Tema 15

Aplicaciones de Dos Capas

Una aplicación de dos capas es una en la cual el código está separado del sistema de administración de la base de datos. Esas capas pueden estar separadas lógicamente, es decir cada una ejecuta en un proceso diferente en la misma computadora, o separadas físicamente, es decir cada una ejecuta en procesos diferentes en computadoras diferentes.

Una aplicación de dos capas, realmente tiene tres partes: un cliente, un servidor y un protocolo. El protocolo es el puente entre las capas del cliente y el servidor. En la capa del cliente reside la interfaz del usuario y la lógica del negocio, mientras que en el servidor está un administrador de bases de datos.

El protocolo en una aplicación de dos capas comunica una capa con la otra, que en ocasiones pueden residir en dos computadoras diferentes. El cliente le hace solicitudes al servidor y el servidor le regresa al cliente los resultados de esas solicitudes.

En Java, ese protocolo toma la forma de una API para facilitarnos la construcción de aplicaciones cliente – servidor. Los métodos de las clases de esta API establecen un mecanismo estándar para que un cliente escrito en Java se comunique con administradores de bases de datos de diferentes fabricantes. Esta API llamada JDBC se encuentran en los paquetes:

java.sql
javax.sql

Adicionalmente a la API, se requiere para cada base de datos, un manejador JDBC, el cual es un componente que permite la comunicación entre la API y un administrador de bases de datos. Este manejador, normalmente no los proporciona el fabricante del administrador de base de datos.

En la figura 15.1 se muestra un diagrama de clases parcial de las interfaces y clases que forman parte de la API java:

- `java.sql.Driver`: Esta interfaz declara los métodos que toda clase que implementa un manejador JDBC debe tener. Normalmente los manejadores son proporcionados por los fabricantes de administradores de bases de datos.

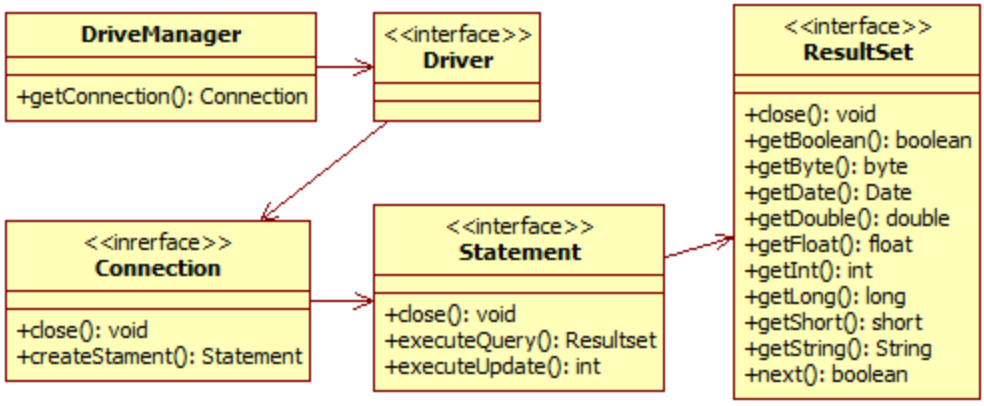

Figura 15.1. Diagrama de clases parcial de la API JDBC de Java

- `java.sql.DriverManager`: Es la clase que permite la administración de los manejadores JDBC. `DriveManager` cargará a memoria el manejador JDBC de cada base de datos a la que nuestro programa desea comunicarse. Para cargar un manejador JDBC a memoria, `DriveManager` crea una instancia de la clase que implementa al manejador. Sin embargo en lugar de utilizar el operador `new` para crear la instancia de la clase, `DriveManager` utliza un mecanismo que permite especificar el nombre de la clase en tiempo de ejecución en lugar de en tiempo de compilación. Este mecanismo permite crear una instancia de una clase dado el nombre de la clase y utiliza la siguiente sintaxis:

  ```
  Class.forName("nombreManejadorJDBC").newInstance();
  ```

  Donde *`nombreManejadorJDBC`* es el nombre de del manejador JDBC.

  Una vez que se ha cargado el manejador JDBC en memoria necesitamos establecer una conexión entre la API JDBC y el administrador de la base de datos. Para ello invocamos al método estático `getConnection()` de `DriveManager`, cuya sintaxis se muestra en la tabla 15.1:

- `java.sql.Connection`: Esta interfaz declara los métodos que debe tener una clase que represente una conexión lógica entre la API JDBC y la base de datos. Esta conexión nos permiten enviarle al manejador de la base de datos sentencias SQL y recuperar las repuestas del manejador de bases de datos. Estas sentencias se encapsulas en objetos que implementan la interfaz `java.sql.Statement`. Para crear una sentencia SQL Connection tiene un método llamado createStatement cuya sintaxis se muestra en la tabla 15.2:

Tabla 15.1. Método `getConnection()` de la clase `DriveManager`

`public static Connection getConnection(String` *`url`*`, `**`String`** *`user`*`,`
`String` *`password`***`) throws SQLException`**

Establece la conexión con la base de datos.

Parámetros:

url. URL de la base de datos. La sintaxis de *url* es:

"**`jdbc`**`:`*`subprotocolo`*`://`*`dirIP_Servidor`*`/`*`nomBaseDatos`*"

donde

`subprotocolo` identifica al vendedor del manejador de base de datos.
`dirIP_Servidor` es la dirección IP del servidor. Si la base de datos se encuentra en la misma computadora que contiene al cliente, podemos utilizar la palabra **`localhost`** en lugar de la dirección IP.
`nomBaseDatos` es el nombre de la base de datos.

`user`. Es el nombre de un usuario con acceso a la base de datos.

`password`. Es la contraseña del usuario con acceso a la base de datos.

Regresa:

Un objeto del tipo `Connection`, la conexión a la base de datos..

Lanza:

`SQLException` – Si no se puede establecer la conexión con la base datos.

Tabla 15.2. Métodos de la interfaz `Connection`

`public void close()` `throws SQLException`

Cierra la conexión con la base de datos y libera los recursos asignados a esta conexión.

Lanza:

`SQLException` – Si hay un problema al acceder a la base datos.

`public Statement createStatement()throws SQLException`

Crea un objeto del tipo `Statement` para enviar sentencias SQL a la base de datos.

Regresa:

Un objeto del tipo `Statement.`

Lanza:

`SQLException` – Si no se puede establecer la conexión con la base datos.

- `java.sql.Statement`: Esta interfaz declara los métodos que debe tener una clase que represente el mecanismo que nos permita enviarle al manejador de la base de datos sentencias SQL y recuperar las repuestas del manejador de bases de datos. Los métodos de esta interfaz se muestran en la tabla 15.3:

Tabla 15.3. Métodos de la interfaz `Statement`

`public void close()` `throws SQLException` Cierra la sentencia y libera los recursos asignados a esta sentencia. Si hay un objeto del tipo ResultSet asociado a esta sentencia también se cierra. **Lanza**: `SQLException` – Si hay un problema al acceder a la base datos.
`public ResultSet executeQuery(String sql) throws SQLException` Ejecuta la sentencia SQL dada por el parámetro, la cual regresa un solo objeto con la respuesta a la sentencia. **Parámetro:** `sql` – Sentencia SQL a enviarse al manejador de base de datos. Normalmente una sentencia `select`. **Regresa**: Un objeto del tipo `ResultSet` con los datos producidos por la consulta, nunca `null`**.** **Lanza**: `SQLException` – Si no se puede acceder a la base datos o se produce cualquier respuesta que no sea del tipo `ResultSet`.
`public` `int executeUpdate(String` ***`sql`***`)` **`throws SQLException`** Ejecuta la sentencia SQL dada por el parámetro, la cual puede ser una sentencia `insert`, `update`, `delete`. También pueden ser sentencias `create table`, `alter table` y `drop table`, entre otras. **Parámetro:** `sql` – Sentencia `insert`, `update`, `delete` a enviarse al manejador de base de datos. También puede ser una sentencia que no regrese nada. **Regresa**: Número de renglones modificados por la sentencia o 0 si la sentencia no regresa nada. **Lanza**: `SQLException` – Si no se puede acceder a la base datos o se produce cualquier respuesta que sea del tipo `ResultSet`.

- `java.sql.ResultSet`: Esta interfaz declara los métodos que debe tener una clase que represente una tabla de datos producida por una consulta a la base de datos. Un objeto de esta clase mantiene un cursor apuntando al renglón actual de la tabla. Inicialmente, el cursor está posicionado antes del primer renglón. Para obtener el siguiente renglón de la tabla se invoca al método `next()`. La interfaz declara métodos que permiten recuperar el valor de una columna de un renglón de la tabla. Los métodos de esta interfaz se muestran en la tabla 15.4:

Tabla 15.4. Métodos de la interfaz `ResultSet`

`public void close()` `throws SQLException`

Cierra el objeto `ResultSet` y libera los recursos asignados a este objeto. Un objeto del tipo `ResultSet` también se cierra cuando su sentencia asociada se cierra.

