Tema 6

Colecciones

Una *colección,* a veces llamada un contenedor, es simplemente un objeto que agrupa múltiples elementos en una simple unidad. Las colecciones se usan para almacenar, recuperar, manipular y comunicar datos compuestos. Normalmente representan datos que forman grupos naturalmente, como una mano de poker (una colección de cartas), una carpeta de correo (una colección de cartas) o un directorio telefónico (Un mapeo de nombres a números telefónicos).

Marco de Colecciones

Un *marco de colecciones* es una arquitectura unificada para representar y manipular colecciones. Todos los marcos de colecciones contienen lo siguiente:

- **Interfaces:** Son tipos de datos abstractos que representan colecciones. Las interfaces permiten manipular a las colecciones independientemente de los detalles de su implementación.

- **Implementaciones:** Son las implementaciones concretas de las interfaces de la la colección. Son estructuras de datos reutilizables

- **Algoritmos:** Son métodos que realizan computaciones útiles, como búsquedas, ordenamientos, sobre los objetos que implementan interfaces de la colección.

Marco de Colecciones de Java

La API de Java nos proporciona un marco de colecciones formada de una jerarquía de interfaces y clases que implementan esas interfaces. La figura 6.1 es diagrama de clases parcial de esa jerarquía. Note que todos los nombres de las interfaces y clases del marco de colecciones tienen un sufijo, <E>, <T>, <K, V>, etc. Estos sufijos indican que las interfaces y clases del marco son genéricas. Al declarar una instancia de este marco se debe especificar el tipo del objeto que se va a almacenar en la colección. Esto permite que el compilador pueda verificar que el tipo de objeto que se guarde en la colección es correcto. Por ejemplo

```
List<Integer> listaEnteros = new ArrayList();
Set<Cancion> canciones = new Hashset();
```

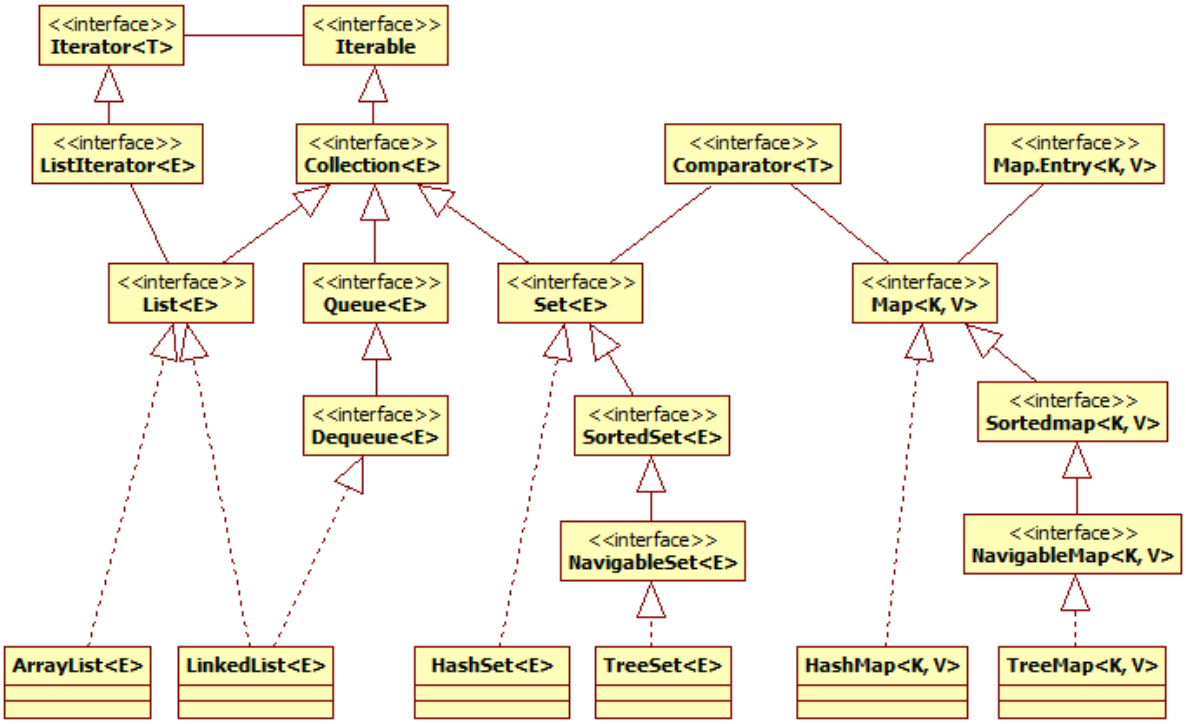

Figura 6.1

Interfaz `Collection<E>`

La interfaz `Collection<E>` representa la raíz de la jerarquía del marco de colecciones de Java. Representa un grupo de objetos, conocidos como sus elementos. Algunas colecciones permiten elementos repetidos, otras no. En algunas colecciones los elementos están ordenados en otras no. Los diferentes tipos de colecciones están representadas por las subinterfaces de `Collection<E>`: `List<E>`, `Queue<E>`, `Deque<E>`, `Set<E>`, `SortedSet<E>` y `NavigableSet<E>`.

Aunque no hay ninguna clase que implemente directamente la interfaz `Collection<E>`, si hay clases que implementan sus subinterfaces y a través de éstas, los métodos de `Collection<E>`. En la figura 6.2 se muestran los métodos declarados por la interfaz `Collection<E>` y en la tabla 6.1 la descripción de éstos métodos.

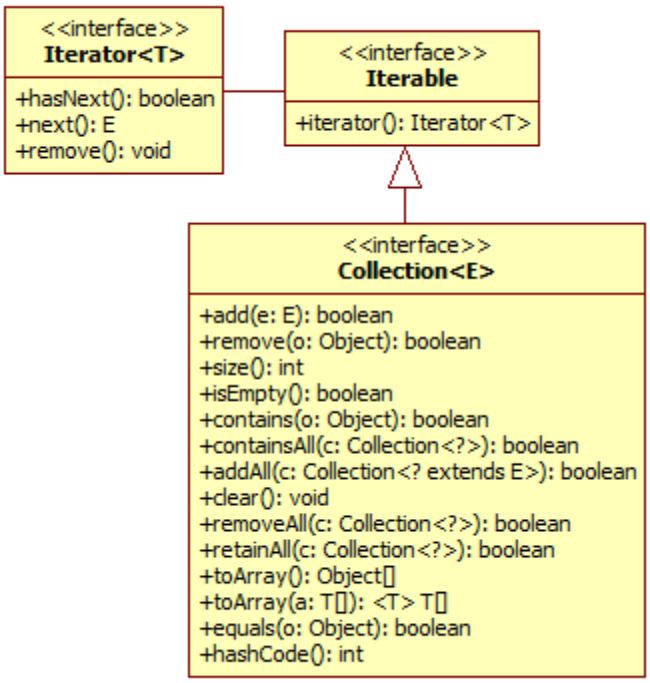

Figura 6.2

Tabla 6.1 Métodos de la Interfaz `Collection<E>`

`boolean add(E e)` Agrega el elemento dado por el parámetro a la colección. Si la colección soporta la operación. Regresa `true` si se pudo agregar el elemento, `false` en el caso que la colección ya tenga al elemento y la colección no soporta duplicados. **Lanza**: `UnsupportedOperationException` - Si la colección no soporta la operación. `ClassCastException` – Si no se puede agregar un elemento de la clase del elemento. `NullPointerException` – Si el elemento es nulo y la colección no admite elementos nulos. `IllegalArgumentException` – Algún aspecto del elemento impide agregarlo a la colección.
`boolean remove(Object o)` Remueve una sola instancia del elemento del parámetro de la colección, si se encuentra. Regresa `true` si se pudo remover el elemento, `false` en el caso contrario. **Lanza**: `UnsupportedOperationException` - Si la colección no soporta la operación. `ClassCastException` – Si la clase del parámetro no es compatible con la clase de los elementos. `NullPointerException` – Si el elemento es nulo y la colección no admite elementos nulos
`int size()` Regresa el número de elementos de esta colección. Si el número de elementos es mayor que `Integer.MAX_VALUE`, la función regresa `Integer.MAX_VALUE`.

Tabla 6.1 Métodos de la Interfaz `Collection<E>`. Cont.

`boolean isEmpty()` Regresa verdadero si la colección no contiene elementos.
`boolean contains(Object o)` Regresa verdadero si la colección contiene al elemento del parámetro. **Lanza**: `ClassCastException` – Si la clase del parámetro no es compatible con la clase de los elementos. `NullPointerException` – Si el elemento es nulo y la colección no admite elementos nulos
`boolean containsAll(Collection<?> c)` Regresa verdadero si la colección contiene a todos los elementos de la colección del parámetro. **Lanza**: `ClassCastException` – Si la clase de uno o más elementos de la colección del parámetro no es compatible con la clase de los elementos de esta colección. `NullPointerException` – Si uno o más de los elementos de la colección del parámetro es nulo y esta colección no admite elementos nulos `NullPointerException` – Si la colección del parámetro es nula.
`boolean addAll(Collection<? extends E> c)` Agrega todos los elementos de la colección del parámetro a esta colección. Regresa `true` si se pudieron agregar los elementos, `false` en el caso contrario. **Lanza**: `UnsupportedOperationException` - Si la colección no soporta la operación. `ClassCastException` – Si la clase de uno o más elementos de la colección del parámetro no es compatible con la clase de los elementos de esta colección. `NullPointerException` – Si uno o más de los elementos de la colección del parámetro es nulo y esta colección no admite elementos nulos `IllegalArgumentException` – Si algún aspecto de uno de los elementos del parámetro impide agregarlo a la colección.
`void clear()` Remueve todos los elementos de esta colección. Después de la operación la colección estará vacía a menos de que ocurra una excepción. **Lanza**: `UnsupportedOperationException` - Si la colección soporta la operación.

Tabla 6.1 Métodos de la Interfaz `Collection<E>`. Cont.

`boolean removeAll(Collection<?>` *`c`*`)` Remueve todos los elementos de esta colección que también estén en la colección del parámetro. Regresa `true` si se pudieron remover los elementos, `false` en el caso contrario. **Lanza**: `UnsupportedOperationException` - Si la colección no soporta la operación. `ClassCastException` – Si la clase de uno o más elementos de la colección del parámetro no es compatible con la clase de los elementos de esta colección. `NullPointerException` – Si uno o más de los elementos de la colección del parámetro es nulo y esta colección no admite elementos nulos `NullPointerException` – Si la colección del parámetro es nula.
`boolean retainAll(Collection<?>` *`c`*`)` Retiene todos los elementos de esta colección que también estén en la colección del parámetro. Regresa `true` si se pudieron retener los elementos, `false` en el caso contrario. **Lanza**: `UnsupportedOperationException` - Si la colección no soporta la operación. `ClassCastException` – Si la clase de uno o más elementos de la colección del parámetro no es compatible con la clase de los elementos de esta colección. `NullPointerException` – Si uno o más de los elementos de la colección del parámetro es nulo y esta colección no admite elementos nulos `NullPointerException` – Si la colección del parámetro es nula.
`Object[] toArray()` Regresa un arreglo con todos los elementos de esta colección. Si la colección mantiene ordenados a sus elementos, los elementos en el arreglo estarán ordenados en ese mismo orden.
`<T> T[] toArray(T[]` *`a`*`)` Regresa un arreglo con todos los elementos de esta colección. El tipo del arreglo regresado es el del tipo del arreglo del parámetro. Si la colección cabe en el arreglo del parámetro, se regresa en él. De otro modo se crea un arreglo. Si el tamaño del arreglo del parámetro es mayor que la colección, el siguiente elemento del arreglo después de la colección se establece a null. Si la colección mantiene ordenados a sus elementos, los elementos en el arreglo estarán ordenados en ese mismo orden. **Lanza**: `ArrayStoreException` – Si el tipo del arreglo del parámetro no es un supertipo de todos los elementos de la colección. `NullPointerException` – Si el arreglo del parámetro es nulo.
`boolean equals(Object` *`o`*`)` Regresa verdadero si esta colección es igual al objeto del parámetro
`int hashCode()` Regresa el valor del código Hash para esta colección.

Recorrido de Colecciones

Una de las operaciones que más se realizan con las colecciones es recorrerlas para procesar sus elementos. Hay dos formas de recorrer una colección, con un ciclo for-each o usando iteradores.

