Tema 5

Arreglos y Cadenas en Java

Arreglos

Un arreglo es una lista de variables del mismo tipo que se encuentran en localidades contiguas de memoria y que comparten el mismo nombre. Cada una de las variables de la lista, llamadas **elementos**, puede ser accedida por el nombre del arreglo y su posición en él. Dependiendo de la distribución lógica (no física) de sus elementos, los arreglos pueden clasificarse en:

- Unidimensionales, si visualizamos a los elementos acomodados en una fila (o columna).
- Bidimensionales, si los visualizamos acomodados en una tabla.
- Tridimensionales, si los visualizamos acomodados en varias tablas formando una estructura tridimensional.
- Etc. No hay restricciones en el número de dimensiones que un arreglo pueda tener.

Arreglos Unidimensionales

En un arreglo unidimensional, la posición de un elemento en el arreglo está dada por un valor entero llamado **índice** que representa la distancia de ese elemento con respecto al primer elemento del arreglo. Por ejemplo si deseamos almacenar las calificaciones de un grupo de alumnos podemos tener un arreglo llamado califs formado de 50 variables:

`califs[0]` representa la calificación del primer alumno,
`califs[1]` representa la calificación del segundo alumno, etc.

El número entre corchetes es el índice. Note que el primer elemento tiene un índice igual a 0 y el último un índice igual a n-1, donde n es el número de elementos del arreglo.

Referencia a un Arreglo Unidimensional

Al igual que con los objetos, el nombre de un arreglo es una referencia al arreglo, esto es, utilizamos el nombre para acceder a los elementos del arreglo.

La sintaxis para declarar la referencia a un arreglo unidimensional es la siguiente:

```
tipo nomArreglo[];
```

ó

```
tipo[] nomArreglo;
```

tipo es el **tipo base** del arreglo, el tipo de cada una de las variables que forman el arreglo, y puede ser cualquier tipo de datos: primitivos o clases. *nomArreglo* es un identificador, el nombre de la referencia al arreglo.

Por ejemplo:

```
int califs[];
double[] precios;
Cancion canciones[];
```

Creación de un Arreglo Unidimensional

Al igual que con los objetos, los arreglos deben crearse antes de emplearse, esto reserva memoria para el arreglo. La sintaxis para crear un arreglo es la siguiente:

```
nomArreglo = new tipo[tamArreglo];
```

tamArreglo es una expresión de tipo entero que indica el tamaño del arreglo, el número de variables de la lista. Los corchetes encerrando a *tamArreglo* no indican que éste sea opcional. Aquí forman parte de la sintaxis de la creación de un arreglo.

Por ejemplo:

```
califs = new int[50];
precios = new double[100];
canciones = new Cancion[20];
```

La declaración de la referencia a un arreglo y el reservado de memoria para el mismo pueden combinarse en una sentencia cuya sintaxis es:

```
tipo nomArreglo[] = new tipo[tamArreglo];
```

ó

```
tipo[] nomArreglo = new tipo[tamArreglo];
```

Por ejemplo:

```
int califs[] = new int[50];
double[] precios = new double[100];
Cancion canciones[] = new Cancion[20];
```

Inicialización de Arreglos Unidimensionales

Los arreglos al igual que las variables de los tipos básicos pueden inicializarse al momento de crearse. La sintaxis para inicializar un arreglo es

```
tipo nomArreglo[] = {cte1 [, cte2] ... }
```

ó

```
tipo[] nomArreglo = {cte1 [, cte2] ... }
```

Donde *cte1* `[,` *cte2*`] ...` es una lista de constantes encerrada entre llaves, `{` y `}`, que son los valores a los que se inicializan los elementos del arreglo. El tamaño del arreglo es el número de constantes. Por ejemplo, en la declaración

```
int x[] = {0, 1, 2, 3, 4};
```

estamos declarando al arreglo x y lo estamos inicializando a:

x	0	1	2	3	4

Acceso a los elementos de un Arreglo Unidimensional

Para acceder a un elemento de un arreglo unidimensional se utiliza el nombre del elemento, tal como lo haríamos con cualquier variable. El nombre de cada elemento de un arreglo está formado por el nombre del arreglo y su posición en el arreglo. Esta posición se conoce como el **índice** del elemento. El índice del primer elemento de un arreglo siempre es cero. El índice de un elemento puede expresarse mediante cualquier expresión entera que al evaluarse nos indica el elemento en particular al que queremos acceder. Por ejemplo

```
califs[4] = 8;
```

le asigna al quinto elemento del arreglo `califs` el valor de 8.

```
califs[i]
```

accede al i-ésimo elemento del arreglo `califs`, mientras que

```
mats[j + 3]
```

accede al j-ésimo + 3 elemento del arreglo `mats`.

Por ejemplo supongamos que deseamos acceder en forma secuencial a los elementos del arreglo `calif`s para sacar el promedio de las calificaciones. El código para hacer esto sería

```
suma = 0;
for(i = 0; i < nAlumnos; i++) suma += califs[i];
promedio = (double)suma/nAlumnos;
```

donde `nAlumnos` es el número de calificaciones que están almacenadas en el arreglo. Este valor puede ser menor o igual al tamaño del arreglo `calif`s.

Cada vez que se le asigna un valor u objeto a un elemento de un arreglo, el sistema de ejecución de Java verifica que el tipo del arreglo sea compatible con el tipo del valor u objeto asignársele. Si no son compatibles ocurre una excepción del tipo: `java.lang.ArrayStoreException`.

Cada vez que se accede a un elemento de un arreglo, el sistema de ejecución de Java verifica que el valor del índice es un valor válido, esto es, que es número comprendido entre 0 y *tamArreglo* – 1. De no ser así ocurre una excepción del tipo: `java.lang.ArrayIndexOutOfBoundsException`.

Arreglos Unidimensionales y Métodos

En Java los arreglos son objetos. Podemos obtener el tamaño del arreglo en elementos de la siguiente manera:

```
nomArreglo.length
```

En Java podemos pasarle como parámetro a un método, una referencia a un arreglo. No se le está pasando una copia del arreglo, sólo la referencia. La sintaxis de un parámetro que representa la referencia a un arreglo unidimensional es la siguiente:

```
tipo nomParArreglo[]
```

donde *`tipo`* es el **tipo base** del arreglo y *`nomParArreglo`* es el nombre del parámetro que representa la referencia al arreglo. Note que los corchetes están vacíos.

Al llamar al método, el argumento será la referencia al arreglo, el cual es el nombre del arreglo:

```
nomArreglo
```

También es posible que un método nos regrese una referencia a un arreglo. La sintaxis para declarar que el método regresa una referencia a un arreglo unidimensional es:

```
tipo[] nomMetodo() {
  ...
}
```

donde *tipo* es el **tipo base** del arreglo y *nomMetodo* es el nombre del método que nos regresa la referencia a un arreglo.

Ejemplos Sobre Arreglos Unidimensionales

Para el ejemplo del programa del amante de la música y el cine, se desea implementar un mecanismo que permita catalogar (almacenar y consultar) su colección. En este tema el mecanismo se implementará usando arreglos, aunque los arreglos no permiten almacenar permanentemente los datos. Más adelante se modificará este mecanismo para que emplee archivos y una base de datos.

Dado que los arreglos tienen un tamaño fijo definido al tiempo de compilación, es posible que el arreglo se llene y no se puedan agregar más datos. En lugar de verificar si hay espacio en el arreglo para almacenar un dato, dejaremos que el sistema de ejecución de Java lance una excepción del tipo `java.lang.ArrayIndexOutOfBoundsException` si queremos acceder más allá de los límites del arreglo, atraparemos esa excepción y relanzaremos una excepción encadenada del tipo `PersistenciaException`. Por otro lado si al querer modificar o eliminar un dato, el dato no existe, también se lanzará una excepción del tipo `PersistenciaException`. El siguiente código muestra la implementación de la excepción `PersistenciaException`.

PersistenciaException.java

```
/*
 * PersistenciaException.java
 *
 * Creada el 13 de septiembre de 2007, 12:01 AM
 */

package excepciones;

/**
 * Esta clase representa a las excepciones lanzadas por las clases
 * que encapsulan la persistencia.
 *
 * @author mdomitsu
 */
public class PersistenciaException extends RuntimeException {
  /**
   * Constructor por omisión. Construye una excepción con un mensaje de error
   * nulo.
   */
  public PersistenciaException() {
  }

  /**
   * Construye una excepción con el mensaje de error del parámetro.
```

```
   * @param msj Mensaje de error.
   */
  public PersistenciaException(String msj) {
    super(msj);
  }

  /**
   * Construye una excepción con el mensaje de error del parámetro y la causa
   * original del error.
   * @param msj Mensaje de error.
   * @param causa Causa original del error.
   */
  public PersistenciaException(String msj, Throwable causa) {
    super(msj, causa);
  }

  /**
   * Construye una excepción la causa original del error.
   * @param causa Causa original del error.
   */
  public PersistenciaException(Throwable causa) {
    super(causa);
  }
}
```

Para encapsular el mecanismo de almacenamiento de los datos se construyen cuatro clases nuevas: `Medios`, `Canciones`, `Peliculas` y `Generos` se muestran en el diagrama de clases de la figura 5.1

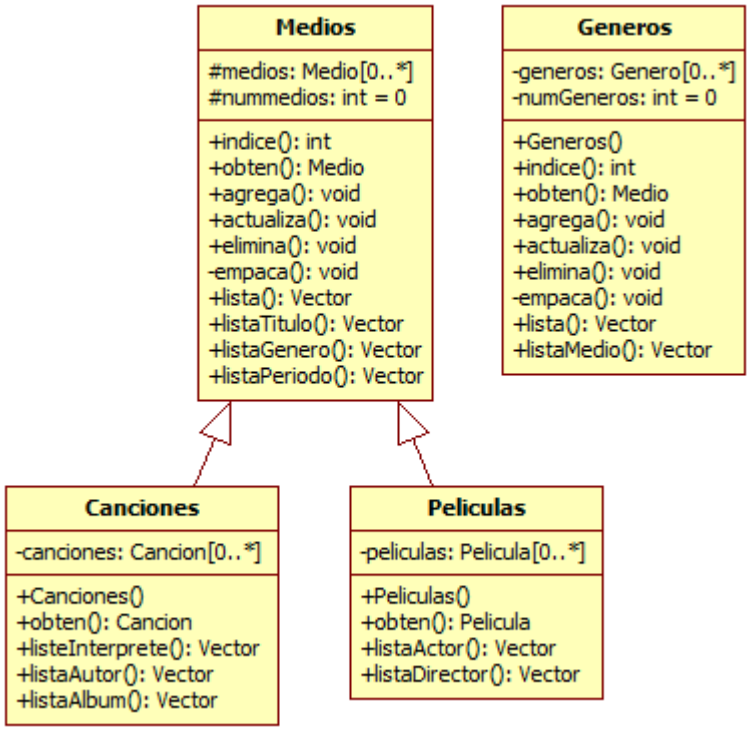

Figura 5.1

La clase `Medios` tiene como atributos una referencia a un arreglo de tipo `Medio` y un entero con el número de medios en el catálogo. Los métodos de la clase `Medios` permiten buscar un medio dentro del catálogo dada su clave, agregar, actualizar y eliminar un medio al catálogo, listar los medios que sean de un género dado, listar los medios que sean de un período dado y listar el catálogo. El código parcial de la clase `Medios` es el siguiente:

Medios.java

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

package persistencia;

import java.util.Vector;

import objetosServicio.*;
import objetosNegocio.Medio;
import excepciones.PersistenciaException;

/**
 * Esta clase permite almacenar, actualizar, eliminar y consultar
 * medios (canciones y películas) del programa AmanteMusica.
 * Los datos se almacenan en arreglos.
 *
 * @author mdomitsu
 */
public class Medios {
  protected Medio medios[];
  protected int numMedios = 0;

  /**
   * Busca el medio del parámetro en el arreglo. Las claves de los medios del
   * arreglo y del parametro deben coincidir
   * @param medio Medio a buscar
   * @return La posición del medio buscado si se encuentra,
   * -1 en caso contrario
   */
  public int indice(Medio medio) {
    // Recorre linealmente el arreglo en busca del medio
    for(int i = 0; i < numMedios; i++) {
      // Si lo haya regresa su índice
      if(medio.equals(medios[i])) return i;
    }
    // si no, regresa -1
    return -1;
  }

  /**
   * Regresa el medio del arreglo que coincida con el medio del parametro.
   * Las claves de los medios del arreglo y del parametro deben coincidir
   * @param medio Medio con la clave del medio a regresar
   * @return El medio cuya clave es igual a la clave del medio
```