Lanza:

`SQLException` – Si hay un problema al acceder a la base datos.

`public boolean getBoolean(int` *`columnIndex`***`) throws SQLException`**
`public boolean getBoolean(String` *`columnName`***`) throws SQLException`**
`public byte getByte(int` *`columnIndex`***`) throws SQLException`**
`public byte getByte(String` *`columnName`***`) throws SQLException`**
`public short getShort(int` *`columnIndex`***`) throws SQLException`**
`public short getShort(String` *`columnName`***`) throws SQLException`**
`public int getInt(int` *`columnIndex`***`) throws SQLException`**
`public int getInt(String` *`columnName`***`) throws SQLException`**
`public long getLong(int` *`columnIndex`***`) throws SQLException`**
`public long getLong(String` *`columnName`***`) throws SQLException`**
`public float getFloat(int` *`columnIndex`***`) throws SQLException`**
`public float getFloat(String` *`columnName`***`) throws SQLException`**
`public double getDouble(int` *`columnIndex`***`) throws SQLException`**
`public double getDouble(String` *`columnName`***`) throws SQLException`**
`public String getString(int` *`columnIndex`***`) throws SQLException`**
`public String getString(String` *`columnName`***`) throws SQLException`**
`public Date getDate(int` *`columnIndex`***`) throws SQLException`**
`public Date getDate(String` *`columnName`***`) throws SQLException`**

Obtiene el valor de la columna dada por el parámetro, en el renglón actual del objeto de tipo `ResultSet` como un valor del tipo regresado por el método.

Parámetro:

`columnIndex` – Número de la columna. La primera columna tiene el valor de 1, la segunda el valor de 2, etc.
`columnName` – Nombre SQL de la columna.

Regresa:

El valor de la columna como un valor del tipo regresado por el método. Si el valor SQL es `NULL` regresa `false`, `0`, o `null`**.**

Lanza:

`SQLException` – Si no se puede acceder a la base.

`Public Boolean next()throws SQLException`

Mueve el cursor al siguiente renglón de la tabla del objeto de tipo `ResultSet`. Inicialmente, el cursor se encuentra antes del primer renglón de la tabla por lo que la primera llamada a este método posiciona el cursor en el primer renglón de la tabla.

Regresa:

Si el cursor se mueve después del último renglón de la tabla, el método regresa falso, verdadero en caso contrario.

Lanza:

`SQLException` – Si no se puede acceder a la base.

Para encapsular la funcionalidad de las clases e interfaces de la API JDBC anteriores crearemos dos clases: `Conexión` y `Tabla` que proveen métodos para crear una conexión con la base de datos y ejecutar sentencias SQL. También el acceso a cada tabla de una base de datos será encapsulada mediante una clase que permitirá obtener, agregar, actualizar, eliminar y consultar renglones a esa tabla. Esas clases se implementarán como subclases de la clase `Tabla`. El diagrama de estas clases en el programa Amante Música se muestra en la figura 15.2:

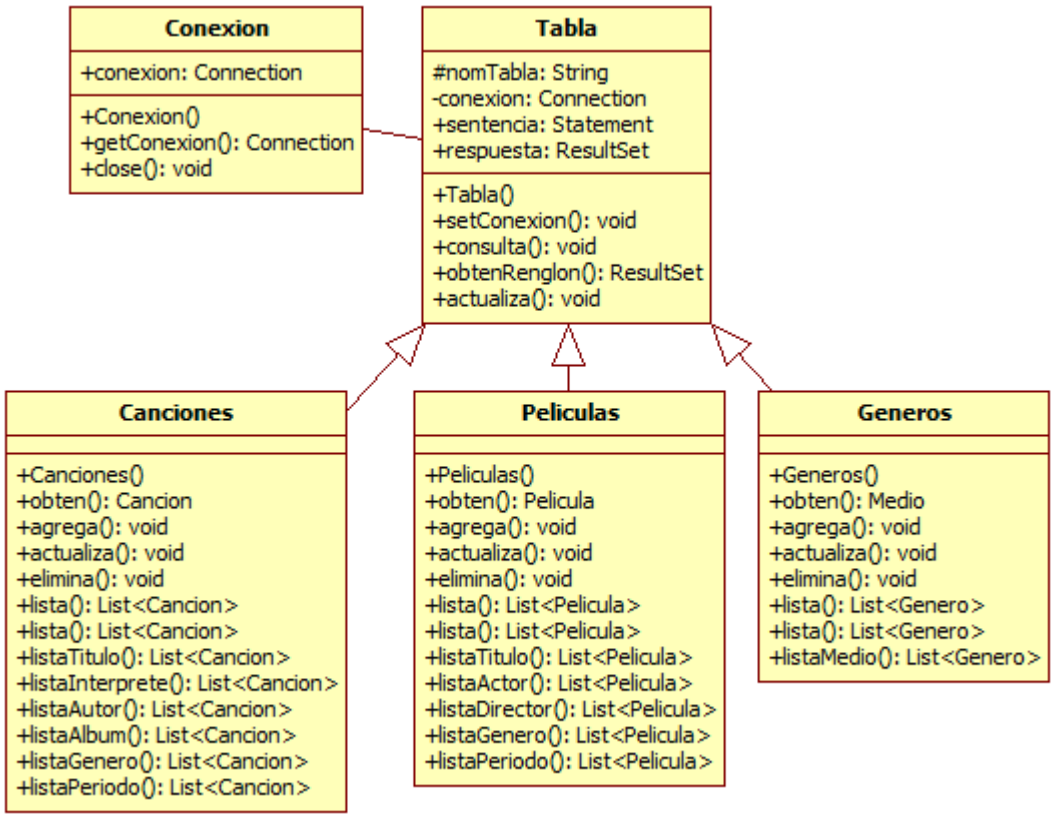

Figura 15.2

El constructor de la clase `Conexión` nos permite conectarnos con la base de datos. El método `getConexion()` nos regresa esa conexión, mientras que el método `close()` Termina la conexión con la base de datos. El código de la clase `Conexión` es el siguiente:

Conexion.java

```
/*
 * Conexion.java
 *
 * Creada el  15 de septiembre de 2007, 12:21 PM
 */

package persistencia;

import java.net.*;
```

```
import java.sql.*;
import excepciones.PersistenciaException;

/**
 * Esta clase establece una conexión con una base de datos mySQL.
 *
 * @author mdomitsu
 */
public class Conexion {

  // Conector a la base de datos.
  private Connection conexion;

  /**
   * Establece una conexión con la base de datos.
   * @param url URL de la base de datos
   * @param usuario Cuenta de usuario
   * @param password Contraseña del usuario
   * @throws java.lang.Exception Si no puede establecer la conexión con la
   * base de datos
   */
  public Conexion(String url, String usuario, String password)
      throws PersistenciaException {
    try {
      // Se crea una instancia de la clase manejadora de mySQL
      Class.forName("org.gjt.mm.mysql.Driver").newInstance();

      // Se conecta con la base de datos
      conexion = DriverManager.getConnection(url, usuario, password);
    }
    catch(Exception e) {
      throw new
        PersistenciaException("Error al conectarse a la base de datos", e);
    }
  }

  /**
   * Obtiene la conexión con la base de datos.
   * @return La conexión con la base de datos
   */
  public Connection getConexion() {
    return conexion;
  }

  public void close() throws PersistenciaException {
    try {
      conexion.close();
    }
    catch(Exception e) {
      throw new
PersistenciaException("Error al cerrar la conexión con la base de datos", e);
    }
  }
}
```

La clase siguiente nos proporciona métodos para ejecutar las sentencias SQL básicas:

Tabla.java

```
/*
 * Tabla.java
 *
 * Creada el  15 de septiembre de 2007, 12:21 PM
 */

package persistencia;

import java.net.*;
import java.sql.*;
import excepciones.PersistenciaException;

/**
 * Esta clase implementa las operaciones de consultar, actualizar y obtener
 * el siguiente renglón de una tabla de una base de datos SQL.
 *
 * @author mdomitsu
 */
public class Tabla {
  // Nombre de la tabla en la que se almacenan los datos
  protected String nomTabla;
  // Conexión con la base de datos */
  private Connection conexion;
  // Sentencia SQL que se le envia al manejador de la base de datos para
  // consultar, insertar, actualizar, borrar.
  private Statement sentencia;
  // Respuesta del manejador de la base de datos a la sentencia SQL.
  private ResultSet respuesta;

  /**
   * Constructor del método. Establece el nombre de la tabla
   * en la que se almacenan los datos
   * @param nomTabla Nombre de la tabla en la que se almacenan los datos
   */
  public Tabla(String nomTabla) {
    // Establece el nombre de la tabla
    this.nomTabla = nomTabla;
  }

  /**
   * Establece la conexion con la base de datos
   * @param conexion Conexion con la base de datos
   */
  public void setConexion(Connection conexion) {
    this.conexion = conexion;
  }

  /**
   * Este método permite consultar una tabla de una base de datos mySQL.
   * @param sql Cadena con la sentencia con la consulta.
   * @throws SQLException Si no puede crear el Statement o ejecutar la
   * consulta
   */
  public void consulta(String sql) throws PersistenciaException {
    try {
```

```
      // Crea una sentencia para hacer la consulta
      sentencia = conexion.createStatement();

      // Ejecuta la consulta. El método executeQuery() regresa una tabla
      // que genera como resultado de la consulta
      respuesta = sentencia.executeQuery(sql);
    }
    catch(SQLException se) {
      throw new
       PersistenciaException("No se puede consultar a la base de datos", se);
    }
}

/**
 * Este método obtiene el siguiente renglón de la tabla generada por la
 * consulta hecha por el método executeSelect().
 * @return Si hay renglones, el siguiente renglón, null en caso contrario.
 * @throws PersistenciaException Si no puede obtener el siguiente renglon
 * de la tabla.
 */
public ResultSet obtenRenglon() throws PersistenciaException {
  try {
      // Si hay otro renglon en la tabla, regrésalo.
      if (respuesta.next())
        return respuesta;
      // Si no, cierra la sentencia, la tabla y regresa null.
      else {
        // Cierra la sentencia y la tabla y regresa null.
        respuesta.close();
        sentencia.close();
        return null;
      }
    }
    catch(SQLException se) {
      throw new
       PersistenciaException("No se puede consultar a la base de datos", se);
    }
 }

 /**
  * Este método permite modificar, insertar y borrar renglones de una tabla
  * de una base de datos mySQL.
  * @param sql Cadena con la sentencia para modificar, insertar o borrar.
  * @throws PersistenciaException Si no puede crear el Statement o ejecutar
  * la actualización
  */
 public void actualiza(String sql) throws PersistenciaException {
    try {
      // Crea una sentencia para hacer la modificación.
      sentencia = conexion.createStatement();

      // Ejecuta la modificación.
      int i = sentencia.executeUpdate(sql);

      // Cierra la sentencia y la tabla.
      sentencia.close();
    }
```

```
    catch(SQLException se) {
      throw new
      PersistenciaException("No se puede actualizar a la base de datos", se);
    }
  }
}
```

La figura 15.3 muestra el diagrama Entidad - Relación de las tablas **canciones**, **películas** y **generos** en la base de datos musica.

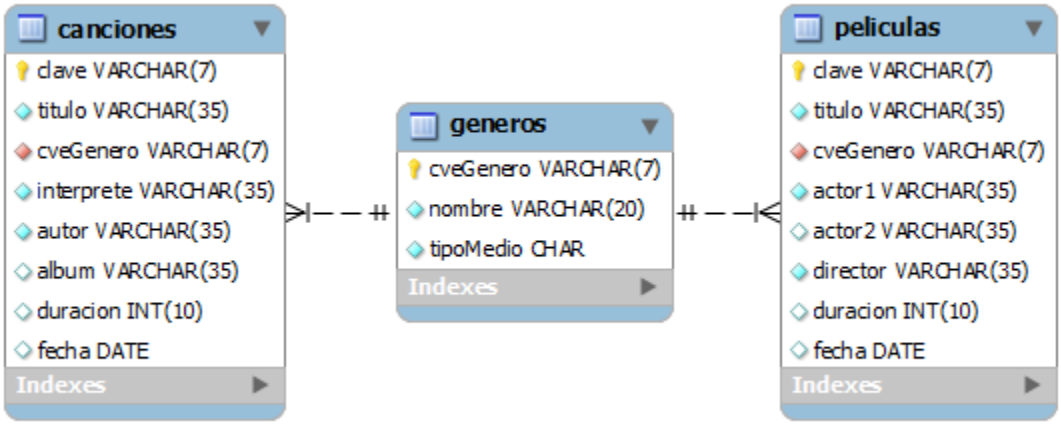

Figura 15.3

El siguiente es el listado parcial de la clase `Canciones` que implementa los métodos que nos permiten obtener, consultar, insertar, actualizar y eliminar renglones de la tabla canciones.

Cada renglón de la tabla canciones proviene de o se convierte en un objeto del tipo Cancion. Los métodos de esta clase lanzan excepciones del tipo `PersistenciaException`, la cual fue vista en el tema 6: Colecciones.

