Tema 8

Desarrollo de Aplicaciones en Java

En el análisis y Diseño Orientado a objetos, la codificación de una aplicación parte de los diagramas de clase – diseño y los diagramas de secuencia. Los diagramas de clase describen los componentes individuales, objetos, a partir de los cuales se construirá la aplicación y su relación entre ellos. Sin embargo, los diagramas de clase no establecen como los diferentes componentes interactúan entre sí para lograr las funcionalidades de la aplicación, quienes realizan esta tarea son los diagramas de secuencia.

Un diagrama de secuencia establece qué tareas y en qué orden deben realizar los componentes de una aplicación para realizar un caso de uso. Cada componente sabe realizar un conjunto de tareas. Esas tareas están descritas por el código de los métodos de su respectiva clase. Para que un componente realice una tarea, su correspondiente método debe ser invocado. Esa invocación puede ser hecha por otro componente o por sí mismo. Por lo tanto, un diagrama de secuencia describe el orden en que ocurren las invocaciones de los métodos de los diferentes objetos que participan en la realización de un caso de uso.

Un diagrama de secuencia puede codificarse mediante un método, cada una de las sentencias de dicho método representa una invocación del diagrama de secuencia. Esos métodos que implementan las realizaciones de los casos de uso se agrupan en una o más clases llamadas clases de control.

Ejemplo sobre una Aplicación con una Interfaz de Usuario Gráfica

Por ejemplo, suponga que la aplicación Amante Música, debe tener las funcionalidades descritas en el diagrama de casos de uso de la figura 8.1.

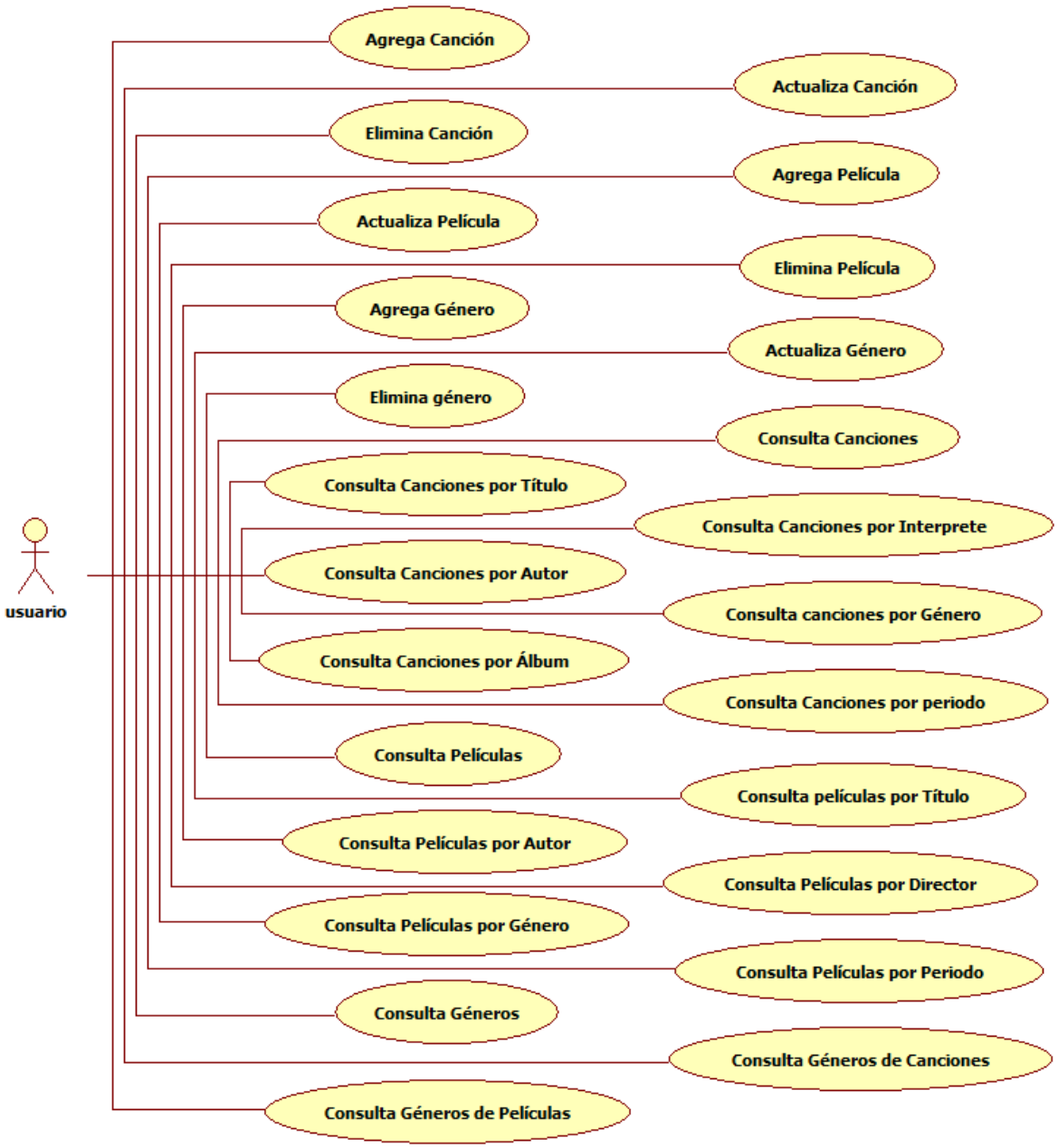

Figura 8.1 Diagramas de Caso de Uso de la aplicación Amante Música

El caso de uso **Agregar Canción** nos permite agregar una canción al catálogo de canciones. El diagrama de secuencia de la figura 8.2 realiza dicho caso de uso.