```
   * dado por el parámetro, si se encuentra, null en caso contrario
   */
  public Medio obten(Medio medio) {
    // Busca el índice del medio de clave
    int pos = indice(medio);

    // Si lo encontró, regrésalo
    if(pos >= 0) return medios[pos];

    // si no, regresa null
    return null;
  }

  /**
   * Agrega un medio al arreglo de medios
   * @param medio Medio a agregar.
   * @throws PersistenciaException Si el catálogo está lleno
   */
  public void agrega(Medio medio) throws PersistenciaException {
    // Si no hay espacio en el arreglo no se agrega
    if(numMedios >= medios.length)
      throw new PersistenciaException("Arreglo lleno");

    // Agrega la canción o película al arreglo
    medios[numMedios] = medio;
    numMedios++;
  }

  /**
   * Actualiza el medio del arreglo que coincida con el medio del parametro.
   * Las claves de los medios del arreglo y del parametro deben coincidir
   * @param medio Medio a actualizar.
   * @throws PersistenciaException Si el medio no existe.
   */
  public void actualiza(Medio medio) throws PersistenciaException  {
    int pos;

    // Busca un medio con la misma clave.
    pos = indice(medio);

    // Si no lo hay, no se actualiza
    if(pos < 0) throw new PersistenciaException("Medio inexistente");

    // Si lo hay, actualizalo
    medios[pos] = medio;
  }

  /**
   * Elimina el medio del arreglo que coincida con el medio del parametro.
   * Las claves de los medios del arreglo y del parametro deben coincidir
   * @param medio Medio a eliminar.
   * @throws PersistenciaException Si el medio no existe.
   */
  public void elimina(Medio medio) throws PersistenciaException  {
    int pos;

    // Busca un medio con la misma clave.
```

```
    pos = indice(medio);

    // Si no lo hay, no se borra
    if(pos < 0) throw
       new PersistenciaException("La Canción o Película no existe");

    // Si lo hay, borralo
    medios[pos] = null;
    empaca();
  }

  /**
   * Elimina los medios que se hayan eliminado, recorriendo el resto una
   * posición hacia adelante
   */
  private void empaca() {
    for(int i=0; i < numMedios; i++) {
      // Si encuentra un medio borrado
      if(medios[i] == null) {

        // Recorre todas los medios una posicion hacia
        // arriba para recuperar el espacio
        for(int j=i; j < numMedios-1; j++) {
          medios[j] = medios[j+1];
        }
        numMedios--;
      }
    }
  }

  ...
}
```

La clase `Canciones` crea un arreglo de tipo `Cancion` que contendrá el catálogo de canciones. Los métodos de la clase `Canciones` permiten obtener una canción dada su clave, listar las canciones que sean de un intérprete, de un autor o de un álbum dado. El código parcial de la clase `Canciones` es el siguiente.

Canciones.java

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

package persistencia;

import objetosNegocio.Cancion;

/**
 * Esta clase permite consultar canciones del programa AmanteMusica
 * Los datos se almacenan en arreglos.
 *
 * @author mdomitsu
 */
```

```
public class Canciones extends Medios {
  // Arreglo para almacenar las canciones
  private Cancion canciones[];

  /**
   * Crea el arreglo para almacenar las canciones de tamaño igual al parámetro
   * @param tamCatalogo Tamaño del arreglo canciones
   */
  public Canciones(int tamCatalogo) {
    // Crea el arreglo para almacenar las canciones
    canciones = new Cancion[tamCatalogo];

    // Hace que la referencia medios de la clase Medios apunte al arreglo
    // canciones
    medios = canciones;
  }

  /**
   * Regresa la canción del arreglo cuya clave coincida con la clave
   * de la canción dado por el parámetro
   * @param cancion Canción con la clave de la canción a regresar
   * @return La canción cuya clave es igual a la clave de la canción dada por
   * el parámetro
   */
  public Cancion obten(Cancion cancion) {
    // Obtiene la canción invocando al método obten() de la clase padre Medios
    return (Cancion)super.obten(cancion);
  }

  ...
}
```

La clase `Peliculas` crea un arreglo de tipo `Pelicula` que contendrá el catálogo de películas. Los métodos de la clase `Peliculas` permiten obtener una película dada su clave, listar las películas que sean de un actor o de un director dados. El código parcial de la clase Peliculas es el siguiente.

Peliculas.java

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

import objetosNegocio.Pelicula;

/**
 * Esta clase permite consultar películas del programa AmanteMusica
 * Los datos se almacenan en arreglos.
 *
 * @author mdomitsu
 */
public class Peliculas extends Medios {
  // Arreglo para almacenar las películas
```

```
  private Pelicula peliculas[];

  /**
   * Crea el arreglo para almacenar las peliculas de tamaño igual al parámetro
   * @param tamCatalogo amCatalogo Tamaño del arreglo peliculas
   */
  public Peliculas(int tamCatalogo) {
    // Crea el arreglo para almacenar las películas
    peliculas = new Pelicula[tamCatalogo];

    // Hace que la referencia medios de la clase Medios apunte al arreglo
    // peliculas
    medios = peliculas;
  }

  /**
   * Regresa la película del arreglo cuya clave coincida con la clave
   * de la película dado por el parámetro
   * @param pelicula Película con la clave de la película a regresar
   * @return La película cuya clave es igual a la clave de la película
   * dada por el parámetro
   */
  public Pelicula obten(Pelicula pelicula) {
    // Obtiene la canción invocando al método obten() de la clase padre Medios
    return (Pelicula)super.obten(pelicula);
  }

  ...
}
```

La clase `Generos` tiene como atributos una referencia a un arreglo de tipo `Genero` y un entero con el número de géneros en el catálogo. Los métodos de la clase `Generos` permiten buscar un género dentro del catálogo dada su clave, agregar, actualizar y eliminar un género al catálogo, listar el catálogo. El código parcial de la clase `Generos` es el siguiente:

Generos.java

```
/*
 * Generos.java
 *
 * Creada el 30 de julio de 2008, 11:36 AM
 */

package persistencia;

import java.util.Vector;

import objetosNegocio.Genero;
import excepciones.PersistenciaException;

/**
 * Esta clase permite almacenar, actualizar, eliminar y consultar
 * generos de canciones y películas del programa AmanteMusica.
 * Los datos se almacenan en arreglos.
 *
```

```
 * @author mdomitsu
 */
public class Generos {
  protected Genero generos[];
  protected int numGeneros = 0;

  /**
   * Crea el arreglo para almacenar los géneros de tamaño igual al parámetro
   * @param tamCatalogo Tamaño del arreglo generos
   */
  public Generos(int tamCatalogo) {
    // Crea el arreglo para almacenar los géneros
    generos = new Genero[tamCatalogo];
  }

  /**
   * Busca el genero del parámetro en el arreglo. Las claves de los generos del
   * arreglo y del parametro deben coincidir
   * @param genero Genero a buscar
   * @return La posición del genero buscado si se encuentra,
   * -1 en caso contrario
   */
  public int indice(Genero genero) {
    // Recorre linealmente el arreglo en busca del genero
    for(int i = 0; i < numGeneros; i++) {
      // Si lo halla regresa su índice
      if(genero.equals(generos[i])) return i;
    }
    // si no, regresa -1
    return -1;
  }

  /**
   * Regresa el genero del arreglo que coincida con el genero del parametro.
   * Las claves de los generos del arreglo y del parametro deben coincidir
   * @param genero Genero con la clave del genero a regresar
   * @return El genero cuya clave es igual a la clave del genero
   * dado por el parámetro, si se encuentra, null en caso contrario
   */
  public Genero obten(Genero genero) {
    // Busca un genero con la misma clave.
    int pos = indice(genero);

    // Si lo encontró, regrésalo
    if(pos >= 0) return generos[pos];

    // si no, regresa null
    return null;
  }

  /**
   * Agrega un genero al arreglo de generos
   * @param genero Genero a agregar.
   * @throws PersistenciaException Si el catálogo está lleno
   */
  public void agrega(Genero genero) throws PersistenciaException {
    int pos;
```

```
    // Si no hay espacio en el arreglo no se agrega
    if(numGeneros >= generos.length)
        throw new PersistenciaException("Arreglo lleno");

    // Se agrega el genero
    generos[numGeneros] = genero;
    numGeneros++;
  }

  /**
   * Actualiza el genero del arreglo que coincida con el genero del parametro.
   * Las claves de los generos del arreglo y del parametro deben coincidir
   * @param genero Género a actualizar.
   * @throws PersistenciaException Si el género no existe.
   */
  public void actualiza(Genero genero) throws PersistenciaException {
    int pos;

    // Busca un genero con la misma clave.
    pos = indice(genero);

    // Si no lo hay, no se actualiza
    if(pos < 0) throw new PersistenciaException("Género inexistente");

    // Si lo hay, actualizalo
    generos[pos] = genero;
  }

  /**
   * Elimina el genero del arreglo que coincida con el genero del parametro.
   * Las claves de los generos del arreglo y del parametro deben coincidir
   * @param genero Genero a eliminar.
   * @throws PersistenciaException Si el género no existe.
   */
  public void elimina(Genero genero) throws PersistenciaException {
    int pos;

    // Busca un genero con la misma clave.
    pos = indice(genero);

    // Si no lo hay, no se borra
    if(pos < 0) throw new PersistenciaException("Género inexistente");

    // Si lo hay, borralo
    generos[pos] = null;
    empaca();
  }

  /**
   * Elimina los generos que se hayan eliminado, recorriendo el resto una
   * posición hacia adelante
   */
  private void empaca() {
    // Recorre linealmente el arreglo en busca de generos eliminados
    for(int i=0; i < numGeneros; i++) {
      // Si encuentra un genero borrado
```

```
      if(generos[i] == null) {

        // Recorre todas los generos una posicion hacia
        // arriba para recuperar el espacio
        for(int j=i; j < numGeneros-1; j++) {
          generos[j] = generos[j+1];
        }
        numGeneros--;
      }
    }
  }

  ...
}
```

Arreglos Multidimensionales

En un arreglo bidimensional, la posición de un elemento en el arreglo está dada dos valores que representan el renglón y la columna del elemento en la tabla. En un arreglo tridimensional, la posición de un elemento en el arreglo está dada por tres valores que representan la tabla, el renglón y la columna del elemento en el arreglo, etc.

Referencia a Arreglos Multidimensionales

Las sintaxis para declarar referencias a arreglos en dos, tres, ... dimensiones son las siguientes:

```
      tipo nomArreglo[][];
      tipo nomArreglo[][][];
      ...
ó
      tipo[][] nomArreglo;
      tipo[][][] nomArreglo;
      ...
```

tipo es el **tipo base** del arreglo y puede ser es cualquier tipo primitivo o clase. . *nomArreglo* es un identificador, el nombre de la referencia al arreglo.

Por ejemplo:

```
      int calPars[][];
      double[][][] ventas;
```

Creación de Arreglos Multidimensionales

Para crear arreglos en dos, tres, ... dimensiones se usa las siguientes sintaxis:

```
nomArreglo = new tipo[numFilas][numCols];
nomArreglo = new tipo[numTablas][numFilas][numCols];
...
```

numTablas, numFilas, numCols son expresiones enteras que representan los números de tablas, filas y columnas de los arreglos.

Por ejemplo:

```
calPars = new int[4][50];
ventas = new double[3][4][12];
```

La declaración de referencias a arreglos multidimensionales y el reservado de memoria para ellos mismos pueden combinarse. Por ejemplo la sintaxis para declarar y crear arreglos bidimensionales y tridimensionales es:

```
tipo nomArreglo[][] = new tipo[numFilas][numCols];
tipo nomArreglo[][][] = new tipo[numTablas][numFilas][numCols];
```

ó

```
tipo[][] nomArreglo = new tipo[numFilas][numCols];
tipo[][][] nomArreglo = new tipo[numTablas][numFilas][numCols];
```

Por ejemplo:

```
int calPars[][] = new int[4][50];
double[][][] ventas = new double[3][4][12];
```

Inicialización de Arreglos Multidimensionales

Los arreglos multidimensionales también pueden inicializarse al momento de declararse. Por ejemplo la siguiente declaración

```
int x[][] = {{0, 1, 2}, {3, 4, 5}, {6, 7, 8}};
```

declara al arreglo x y lo inicializa a

x

0	1	2
3	4	5
6	7	8

Un ejemplo de la inicialización de un arreglo en tres dimensiones sería:

```
int y[][][] = {{{0, 1}, {2, 3}, {4, 5}},
               {{6, 7}, {8, 9}, {10, 11}}
               {{12, 13}, {14, 15}, {16, 17}}
               {{18, 19}, {20, 21}, {22, 23}}};
```

Acceso a los elementos de un Arreglo Multidimensional

El acceso a los elementos de un arreglo multidimensional es similar al visto para los arreglos en una dimensión, nada más que en lugar de usar un sólo índice utilizaremos índices múltiples, dos índices para un arreglo en dos dimensiones, tres para un arreglo en tres dimensiones, etc. Por ejemplo

```
calPars[4][2] = 9;
```

le asigna al elemento que se encuentra en la quinta fila, tercera columna del arreglo calPars el valor de 9.