Canciones.java

```
/*
 * Canciones.java
 *
 * Creada el 15 de septiembre de 2007, 12:21 PM
 */

package persistencia;

import java.sql.*;
import java.util.List;
import java.util.ArrayList;
import objetosServicio.*;
import objetosNegocio.*;
import excepciones.PersistenciaException;
```

```
/**
 * Esta clase permite agregar, eliminar y consultar canciones del programa
 * AmanteMusica en su versión que usa una base de datos
 *
 * @author mdomitsu
*/
public class Canciones extends Tabla {

  /**
   * Constructor del método. Establece el nombre de la tabla
   * en la que se almacenan los datos
   * @param nomTabla Nombre de la tabla en la que se almacenan los datos
   */
  public Canciones(String nomTabla) {
    super(nomTabla);
  }

  /**
   * Este método obtiene una canción cuya clave es igual a la clave de la
   * canción dada por el parámetro.
   * @param cancion Objeto de tipo Cancion con la clave de la canción a
   * buscar
   * @return La Cancion si la encuentra. null en caso contrario.
   * @throws PersistenciaException Si no puede acceder a la base de datos
   */
  public Cancion obten(Cancion cancion) throws PersistenciaException {
    ResultSet renglon;

    // Crea la sentencia SQL para buscar la cancion en la tabla canciones.
    String sql = "SELECT * FROM "   + nomTabla +
                 " WHERE clave = '" + cancion.getClave() + "';";

    // Ejecuta la consulta
    consulta(sql);

    // Si hay un renglón en la tabla con la cancion buscada
    if ( (renglon = obtenRenglon()) != null) {
      try {
        // Crea un objeto del tipo Cancion con los campos del renglon de la
        // tabla.
        Cancion cancionHayada = new Cancion(renglon.getString("clave"),
                                            renglon.getString("titulo"),
                                            new Genero(renglon
                                                    .getString("cveGenero")),
                                            renglon.getString("interprete"),
                                            renglon.getString("autor"),
                                            renglon.getString("album"),
                                            renglon.getInt("duracion"),
                                            new Fecha(renglon
                                                      .getDate("fecha")));
        return cancionHayada;
      }
      catch (SQLException sqle) {
        throw new
          PersistenciaException("Error al acceder a la base de datos", sqle);
      }
    }
```

```
    // Si no, regresa null
    return null;
  }

  /**
   * Este método permite insertar un renglón de la tabla con los atributos de
   * los objetos de la clase Cancion.
   * @param cancion Cancion a insertar en la tabla cancioness
   * @throws PersistenciaException Si no puede acceder a la base de datos
   */
  public void agrega(Cancion cancion) throws PersistenciaException {
    String sql = "";

    // Crea la sentencia para insertar.
    // La sentencia debe terminar en un ;
    sql += "INSERT "           + nomTabla;
    sql += " SET clave = '"    + cancion.getClave() + "'";
    sql += ", titulo = '"      + cancion.getTitulo() + "'";
    sql += ", cveGenero = '"   + cancion.getGenero().getCveGenero() + "'";
    sql += ", interprete = '"  + cancion.getInterprete() + "'";
    sql += ", autorLetra = '"  + cancion.getAutor() + "'";
    sql += ", album = '"       + cancion.getAlbum() + "'";
    sql += ", duracion = '"    + cancion.getDuracion() + "'";
    sql += ", fecha = \""      + cancion.getFecha().toDateString()+ "\";";

    // Ejecuta la sentencia
    actualiza(sql);
  }

  /**
   * Este método permite modificar un renglón de la tabla con los atributos
   * de los objetos de la clase Cancion.
   * @param cancion El usuario a modificar
   * @throws PersistenciaException Si no puede acceder a la base de datos
   */
  public void actualiza(Cancion cancion) throws PersistenciaException {
    String sql = "";

    // Crea la sentencia con la actualización.
    // La sentencia debe terminar en un ;
    sql += "UPDATE "           + nomTabla;
    sql += " SET clave = '"    + cancion.getClave() + "'";
    sql += ", titulo = '"      + cancion.getTitulo() + "'";
    sql += ", cveGenero = '"   + cancion.getGenero().getCveGenero() + "'";
    sql += ", interprete = '"  + cancion.getInterprete() + "'";
    sql += ", autorLetra =  '" + cancion.getAutor() + "'";
    sql += ", album = '"       + cancion.getAlbum() + "'";
    sql += ", duracion = '"    + cancion.getDuracion() + "'";
    sql += ", fecha = \""      + cancion.getFecha().toDateString()+ "\"";
    sql += " WHERE clave = '"  + cancion.getClave() + "';";

    // Ejecuta la sentencia
    actualiza(sql);
  }

  /**
   * Este método permite borrar un renglón de la tabla con los atributos de
```

```
   * los objetos de la clase Cancion.
   * @param cancion Cancion a borrar
   * @throws PersistenciaException Si no puede acceder a la base de datos
   */
  public void elimina(Cancion cancion) throws PersistenciaException {
    String sql = "";

    // Crea la sentencia para borrar.
    // La sentencia debe terminar en un ;
    sql += "DELETE FROM "     + nomTabla;
    sql += " WHERE clave = '" + cancion.getClave() + "';";

    // Ejecuta la sentencia
    actualiza(sql);
  }

  /**
   * Este método permite consultar la tabla con los atributos de los objetos
   * de la clase Cancion usando el comando de consulta dado por el parámetro.
   * @param sql Comando de consulta dado por el parámetro.
   * @return Una lista de los objetos del tipo Cancion de la tabla
   * @throws PersistenciaException Si no puede acceder a la base de datos
   */
  public List<Cancion> lista(String sql) throws PersistenciaException {
    ResultSet renglon;
    List<Cancion> lista = new ArrayList<Cancion>();

    // Ejecuta la consulta
    consulta(sql);

    // Mientras haya canciones en la tabla
    while ( (renglon = obtenRenglon()) != null) {
      try {
      // Crea un objeto del tipo Cancion con los campos del renglon de la
      // tabla.
        Cancion cancion = new Cancion(renglon.getString("clave"),
                                      renglon.getString("titulo"),
                                      new Genero(renglon
                                                   .getString("cveGenero")),
                                      renglon.getString("interprete"),
                                      renglon.getString("autor"),
                                      renglon.getString("album"),
                                      renglon.getInt("duracion"),
                                      new Fecha(renglon.getDate("fecha")));

        // Agrega la película al vector de película
        lista.add(cancion);
      }
      catch (SQLException sqle) {
        throw
          new PersistenciaException("Error al acceder a la base de datos",
                                    sqle);
      }
    }
    return lista;
  }
```

```
  /**
   * Este método permite consultar la tabla con los atributos de los objetos
   * de la clase Cancion.
   * @return Una lista de los objetos del tipo Cancion de la tabla
   * @throws PersistenciaException Si no puede acceder a la base de datos
   */
  public List<Cancion> lista() throws PersistenciaException {
    String sql = "SELECT * FROM " + nomTabla + ";";

    return lista(sql);
  }

  /**
   * Este método permite consultar la tabla con los atributos de los objetos
   * de la clase Cancion y que tienen el mismo título.
   * @param titulo Título de la canción a buscar
   * @return Una lista de los objetos del tipo Cancion de la tabla.
   * @throws PersistenciaException Si no puede acceder a la base de datos
   */
  public List<Cancion> listaTitulo(String titulo)
    throws PersistenciaException {
    String sql = "SELECT * FROM " + nomTabla + " WHERE titulo = '" + titulo
                 + "'";

    return lista(sql);
  }

  ...

  /**
   * Este método permite consultar la tabla con los atributos de los objetos
   * de la clase Cancion y que tienen el mismo genero.
   * @param cveGenero Clave del género de la canción a buscar
   * @return Una lista de los objetos del tipo Cancion de la tabla
   * @throws PersistenciaException Si no puede acceder a la base de datos
   */
  public List<Cancion> listaGenero(String cveGenero)
    throws PersistenciaException {
    String sql = "SELECT * FROM " + nomTabla + " WHERE cveGenero = '"
                 + cveGenero + "'";

    return lista(sql);
  }

  /**
   * Este método permite consultar la tabla con los atributos de los objetos
   * de la clase Cancion y que tienen el mismo periodo.
   * @param periodo Periodo de la canción a buscar
   * @return Una lista de los objetos del tipo Cancion de la tabla
   * @throws PersistenciaException Si no puede acceder a la base de datos
   */
  public List<Cancion> listaPeriodo(Periodo periodo)
    throws PersistenciaException {
    String desde = periodo.getDesde().toDateString();
    String hasta = periodo.getHasta().toDateString();

    String sql = "SELECT * FROM " + nomTabla + " WHERE fecha BETWEEN \"" +
```

```
        desde + "\" AND \"" + hasta + "\";";

    return lista(sql);
  }
}
```

El siguiente es el listado parcial de la clase `Peliculas` que implementa los métodos que nos permiten obtener, consultar, insertar, actualizar y eliminar renglones de la tabla peliculas.
Cada renglón de la tabla peliculas proviene de o se convierte en un objeto del tipo `Pelicula`. Los métodos de esta clase lanzan excepciones del tipo `PersistenciaException`, la cual fue vista en el tema 6: Colecciones.

