género, y libro), van a actuar como verdaderas clases encapsuladoras.

## EL TRATAMIENTO DE EVENTOS

La aportación de los menús contextuales asociados a los nodos tipo género y tipo libro proporciona al *JTree* la interactividad necesaria para obtener la funcionalidad de un CRUD. Dichos menús contextuales se activan cuando el *JTree* correspondiente lanza un evento. Detallaremos los mecanismos utilizados en la detección del nodo concreto sobre el cual hemos pulsado el botón derecho del ratón. Dicha detección nos permitirá discernir cuál de los dos *JPopupMenu* definidos hemos de visualizar y la posición de la pantalla en que se ha de producir dicha visualización. Detallamos los pasos que se han seguido para todo este proceso:

- El *JTree* se registra con la escucha de eventos *MouseListener*:  

```
jTree.addMouseListener(controller);
```
- El tratamiento del evento se produce en el método de la clase **Controller**:

```
public void mouseClicked(MouseEvent e) {  
    pantallaOpcion.setComponenteJPanel(new  
    Integer(e.getX()), 13);  
    // Fila de la posición en que se ha pulsado botón  
    // de ratón  
    pantallaOpcion.setComponenteJPanel(new  
    Integer(e.getY()), 14);  
    // Columna de la posición en que se ha pulsado  
    // botón de ratón  
    pantallaOpcion.setComponenteJPanel(e.getComponent(
```

```

), 15);
// Componente sobre el que se hace click con el
ratón
pantallaOpcion.setComponenteJPanel(e, 16);
// MouseEvent
centralizar("ratonClicked");
}

```

Observamos que nos limitamos a registrar en componentes del array **componentes JPanel** la información necesaria (aparece en los comentarios del código) para cuando se derive finalmente el flujo del tratamiento al método **responderAController()** del *JPanel*.

- En el método **responderAController()**, concretamente en el bloque asociado al *actionCommand* “**ratonClicked**” es donde se produce la detección del nodo en concreto en que se ha producido el evento lanzado al haber pulsado el botón derecho del ratón. La lógica algorítmica de dicho bloque implica la utilización de la información registrada asociada al evento que acabamos de detallar, combinada con la invocación de los métodos:
  - `getPathForLocation()`
  - `getLastSelectedPathComponent()`
  - `getLastPathComponent()`
  - `getParent()`
  - `SwingUtilities.isRightMouseButton()`

Todos ellos reseñados y descritos en el Anexo de este libro destinado al *JTree*.

- Visualización del correspondiente *JPopupMenu* al invocar al método:

```
popupMenu1.show( (Component)componentesJPanel[15],
```

```
((Integer)componentes JPanel[13]).intValue(),  
((Integer)componentes JPanel[14]).intValue() );
```

donde el primer parámetro es la referencia a la instancia del componente invocador, en este caso el *JTree*, y el segundo y tercer parámetro son la coordenada X y la coordenada Y de la pantalla, respectivamente, en que se posicionará la visualización del componente.

## RECONSTRUCCIÓN DEL JTree

En la aplicación **PrediccionMeteorologicaJTree** nos limitábamos a construir el *JTree* ajustándonos a una configuración única del esquema de nodos, es decir, siempre el mismo número fijo de nodos y siempre en los mismos niveles. Ello fue posible utilizando únicamente el método *add()*. En el caso de esta aplicación, dada la pretensión de recrear todas las funcionalidades de un CRUD, la configuración del esquema de nodos estará en constante variación según vayan produciéndose actualizaciones de nodos tipo **libro**. Esta circunstancia nos obliga a ampliar el repertorio de métodos a utilizar en la reconstrucción del *JTree* cada vez que se produzca algún tipo de actualización, encontrándonos con la invocación a los siguientes métodos:

- *add()*
- *getChildCount()*
- *removeNodeFromParent()*
- *getChild()*
- *insertNodeInto()*
- *getChildCount*
- *removeNodeFromParent()*

- getIndexOfChild()

Todos ellos reseñados y descritos en el Anexo de este libro destinado al *JTree*, siendo invocados desde los métodos del **JPanel VistaArbol**:

- cargarArbol()
- insertarNodoLibro()
- eliminarNodoLibro()

De la observación del código implementado en los métodos citados, e incluso también en alguno de la clase **EdicionNodo**, deducirá el lector que es necesario disponer en los nodos tipo género y tipo libro, de la información asociada al género y al libro que representan, respectivamente. Esta necesidad fundamenta la vertiente de clase encapsuladora que le hemos dado a dichos tipos de nodo, encapsulando (valga la redundancia) en cada instancia, el identificativo, en ambos tipos de nodo, y en los nodos tipo libro, adicionalmente el título. Comprobamos esta circunstancia cuando, en los citados métodos, observamos:

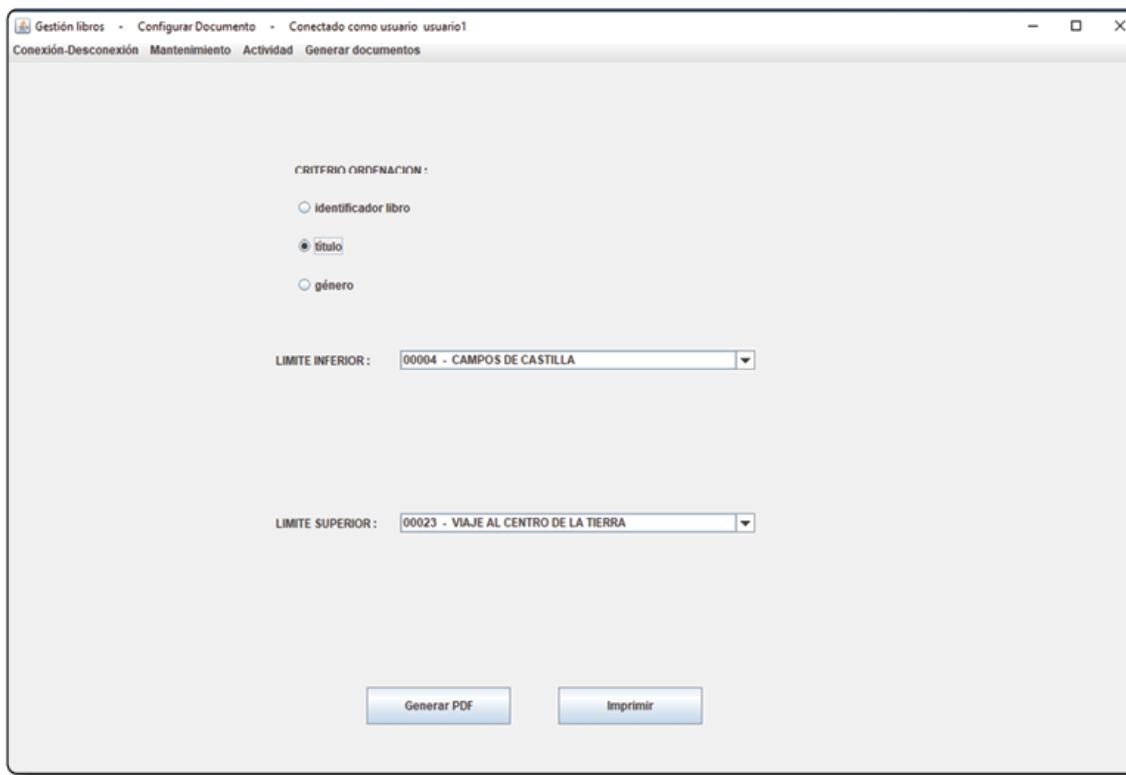
```
libro.setGenero(nodoGeneroSeleccion.getIdentificativo()
).substring(0, 1))
nodoLibroSeleccion.getIdentificativo()
((NodoLibro)modelo.getChildAt(nodoGeneroSeleccion,
i)).getTitulo()
```

Disponiendo en el correspondiente Anexo de la descripción de los métodos utilizados, así como de la justificación de la naturaleza encapsuladora de las clases que modelan los nodos, y teniendo en cuenta el carácter avanzado de este libro, no

dudamos de que el lector podrá deducir, sin ningún tipo de dificultad, la lógica algorítmica utilizada en la gestión del *JTree*.

## OPCIÓN: CONFIGURAR DOCUMENTO - IMPRESIÓN

La selección de esta opción presenta la siguiente pantalla:



La finalidad de la existencia de esta opción es la posibilidad de generar dos tipos de documentos de salida de información:

- Documento PDF
- Documento destinado a salida por impresora

Para configuración de las filas a contemplar en el documento de salida, la pantalla dispone de:

- Un grupo de *JRadioButton*. que permite al usuario determinar el criterio de ordenación de las filas. Este recurso ya se ha aplicado en otras opciones de esta aplicación cuyo estudio se ha abordado previamente.
- Dos *JComboBox* que permiten definir los límites inferior y superior de las filas a listar, respectivamente. Los valores que aparecen por defecto en cada uno de estos dos componentes responden a los valores inicial y final en función del criterio de ordenación seleccionado. Podemos observar al ejecutar la aplicación, que el cambio del criterio de ordenación repercute automáticamente en los valores que por defecto aparecen en el *JComboBox*, así como la ordenación aplicada a los ítems de dichos componentes.

Exponemos a continuación una muestra de documento generado destinado a impresora:



BIBLIOTECA MUNICIPAL DE VILLAR DEL MONTE		
Código	Título	Género
00004	CAMPOS DE CASTILLA	8 - Literatura
00007	CUENTOS DE LOS HERMANOS GRIMM	8 - Literatura
00038	ECOSISTEMAS FLUVIALES	6 - Ciencias aplicadas
00016	EL APARATO DIGESTIVO	6 - Ciencias aplicadas
00030	EL LAZARILLO DE TORMES	8 - Literatura
00001	EL QUIJOTE	8 - Literatura
00015	EL SISTEMA LINFATICO	6 - Ciencias aplicadas
00006	FUENTE OVEJUNA	8 - Literatura
00013	GENETICA	5 - Matematicas. Ciencias naturales
00029	GUERRA Y PAZ	8 - Literatura

pág. 1



## BIBLIOTECA MUNICIPAL DE VILLAR DEL MONTE

Fecha : 08-03-2020

Código	Título	Género
00005	HAMLET	8 - Literatura
00024	HISTORIA DE DOS CIUDADES	8 - Literatura
00009	HISTORIA DE ROMA	9 - Geografia. Historia
00010	JULIO CESAR	9 - Geografia. Historia
00014	LA CELULA	5 - Matematicas. Ciencias naturales
00012	LA FOTOSINTESIS	5 - Matematicas. Ciencias naturales
00025	LA ILIADA	8 - Literatura
00028	LA ODISEA	8 - Literatura
00033	LA SOMBRA DEL VIENTO	8 - Literatura
00031	LA TIA TULA	8 - Literatura

pág. 2



## BIBLIOTECA MUNICIPAL DE VILLAR DEL MONTE

Fecha : 08-03-2020

Código	Título	Género
00021	LA VUELTA AL MUNDO EN 80 DIAS	8 - Literatura
00008	LOS IBEROS	9 - Geografia. Historia
00020	LOS MISERABLES	8 - Literatura
00022	LOS TRES MOSQUETEROS	8 - Literatura
00034	MANUAL DE ANATOMIA	6 - Ciencias aplicadas
00011	MANUAL DE AUTOAYUDA	1 - Filosofia. Psicologia
00017	MANUAL DE FISIOTERAPIA	6 - Ciencias aplicadas
00026	MIGUEL STROGOFF	8 - Literatura
00018	NEUROLOGIA	6 - Ciencias aplicadas
00027	OLIVER TWIST	8 - Literatura