```
ventas[2][3][1] = 1252.30
```

le asigna al elemento que se encuentra en el tercer plano, cuarta fila, segunda columna del arreglo ventas el valor de 1252.30.

Por ejemplo el siguiente código nos permite calcular el promedio de las calificaciones de cada alumno y la calificación promedio del grupo:

```
int i, j, suma;
double sumaT, promedioT;
int calPars[][] = new int[50][4];
double promedios[] = new double[50];

...
sumaT = 0;
for(i = 0; i < nAlums; i++)
{
   suma = 0;
   for(j = 0; j < 4; j++) suma += calPars[i][j];
   promedios[i] = (double)suma/4;
   sumaT += promedios[i];
}
promedioT = sumaT/nAlums;
```

En realidad los arreglos multidimensionales son arreglos de arreglos. Un arreglo bidimensional es un arreglo donde cada elemento del arreglo es a su vez un arreglo unidimensional. Un arreglo tridimensional es un arreglo donde cada elemento del arreglo es a su vez un arreglo bidimensional, etc.

Cada uno de los arreglos individuales de un arreglo multidimensional puede ser referenciado independientemente. Por ejemplo, para las declaraciones:

```
int calPars[][] = new int[4][50];
double[][][] ventas = new double[3][4][12];
```

`calPars[i]` es una referencia al i-ésimo arreglo unidimensional del arreglo bidimensional `calPars`, esto es, el i-ésimo renglón de la tabla `calPars`; `ventas[i]` es una referencia al i-ésimo arreglo bidimensional del arreglo tridimensional `ventas`, esto es, la i-ésima tabla del arreglo tridimensional `ventas`; y `ventas[i][j]` es una referencia al i-ésimo, j-ésimo arreglo unidimensional del arreglo tridimensional `ventas`, esto es, al j-ésimo renglón de la i-ésima tabla del arreglo tridimensional `ventas`.

Arreglos Multidimensionales y Métodos

Al igual que con los arreglos unidimensionales, en Java podemos pasarle como parámetro a un método, una referencia a un arreglo multidimensional. La sintaxis de los parámetros que representan las referencias a arreglos en dos y tres dimensiones es, respectivamente:

```
tipo nomParArreglo[][]
tipo nomParArreglo[][][]
```

donde *`tipo`* es el **tipo base** del arreglo y *`nomParArreglo`* es el nombre del parámetro que representa la referencia al arreglo. Note que los corchetes están vacíos.

Al llamar al método, el argumento será la referencia al arreglo, el cual es el nombre del arreglo:

```
nomArreglo
```

También es posible que un método nos regrese una referencia a un arreglo multidimensional. La sintaxis para declarar métodos que regresan referencias a arreglos en dos y tres dimensiones es, respectivamente:

```
tipo[][] nomMetodo() {
  ...
}

tipo[][][] nomMetodo() {
  ...
}
```

donde *`tipo`* es el **tipo base** del arreglo y *`nomMetodo`* es el nombre del método que nos regresa la referencia a un arreglo.

Cadenas

Junto con el procesamiento numérico, una de las tareas que más frecuentemente realiza la computadora es el procesamiento de textos. Algunas de las aplicaciones en las que se utiliza el procesamiento de texto son:

- Bases de datos como directorios telefónicos, nóminas de una empresa, expedientes de alumnos de una escuela, etc.
- Procesadores de palabras y programas para publicación
- Compiladores e intérpretes.

Todos los programas anteriores requieren de la capacidad de almacenar y manipular texto, es decir, secuencias de caracteres. A estas secuencias de caracteres se les conoce como **cadenas**. En una computadora sólo se pueden almacenar números y por lo tanto cada uno de los caracteres de una cadena es codificado mediante un número. El esquema de codificación más empleado en las computadoras es el esquema de codificación ASCII (por las siglas en inglés del comité que lo creo: American Standard Code for Information Interchange), el cual codifica cada carácter mediante un número en el rango 0 – 127, por lo que este esquema sólo permite codificar 128 caracteres, lo que es insuficiente para representar todos los caracteres empleados en los diferentes lenguajes empleados en los diferentes países. Debido a esto, Java utiliza para codificar los caracteres un nuevo esquema de codificación llamado Unicode. **Unicode** provee un código único para cada carácter, sin importar la plataforma, programa o lenguaje. Para lograr esto Unicode utiliza dos bytes para codificar cada carácter, lo que le permite representar 65,536 caracteres diferentes. Por compatibilidad con el código ASCII, los primeros 128 códigos de Unicode se utilizan para representar los mismos caracteres que los representados por el código ASCII.

Para el manejo de cadenas, la API de Java nos proporciona varias clases entre ellas las clases `String` y `StringBuffer`.

Clase `String`

La clase `String` representa una cadena de caracteres. Las instancias de la clase `String` son constantes. Su valor no puede cambiarse después de ser creadas.

Todas las literales de tipo cadena en Java se implementan como instancias de la clase String. Por ejemplo, la siguiente literal:

```
"Hola"
```

crea una instancia de tipo `String` con la secuencia de caracteres: ‘H’, ‘o’, ‘l’, ‘a’.

Al igual que con cualquier otro objeto, podemos declarar una referencia a `String` y al mismo tiempo crear un objeto y asociárselo. Por ejemplo:

```
String mensaje = "Hola";
```

El lenguaje Java define el operador (+) para concatenación cadenas y el operador (+=) que combina la concatenación con la asignación. Por ejemplo, siguiendo la declaración anterior, podemos tener la siguiente sentencia:

```
mensaje = mensaje + " Juan";
```

la cual es equivalente a:

```
mensaje += " Juan";
```

Las sentencias anteriores podrían interpretarse como que la cadena mensaje ha cambiado su contenido al agregársele la cadena `" Juan"`. Sin embargo, lo que ha pasado es que al concatenarse las cadenas `mensaje` y `" Juan"`, se crea un nuevo objeto de tipo `String` con la cadena resultante y este nuevo objeto se asigna a la referencia `mensaje`.

El lenguaje Java también define la conversión de objetos a cadenas mediante el método `toString()` definido para la clase `Object` de la cual heredan todas las clases de Java.

La Clase String posee un conjunto muy rico de métodos. La tabla 5.1 muestra algunos de esos métodos:

Tabla 5.1 Algunos Métodos de la Clase `String`

`public String(char[] value)` Construye una cadena con la secuencia de caracteres contenida en el arreglo dado por el parámetro. **Parámetro:** `value` **–** Caracteres con los que se formará la cadena **Lanza:** `NullPointerException` – Si el arreglo `value` es nulo.
`public String(StringBuffer buffer)` Construye una cadena con la secuencia de caracteres contenida en el objeto de tipo `StringBuffer` dado por el parámetro. **Parámetro:** `buffer` – Objeto del tipo `StringBuffer` con los caracteres con los que se formará la cadena **Lanza:** `NullPointerException` – Si `buffer` es nulo.

Tabla 5.1 Algunos Métodos de la Clase `String`. Continuación

`public char charAt(int index)`

Regresa el carácter cuyo índice está dado por el parámetro. El primer carácter tiene índice 0.

Parámetro:
`index` – Índice del carácter deseado.

Regresa:
El carácter con el índice dado por el parámetro.

Lanza:
`IndexOutOfBoundsException` – Si el índice es negativo o mayor que la longitud de esta cadena.

`public int compareTo(String str)`

Compara lexicográficamente esta cadena con la cadena dada por el parámetro.

Parámetro:
`str` –Cadena con la que se va a comparar esta cadena.

Regresa:
El valor de 0 si esta cadena es igual a la cadena dada por el parámetro.
Un valor menor que 0 si esta cadena es menor lexicográficamente que la cadena dada por el parámetro.
Un valor mayor que 0 si esta cadena es mayor lexicográficamente que la cadena dada por el parámetro.

Lanza:
`NullPointerException` – Si la cadena dada por el parámetro es `null`.

Una cadena es menor lexicográficamente que otra si apareciera listada en un diccionario antes que la otra.

`public int compareToIgnoreCase(String str)`

Compara lexicográficamente esta cadena con la cadena dada por el parámetro sin considerar mayúsculas. Una cadena es menor lexicográficamente que otra si apareciera listada en un diccionario antes que la otra.

Parámetro:
`str` – Cadena con la que se va a comparar esta cadena.

Regresa:
El valor de 0 si esta cadena es igual a la cadena dada por el parámetro.
Un valor menor que 0 si esta cadena es menor lexicográficamente que la cadena dada por el parámetro.
Un valor mayor que 0 si esta cadena es mayor lexicográficamente que la cadena dada por el parámetro.

Lanza:
`NullPointerException` – Si la cadena dada por el parámetro es `null`.

Tabla 5.1 Algunos Métodos de la Clase `String`. Continuación.

`public boolean contentEquals(StringBuffer buffer)`

Regresa `true` si y sólo sí esta cadena representa la misma secuencia que el objeto de tipo `StringBuffer` dado por el parámetro, `false` en caso contrario.

Parámetro:
`buffer` – Objeto de tipo `StringBuffer` con el que se va a comparar esta cadena.

Regresa:
`true` si y sólo sí esta cadena representa la misma secuencia que el objeto de tipo `StringBuffer` dado por el parámetro, `false` en caso contrario.

Lanza:
`NullPointerException` – Si la cadena dada por el parámetro es `null`.

`public boolean endsWith(String suffix)`

Regresa `true` si y sólo sí esta cadena termina con la misma secuencia de caracteres que la cadena dada por el parámetro, `false` en caso contrario.

Parámetro:
`suffix` – Cadena sufija.

Regresa:
`true` si y sólo sí esta cadena termina con la misma secuencia de caracteres que la cadena dada por el parámetro, `false` en caso contrario.

Lanza:
`NullPointerException` – Si la cadena dada por el parámetro es `null`.

`public boolean equals(Object object)`

Regresa `true` si y sólo sí el objeto dado por el parámetro no es `null` y es una cadena que representa la misma secuencia de caracteres que esta cadena, `false` en caso contrario.

Parámetro:
`object` – El objeto con que se compara esta cadena.

Regresa:
`true` si y sólo sí el objeto dado por el parámetro no es `null` y es una cadena que representa la misma secuencia de caracteres que esta cadena, `false` en caso contrario.

`public boolean equalsIgnoreCase(String str)`

Regresa `true` si y sólo sí esta cadena es igual a la cadena dada por el parámetro, sin tomar en cuenta las mayúsculas, `false` en caso contrario.

Parámetro:
`str` – La cadena con que se compara esta cadena.

Regresa:
`true` si y sólo sí esta cadena es igual a la cadena dada por el parámetro, sin tomar en cuenta las mayúsculas, `false` en caso contrario.

Tabla 5.1 Algunos Métodos de la Clase `String`. Continuación.

`public int indexOf(int ch)` Regresa el índice de la primera ocurrencia dentro de esta cadena del carácter `ch` dado por el parámetro. Si `ch` no está en esta cadena regresa -1. **Parámetro:** `str` – Carácter a buscar. **Regresa:** Índice de la primera ocurrencia dentro de esta cadena del carácter `ch` dado por el parámetro. Si `ch` no está en esta cadena regresa -1.
`public int indexOf(String str)` Regresa el índice de la primera ocurrencia dentro de esta cadena de la subcadena dada por el parámetro `str`. Si `str` no está en esta cadena regresa -1. **Parámetro:** `str` – Subcadena a buscar. **Regresa:** Índice de la primera ocurrencia dentro de esta cadena de la subcadena dada por el parámetro `str`. Si `str` no está en esta cadena regresa -1. **Lanza:** `NullPointerException` – Si la cadena dada por el parámetro es `null`.
`public int length()` Regresa la longitud de esta cadena, esto es, el número de caracteres. **Regresa:** La longitud de esta cadena
`public boolean startsWith(String prefix)` Regresa `true` si y sólo sí esta cadena empieza con la misma secuencia de caracteres que la cadena dada por el parámetro, `false` en caso contrario. **Parámetro:** `prefix` – Cadena prefija. **Regresa:** `true` si y sólo sí esta cadena empieza con la misma secuencia de caracteres que la cadena dada por el parámetro, `false` en caso contrario. **Lanza:** `NullPointerException` – Si la cadena dada por el parámetro es `null`.

Tabla 5.1 Algunos Métodos de la Clase `String`. Continuación.

`public String substring(int beginIndex)` Regresa una nueva cadena que es un subcadena de esta cadena. La subcadena empieza con el carácter en el índice dado por el parámetro y se extiende hasta el final de esta cadena. **Parámetro:** `beginIndex` – Índice inicial de la subcadena inclusive. **Regresa:** `La subcadena especificada`. **Lanza:** `IndexOutOfBoundsException` – Si `beginIndex` es negativo o mayor que la longitud de esta cadena.
`public String substring(int beginIndex, int endIndex)` Regresa una nueva cadena que es un subcadena de esta cadena. La subcadena empieza con el carácter en el índice dado por el parámetro `beginIndex` y se extiende hasta el carácter en el índice `endIndex - 1`. **Parámetros:** `beginIndex` – Índice inicial de la subcadena inclusive. **`end`**`Index` – Índice final de la subcadena exclusive. **Regresa:** `La subcadena especificada`. **Lanza:** `IndexOutOfBoundsException` – Si `beginIndex` es negativo o **`end`**`Index` es mayor que la longitud de esta cadena o `beginIndex` es mayor que **`end`**`Index`.
`public char[] toCharArray()` Convierte esta cadena a un arreglo de caracteres. **Regresa:** Un arreglo cuya longitud es el tamaño de esta cadena y cuyo contenido se inicializa a la secuencia de caracteres de esta cadena.
`public String toLowerCase()` Convierte los caracteres de esta cadena a minúsculas. **Regresa:** Una nueva cadena con los caracteres de esta cadena convertidos a minúsculas.
`public String toUpperCase()` Convierte los caracteres de esta cadena a mayúsculas. **Regresa:** Una nueva cadena con los caracteres de esta cadena convertidos a mayúsculas.

Tabla 5.1 Algunos Métodos de la Clase String. Continuación.

`public String trim()` Regresa una copia de esta cadena eliminando los caracteres blancos iniciales y finales. Un carácter blanco es aquel cuyo código Unicode (o ASCII) es menor o igual a 32, **Regresa:** Una copia de esta cadena eliminando los caracteres blancos iniciales y finales.

Clase `StringBuffer`

Los objetos de tipo `StringBuffer` son cadenas como las del tipo `String` pero su contenido puede modificarse invocando una serie de métodos.

Tabla 5.2 Algunos Métodos de la Clase `StringBuffer`

`public StringBuffer()` Construye una instancia de `StringBuffer` vacío y con una capacidad inicial de 16 caracteres.
`public StringBuffer(int length)` Construye una instancia de `StringBuffer` vacío y con una capacidad inicial dada por el parámetro. **Parámetro:** `length` **–** Capacidad inicial **Lanza:** `NegativeArraySizeException` – Si el parámetro es negativo.
`public StringBuffer(String str)` Construye una instancia de `StringBuffer` con la secuencia de caracteres contenida en el parámetro. **Parámetro:** `str` – El contenido inicial de la instancia de `StringBuffer.` **Lanza:** `NullPointerException` – Si `str` es nulo.
`public StringBuffer append(boolean b)` **`public StringBuffer append(char c)`** **`public StringBuffer append(char [] str)`** **`public StringBuffer append(double)`** **`public StringBuffer append(float f)`** **`public StringBuffer append(int i)`** **`public StringBuffer append(long l)`** **`public StringBuffer append(Object o)`** **`public StringBuffer append(String str)`** **`public StringBuffer append(StringBuffer sb)`** Le agrega a esta cadena, una cadena con la representación del parámetro. **Regresa:** Una referencia a esta cadena

Tabla 5.2 Algunos Métodos de la Clase `StringBuffer`. Continuación

`public StringBuffer delete(int start, int end)` Elimina los caracteres de una subcadena de esta cadena. La subcadena empieza con el carácter en el índice dado por el parámetro `start` y se extiende hasta el carácter en el índice `end` **-** 1. **Parámetro:** `start` – Índice inicial de la subcadena inclusive. **end** – Índice final de la subcadena exclusive. **Regresa:** Una referencia a esta cadena. **Lanza:** `StringIndexOutOfBoundsException` – Si `start` es negativo, mayor que la longitud de esta cadena o mayor que **end**.
`public StringBuffer deleteCharAt(int index)` Elimina de esta cadena el carácter con índice dado por el parámetro. **Parámetro:** `index` – Índice del carácter a eliminar. **Regresa:** Una referencia a esta cadena. **Lanza:** `StringIndexOutOfBoundsException` – Si `index` es negativo, mayor o igual que la longitud de esta cadena.
`public int indexOf(String str)` Regresa el índice de la primera ocurrencia dentro de esta cadena de la subcadena dada por el parámetro `str`. Si `str` no está en esta cadena regresa -1. **Parámetro:** `str` – Subcadena a buscar. **Regresa:** Índice de la primera ocurrencia dentro de esta cadena de la subcadena dada por el parámetro `str`. Si `str` no está en esta cadena regresa -1. **Lanza:** `NullPointerException` – Si la cadena dada por el parámetro es `null`.

Tabla 5.2 Algunos Métodos de la Clase StringBuffer. Continuación

`public StringBuffer insert(int offset, boolean b)`
`public StringBuffer insert(int offset, char c)`
`public StringBuffer insert(int offset, char[] str)`
`public StringBuffer insert(int offset, double d)`
`public StringBuffer insert(int offset, float f)`
`public StringBuffer insert(int offset, int i)`
`public StringBuffer insert(int offset, long l)`
`public StringBuffer insert(int offset, Object obj)`
`public StringBuffer insert(int offset, String str)`

Inserta en esta cadena una subcadena con la representación del segundo parámetro en la posición dada por el parámetro `offset`.

Parámetro:
`offset` – Posición donde se agregará la subcadena.

Regresa:
Una referencia a esta cadena.

Lanza:
`StringIndexOutOfBoundsException` – Si `start` es inválido.

`public int length()`

Regresa la longitud de esta cadena, esto es, el número de caracteres.

Regresa:
La longitud de esta cadena

`public StringBuffer reverse()`

Invierte los caracteres de la cadena. Se intercambian los caracteres con índice `0` y `length()` – 1, los caracteres con índice `1` y `length()` – 2, etc.

Regresa:
Una referencia a esta cadena.

`public void setCharAt(int index, char ch)`

Reemplaza el carácter de esta cadena con índice dado por el parámetro `index`, por el carácter dado por el parámetro ch.

Parámetro:
`index` – Índice del carácter a reemplazar.
`ch` – Caracter nuevo

Lanza:
`IndexOutOfBoundsException` – Si `index` es negativo o mayor o igual que la longitud de esta cadena.

Tabla 5.2 Algunos Métodos de la Clase StringBuffer. Continuación

`public String substring(int start)` Regresa una nueva cadena de tipo `String` con la secuencia de caracteres de esta cadena a partir del carácter con índice `start` y hasta el final de esta cadena. **Parámetro:** **`start`** – Índice del carácter a reemplazar. **Regresa:** Una nueva cadena de tipo `String`. **Lanza:** `StringIndexOutOfBoundsException` – Si `start` es negativo o mayor que la longitud de esta cadena.
`public String substring(int start, int end)` Regresa una nueva cadena de tipo `String` con la secuencia de caracteres de esta cadena a partir del carácter con índice `start` y hasta el carácter con índice `end` – 1 de esta cadena. **Parámetro:** **`start`** – Índice inicial, inclusive. **`end`** – Índice final, exclusive. **Regresa:** Una nueva cadena de tipo `String`. **Lanza:** `StringIndexOutOfBoundsException` – Si `start` o `end` son negativos, o mayores que la longitud de esta cadena o `start` es mayor que `end`.

Clase `Vector`

La clase `Vector` implementa un arreglo de objetos. Los elementos pueden ser accedidos usando un índice entero. El tamaño del vector puede crecer o disminuir en forma dinámica al agregar o eliminar elementos una vez que el vector ha sido creado. La clase `Vector` se encuentra en el paquete `java.util`.

La tabla 5.3 muestra los atributos de la clase `Vector`:

Tabla 5.3 Atributos de la Clase Vector

Atributo	Descripción
`protected int capacityIncrement`	Cantidad por la que la capacidad de un vector se incrementa automáticamente cuando su tamaño es mayor que su capacidad.
`protected int elementCount`	Número de componentes en el Vector
`protected object[]`	El arreglo en el que se almacenan las componentes del vector

La tabla 5.4 muestra algunos de los métodos de la clase Vector:

Tabla 5.4 Algunos Métodos de la Clase Vector

`public Vector(int initialCapacity, int capacityIncrement)` Construye un vector vacío con la capacidad inicial e incremento de la capacidad dadas por sus parámetros. **Parámetros:** `initialCapacity` **-** Capacidad inicial del vector `capacityIncrement` **–** Cantidad en la que se incrementa la capacidad del vector cuando se desborda. **Lanza:** `IllegalArgumentException` – Si la capacidad inicial es negativa.
`public Vector(int initialCapacity)` Construye un vector vacío con la capacidad inicial dada por su parámetro e incremento de la capacidad de 0. **Parámetro:** `initialCapacity` **-** Capacidad inicial del vector **Lanza:** `IllegalArgumentException` – Si la capacidad inicial es negativa.
`public Vector()` Construye un vector vacío con la capacidad inicial de 10 e incremento de la capacidad de 0.
`public void add(int index, Object element)` Inserta el elemento dado por el parámetro `element` en el vector en la posición dada por `index`. Recorriendo los elementos restantes una posición a la derecha. **Parámetros**: `index` – Posición donde se insertará el elemento. `element` – Elemento a insertar. **Lanza:** `ArrayIndexOutOfBoundsException` – Si el índice está fuera de rango (index < 0 || index > size()).
`public boolean add(Object o)` Agrega el elemento dado por el parámetro al final de la lista. **Parámetro:** **`o`** – Elemento a agregarse al final de la lista. **Regresa:** `true` si se agregó el elemento, `false` en caso contrario.

Tabla 5.4 Algunos Métodos de la Clase Vector. Continuación

`public boolean contains(Object elem)` Prueba si el objeto dado por el parámetro es un componente del vector. **Parámetro:** `elem` – Un objeto **Regresa:** `true` si y sólo sí el objeto dado por el parámetro es el mismo que un elemento del vector, comparado con el método equals(); `false` en caso contrario.
`public Object elementAt(int index)` Regresa el componente en la posición dada por índice. **Parámetro:** `index` – Índice del elemento a regresar. **Regresa:** El elemento en la posición dada por el parámetro. **Lanza:** `ArrayIndexOutOfBoundsException` – Si el índice está fuera de rango (index < 0 || index > size()).
`public Object firstElement()` Regresa el primer elemento (posición 0) del vector. **Regresa:** El primer elemento del vector. **Lanza:** `NoSuchElementException` – Si el vector está vacío.
`public int indexOf(Object elem)` Busca la primera ocurrencia de elemento dado por el parámetro, comparado con el método equals(); `false` en caso contrario. **Parámetro:** `elem` – Un objeto **Regresa:** El índice de la primera ocurrencia de elemento dado por el parámetro
`public boolean isEmpty()` Prueba si el vector está vacío. **Regresa:** `true` si y sólo sí el vector no tiene componentes, `false` en caso contrario.

Tabla 5.4 Algunos Métodos de la Clase Vector. Continuación

`public Object lastElement()`

Regresa el último elemento (posición `size() – 1`) del vector.

Regresa:
El último elemento del vector.

Lanza:
`NoSuchElementException` – Si el vector está vacío.

`public Object remove(int index)`

Extrae el elemento en la posición dada por el parámetro. Recorriendo los elementos restantes una posición a la izquierda.

Parámetro:
`index` – Índice del elemento a extraer.

Regresa:
El elemento extraído.

Lanza:
`ArrayIndexOutOfBoundsException` – Si el índice está fuera de rango (index < 0 || index > size()).

`public boolean remove(Object o)`

Elimina la primera ocurrencia del elemento dado por el parámetro.

Parámetro:
o - Elemento a eliminar.

Regresa:
`true` si el vector contenía al elemento, `false` en caso contrario.

`public void setElementAt(Object obj, int index)`

Reemplaza el elemento en la posición dada por el parámetro `index` por el objeto dado por el parámetro `obj`.

Parámetros:
`obj` – Nuevo elemento.
`index` – Posición del elemento a reemplazar.

Lanza:
`ArrayIndexOutOfBoundsException` – Si el índice está fuera de rango (index < 0 || index > size()).

`public int size()`

Regresa el número de componentes en este vector.

Regresa:
El número de componentes en este vector.

Tabla 5.4 Algunos Métodos de la Clase Vector. Continuación

public String toString()
Regresa una cadena con la representación de este vector, conteniendo la representación de cada elemento. **Regresa:** Una cadena con la representación de este vector

Ejemplos Sobre Vectores

En el siguiente fragmento de código se muestra la implementación en la clase `Medios` de los métodos que permiten listar los medios del catálogo de medios, los medios que sean del mismo tipo, los medios que sean de un género dado y los medios que sean de un período dado:

Medios.java

```
/*
 * Medios.java
 *
 * Created on 15 de septiembre de 2007, 12:21 PM
 */

package persistencia;

import java.util.Vector;

import objetosServicio.*;
import objetosNegocio.Medio;

/**
 * Esta clase permite almacenar, actualizar, eliminar y consultar
 * medios (canciones y películas) del programa AmanteMusica.
 * Los datos se almacenan en arreglos.
 *
 * @author mdomitsu
 */
public class Medios {
  protected Medio medios[];
  protected int numMedios = 0;

  ...