Ciclo for-each

En el Tema 5 se vio una variante del ciclo for llamada ciclo for-each. Esa variante que puede utilizarse para iterar sobre los elementos de un arreglo también puede utilizarse para iterar sobre los elementos de una colección. La sintaxis, en el caso de una colección, es la siguiente:

```
for(tipo nomVariable: nomColeccion)
 {
    cuerpo del ciclo
}
```

Por ejemplo, el siguiente código despliega cada uno de los elementos de una colección:

```
for(Object o: nomColeccion)
    System.out.println(o);
```

En el caso de las colecciones, el ciclo for-each tiene las mismas limitaciones que su uso con arreglos:

- Sólo podemos acceder a los elementos de una collección. No podemos asignarle un valor a los elementos de una colección.
- Sólo se puede trabajar con un solo colección. No podemos, por ejemplo comparar dos colecciones.
- Sólo se puede accede un elemento a la vez. No podemos comparar, por ejemplo elementos consecutivos.
- Sólo se puede recorrer la colección hacia adelante y en incrementos de uno.
- No puede iterarse sobre sólo parte de la colección. El ciclo `for-each` recorre todo la colección.
- No se use si se desea compatibilidad con código previo a la versión 5 de java.

Iteradores

Un iterador es un objeto que permite recorrer una colección y remover elementos si se desea. Para obtener un iterador de una colección se invoca a su método `iterator()`. Todas las colecciones tienen ese método porque la interfaz `Collection<E>` hereda la Interfaz `Iterable<T>` que declara dicho método, tabla 6.2.

Interfaz `Iterable<T>`

Tabla 6.2 Método de la Interfaz `Iterable<E>`

`Iterator<T> iterator()` Regresa un iterador sobre un conjunto de elementos de tipo T

Interfaz `Iterator<T>`

Los iteradores deben implementar la interfaz `Iterator<T>` que declara los métodos que nos permiten iterar sobre una colección.

Tabla 6.3 Métodos de la Interfaz `Iterator<E>`

`boolean hasNext()` Regresa verdadero si la iteración tiene más elementos.
`E next()` Regresa el siguiente elemento en la iteración. **Lanza**: `NoSuchElementException` – Si la iteración no tiene más elementos.
`void remove``()` Remueve de la colección el último elemento regresado por el iterador. El método sólo puede llamarse una vez por cada llamada al método `next()`. **Lanza**: `UnsupportedOperationException` – Si la operación no es soportada por este iterador. `IllegalStateException` – Si el método `next()` no ha sido llamado aún, o si el método `remove()` ya ha sido llamado después de la última llamada al método `next()`.

Por ejemplo, el ciclo for-each:

```
for(tipo nomVar: nomColeccion)sentencia;
```

es equivalente al siguiente fragmento de código que emplea los métodos de un iterador:

```
for (Iterator<tipo> iter = nomColeccion.iterator(); iter.hasNext(); ) {
    tipo nomVar = iter.next();
    sentencia;
}
```

Listas

Una lista es una colección ordenada. El usuario de la interfaz `List<E>` tiene un control preciso sobre la posición en la lista en que cada elemento se inserta. El usuario puede acceder a los elementos de la lista por su índice (su posición en la lista) y buscar elementos en la lista.

Normalmente las listas permiten elementos duplicados, formalmente, las listas permiten pares de elementos `e1` y `e2` tal que `e1.equals(e2)`, y permite múltiples elementos nulos si es que permite elementos nulos. En la figura 6.3 se muestran las interfaces y clases que implementan las listas.

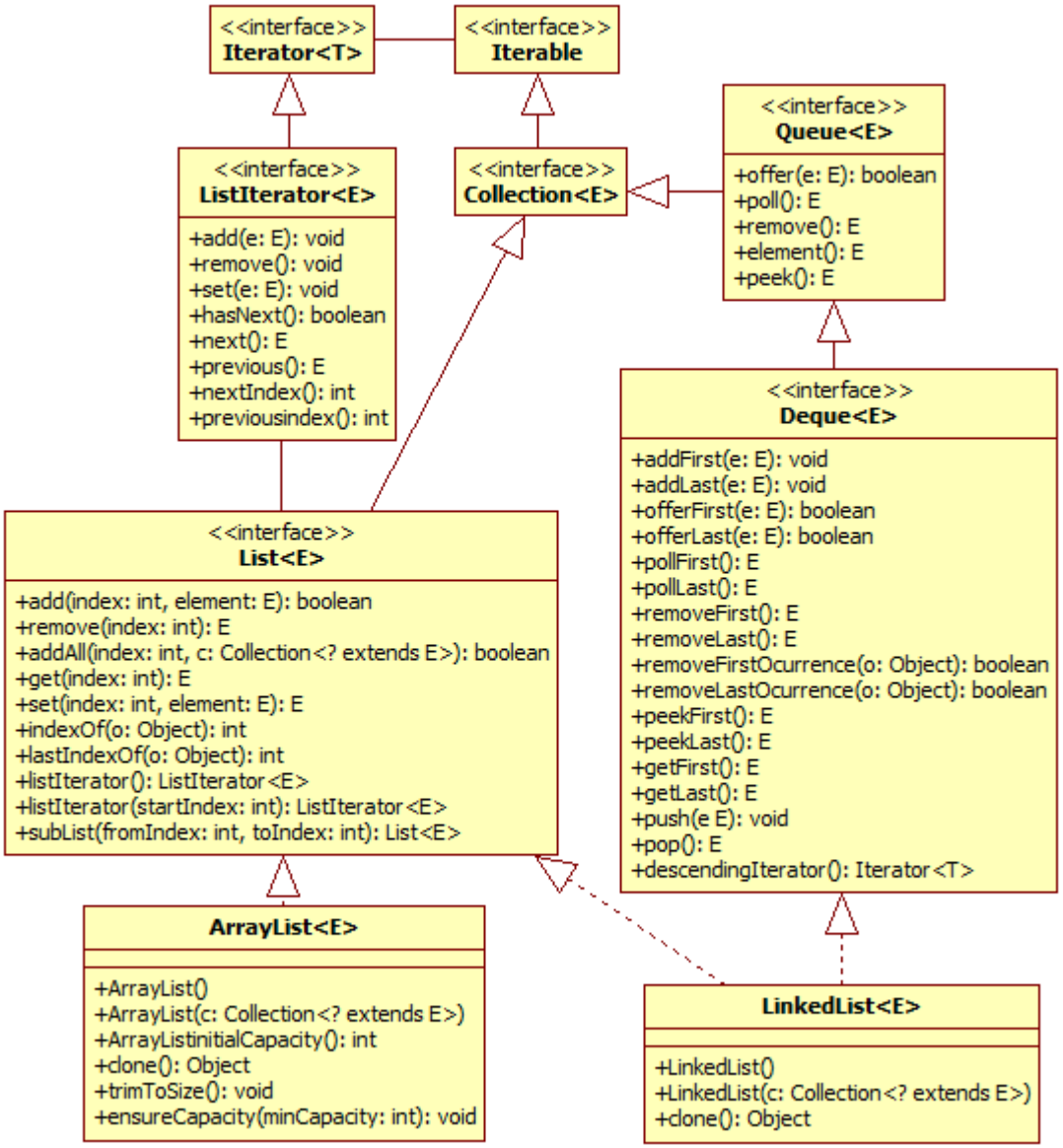

Figura 6.3

Interfaz `List<E>`

La interfaz `List<E>` establece los métodos comunes a todas las listas y su descripción se encuentra en la tabla 6.4.

Tabla 6.4 Métodos de la Interfaz `List<E>`

`void add(int` *`index`*`, E` *`element`***`)`** Inserta el elemento dado por el parámetro `element` en la posición dada por el parámetro `index`. El elemento en la posición actual de la lista y los elementos subsiguientes son desplazados a la derecha (aumentan su índice en uno). **Lanza**: `UnsupportedOperationException` - Si la lista no soporta la operación. `ClassCastException` – Si la clase del elemento del impide que sea agregado a la lista. `NullPointerException` – Si el elemento es nulo y la lista no admite elementos nulos `IllegalArgumentException` – Si algún aspecto del elemento del parámetro impide que sea agregado a la lista. `IndexOutOfBoundsException` – Si el índice se encuentra fuera de rango (`index < 0 || index > size()`).
`E remove(int` *`index`***`)`** Extrae el elemento de la posición dada por el parámetro `index`. Los elementos subsiguientes son desplazados a la izquierda (disminuyen su índice en uno). **Lanza**: `UnsupportedOperationException` - Si la lista no soporta la operación. `IndexOutOfBoundsException` – Si el índice se encuentra fuera de rango (`index < 0 || index > size()`).
`boolean addAll(int` *`index`*`, Collection<? extends E>` *`c`***`)`** Agrega todos los elementos de la colección dada por el parámetro c en la posición dada por el parámetro `index`. El elemento en la posición actual de la lista y los elementos subsiguientes son desplazados a la derecha (aumentan su índice en uno). Los nuevos elementos son agregados en el orden en que son regresados por el iterador de la colección. Regresa `true` si se pudieron agregar los elementos, `false` en el caso contrario. **Lanza**: `UnsupportedOperationException` - Si la lista no soporta la operación. `ClassCastException` – Si la clase de uno de los elementos de la colección impide que sea agregado a la lista. `NullPointerException` – Si alguno de los elementos de la colección es nulo y la lista no admite elementos nulos `IllegalArgumentException` – Si algún aspecto de alguno de los elementos de la colección impide que sea agregado a la lista. `IndexOutOfBoundsException` – Si el índice se encuentra fuera de rango (`index < 0 || index > size()`).

Tabla 6.4 Métodos de la Interfaz `List<E>`. Cont.

`E get(int` *`index`*`)` Regresa el elemento que se encuentra en la posición especificada por el parámetro. **Lanza**: `IndexOutOfBoundsException` – Si el índice se encuentra fuera de rango (`index < 0 || index > size()`).
`void set(int` *`index`*`, E` *`element`*`)` Reemplaza el elemento en la posición dada por el parámetro `index` por el elemento dado por el parámetro `element`. El elemento en la posición actual de la lista y los elementos subsiguientes son desplazados a la derecha (aumentan su índice en uno). **Lanza**: `UnsupportedOperationException` - Si la lista no soporta la operación. `ClassCastException` – Si la clase del elemento del impide que sea agregado a la lista. `NullPointerException` – Si el elemento es nulo y la lista no admite elementos nulos. `IllegalArgumentException` – Si algún aspecto del elemento del parámetro impide que sea agregado a la lista. `IndexOutOfBoundsException` – Si el índice se encuentra fuera de rango (`index < 0 || index > size()`).
`int indexOf(Object` *`o`*`)` Regresa el índice de la primera ocurrencia del elemento del parámetro. -1 si no se encuentra. **Lanza:** `ClassCastException` – Si la clase del elemento es incompatible con esta lista. `NullPointerException` – Si el elemento es nulo y la lista no admite elementos nulos.
`int lastIndexOf(Object` *`o`*`)` Regresa el índice de la última ocurrencia del elemento del parámetro. -1 si no se encuentra. **Lanza:** `ClassCastException` – Si la clase del elemento es incompatible con esta lista. `NullPointerException` – Si el elemento es nulo y la lista no admite elementos nulos.
`ListIterator<E> listIterator()` Regresa un iterador de lista de los elementos de esta lista.
`ListIterator<E> listIterator``(int index)` Regresa un iterador de lista de los elementos de esta lista empezando en la posición indicada por el parámetro `index`. El índice indica el primer elemento que será regresado por la llamada al método `next()`. Si inicialmente se llama al método `previous()` se obtendrá el elemento `indice-1`. **Lanza:** `IndexOutOfBoundsException` – Si el índice se encuentra fuera de rango (`index < 0 || index > size()`).

Tabla 6.4 Métodos de la Interfaz `List<E>`. Cont.

`List<E> subList(int` *`fromIndex`***`, int`** *`toIndex`***`)`** Regresa una vista de una porción de la esta lista desde `fromIndex` inclusive hasta `toIndex` exclusive. Si `fromIndex` y `toIndex` son iguales, la sublista es nula. Cualquier cambio en la sublista afecta a la lista y viceversa. **Lanza:** `IndexOutOfBoundsException` – Si uno de los índices tiene un valor ilegal (`fromIndex < 0 || toIndex > size() || fromIndex < 0 || toIndex`).