pág. 3



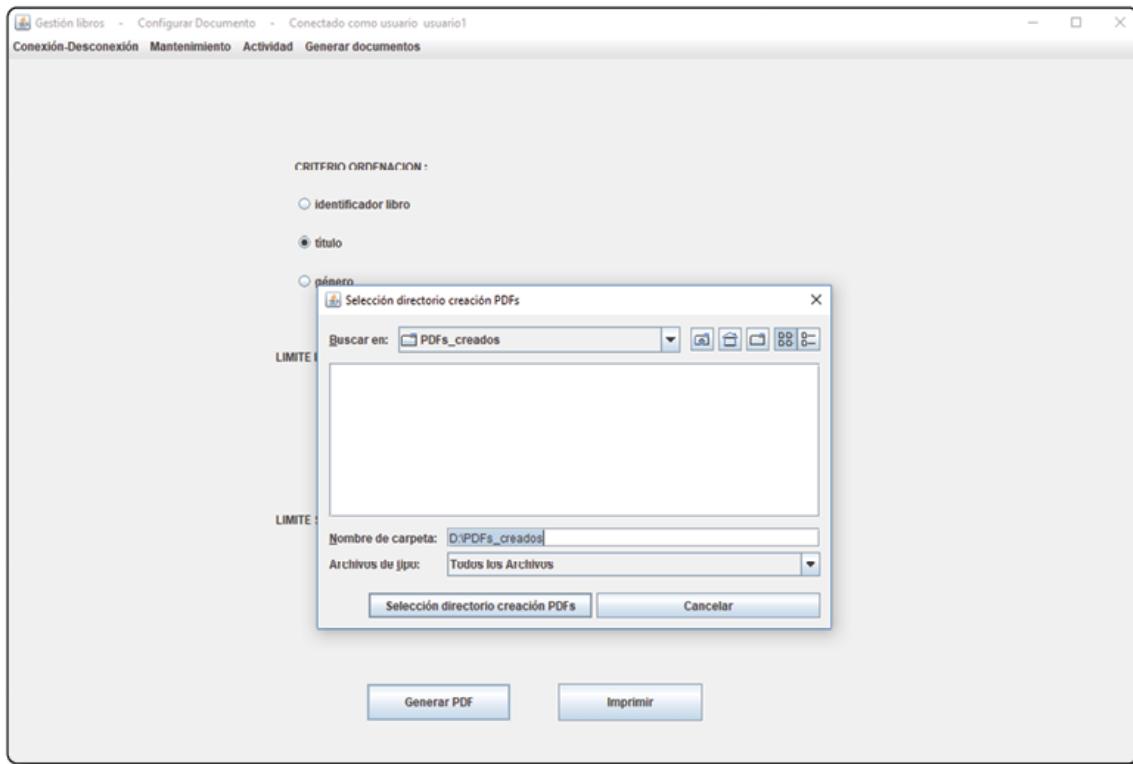
## BIBLIOTECA MUNICIPAL DE VILLAR DEL MONTE

Fecha : 08-03-2020

Código	Título	Género
00003	POESIAS COMPLETAS	8 - Literatura
00002	PROGRAMACION ORIENTADA A OBJETOS EN JAVA	0 - Generalidades
00019	TRAUMATOLOGIA	6 - Ciencias aplicadas
00023	VIAJE AL CENTRO DE LA TIERRA	8 - Literatura

pág. 4

El fichero PDF generado al pulsar el botón correspondiente es totalmente análogo al que acabamos de exponer. No lo vamos a reproducir para no incrementar innecesariamente la extensión del libro. La aplicación **VotacionPropuesta**, tratada con anterioridad, ya contemplaba la generación de ficheros PDF como documentos de salida de información. Así que en lo que concierne a esta posibilidad, solamente abordaremos en esta aplicación la aparición novedosa del componente *JFileChooser*, cuya ventana de diálogo presentamos a continuación:



La utilización que de este componente hacemos en esta aplicación está enfocada a permitir al usuario que seleccione el directorio del sistema de archivos donde se generará el fichero PDF. Reproducimos el método que recoge la implementación de esta utilidad:

```
private String seleccionDirectorioCreacionPDFs() {  
    JFileChooser jFileChooser = new  
    JFileChooser("D:\\PDFs_creados");  
    jFileChooser.setFileSelectionMode(JFileChooser.DIRECTO  
RIES_ONLY);  
    jFileChooser.showDialog(this, "Selección directorio  
creación PDFs");  
    return  
    jFileChooser.getSelectedFile().getAbsolutePath();  
}
```

Este método devuelve un *String* con la ruta del directorio seleccionado por el usuario a tal efecto.

Abordaremos a continuación el proceso de generación del documento de salida destinado a impresora, cuya implementación corre totalmente a cargo del API Swing. La lógica algorítmica aplicada a la implementación de la estructura del listado es la misma, tanto para la generación de fichero PDF, como para la del documento destinado a impresora. El número de filas por página es parametrizable externamente al código de la aplicación, viniendo definido en el fichero repositorio.xml:

```
<PARAMETROS_LISTADO>
<NUMERO_FILAS_POR_PAGINA>10</NUMERO_FILAS_POR_PAGINA>
</PARAMETROS_LISTADO>
```

Se ha definido la clase **ParametrosListado** a efectos de encapsular dicho dato. La implementación de la generación del documento destinado a impresora se distribuye entre las clases **LibrosImpresora** e **ImpresorPagina**, ubicadas en la capa **datos** por tratarse de salida utilizando tecnología específica. El proceso de impresión se produce con la invocación al método **imprimir()** de la clase **LibrosImpresora**, que recibe:

- Un *ArrayList* de objetos de la clase Libro, cada uno de los cuales encapsula la información correspondiente a cada línea del listado.
- Un objeto de la clase **ParametrosListado** citado anteriormente.

En este método:

- Instanciamos objetos de las clases:

- **PrinterJob.** Es la clase principal que corre a cargo del control del job de impresión. No se instancia mediante el operador *new* sino mediante la clase *static*:

```
PrinterJob printerJob =
PrinterJob.getPrinterJob();
```

- **Book.** Implementa la *interface Pageable*, que viene a representar la capacidad de asumir el papel de “colección” de todas las páginas a imprimir.
- **PageFormat.** Nos permite establecer la orientación del papel.
- **Paper.** Nos permite configurar tamaño de la hoja y delimitar el área de impresión de la misma.
- Para cada una de las páginas a imprimir, añadimos al Book, un objeto de la clase **ImpresorPagina**. Esta clase debe implementar la *interface Printable*, y como consecuencia de ello debe implementar el método *print()*. Al constructor de la misma le transferimos el valor del número de página y un array de objetos de la clase Libro, con tantos objetos de esta clase encapsuladora como líneas corresponda imprimir en la página:

```
book.append(new ImpresorPagina(lineasPagina,
numeroPagina++), pageFormat);
```

- Implementamos las acciones finales que determinan el lanzamiento de la impresión:

```
printerJob.setPageable(book);
if (printerJob.printDialog())
printerJob.print();
```

Abordemos ahora la clase **ImpresorPagina**. Vuelve a aparecer en el método **print()** el objeto *Graphics*, clase sobre la que hablamos cuando apareció en el ejercicio-ejemplo **InterfazGrafica6**. Podrá comprobar el lector que este objeto vuelve a representar aquí el área o contexto de dibujo o impresión, permitiéndonos utilizar los mismos mecanismos que en la citada aplicación:

- Trazado de líneas:

```
g.drawLine(50, 120, 580, 120);
```

- Escritura de cadenas:

```
g.drawString(lineasPagina[i].getTitulo(), 105,  
160+(i*15));
```

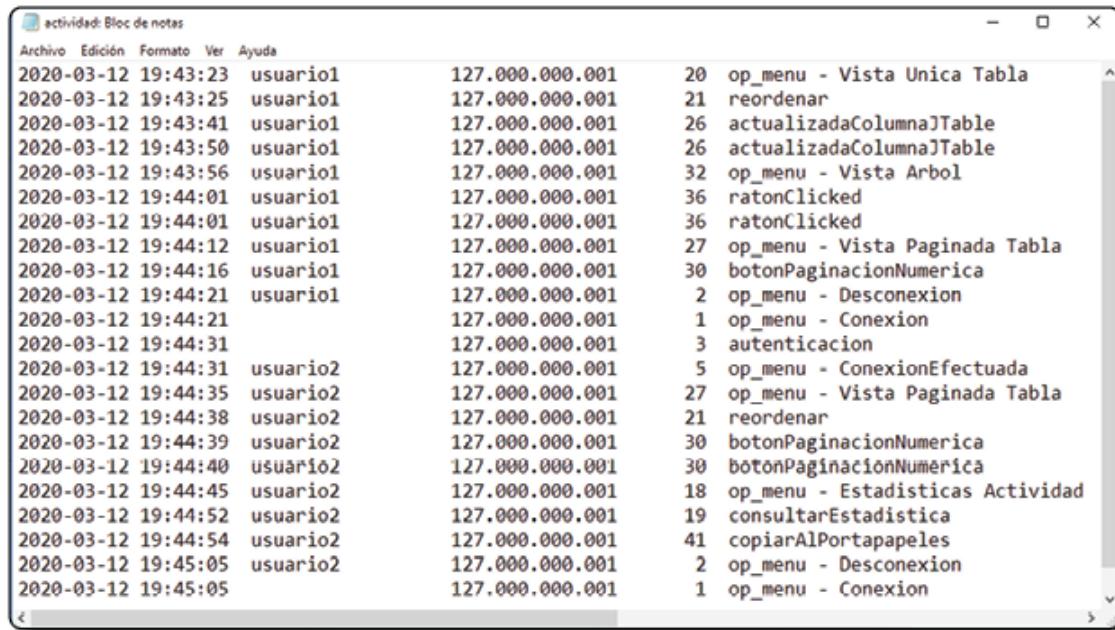
Novedosamente, en esta aplicación utilizamos la clase *Font*, cuyo cometido no necesita ningún tipo de explicación. Para la carga de la imagen, recurrimos a un objeto *MediaTracker*. Estos se utilizan para la regulación y control del estado en un proceso de carga de imágenes, audios, videos, etc.

## OPCIÓN: GESTIÓN INCIDENCIAS

La cara visible de esta utilidad está representada por los *JPanel* **VolcadoaBD** y **EstadisticasActividad**. Supone una ampliación de la gestión de las excepciones que en su momento se aplicó en **VotacionPropuesta**. La clase **GestorIncidencias**, que reemplaza el papel que representaba **GestorExcepciones**, sigue contando con la presencia del método **gestionarExcepcion()**, ejerciendo las mismas funciones, y se ve ampliada con el método **gestionarActividad()**. El cometido de este método es dejar constancia en el fichero **actividad.txt**, del directorio **log** del proyecto, el registro de la actividad realizada por el usuario en su

interacción con la aplicación, de forma análoga a como se realiza con el registro de las excepciones producidas. El citado método desencadena la invocación a las capas más profundas de la aplicación, de tal forma que es finalmente, en el método **escribirFichero()** de la clase **IncidenciasDatos**, donde se registra la correspondiente línea en el mencionado fichero. Este método es ambivalente para el registro de excepciones o de actividad, registrando una línea de incidencia en el fichero que se le pasa como tercer parámetro.