  /**
   * Regresa la lista de todos los medios.
   * @return Lista de todos los medios
   */
  public Vector lista() {
    Vector lista = new Vector();

    // Recorre el arreglo
    for(int i = 0; i < numMedios; i++) {
      // Agrega el medio a la lista
      lista.add(medios[i]);
```

```
    }

    return lista;
  }

  /**
   * Regresa la lista de los medios del mismo título que el parámetro
   * @param titulo Título de los medios a listar
   * @return Lista de medios del mismo título que el parámetro
   */
  public Vector listaTitulo(String titulo) {
    Vector lista = new Vector();

    // Recorre el arreglo
    for(int i = 0; i < numMedios; i++) {
      // Si es el título especificado
      if(titulo.equals(medios[i].getTitulo()))
        // Agrega el medio a la lista
        lista.add(medios[i]);
    }

    return lista;
  }

  /**
   * Crea la lista de los medios del mismo género que el parámetro
   * @param cveGenero Clave del género de los medios a listar
   * @return Lista de medios del mismo género que el parámetro
   */
  public Vector listaGenero(String cveGenero) {
    Vector lista = new Vector();

    // Recorre el arreglo
    for(int i = 0; i < numMedios; i++) {
      // Si es el género especificado
      if(cveGenero.equals(medios[i].getGenero().getCveGenero()))
        // Agrega el medio a la lista
        lista.add(medios[i]);
    }

    return lista;
  }

  /**
   * Crea la lista de los medios del mismo periodo que el parámetro
   * @param periodo Periodo de los medios a listar
   * @return Lista de los medios del mismo periodo que el parámetro
   */
   public Vector listaPeriodo(Periodo periodo) {
    Vector lista = new Vector();

    // Recorre el arreglo
    for(int i = 0; i < numMedios; i++) {
      // Si es el periodo especificado
      if(periodo.contiene(medios[i].getFecha()))
        // Agrega el medio a la lista
        lista.add(medios[i]);
```

```
    }

    return lista;
  }
}
```

En el siguiente fragmento de código se muestra la implementación en la clase `Canciones` de los métodos que permiten listar las canciones que sean de un intérprete dado, las canciones que sean de un autor dado y listar las canciones que sean de un álbum dado:

Canciones.java

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

import java.util.Vector;

import objetosNegocio.Cancion;

/**
 * Esta clase permite consultar canciones del programa AmanteMusica
 * Los datos se almacenan en arreglos.
 *
 * @author mdomitsu
 */
public class Canciones extends Medios {
  // Crea el arreglo para almacenar las canciones
  private Cancion canciones[];

  ...

  /**
   * Regresa la lista de canciones con el mismo interprete que el parámetro
   * @param interprete Interprete de las canciones a listar
   * @return La lista de canciones con el mismo interprete que el parámetro
   */
  public Vector listaInterprete(String interprete) {
    Vector lista = new Vector();

    // Recorre el arreglo
    for(int i = 0; i < numMedios; i++) {
      // Si es el intérprete especificado
      if(interprete.equals(canciones[i].getInterprete()))
        // Agrega la canción a la lista
        lista.add(canciones[i]);
    }

    return lista;
  }

  /**
```

```
   * Regresa la lista de canciones con el mismo autor que el parámetro
   * @param autor Autor de las canciones a listar
   * @return La lista de canciones con el mismo autor que el parámetro
   */
  public Vector listaAutor(String autor) {
    Vector lista = new Vector();

    // Recorre el arreglo
    for(int i = 0; i < numMedios; i++) {
      // Si es el autor especificado
      if(autor.equals(canciones[i].getAutor()))
        // Agrega la canción a la lista
        lista.add(canciones[i]);
    }
    return lista;
  }

  /**
   * Regresa la lista de canciones del mismo álbum que el parámetro
   * @param album Álbum de las canciones a listar
   * @return La lista de canciones del mismo álbum
   */
  public Vector listaAlbum(String album) {
    Vector lista = new Vector();

    // Recorre el arreglo
    for(int i = 0; i < numMedios; i++) {
      // Si es el álbum especificado
      if(album.equals(canciones[i].getAlbum()))
        // Agrega la canción a la lista
        lista.add(canciones[i]);
    }

    return lista;
  }
}
```

En el siguiente fragmento de código se muestra la implementación en la clase `Peliculas` de los métodos que permiten listar las películas que sean de un actor dado y listar las películas que sean de un director dado:

Peliculas.java

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

import java.util.Vector;

import objetosNegocio.Pelicula;

/**
 * Esta clase permite consultar películas del programa AmanteMusica
```

```
 * Los datos se almacenan en arreglos.
 *
 * @author mdomitsu
 */
public class Peliculas extends Medios {
  // Arreglo para almacenar las películas
  private Pelicula peliculas[];

  ...

  /**
   * Regresa la lista de peliculas con el mismo actor que el parámetro
   * @param actor Actor de las películas a listar
   * @return La lista de peliculas con el mismo actor que el parámetro
   */
  public Vector listaActor(String actor) {
    Vector lista = new Vector();

    // Recorre el arreglo
    for(int i = 0; i < numMedios; i++) {
      // Si es el actor especificado
      if(peliculas[i].getActor1().equals(actor) ||
         peliculas[i].getActor2().equals(actor))
        // Agrega la película a la lista
        lista.add(peliculas[i]);
    }

    return lista;
  }

  /**
   * Regresa la lista de peliculas con el mismo director que el parámetro
   * @param diretor Director de las películas a listar
   * @return La lista de peliculas con el mismo director que el parámetro
   */
  public Vector listaDirector(String director) {
    Vector lista = new Vector();

    // Recorre el arreglo
    for(int i = 0; i < numMedios; i++) {
      // Si es el director especificado
      if(director.equals(peliculas[i].getDirector()))
        // Agrega la película a la lista
        lista.add(peliculas[i]);
    }

    return lista;
  }
}
```

En el siguiente fragmento de código se muestra la implementación en la clase `Generos` del método que permite listar los géneros:

Generos.java

```
/*
 * Generos.java
 *
 * Creada el 15 de septiembre de 2007, 11:40 AM
 */

package persistencia;

import java.util.Vector;

import objetosNegocio.Genero;

/**
 * Esta clase permite almacenar, actualizar, eliminar y consultar
 * géneros de canciones y películas del programa AmanteMusica.
 * Los datos se almacenan en arreglos.
 *
 * @author mdomitsu
 */
public class Generos {
  protected Genero generos[];
  protected int numGeneros = 0;

  ...

  /**
   * Regresa una lista de todos los géneros.
   * @return Lista de todos los géneros
   */
  public Vector lista() {
    Vector lista = new Vector();

    // Recorre el arreglo
    for(int i = 0; i < numGeneros; i++) {
      // Agrega el género a la lista
      lista.add(generos[i]);
    }

    return lista;
  }

  /**
   * Crea la lista de los géneros del mismo tipo de medio que el parámetro
   * @param tipoMedio Tipo del medio de los géneros a listar
   * @return Lista de los géneros del mismo tipo de medio que el parámetro
   */
  public Vector listaMedio(char tipoMedio) {
    Vector lista = new Vector();

    // Recorre el arreglo
    for(int i = 0; i < numGeneros; i++) {
      // Si es el medio especificado
      if(tipoMedio == generos[i].getTipoMedio())
        // Agrega el medio a la lista
        lista.add(generos[i]);
```

```
    }

    return lista;
  }
}
```

Diseño en Capas y la Clase Fachada

En el Tema 4: Excepciones, se habló del diseño en capas. Las clases `Medios`, `Canciones` y `Peliculas` que emplean arreglos para almacenar objetos de tipo `Cancion`, `Pelicula` y `Genero` en el programa sobre el Amante de la Música y del Cine conforman la capa inferior de la aplicación y en este tipo de aplicaciones puede suceder lo siguiente:

- La capa superior de la aplicación sólo va a acceder a una parte de los métodos de una capa inferior y desearíamos que el desarrollador de la capa superior sólo tuviera acceso a los métodos que necesita, así no tendrá que preocuparse por el resto de métodos que no requiere.
- La capa superior de la aplicación requiere de ciertas transacciones que involucren varios métodos de las clases de la capa inferior. Por ejemplo, podríamos tener una restricción de que no se permiten claves repetidas en los arreglos de canciones o películas. El método para agregar un medio en la clase medio no nos proporciona la verificación.
- Los datos que se desean almacenar pueden utilizar diferentes mecanismos, por ejemplo arreglos, archivos y base de datos. Aunque en una aplicación real almacenar los datos en arreglos no es una solución adecuada ya que los arreglos se crean en la memoria RAM de la computadora y al salir de la aplicación o al apagarse la computadora se destruyen, perdiéndose su información.

Como solución a los problemas anteriores podríamos construir una capa intermedia que tuviera uno o más módulos, cada uno se encargará de lo siguiente:

- Mostrar sólo los métodos de la capa inferior que la capa superior requiere.
- Manejar las transacciones.
- Ocultarle a la capa superior el mecanismo empleado para almacenar los datos, proporcionándole a la capa superior el mismo mecanismo para acceder a los datos independientemente de cómo se almacenan los datos. Esto permite modificar o sustituir los métodos de la capa inferior sin que le afecte a la capa superior.
- Traducir los mensajes de errores de la capa inferior a mensajes más amigables para la capa superior, conservando la información de los errores para fines de depuración.

A los módulos que realizan las tareas anteriores se conocen como Fachadas. Cada fachada estará formada por:

- Una interfaz que establezca qué métodos de la capa inferior mostrará la fachada
- Una o más clases que implementen la fachada anterior. Una para cada mecanismo de almacenamiento.
- Una Excepción que envolverá a las excepciones de la capa inferior.

El diagrama de clases de la figura 5.2 muestra la interfaz y la fachada que implementa los métodos declarados por la interfaz para programa sobre el amante de la música y el cine, cuando el almacenamiento de los datos es en arreglos:

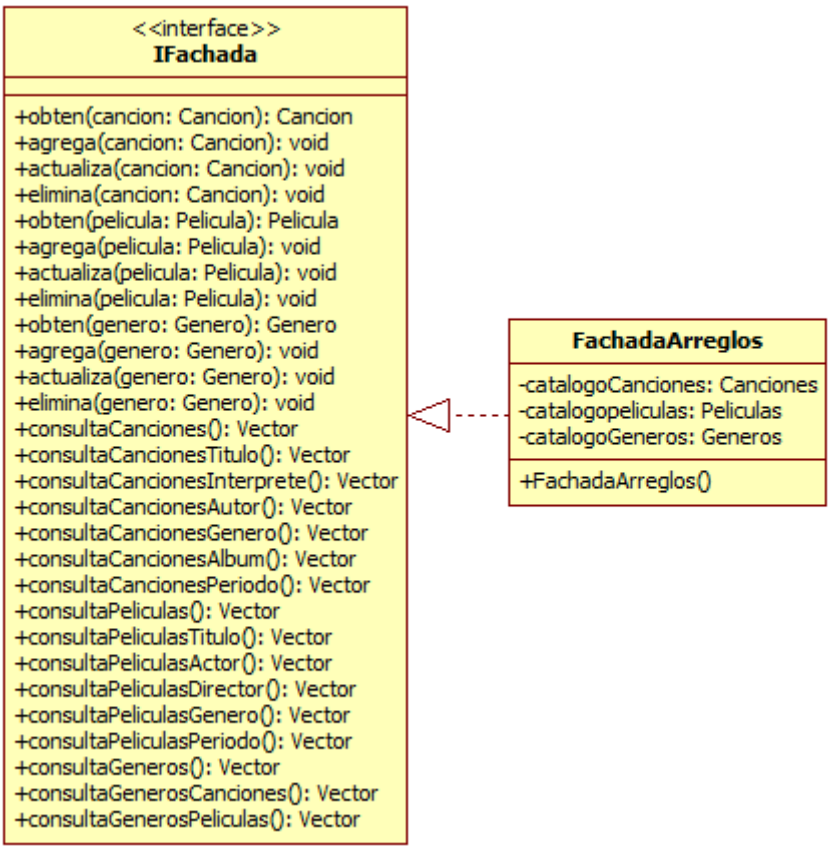

Figura 5.2 Fachada del Programa AmanteMusica, Versión Arreglos

El siguiente código muestra la clase `FachadaException` que será la Excepción que envuelva las excepciones de la capa inferior del programa sobre el amante de la música:

FachadaException.java