Interfaz `ListIterator<E>`

Aparte de poder iterar sobre sus elementos mediante un iterador que implemente la interfaz `Iterator<T>`, una lista puede usar un iterador para listas, que implementa la interfaz `ListIterator<E>`, que permite recorrer una lista en ambas direcciones, modificar la lista durante una iteración y la posición actual del iterador en la lista. El iterador para listas no tiene un elemento actual; la posición del cursor siempre está entre los elementos regresados por los métodos `previous()` y `next()`. En una lista de longitud n, hay n+1 valores válidos, de 0 a n, inclusive.

En la tabla 6.5 se muestran los métodos declarados por la interfaz `ListIterator<E>`.

Tabla 6.5 Métodos de la Interfaz `ListIterator<E>`

`void add(E` *`e`***`)`** Inserta el elemento dado por el parámetro en la lista. El elemento será insertado inmediatamente antes que el elemento regresado por la llamada al método `next()`, si existe, e inmediatamente después del elemento regresado por la llamada al método `previous()`, si existe. Si la lista no contiene elementos, el elemento será el único elemento de la lista. **Lanza:** `UnsupportedOperationException` – Si la operación no es soportada por este iterador. `ClassCastException` – Si la clase del elemento del parámetro le impide ser agregado a la lista. `IllegalArgumentException` – Si algún aspecto del elemento del parámetro impide agregarlo a la colección.
`void remove``()` Remueve de la lista el último elemento regresado por `next()` o `previous()`. Esta operación sólo puede llamarse una vez por llamada a `next()` o `previous()`. Esta operación sólo puede hacerse si `ListIterator.add()` no ha sido llamada después de la última llamada a `next()` o `previous()`. **Lanza:** `UnsupportedOperationException` – Si la operación no es soportada por este iterador. `IllegalStateException` – Si el método `next()`o `previous()`no han sido llamados aún, o si el método `remove` ya ha sido llamado después de la última llamada al método `next()`o `previous()`.

Tabla 6.5 Métodos de la Interfaz `ListIterator<E>`. Cont.

`void set(E e)` Remplaza de la lista el último elemento regresado por `next()` o `previous()` con el elemento especifícado por el parámetro. Esta operación sólo puede hacerse si ninguno de `ListIterator.remove()` o `ListIterator.add()` no ha sido llamada después de la última llamada a `next()` o `previous()`. **Lanza:** `UnsupportedOperationException` – Si la operación no es soportada por este iterador. `ClassCastException` – Si la clase del elemento del parámetro le impide ser agregado a la lista. `IllegalArgumentException` – Si algún aspecto del elemento del parámetro impide agregarlo a la colección. `IllegalStateException` – Si el método `next()`o `previous()`no han sido llamados aún, o si el método `remove` ya ha sido llamado después de la última llamada al método `next()`o `previous()`.
`boolean hasNext()` Regresa verdadero si este iterador tiene más elementos cuando se recorre la lista hacia adelante.
`boolean hasPrevious()` Regresa verdadero si este iterador tiene más elementos cuando se recorre la lista hacia atrás.
`E next()` Regresa el siguiente elemento de la lista. **Lanza:** `NoSuchElementException` – Si la iteración no tiene elemento siguiente.
`E previous()` Regresa el anterior elemento de la lista. **Lanza:** `NoSuchElementException` – Si la iteración no tiene elemento anterior.
`int nextIndex()` Regresa el índice del elemento que sería regresado por la siguiente llamada a `next()`, o el tamaño de la lista si el iterador se encuentra al final de la lista.
`int previousIndex()` Regresa el índice del elemento que sería regresado por la siguiente llamada a `previous()`, o -1 si el iterador se encuentra al inicio de la lista.

Interfaz `Queue<E>`

Las colas son objetos que implementan la interfaz `Queue<E>` permiten almacenar un elemento previo a su procesamiento. La interfaz `Queue<E>` aparte de las operaciones de la interfaz `Collection<E>`, provee las operaciones para insertar, extraer e

inspeccionar. Las colas por lo general ordenan los elementos de forma PEPS (primeros en entrar, primeros en salir).

En la tabla 6.6 se muestran los métodos declarados por la interfaz `Queue<E`.

Tabla 6.6 Métodos de la Interfaz Queue<E>

`boolean offer(E )` Inserta el elemento dado por el parámetro, si es posible. **Regresa**: Verdadero si fue posible agregar al elemento, falso en caso contrario.
`E poll()` Obtiene y remueve un elemento de la cabeza de la cola. **Regresa:** La cabeza de la cola, `null` en caso contrario.
`E remove()` Obtiene y remueve un elemento de la cabeza de la cola. **Lanza:** `NoSuchElementException` – Si la iteración no tiene elemento anterior.
`E peek()` Obtiene pero no remueve un elemento de la cabeza de la cola. **Regresa:** La cabeza de la cola, `null` en caso contrario.
`E element()` Obtiene pero no remueve un elemento de la cabeza de la cola. **Lanza:** `NoSuchElementException` – Si la iteración no tiene elemento anterior.

Interfaz `Deque<E>`

Las colas dobles son objetos que implementan la interfaz `Deque<E>`. Las colas dobles permiten la inserción y remoción de elementos en ambos lados de la cola. La mayoría de las clases que implementan esta interfaz no imponen un tamaño fijo en el número de elementos que pueden contener, pero la interfaz también soporta colas dobles de tamaño fijo.

En la tabla 6.7 se muestran los métodos declarados por la interfaz `Deque<E`.

Tabla 6.7 Métodos de la Interfaz Deque<E>

`void addFirst(E e)` Inserta el elemento dado por el parámetro, en el frente de la cola, si es posible sin violar las restricciones de capacidad. **Lanza**: `IllegalStateException` – Si el elemento no puede agregarse en este momento debido a las restricciones de capacidad. `ClassCastException` – Si la clase del elemento impide que sea agregado a la cola doble. `NullPointerException` – Si el elemento es nulo y la cola doble no admite elementos nulos. `IllegalArgumentException` – Si algún atributo del elemento del parámetro impide agregarlo a la colección.
`void addLast(E e)` Inserta el elemento dado por el parámetro, en el frente de la cola, si es posible sin violar las restricciones de capacidad. **Lanza**: `IllegalStateException` – Si el elemento no puede agregarse en este momento debido a las restricciones de capacidad. `ClassCastException` – Si la clase del elemento impide que sea agregado a la cola doble. `NullPointerException` – Si el elemento es nulo y la cola doble no admite elementos nulos. `IllegalArgumentException` – Si algún atributo del elemento del parámetro impide agregarlo a la colección.
`boolean offerFirst(E e)` Inserta el elemento dado por el parámetro al frente de esta cola doble a menos que se viole las restricciones de capacidad. Si la cola doble tiene restricciones de capacidad es preferible este método al de `addFirst()`. Regresa `true` si se agregó el elemento a la cola, `false` en caso contrario. **Lanza**: `ClassCastException` – Si la clase del elemento impide que sea agregado a la cola doble. `NullPointerException` – Si el elemento es nulo y la cola doble no admite elementos nulos. `IllegalArgumentException` – Si algún atributo del elemento del parámetro impide agregarlo a la colección.
`boolean offerLast(E e)` Inserta el elemento dado por el parámetro al final de esta cola doble a menos que se viole las restricciones de capacidad. Si la cola doble tiene restricciones de capacidad es preferible este método al de `addLast()`. Regresa `true` si se agregó el elemento a la cola, `false` en caso contrario. **Lanza**: `ClassCastException` – Si la clase del elemento impide que sea agregado a la cola doble. `NullPointerException` – Si el elemento es nulo y la cola doble no admite elementos nulos. `IllegalArgumentException` – Si algún atributo del elemento del parámetro impide agregarlo a la colección.
`E pollFirst()` Regresa y elimina el primer elemento de la cola doble si lo hay. En caso contrario regresa `null`.

Tabla 6.7 Métodos de la Interfaz `Deque<E>`. Cont.

`E pollLast()` Regresa y elimina el último elemento de la cola doble si lo hay. En caso contrario regresa `null`.
`E removeFirst()` Regresa y elimina el primer elemento de la cola doble. **Lanza**: `NoSuchElementException` – Si la cola doble está vacía.
`E removeLast()` Regresa y elimina el último elemento de la cola doble. **Lanza**: `NoSuchElementException` – Si la cola doble está vacía.
`boolean removeFirstOccurrence(Object o)` Remueve la primera ocurrencia del elemento dado por el parámetro de esta cola doble. Si la cola no contiene al elemento la deja sin alterar. Regresa `true` si se removió el elemento de la cola, `false` en caso contrario. **Lanza**: `ClassCastException` – Si la clase del elemento especificado por el parámetro es incompatible con los elementos de la cola doble. `NullPointerException` – Si el elemento es nulo y la cola doble no admite elementos nulos.
`boolean removeLastOccurrence(Object o)` Remueve la última ocurrencia del elemento dado por el parámetro de esta cola doble. Si la cola no contiene al elemento la deja sin alterar. Regresa `true` si se removió el elemento de la cola, `false` en caso contrario. **Lanza**: `ClassCastException` – Si la clase del elemento especificado por el parámetro es incompatible con los elementos de la cola doble. `NullPointerException` – Si el elemento es nulo y la cola doble no admite elementos nulos.
`E peelFirst()` Regresa, pero no elimina el primer elemento de la cola doble si lo hay. En caso contrario regresa `null`.
`E peekLast()` Regresa, pero no elimina el último elemento de la cola doble si lo hay. En caso contrario regresa `null`.
`E getFirst()` Regresa, pero no elimina el primer elemento de la cola doble. **Lanza**: `NoSuchElementException` – Si la cola doble está vacía.
`E getLast()` Regresa, pero no elimina el último elemento de la lista. **Lanza**: `NoSuchElementException` – Si la cola doble está vacía.

Tabla 6.7 Métodos de la Interfaz `Deque<E>`. Cont.

`void push(E e)` Inserta el elemento dado por el parámetro en la pila representada por esta cola, si es posible sin violar las restricciones de capacidad. Este método es equivalente al método `addFirst()`. **Lanza**: `IllegalStateException` – Si el elemento no puede agregarse en este momento debido a las restricciones de capacidad. `ClassCastException` – Si la clase del elemento impide que sea agregado a la cola doble. `NullPointerException` – Si el elemento es nulo y la cola doble no admite elementos nulos. `IllegalArgumentException – Si` algún atributo del elemento del parámetro impide agregarlo a la colección.
`E pop()` Extrae un elemento de la pila representada por esta cola. Este método es equivalente al método `removeFirst()`. **Lanza**: `NoSuchElementException` – Si la lista está vacía.
`Iterator<E> descendingIterator()` Regresa un iterador para los elementos de la cola doble en orden secuencial invertido. Los elementos serán regresados del último al primero.

Clase ArrayList<E>

La clase `ArrayList<E>` implementa la interfaz `List<E>` (y por lo tanto la interfaz `Collection<E>`) mediante un arreglo que cambia su tamaño de acuerdo a las necesidades. La clase `ArrayList<E>` permite todo tipo de elementos, incluso `null`. Además de los métodos que implementan la interfaz `List<E>`, esta clase provee métodos para manipular el tamaño del arreglo usado para almacenar la lista.

En la tabla 6.8 se muestran los métodos exclusivos de la clase `ArrayList<E>`.

Tabla 6.8 Métodos exclusivos de la Clase `Arraylist<E>`

`ArrayList()` Construye una lista vacía con una capacidad inicial de 10.
`ArrayList(int inicialCapacity)` Construye una lista vacía con la capacidad inicial dada por el parámetro. **Lanza**: `IllegalArgumentException` – Si el valor de la capacidad inicial es negativo.

Tabla 6.8 Métodos exclusivos de la Clase `Arraylist<E>`. Cont.

`ArrayList(Collection<? extends E>` *`c`*`)` Construye una lista que contiene los elementos de la colección del parámetro. En el orden en que son regresados por el iterador de la colección. La lista tiene una capacidad inicial del 110% del tamaño de la colección. **Lanza**: `NullPointerException` – Si la colección del parámetro es nula.
`Object clone()` Regresa una copia de la lista. Los elementos de la lista no son copiados.
`void trimToSize()` Reduce la capacidad de la lista al tamaño actual de la lista
`void ensureCapacity(int` *`minCapacity`*`)` Incrementa la capacidad mínima de la lista, si es necesario, para asegurarse de que tenga al menos la capacidad especificada por el parámetro.