Exponemos a continuación un ejemplo del contenido del fichero actividad.txt:



Fecha	Hora	Usuario	IP	Detalles
2020-03-12	19:43:23	usuario1	127.000.000.001	20 op_menu - Vista Unica Tabla
2020-03-12	19:43:25	usuario1	127.000.000.001	21 reordenar
2020-03-12	19:43:41	usuario1	127.000.000.001	26 actualizadaColumnaJTable
2020-03-12	19:43:50	usuario1	127.000.000.001	26 actualizadaColumnaJTable
2020-03-12	19:43:56	usuario1	127.000.000.001	32 op_menu - Vista Arbol
2020-03-12	19:44:01	usuario1	127.000.000.001	36 ratonClicked
2020-03-12	19:44:01	usuario1	127.000.000.001	36 ratonClicked
2020-03-12	19:44:12	usuario1	127.000.000.001	27 op_menu - Vista Paginada Tabla
2020-03-12	19:44:16	usuario1	127.000.000.001	30 botonPaginacionNumerica
2020-03-12	19:44:21	usuario1	127.000.000.001	2 op_menu - Desconexion
2020-03-12	19:44:21		127.000.000.001	1 op_menu - Conexion
2020-03-12	19:44:31		127.000.000.001	3 autenticacion
2020-03-12	19:44:31	usuario2	127.000.000.001	5 op_menu - ConexionEfectuada
2020-03-12	19:44:35	usuario2	127.000.000.001	27 op_menu - Vista Paginada Tabla
2020-03-12	19:44:38	usuario2	127.000.000.001	21 reordenar
2020-03-12	19:44:39	usuario2	127.000.000.001	30 botonPaginacionNumerica
2020-03-12	19:44:40	usuario2	127.000.000.001	30 botónPaginaciónNuméricā
2020-03-12	19:44:45	usuario2	127.000.000.001	18 op_menu - Estadisticas Actividad
2020-03-12	19:44:52	usuario2	127.000.000.001	19 consultarEstadistica
2020-03-12	19:44:54	usuario2	127.000.000.001	41 copiarAlPortapapeles
2020-03-12	19:45:05	usuario2	127.000.000.001	2 op_menu - Desconexion
2020-03-12	19:45:05		127.000.000.001	1 op_menu - Conexion

El registro de la actividad del usuario se produce en el método **centralizar()** de la clase **Controller**, que como ya hemos comentado, supone un “un punto de paso obligado” de toda actividad que se produzca en la aplicación consecuencia de la interacción con el usuario:

new

```
GestorIncidencias().gestionarActividad(usuarioAutenticado,  
actividadesCodificadas.get(actionCommand).intValue(),  
actionCommand);
```

Se ha dispuesto la codificación de todas las actuaciones generadas por la interacción del usuario en la tabla **codificacion\_actividades** de la Base de Datos. Remitimos al lector a que revise el script de creación de tablas e inserción de datos correspondiente a esta aplicación, disponible en el material descargable. Paralelamente a contar con la persistencia de la mencionada codificación, en la clase **Controller**, se utiliza:

```
HashMap<String, Integer> actividadesCodificadas;
```

para gestionar en memoria la correspondencia entre código de actividad y descripción de la misma. Esta estructura de datos se deja disponible en el constructor de la clase **Controller**, al iniciar una cadena de invocaciones a capas las capas más profundas en:

```
actividadesCodificadas = new IncidenciasNegocio().  
consultarCodificacionActividades((BaseDatos)repositorio[0]);
```

que permite finalmente que en el método **consultarCodificacionActividades()** de la clase **ActividadUsuariosDatos** se cargue el contenido de la tabla en la estructura de datos.

Observando la descripción de las actividades, deduciremos que coincide con los **Command** utilizados en la aplicación al ser lanzado cualquier evento. Cuando se trata de la selección de una opción de menú, asociada a un *JMenuItem*, se le añade el prefijo “**op\_menu -**” a efectos de darles un tratamiento diferenciado en el método **centralizar()** de la clase **Controller**.

## VolcadoaBD

A diferencia del fichero log errores.txt de registro de las excepciones producidas, que actúa como destino final, el contenido del fichero actividad.txt tiene carácter provisional. La aplicación presenta el *JPanel VolcadoaBD*, con un solo *JButton*, que al ser pulsado por el usuario:

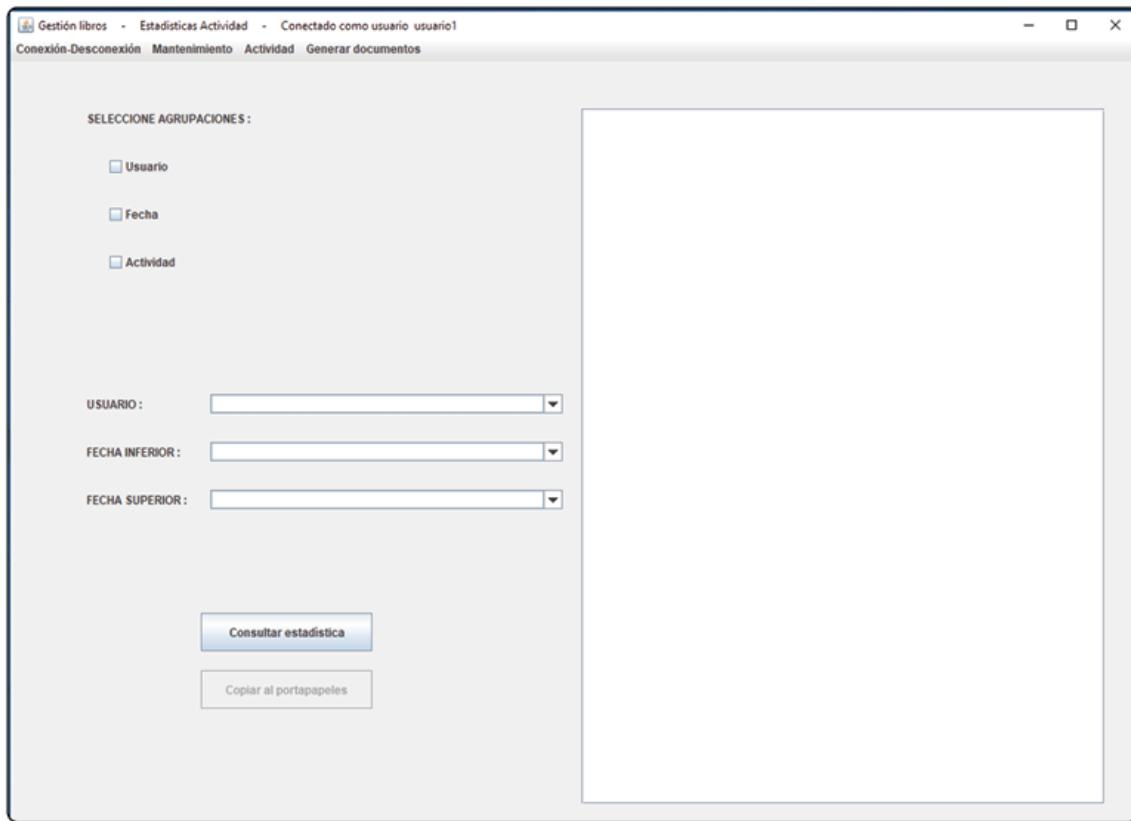
- Inserta una fila en la tabla **actividad\_usuarios** para cada una de las líneas del fichero **actividad.txt**.
- Al final del proceso de traspaso de filas fichero → tabla, se elimina el fichero, que será creado automáticamente cuando se vayan a generar nuevas líneas en el mismo.

## EstadísticasActividad

El sentido del citado traspaso está fundamentado en la posibilidad de obtener diversos tipos de estadísticas mediante consultas SQL en relación a la actividad de usuarios. A tal efecto se ha dispuesto este *JPanel*. Obviamente, puesta la aplicación en producción, esta opción, al igual que la anterior, solamente estarían disponibles a ejecución bajo el perfil de Administrador, bien en la misma aplicación, mediante el adecuado control de autenticación y conexión, o en otra aplicación diferente, solamente instalada en determinados equipos. Pensemos que, de estar distribuidas las aplicaciones en varios equipos, todos ellos accederían a una Base de Datos centralizada en el correspondiente servidor. Idénticamente ocurriría con los ficheros de **log**, que estarían ubicados en un equipo concreto dentro de la red. Antes de volver a la cuestión técnica, meditemos sobre la utilidad de aplicar estadísticas a la actividad de los usuarios, que iría desde la aplicación de controles al rendimiento del trabajo (incluso en teletrabajo), hasta la detección de transacciones fraudulentas. Para aplicar las estadísticas a esta última finalidad, tal vez sería

necesario registrar información adicional, tal como valores de claves primarias, importes (en su caso), etc. El esquema está trazado, la adaptación del mismo a cada situación, escenario o contexto concretos, lo dejamos ya de la mano del lector.

La vista ofrecida al usuario mediante el *JPanel EstadísticasActividad* es:



La vista dispone de una serie de componentes que permiten configurar la consulta SQL que obtendrá la estadística solicitada al pulsar el *JButton Consultar estadística*. La consulta SQL mencionada responde a la descripción número 15 de la relación de instrucciones SQL contempladas en esta aplicación. Dichos componentes son:

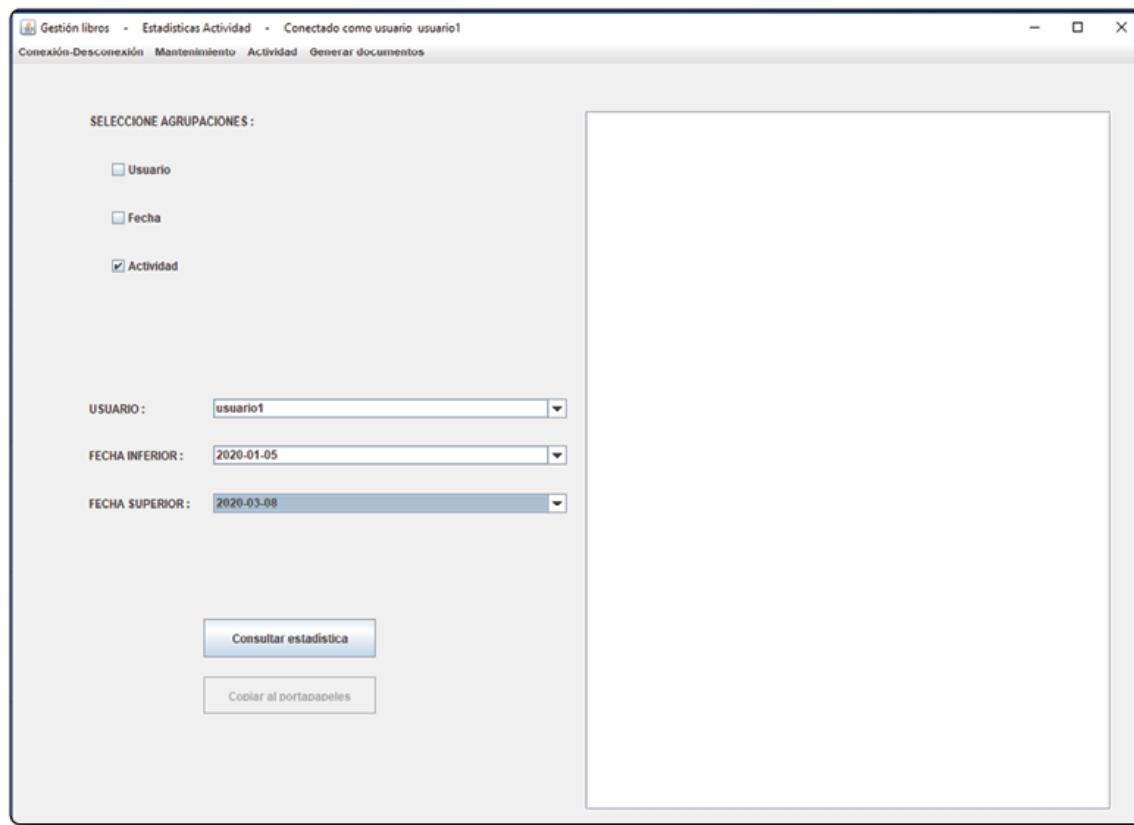
- **JCheckBox.** Se establecerán los correspondientes criterios de agrupación en la cláusula GROUP BY de la consulta SQL en función de aquellos *JCheckBox* que se encuentren seleccionados.
- **JComboBox jComboBoxSeleccionUsuario.** Por defecto aparece seleccionado ítem en blanco. Esta circunstancia determina que la estadística se aplicará a todos los usuarios en que se haya registrado actividad. Si se despliega el componente, aparecerá la relación de usuarios con actividad. De seleccionar uno de ellos, la estadística se aplicará solamente a ese usuario.
- **JComboBox de selección de fecha inferior y fecha superior.** La selección de dichas fechas permite acotar el período de tiempo sobre el que se aplicará la estadística. Idénticamente al caso anterior, por defecto aparece seleccionado ítem en blanco. De quedar seleccionado dicho ítem, no se aplica la correspondiente condición en la cláusula WHERE de la instrucción SQL. Si desplegamos dichos *JComboBox* aparecerá la relación de fechas en que se ha registrado actividad. De seleccionar una fecha en un *JComboBox*, en el otro, o en los dos, determinará que se aplique el valor de fecha seleccionado en la correspondiente condición de la cláusula WHERE.