```
/*
 * FachadaException.java
 *
 * Creada el 13 de septiembre de 2007, 12:01 AM
 */

package excepciones;

/**
 * Esta clase representa a las excepciones lanzadas por las
 * clases Fachada y envuelve a las excepciones lanzadas por
 * las clases que encapsulan la persistencia.
 *
 * @author mdomitsu
 */
public class FachadaException extends RuntimeException {

  /**
   * Construye una excepción con un mensaje de error nulo.
   */
  public FachadaException() {
  }

  /**
   * Construye una excepción con el mensaje de error del parámetro.
   * @param msj Mensaje de error.
   */
  public FachadaException(String msj) {
    super(msj);
  }

  /**
   * Construye una excepción con el mensaje de error del parámetro y la causa
   * original del error.
   * @param msj Mensaje de error.
   * @param causa Causa original del error.
   */
  public FachadaException(String msj, Throwable causa) {
    super(msj, causa);
  }

  /**
   * Construye una excepción la causa original del error.
   * @param causa Causa original del error.
   */
  public FachadaException(Throwable causa) {
    super(causa);
  }
}
```

El siguiente código muestra la interfaz de la fachada, `IFachada`, de la capa inferior del programa sobre el amante de la música:

IFachada. java

```
/*
 * IFachada.java
 *
 * Creada el 13 de septiembre de 2007, 12:01 AM
 */
package interfaces;

import java.util.Vector;

import objetosServicio.*;
import objetosNegocio.*;
import excepciones.FachadaException;

/**
 * Esta Interfaz establece los métodos que deben implementar las clases
 * que encapsulen el mecanismo de persistencia del programa AmanteMusica
 *
 * @author mdomitsu
 */
public interface IFachada {

  /**
   * Obtiene una canción del catálogo de canciones
   * @param cancion Cancion a obtener
   * @return La canción si existe, null en caso contrario
   * @throws FachadaException Si no se puede acceder al catálogo
   * de canciones.
   */
  public Cancion obten(Cancion cancion) throws FachadaException;

  /**
   * Agrega una canción al catálogo de canciones. No se permiten canciones
   * con claves repetidas
   * @param cancion Cancion a agregar
   * @throws FachadaException Si no se puede agregar la canción al
   * catálogo de canciones.
   */
  public void agrega(Cancion cancion) throws FachadaException;

  /**
   * Actualiza una canción del catálogo de canciones
   * @param cancion Cancion a actualizar
   * @throws FachadaException Si no se puede actualizar la canción
   * del catálogo de canciones.
   */
  public void actualiza(Cancion cancion) throws FachadaException;

  /**
   * Elimina una canción del catálogo de canciones
   * @param cancion Cancion a eliminar
   * @throws FachadaException Si no se puede eliminar la canción
   * del catálogo de canciones.
   */
  public void elimina(Cancion cancion) throws FachadaException;
```

```
  /**
   * Obtiene una película del catálogo de peliculas
   * @param pelicula Pelicula a obtener
   * @return La película si existe, null en caso contrario
   * @throws FachadaException Si no se puede acceder al
   * catálogo de películas.
   */
  public Pelicula obten(Pelicula pelicula) throws FachadaException;

  /**
   * Agrega una película al catálogo de películas. No se permiten películas
   * con claves repetidas

   * @param pelicula Pelicula a agregar
   * @throws FachadaException Si no se puede agregar la película al
   * catálogo de películas.
   */
  public void agrega(Pelicula pelicula) throws FachadaException;

  /**
   * Actualiza una película del catálogo de peliculas
   * @param pelicula Pelicula a actualizar
   * @throws FachadaException Si no se puede actualizar la película
   * del catálogo de películas.
   */
  public void actualiza(Pelicula pelicula) throws FachadaException;

  /**
   * Elimina una película del catálogo de peliculas
   * @param pelicula Pelicula a eliminar
   * @throws FachadaException Si no se puede eliminar la película
   * del catálogo de películas.
   */
  public void elimina(Pelicula pelicula) throws FachadaException;

  /**
   * Obtiene un género del catálogo de géneros
   * @param genero Género a obtener
   * @return El género si existe, null en caso contrario
   * @throws FachadaException Si no se puede acceder al
   * catálogo de géneros.
   */
  public Genero obten(Genero genero) throws FachadaException;

  /**
   * Agrega un género al catálogo de géneros. No se permiten géneros
   * con claves repetidas
   * @param genero Género a agregar
   * @throws FachadaException Si no se puede agregar el género al
   * catálogo de géneros.
   */
  public void agrega(Genero genero) throws FachadaException;

  /**
   * Actualiza un género del catálogo de géneros
   * @param genero Género a actualizar
   * @throws FachadaException Si no se puede actualizar el género del
```

```
   * catálogo de géneros.
   */
  public void actualiza(Genero genero) throws FachadaException;

  /**
   * Elimina un género del catálogo de géneros
   * @param genero Género a eliminar
   * @throws FachadaException Si no se puede eliminar el género del
   * catálogo de géneros.
   */
  public void elimina(Genero genero) throws FachadaException;

  /**
   * Obtiene una lista todas las canciones
   * @return Vector con la lista de todas las canciones
   * @throws FachadaException Si no se puede acceder al catálogo
   * de canciones.
   */
  public Vector consultaCanciones() throws FachadaException;

  /**
   * Obtiene una lista de todas las canciones con el mismo título.
   * @param titulo Titulo de las canciones de la lista
   * @return Vector con la lista de todas las canciones con el mismo
   * título.
   * @throws FachadaException Si no se puede acceder al catálogo
   * de canciones.
   */
  public Vector consultaCancionesTitulo(String titulo)
    throws FachadaException;

  /**
   * Obtiene una lista de todas las canciones con el mismo intérprete.
   * @param interprete Intérprete de las canciones de la lista
   * @return Vector con la lista de todas las canciones con el mismo
   * intérprete.
   * @throws FachadaException Si no se puede acceder al catálogo
   * de canciones.
   */
  public Vector consultaCancionesInterprete(String interprete)
    throws FachadaException;

  /**
   * Obtiene una lista de todas las canciones con el mismo autor.
   * @param autor Autor de las canciones de la lista.
   * @return Vector con la lista de todas las canciones con el mismo autor.
   * @throws FachadaException Si no se puede acceder al catálogo
   * de canciones.
   */
  public Vector consultaCancionesAutor(String autor)
    throws FachadaException;

  /**
   * Obtiene una lista de todas las canciones con el mismo género.
   * @param cveGenero Clave del género de las canciones de la lista
   * @return Vector con la lista de todas las canciones con el mismo
   * género.
```

```
   * @throws FachadaException Si no se puede acceder al catálogo
   * de canciones.
   */
  public Vector consultaCancionesGenero(String cveGenero)
    throws FachadaException;

  /**
   * Obtiene una lista de todas las canciones del mismo álbum.
   * @param album Álbum de las canciones de la lista
   * @return Vector con la lista de todas las canciones del mismo álbum.
   * @throws FachadaException Si no se puede acceder al catálogo
   * de canciones.
   */
  public Vector consultaCancionesAlbum(String album) throws FachadaException;

  /**
   * Obtiene una lista de todas las canciones del mismo periodo.
   * @param periodo Período de las canciones de la lista
   * @return Vector con la lista de todas las canciones del mismo período.
   * @throws FachadaException Si no se puede acceder al catálogo
   * de canciones.
   */
  public Vector consultaCancionesPeriodo(Periodo periodo)
    throws FachadaException;

  /**
   * Obtiene una lista de todas las películas
   * @return Vector con la lista de todas las películas.
   * @throws FachadaException Si no se puede acceder al catálogo
   * de películas.
   */
  public Vector consultaPeliculas() throws FachadaException;

  /**
   * Obtiene una lista de todas las películas del mismo título
   * @param titulo Título de las películas de la lista
   * @return Vector con la lista de todas las peliculas del mismo título.
   * @throws FachadaException Si no se puede acceder al catálogo
   * de películas.
   */
  public Vector consultaPeliculasTitulo(String titulo)
    throws FachadaException;

  /**
   * Obtiene una lista de todas las películas del mismo actor
   * @param actor Actor de las peliculas de la lista
   * @return Vector con la lista de todas las peliculas del mismo actor.
   * @throws FachadaException Si no se puede acceder al catálogo
   * de películas.
   */
  public Vector consultaPeliculasActor(String actor) throws FachadaException;

  /**
   * Obtiene una lista de todas las películas del mismo director
   * @param director Director de las peliculas de la lista
   * @return Vector con la lista de todas las peliculas del mismo director.
   * @throws FachadaException Si no se puede acceder al catálogo
```

```
   * de películas.
   */
  public Vector consultaPeliculasDirector(String director)
    throws FachadaException;

  /**
   * Obtiene una lista de todas las películas del mismo género
   * @param cveGenero Clave del género de las películas de la lista
   * @return Vector con la lista de todas las películas del mismo género.
   * @throws FachadaException Si no se puede acceder al catálogo
   * de películas.
   */
  public Vector consultaPeliculasGenero(String cveGenero)
   throws FachadaException;

  /**
   * Obtiene una lista de todas las películas del mismo periodo
   * @param periodo Periodo de las peliculas de la lista
   * @return Vector con la lista de todas las peliculas del mismo periodo.
   * @throws FachadaException Si no se puede acceder al catálogo
   * de películas.
   */
  public Vector consultaPeliculasPeriodo(Periodo periodo)
    throws FachadaException;

  /**
   * Obtiene una lista de todos los géneros
   * @return Vector con la lista de todos los géneros.
   * @throws FachadaException Si no se puede acceder al catálogo
   * de géneros.
   */
  public Vector consultaGeneros() throws FachadaException;

  /**
   * Obtiene una lista de los géneros de canciones
   * @return Vector con la lista de los géneros canciones
   * @throws FachadaException Si no se puede acceder al catálogo
   * de géneros.
   */
  public Vector consultaGenerosCanciones() throws FachadaException;

  /**
   * Obtiene una lista de los géneros de películas
   * @return Vector con la lista de los géneros películas
   * @throws FachadaException Si no se puede acceder al catálogo
   * de géneros.
   */
  public Vector consultaGenerosPeliculas() throws FachadaException;
}
```

El siguiente código muestra la clase Fachada del programa sobre el amante de la música, cuando el almacenamiento de los datos es en arreglos. Esta clase implementa la interfaz `IFachada`. Hay que notar que aunque en la interfaz `IFachada` se declararon que todos los métodos lanzan una excepción del tipo `FachadaException`,

en las implementaciones de los métodos no se requiere que los métodos declaren ni lancen la excepción.

FachadaArreglos.java

```
/*
 * FachadaArreglos.java
 *
 * Created on 15 de septiembre de 2007, 12:21 PM
 */

package fachadas;

import java.util.Vector;

import objetosServicio.*;
import objetosNegocio.*;
import excepciones.*;
import interfaces.IFachada;
import persistencia.*;

/**
 * Esta clase implementa la interfaz IFachada del mecanismo de persistencia
 * de arreglos del programa AmanteMusica
 *
 * @author mdomitsu
 */
public class FachadaArreglos implements IFachada {
  private Generos catalogoGeneros;
  private Canciones catalogoCanciones;
  private Peliculas catalogoPeliculas;

  /**
   * Este constructor crea los objetos con los arreglos para
   * almacenar los géneros, canciones y películas
   */
  public FachadaArreglos() {
    // Crea un objeto del tipo Generos para accesar a la tabla canciones
    catalogoGeneros = new Generos(50);

    // Crea un objeto del tipo Canciones para accesar a la tabla canciones
    catalogoCanciones = new Canciones(1000);

    // Crea un objeto del tipo Peliculas para accesar a la tabla peliculas
    catalogoPeliculas = new Peliculas(100);
  }