Clase `LinkedList<E>`

La clase `LinkedList<E>` implementa las interfaces `List<E>` (y por lo tanto la interfaz `Collection<E>`) y `Deque<E>` (y por lo tanto la interfaz `Queue<E>`) mediante una lista ligada. La clase `LinkedList<E>` permite todo tipo de elementos, incluso `null`. Los métodos que implementa la clase `LinkedList<E>`, permite usarla como pila, cola o doble cola.

En la tabla 6.9 se muestran los métodos exclusivos de la clase `LinkedList<E`.

Tabla 6.9 Métodos exclusivos de la Clase `Linkedlist<E>`

`LinkedList()` Construye una lista vacía.
`LinkedList(Collection<? extends E>` *`c`*`)` Construye una lista que contiene los elementos de la colección del parámetro. En el orden en que son regresados por el iterador de la colección. **Lanza**: `NullPointerException` – Si la colección del parámetro es nula.
`Object clone()` Regresa una copia de la lista. Los elementos de la lista no son copiados.

Ejemplos Sobre Listas

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 listas, aunque las listas no permiten

almacenar permanentemente los datos. Más adelante se modificará este mecanismo para que emplee archivos y una base de datos.

Si al querer modificar o eliminar un dato, el dato no existe, se lanzará una excepción del tipo `DAOException`. El siguiente código muestra la implementación de la excepción `DAOException`.

DAOException.java

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

/**
 * Esta clase representa a las excepciones lanzadas por las clases
 * que se encargan de acceder a los datos en el mecanismo de persistencia.
 *
 * @author mdomitsu
 */
public class DAOException extends RuntimeException {
  /**
   * Constructor por omisión. Construye una excepción con un mensaje de error
   * nulo.
   */
  public DAOException() {
  }

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

  /**
   * Construye una excepción la causa original del error.
   * @param causa Causa original del error.
   */
  public DAOException(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 6.4

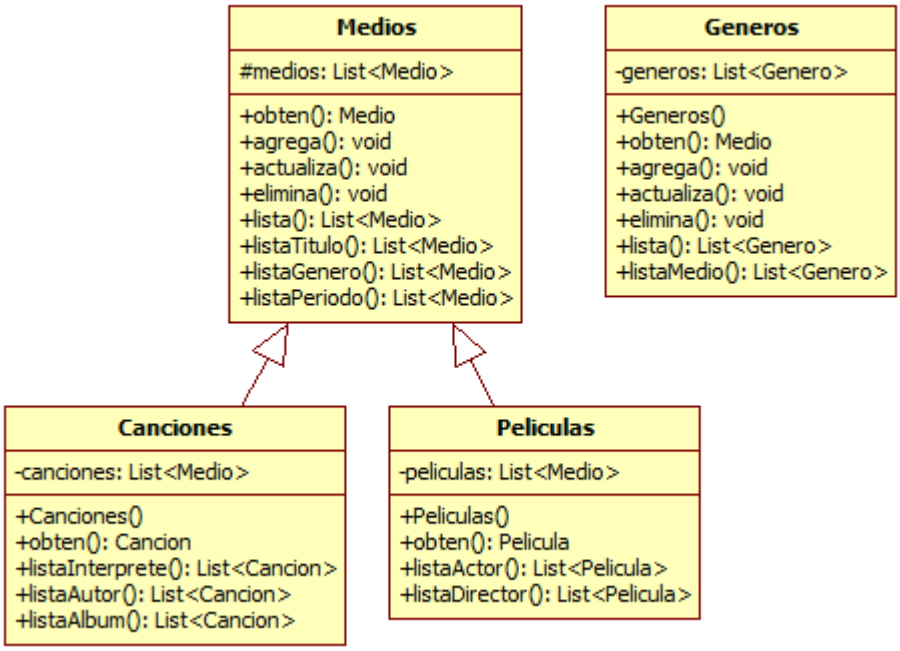

Figura 6.4

La clase `Medios` tiene como atributo una lista de tipo `Medio`. 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 del mismo título dado, del mismo género dado y del mismo período dado. El código de la clase `Medios` es el siguiente:

Medios.java

```
/*
 * Medios.java
 */
package dao;

import java.util.List;
import java.util.ArrayList;
import java.util.Iterator;
import objetosServicio.*;
import objetosNegocio.Medio;
import excepciones.DAOException;

/**
 * Esta clase permite almacenar, actualizar, eliminar y consultar
 * medios (canciones y películas) del programa AmanteMusica.
 * Los datos se almacenan en una lista del tipo ArrayList.
 *
 * @author mdomitsu
```

```
 */
public class Medios {
  protected List<Medio> medios;

  /**
   * Crea una lista para almacenar los medios
   */
  public Medios() {
    // Crea la lista para almacenar las canciones
    medios = new ArrayList<Medio>();
  }

  /**
   * Regresa el medio de la lista que coincida con el medio del parametro.
   * Las claves de los medios de la lista 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 = medios.indexOf(medio);

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

    // si no, regresa null
    return null;
  }

  /**
   * Agrega un medio a la lista de medios
   * @param medio Medio a agregar.
   */
  public void agrega(Medio medio) {
    // Agrega la canción o película a la lista
    medios.add(medio);
  }

  /**
   * Actualiza el medio de la lista que coincida con el medio del parametro.
   * Las claves de los medios de la lista y del parametro deben coincidir
   * @param medio Medio a actualizar.
   * @throws DAOException Si el medio no existe.
   */
  public void actualiza(Medio medio) throws DAOException  {
    // Busca un medio con la misma clave.
    int pos = medios.indexOf(medio);

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

    // Si lo hay, actualizalo
    medios.set(pos, medio);
  }

  /**
```

```
   * Elimina el medio de la lista que coincida con el medio del parametro.
   * Las claves de los medios de la lista y del parametro deben coincidir
   * @param medio Medio a eliminar.
   * @throws DAOException Si el medio no existe.
   */
  public void elimina(Medio medio) throws DAOException  {
    // Si el medio existe se borra, en caso contrario lanza excepción.
    if(¡medios.remove(medio))
        throw new DAOException("La Canción o Película no existe");
  }

  /**
   * Regresa una lista de todos los medios.
   * @return Lista de todos los medios
   */
  public List lista() {

    return medios;
  }

  /**
   * 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 List listaTitulo(String titulo) {
    List<Medio> lista = new ArrayList<Medio>();

    // Recorre la lista
    for(Iterator<Medio> iterador = medios.iterator(); iterador.hasNext(); ) {
      Medio medio = iterador.next();
      // Si es el título especificado
      if(titulo.equals(medio.getTitulo()))
        // Agrega el medio a la lista
        lista.add(medio);
    }

    return lista;
  }

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

    // Recorre la lista
    for(Iterator<Medio> iterador = medios.iterator(); iterador.hasNext(); ) {
      Medio medio = iterador.next();
      // Si es el género especificado
      if(cveGenero.equals(medio.getGenero().getCveGenero()))
        // Agrega el medio a la lista
        lista.add(medio);
    }
```

```
    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 List listaPeriodo(Periodo periodo) {
    List<Medio> lista = new ArrayList<Medio>();

    // Recorre la lista
    for(Iterator<Medio> iterador = medios.iterator(); iterador.hasNext(); ) {
      Medio medio = iterador.next();
      // Si es el periodo especificado
      if(periodo.contiene(medio.getFecha()))
        // Agrega el medio a la lista
        lista.add(medio);
    }

    return lista;
  }
}
```

La clase `Canciones` tiene una referencia a la lista de tipo `Medio` creada en la clase Medios. 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 de la clase `Canciones` es el siguiente.

Canciones.java

```
/*
 * Canciones.java
 */
package dao;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import objetosNegocio.Cancion;
import objetosNegocio.Medio;

/**
 * Esta clase permite consultar canciones del programa AmanteMusica
 * Los datos se almacenan en una lista del tipo ArrayList.
 *
 * @author mdomitsu
 */
public class Canciones extends Medios {
  // Crea la lista para almacenar las canciones
  private List<Medio> canciones;

  /**
   * Constructor vacio
   */
  public Canciones() {
```

```
    // Hace que la referencia canciones apunte a la lista medios
    // de la clase Medios
    canciones = medios;
  }

  /**
   * Regresa la canción de la lista 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);
  }

  /**
   * 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 List<Cancion> listaInterprete(String interprete) {
    List<Cancion> lista = new ArrayList<Cancion>();

    // Recorre la lista
   for(Iterator<Medio> iterador = canciones.iterator(); iterador.hasNext();){
      Cancion cancion = (Cancion)iterador.next();
      // Si es el intérprete especificado
      if(interprete.equals(cancion.getInterprete()))
        // Agrega la canción a la lista
        lista.add(cancion);
    }

    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 List<Cancion> listaAutor(String autor) {
    List<Cancion> lista = new ArrayList<Cancion>();

    // Recorre la lista
  for(Iterator<Medio> iterador = canciones.iterator(); iterador.hasNext();){
      Cancion cancion = (Cancion)iterador.next();
      // Si es el autor de la letra especificado
      if(autor.equals(cancion.getAutor()))
        // Agrega la canción a la lista
        lista.add(cancion);
    }
    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 List<Cancion> listaAlbum(String album) {
    List<Cancion> lista = new ArrayList<Cancion>();

    // Recorre la lista
   for(Iterator<Medio> iterador = canciones.iterator(); iterador.hasNext();){
      Cancion cancion = (Cancion)iterador.next();
      // Si es el álbum especificado
      if(album.equals(cancion.getAlbum()))
        // Agrega la canción a la lista
        lista.add(cancion);
    }

    return lista;
  }
}
```

La clase `Peliculas` tiene una referencia a la lista de tipo `Medio` creada en la clase Medios. 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 de la clase Peliculas es el siguiente.

Peliculas.java

```
/*
 * Peliculas.java
 */
package dao;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import objetosNegocio.Medio;
import objetosNegocio.Pelicula;

/**
 * Esta clase permite consultar películas del programa AmanteMusica
* Los datos se almacenan en una lista del tipo ArrayList.
 *
 * @author mdomitsu
 */
public class Peliculas extends Medios {
  // Crea la lista para almacenar las peliculas
  private List<Medio> peliculas;

  /**
   * Constructor vacio
   */
  public Peliculas() {
    // Hace que la referencia peliculas apunte a la lista medios
    // de la clase Medios
    peliculas = medios;
  }
```

```
  /**
   * Regresa la película de la lista 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);
  }

  /**
   * 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 List<Pelicula> listaActor(String actor) {
    List<Pelicula> lista = new ArrayList<Pelicula>();

    // Recorre la lista
    for(Iterator<Medio> iterador = peliculas.iterator();
         iterador.hasNext(); ) {
      Pelicula pelicula = (Pelicula)iterador.next();
      // Si es el actor especificado
      if(pelicula.getActor1().equals(actor) ||
         pelicula.getActor2().equals(actor))
        // Agrega la película a la lista
        lista.add(pelicula);
    }

    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 List<Pelicula> listaDirector(String director) {
    List<Pelicula> lista = new ArrayList<Pelicula>();

    // Recorre la lista
    for(Iterator<Medio> iterador = peliculas.iterator();
      iterador.hasNext(); ) {
      Pelicula pelicula = (Pelicula)iterador.next();
      // Si es el director especificado
      if(director.equals(pelicula.getDirector()))
        // Agrega la película a la lista
        lista.add(pelicula);
    }

    return lista;
  }
}
```

La clase `Generos` tiene como atributo una lista de tipo `Genero`. 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 de la clase `Generos` es el siguiente:

Generos.java

```
/*
 * Generos.java
 */
package dao;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import objetosNegocio.Genero;
import excepciones.DAOException;

/**
 * Esta clase permite almacenar, actualizar, eliminar y consultar
 * generos de canciones y películas del programa AmanteMusica.
 * Los datos se almacenan en listas.
 *
 * @author mdomitsu
 */
public class Generos {
  protected List<Genero> generos;

  /**
   * Crea una lista del tipo ArrayList para almacenar los géneros
   */
  public Generos() {
    // Crea la lista para almacenar los géneros
    generos = new ArrayList<Genero>();
  }

  /**
   * Regresa el genero de la lista que coincida con el genero del parametro.
   * Las claves del genero de la lista 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 = generos.indexOf(genero);

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

    // si no, regresa null
    return null;
  }

  /**
```

```
   * Agrega un genero a la lista de generos
   * @param genero Genero a agregar.
   */
  public void agrega(Genero genero) {
    // Se agrega el genero
    generos.add(genero);
  }

  /**
   * Actualiza el genero de la lista que coincida con el genero del parametro.
   * Las claves del genero dla lista y del parametro deben coincidir
   * @param genero Género a actualizar.
   * @throws DAOException Si el género no existe.
   */
  public void actualiza(Genero genero) throws DAOException {
    // Busca un genero con la misma clave.
    int pos = generos.indexOf(genero);

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

    // Si lo hay, actualizalo
    generos.set(pos, genero);
  }

  /**
   * Elimina el genero dla lista que coincida con el genero del parametro.
   * Las claves del genero dla lista y del parametro deben coincidir
   * @param genero Genero a eliminar.
   * @throws DAOException Si el género no existe.
   */
  public void elimina(Genero genero) throws DAOException {
    // Si el genero existe se borra, en caso contrario lanza excepción.
    if(generos.remove(genero))
        throw new DAOException("Género inexistente");
  }

  /**
   * Regresa una lista de todos los géneros.
   * @return ULista de todos los géneros
   */
  public List<Genero> lista() {
    return generos;
  }

  /**
   * Crea la lista de los géneros del mismo 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 List<Genero> listaMedio(char tipoMedio) {
    List<Genero> lista = new ArrayList<Genero>();

    // Recorre la lista
    for(Iterator<Genero> iterador = generos.iterator(); iterador.hasNext(); ) {
      Genero genero = iterador.next();
```

```
      // Si es el medio especificado
      if(tipoMedio == genero.getTipoMedio())
        // Agrega el medio a la lista
        lista.add(genero);
    }

    return lista;
  }
}
```

Diseño en Capas y la Fachada de la Capa de Persistencia

En el Tema 4: Excepciones, se habló del diseño en capas. Las clases `Medios`, `Canciones` y `Peliculas` que emplean listas 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 las listas 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 listas, archivos y base de datos. Aunque en una aplicación real almacenar los datos en listas no es una solución adecuada ya que las listas 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 6.5 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 listas:

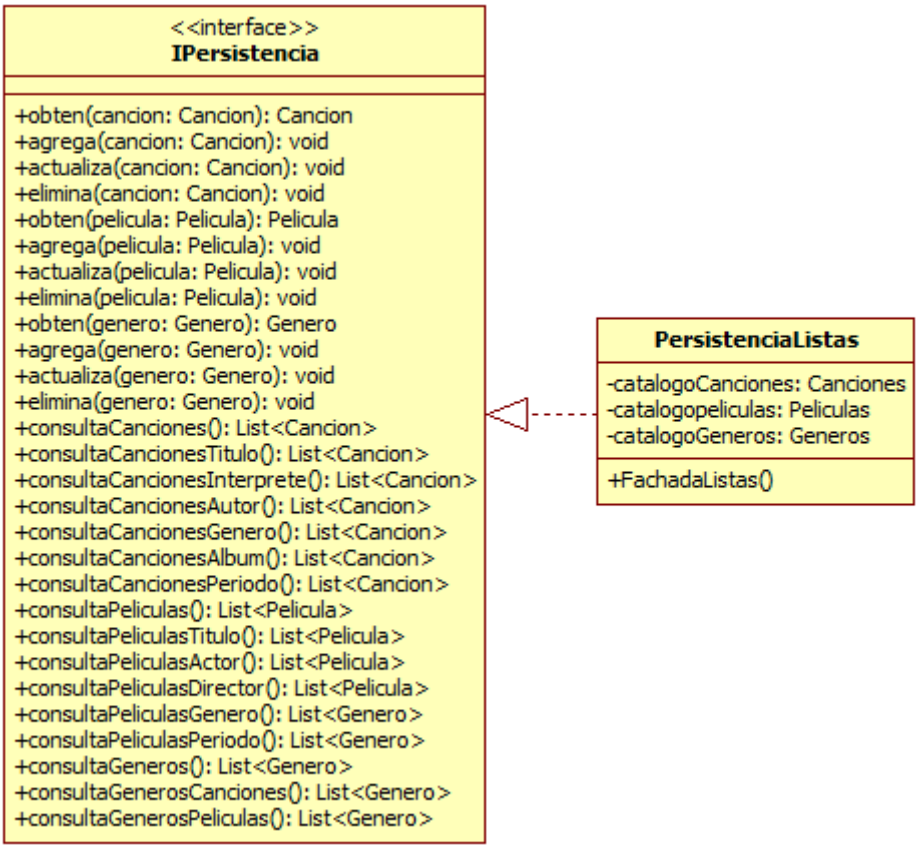

Figura 6.5 Fachada de la Capa de Persistencia del Programa AmanteMusica, Versión Listas

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

PersistenciaException.java

```
/*
 * PersistenciaException.java
 */
package excepciones;

/**
 * Esta clase representa a las excepciones lanzadas por la fachada
 * de la Capa de Persistencia y envuelve a las excepciones lanzadas
 * por las clases que encapsulan la persistencia.
 *
 * @author mdomitsu
 */
public class PersistenciaException extends RuntimeException {
  /**
   * 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);
  }
}
```

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

IPersistencia. java

```
/*
 * IPersistencia.java
 */
package interfaces;
```

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

/**
 * Esta Interfaz establece los métodos que deben implementar las clases
 * que encapsulen el mecanismo de persistencia del programa AmanteMusica
 *
 * @author mdomitsu
 */
public interface IPersistencia {
  /**
   * Obtiene una canción del catálogo de canciones
   * @param cancion Cancion a obtener
   * @return La canción si existe, null en caso contrario
   * @throws PersistenciaException Si no se puede acceder al
   * catálogo de canciones.
   */
  public Cancion obten(Cancion cancion) throws PersistenciaException;

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

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

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

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

  /**
```

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

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

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

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

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

  /**
   * Actualiza un género del catálogo de géneros
   * @param genero Género a actualizar
   * @throws PersistenciaException Si no se puede actualizar el género del
   * catálogo de géneros.
   */
  public void actualiza(Genero genero) throws PersistenciaException;

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

```
  /**
   * Obtiene una lista de todas las canciones
   * @return Lista de todas las canciones
   * @throws PersistenciaException Si hay un problema al acceder al catálogo
   * de canciones
   */
  public List<Cancion> consultaCanciones() throws PersistenciaException;

  /**
   * Obtiene una lista de todas las canciones con el mismo título.
   * @param titulo Titulo de las canciones de la lista
   * @return Lista de todas las canciones con el mismo título.
   * @throws PersistenciaException Si hay un problema al acceder al catálogo
   * de canciones
   */
  public List<Cancion> consultaCancionesTitulo(String titulo)
     throws PersistenciaException;

  /**
   * Obtiene una lista de todas las canciones con el mismo intérprete.
   * @param interprete Intérprete de las canciones de la lista
   * @return Lista de todas las canciones con el mismo intérprete.
   * @throws PersistenciaException Si hay un problema al acceder al catálogo
   * de canciones
   */
  public List<Cancion> consultaCancionesInterprete(String interprete)
     throws PersistenciaException;

  /**
   * Obtiene una lista de todas las canciones con el mismo autor.
   * @param autor Autor de las canciones de la lista
   * @return Lista de todas las canciones con el mismo autor.
   * @throws PersistenciaException Si hay un problema al acceder al catálogo
   * de canciones
   */
  public List<Cancion> consultaCancionesAutor(String autor)
     throws PersistenciaException;

  /**
   * Obtiene una lista de todas las canciones con el mismo género.
   * @param cveGenero Clave del género de las canciones de la lista
   * @return Lista de todas las canciones con el mismo género.
   * @throws PersistenciaException Si hay un problema al acceder al catálogo
   * de canciones
   */
  public List<Cancion> consultaCancionesGenero(String cveGenero)
     throws PersistenciaException;

  /**
   * Obtiene una lista de todas las canciones del mismo álbum.
   * @param album Álbum de las canciones de la lista
   * @return Lista de todas las canciones del mismo álbum.
   * @throws PersistenciaException Si hay un problema al acceder al catálogo
   * de canciones
   */
  public List<Cancion> consultaCancionesAlbum(String album)
```

```
      throws PersistenciaException;

   /**
    * Obtiene una lista de todas las canciones del mismo periodo.
    * @param periodo Período de las canciones de la lista
    * @return Lista de todas las canciones del mismo período.
    * @throws PersistenciaException Si hay un problema al acceder al catálogo
    * de canciones
    */
   public List<Cancion> consultaCancionesPeriodo(Periodo periodo)
      throws PersistenciaException;

   /**
    * Obtiene una lista de todas las películas
    * @return Lista de todas las películas.
    * @throws PersistenciaException Si hay un problema al acceder al catálogo
    * de películas
    */
   public List<Pelicula> consultaPeliculas() throws PersistenciaException;

   /**
    * Obtiene una lista de todas las películas del mismo título
    * @param titulo Título de las películas de la lista
    * @return Lista de todas las peliculas del mismo título.
    * @throws PersistenciaException Si hay un problema al acceder al catálogo
    * de películas
    */
   public List<Pelicula> consultaPeliculasTitulo(String titulo)
      throws PersistenciaException;

   /**
    * Obtiene una lista de todas las películas del mismo actor
    * @param actor Actor de las peliculas de la lista
    * @return Lista de todas las peliculas del mismo actor.
    * @throws PersistenciaException Si hay un problema al acceder al catálogo
    * de películas
    */
   public List<Pelicula> consultaPeliculasActor(String actor)
      throws PersistenciaException;

   /**
    * Obtiene una lista de todas las películas del mismo director
    * @param director Director de las peliculas de la lista
    * @return Lista de todas las peliculas del mismo director.
    * @throws PersistenciaException Si hay un problema al acceder al catálogo
    * de películas
    */
   public List<Pelicula> consultaPeliculasDirector(String director)
      throws PersistenciaException;

   /**
    * Obtiene una lista de todas las películas del mismo género
    * @param cveGenero Clave del género de las peliculas de la lista
    * @return Lista de todas las peliculas del mismo género.
    * @throws PersistenciaException Si hay un problema al acceder al catálogo
    * de películas
    */
```

```
  public List<Pelicula> consultaPeliculasGenero(String cveGenero)
     throws PersistenciaException;

  /**
   * Obtiene una lista de todas las películas del mismo periodo
   * @param periodo Periodo de las peliculas de la lista
   * @return Lista de todas las peliculas del mismo periodo.
   * @throws PersistenciaException Si hay un problema al acceder al catálogo
   * de películas
   */
  public List<Pelicula> consultaPeliculasPeriodo(Periodo periodo)
     throws PersistenciaException;

  /**
   * Obtiene una lista de todos los géneros
   * @return Lista de todos los géneros.
   * @throws PersistenciaException Si no se puede acceder al catálogo
   * de géneros.
   */
  public List<Genero> consultaGeneros() throws PersistenciaException;

  /**
   * Obtiene una lista de los géneros de canciones
   * @return Lista de los géneros canciones
   * @throws PersistenciaException Si no se puede acceder al catálogo
   * de géneros.
   */
  public List<Genero> consultaGenerosCanciones()
     throws PersistenciaException;

  /**
   * Obtiene una lista de los géneros de películas
   * @return Lista de los géneros películas
   * @throws PersistenciaException Si no se puede acceder al catálogo
   * de géneros.
   */
  public List<Genero> consultaGenerosPeliculas()
     throws PersistenciaException;
}
```

El siguiente código muestra la clase `PersistenciaListas` que implementa la fachada de la capa de persistencia del programa sobre el amante de la música, cuando el almacenamiento de los datos es en listas. Esta clase implementa la interfaz `IPersistencia`. Hay que notar que aunque en la interfaz `IPersistencia` se declararon que todos los métodos lanzan una excepción del tipo `PersistenciaException`, en las implementaciones de los métodos no se requiere que los métodos declaren ni lancen la excepción.

PersistenciaListas.java

```
/*
 * PersistenciaListas.java
 */

package persistencia;
```

```
import java.util.List;
import objetosServicio.*;
import objetosNegocio.*;
import excepciones.*;
import interfaces.IPersistencia;
import persistencia.*;