En la descripción de la SQL podrá observar el lector diversas concreciones de la misma condicionadas por la interacción del usuario con los componentes que acabamos de detallar.

Observamos en la pantalla mostrada que el *JButton Copiar al portapapeles* aparece desactivado. Ello es debido a que todavía no se ha solicitado ninguna estadística, en consecuencia, el

*JTextArea* donde se mostrará el resultado de la consulta aparece vacío, lo cual hace que no haya nada que copiar.

Mostramos a continuación un ejemplo de configuración de estadística:

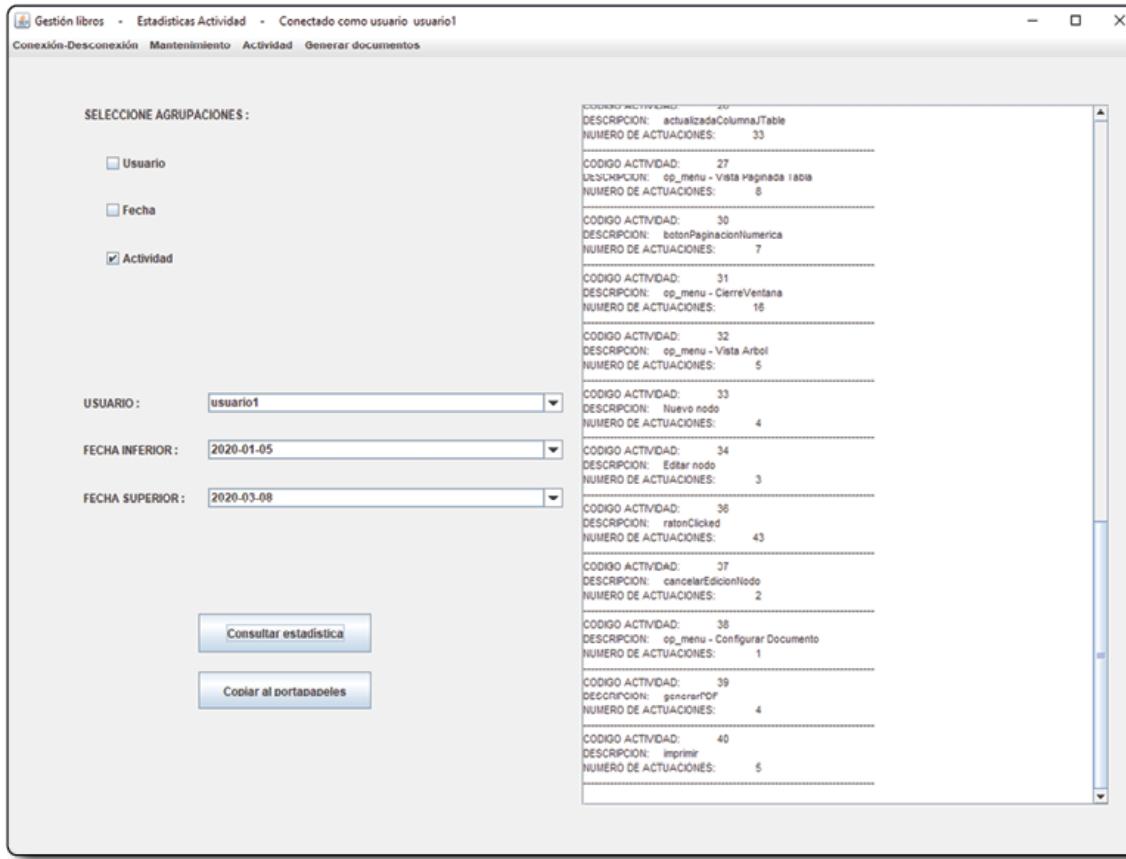


Al pulsar el *JButton Consultar estadística* se lanza a la Base de Datos la consulta que responde a dicha configuración:

```
SELECT
codificacion_actividades.codigo,descripcion,COUNT(*)
FROM codificacion_actividades INNER JOIN
actividad_usuarios
ON codificacion_actividades.codigo =
actividad_usuarios.codigo
WHERE id_usuario = CAST(? AS CHAR(15))
AND DATE_FORMAT(fecha_hora, '%Y-%m-%d') >= ?
AND DATE_FORMAT(fecha_hora, '%Y-%m-%d') <= ?
```

```
GROUP BY codificacion_actividades.codigo,descripcion  
ORDER BY codificacion_actividades.codigo
```

cuyo resultado se muestra en el *JTextArea*:



Observamos que nos encontramos el *JTextArea* con el resultado de la consulta, y el *JButton* **Copiar al portapapeles** activado, de tal modo que, si se pulsa, todo el contenido del componente de salida (el *JTextArea*, establecido a no editable) se copia en el portapapeles del sistema, permitiendo ser pegado genéricamente a cualquier otra aplicación. Para que la aplicación pueda asumir la gestión del portapapeles que acabamos de describir son necesarias las siguientes actuaciones:

1. Obtener referencia a objeto que permite acceso al portapapeles del sistema:

```
private Clipboard clipboard;  
clipboard = getToolkit().getSystemClipboard();
```

2. Establecer la clase, en esta aplicación, **Controller**, que actuará como receptora del evento de transferencia de contenido al portapapeles del sistema

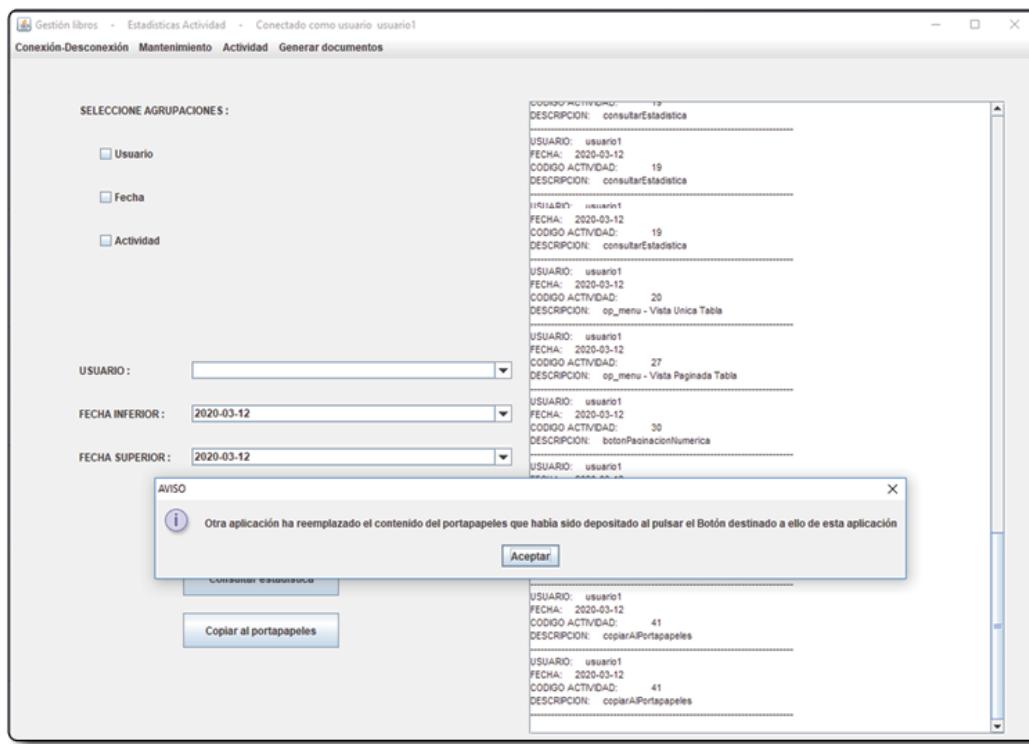
- Implementando la correspondiente *interface*:

```
implements ClipboardOwner
```

- Método de implementación obligatoria por la interface implementada:

```
public void lostOwnership(Clipboard clipb,  
Transferable transferable) {  
JOptionPane.showMessageDialog(null, "Otra  
aplicación ha reemplazado el contenido del  
portapapeles que había sido depositado al pulsar  
el Botón destinado a ello de esta aplicación",  
"AVISO", JOptionPane.INFORMATION_MESSAGE);  
}
```

Este método se ejecuta cuando el contenido que al portapapeles ha sido transferido por nuestra aplicación es reemplazado por una transferencia de contenido no originada por la misma, sino por cualquiera de los procedimientos de carácter genérico que aporta el Sistema Operativo para ello. Mostramos la ventana abierta originada por la ejecución del método:



### 3. Traspaso al portapapeles del contenido pretendido:

```
clipboard.setContents(new
StringSelection(jTextArea.getText()), controller);
```

En este caso efectuamos dicho traspaso desde el método **responderAController()** de la clase **EstadisticasActividad**. Observemos que el primer parámetro es el contenido a transferir, y el segundo la clase que se ha registrado como escucha del correspondiente evento.

## Anexo 1

### JDBC

#### A1.1 INTERFACES JDBC Y OTROS ASPECTOS CONTEMPLADOS EN LAS APLICACIONES

Componente	Ejercicio JDBC1	Ejercicio JDBC2	Ejercicio JDBC3	Ejercicio JDBC4	Ejercicio JDBC5
Arquitectura a tres capas					
Pool conexiones					
Connection	X	X	X	X	
Statement	X	X			
PreparedStatement			X	X	
CallableStatement					
ResultSet	X	X	X		
DatabaseMetaData					
ResultSetMetaData					

## Anexo 2

# SWING

### A2. 1 MÉTODOS DE CLASES SWING

- ***add*** (componente). Añade el componente al contenedor.
- ***addElement*** (String). Añade un elemento al componente.
- ***addImage*** (Image, int prioridad). Añade la imagen al objeto de la clase *MediaTracker*.
- ***addItem*** (Object). Añade un Item al componente.
- ***add<TipoEvento>Listener*** (oyente de eventos). Registra el componente con el objeto oyente de eventos.
- ***append*** (String). Añade contenido al componente.
- ***dispose()***. Libera los recursos asignados al objeto.
- ***drawImage*** (Imagen, int x, int y, ImageObserver). Dibuja la imagen en las coordenadas especificadas. El cuarto argumento es la referencia a un objeto que implemente la interfaz *ImageObserver*; normalmente es el objeto en el que la imagen se exhibe.
- ***drawLine*** (int x1, int y1, int x2, int y2). Dibuja linea con las coordenadas especificadas.
- ***drawOval*** (int x, int y, int anchura, int altura). Dibuja un óvalo descrito por el rectángulo determinado por los

parámetros, de forma análoga a *drawRect()*.