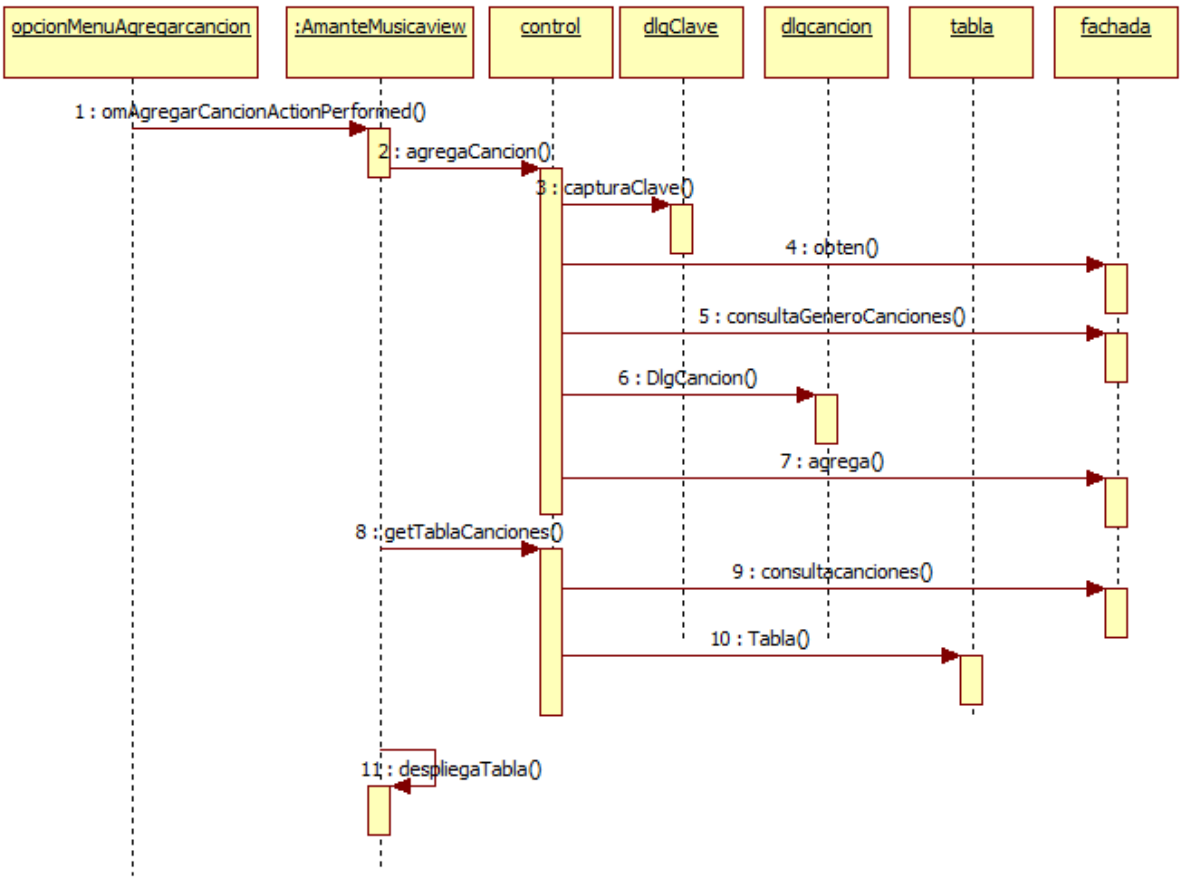

Figura 8.2 Diagrama de secuencia – Agregar canción

El caso de uso empieza cuando el usuario selecciona del menú de la ventana de la aplicación la opción Agregar Canción, haciendo que la opción del menú invoque al método oyente `opcionMenuAgregarCancionActionPerformed()`, cuyo código se muestra en el siguiente fragmento de código:

FrmAmanteMusica.java (Fragmento)

```
/*
 * FrmAmanteMusica.java
 */
package interfazUsuario;

import control.Control;
import control.Tabla;
import java.awt.Dimension;
import java.awt.Toolkit;

/**
 * Esta es clase es la clase principal de la aplicación Amante musica.
También
 * es la ventana principal de la aplicacion
 *
 * @author Manuel Domitsu
```

```
 */
public class FrmAmanteMusica extends javax.swing.JFrame {
  ...

  /**
   * Metodo oyente que agrega una cancion al catalogo de canciones
   *
   * @param evt Evento al que escucha
   */
  private void
    opcionMenuAgregarCancionActionPerformed(java.awt.event.ActionEvent evt) {
    // Agrega la nueva canción
    if (control.agregaCancion(this)) {

      // Obtiene la lista de canciones
      Tabla tablaCanciones = control.getTablaCanciones(this);

      // Despliega la lista de canciones
      despliegaTabla(tablaCanciones);
    }
  }                                                          ...

    private javax.swing.JTable jtabla;

    Control control = new Control();
}
```

El método oyente `opcionMenuAgregarCancionActionPerformed()` primero invoca al método `agregaCanción()` de la clase **Control** que captura la clave de la canción a agregar, verifica que la canción no existe en el catálogo de canciones, captura el resto de los datos de la canción y guarda la canción en el catálogo de canciones. A continuación el método oyente invoca al método `getTablaCanciones()` que obtiene la tabla de canciones y por último el método oyente invoca al método `despliegaTabla()` para que despliegue la lista de canciones.

La clase Control se muestra en el diagrama de clases de la figura 8.3.

En el fragmento de código siguiente se muestra el método `agregaCanción()` encargado de agregar la canción al catálogo de canciones y el método `getTablaCanciones()` que obtiene la lista de canciones del catálogo de canciones.