Peliculas.java

```
/*
 * Peliculas.java
 *
 * Creada el  15 de septiembre de 2007, 12:21 PM
 */

package persistencia;

import java.sql.*;
import java.util.List;
import java.util.ArrayList;
import objetosServicio.*;
import objetosNegocio.*;
import excepciones.PersistenciaException;

/**
 * Esta clase permite agregar, eliminar y consultar películas del programa
 * AmanteMusica en su versión que usa una base de datos
 *
 * @author mdomitsu
 */
public class Peliculas extends Tabla {

  /**
   * Constructor del método. Establece el nombre de la tabla
   * en la que se almacenan los datos
   * @param nomTabla Nombre de la tabla en la que se almacenan los datos
   */
  public Peliculas(String nomTabla) {
    super(nomTabla);
  }

  /**
   * Este método obtiene una película cuya clave es igual a la clave de la
   * película dada por el parámetro.
   * @param pelicula Objeto de tipo Pelicula con la clave de la película a
   * buscar
   * @return La Película si la encuentra, null en caso contrario
   * @throws PersistenciaException Si no puede acceder a la base de datos
   */
```

```
  public Pelicula obten(Pelicula pelicula) throws PersistenciaException {
    ResultSet renglon;

    // Crea la sentencia SQL para buscar una pelicula en la tabla peliculas.
    String sql = "SELECT * FROM "    + nomTabla +
                 " WHERE clave = '" + pelicula.getClave() + "';";

    // Ejecuta la consulta
    consulta(sql);

    // Si hay un renglón en la tabla con el titulo deseado
    if ( (renglon = obtenRenglon()) != null) {
      try {
        // Crea un objeto del tipo Pelicula con los campos del renglón de la
        // tabla.
        Pelicula peliculaHallada = new Pelicula(renglon.getString("clave"),
                                                renglon.getString("titulo"),
                                                new Genero(renglon
                                                    .getString("cveGenero")),
                                                renglon.getString("actor1"),
                                                renglon.getString("actor2"),
                                                renglon
                                                      .getString("director"),
                                                renglon.getInt("duracion"),
                                                new Fecha(renglón
                                                         .getDate("fecha")));
        return peliculaHallada;
      }
      catch(SQLException sqle) {
        throw new
          PersistenciaException("Error al acceder a la base de datos", sqle);
      }
    }
    // Si no, regresa null
    return null;
  }

  /**
   * Este método permite insertar un renglón de la tabla con los atributos de
   * los objetos de la clase Pelicula.
   * @param pelicula Pelicula a insertar en la tabla peliculas
   * @throws PersistenciaException Si no puede acceder a la base de datos
   */
    public void agrega(Pelicula pelicula) throws PersistenciaException {
    String sql = "";

    // Crea la sentencia para insertar.
    // La sentencia debe terminar en un ;
    sql += "INSERT "          + nomTabla;
    sql += " SET clave =   '" + pelicula.getClave() + "'";
    sql += ", titulo = '"     + pelicula.getTitulo() + "'";
    sql += ", cveGenero = '"  + pelicula.getGenero().getCveGenero() + "'";
    sql += ", actor1 = '"     + pelicula.getActor1() + "'";
    sql += ", actor2 = '"     + pelicula.getActor2() + "'";
    sql += ", director = '"   + pelicula.getDirector() + "'";
    sql += ", duracion = '"   + pelicula.getDuracion() + "'";
    sql += ", fecha = \""     + pelicula.getFecha().toDateString()+ "\";";
```

```
    // Ejecuta la sentencia
    actualiza(sql);
  }

  /**
   * Este método permite modificar un renglón de la tabla con los atributos
   * de los objetos de la clase Pelicula.
   * @param pelicula El usuario a modificar
   * @throws PersistenciaException Si no puede acceder a la base de datos
   */
  public void actualiza(Pelicula pelicula) throws PersistenciaException {
    String sql = "";

    // Crea la sentencia con la actualización.
    // La sentencia debe terminar en un ;
    sql += "UPDATE "          + nomTabla;
    sql += " SET clave = '"   + pelicula.getClave() + "'";
    sql += ", titulo = '"     + pelicula.getTitulo() + "'";
    sql += ", cveGenero = '"  + pelicula.getGenero().getCveGenero() + "'";
    sql += ", actor1 =  '"    + pelicula.getActor1() + "'";
    sql += ", actor2 =  '"    + pelicula.getActor2() + "'";
    sql += ", director =  '"  + pelicula.getDirector() + "'";
    sql += ", duracion = '"   + pelicula.getDuracion() + "'";
    sql += ", fecha = \""     + pelicula.getFecha().toDateString()+ "\"";
    sql += " WHERE clave = '" + pelicula.getClave() + "';";

    // Ejecuta la sentencia
    actualiza(sql);
  }

  /**
   * Este método permite borrar un renglón de la tabla con los atributos de
   * los objetos de la clase Pelicula.
   * @param pelicula Película a borrar
   * @throws PersistenciaException Si no puede acceder a la base de datos
   */
  public void elimina(Pelicula pelicula) throws PersistenciaException {
    String sql = "";

    // Crea la sentencia para borrar.
    // La sentencia debe terminar en un ;
    sql += "DELETE FROM "     + nomTabla;
    sql += " WHERE clave = '" + pelicula.getClave() + "';";

    // Ejecuta la sentencia
    actualiza(sql);
  }

  /**
   * Este método permite consultar la tabla con los atributos de los objetos
   * de la clase Pelicula usando el comando de consulta dado por el
   * parámetro.
   * @param sql Comando de consulta dado por el parámetro.
   * @return Una lista de los objetos del tipo Pelicula de la tabla
   * @throws PersistenciaException Si no puede acceder a la base de datos
   */
```

```
  public List<Pelicula> lista(String sql) throws PersistenciaException {
    ResultSet renglon;
    List<Pelicula> lista = new ArrayList<Pelicula>();

    // Ejecuta la consulta
    consulta(sql);

      // Mientras haya películas en la tabla
      while ( (renglon = obtenRenglon()) != null) {
        try {

          // Crea un objeto del tipo Pelicula con los campos del renglón de
          // la tabla.
          Pelicula pelicula = new Pelicula(renglon.getString("clave"),
                                           renglon.getString("titulo"),
                                           new Genero(renglon
                                                    .getString("cveGenero")),
                                           renglon.getString("actor1"),
                                           renglon.getString("actor2"),
                                           renglon.getString("director"),
                                           renglon.getInt("duracion"),
                                           new Fecha(renglón
                                                        .getDate("fecha")));

          // Agrega la película al vector de peliculas
          lista.add(pelicula);
        }
        catch (SQLException sqle) {
          throw new
          PersistenciaException("Error al acceder a la base de datos", sqle);
        }
      }
      return lista;
  }

  /**
   * Este método permite consultar la tabla con los atributos de los objetos
   * de la clase Pelicula.
   * @return Una lista de los objetos del tipo Pelicula de la tabla
   * @throws PersistenciaException Si no puede acceder a la base de datos
   */
  public List<Pelicula> lista() throws PersistenciaException {
    String sql = "SELECT * FROM " + nomTabla + ";";

    return lista(sql);
  }

  ...

  /**
   * Este método permite consultar la tabla con los atributos de los objetos
   * de la clase Pelicula y que tienen el mismo actor.
   * @param actor Actor a buscar
   * @return Una lista de los objetos del tipo Pelicula de la tabla
   * @throws PersistenciaException Si no puede acceder a la base de datos
   */
  public List<Pelicula> listaActor(String actor)
```

```
    throws PersistenciaException {
    String sql = "SELECT * FROM " + nomTabla +
        " WHERE actor1 = '" + actor + "' OR actor2 = '" + actor + "';";

    return lista(sql);
  }

  ...
}
```

El siguiente es el listado de la clase `Generos` implementa los métodos que nos permiten obtener, consultar, insertar, actualizar y eliminar renglones de la tabla generos.

Cada renglón de la tabla generos proviene de o se convierte en un objeto del tipo `Genero`. Los métodos de esta clase lanzan excepciones del tipo `PersistenciaException`, la cual fue vista en el tema 6: Colecciones.

Generos.java

```
/*
 * Generos.java
 *
 * Creada el  28 de agosto de 2008, 10:58 AM
 */

package persistencia;

import java.sql.*;
import java.util.List;
import java.util.ArrayList;
import objetosNegocio.Genero;
import excepciones.PersistenciaException;

/**
 * Esta clase permite agregar, eliminar y consultar generoes del programa
 * AmanteMusica en su versión que usa una base de datos
 *
 * @author mdomitsu
 */
public class Generos extends Tabla {
  /**
   * Constructor del método. Establece el nombre de la tabla
   * en la que se almacenan los datos
   * @param nomTabla Nombre de la tabla en la que se almacenan los datos
   */
  public Generos(String nomTabla) {
    super(nomTabla);
  }

  /**
   * Este método obtiene un género cuya clave es igual a la clave del
   * género dada por el parámetro.
   * @param genero Objeto de tipo Genero con la clave de la género a buscar
   * @return El género si lo encuentra. null en caso contrario.
```

```
   * @throws PersistenciaException Si no puede acceder a la base de datos
   */
  public Genero obten(Genero genero) throws PersistenciaException {
    ResultSet renglon;

    // Crea la sentencia SQL para buscar el género en la tabla géneros.
    String sql = "SELECT * FROM "   + nomTabla +
                 " WHERE cveGenero = '" + genero.getCveGenero() + "';";

    // Ejecuta la consulta
    consulta(sql);

    // Si hay un renglón en la tabla con el género buscado
    if ( (renglon = obtenRenglon()) != null) {
      try {
        // Crea un objeto del tipo Genero con los campos del renglon de la
        // tabla.
        Genero generoHallado = new Genero(renglon.getString("cveGenero"),
                                          renglon.getString("nombre"),
                                          renglon.getString("tipoMedio")
                                                                .charAt(0));
        return generoHallado;
      }
      catch (SQLException sqle) {
        throw new
          PersistenciaException("Error al acceder a la base de datos", sqle);
      }
    }
    // Si no, regresa null
    return null;
  }

  /**
   * Este método permite insertar un renglón de la tabla con los atributos de
   * los objetos de la clase Genero.
   * @param genero Genero a insertar en la tabla generoess
   * @throws PersistenciaException Si no puede acceder a la base de datos
   */
    public void agrega(Genero genero) throws PersistenciaException {
    String sql = "";

    // Crea la sentencia para insertar.
    // La sentencia debe terminar en un ;
    sql += "INSERT "            + nomTabla;
    sql += " SET cveGenero = '" + genero.getCveGenero() + "'";
    sql += ", nombre = '"       + genero.getNombre() + "'";
    sql += ", tipoMedio = '"    + genero.getTipoMedio() + "';";

    // Ejecuta la sentencia
    actualiza(sql);
  }

  /**
   * Este método permite modificar un renglón de la tabla con los atributos
   * de los objetos de la clase Genero.
   * @param genero El usuario a modificar
   * @throws PersistenciaException Si no puede acceder a la base de datos
```

```
   */
  public void actualiza(Genero genero) throws PersistenciaException {
    String sql = "";

    // Crea la sentencia con la actualización.
    // La sentencia debe terminar en un ;
    sql += "UPDATE "            + nomTabla;
    sql += " SET cveGenero = '" + genero.getCveGenero() + "'";
    sql += ", nombre = '"       + genero.getNombre() + "'";
    sql += ", tipoMedio = '"    + genero.getTipoMedio() + "'";
    sql += " WHERE cveGenero = '"  + genero.getCveGenero() + "';";

    // Ejecuta la sentencia
    actualiza(sql);
  }

  /**
   * Este método permite borrar un renglón de la tabla con los atributos de
   * los objetos de la clase Genero.
   * @param genero Genero a borrar
   * @throws PersistenciaException Si no puede acceder a la base de datos
   */
  public void elimina(Genero genero) throws PersistenciaException {
    String sql = "";

    // Crea la sentencia para borrar.
    // La sentencia debe terminar en un ;
    sql += "DELETE FROM "         + nomTabla;
    sql += " WHERE cveGenero = '" + genero.getCveGenero() + "';";

    // Ejecuta la sentencia
    actualiza(sql);
  }

  /**
   * Este método permite consultar la tabla con los atributos de los objetos
   * de la clase Genero usando el comando de consulta dado por el parámetro.
   * @param sql Comando de consulta dado por el parámetro.
   * @return Una lista de los objetos del tipo Genero de la tabla
   * @throws PersistenciaException Si no puede acceder a la base de datos
   */
  public List<Genero> lista(String sql) throws PersistenciaException {
    ResultSet renglon;
    List<Genero> lista = new ArrayList<Genero>();

    // Ejecuta la consulta
    consulta(sql);

    // Mientras haya generoes en la tabla
    while ( (renglon = obtenRenglon()) != null) {
      try {
      // Crea un objeto del tipo Genero con los campos del renglón de la
      // tabla.
        Genero genero = new Genero(renglon.getString("cveGenero"),
                                   renglon.getString("nombre"),
                                   renglon.getString("tipoMedio").charAt(0));
```

```
        // Agrega la película al vector de película
        lista.add(genero);
      }
      catch (SQLException sqle) {
        throw new
          PersistenciaException("Error al acceder a la base de datos", sqle);
      }
    }
    return lista;
  }

  /**
   * Este método permite consultar la tabla con los atributos de los objetos
   * de la clase Genero.
   * @return Una lista de los objetos del tipo Genero de la tabla
   * @throws PersistenciaException Si no puede acceder a la base de datos
   */
  public List<Genero> lista() throws PersistenciaException {
    String sql = "SELECT * FROM " + nomTabla + ";";

    return lista(sql);
  }

  /**
   * Este método permite consultar la tabla con los atributos de los objetos
   * de la clase Genero y que tienen el mismo tipo de medio.
   * @param titulo Título de la género a buscar
   * @return Una lista de los objetos del tipo Genero de la tabla
   * @throws PersistenciaException Si no puede acceder a la base de datos
   */
  public List<Genero> listaMedio(char tipoMedio)
    throws PersistenciaException {
    String sql = "SELECT * FROM " + nomTabla + " WHERE tipoMedio = '"
                 + tipoMedio + "'";

    return lista(sql);
  }
}
```