- ***drawRect*** (int x, int y, int anchura, int altura). Dibuja un rectángulo descrito por los cuatro parámetros, siendo *x*, *y* las coordenadas de la esquina superior izquierda.
- ***drawRoundRect*** (int x, int y, int anchura, int altura, int anchuraDeArco, int alturaDeArco). Dibuja un rectángulo con esquinas redondeadas. Los dos primeros argumentos especifican las coordenadas de la esquina superior izquierda del rectángulo si éste no estuviese redondeado.
- ***drawString*** (String, int x, int y). Dibuja un *String* en las coordenadas especificadas.
- ***end ()***. Comienza a imprimir el contexto gráfico.
- ***getActionCommand ()***. Devuelve un *String* con el identificador del componente cuando éste es enviado asociado a un evento por un *listener*.
- ***getColumn ()***. Devuelve la columna donde se ha producido el cambio que ha generado el evento.
- ***getContentPane ()***. Devuelve el objeto *contentPane* para el frame.
- ***getContents (Object)***. Devuelve un *Transferable* con el actual contenido del portapapeles. Si el portapapeles está vacío devolverá *null*.
- ***getFirstRow ()***. Devuelve la primera fila donde se ha producido el cambio que ha generado el evento.
- ***getGraphics()***. Devuelve el objeto gráfico sobre el que se debe pintar lo que se desea imprimir.
- ***getImage (String)***. Pone en marcha un hilo de ejecución independiente que carga la imagen (o la descarga de

internet) desde la “ruta” especificada en el String.

- **getItemCount()**. Devuelve un *int* con el número de Items del componente.
- **getItemSelectable ()**. Devuelve la referencia al objeto que ha generado el evento.
- **getMinSelectionIndex ()**. Devuelve el primer índice de la selección o -1 si la selección está vacía.
- **getPassword ()**. Devuelve un *char[]* correspondiente al contenido del componente. Para tratar dicho contenido como *String*, es necesario instanciar un *String* utilizando el constructor *public String(char [])*.
- **getPrintJob (contenedor,String,Properties)**. Obtiene una instancia de la clase abstracta *PrintJob*.
- **getSelectedFile ()**. Devuelve el directorio o archivo seleccionado.
- **getSelectedIndex()**. Devuelve un *int* el número de orden del Item seleccionado en el componente.
- **getSelectedItem()**. Devuelve el *Object* que supone el Item seleccionado en el componente.
- **getSelectedValue()**. Devuelve el *Object* que supone el elemento seleccionado en el componente.
- **getSize ()**. Devuelve un *int* con el número de elementos del componente.
- **getSource ()**. Devuelve la referencia al objeto que ha generado el evento.
- **getSystemClipboard ()**. Devuelve el portapapeles.

- ***getText()***. Devuelve el *String* introducido en el componente.
- ***getToolkit ()***. devuelve el toolkit del frame.
- ***getValue ()***. Devuelve un *int* con el valor que en ese momento viene determinado por la posición del apuntador de la *Slider*.
- ***getX ()***. Devuelve coordenada “x” de la posición del ratón.
- ***getY ()***. Devuelve coordenada “y” de la posición del ratón.
- ***isSelected ()***. Devuelve un *boolean* indicando si el componente está seleccionado o no.
- ***isSelectionEmpty ()***. Devuelve un valor *boolean* indicando con *false* que hay algún elemento seleccionado en el componente y con *true* que no lo hay.
- ***paint (Graphics)***. Dibuja el componente.
- ***remove (int)***. Elimina el elemento correspondiente al número de orden determinado por el parámetro.
- ***removeAllItems()***. Elimina todos los Items del componente.
- ***removeAllElements ()***. Elimina todos los elementos del componente.
- ***removeItemAt(int)***. Elimina el Item correspondiente al número de orden determinado por el parámetro.
- ***removeSelectionInterval (índiceInicial, índiceFinal)***. Elimina el intervalo de elementos seleccionados del componente entre índiceInicial e índiceFinal.
- ***remove<TipoEvento>Listener* (oyente de eventos)**. Elimina el componente del objeto oyente de eventos.
- ***repaint ()***. Redibuja el componente.

- ***setActionCommand*** (*String*). Establece el identificador al componente cuando éste sea enviado asociado a un evento por un *listener*.
- ***setBackground*** (*color* ). Establece el color de fondo del componente.
- ***setBounds*** (*x,y,ancho,alto*). Establece posición y dimensiones del componente.
- ***setContents*** (*Transferable,ClipboardOwner*). Establece el nuevo contenido del portapapeles, indicando el nuevo dueño de dicho contenido.
- ***setEchoChar*** (*char*). Establece el *char* correspondiente al “echo” que se produzca al editar el componente.
- ***setEditable*** (*boolean*). Permite o no permite que el componente sea editable.
- ***setEnabled*** (*boolean*). Activa o desactiva el componente.
- ***setFont*** (*fuente*). Establece la fuente para el texto del componente.
- ***setLayout*** (*administrador de disposición*). Establece el administrador de disposición o diseño.
- ***setLocation*** (*int x,int y*). Establece las coordenadas de la esquina superior izquierda del componente para su posicionamiento. Se puede aplicar a contenedores como el JFrame.
- ***setMajorTickSpacing*** (*int*). Establece el número de valores entre las marcas mayores.
- ***setMaximum***(*int*). Establece el valor máximo para la *JProgressBar*.

- ***setMinimum(int)***. Establece el valor mínimo para la *JProgressBar*.
- ***setMinorTickSpacing (int)***. Establece el número de valores entre las marcas menores.
- ***setPaintTicks (boolean)***. Establece que las marcas sean pintadas o no.
- ***setSelected (boolean)***. Establece que el componente quede seleccionado o no seleccionado.
- ***setSelectedValue (Object,boolean)***. Establece como seleccionado el elemento del componente correspondiente al *Object* que se pasa como argumento.
- ***setSelectedIndex(int)***. Establece como seleccionado el Item del componente determinado por el número de orden que supone *int*.
- ***setSelectedItem (Object)***. Establece como seleccionado el Item del componente correspondiente al *Object* que se pasa como argumento.
- ***setSize (ancho,alto)***. Establece el tamaño del contenedor.
- ***setText (String)***. Establece el texto al componente.
- ***setTitle (String)***. Establece título al contenedor.
- ***setValue (int)***. Establece el valor que marcará la *JProgressBar*.
- ***setVisible (boolean)***. Visualiza u oculta el contenedor.
- ***show (contenedor, String)***. Muestra el panel especificado por el segundo parámetro, que supone el nombre que se utilizó para ser añadido al *CardLayout*.

- ***showDialog(Object, String)***. Muestra una ventana modal con el selector de archivos. El primer parámetro es la referencia a un contenedor; y el segundo es el título de dicha ventana modal.
- ***showMessageDialog (componentePadre, mensaje, titulo, tipoDeMensaje)***. Muestra un cuadro de diálogo modal que presenta información al usuario y tiene un botón de aceptación.
- ***waitForAll()***. Espera a que todos los medios estén cargados.

## A2.2 JTABLE

### Constructores de JTable

- ***JTable (array de objetos de 2 dimensiones , array de Strings de 1 dimensión con los nombres de las columnas)***
- ***JTable (clase que herede de AbstractTableModel)***

### Métodos de AbstractTableModel

```
public int getColumnCount()
{
    return( datos[0].length );
}
```

Número de columnas de la tabla.

```
public int getRowCount()
{
    return( datos.length );
}
```

Número de filas de la tabla.

```
public Object getValueAt(int fila, int columna)
{
return datos[fila][columna];
}
```

Contenido de una celda de la tabla.

```
public void setValueAt(Object valor, int fila, int
col)
{
copiaDato = datos[fila][col];
datos[fila][col] = valor;
fireTableCellUpdated(fila,col);
}
```

Es necesario implementar este método si la tabla va a ser editable; y en él especificamos las acciones que queremos que se produzcan cuando se edita cualquier celda de la tabla. Permite que se escuchen los eventos *TableModelEvent*, claro está, si el Modelo de Datos se ha registrado con una clase de escucha que implemente *TableModelListener*. Para que en el método *tableChanged(TableModelEvent evt)* de la clase escucha de eventos podamos detectar qué tipo de cambios han sido realizados, debemos incluir en este método la variante que nos interese de *fireTable<acción>()* (son métodos, obviamente, de la clase *AbstractTableModel*).

```
public boolean isCellEditable(int fila,int col)
{
if (col==0)
return( false );
else
return( true );
}
```

Permite especificar si las celdas de la tabla van a ser editables. Utilizando estructuras alternativas, como en este caso, podemos

concretar que columnas van a ser editables y cuáles no.

```
public String getColumnNome(int col)
{
    return nombreColumnas[col];
}
```

Nombre de las columnas.

```
public Class getColumnClass(int col)
{
    return getValueAt(0,col).getClass();
}
```

Permite que se tome el editor más apropiado al tipo de dato asociado a la columna; por ejemplo un *JCheckBox* para un *Boolean*.

## Métodos de ListSelectionEvent

- *getValueIsAdjusting()*. Con respecto a la selección realizada, devuelve *true* cuando el botón izquierdo del ratón está pulsado, y *false* al soltarlo.

## A2.3 JTREE

### Métodos de JTree

- *getLastSelectedPathComponent()*. Devuelve un *Object*, correspondiente a la referencia al último nodo del path de la actual selección en el *JTree*.
- *getPathForLocation(int x, int y)*. Devuelve el *path* del nodo especificado por las coordenadas x, y;

### Métodos de TreePath

- ***getLastPathComponent()***. Devuelve un *Object*, correspondiente a la referencia al último nodo del *path* con respecto al cual se ejecuta el método.

## Métodos de DefaultTreeModel

- ***getChild(Object, int)***. Devuelve un *Object*, correspondiente a la referencia al nodo hijo que ocupa la posición especificada por el segundo parámetro, con respecto al conjunto de hijos del nodo que se pasa como primer parámetro.
- ***getChildCount(Object)***. Devuelve un *int* correspondiente al número de hijos del nodo que se pasa como parámetro.
- ***getIndexofChild(NODOPADRE , NODOHIJO)***. Devuelve la posición que ocupa el nodo hijo dentro del conjunto de nodos del nodo padre.
- ***insertNodeInto( nodoGenero, raiz, modelo.getChildCount(raiz) )***. Añade como nodo hijo del nodo referenciado por el segundo parámetro, el nodo referenciado por el primer parámetro, situándolo en la posición especificada por el tercer parámetro con respecto al conjunto de nodos hijos.
- ***reload()***. Invocamos a este método cuando hemos modificado el esquema de nodos del árbol. El Model notificará a todos los Listeners que se han producido cambios.
- ***removeNodeFromParent(MutableTreeNode)***. Elimina el nodo referenciado por el primer parámetro de la posición que ocupa en el árbol.

## Métodos de DefaultMutableTreeNode

- ***add***(MutableTreeNode). Añade el nodo referenciado por el parámetro como nodo hijo del nodo con respecto al que se ejecuta el método.
- ***getParent()***. Devuelve el nodo padre con respecto al que se ejecuta el método.
- ***removeAllChildren()***. Elimina todos los nodos hijos del nodo actual.
- ***setUserObject(String)***. Establece el texto que acompaña la visualización del nodo.

## A2.4 OTRAS CLASES

### Métodos de SwingUtilities

- ***isRightMouseButton*** (MouseEvent). Devuelve *true* si el evento transferido como parámetro se ha lanzado como consecuencia de haber pulsado el botón derecho del ratón.

### Métodos de PrinterJob

- ***print()***. Imprime el conjunto de páginas que han sido asignadas al *PrinterJob* en un lote mediante el *Book*.
- ***printDialog()***. Visualiza la ventana de diálogo para la impresión del trabajo.
- ***setPageable*** (*Book*). Establece al *PrinterJob* el *Book* con el conjunto de objetos que implementan *Printable* a imprimir.