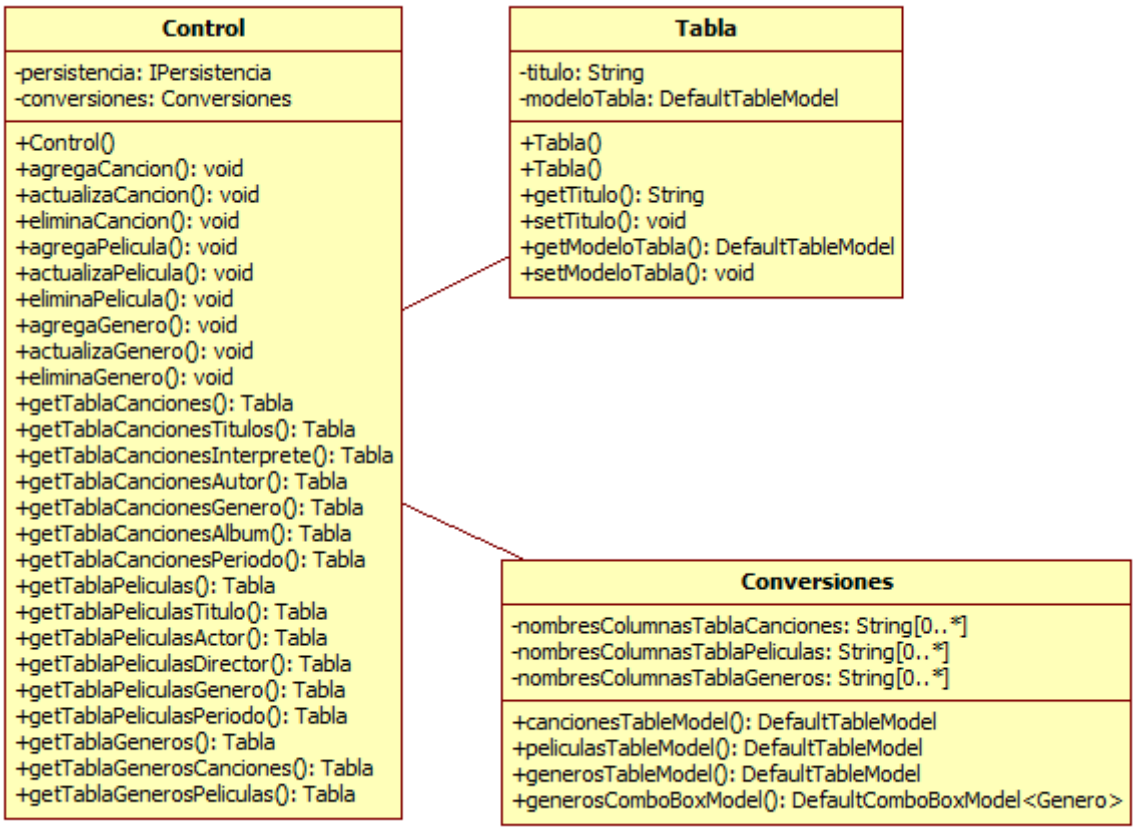

Figura 8.3 Diagrama de clases del paquete control

Control.java (Fragmento)

```
/*
 * Control.java
 */
package control;

import interfaces.IPersistencia;
import interfazUsuario.*;
import java.util.List;
import javax.swing.DefaultComboBoxModel;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import objetosNegocio.Cancion;
import objetosNegocio.Genero;
import objetosNegocio.Pelicula;
import objetosServicio.Periodo;
import persistencia.PersistenciaBD;

/**
 * Esta clase implementa los casos de uso de la aplicacion Amante Musica.
 * @author mdomitsu
 */
public class Control {
  // Acceso a los objetos del negocio
```

```
  IPersistencia persistencia;
  Conversiones conversiones;

  /**
   * Constructor.
   */
  public Control() {
    // Crea un objeto del tipo persistencia
    persistencia = new PersistenciaBD();

    conversiones = new Conversiones();
  }

  /**
   * Agrega una cancion al catalogo de canciones
   * @param frame Ventana sobre la que se despliega el cuadro de dialogo para
   *              capturar los datos de la cancion
   * @return Regresa true si se pudo agregar la cancion, false en caso
   * contrario
   */
  public boolean agregaCancion(JFrame frame) {
    Cancion cancion, bCancion;
    StringBuffer respuesta = new StringBuffer("");
    DlgCancion dlgCancion;
    List<Genero> listaGenerosCanciones;
    DefaultComboBoxModel<Genero> generosCancionesComboBoxModel;

    // Captura la clave de la cancion
    String clave = JOptionPane.showInputDialog(frame, "Clave de la cancion:",
                                               "Agregar cancion",
                                               JOptionPane.QUESTION_MESSAGE);

    // Si el usuario presiono el boton Cancelar
    if(clave == null) return false;

    // Crea un objeto Cancion con solo la clave
    cancion = new Cancion(clave);

    try {
      // Obten la cancion del catalogo de canciones
      bCancion = persistencia.obten(cancion);

      // Obtiene la listade generos de canciones
      listaGenerosCanciones = persistencia.consultaGenerosCanciones();

    }
    catch (Exception e) {
      // Si ocurrio un error al leer del catalogo de canciones,
      // despliega mensaje de error
      JOptionPane.showMessageDialog(frame, e.getMessage(), "Error!!.",
                                    JOptionPane.ERROR_MESSAGE);
      return false;
    }

    generosCancionesComboBoxModel = conversiones.
            generosComboBoxModel(listaGenerosCanciones);
```

```
    // Si la cancion existe, despliega sus datos
    if(bCancion != null) {
        dlgCancion = new DlgCancion(frame,
                                    "La cancion ya esta en el catalogo",
                                    true, bCancion,
                                    generosCancionesComboBoxModel,
                                    ConstantesGUI.DESPLEGAR, respuesta);

      return false;
    }

    // Si la cancion no existe captura los datos de la nueva cancion
    dlgCancion = new DlgCancion(frame, "Captura Datos Cancion", true,
                                cancion, generosCancionesComboBoxModel,
                                ConstantesGUI.AGREGAR, respuesta);

    // Si el usuario presiono el boton Cancelar
    if (respuesta.substring(0).equals(ConstantesGUI.CANCELAR)) return false;

    // Agrega la nueva cancion al catalogo de canciones
    try {
      persistencia.agrega(cancion);
    }
    catch(Exception e)  {
      // Si ocurrio un error al escribir al catalogo de canciones,
      // despliega mensaje de error
      JOptionPane.showMessageDialog(frame, e.getMessage(), "Error!!.",
                                    JOptionPane.ERROR_MESSAGE);
      return false;
    }
    return true;
  }
  ...

  /**
   * Regresa un objeto Tabla con todas las canciones
   * @param frame Ventana sobre la que se despliega el mensaje de error
   * @return Objeto Tabla con todas las canciones, null si hay un error
   */
  public Tabla getTablaCanciones(JFrame frame) {
    List<Cancion> listaCanciones;

    try {
      // Obtiene la lista de canciones
      listaCanciones = persistencia.consultaCanciones();
    }
    catch (Exception e) {
      // Si ocurrio un error al obtener la lista de la base de datos,
      // despliega mensaje de error
      JOptionPane.showMessageDialog(frame, e.getMessage(), "Error!!.",
                                    JOptionPane.ERROR_MESSAGE);
      return null;
    }

    // Regresa el objeto Tabla con todas las canciones
    return new Tabla("Lista de Canciones",
                     conversiones.cancionesTableModel(listaCanciones));
```

```
  }
  ...
}
```