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

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

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

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

  /**
   * 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 PersistenciaException Si no se puede agregar la canción al
   * catálogo de canciones.
   */
  public void agrega(Cancion cancion) throws PersistenciaException {
    Cancion cancionBuscada;

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

```
    // Si lo hay, no se agrega a la lista
    if(cancionBuscada != null)
       throw new PersistenciaException("Canción repetida");

    // Agrega la nueva canción al catálogo
    catalogoCanciones.agrega(cancion);
  }

  /**
   * Actualiza una canción del catálogo de canciones
   * @param cancion Cancion a actualizar
   * @throws PersistenciaException Si no se puede actualizar la canción del
   * catálogo de canciones.
   */
  public void actualiza(Cancion cancion) throws PersistenciaException {
    try {
      // Actualiza la  canción del catálogo
      catalogoCanciones.actualiza(cancion);
    }
    catch(DAOException pae) {
      throw new PersistenciaException("No se puede actualizar la canción",
         pae);
    }
  }

  /**
   * Elimina una canción del catálogo de canciones
   * @param cancion Cancion a eliminar
   * @throws PersistenciaException Si no se puede eliminar la canción del
   * catálogo de canciones.
   */
  public void elimina(Cancion cancion) throws PersistenciaException {
    // Elimina la  canción del catálogo
    try {
      catalogoCanciones.elimina(cancion);
    }
    catch(DAOException pae) {
      throw new PersistenciaException("No se puede eliminar la canción",
         pae);
    }
  }

  /**
   * Obtiene una película del catálogo de peliculas
   * @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 peliculas. No se permiten películas
   * con claves repetidas
   * @param pelicula Pelicula a agregar
   * @throws PersistenciaException Si no se puede agregar la película al
```

```
   * catálogo de películas.
   */
  public void agrega(Pelicula pelicula) throws PersistenciaException {
    Pelicula peliculaBuscada;

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

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

    // Agrega la nueva película al catálogo
    catalogoPeliculas.agrega(pelicula);
  }

  /**
   * Actualiza una película del catálogo de peliculas
   * @param pelicula Pelicula a actualizar
   * @throws PersistenciaException Si no se puede actualizar la película del
   * catálogo de películas.
   */
  public void actualiza(Pelicula pelicula) throws PersistenciaException {
    // Actualiza la  película del catálogo
    try {
      catalogoPeliculas.actualiza(pelicula);
    }
    catch(DAOException pae) {
      throw new PersistenciaException("No se puede actualizar la película",
         pae);
    }
  }

  /**
   * Elimina una película del catálogo de peliculas
   * @param pelicula Pelicula a eliminar
   * @throws PersistenciaException Si no se puede eliminar la película del
   * catálogo de películas.
   */
  public void elimina(Pelicula pelicula) throws PersistenciaException {
    // Elimina la  película del catálogo
    try {
      catalogoPeliculas.elimina(pelicula);
    }
    catch(DAOException pae) {
      throw new PersistenciaException("No se puede eliminar la película",
         pae);
    }
  }

  /**
   * Obtiene un género del catálogo de géneros
   * @param genero Género a obtener
   * @return El género si existe, null en caso contrario
   * @throws PersistenciaException Si no se puede acceder al
   * catálogo de géneros.
   */
```

```
  public Genero obten(Genero genero) throws PersistenciaException {
    // 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 PersistenciaException Si no se puede agregar el género al
   * catálogo de géneros.
   */
  public void agrega(Genero genero) throws PersistenciaException {
    Genero generoBuscado;

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

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

    // Agrega el nuevo género al catálogo
      catalogoGeneros.agrega(genero);
  }

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

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

```
  }

  /**
   * Obtiene una lista todas las canciones
   * @return Lista con la lista de todas las canciones
   */
  public List<Cancion> 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 Lista con la lista de todas las canciones con el mismo
   * título.
   */
  public List<Cancion> 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 Lista con la lista de todas las canciones con el mismo
   * intérprete.
   */
  public List<Cancion> 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 Lista con la lista de todas las canciones con el mismo autor.
   */
  public List<Cancion> 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 Lista con la lista de todas las canciones con el mismo
   * género.
   */
  public List<Cancion> 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 Lista con la lista de todas las canciones del mismo álbum.
   */
  public List<Cancion> 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 Lista con la lista de todas las canciones del mismo período.
   */
  public List<Cancion> consultaCancionesPeriodo(Periodo periodo) {
    // Regresa el vector con la lista de canciones
    return catalogoCanciones.listaPeriodo(periodo);
  }

  /**
   * Obtiene una lista de todas las películas
   * @return Lista con la lista de todas las películas.
   */
  public List<Pelicula> 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 Lista con la lista de todas las peliculas del mismo título.
   */
  public List<Pelicula> 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 Lista con la lista de todas las peliculas del mismo actor.
   */
  public List<Pelicula> 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 Lista con la lista de todas las peliculas del mismo director.
   */
  public List<Pelicula> 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 peliculas de la lista
    * @return Lista con la lista de todas las peliculas del mismo género.
    */
  public List<Pelicula> consultaPeliculasGenero(String cveGenero) {
    // Regresa el vector con la lista de peliculas
    return catalogoPeliculas.listaGenero(cveGenero);
  }

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

  /**
   * Obtiene una lista de todos los géneros
   * @return Lista con la lista de todos los géneros.
   */
  public List<Genero> consultaGeneros() {
    // Regresa el vector con la lista de géneros
    return catalogoGeneros.lista();
  }

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

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

La siguiente clase es utilizada para probar la clase `PersistenciaListas` 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 `PersistenciaListas` y se le asigna una referencia del tipo de la interfaz IPersistencia, esto es válido ya que se considera que cualquier clase que implementa una interfaz es del tipo de la interfaz. La clase `PersistenciaListas` oculta el mecanismo de almacenamiento que son listas 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
 */
package pruebas;

import java.util.List;
import objetosServicio.*;
import objetosNegocio.*;
import excepciones.PersistenciaException;
import interfaces.IPersistencia;
import persistencia.PersistenciaListas;

/**
 * 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
    IPersistencia fachada = new PersistenciaListas();
    List<Genero> listaGeneros;
    List<Cancion> listaCanciones;
    List<Pelicula> listaPeliculas;
    Genero genero = null;
    Cancion cancion = null;

    // Se agrega el género 1 al catálogo de géneros
    try {
      fachada.agrega(new Genero("GCB0001", "Balada", 'C'));
      System.out.println("Se agrego el género 1 al catálogo de géneros");
    } catch(PersistenciaException 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(new Genero("GCB0002", "Bossanova", 'C'));
      System.out.println("Se agrego el género 2 al catálogo de géneros");
    } catch(PersistenciaException 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(new Genero("GCR0003", "Rock", 'C'));
      System.out.println("Se agrego el género 3 al catálogo de géneros");
    } catch(PersistenciaException 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(new Genero("GCB0001", "Balada", 'C'));
      System.out.println("Se agrego el género 1 al catálogo de géneros");
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage() + " 1");
    }

    // Se agrega el género 4 al catálogo de géneros
    try {
      fachada.agrega(new Genero("GPD0001", "Drama", 'P'));
      System.out.println("Se agrego el género 4 al catálogo de géneros");
    } catch(PersistenciaException 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(new Genero("GPC0002", "Ciencia Ficción", 'P'));
      System.out.println("Se agrego el género 5 al catálogo de géneros");
    } catch(PersistenciaException 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(new Genero("GPC0003", "Comedia", 'P'));
      System.out.println("Se agrego el género 6 al catálogo de géneros");
    } catch(PersistenciaException 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 {
      listaGeneros = fachada.consultaGeneros();
      System.out.println(listaGeneros);
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage());
    }
```

```
    // Se modifica el genero de clave "GCB0002", a "Samba"
    try {
      genero = fachada.obten(new Genero("GCB0002"));
      genero.setNombre("Samba");
      fachada.actualiza(genero);
      System.out.println("Se actualizo el genero de clave GCB0002");
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage() + "GCB0002");
    }

    // Se elimina el género "GPC0003" del catalogo de generos
    try {
      fachada.elimina(new Genero("GPC0003"));
      System.out.println("Se elimino el genero de clave GPC0003");
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage() + " GPC0003");
    }

    // Se elimina el género "GPM0004" (inexistente) del catalogo de generos
    try {
      fachada.elimina(new Genero("GPM0004"));
      System.out.println("Se elimino el genero de clave GPM0004");
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage() + " GPM0004");
    }

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

```
    }

    // Se obtiene el genero cuya clave es "GCB0001"
    try {
        genero = fachada.obten(new Genero("GCB0001"));
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage());
    }

    // Se agrega la canción 1 al catálogo de canciones
    try {
      fachada.agrega(new Cancion("CBB0001", "The long and winding road",
                                 genero, "The Beatles", "John Lennon",
                                 "Let it be", 3, new Fecha(24, 3, 1970)));
      System.out.println("Se agrego la cancion 1");
    } catch(PersistenciaException 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(new Cancion("CBB0001", "The long and winding road",
                                 genero, "The Beatles", "John Lennon",
                                 "Let it be", 3, new Fecha(24, 3, 1970)));
      System.out.println("Se agrego la cancion 1");
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage() + " 1");
    }

    // Se obtiene el genero cuya clave es "GCB0002"
    try {
        genero = fachada.obten(new Genero("GCB0002"));
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage());
    }

    // Se agrega la canción 2 al catálogo de canciones
    try {
      fachada.agrega(new Cancion("CSD0002", "Garota de Ipanema", genero,
                              "Los Indios Tabajaras", "Antonio Carlos Jobim",
                         "Bossanova Jazz Vol. 1", 3, new Fecha(1, 12, 1970)));
      System.out.println("Se agrega la cancion 2");
    } catch(PersistenciaException 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(new Cancion("CSB0003", "Desafinado", genero,
                                 "Joao Gilberto", "Joao Gilberto",
                         "Bossanova Jazz Vol. 1", 3, new Fecha(3, 12, 1980)));
      System.out.println("Se agrega la cancion 3");
```

```
    } catch(PersistenciaException 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 {
      listaCanciones = fachada.consultaCanciones();
      System.out.println(listaCanciones);
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage());
    }

    // Se actualiza la canción de clave "CBB0001" al genero "GCR0003"
    try {
      // Obtiene el género 3 del cátalogo de géneros de canciones
      genero = fachada.obten(new Genero("GCR0003"));
      if(genero != null) {
        // Obtiene la canción 1 del cátalogo de canciones
        cancion = fachada.obten(new Cancion("CBB0001"));
        if(cancion != null) {
          // Se actualiza la canción 1
          cancion.setGenero(genero);
          fachada.actualiza(cancion);
          System.out.println(
               "Se actualizo la canción de clave CBB0001 al genero GCR0003");
        } else System.out.println("No existe la canción CBB0001");
      } else System.out.println("No existe el género GCR0003");
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage());
    }

    // Se elimina la canción de clave "CSB0003"
    try {
        fachada.elimina(new Cancion("CSB0003"));
        System.out.println("Se elimina la cancion de clave CSB0003");
    } catch(PersistenciaException fe) {
        // Muestra el mensaje de error amistoso
        System.out.println("Error: " + fe.getMessage() + " CSB0003");
    }

    // Se lista el catálogo de canciones
    System.out.println("Lista de canciones:");
    try {
      listaCanciones = fachada.consultaCanciones();
      System.out.println(listaCanciones);
    } catch(PersistenciaException 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 {
```

```
      listaCanciones = fachada.consultaCancionesInterprete("The Beatles");
      System.out.println(listaCanciones);
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage());
    }

    // Se lista las canciones de samba, "GCB0002"
    System.out.println("Lista de canciones de Samba:");
    try {
      listaCanciones = fachada.consultaCancionesGenero("GCB0002");
      System.out.println(listaCanciones);
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage());
    }

    // Se obtiene el genero cuya clave es "GPD0001"
    try {
        genero = fachada.obten(new Genero("GPD0001"));
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage());
    }

    // Se agrega la película 1 al catálogo de películas
    try {
      fachada.agrega(new Pelicula("PED0001", "Casa Blanca", genero,
                                  "Humphrey Bogart", "Ingrid Bergman",
                             "Michael Curtiz", 102, new Fecha(1, 1, 1942)));
      System.out.println("Se agrego la película 1");
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage() + " 1");
    }

    // Se obtiene el genero cuya clave es "GPC0002"
    try {
        genero = fachada.obten(new Genero("GPC0002"));
    } catch(PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage());
    }
    // Se agrega la película 2 al catálogo de películas
    try {
      fachada.agrega(new Pelicula("PCF0002", "2001 Space Odyssey",
                                  genero, "Keir Dullea", "Gary Lockwood",
                            "Stanley Kubrick", 141, new Fecha(1, 1, 1968)));
      System.out.println("Se agrego la película 2");
    } catch(PersistenciaException 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 {
```

```
      listaPeliculas = fachada.consultaPeliculas();
      System.out.println(listaPeliculas);
    } catch(PersistenciaException 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 {
      listaPeliculas = fachada.consultaPeliculasActor("Ingrid Bergman");
      System.out.println(listaPeliculas);
    } catch(PersistenciaException 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 {
      listaPeliculas = fachada.consultaPeliculasPeriodo(periodo);
      System.out.println(listaPeliculas);
    } catch(PersistenciaException 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:

```
...
Se agrego el género 1 al catálogo de géneros
Se agrego el género 2 al catálogo de géneros
Se agrego el género 3 al catálogo de géneros
Error: Género repetido 1
Se agrego el género 4 al catálogo de géneros
Se agrego el género 5 al catálogo de géneros
Se agrego el género 6 al catálogo de géneros
Lista de géneros
[GCB0001, Balada, C, GCB0002, Bossanova, C, GCR0003, Rock, C, GPD0001,
Drama, P, GPC0002, Ciencia Ficción, P, GPC0003, Comedia, P]
Se actualizo el genero de clave GCB0002
Se elimino el genero de clave GPC0003
Error: No se puede eliminar el género GP004
Lista de géneros
[GCB0001, Balada, C, GCB0002, Samba, C, GCR0003, Rock, C, GPD0001,
Drama, P, GPC0002, Ciencia Ficción, P]
Lista de géneros de canciones
[GCB0001, Balada, C, GCB0002, Samba, C, GCR0003, Rock, C]
Lista de géneros de películas
[GPD0001, Drama, P, GPC0002, Ciencia Ficción, P]
...
```

Podemos notar el mensaje de error que se despliega querer eliminar un género inexistente aunque. Este mensaje de error es amistoso para el usuario ya que la excepción original lanzada por el método `elimina()` de la clase Generos es atrapada y relanzada por la clase Fachada. Este 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 `PersistenciaListas`:

```
...
// Se elimina el género "GPM0004" (inexistente) del catalogo de generos
try {
  fachada.elimina(new Genero("GPM0004"));
  System.out.println("Se elimino el genero de clave GPM0004");
} catch(PersistenciaException fe) {
  // Muestra el mensaje de error amistoso
  System.out.println("Error: " + fe.getMessage() + " GPM0004");
  fe.printStackTrace();
}
...
```

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

```
...
excepciones.PersistenciaException: No se puede eliminar el género
        at
persistencia.PersistenciaListas.elimina(PersistenciaListas.java:227)
        at pruebas.Prueba.main(Prueba.java:135)
...
```

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

```
...
excepciones.PersistenciaException: No se puede eliminar el género
        at
persistencia.PersistenciaListas.elimina(PersistenciaListas.java:227)
        at pruebas.Prueba.main(Prueba.java:135)
Caused by: excepciones.DAOException: Género inexistente
        at dao.Generos.elimina(Generos.java:86)
```

```
        at
persistencia.PersistenciaListas.elimina(PersistenciaListas.java:224)
        ... 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.

Conjuntos

Un conjunto es una colección que no contiene elementos duplicados. Más formalmente, un conjunto no contiene pares de elementos `e1` y `e2` tal que `e1.equals(e2)`. Esta colección modela la abstracción matemática de conjunto.

En la figura 6.6 se muestran las interfaces y clases que implementan las listas.

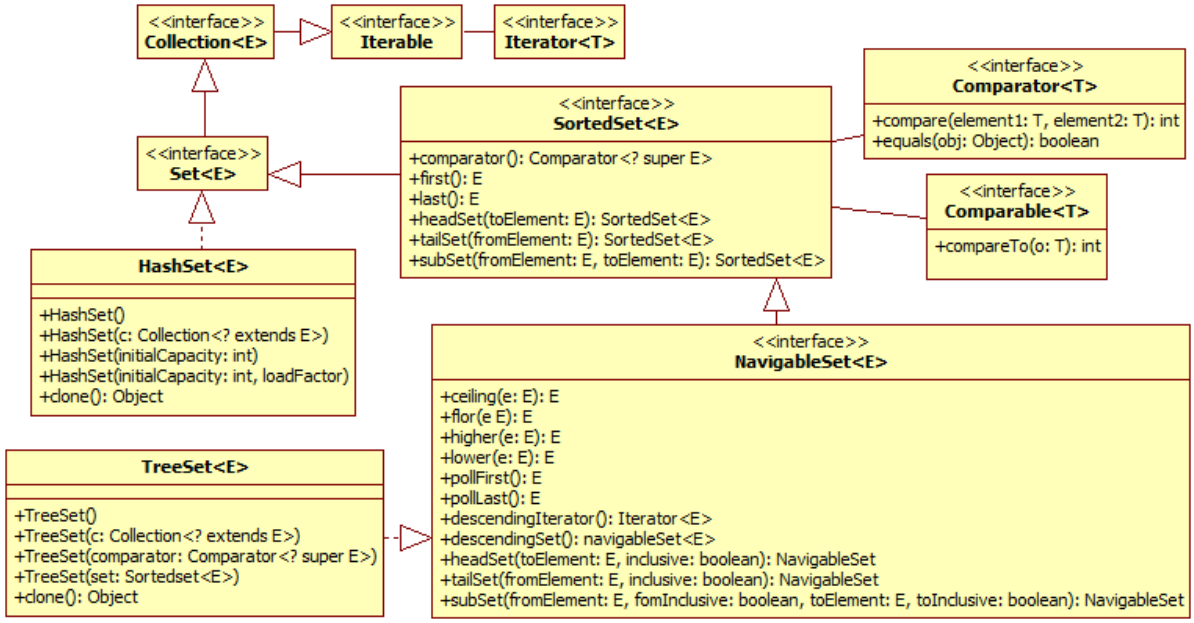

Figura 6.6

Interfaz `Set<T>`

La interfaz `Set<T>` establece condiciones adicionales a las heredadas de la interfaz `Collection<T>`, en los contratos de todos los constructores y en los contratos de los métodos `add()`, `equals()` y `hashCode()`.

Interfaz `Comparator<T>`

Esta interfaz declara un método de comparación que impone un ordenamiento total en algunas colecciones de objetos. Los comparadores pueden pasarse como parámetro a un método de ordenamiento como `Collections.sort()` para permitir un control preciso del ordenenamiento. Los comparadores también pueden usarse para para controlar ciertas estructuras de datos, tal como `Treeset` o `TreeMap`.

En la tabla 6.10 se muestran los métodos declarados por la interfaz `Comparator<E>`.

Tabla 6.10 Métodos de la Interfaz `Comparator<E>`

`int` **`compare`**`(T element1, T element2)` Compara sus dos argumentos por orden. Regresa un entero negativo, cero o un entero positivo si el primer argumento es menor, igual o mayor que el segundo. La clase que implemente esta interfaz debe asegurarse que: • `sgn(compare(x, y)) == -sgn(compare(y, x))`. • La relación sea transitiva. `((compare(x, y)>0) && (compare(y, z)>0))` implica que `compare(x, z)>0`. • `compare(x, y)==0` implica que `sgn(compare(x, z))==sgn(compare(y, z))` para toda `z`. **Lanza**: `ClassCastException` – Si los objetos de sus parámetros no pueden ser comparados.
`boolean` **`equals`**`(Object object)` Regresa verdadero sólo si el objeto del parámetro es un comparador y es igual a este comparador.

Interfaz `SortedSet<E>`

La interfaz `SortedSet<E>` garantiza que su iterador recorrerá el conjunto en orden ascendente de sus elementos, o por un comparador establecido por su constructor. Todos los elementos que se inserten en un conjunto ordenado deben implementar la interfaz `Comparable` (o ser aceptados por el comparador especificado). En la tabla 6.11 se muestran los métodos declarados por la interfaz `SortedSet<E>`.

Tabla 6.11 Métodos de la Interfaz `SortedSet<E>`

`Comparator<? super E>` **`comparator`**`()` Regresa el comparador asociado con este conjunto ordenado, o nulo si usa el ordenamiento de elementos natural.
`E` **`first`**`()` Regresa el primer elemento en este conjunto ordenado. **Lanza**: `NoSuchElementException` – Si el conjunto ordenado está vacío.

Tabla 6.11 Métodos de la Interfaz `SortedSet<E>`. Cont.

`E` **`last`**`()` Regresa el último elemento en este conjunto ordenado. **Lanza**: `NoSuchElementException` – Si el conjunto ordenado está vacío.
`SortedSet<E>` **`headSet`**`(E toElement)` Regresa una vista de una porción de la este conjunto ordenado cuyos elementos son estrictamente menores que `toElement`. Cualquier cambio en el subconjunto afecta al conjunto y viceversa. **Lanza**: `ClassCastException` – Si `toElement` no es compatible con el comparador de este conjunto. `IllegalArgumentException` – Si este conjunto es un subconjunto, un headset o tailset y `toElement` no está en el rango especificado del subconjunto, headset o tailset. `NullPointerException` – Si `toElement` es nulo y la lista no admite elementos nulos.
`SortedSet<E>` **`tailSet`**`(E fromElement)` Regresa una vista de una porción de la este conjunto ordenado cuyos elementos son mayores o iguales que `fromElement`. Cualquier cambio en el subconjunto afecta al conjunto y viceversa. **Lanza**: `ClassCastException` – Si `fromElement` no es compatible con el comparador de este conjunto. `IllegalArgumentException` – Si este conjunto es un subconjunto, un headset o tailset y `fromElement` no está en el rango especificado del subconjunto, headset o tailset. `NullPointerException` – Si `fromElement` es nulo y la lista no admite elementos nulos.
`SortedSet<E>` **`subSet`**`(E fromElement, E toElement)` Regresa una vista de una porción de la este conjunto ordenado cuyos elementos están en el rango de `fromElement` inclusive hasta `toElement` exclusive. Si `fromElement` y `toElement` son iguales, el subconjunto es vacío. Cualquier cambio en el subconjunto afecta al conjunto y viceversa. **Lanza**: `ClassCastException` – Si `fromElement` y `toElement` no pueden compararse entre si usando el comparador de este conjunto. `IllegalArgumentException` – Si `fromElement` es mayor que `toElement`; o si el conjunto es un subconjunto, un headset o tailset y `fromElement` o `toElement` no están en el rango especificado del subconjunto, headset o tailset. `NullPointerException` – Si `fromElement` o `toElement` es nulo y la lista no admite elementos nulos.

Interfaz `Comparable<T>`

Esta interfaz impone un ordenamiento total en cada clase que lo implementa. Este ordenamiento se conoce como el ordenamiento natural y su único método `compareTo()` se conoce como el método de comparación natural.

Las listas (y los arreglos) de objetos que impletan esta interfaz pueden ordenarse automáticamente por el método de ordenamiento como `Collections.sort()` (o `Arrays.sort()`). para permitir un control preciso del ordenenamiento. Los comparadores también pueden usarse para para controlar ciertas estructuras de datos, tal como `Treeset` o `TreeMap`. Los objetos que implementanesta interfazpueden usarse como llaves en un mapa ordenado o como elementos en un conjunto ordenado, sin la ayuda de un comparador.

En la tabla 6.12 se muestran los métodos declarados por la interfaz `Comparable<T>`.

Tabla 6.12 Métodos de la Interfaz `Comparable<T>`

`int` **`compareTo`**`(T o)` Compara este objeto con el objeto o de su parámetro.sus dos argumentos por orden. Regresa un entero negativo, cero o un entero positivo si este objeto es menor, igual o mayor que el objeto del parámetro. **Lanza**: `ClassCastException` – Si este objeto no puede compararse con el objeto del parámetro.