## A2.5 EVENTOS

### A2.5.1 Interfaces oyentes y métodos

Componente	GestionLibros	VotacionPropuesta	PrediccionMeteorologicaJTree	PoolConexiones	EjercicioJDBC9	EjercicioJDBC8	EjercicioJDBC7	EjercicioJDBC6	EjercicioJDBC5	EjercicioJDBC4	EjercicioJDBC3	EjercicioJDBC2	EjercicioJDBC1
Arquitectura a tres capas	X	X	X	X	X	X	X	X	X	X	X	X	X
Pool conexiones											X		
Connection	X	X	X	X	X	X	X	X	X	X	X	X	X
Statement	X	X			X			X	X	X	X	X	X
PreparedStatement			X	X						X			X
CallableStatement						X	X						
ResultSet	X	X	X					X	X				X
DatabaseMetaData									X				
ResultSetMetaData									X				

### A2.5.2 Componentes Swing y eventos que pueden lanzar

Componente Swing	Eventos que puede generar						Table Model
	Action	Item	Change	Caret	List Selection	Mouse	
AbstractTableModel							X
JButton	X	X	X				
JCheckbox	X	X	X				
JComboBox	X	X	X				
JFileChooser	X						
JList						X	
JMenuItem	X		X				
JPasswordField	X				X		
JRadioButton	X	X	X				
JSlider			X				
JTabbedPane			X				
JTextArea					X		
JTextField	X				X		
JToggleButton	X	X	X				
JTree							X
ListSelectionModel						X	
SpinnerDateModel			X				
SpinnerListModel			X				
SpinnerNumberModel			X				

## A2.5.3 Clases Swing y AWT utilizadas en las aplicaciones

ButtonGroup		X			X
Canvas		X X			
Clipboard					X
Color					X
Component					X
DefaultCellEditor					X
DefaultListModel			X		
DefaultMutableTreeNode				X X	
DefaultTableCellRenderer					X
DefaultTreeModel				X X	
Font					X
Graphics		X X			X
Image					X
JButton	X X X X X X	X X X X X X	X X X X X X	X X X X X X	X X
JCheckbox		X			X
JComboBox		X			X X X
JDialog			X		
JFileChooser					X
JLabel	X X X X X X X		X X X X X X	X X X X X X	
JList			X		X
JMenu				X	X
JMenuBar				X	X
JMenuItem				X	X
JOptionPane					X X
JPasswordField					X
JPopupMenu					X
JProgressBar			X		
JRadioButton		X			X
JScrollPane		X		X X	X X
JSeparator				X	
JSlider		X			X
JSpinner			X		
JSplitPane					X
JTabbedPane				X	
JTable					X X
JTextArea					X
JTextField		X X			X X
JToggleButton		X			
JTree				X	X
ListSelectionModel					X
MediaTracker					X
PageFormat					X
Paper					X
Printable					X
PrinterJob					X
SpinnerDateModel			X		
SpinnerListModel			X		
SpinnerNumberModel			X		
StringSelection					X
SwingUtilities					X
TableColumn				X X	
TitledBorder					X
Toolkit					X
TreePath					X

## A2.5.4 Listeners registrados en las aplicaciones



# Anexo 3

## SCRIPTS SQL

### MySQL - BIBLIOTECA

```
DROP DATABASE IF EXISTS `biblioteca`;
CREATE DATABASE IF NOT EXISTS `biblioteca` /*!40100
DEFAULT CHARACTER SET utf8 */;
USE `biblioteca`;
DROP TABLE IF EXISTS libros;
DROP TABLE IF EXISTS generos;
DROP TABLE IF EXISTS actividad_usuarios;
DROP TABLE IF EXISTS codificacion_actividades;
DROP TABLE IF EXISTS usuarios_biblioteca;
CREATE TABLE usuarios_biblioteca
(
id_usuario CHAR(15),
password VARCHAR(12),
CONSTRAINT pk_usuariosbiblioteca PRIMARY KEY
(id_usuario)
);
CREATE TABLE codificacion_actividades
(
codigo INT,
descripcion VARCHAR(50),
CONSTRAINT pk_codificacionactividades PRIMARY KEY
(codigo)
);
CREATE TABLE actividad_usuarios
(
fecha_hora DATETIME,
```

```
id_usuario CHAR(15),
ip_cliente VARCHAR(18),
codigo INT,
CONSTRAINT fk_actividad_codificacion FOREIGN KEY
(codigo) REFERENCES codificacion_actividades (codigo)
);
CREATE TABLE generos
(
codigo CHAR(1),
descripcion VARCHAR(32),
CONSTRAINT pk_generos PRIMARY KEY (codigo)
);
CREATE TABLE libros
(
id_libro CHAR(5),
titulo VARCHAR(60),
genero CHAR(1),
fecha_edicion DATETIME,
numero_paginas INT,
premiado INT,
CONSTRAINT pk_libros PRIMARY KEY (id_libro),
CONSTRAINT fk_libros_generos FOREIGN KEY (genero)
REFERENCES generos (codigo)
);
-----
-----
-----
-- MEDIANTE LA SIGUIENTE TABLA, PROCEDIMIENTOS Y
-- FUNCION SIMULAMOS EN MySQL LAS SECUENCIAS DE ORACLE
-----
-----
-----
DROP TABLE IF EXISTS secuencias;
DROP PROCEDURE IF EXISTS crea_secuencia;
DROP PROCEDURE IF EXISTS elimina_secuencia;
DROP PROCEDURE IF EXISTS secuencia_set_valor;
DROP PROCEDURE IF EXISTS secuencia_set_incremendo;
```

```
DROP FUNCTION IF EXISTS secuencia_next_valor;
CREATE TABLE secuencias
(
secuencia_nombre VARCHAR(35) NOT NULL PRIMARY KEY,
secuencia_valor INT UNSIGNED NOT NULL,
secuencia_incremto INT UNSIGNED NOT NULL
);
DELIMITER //
CREATE PROCEDURE crea_secuencia(secuenciaNombre
VARCHAR(35), valorIncialSecuencia INT UNSIGNED,
incremento INT UNSIGNED)
BEGIN
IF (SELECT COUNT(*) FROM secuencias WHERE
secuencia_nombre = secuenciaNombre) = 0 THEN
INSERT INTO secuencias (secuencia_nombre,
secuencia_valor, secuencia_incremto) VALUES
(secuenciaNombre, valorIncialSecuencia, incremento);
ELSE
SIGNAL SQLSTATE '50000' SET MESSAGE_TEXT = "No puede
crear la secuencia especificada porque ya existe";
END IF;
END//
CREATE PROCEDURE elimina_secuencia(secuenciaNombre
VARCHAR(35))
BEGIN
IF (SELECT COUNT(*) FROM secuencias WHERE
secuencia_nombre = secuenciaNombre) > 0 THEN
DELETE FROM secuencias WHERE secuencia_nombre =
secuenciaNombre;
ELSE
SIGNAL SQLSTATE '50000' SET MESSAGE_TEXT = "La
secuencia especificada no existe";
END IF;
END//
CREATE PROCEDURE secuencia_set_valor(secuenciaNombre
VARCHAR(35), secuenciaValor INT UNSIGNED)
BEGIN
```

```

IF (SELECT COUNT(*) FROM secuencias WHERE
secuencia_nombre = secuenciaNombre) > 0 THEN
UPDATE secuencias SET secuencia_valor = secuenciaValor
WHERE secuencia_nombre = secuenciaNombre;
ELSE
SIGNAL SQLSTATE '50000' SET MESSAGE_TEXT = "No existe
la secuencia";
END IF;
END//
```

CREATE PROCEDURE

```

secuencia_set_incremto(secuenciaNombre VARCHAR(35),
incremento INT UNSIGNED)
BEGIN
IF (SELECT COUNT(*) FROM secuencias WHERE
secuencia_nombre = secuenciaNombre) > 0 THEN
UPDATE secuencias SET secuencia_incremto =
incremento WHERE secuencia_nombre = secuenciaNombre;
ELSE
SIGNAL SQLSTATE '50000' SET MESSAGE_TEXT = "No existe
la secuencia";
END IF;
END//
```

CREATE FUNCTION secuencia\_next\_valor(secuenciaNombre

```

VARCHAR(35)) RETURNS INT UNSIGNED
BEGIN
DECLARE valorActual INT;
SET valorActual = (SELECT secuencia_valor FROM
secuencias WHERE secuencia_nombre = secuenciaNombre);
IF valorActual IS NOT NULL THEN
UPDATE secuencias SET secuencia_valor = valorActual +
secuencia_incremto WHERE secuencia_nombre =
secuenciaNombre;
ELSE
SIGNAL SQLSTATE '50000' SET MESSAGE_TEXT = "No existe
la secuencia";
END IF;
RETURN valorActual;
```

```
END//  
DELIMITER ;  
--  
--  
-- CALL elimina_secuencia("secuencia_libros");  
CALL crea_secuencia("secuencia_libros", 1, 1);  
-- CALL secuencia_set_valor("secuencia_libros", 100);  
-- CALL secuencia_set_increemento("secuencia_libros",  
5);  
-- SELECT secuencia_next_valor("secuencia_libros");  
--  
--  
--  
--  
--  
INSERT INTO usuarios_biblioteca VALUES  
(‘usuario1’,’password1’);  
INSERT INTO usuarios_biblioteca VALUES  
(‘usuario2’,’password2’);  
INSERT INTO usuarios_biblioteca VALUES  
(‘usuario3’,’password3’);  
INSERT INTO codificacion_actividades VALUES  
(1,’op_menu - Conexion’);  
INSERT INTO codificacion_actividades VALUES  
(2,’op_menu - Desconexion’);  
INSERT INTO codificacion_actividades VALUES  
(3,’autenticacion’);  
INSERT INTO codificacion_actividades VALUES  
(4,’desconectar’);  
INSERT INTO codificacion_actividades VALUES  
(5,’op_menu - ConexionEfectuada’);  
INSERT INTO codificacion_actividades VALUES  
(6,’op_menu - Volcado a BD’);  
INSERT INTO codificacion_actividades VALUES  
(7,’volcarIncidencias’);  
INSERT INTO codificacion_actividades VALUES  
(8,’op_menu - Vista Formulario’);  
INSERT INTO codificacion_actividades VALUES
```

```
(9,'comboLibros');
INSERT INTO codificacion_actividades VALUES
(10,'nuevoLibro');
INSERT INTO codificacion_actividades VALUES
(11,'aplicarCambios');
INSERT INTO codificacion_actividades VALUES
(12,'eliminarLibro');
INSERT INTO codificacion_actividades VALUES
(13,'actualizadoTitulo');
INSERT INTO codificacion_actividades VALUES
(14,'actualizadoGenero');
INSERT INTO codificacion_actividades VALUES
(15,'actualizadoFechaEdicion');
INSERT INTO codificacion_actividades VALUES
(16,'actualizadoNumeroPaginas');
INSERT INTO codificacion_actividades VALUES
(17,'actualizadoPremiado');
INSERT INTO codificacion_actividades VALUES
(18,'op_menu - Estadisticas Actividad');
INSERT INTO codificacion_actividades VALUES
(19,'consultarEstadistica');
INSERT INTO codificacion_actividades VALUES
(20,'op_menu - Vista Unica Tabla');
INSERT INTO codificacion_actividades VALUES
(21,'reordenar');
INSERT INTO codificacion_actividades VALUES
(22,'insertarFila');
INSERT INTO codificacion_actividades VALUES
(23,'cancelarInsercionFila');
INSERT INTO codificacion_actividades VALUES
(24,'guardarFilaInsertada');
INSERT INTO codificacion_actividades VALUES
(25,'eliminarFilaSeleccionada');
INSERT INTO codificacion_actividades VALUES
(26,'actualizadaColumnaJTable');
INSERT INTO codificacion_actividades VALUES
(27,'op_menu - Vista Paginada Tabla');
```

```
INSERT INTO codificacion_actividades VALUES  
(28,'botonPaginacionAnterior');  
INSERT INTO codificacion_actividades VALUES  
(29,'botonPaginacionSiguiente');  
INSERT INTO codificacion_actividades VALUES  
(30,'botonPaginacionNumerica');  
INSERT INTO codificacion_actividades VALUES  
(31,'op_menu - CierreVentana');  
INSERT INTO codificacion_actividades VALUES  
(32,'op_menu - Vista Arbol');  
INSERT INTO codificacion_actividades VALUES (33,'Nuevo  
nodo');  
INSERT INTO codificacion_actividades VALUES  
(34,'Editar nodo');  
INSERT INTO codificacion_actividades VALUES  
(35,'Eliminar nodo');  
INSERT INTO codificacion_actividades VALUES  
(36,'ratonClicked');  
INSERT INTO codificacion_actividades VALUES  
(37,'cancelarEdicionNodo');  
INSERT INTO codificacion_actividades VALUES  
(38,'op_menu - Configurar Documento');  
INSERT INTO codificacion_actividades VALUES  
(39,'generarPDF');  
INSERT INTO codificacion_actividades VALUES  
(40,'imprimir');  
INSERT INTO codificacion_actividades VALUES  
(41,'copiarAlPortapapeles');  
INSERT INTO generos VALUES ('0','Generalidades');  
INSERT INTO generos VALUES ('1','Filosofia.  
Psicologia');  
INSERT INTO generos VALUES ('2','Religion. Teologia');  
INSERT INTO generos VALUES ('3','Ciencias sociales');  
INSERT INTO generos VALUES ('4','No clasificado');  
INSERT INTO generos VALUES ('5','Matematicas. Ciencias  
naturales');  
INSERT INTO generos VALUES ('6','Ciencias aplicadas');
```

```

INSERT INTO generos VALUES ('7','Bellas artes.
Deportes');
INSERT INTO generos VALUES ('8','Literatura');
INSERT INTO generos VALUES ('9','Geografia.
Historia');
INSERT INTO libros VALUES
(LPAD(FORMAT(secuencia_next_valor("secuencia_libros")),
0),5,'0'),'EL QUIJOTE','8','1987-04-12
00:00:00',890,1);
INSERT INTO libros VALUES
(LPAD(FORMAT(secuencia_next_valor("secuencia_libros")),
0),5,'0'),'PROGRAMACION ORIENTADA A OBJETOS EN
JAVA','0','2019-12-11 00:00:00',588,0);
INSERT INTO libros VALUES
(LPAD(FORMAT(secuencia_next_valor("secuencia_libros")),
0),5,'0'),'POESIAS COMPLETAS','8','2006-06-18
00:00:00',540,1);
/*
Se han suprimido las instrucciones SQL que
correspondería estuviesen ubicadas en este lugar a
efectos de minimizar la extensión del libro.
*/
INSERT INTO libros VALUES
(LPAD(FORMAT(secuencia_next_valor("secuencia_libros")),
0),5,'0'),'GUERRA Y PAZ','8','1987-11-12
00:00:00',810,0);
INSERT INTO libros VALUES
(LPAD(FORMAT(secuencia_next_valor("secuencia_libros")),
0),5,'0'),'EL LAZARILLO DE TORMES','8','1983-04-22
00:00:00',358,0);
COMMIT;