La clase `Tabla`, cuyo código se muestra enseguida, encapsula los datos de una tabla: el título de la tabla, y un objeto de tipo `DefaultTableModel` con los títulos de las columnas de la tabla y los valores de la tabla.

Tabla.java

```
/*
 * Tabla.java
 */

package control;

import javax.swing.table.DefaultTableModel;

/**
 * Esta clase encapsula el titulo de una tabla y un objeto TableModel con
 * los datos de una tabla que seran desplegados en una tabla del tipo JTable
 *
 * @author mdomitsu
 */
public class Tabla {
    private String titulo;
    private DefaultTableModel modeloTabla;

    /**
     * Constructor sin parametros
     */
    public Tabla() {
    }

    /**
     * Constructor que inicializa los atributos de la clase
     *
     * @param titulo Titulo de la tabla
     * @param modeloTabla Objeto TableModel con los datos de la tabla.
     */
    public Tabla(String titulo, DefaultTableModel modeloTabla) {
        this.titulo = titulo;
        this.modeloTabla = modeloTabla;
     }

    /**
     * Regresa el titulo de la tabla
     * @return El titulo de la tabla
     */
    public String getTitulo() {
        return titulo;
    }

    /**
     * Establece el titulo de la tabla
     * @param titulo Titulo de la tabla
```

```
     */
    public void setTitulo(String titulo) {
        this.titulo = titulo;
    }

    /**
     * Regresa los datos de la tabla
     * @return Objeto TableModel con los datos de la tabla
     */
    public DefaultTableModel getModeloTabla() {
        return modeloTabla;
    }

    /**
     * Establece los datos de la tabla
     * @param modeloTabla Objeto TableModel con los datos de la tabla
     */
    public void setModeloTabla(DefaultTableModel modeloTabla) {
        this.modeloTabla = modeloTabla;
    }
}
```

La clase `Conversiones` contiene métodos que encapsulan listas de canciones, películas o géneros en objetos del tipo `DefaultTableModel` usadas por el componente`JTable` para desplegar tablas. También contiene un método que encapsula una lista de géneros en un objeto del tipo `DefaultComboBoxModel` empleado por el componente `JComboBox` para desplegar la lista de opciones. El código de la clase `Conversiones` es el siguiente.