De la misma manera que se procedió en el Tema 6: Colecciones y Tema 8: Entrada y Salida, por encima de las clases `Canciones`, `Peliculas` y `Generos` que nos permiten el acceso a las tablas de la base de datos musica, se tendrá una clase de fachada `FachadaBD` que nos encapsulará el mecanismo de persistencia. La clase `FachadaBD` implementa la interfaz `IFachada`, como se muestra en la figura 15.4:

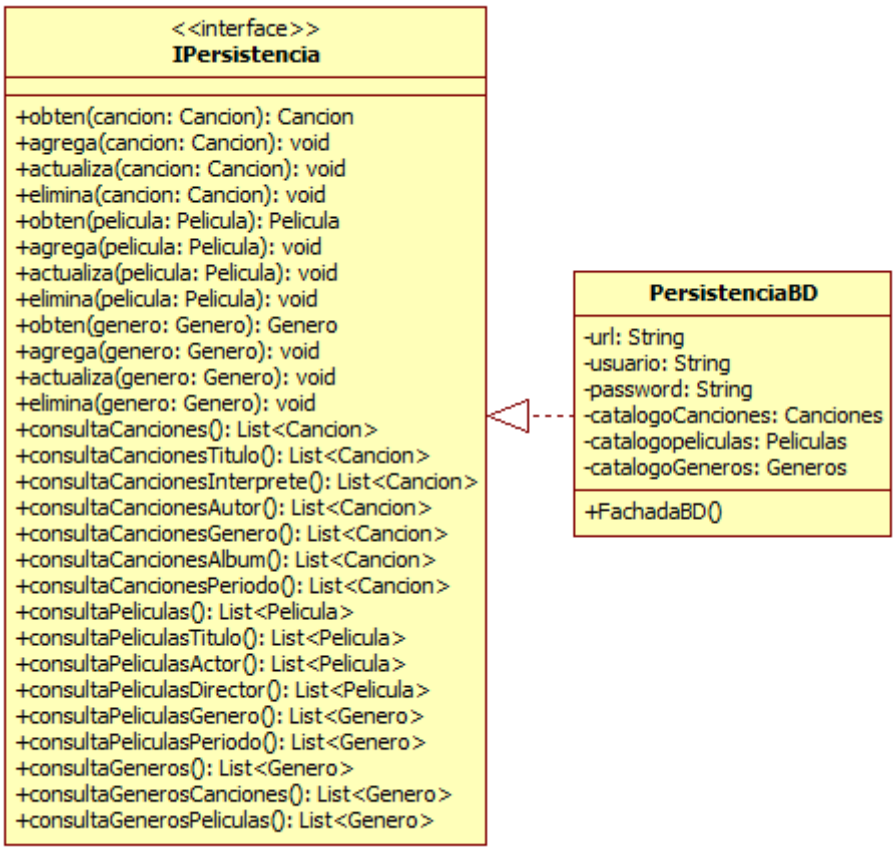

Figura 15.4. Fachada de la Capa de Persistencia del Programa AmanteMusica, Versión Base de Datos

Los métodos de la clase `FachadaBD` lanzan excepciones de la clase `FachadaException` vista en Tema 6: Colecciones. El código parcial de la clase `FachadaBD` es:

FachadaBD.java

```
/*
 * FachadaBD.java
 *
 * Creada el 15 de septiembre de 2007, 12:21 PM
 */

package fachadas;

import java.sql.*;
import java.util.List;
import java.util.ArrayList;
import objetosServicio.*;
import objetosNegocio.*;
import persistencia.*;
import interfaces.IFachada;
import excepciones.*;
```

```
/**
 * Esta clase implementa la interfaz IFachada del mecanismo de persistencia
 * de base de datos del programa AmanteMusica
 *
 * @author mdomitsu
 */
public class FachadaBD implements IFachada {
  // URL de la base de datos. Sintaxis:
  //   "jdbc:mysql://dir IP del servidor/base de datos"
  // Si la base de datos está en la máquina local use localhost en lugar de
  // la dirección IP del servidor
  private static final String url = "jdbc:mysql://localhost/musica";
  // Nombre del usuario que tiene acceso a la base de datos
  private static final String usuario = "root";
  // Password del usuario
  private static final String password = "sesamo";

  private Canciones catalogoCanciones;
  private Peliculas catalogoPeliculas;
  private Generos catalogoGeneros;

  /**
   * Constructor predeterminado
   */
  public FachadaBD() {
    // Crea un objeto del tipo catalogoCanciones para accesar a la tabla
    // canciones
    catalogoCanciones = new Canciones("canciones");

    // Crea un objeto del tipo catalogoPeliculas para accesar a la tabla
    // peliculas
    catalogoPeliculas = new Peliculas("peliculas");

    // Crea un objeto del tipo catalogoGeneros para accesar a la tabla
    // generos
    catalogoGeneros = new Generos("generos");
  }

  /**
   * Obtiene una canción de la tabla canciones
   * @param cancion Cancion a obtener
   * @return La canción si existe, null en caso contrario
   * @throws FachadaException Si no se puede obtener la canción.
   */
  public Cancion obten(Cancion cancion) throws FachadaException {
    Conexion conexion = null;
    Cancion cancionHallada;
    Genero generoHallado;

    try {
      // Establece la conexion con la base de datos
      conexion = new Conexion(url, usuario, password);
      Connection conn = conexion.getConexion();
      catalogoCanciones.setConexion(conn);
      catalogoGeneros.setConexion(conn);
```

```
      // Obten la canción
      cancionHallada = catalogoCanciones.obten(cancion);

      if(cancionHallada != null) {
        // Obten el género de la canción
        generoHallado = catalogoGeneros.obten(cancionHallada.getGenero());

        // Se agrega el género a la canción
        cancionHallada.setGenero(generoHallado);
      }
      // Regresa la canción
      return cancionHallada;
    }
    catch (PersistenciaException pe) {
      throw new FachadaException("No se puede obtener la canción", pe);
    }
    finally {
      // Cierra la conexión
      try {
        if(conexion != null) conexion.close();
      }
      catch (PersistenciaException pe) {
        throw new FachadaException("No se puede obtener la canción", pe);
      }
    }
  }

  /**
   * Agrega una canción a la tabla canciones. No se permiten canciones
   * con claves repetidas
   * @param cancion Cancion a agregar
   * @throws FachadaException Si la canción está repetida o no se puede
   * agregar la canción a la tabla canciones.
   */
  public void agrega(Cancion cancion) throws FachadaException {
    Conexion conexion = null;

    try {
      // Establece la conexion con la base de datos
      conexion = new Conexion(url, usuario, password);
      Connection conn = conexion.getConexion();
      catalogoCanciones.setConexion(conn);

      // Busca la canción en la tabla canciones
      Cancion cancionBuscada = catalogoCanciones.obten(cancion);

      // Si la hay, no se agrega a la tabla
      if(cancionBuscada != null)
        throw new FachadaException("Canción repetida");

      // Agrega la nueva canción a la tabla
      catalogoCanciones.agrega(cancion);
    }
    catch (PersistenciaException pe) {
      throw new FachadaException("No se puede agregar la canción", pe);
    }
    finally {
```

```
      // Cierra la conexión
      try {
        if(conexion != null) conexion.close();
      }
      catch (PersistenciaException pe) {
        throw new FachadaException("No se puede agregar la canción", pe);
      }
    }
  }

  /**
   * Actualiza una canción de la tabla canciones
   * @param cancion Cancion a actualizar
   * @throws FachadaException Si no se puede actualizar la canción de la
   * tabla canciones.
   */
  public void actualiza(Cancion cancion) throws FachadaException {
    Conexion conexion = null;

    try {
      // Establece la conexion con la base de datos
      conexion = new Conexion(url, usuario, password);
      Connection conn = conexion.getConexion();
      catalogoCanciones.setConexion(conn);

      // Actualiza la  canción de la tabla canciones
      catalogoCanciones.actualiza(cancion);
    }
    catch (PersistenciaException pe) {
      throw new FachadaException("No se puede actualizar la canción", pe);
    }
    finally {
      // Cierra la conexión
      try {
        if(conexion != null) conexion.close();
      }
      catch (PersistenciaException pe) {
        throw new FachadaException("No se puede actualizar la canción", pe);
      }
    }
  }

  /**
   * Elimina una canción de la tabla canciones
   * @param cancion Cancion a eliminar
   * @throws FachadaException Si no se puede eliminar la canción de la
   * tabla canciones.
   */
  public void elimina(Cancion cancion) throws FachadaException {
    Conexion conexion = null;

    try {
      // Establece la conexion con la base de datos
      conexion = new Conexion(url, usuario, password);
      Connection conn = conexion.getConexion();
      catalogoCanciones.setConexion(conn);
```

```
      // Elimina la  canción de la tabla canciones
        catalogoCanciones.elimina(cancion);
    }
    catch (PersistenciaException pe) {
      throw new FachadaException("No se puede eliminar la canción", pe);
    }
    finally {
      // Cierra la conexión
      try {
        if(conexion != null) conexion.close();
      }
      catch (PersistenciaException pe) {
        throw new FachadaException("No se puede eliminar la canción", pe);
      }
    }
  }

  /**
   * Obtiene una película de la tabla peliculas
   * @param pelicula Pelicula a obtener
   * @return La película si existe, null en caso contrario
   * @throws FachadaException Si no se puede obtener la película de la tabla
   * peliculas.
   */
  public Pelicula obten(Pelicula pelicula) throws FachadaException {
    Conexion conexion = null;
    Pelicula peliculaHallada;
    Genero generoHallado;

    try {
      // Establece la conexion con la base de datos
      conexion = new Conexion(url, usuario, password);
      Connection conn = conexion.getConexion();
      catalogoPeliculas.setConexion(conn);
      catalogoGeneros.setConexion(conn);

      // Obten la película
      peliculaHallada = catalogoPeliculas.obten(pelicula);

      if (peliculaHallada != null) {
        // Obten el género de la canción
        generoHallado = catalogoGeneros.obten(peliculaHallada.getGenero());

        // Se agrega el género a la película
        peliculaHallada.setGenero(generoHallado);
      }

      // Regresa la película
      return peliculaHallada;
    }
    catch (PersistenciaException pe) {
      throw new FachadaException("No se puede obtener la película", pe);
    }
    finally {
      // Cierra la conexión
      try {
        if(conexion != null) conexion.close();
```

```
      }
      catch (PersistenciaException pe) {
        throw new FachadaException("No se puede obtener la película", pe);
      }
    }
  }

  /**
   * Agrega una película a la tabla peliculas. No se permiten peliculas
   * con claves repetidas
   * @param pelicula Película a agregar
   * @throws FachadaException Si la película está repetida o no se puede
   * agregar la película a la tabla peliculas.
   */
  public void agrega(Pelicula pelicula) throws FachadaException {
    Conexion conexion = null;

    try {
      // Establece la conexion con la base de datos
      conexion = new Conexion(url, usuario, password);
      Connection conn = conexion.getConexion();
      catalogoPeliculas.setConexion(conn);

      // Busca la película en la tabla peliculas
      Pelicula peliculaBuscada = catalogoPeliculas.obten(pelicula);

      // Si lo hay, no se agrega a la tabla
      if(peliculaBuscada != null)
        throw new FachadaException("Película repetida");

      // Agrega la nueva película a la tabla
      catalogoPeliculas.agrega(pelicula);
    }
    catch (PersistenciaException pe) {
      throw new FachadaException("No se puede agregar la película", pe);
    }
    finally {
      // Cierra la conexión
      try {
        if(conexion != null) conexion.close();
      }
      catch (PersistenciaException pe) {
        throw new FachadaException("No se puede agregar la película", pe);
      }
    }
  }

  /**
   * Actualiza una película de la tabla peliculas
   * @param pelicula Pelicula a actualizar
   * @throws FachadaException Si no se puede actualizar la película de la
   * tabla películas.
   */
  public void actualiza(Pelicula pelicula) throws FachadaException {
    Conexion conexion = null;

    try {
```

```
      // Establece la conexion con la base de datos
      conexion = new Conexion(url, usuario, password);
      Connection conn = conexion.getConexion();
      catalogoPeliculas.setConexion(conn);

      // Actualiza la  película de la tabla
        catalogoPeliculas.actualiza(pelicula);
    }
    catch (PersistenciaException pe) {
      throw new FachadaException("No se puede actualizar la película", pe);
    }
    finally {
      // Cierra la conexión
      try {
        if(conexion != null) conexion.close();
      }
      catch (PersistenciaException pe) {
        throw new FachadaException("No se puede actualizar la película", pe);
      }
    }
  }

  /**
   * Elimina una película de la Tabla peliculas
   * @param pelicula Pelicula a eliminar
   * @throws FachadaException Si no se puede eliminar la película de la
   * tabla películas.
   */
  public void elimina(Pelicula pelicula) throws FachadaException {
    Conexion conexion = null;

    try {
      // Establece la conexion con la base de datos
      conexion = new Conexion(url, usuario, password);
      Connection conn = conexion.getConexion();
      catalogoPeliculas.setConexion(conn);

      // Elimina la  película del catálogo
      catalogoPeliculas.elimina(pelicula);
    }
    catch (PersistenciaException pe) {
      throw new FachadaException("No se puede eliminar la película", pe);
    }
    finally {
      // Cierra la conexión
      try {
        if(conexion != null) conexion.close();
      }
      catch (PersistenciaException pe) {
        throw new FachadaException("No se puede eliminar la película", pe);