```

## MySQL - VOTACIONES

```

DROP DATABASE IF EXISTS `votaciones`;
CREATE DATABASE IF NOT EXISTS `votaciones` /*!40100
DEFAULT CHARACTER SET utf8 */;
```

```
USE `votaciones`;
DROP TABLE IF EXISTS votos_emitidos;
DROP TABLE IF EXISTS votaciones;
DROP TABLE IF EXISTS titulares;
CREATE TABLE titulares
( codigo CHAR(3),
nombre VARCHAR(50),
cuota_participacion DOUBLE,
CONSTRAINT pk_titulares PRIMARY KEY (codigo)
);
CREATE TABLE votaciones
( id_votacion CHAR(19),
tema_votado VARCHAR(200),
CONSTRAINT pk_votaciones PRIMARY KEY (id_votacion)
);
CREATE TABLE votos_emitidos
( codigo_titular CHAR(3),
id_votacion CHAR(19),
opcion_ausente INT,
opcion_abstencion INT,
opcion_afirmativo INT,
opcion_negativo INT,
CONSTRAINT pk_votosemitidos PRIMARY KEY
(codigo_titular, id_votacion),
CONSTRAINT fk_votosemitidos_titulares FOREIGN KEY
(codigo_titular) REFERENCES titulares (codigo),
CONSTRAINT fk_votosemitidos_votaciones FOREIGN KEY
(id_votacion) REFERENCES votaciones (id_votacion)
);
-----
-----
-- MEDIANTE LA SIGUIENTE TABLA, PROCEDIMIENTOS Y
-- FUNCION SIMULAMOS EN MySQL LAS SECUENCIAS DE ORACLE
-----
```

```
DROP TABLE IF EXISTS secuencias;
DROP PROCEDURE IF EXISTS crea_secuencia;
DROP PROCEDURE IF EXISTS elimina_secuencia;
DROP PROCEDURE IF EXISTS secuencia_set_valor;
DROP PROCEDURE IF EXISTS secuencia_set_incremeto;
DROP FUNCTION IF EXISTS secuencia_next_valor;
CREATE TABLE secuencias
(
secuencia_nombre VARCHAR(35) NOT NULL PRIMARY KEY,
secuencia_valor INT UNSIGNED NOT NULL,
secuencia_incremeto INT UNSIGNED NOT NULL
);
DELIMITER //
CREATE PROCEDURE crea_secuencia(secuenciaNombre
VARCHAR(35), valorIncialSecuencia INT UNSIGNED,
incremento INT UNSIGNED)
BEGIN
IF (SELECT COUNT(*) FROM secuencias WHERE
secuencia_nombre = secuenciaNombre) = 0 THEN
INSERT INTO secuencias (secuencia_nombre,
secuencia_valor, secuencia_incremeto) VALUES
(secuenciaNombre, valorIncialSecuencia, incremento);
ELSE
SIGNAL SQLSTATE '50000' SET MESSAGE_TEXT = "No puede
crear la secuencia especificada porque ya existe";
END IF;
END//
CREATE PROCEDURE elimina_secuencia(secuenciaNombre
VARCHAR(35))
BEGIN
IF (SELECT COUNT(*) FROM secuencias WHERE
secuencia_nombre = secuenciaNombre) > 0 THEN
DELETE FROM secuencias WHERE secuencia_nombre =
secuenciaNombre;
ELSE
SIGNAL SQLSTATE '50000' SET MESSAGE_TEXT = "La
secuencia especificada no existe";
```

```
END IF;
END//  
CREATE PROCEDURE secuencia_set_valor(secuenciaNombre  
VARCHAR(35), secuenciaValor INT UNSIGNED)
BEGIN
IF (SELECT COUNT(*) FROM secuencias WHERE  
secuencia_nombre = secuenciaNombre) > 0 THEN
UPDATE secuencias SET secuencia_valor = secuenciaValor
WHERE secuencia_nombre = secuenciaNombre;
ELSE
SIGNAL SQLSTATE '50000' SET MESSAGE_TEXT = "No existe  
la secuencia";
END IF;
END//  
CREATE PROCEDURE  
secuencia_set_increemento(secuenciaNombre VARCHAR(35),  
incremento INT UNSIGNED)
BEGIN
IF (SELECT COUNT(*) FROM secuencias WHERE  
secuencia_nombre = secuenciaNombre) > 0 THEN
UPDATE secuencias SET secuencia_increemento =  
incremento WHERE secuencia_nombre = secuenciaNombre;
ELSE
SIGNAL SQLSTATE '50000' SET MESSAGE_TEXT = "No existe  
la secuencia";
END IF;
END//  
CREATE FUNCTION secuencia_next_valor(secuenciaNombre  
VARCHAR(35)) RETURNS INT UNSIGNED
BEGIN
DECLARE valorActual INT;
SET valorActual = (SELECT secuencia_valor FROM  
secuencias WHERE secuencia_nombre = secuenciaNombre);
IF valorActual IS NOT NULL THEN
UPDATE secuencias SET secuencia_valor = valorActual +  
secuencia_increemento WHERE secuencia_nombre =  
secuenciaNombre;
```

```

ELSE
SIGNAL SQLSTATE '50000' SET MESSAGE_TEXT = "No existe
la secuencia";
END IF;
RETURN valorActual;
END//  

DELIMITER ;
-----  

-----  

-----  

-- CALL elimina_secuencia("secuencia_codigo_titular");
CALL crea_secuencia("secuencia_codigo_titular", 1, 1);
-- CALL
secuencia_set_valor("secuencia_codigo_titular", 100);
-- CALL
secuencia_set_incremendo("secuencia_codigo_titular",
5);
-- SELECT
secuencia_next_valor("secuencia_codigo_titular");
-----  

-----  

-----  

INSERT INTO titulares
(codigo,nombre,cuota_participacion) VALUES
(LPAD(FORMAT(secuencia_next_valor("secuencia_codigo_titular"),0),3,'0'),'ALARCON LAGUNA, CONCEPCION',1.143);
INSERT INTO titulares
(codigo,nombre,cuota_participacion) VALUES
(LPAD(FORMAT(secuencia_next_valor("secuencia_codigo_titular"),0),3,'0'),'ALPUENTE CARRASCO, VICENTA',1.480);
INSERT INTO titulares
(codigo,nombre,cuota_participacion) VALUES
(LPAD(FORMAT(secuencia_next_valor("secuencia_codigo_titular"),0),3,'0'),'APARISI SERRANO, FELIX',0.937);
/*
Se han suprimido las instrucciones SQL que
correspondería estuviesen ubicadas en este lugar a

```

```

efectos de minimizar la extensión del libro.
*/
INSERT INTO titulares
(codigo,nombre,cuota_participacion) VALUES
(LPAD(FORMAT(secuencia_next_valor("secuencia_codigo_titular"),0),3,'0'),'ZAPATERO MARCO, SALVADOR',0.905);
INSERT INTO titulares
(codigo,nombre,cuota_participacion) VALUES
(LPAD(FORMAT(secuencia_next_valor("secuencia_codigo_titular"),0),3,'0'),'ZARAGOZA ASENSI, ROSA',1.458);
COMMIT;

```

## Oracle - BIBLIOTECA

```

DROP TABLE libros;
DROP TABLE generos;
DROP TABLE actividad_usuarios;
DROP TABLE codificacion_actividades;
DROP TABLE usuarios_biblioteca;
CREATE TABLE usuarios_biblioteca
(
id_usuario CHAR(15),
password VARCHAR(12),
CONSTRAINT pk_usuariosbiblioteca PRIMARY KEY
(id_usuario)
);
CREATE TABLE codificacion_actividades
(
codigo NUMBER,
descripcion VARCHAR(50),
CONSTRAINT pk_codificacionactividades PRIMARY KEY
(codigo)
);
CREATE TABLE actividad_usuarios
(
fecha_hora TIMESTAMP,
id_usuario CHAR(15),