  /**
   * Obtiene una canción del catálogo de canciones
   * @param cancion Cancion a obtener
   * @return La canción si existe, null en caso contrario
   */
  public Cancion obten(Cancion cancion) {
    // Obten la canción
    return catalogoCanciones.obten(cancion);
  }
```

```
  /**
   * Agrega una canción al catálogo de canciones. No se permiten canciones
   * con claves repetidas
   * @param cancion Cancion a agregar
   * @throws FachadaException Si no se puede agregar la canción al
   * catálogo de canciones.
   */
  public void agrega(Cancion cancion) throws FachadaException {
    Cancion cancionBuscada;

    // Busca la canción en el arreglo con la misma clave.
    cancionBuscada = catalogoCanciones.obten(cancion);

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

    // Agrega la nueva canción al catálogo
    try {
      catalogoCanciones.agrega(cancion);
    }
    catch(PersistenciaException pe) {
      // Aquí se envuelve la excepción PersistenciaException
      // en la excepción FachadaException y se relanza
      throw new FachadaException("No se puede agregar la canción", pe);
    }
  }

  /**
   * Actualiza una canción del catálogo de canciones
   * @param cancion Cancion a actualizar
   * @throws FachadaException Si no se puede actualizar la canción del
   * catálogo de canciones.
   */
  public void actualiza(Cancion cancion) throws FachadaException {
    try {
      // Actualiza la  canción del catálogo
      catalogoCanciones.actualiza(cancion);
    }
    catch(PersistenciaException pe) {
      // Aquí se envuelve la excepción PersistenciaException
      // en la excepción FachadaException y se relanza
      throw new FachadaException("No se puede actualizar la canción", pe);
    }
  }

  /**
   * Elimina una canción del catálogo de canciones
   * @param cancion Cancion a eliminar
   * @throws FachadaException Si no se puede eliminar la canción
   * del catálogo de canciones.
   */
  public void elimina(Cancion cancion) throws FachadaException {
    // Elimina la  canción del catálogo
    try {
      catalogoCanciones.elimina(cancion);
    }
```

```
    catch(PersistenciaException pe) {
      // Aquí se envuelve la excepción PersistenciaException
      // en la excepción FachadaException y se relanza
      throw new FachadaException("No se puede eliminar la canción", pe);
    }
  }

  /**
   * Obtiene una película del catálogo de películas
   * @param pelicula Pelicula a obtener
   * @return La película si existe, null en caso contrario
   */
  public Pelicula obten(Pelicula pelicula) {
    // Obten la película
    return catalogoPeliculas.obten(pelicula);
  }

  /**
   * Agrega una película al catálogo de películas. No se permiten películas
   * con claves repetidas
   * @param pelicula Pelicula a agregar
   * @throws FachadaException Si no se puede agregar la película al
   * catálogo de películas.
   */
  public void agrega(Pelicula pelicula) throws FachadaException {
    Pelicula peliculaBuscada;

    // Busca la película en el arreglo con la misma clave.
    peliculaBuscada = catalogoPeliculas.obten(pelicula);

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

    // Agrega la nueva película al catálogo
    try {
      catalogoPeliculas.agrega(pelicula);
    }
    catch(PersistenciaException pe) {
      // Aquí se envuelve la excepción PersistenciaException
      // en la excepción FachadaException y se relanza
      throw new FachadaException("No se puede agregar la película", pe);
    }
  }

  /**
   * Actualiza una película del catálogo de peliculas
   * @param pelicula Pelicula a actualizar
   * @throws FachadaException Si no se puede actualizar la película
   * del catálogo de películas.
   */
  public void actualiza(Pelicula pelicula) throws FachadaException {
    // Actualiza la  película del catálogo
    try {
      catalogoPeliculas.actualiza(pelicula);
    }
    catch(PersistenciaException pe) {
```

```
      // Aquí se envuelve la excepción PersistenciaException
      // en la excepción FachadaException y se relanza
      throw new FachadaException("No se puede actualizar la película", pe);
    }
  }

  /**
   * Elimina una película del catálogo de películas
   * @param pelicula Película a eliminar
   * @throws FachadaException Si no se puede eliminar la película
   * del catálogo de películas.
   */
  public void elimina(Pelicula pelicula) throws FachadaException {
    // Elimina la  película del catálogo
    try {
      catalogoPeliculas.elimina(pelicula);
    }
    catch(PersistenciaException pe) {
      // Aquí se envuelve la excepción PersistenciaException
      // en la excepción FachadaException y se relanza
      throw new FachadaException("No se puede eliminar la película", pe);
    }
  }

  /**
   * Obtiene un género del catálogo de géneros
   * @param genero Género a obtener
   * @return El género si existe, null en caso contrario
   * @throws FachadaException Si no se puede acceder al
   * catálogo de géneros.
   */
  public Genero obten(Genero genero) throws FachadaException {
    // Obten el género
    return catalogoGeneros.obten(genero);
  }

  /**
   * Agrega un género al catálogo de géneros. No se permiten géneros
   * con claves repetidas
   * @param genero Género a agregar
   * @throws FachadaException Si no se puede agregar el género al
   * catálogo de géneros.
   */
  public void agrega(Genero genero) throws FachadaException {
    Genero generoBuscado;

    // Busca el género en el arreglo con la misma clave.
    generoBuscado = catalogoGeneros.obten(genero);

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

    // Agrega el nuevo género al catálogo
    try {
      catalogoGeneros.agrega(genero);
    }
    catch(PersistenciaException pae) {
```

```
      throw new FachadaException("No se puede agregar el género", pae);
    }
  }

  /**
   * Actualiza un género del catálogo de géneros
   * @param genero Género a actualizar
   * @throws FachadaException Si no se puede actualizar el género del
   * catálogo de géneros.
   */
  public void actualiza(Genero genero) throws FachadaException {
    // Actualiza el género del catálogo
    try {
      catalogoGeneros.actualiza(genero);
    }
    catch(PersistenciaException pae) {
      throw new FachadaException("No se puede actualizar el género", pae);
    }
  }

  /**
   * Elimina un género del catálogo de géneros
   * @param genero Género a eliminar
   * @throws FachadaException Si no se puede eliminar el género del
   * catálogo de géneros.
   */
  public void elimina(Genero genero) throws FachadaException {
    // Elimina el género del catálogo
    try {
      catalogoGeneros.elimina(genero);
    }
    catch(PersistenciaException pae) {
      throw new FachadaException("No se puede eliminar el género", pae);
    }
  }

  /**
   * Obtiene una lista todas las canciones
   * @return Vector con la lista de todas las canciones
   */
  public Vector consultaCanciones() {
    // Regresa el vector con la lista de canciones
    return catalogoCanciones.lista();
  }

  /**
   * Obtiene una lista de todas las canciones con el mismo título.
   * @param titulo Titulo de las canciones de la lista
   * @return Vector con la lista de todas las canciones con el mismo
   * título.
   */
  public Vector consultaCancionesTitulo(String titulo) {
    // Regresa el vector con la lista de canciones
    return catalogoCanciones.listaTitulo(titulo);
  }

  /**
```

```
   * Obtiene una lista de todas las canciones con el mismo intérprete.
   * @param interprete Intérprete de las canciones de la lista
   * @return Vector con la lista de todas las canciones con el mismo
   * intérprete.
   */
  public Vector consultaCancionesInterprete(String interprete) {
    // Regresa el vector con la lista de canciones
    return catalogoCanciones.listaInterprete(interprete);
  }

  /**
   * Obtiene una lista de todas las canciones con el mismo autor.
   * @param autor Autor de las canciones de la lista
   * @return Vector con la lista de todas las canciones con el mismo autor.
   */
  public Vector consultaCancionesAutor(String autor) {
    // Obtiene el vector con la lista de canciones
    return catalogoCanciones.listaAutor(autor);
  }

  /**
   * Obtiene una lista de todas las canciones con el mismo género.
   * @param cveGenero Clave del género de las canciones de la lista
   * @return Vector con la lista de todas las canciones con el mismo
   * género.
   */
  public Vector consultaCancionesGenero(String cveGenero) {
    // Regresa el vector con la lista de canciones
    return catalogoCanciones.listaGenero(cveGenero);
  }

  /**
   * Obtiene una lista de todas las canciones del mismo álbum.
   * @param album Álbum de las canciones de la lista
   * @return Vector con la lista de todas las canciones del mismo álbum.
   */
  public Vector consultaCancionesAlbum(String album) {
    // Regresa el vector con la lista de canciones
    return catalogoCanciones.listaAlbum(album);
  }

  /**
   * Obtiene una lista de todas las canciones del mismo periodo.
   * @param periodo Período de las canciones de la lista
   * @return Vector con la lista de todas las canciones del mismo período.
   */
  public Vector consultaCancionesPeriodo(Periodo periodo) {
    // Regresa el vector con la lista de canciones
    return catalogoCanciones.listaPeriodo(periodo);
  }

  /**
   * Obtiene una lista de todas las películas
   * @return Vector con la lista de todas las películas.
   */
  public Vector consultaPeliculas() {
    // Regresa el vector con la lista de peliculas
```

```
    return catalogoPeliculas.lista();
  }

  /**
   * Obtiene una lista de todas las películas del mismo título
   * @param titulo Título de las películas de la lista
   * @return Vector con la lista de todas las peliculas del mismo título.
   */
  public Vector consultaPeliculasTitulo(String titulo) {
    // Regresa el vector con la lista de peliculas
    return catalogoPeliculas.listaTitulo(titulo);
  }

  /**
   * Obtiene una lista de todas las películas del mismo actor
   * @param actor Actor de las peliculas de la lista
   * @return Vector con la lista de todas las peliculas del mismo actor.
   */
  public Vector consultaPeliculasActor(String actor) {
    // Regresa el vector con la lista de peliculas
    return catalogoPeliculas.listaActor(actor);
  }

  /**
   * Obtiene una lista de todas las películas del mismo director
   * @param director Director de las peliculas de la lista
   * @return Vector con la lista de todas las peliculas del mismo director.
   */
  public Vector consultaPeliculasDirector(String director) {
    // Regresa el vector con la lista de peliculas
    return catalogoPeliculas.listaDirector(director);
  }

  /**
   * Obtiene una lista de todas las películas del mismo género
   * @param cveGenero Clave del género de las películas de la lista
   * @return Vector con la lista de todas las películas del mismo género.
   */
  public Vector consultaPeliculasGenero(String cveGenero) {
    // Regresa el vector con la lista de películas
    return catalogoPeliculas.listaGenero(cveGenero);
  }

  /**
   * Obtiene una lista de todas las películas del mismo periodo
   * @param periodo Periodo de las peliculas de la lista
   * @return Vector con la lista de todas las peliculas del mismo periodo.
   */
  public Vector consultaPeliculasPeriodo(Periodo periodo) {
    // Regresa el vector con la lista de peliculas
    return catalogoPeliculas.listaPeriodo(periodo);
  }

  /**
   * Obtiene una lista de todos los géneros
   * @return Vector con la lista de todos los géneros.
   */
```

```
  public Vector consultaGeneros() {
    // Regresa el vector con la lista de géneros
    return catalogoGeneros.lista();
  }

  /**
   * Obtiene una lista de los géneros de canciones
   * @return Vector con la lista de los géneros canciones
   */
  public Vector consultaGenerosCanciones() {
    // Regresa el vector con la lista de géneros de canciones
    return catalogoGeneros.listaMedio('C');
  }

  /**
   * Obtiene una lista de los géneros de películas
   * @return Vector con la lista de los géneros películas
   */
  public Vector consultaGenerosPeliculas() {
    // Regresa el vector con la lista de géneros de películas
    return catalogoGeneros.listaMedio('P');
  }
}
```

La siguiente clase es utilizada para probar la clase `FachadaArreglos` del programa sobre el amante de la música y el cine. En el método `main()` de esta clase se crean una instancia de la clase `FachadaArreglos` y se le asigna una referencia del tipo de la interfaz `IFachada`, esto es válido ya que se considera que cualquier clase que implementa una interfaz es del tipo de la interfaz. La clase `FachadaArreglos` oculta el mecanismo de almacenamiento que son arreglos y sólo deja ver las operaciones para agregar, actualizar, eliminar canciones y películas. Se listan los catálogos o se hacen listas parciales

Prueba3.java

```
/*
 * Prueba3.java
 *
 * Creada el 15 de septiembre de 2007, 12:21 PM
 */

package pruebas;

import java.util.Vector;

import objetosServicio.*;
import objetosNegocio.*;
import excepciones.FachadaException;
import interfaces.IFachada;
import fachadas.FachadaArreglos;

/**
 * Esta clase se utiliza para probar la clase FachadaArreglos
 *
 * @author mdomitsu
```