      }
    }
  }

  /**
   * Obtiene un género de la tabla generos
   * @param genero Género a obtener
```

```
   * @return El género si existe, null en caso contrario
   * @throws FachadaException Si no se puede obtener el género.
   */
  public Genero obten(Genero genero) throws FachadaException {
    Conexion conexion = null;

    try {
      // Establece la conexion con la base de datos
      conexion = new Conexion(url, usuario, password);
      Connection conn = conexion.getConexion();
      catalogoGeneros.setConexion(conn);

      // Obten el género
      return catalogoGeneros.obten(genero);
    }
    catch (PersistenciaException pe) {
      throw new FachadaException("No se puede obtener el género", pe);
    }
    finally {
      // Cierra la conexión
      try {
        if(conexion != null) conexion.close();
      }
      catch (PersistenciaException pe) {
        throw new FachadaException("No se puede obtener el género", pe);
      }
    }
  }

  /**
   * Agrega un género a la tabla géneros. No se permiten géneros
   * con claves repetidas
   * @param genero Género a agregar
   * @throws FachadaException Si el género está repetido o no se puede
   * agregar el género a la tabla generos.
   */
  public void agrega(Genero genero) throws FachadaException {
    Conexion conexion = null;

    try {
      // Establece la conexion con la base de datos
      conexion = new Conexion(url, usuario, password);
      Connection conn = conexion.getConexion();
      catalogoGeneros.setConexion(conn);

      // Busca el género en la tabla generos
      Genero generoBuscado = catalogoGeneros.obten(genero);

      // Si lo hay, no se agrega a la tabla
      if(generoBuscado != null) throw new FachadaException("Género
repetido");

      // Agrega el nuevo género a la tabla
      catalogoGeneros.agrega(genero);
    }
    catch (PersistenciaException pe) {
      throw new FachadaException("No se puede agregar el género", pe);
```

```
    }
    finally {
      // Cierra la conexión
      try {
        if(conexion != null) conexion.close();
      }
      catch (PersistenciaException pe) {
        throw new FachadaException("No se puede agregar el género", pe);
      }
    }
  }

  /**
   * Actualiza un género de la tabla géneros
   * @param genero Género a actualizar
   * @throws FachadaException Si no se puede actualizar el género de la
   * tabla generos.
   */
  public void actualiza(Genero genero) throws FachadaException {
    Conexion conexion = null;

    try {
      // Establece la conexion con la base de datos
      conexion = new Conexion(url, usuario, password);
      Connection conn = conexion.getConexion();
      catalogoGeneros.setConexion(conn);

      // Actualiza el género de la tabla generos
      catalogoGeneros.actualiza(genero);
    }
    catch (PersistenciaException pe) {
      throw new FachadaException("No se puede actualizar el género", pe);
    }
    finally {
      // Cierra la conexión
      try {
        if(conexion != null) conexion.close();
      }
      catch (PersistenciaException pe) {
        throw new FachadaException("No se puede actualizar el género", pe);
      }
    }
  }

  /**
   * Elimina un género de la tabla generos
   * @param genero Género a eliminar
   * @throws FachadaException Si no se puede eliminar el género de la
   * tabla generos.
   */
  public void elimina(Genero genero) throws FachadaException {
    Conexion conexion = null;

    try {
      // Establece la conexion con la base de datos
      conexion = new Conexion(url, usuario, password);
      Connection conn = conexion.getConexion();
```

```
      catalogoGeneros.setConexion(conn);

      // Elimina el género de la tabla generos
        catalogoGeneros.elimina(genero);
    }
    catch (PersistenciaException pe) {
      throw new FachadaException("No se puede eliminar el género", pe);
    }
    finally {
      // Cierra la conexión
      try {
        if(conexion != null) conexion.close();
      }
      catch (PersistenciaException pe) {
        throw new FachadaException("No se puede eliminar el género", pe);
      }
    }
  }

  /**
   * Obtiene una lista todas las canciones
   * @return Una lista de todas las canciones
   * @throws FachadaException Si no se puede obtener la lista de canciones
   */
  public List<Cancion> consultaCanciones() throws FachadaException {
    Conexion conexion = null;

    try {
      // Establece la conexion con la base de datos
      conexion = new Conexion(url, usuario, password);
      Connection conn = conexion.getConexion();
      catalogoCanciones.setConexion(conn);

      // Regresa la lista de canciones
      return agregaGeneroCanciones(catalogoCanciones.lista());
    }
    catch (PersistenciaException pe) {
      throw new
           FachadaException("No se puede obtener la lista de canciones", pe);
    }
    finally {
      // Cierra la conexión
      try {
        if(conexion != null) conexion.close();
      }
      catch (PersistenciaException pe) {
        throw new
           FachadaException("No se puede obtener la lista de canciones", pe);
      }
    }
  }

  /**
   * Le agrega los atributos del género a cada canción de la lista
   * @param listaCanciones Lista de las canciones a las que se les
   * agregará los atributos del género
   * @return Una lista de canciones
```

```
   * @throws FachadaException Si hay un problema al conectarse a la
   * base de datos
   */
  private List<Cancion> agregaGeneroCanciones(List<Cancion> listaCanciones)
          throws FachadaException  {
      Genero genero;
      Cancion cancion;
      Conexion conexion = null;

      try {
        // Establece la conexion con la base de datos
        conexion = new Conexion(url, usuario, password);
        Connection conn = conexion.getConexion();
        catalogoGeneros.setConexion(conn);

        // Para cada canción de la lista
        for (int i = 0; i < listaCanciones.size(); i++) {
          // Obtén la canción de la lista
          cancion = (Cancion) listaCanciones.get(i);

          // Obten el género de la canción del catálogo de géneros
          genero = catalogoGeneros.obten(cancion.getGenero());

          // Agrega el género a la canción
          cancion.setGenero(genero);
          listaCanciones.set(i, cancion);
        }

        // Regresa la lista canciones
        return listaCanciones;
      }
      catch (PersistenciaException pe) {
        throw new
           FachadaException("No se puede obtener la lista de canciones", pe);
      }
      finally {
        // Cierra la conexión
        try {
          if(conexion != null) conexion.close();
        }
        catch (PersistenciaException pe) {
          throw new
           FachadaException("No se puede obtener la lista de canciones", pe);
        }
      }
  }

  /**
   * Obtiene una lista de todas las canciones con el mismo título.
   * @param titulo Titulo de las canciones de la lista
   * @return Una lista de todas las canciones con el mismo
   * título.
   * @throws FachadaException Si no se puede obtener la lista de canciones
   */
  public List<Cancion> consultaCancionesTitulo(String titulo)
    throws FachadaException {
    Conexion conexion = null;
```

```
    try {
      // Establece la conexion con la base de datos
      conexion = new Conexion(url, usuario, password);
      Connection conn = conexion.getConexion();
      catalogoCanciones.setConexion(conn);

      // Regresa la lista de canciones
      return agregaGeneroCanciones(catalogoCanciones.listaTitulo(titulo));
    }
    catch (PersistenciaException pe) {
      throw new
           FachadaException("No se puede obtener la lista de canciones", pe);
    }
    finally {
      // Cierra la conexión
      try {
        if(conexion != null) conexion.close();
      }
      catch (PersistenciaException pe) {
        throw new
           FachadaException("No se puede obtener la lista de canciones", pe);
      }
    }
  }

  ...

  /**
   * Obtiene una lista de los géneros de canciones
   * @return Una lista de los géneros canciones
   * @throws FachadaException Si no se puede obtener la lista de géneros
   */
  public List<Generos> consultaGenerosCanciones() throws FachadaException {
    Conexion conexion = null;

    try {
      // Establece la conexion con la base de datos
      conexion = new Conexion(url, usuario, password);
      Connection conn = conexion.getConexion();
      catalogoGeneros.setConexion(conn);

      // Regresa la lista de géneros de canciones
      return catalogoGeneros.listaMedio('C');
    }
    catch (PersistenciaException pe) {
      throw new
             FachadaException("No se puede obtener la lista de géneros", pe);
    }
    finally {
      // Cierra la conexión
      try {
        if(conexion != null) conexion.close();
      }
      catch (PersistenciaException pe) {
        throw new
             FachadaException("No se puede obtener la lista de géneros", pe);
```

```
      }
    }
  }

  /**
   * Obtiene una lista de los géneros de películas
   * @return Una lista de los géneros películas
   * @throws FachadaException Si no se puede acceder al catálogo
   * de géneros.
   */
  public List<Genero> consultaGenerosPeliculas() throws FachadaException {
    Conexion conexion = null;

    try {
      // Establece la conexion con la base de datos
      conexion = new Conexion(url, usuario, password);
      Connection conn = conexion.getConexion();
      catalogoGeneros.setConexion(conn);

      // Regresa la lista de géneros de canciones
      return catalogoGeneros.listaMedio('P');
    }
    catch (PersistenciaException pe) {
      throw new
             FachadaException("No se puede obtener la lista de géneros", pe);
    }
    finally {
      // Cierra la conexión
      try {
        if(conexion != null) conexion.close();
      }
      catch (PersistenciaException pe) {
        throw new
             FachadaException("No se puede obtener la lista de géneros", pe);
      }
    }
  }
}
```

Para probar la clase `FachadaBD` y las clases `Canciones` y `Peliculas` en su versión que encapsulan las tablas de base de datos podemos usar la clase de prueba Prueba4 vista en el Tema 8: Entrada y Salida modificando sólo dos líneas:

- Cambiar la línea:

  ```
  import fachadas.FachadaArchivos;
  ```

  por ésta:

  ```
  import fachadas.FachadaBD;
  ```

- Cambiar la línea:

  ```
  IFachada fachada = new FachadaArchivos();
  ```

por ésta:

```
IFachada fachada = new FachadaBD();
```

Estos dos cambios son también los únicos que hay que hacerle a la clase de control Control vista en el Tema 7: Desarrollo de Aplicaciones para cambiar el mecanismo de persistencia, obviamente también es necesario sustituir la clase de fachada y las clases `Canciones` y `Peliculas` a sus versiones de base de datos.