Conversiones.java

```
/*
 * Conversiones.java
 *
 */
package control;

import java.util.List;
import javax.swing.DefaultComboBoxModel;
import javax.swing.table.DefaultTableModel;
import objetosNegocio.Cancion;
import objetosNegocio.Genero;
import objetosNegocio.Pelicula;

/**
 * Esta clase contiene métodos que generan objetos del tipo DefaultTableModel
 * y DefaultComboBoxModel para crear instancias de Jtable y JComboBox.
 * @author mdomitsu
 */
public class Conversiones {
  // Arreglos con los nombres de las columnas de las tablas

  String nombresColumnasTablasCanciones[] = {"Clave", "Titulo", "Interprete",
                                             "Autor", "Genero", "Album",
                                             "Duracion", "Fecha"};
```

```
  String nombresColumnasTablasPeliculas[] = {"Clave", "Titulo", "Actor1",
                                             "Actor2", "Director", "Genero",
                                             "Duracion", "Fecha"};
  String nombresColumnasTablasGeneros[] = {"Clave", "Nombre", "Tipo"};

  /**
   * Genera un objeto de tipo DefaultTableModel a partir de una lista de
   * canciones.
   *
   * @param listaCanciones Lista de canciones a convertir
   * @return Objeto de tipo DefaultTableModel con los atributos de las
   * canciones.
   */
  public DefaultTableModel cancionesTableModel(List<Cancion> listaCanciones){
    Object tabla[][];

    if (listaCanciones != null) {
      tabla = new Object[listaCanciones.size()][8];
      for (int i = 0; i < listaCanciones.size(); i++) {
        // Obten una canción de la lista de canciones
        Cancion cancion = listaCanciones.get(i);
        // Almacena sus atributos en la fila del arreglo
        tabla[i][0] = cancion.getClave();
        tabla[i][1] = cancion.getTitulo();
        tabla[i][2] = cancion.getInterprete();
        tabla[i][3] = cancion.getAutor();
        tabla[i][4] = cancion.getGenero().getNombre();
        tabla[i][5] = cancion.getAlbum();
        tabla[i][6] = new Integer(cancion.getDuracion());
        tabla[i][7] = cancion.getFecha();
      }
      return new DefaultTableModel(tabla, nombresColumnasTablasCanciones);
    }

    return null;
  }

  /**
   * Genera un objeto de tipo DefaultTableModel a partir de una lista de
   * peliculas.
   *
   * @param listaPeliculas Lista de peliculas a convertir
   * @return Objeto de tipo DefaultTableModel con los atributos de las
   * peliculas.
   */
 public DefaultTableModel peliculasTableModel(List<Pelicula> listaPeliculas){
    Object tabla[][];

    if (listaPeliculas != null) {
      tabla = new Object[listaPeliculas.size()][8];
      for (int i = 0; i < listaPeliculas.size(); i++) {
        // Obten una canción de la lista de canciones
        Pelicula pelicula = listaPeliculas.get(i);
        // Almacena sus atributos en la fila del arreglo
        tabla[i][0] = pelicula.getClave();
        tabla[i][1] = pelicula.getTitulo();
        tabla[i][2] = pelicula.getActor1();
```

```
        tabla[i][3] = pelicula.getActor2();
        tabla[i][4] = pelicula.getDirector();
        tabla[i][5] = pelicula.getGenero().getNombre();
        tabla[i][6] = new Integer(pelicula.getDuracion());
        tabla[i][7] = pelicula.getFecha();
      }
      return new DefaultTableModel(tabla, nombresColumnasTablasPeliculas);
    }

    return null;
  }

  /**
   * Genera un objeto de tipo DefaultTableModel a partir de una lista de
   * generos.
   *
   * @param listaGeneros Lista de generos a convertir
   * @return Objeto de tipo DefaultTableModel con los atributos de los
   * generos.
   */
  public DefaultTableModel generosTableModel(List<Genero> listaGeneros) {
    Object tabla[][];

    if (listaGeneros != null) {
      tabla = new Object[listaGeneros.size()][3];
      for (int i = 0; i < listaGeneros.size(); i++) {
        // Obten un género de la lista de géneros
        Genero genero = (Genero) listaGeneros.get(i);
        // Almacena sus atributos en la fila del arreglo
        tabla[i][0] = genero.getCveGenero();
        tabla[i][1] = genero.getNombre();
        tabla[i][2] = genero.getTipoMedio();
      }
      return new DefaultTableModel(tabla, nombresColumnasTablasGeneros);
    }

    return null;
  }

  /**
   * Genera un objeto de tipo DefaultComboBoxModel a partir de una lista de
   * géneros.
   */
  public DefaultComboBoxModel<Genero> generosComboBoxModel(List<Genero>
                                                           listaGeneros) {
    DefaultComboBoxModel<Genero> defaultComboBoxModel = new
                                                  DefaultComboBoxModel<>();

    if (listaGeneros != null) {
      // Para cada elemento de la Lista
      for (int i = 0; i < listaGeneros.size(); i++)
      {
        // Agregalo a la instancia de la clase DefaultComboBoxModel
        defaultComboBoxModel.addElement(listaGeneros.get(i));
      }

      return defaultComboBoxModel;
```

```
    }
    return null;
  }
}
```

El caso de uso **Actualizar Canción** nos permite actualizar una canción del catálogo de canciones. El diagrama de secuencia de la figura 8.4 realiza dicho caso de uso.