```
 */
public class Prueba3 {

  /**
   * Método main() en el que se invocan a los métodos de la clase
   * FachadaArreglos para probarlos
   * @param args the command line arguments
   */
  public static void main(String[] args) {
    Prueba3 prueba3 = new Prueba3();

    // Crean la fachada de los objetos que permiten almacenar las
    // canciones y peliculas en arreglos
    IFachada fachada = new FachadaArreglos();
    Vector lista = null;
    Genero genero;
    Cancion cancion = null;

    // Se crean tres géneros de canciones
    Genero genero1 = new Genero("GC001", "Balada", 'C');
    Genero genero2 = new Genero("GC002", "Bossanova", 'C');
    Genero genero3 = new Genero("GC003", "Rock", 'C');

    // Se agrega el género 1 al catálogo de géneros
    try {
      fachada.agrega(genero1);
      System.out.println("Se agrego el género 1 al catálogo de géneros");
    } catch(FachadaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage() + " 1");
    }

    // Se agrega el género 2 al catálogo de géneros
    try {
      fachada.agrega(genero2);
      System.out.println("Se agrego el género 2 al catálogo de géneros");
    } catch(FachadaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage() + " 2");
    }

    // Se agrega el género 3 al catálogo de géneros
    try {
      fachada.agrega(genero3);
      System.out.println("Se agrego el género 3 al catálogo de géneros");
    } catch(FachadaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage() + " 3");
    }

    // Se agrega de nuevo el género 1 al catálogo de géneros
    try {
      fachada.agrega(genero1);
      System.out.println("Se agrego el género 1 al catálogo de géneros");
    } catch(FachadaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage() + " 1");
```

```
    }

    // Se crean tres géneros de películas
    Genero genero4 = new Genero("GP001", "Drama", 'P');
    Genero genero5 = new Genero("GP002", "Ciencia Ficción", 'P');
    Genero genero6 = new Genero("GP003", "Comedia", 'P');

    // Se agrega el género 4 al catálogo de géneros
    try {
      fachada.agrega(genero4);
      System.out.println("Se agrego el género 4 al catálogo de géneros");
    } catch(FachadaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage() + " 4");
    }

    // Se agrega el género 5 al catálogo de géneros
    try {
      fachada.agrega(genero5);
      System.out.println("Se agrego el género 5 al catálogo de géneros");
    } catch(FachadaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage() + " 5");
    }

    // Se agrega el género 6 al catálogo de géneros
    try {
      fachada.agrega(genero6);
      System.out.println("Se agrego el género 6 al catálogo de géneros");
    } catch(FachadaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage() + " 6");
    }

    // Despliega el contenido del catalogo de géneros
    System.out.println("Lista de géneros");
    try {
      System.out.println(fachada.consultaGeneros());
    } catch(FachadaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage());
    }

    // Se modifica el genero de clave "GC002", a "Samba"
    try {
      genero = fachada.obten(new Genero("GC002"));
      genero.setNombre("Samba");
      fachada.actualiza(genero);
      System.out.println("Se actualizo el genero de clave GC002");
    } catch(FachadaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage() + "GC002");
    }

    // Se elimina el género "GP003" del catalogo de generos
    try {
      fachada.elimina(new Genero("GP003"));
```

```
      System.out.println("Se elimino el genero de clave GP003");
    } catch(FachadaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage() + "GP003");
    }

    // Se elimina el género "GP004" (inexistente) del catalogo de generos
    try {
      fachada.elimina(new Genero("GP004"));
      System.out.println("Se elimino el genero de clave GP004");
    } catch(FachadaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage() + " GC004");
    }

    // Despliega el contenido del catalogo de géneros
    System.out.println("Lista de géneros");
    try {
      System.out.println(fachada.consultaGeneros());
    } catch(FachadaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage());
    }

    // Despliega el contenido del catalogo de géneros de canciones
    System.out.println("Lista de géneros de canciones");
    try {
      System.out.println(fachada.consultaGenerosCanciones());
    } catch(FachadaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage());
    }

    // Despliega el contenido del catalogo de género de películas
    System.out.println("Lista de géneros de películas");
    try {
      System.out.println(fachada.consultaGenerosPeliculas());
    } catch(FachadaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage());
    }

    // Se crean tres canciones
    Cancion cancion1 = new Cancion("C0001", "The long and winding road",
                                   genero1, "The Beatles", "John Lennon",
                                   "Let it be", 3, new Fecha(24, 3, 1970));
    Cancion cancion2 = new Cancion("C0002", "Garota de Ipanema", genero2,
                                   "Los Indios Tabajaras",
                                   "Antonio Carlos Jobim",
                                   "Bossanova Jazz Vol. 1", 3,
                                   new Fecha(1, 12, 1970));
    Cancion cancion3 = new Cancion("C0003", "Desafinado", genero2,
                                  "Joao Gilberto", "Joao Gilberto",
                                  "Bossanova Jazz Vol. 1", 3,
                                  new Fecha(3, 12, 1980));

    // Se agrega la canción 1 al catálogo de canciones
```

```
    try {
      fachada.agrega(cancion1);
      System.out.println("Se agrego la cancion 1");
    } catch(FachadaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage() + " 1");
    }

    // Se intenta agregar de nuevo la canción 1 al catálogo de canciones
    try {
      fachada.agrega(cancion1);
      System.out.println("Se agrego la cancion 1");
    } catch(FachadaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage() + " 1");
    }

    // Se agrega la canción 2 al catálogo de canciones
    try {
      fachada.agrega(cancion2);
      System.out.println("Se agrega la cancion 2");
    } catch(FachadaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage() + " 2");
    }

    // Se agrega la canción 3 al catálogo de canciones
    try {
      fachada.agrega(cancion3);
      System.out.println("Se agrega la cancion 3");
    } catch(FachadaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage() + " 3");
    }

    // Se lista el catálogo de canciones
    System.out.println("Lista de canciones:");
    try {
      lista = fachada.consultaCanciones();
      System.out.println(lista);
    } catch(FachadaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage());
    }

    // Se actualiza la canción de clave "C0001" al genero "GC003"
    try {
      // Obtiene el género 3 del cátalogo de géneros de canciones
      genero = fachada.obten(new Genero("GC003"));
      if(genero != null) {
        // Obtiene la canción 1 del cátalogo de canciones
        cancion = fachada.obten(new Cancion("C0001"));
        if(cancion != null) {
          // Se actualiza la canción 1
          cancion.setGenero(genero);
          fachada.actualiza(cancion);
          System.out.println("Se actualizo la canción de clave C0001 al
```

```
genero GC003");
        } else System.out.println("No existe la canción C0001");
      } else System.out.println("No existe el género GC003");
    } catch(FachadaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage());
    }

    // Se elimina la canción de clave "C0003"
    try {
        fachada.elimina(new Cancion("C0003"));
        System.out.println("Se elimina la cancion de clave C0003");
    } catch(FachadaException fe) {
        // Muestra el mensaje de error amistoso
        System.out.println("Error: " + fe.getMessage() + " C0003");
    }

    // Se lista el catálogo de canciones
    System.out.println("Lista de canciones:");
    try {
      lista = fachada.consultaCanciones();
      System.out.println(lista);
    } catch(FachadaException fe) {
        // Muestra el mensaje de error amistoso
        System.out.println("Error: " + fe.getMessage());
    }

    // Se lista las canciones con el interprete "The Beatles"
    System.out.println("Lista de canciones de The Beatles:");
    try {
      lista = fachada.consultaCancionesInterprete("The Beatles");
      System.out.println(lista);
    } catch(FachadaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage());
    }

    // Se lista las canciones de samba, "GC002"
    System.out.println("Lista de canciones de Samba:");
    try {
      lista = fachada.consultaCancionesGenero("GC002");
      System.out.println(lista);
    } catch(FachadaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage());
    }

    // Se crean dos peliculas
    Pelicula pelicula1 = new Pelicula("P000001", "Casa Blanca", genero4,
                                      "Humphrey Bogart", "Ingrid Bergman",
                                      "Michael Curtiz", 102,
                                      new Fecha(1, 1, 1942));
    Pelicula pelicula2 = new Pelicula("P000002", "2001 Space Odyssey",
                                      genero5, "Keir Dullea",
                                      "Gary Lockwood", "Stanley Kubrick",
                                      141, new Fecha(1, 1, 1968));
```

```
    // Se agrega la película 1 al catálogo de películas
    try {
      fachada.agrega(pelicula1);
      System.out.println("Se agrego la película 1");
    } catch(FachadaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage() + " 1");
    }

    // Se agrega la película 2 al catálogo de películas
    try {
      fachada.agrega(pelicula2);
      System.out.println("Se agrego la película 2");
    } catch(FachadaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage() + " 2");
    }

    // Se lista el catálogo de películas
    System.out.println("Lista de peliculas:");
    try {
      lista = fachada.consultaPeliculas();
      System.out.println(lista);
    } catch(FachadaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage());
    }

    // Se lista las películas de Ingrid Bergman
    System.out.println("Lista de peliculas de Ingrid Bergman:");
    try {
      lista = fachada.consultaPeliculasActor("Ingrid Bergman");
      System.out.println(lista);
    } catch(FachadaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage());
    }

    // Lista de peliculas en el periodo: 1/03/1970 a 1/05/1970
    Periodo periodo = new Periodo(new Fecha(1, 1, 1960),
                                  new Fecha(1, 1, 1970));
    System.out.println("Lista de peliculas en el periodo: " + periodo);
    try {
      lista = fachada.consultaPeliculasPeriodo(periodo);
      System.out.println(lista);
    } catch(FachadaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage());
    }
  }
}
```

El listado siguiente muestra un fragmento de la corrida del programa anterior al generarse un error al querer eliminar un género inexistente:

```
...
Lista de géneros
[GC001, Balada, C, GC002, Bossanova, C, GC003, Rock, C, GP001, Drama, P,
GP002, Ciencia Ficción, P, GP003, Comedia, P]
Se actualizo el genero de clave GC002
Se elimino el genero de clave GP003
Error: No se puede eliminar el género GC004
Lista de géneros
[GC001, Balada, C, GC002, Samba, C, GC003, Rock, C, GP001, Drama, P,
GP002, Ciencia Ficción, P]
Lista de géneros de canciones
[GC001, Balada, C, GC002, Samba, C, GC003, Rock, C]
Lista de géneros de películas
[GP001, Drama, P, GP002, Ciencia Ficción, P]
...
```

Podemos notar que aunque se despliega un mensaje de error amistoso, útil para el usuario final, no proporciona ninguna información sobre el error original ni sobre el lugar donde ocurrió el error, información necesaria para el desarrollador que depura el código. Para mostrar dicha información podemos agregarle a cada bloque `catch` la sentencia:

```
fe.printStackTrace();
```

necesaria para desplegar la traza de las invocaciones a los métodos en el momento de generarse la excepción. Si sólo hubiéramos envuelto la excepción en otra sin conservar la información de la excepción original usando el siguiente código en los bloques `catch` dentro de los métodos de la clase `FachadaArreglos`:

```
...
try {
  catalogoGeneros.elimina(genero);
}
catch(PersistenciaException pae) {
  throw new FachadaException("No se puede eliminar el género);
}
...
```

Al ejecutar la aplicación, tendremos lo mostrado en el siguiente fragmento del listado de la corrida:

```
...
excepciones.FachadaException: No se puede eliminar el género
        at fachadas.FachadaArreglos.elimina(FachadaArreglos.java:242)
        at pruebas.Prueba3.main(Prueba3.java:144)
...
```

Podemos ver que en este caso la traza, mostrado al principio del listado, sólo nos muestra las invocaciones de métodos hasta el punto donde se envolvió la excepción original lanzando la nueva excepción. Sigue sin saberse dónde se generó la excepción original. Sin embargo si al envolver la excepción original conservamos la información

de la excepción original como se muestra en la clase FachadaArreglos, tendremos lo mostrado en el siguiente fragmento del listado de la corrida al ejecutar la aplicación:

```
excepciones.FachadaException: No se puede eliminar el género
Se agrego el género 1 al catálogo de géneros
Se agrego el género 2 al catálogo de géneros
        at fachadas.FachadaArreglos.elimina(FachadaArreglos.java:242)
        at pruebas.Prueba3.main(Prueba3.java:144)
Caused by: excepciones.PersistenciaException: Género inexistente
        at persistencia.Generos.elimina(Generos.java:118)
        at fachadas.FachadaArreglos.elimina(FachadaArreglos.java:239)
        ... 1 more
```

Podemos ver que en este caso la traza de invocaciones de métodos continúa hasta el punto en que se generó la excepción original.