```

```
ip_cliente VARCHAR(18),
codigo NUMBER,
CONSTRAINT fk_actividad_codificacion FOREIGN KEY
(codigo) REFERENCES codificacion_actividades (codigo)
);
CREATE TABLE generos
(
codigo CHAR(1),
descripcion VARCHAR(32),
CONSTRAINT pk_generos PRIMARY KEY (codigo)
);
CREATE TABLE libros
(
id_libro CHAR(5),
titulo VARCHAR(60),
genero CHAR(1),
fecha_edicion DATE,
numero_paginas NUMBER,
premiado NUMBER,
CONSTRAINT pk_libros PRIMARY KEY (id_libro),
CONSTRAINT fk_libros_generos FOREIGN KEY (genero)
REFERENCES generos (codigo)
);
INSERT INTO libros VALUES ('00001','ES UNA FILA A
ELIMINAR',NULL,TO_DATE('12-04-1987','DD-MM-
YYYY'),890,1);
DELETE FROM libros;
COMMIT;
DROP SEQUENCE secuencia_libros;
CREATE SEQUENCE secuencia_libros MINVALUE 0 START WITH
1;
INSERT INTO usuarios_biblioteca VALUES
('usuario1','password1');
INSERT INTO usuarios_biblioteca VALUES
('usuario2','password2');
INSERT INTO usuarios_biblioteca VALUES
('usuario3','password3');
```

```
INSERT INTO codificacion_actividades VALUES  
(1,'op_menu - Conexion');  
INSERT INTO codificacion_actividades VALUES  
(2,'op_menu - Desconexion');  
INSERT INTO codificacion_actividades VALUES  
(3,'autenticacion');  
INSERT INTO codificacion_actividades VALUES  
(4,'desconectar');  
INSERT INTO codificacion_actividades VALUES  
(5,'op_menu - ConexionEfectuada');  
INSERT INTO codificacion_actividades VALUES  
(6,'op_menu - Volcado a BD');  
INSERT INTO codificacion_actividades VALUES  
(7,'volcarIncidencias');  
INSERT INTO codificacion_actividades VALUES  
(8,'op_menu - Vista Formulario');  
INSERT INTO codificacion_actividades VALUES  
(9,'comboLibros');  
INSERT INTO codificacion_actividades VALUES  
(10,'nuevoLibro');  
INSERT INTO codificacion_actividades VALUES  
(11,'aplicarCambios');  
INSERT INTO codificacion_actividades VALUES  
(12,'eliminarLibro');  
INSERT INTO codificacion_actividades VALUES  
(13,'actualizadoTitulo');  
INSERT INTO codificacion_actividades VALUES  
(14,'actualizadoGenero');  
INSERT INTO codificacion_actividades VALUES  
(15,'actualizadoFechaEdicion');  
INSERT INTO codificacion_actividades VALUES  
(16,'actualizadoNumeroPaginas');  
INSERT INTO codificacion_actividades VALUES  
(17,'actualizadoPremiado');  
INSERT INTO codificacion_actividades VALUES  
(18,'op_menu - Estadisticas Actividad');  
INSERT INTO codificacion_actividades VALUES
```

```
(19,'consultarEstadistica');
INSERT INTO codificacion_actividades VALUES
(20,'op_menu - Vista Unica Tabla');
INSERT INTO codificacion_actividades VALUES
(21,'reordenar');
INSERT INTO codificacion_actividades VALUES
(22,'insertarFila');
INSERT INTO codificacion_actividades VALUES
(23,'cancelarInsertionFila');
INSERT INTO codificacion_actividades VALUES
(24,'guardarFilaInsertada');
INSERT INTO codificacion_actividades VALUES
(25,'eliminarFilaSeleccionada');
INSERT INTO codificacion_actividades VALUES
(26,'actualizadaColumnaJTable');
INSERT INTO codificacion_actividades VALUES
(27,'op_menu - Vista Paginada Tabla');
INSERT INTO codificacion_actividades VALUES
(28,'botonPaginacionAnterior');
INSERT INTO codificacion_actividades VALUES
(29,'botonPaginacionSiguiente');
INSERT INTO codificacion_actividades VALUES
(30,'botonPaginacionNumerica');
INSERT INTO codificacion_actividades VALUES
(31,'op_menu - CierreVentana');
INSERT INTO codificacion_actividades VALUES
(32,'op_menu - Vista Arbol');
INSERT INTO codificacion_actividades VALUES (33,'Nuevo
nodo');
INSERT INTO codificacion_actividades VALUES
(34,'Editar nodo');
INSERT INTO codificacion_actividades VALUES
(35,'Eliminar nodo');
INSERT INTO codificacion_actividades VALUES
(36,'ratonClicked');
INSERT INTO codificacion_actividades VALUES
(37,'cancelarEdicionNodo');
```

```
INSERT INTO codificacion_actividades VALUES  
(38,'op_menu - Configurar Documento');  
INSERT INTO codificacion_actividades VALUES  
(39,'generarPDF');  
INSERT INTO codificacion_actividades VALUES  
(40,'imprimir');  
INSERT INTO codificacion_actividades VALUES  
(41,'copiarAlPortapapeles');  
INSERT INTO generos VALUES ('0','Generalidades');  
INSERT INTO generos VALUES ('1','Filosofia.  
Psicologia');  
INSERT INTO generos VALUES ('2','Religion. Teologia');  
INSERT INTO generos VALUES ('3','Ciencias sociales');  
INSERT INTO generos VALUES ('4','No clasificado');  
INSERT INTO generos VALUES ('5','Matematicas. Ciencias  
naturales');  
INSERT INTO generos VALUES ('6','Ciencias aplicadas');  
INSERT INTO generos VALUES ('7','Bellas artes.  
Deportes');  
INSERT INTO generos VALUES ('8','Literatura');  
INSERT INTO generos VALUES ('9','Geografia.  
Historia');  
INSERT INTO libros VALUES  
(REPLACE(TO_CHAR(secuencia_libros.NEXTVAL,'09999'),'  
' ),'EL QUIJOTE','8',TO_DATE('12-04-1987','DD-MM-  
YYYY'),890,1);  
INSERT INTO libros VALUES  
(REPLACE(TO_CHAR(secuencia_libros.NEXTVAL,'09999'),'  
' ),'PROGRAMACION ORIENTADA A OBJETOS EN  
JAVA','0',TO_DATE('11-12-2019','DD-MM-YYYY'),588,0);  
INSERT INTO libros VALUES  
(REPLACE(TO_CHAR(secuencia_libros.NEXTVAL,'09999'),'  
' ),'POESIAS COMPLETAS','8',TO_DATE('18-06-2006','DD-  
MM-YYYY'),540,1);  
/*  
Se han suprimido las instrucciones SQL que  
correspondería estuviesen ubicadas en este lugar a
```

```

efectos de minimizar la extensión del libro.
*/
INSERT INTO libros VALUES
(REPLACE(TO_CHAR(secuencia_libros.NEXTVAL,'09999'),'
'),'GUERRA Y PAZ','8',TO_DATE('12-11-1987','DD-MM-
YYYY'),810,0);
INSERT INTO libros VALUES
(REPLACE(TO_CHAR(secuencia_libros.NEXTVAL,'09999'),'
'),'EL LAZARILLO DE TORMES','8',TO_DATE('22-04-
1983','DD-MM-YYYY'),358,0);
COMMIT;

```

## Oracle - VOTACIONES

```

DROP TABLE votos_emitidos;
DROP TABLE votaciones;
DROP TABLE titulares;
CREATE TABLE titulares
( codigo CHAR(3),
nombre VARCHAR(50),
cuota_participacion NUMBER,
CONSTRAINT pk_titulares PRIMARY KEY (codigo)
);
CREATE TABLE votaciones
( id_votacion CHAR(19),
tema_votado VARCHAR(200),
CONSTRAINT pk_votaciones PRIMARY KEY (id_votacion)
);
CREATE TABLE votos_emitidos
( codigo_titular CHAR(3),
id_votacion CHAR(19),
opcion_ausente NUMBER,
opcion_abstencion NUMBER,
opcion_afirmativo NUMBER,
opcion_negativo NUMBER,
CONSTRAINT pk_votosemitidos PRIMARY KEY
(codigo_titular, id_votacion),

```

```
CONSTRAINT fk_votosemitidos_titulares FOREIGN KEY
(codigo_titular) REFERENCES titulares (codigo),
CONSTRAINT fk_votosemitidos_votaciones FOREIGN KEY
(id_votacion) REFERENCES votaciones (id_votacion)
);
INSERT INTO titulares
(codigo,nombre,cuota_participacion) VALUES
('001','FILA A ELIMINAR',2.879);
DELETE FROM titulares;
COMMIT;
DROP SEQUENCE secuencia_titulares;
CREATE SEQUENCE secuencia_titulares MINVALUE 0 START
WITH 1;
INSERT INTO titulares
(codigo,nombre,cuota_participacion) VALUES
(REPLACE(TO_CHAR(secuencia_titulares.NEXTVAL,'099'),'
'),'ALARCON LAGUNA, CONCEPCION',1.143);
INSERT INTO titulares
(codigo,nombre,cuota_participacion) VALUES
(REPLACE(TO_CHAR(secuencia_titulares.NEXTVAL,'099'),'
'),'ALPUENTE CARRASCO, VICENTA',1.480);
INSERT INTO titulares
(codigo,nombre,cuota_participacion) VALUES
(REPLACE(TO_CHAR(secuencia_titulares.NEXTVAL,'099'),'
'),'APARISI SERRANO, FELIX',0.937);
/*
Se han suprimido las instrucciones SQL que
correspondería estuviesen ubicadas en este lugar a
efectos de minimizar la extensión del libro.
*/
INSERT INTO titulares
(codigo,nombre,cuota_participacion) VALUES
(REPLACE(TO_CHAR(secuencia_titulares.NEXTVAL,'099'),'
'),'ZAPATERO MARCO, SALVADOR',0.905);
INSERT INTO titulares
(codigo,nombre,cuota_participacion) VALUES
(REPLACE(TO_CHAR(secuencia_titulares.NEXTVAL,'099'),'
```

' ),'ZARAGOZA ASENSI, ROSA',1.458);  
COMMIT;

# MATERIAL ADICIONAL

El lector tiene a su disposición todas las aplicaciones ejemplo presentadas y estudiadas en profundidad a lo largo de este libro. Cada una de ellas se presenta como un directorio independiente con el nombre del Proyecto. Todos ellos son directamente ejecutables. Puede acceder a cada una de dichas aplicaciones de varias formas:

1. Abriendo directamente como Proyecto mediante el IDE NetBeans.
2. Exportando cada uno de dichos Proyectos a otro IDE (como por ejemplo, Eclipse) que el lector crea oportuno utilizar.
3. Trabajando en línea de comandos. Caso de recurrir a esta modalidad de trabajo, podrá localizar los ficheros fuentes en el directorio “src” de cada Proyecto. Así mismo, en los casos de aplicaciones que requieran el uso de librerías, encontrará los ficheros con extensión “jar” correspondientes las mismas en el directorio “lib”. Y cuando se contemple la utilización de ficheros XML como repositorios centralizados de información, los encontrará en el directorio “xml”.

El material adicional de este libro puede descargarlo en nuestro portal web: <http://www.ra-ma.es>.

Debe dirigirse a la ficha correspondiente a esta obra, dentro de la ficha encontrará el enlace para poder realizar la descarga.

Cuando descomprima el fichero obtendrá los archivos que complementan al libro para que pueda continuar con su aprendizaje.

## **INFORMACIÓN ADICIONAL Y GARANTÍA**

- RA-MA EDITORIAL garantiza que estos contenidos han sido sometidos a un riguroso control de calidad.
- Los archivos están libres de virus, para comprobarlo se han utilizado las últimas versiones de los antivirus líderes en el mercado.
- RA-MA EDITORIAL no se hace responsable de cualquier pérdida, daño o costes provocados por el uso incorrecto del contenido descargable.
- Este material es gratuito y se distribuye como contenido complementario al libro que ha adquirido, por lo que queda terminantemente prohibida su venta o distribución.