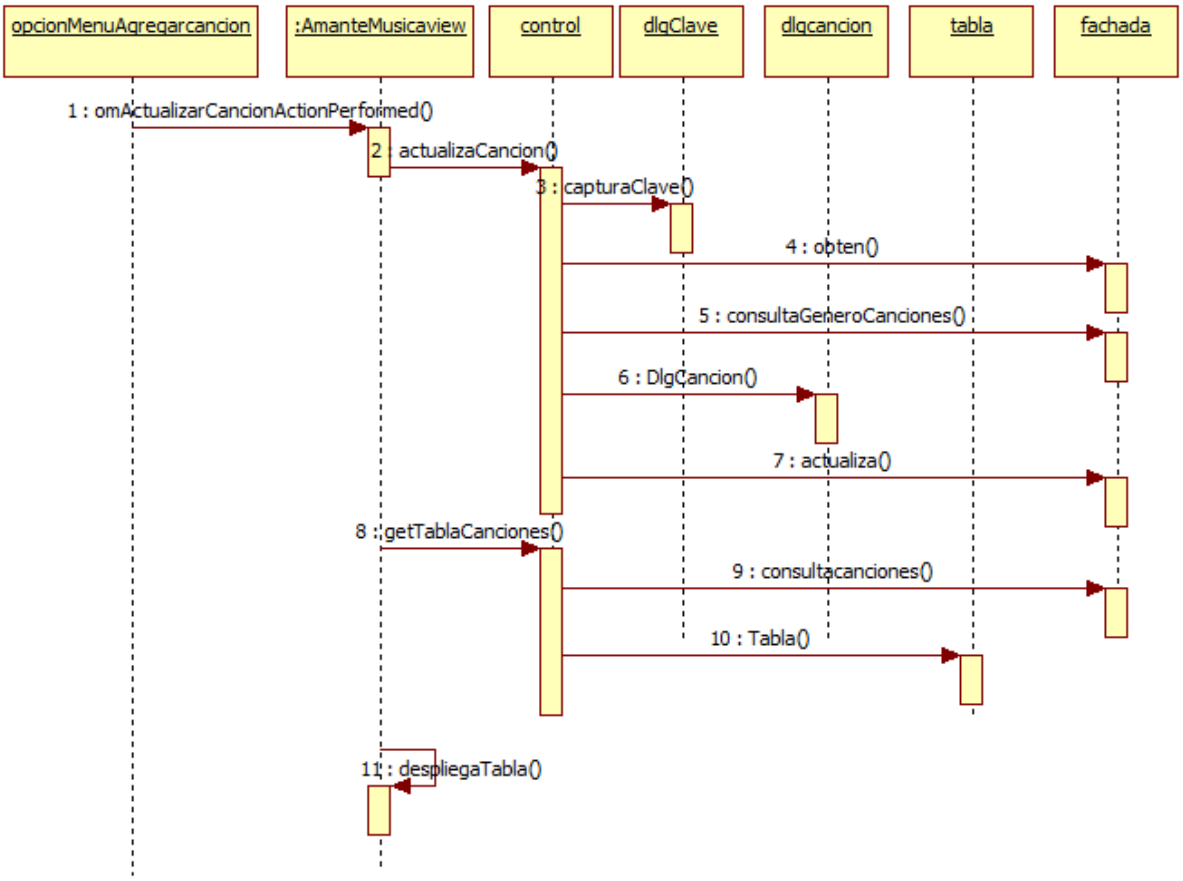

Figura 8.4 Diagrama de Secuencia – Actualizar Canción

El caso de uso empieza cuando el usuario selecciona del menú de la ventana de la aplicación la opción Actualizar Canción, haciendo que la opción del menú invoque al método oyente `opcionMenuActualizarCancionActionPerformed()`, cuyo código se muestra en el siguiente fragmento de código:

FrmAmanteMusica.java (Fragmento)

```
  ...

  /**
   * Metodo oyente que actualiza una cancion del catalogo de canciones
   *
   * @param evt Evento al que escucha
   */
  private void
```

```
opcionMenuActualizarCancionActionPerformed(java.awt.event.ActionEvent evt) {
    // Actualiza la canción
    if (control.actualizaCancion(this)) {

      // Obtiene la lista de canciones
      Tabla tablaCanciones = control.getTablaCanciones(this);

      // Despliega la lista de canciones
      despliegaTabla(tablaCanciones);
    }
  }                                                                    ...
```

El método oyente `opcionMenuActualizarCancionActionPerformed()` primero invoca al método `actualizaCanción()` de la clase **Control** que captura la clave de la canción a actualizar, obtiene la canción del catálogo de canciones, edita los datos de la canción y guarda la canción en el catálogo de canciones. A continuación el método oyente invoca al método `getTablaCanciones()` que obtiene la tabla de canciones y por último el método oyente invoca al método `despliegaTabla()` para que despliegue la lista de canciones.

En el fragmento de código siguiente se muestra el método `actualizaCanción()` encargado de actualizar la canción del catálogo de canciones.

Control.java (Fragmento)

```
  /**
   * Actualiza una cancion del catalogo de canciones
   * @param frame Ventana sobre la que se despliega el cuadro de dialogo para
   *              editar los datos de la cancion
   * @return Regresa true si se pudo actualizar la cancion, false en caso
   * contrario
   */
  public boolean actualizaCancion(JFrame frame) {
    Cancion cancion;
    StringBuffer respuesta = new StringBuffer("");
    DlgCancion dlgCancion;
    List<Genero> listaGenerosCanciones;
    DefaultComboBoxModel<Genero> generosCancionesComboBoxModel;

    // Captura la clave de la cancion
    String clave = JOptionPane.showInputDialog(frame, "Clave de la cancion:",
                                               "Actualizar cancion",
                                               JOptionPane.QUESTION_MESSAGE);

    // Si el usuario presiono el boton Cancelar
    if(clave == null) return false;

    // Crea un objeto Cancion con solo la clave
    cancion = new Cancion(clave);

    try {
      // Obten la cancion del catalogo de canciones
      cancion = persistencia.obten(cancion);
    }
```

```
    catch (Exception e) {
      // Si ocurrio un error al leer del catalogo de canciones,
      // despliega mensaje de error
      JOptionPane.showMessageDialog(frame, e.getMessage(), "Error!!.",
                                    JOptionPane.ERROR_MESSAGE);
      return false;
    }

    // Si la cancion no existe, despliega un mensaje de error
    if(cancion == null) {
      JOptionPane.showMessageDialog(frame,
                                    "La cancion no existe", "Error!!.",
                                    JOptionPane.ERROR_MESSAGE);
      return false;
    }

    try {
      // Obtiene la lista de generos de canciones
      listaGenerosCanciones = persistencia.consultaGenerosCanciones();
    }
    catch (Exception e) {
      // Si ocurrio un error al obtener la lista de la base de datos,
      // despliega mensaje de error
      JOptionPane.showMessageDialog(frame, e.getMessage(), "Error!!.",
                                    JOptionPane.ERROR_MESSAGE);
      return false;
    }

    generosCancionesComboBoxModel = conversiones.
            generosComboBoxModel(listaGenerosCanciones);

    // Si la cancion existe, edita los datos de la cancion
    dlgCancion = new DlgCancion(frame, "Edita Datos Cancion", true, cancion,
                                generosCancionesComboBoxModel,
                                ConstantesGUI.ACTUALIZAR, respuesta);

    // Si el usuario presiono el boton Cancelar
    if (respuesta.substring(0).equals(ConstantesGUI.CANCELAR)) return false;

    // Actualiza la cancion del catalogo de canciones
    try {
      persistencia.actualiza(cancion);
    }
    catch(Exception e)  {
      // Si ocurrio un error al escribir al catalogo de canciones,
      // despliega mensaje de error
      JOptionPane.showMessageDialog(frame, e.getMessage(), "Error!!.",
                                      JOptionPane.ERROR_MESSAGE);
      return false;
    }
    return true;
  }
```

El caso de uso **Eliminar Canción** nos permite eliminar una canción del catálogo de canciones. El diagrama de secuencia de la figura 8.5 realiza dicho caso de uso.

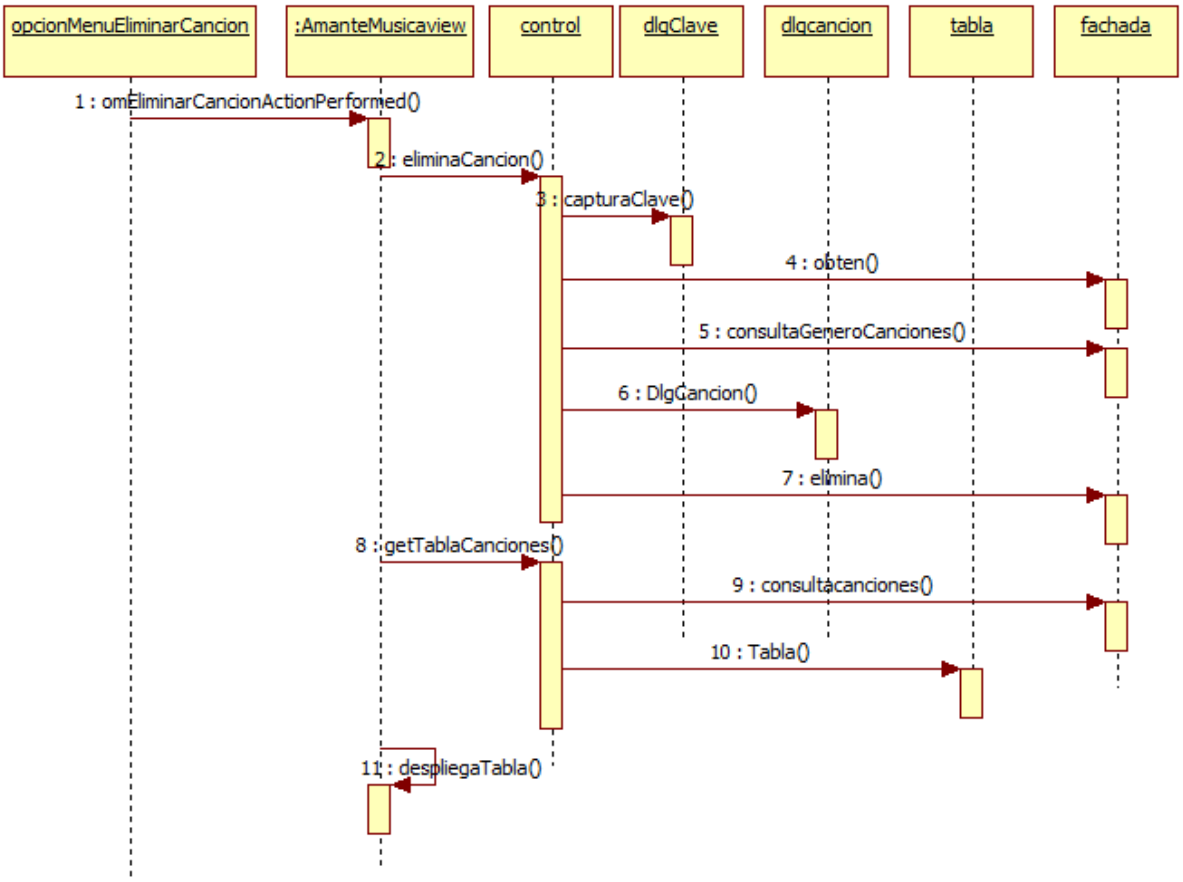

Figura 8.5 Diagrama de Secuencia – Eliminar Canción

El caso de uso empieza cuando el usuario selecciona del menú de la ventana de la aplicación la opción Eliminar Canción, haciendo que la opción del menú invoque al método oyente `opcionMenuEliminarCancionActionPerformed()`, cuyo código se muestra en el siguiente fragmento de código:

FrmAmanteMusica.java (Fragmento)

```
...
  /**
   * Metodo oyente que elimina una cancion del catalogo de canciones
   *
   * @param evt Evento al que escucha
   */
  private void
   opcionMenuEliminarCancionActionPerformed(java.awt.event.ActionEvent evt) {
    if (control.eliminaCancion(this)) {

      // Obtiene la lista de canciones
      Tabla tablaCanciones = control.getTablaCanciones(this);

      // Despliega la lista de canciones
      despliegaTabla(tablaCanciones);
```

```
    }
  }
...
```

El método oyente `opcionMenuEliminarCancionActionPerformed()` primero invoca al método `eliminaCanción()` de la clase **Control** que captura la clave de la canción a eliminar, obtiene la canción del catálogo de canciones, la muestra para confirmar la orden y elimina la canción en el catálogo de canciones. A continuación el método oyente invoca al método `getTablaCanciones()` que obtiene la tabla de canciones y por último el método oyente invoca al método `despliegaTabla()` para que despliegue la lista de canciones.

En el fragmento de código siguiente se muestra el método `eliminaCanción()` encargado de eliminar la canción del catálogo de canciones.

Control.java (Fragmento)

```
...

  /**
   * Elimina una cancion del catalogo de canciones
   * @param frame Ventana sobre la que se despliega el cuadro de dialogo para
   *              desplegar los datos de la cancion
   * @return Regresa true si se pudo eliminar la cancion, false en caso
   * contrario
   */
  public boolean eliminaCancion(JFrame frame) {
    Cancion cancion;
    StringBuffer respuesta = new StringBuffer();
    DlgCancion dlgCancion;
    List<Genero> listaGenerosCanciones;
    DefaultComboBoxModel<Genero> generosCancionesComboBoxModel;

    // Captura la clave de la cancion
    String clave = JOptionPane.showInputDialog(frame, "Clave de la cancion:",
                                               "Eliminar cancion",
                                               JOptionPane.QUESTION_MESSAGE);

    // Si el usuario presiono el boton Cancelar
    if(clave == null) return false;

    // Crea un objeto Cancion con solo la clave
    cancion = new Cancion(clave);

    try {
      // Obten la cancion del catalogo de canciones
      cancion = persistencia.obten(cancion);
    }
    catch (Exception e) {
      // Si ocurrio un error al leer del catalogo de canciones
      // despliega mensaje de error
      JOptionPane.showMessageDialog(frame, e.getMessage(), "Error!!.",
                                    JOptionPane.ERROR_MESSAGE);
      return false;
```

```
    }

    // Si la cancion no existe en el catalogo de canciones,
    if(cancion == null) {
      // despliega mensaje de error
      JOptionPane.showMessageDialog(frame, "La cancion no existe",
                                    "Error!!.", JOptionPane.ERROR_MESSAGE);
      return false;
    }

    try {
      // Obtiene la lista de generos de canciones
      listaGenerosCanciones = persistencia.consultaGenerosCanciones();
    }
    catch (Exception e) {
      // Si ocurrio un error al obtener la lista de la base de datos,
      // despliega mensaje de error
      JOptionPane.showMessageDialog(frame, e.getMessage(), "Error!!.",
                                    JOptionPane.ERROR_MESSAGE);
      return false;
    }

    generosCancionesComboBoxModel = conversiones.
            generosComboBoxModel(listaGenerosCanciones);

    // Si existe la cancion, despliega los datos de la cancion
    dlgCancion = new DlgCancion(frame, "Cancion a borrar", true, cancion,
                                generosCancionesComboBoxModel,
                                ConstantesGUI.ELIMINAR, respuesta);

    // Si el usuario presiono el boton Cancelar
    if(respuesta.substring(0).equals(ConstantesGUI.CANCELAR)) return false;

    try {
      // Elimina la cancion del catalogo de canciones
      persistencia.elimina(cancion);
    }
    catch(Exception e)  {
      // Si ocurrio un error al borrar del catalogo de canciones,
      // despliega mensaje de error
      JOptionPane.showMessageDialog(frame, e.getMessage(), "Error!!.",
                                    JOptionPane.ERROR_MESSAGE);
      return false;
    }
    return true;
  }
...
```

El caso de uso **Consulta Canciones** nos permite listar las canciones del catálogo de canciones. El diagrama de secuencia de la figura 8.6 realiza dicho caso de uso.

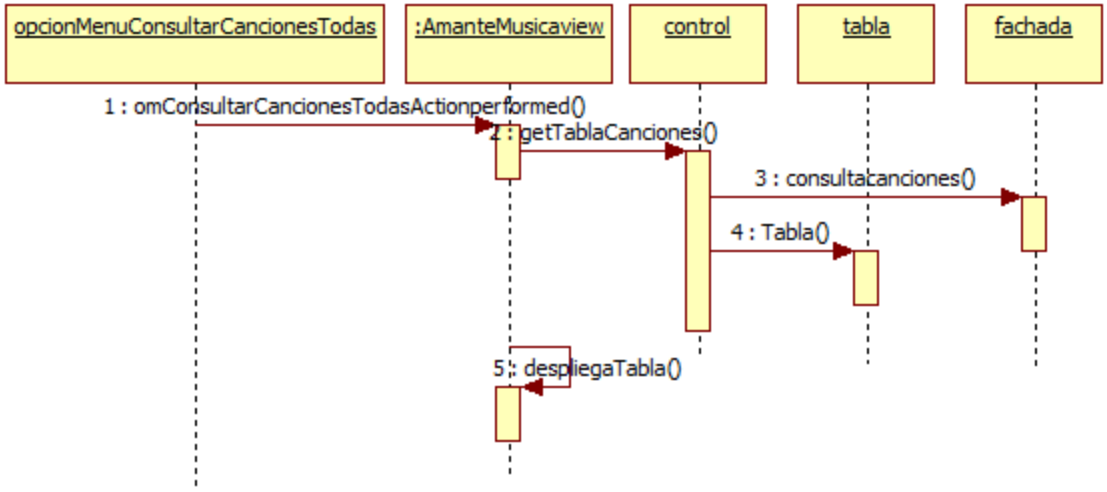

Figura 8.6 Diagrama de Secuencia – Consultar Canciones

El caso de uso empieza cuando el usuario selecciona del menú de la ventana de la aplicación la opción **Consulta Canciones Todas**, haciendo que la opción del menú invoque al método oyente `opcionMenuConsultasCancionesTodasActionPerformed()`, cuyo código se muestra en el siguiente fragmento de código:

FrmAmanteMusica.java (Fragmento)

```
  ...

  /**
   * Metodo oyente que obtiene y despliega la lista de canciones
   *
   * @param evt Evento al que escucha
   */
  private void
opcionMenuConsultasCancionesTodasActionPerformed(java.awt.event.ActionEvent
evt) {
    // Obtiene la lista de canciones
    Tabla tablaCanciones = control.getTablaCanciones(this);

    // Despliega la lista de canciones
    if (tablaCanciones != null) {
      despliegaTabla(tablaCanciones);
    }
  }                                                                    ...
```