Interfaz `NavigableSet<E>`

La interfaz `NavigableSet<E>` extiende la interfaz `SortedSet<E>` con métodos de navegación: `lower()`, `floor()`, `ceiling()` y `higher()` que reportan las coincidencias más cercanas para una búsqueda dada. Un conjunto navegable puede accederse en orden ascendente o descendente. En la tabla 6.13 se muestran los métodos declarados por la interfaz `NavigableSet<E>`.

Tabla 6.13 Métodos de la Interfaz `NavigableSet<E>`

`E` **`ceiling`**`(E e)` Regresa el elemento menor que sea mayor o igual que el elemento del parámetro, o null si no existe ese elemento. **Lanza**: `ClassCastException` – Si el elemento del parámetro no puede compararse con los elementos del conjunto. `NullPointerException` – Si el elemento del parámetro es nulo y el conjunto no admite elementos nulos.
`E` **`floor`**`(E e)` Regresa el elemento mayor que sea menor o igual que el elemento del parámetro, o null si no existe ese elemento. **Lanza**: `ClassCastException` – Si el elemento del parámetro no puede compararse con los elementos del conjunto. `NullPointerException` – Si el elemento del parámetro es nulo y el conjunto no admite elementos nulos.

Tabla 6.13 Métodos de la Interfaz `NavigableSet<E>`. Cont.

`E` **`higher`**`(E e)` Regresa el elemento menor que sea mayor o igual que el elemento del parámetro, o null si no existe ese elemento. **Lanza**: `ClassCastException` – Si el elemento del parámetro no puede compararse con los elementos del conjunto. `NullPointerException` – Si el elemento del parámetro es nulo y el conjunto no admite elementos nulos.
`E` **`lower`**`(E e)` Regresa el elemento mayor que sea estrictamente menor que el elemento del parámetro, o null si no existe ese elemento. **Lanza**: `ClassCastException` – Si el elemento del parámetro no puede compararse con los elementos del conjunto. `NullPointerException` – Si el elemento del parámetro es nulo y el conjunto no admite elementos nulos.
`E` **`pollFirst`**`()` Elimina y regresa el primer (el menor) elemento, o null si el conjunto está vacío.
`E` **`pollLast`**`()` Elimina y regresa el último (el mayor) elemento, o null si el conjunto está vacío.
`Iterator<E>` **`descendingIterator`**`()` Regresa un iterador sobre los elementos de este conjunto, en orden descendente.
`NavigableSet<E>` **`descendingSet`**`()` Regresa una vista en orden invertido de los elementos de este conjunto. El conjunto descendente está respaldado por este conjunto. Los cambios en el conjunto se reflejan en el conjunto descendente y viceversa.
`NavigableSet<E>` **`headSet`**`(E toElement, boolean inclusive)` Regresa una vista de la porción de este conjunto cuyos elementos son menores (o iguales si el parámetro `inclusive` es verdadero) que el elemento del parámetro. El conjunto regresado está respaldado por este conjunto. Los cambios en el conjunto se reflejan en el conjunto regresado y viceversa. **Lanza**: `ClassCastException` – Si el elemento del parámetro no es compatible con el comparador (o si el conjunto no tiene comparador o si el elemento del parámetro no implementa la interfaz comparable). `NullPointerException` – Si el elemento del parámetro es nulo y el conjunto no admite elementos nulos. `IllegalArgumentException` – Si este conjunto tiene un rango restringido y el elemento del parámetro cae fuero del rengo del parámetro.

Tabla 6.13 Métodos de la Interfaz `NavigableSet<E>`. Cont.

`NavigableSet<E>` **`tailSet`**`(E fromElement, boolean inclusive)` Regresa una vista de la porción de este conjunto cuyos elementos son mayores (o iguales si el parámetro `inclusive` es verdadero) que el elemento del parámetro. El conjunto regresado está respaldado por este conjunto. Los cambios en el conjunto se reflejan en el conjunto regresado y viceversa. **Lanza**: `ClassCastException` – Si el elemento del parámetro no es compatible con el comparador (o si el conjunto no tiene comparador o si el elemento del parámetro no implementa la interfaz comparable). `NullPointerException` – Si el elemento del parámetro es nulo y el conjunto no admite elementos nulos. `IllegalArgumentException` – Si este conjunto tiene un rango restringido y el elemento del parámetro cae fuero del rengo del parámetro.

`NavigableSet<E>` **`subSet`**`(E fromElement, boolean fromInclusive,`
`E toElement, boolean toInclusive)`
 	
Regresa una vista de la porción de este conjunto cuyos elementos están en el rango de `fromElement` a `toElement`, si `fromElement` y `toElement` son iguales, el conjunto regresado es vacío a menos que `fromInclusive` y `toInclusive` sean ambas verdaderas. El conjunto regresado está respaldado por este conjunto. Los cambios en el conjunto se reflejan en el conjunto regresado y viceversa.
 	
Lanza:
`ClassCastException` – Si el elemento del parámetro no es compatible con el comparador (o si el conjunto no tiene comparador o si el elemento del parámetro no implementa la interfaz comparable).
`NullPointerException` – Si el elemento del parámetro es nulo y el conjunto no admite elementos nulos.
`IllegalArgumentException` – Si `fromElement` es mayor que `toElement`, o si este conjunto tiene un rango restringido y el elemento del parámetro cae fuera del rengo del parámetro.

Clase `HashSet<E>`

Esta clase implementa la interfaz `Set`, sobre una instancia de `HashMap`. No hay garantías sobre el orden de iteración del conjunto; en particular no hay garantía que el orden permanecera constante con el tiempo. Esta clase permite el elemento nulo.

En la tabla 6.14 se muestran los métodos exclusivos de la clase `HashSet<E>`.

Tabla 6.14 Métodos exclusivos de la Clase `HashSet<E>`

`HashSet``()` Construye un conjunto vacío con una capacidad inicial de 10 y un factor de carga de 0.75.

Tabla 6.14 Métodos exclusivos de la Clase `HashSet<E>`

`HashSet``(Collection<? extends E> c)` Construye un conjunto que contiene los elementos de la colección del parámetro. El factor de carga es de 0.75. El conjunto tiene una capacidad inicial suficiente para contener la colección. **Lanza**: `NullPointerException` – Si la colección del parámetro es nula.
`HashSet``(int` *`inicialCapacity`*`, float loadfactor)` Construye un conjunto vacío con la capacidad inicial y factor de carga dados por los parámetros. **Lanza**: `IllegalArgumentException` – Si el valor de la capacidad inicial es negativo o el factor de carga sea no positivo.
`HashSet``(int` *`inicialCapacity`*`)` Construye un conjunto vacío con la capacidad inicial dada por el parámetro y un factor de carga de 0.75. **Lanza**: `IllegalArgumentException` – Si el valor de la capacidad inicial es negativo.
`Object clone()` Regresa una copia del conjunto. Los elementos del conjunto no son copiados.

Clase `TreeSet<E>`

Esta clase implementa la interfaz `NavigableSet`, sobre una instancia de `TreeMap`. Esta clase garantiza que el conjunto estará ordenado en orden ascendente, de acuerdo al orden natural de sus elementos o por un comparador establecido en el constructor.

En la tabla 6.15 se muestran los métodos exclusivos de la clase `TreeSet<E>`.

Tabla 6.15 Métodos exclusivos de la Clase `TreeSet<E>`

`TreeSet``()` Construye un nuevo conjunto vacío ordenado de acuerdo al orden natural de sus elementos.
`TreeSet``(Collection<? extends E> c)` Construye un nuevo conjunto con los elementos de la colección del parámetro, ordenados de acuerdo al orden natural de los elementos. **Lanza**: `ClassCastException` – Si los elementos de la colección del parámetro no son comparables o no son mutuamente comparables. `NullPointerException` – Si la colección del parámetro es nula.
`TreeSet``(Comparator<? super E> comparator)` Construye un nuevo conjunto vacío, ordenado de acuerdo al comparador del parámetro.

Tabla 6.15 Métodos exclusivos de la Clase `TreeSet<E>`. Cont.

`TreeSet``(SortedSet<E> set)` Construye un nuevo conjunto con los elementos del conjunto ordenado del parámetro, ordenados de acuerdo al mismo orden. **Lanza**: `NullPointerException` – Si conjunto ordenado del parámetro es nulo.
`Object clone()` Regresa una copia del conjunto. Los elementos del conjunto no son copiados.

Ejemplo Sobre Conjuntos

Para realizar un estudio sobre el aprovechamiento escolar se requiere obtener información sobre los cursos que estudian los alumnos. Para ello se requiere un programa que capture en conjuntos ordenados los alumnos que estudian cada materia. El programa deberá generar conjuntos con diferentes combinaciones de alumnos y las materias que llevan. El siguiente diagrama de clases contiene las clases del programa y el código de algunas de ellas se muestra a continuación:

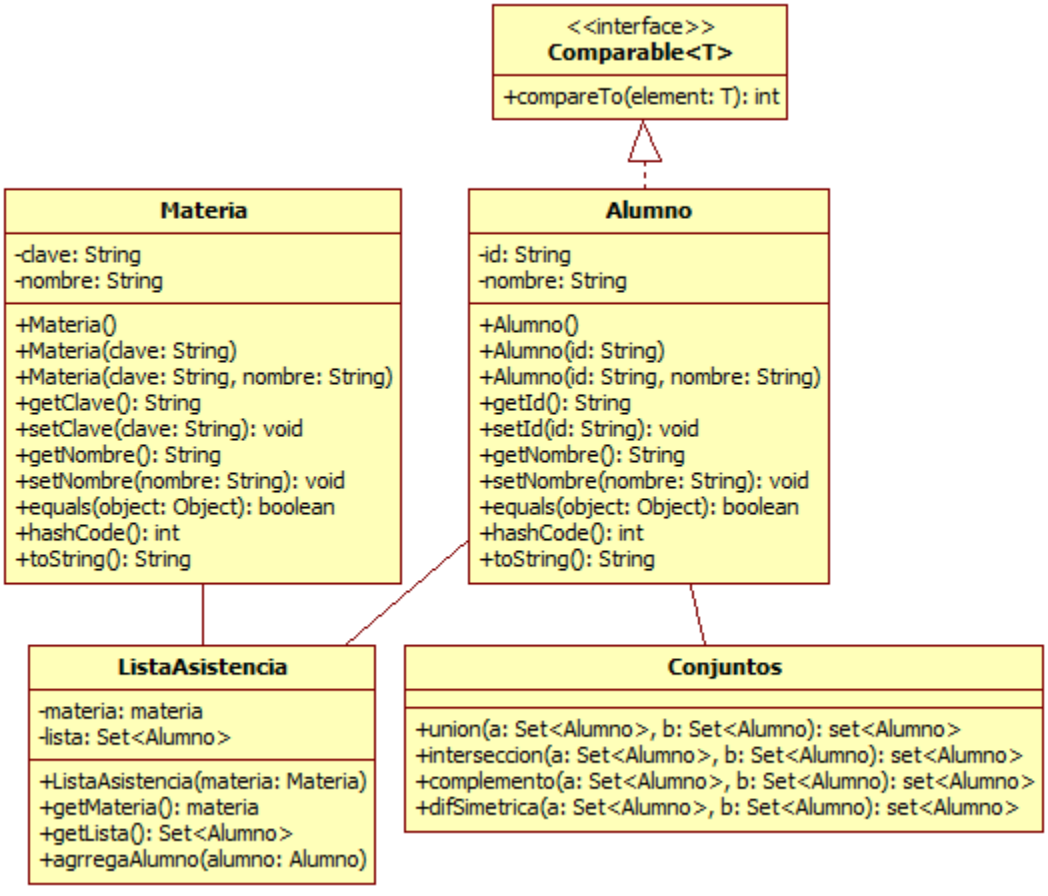

Figura 6.7

Alumno.java

```
/*
 * Alumno.java.
 *
 * @author mdomitsu
 */
package objetosNegocio;

/**
 * Esta clase representa un alumno
 */
public class Alumno implements Comparable<Alumno> {
    private String id;
    private String nombre;

    /**
     * Constructor vacio
     */
    public Alumno() {
    }

    /**
     * Constructor que inicializa el id del alumno
     * @param id Id del alumno
     */
    public Alumno(String id) {
        this.id = id;
    }

    /**
     * Constructor que inicializa los atributos de un alumno
     * @param id Id del alumno
     * @param nombre Nombre del alumno
     */
    public Alumno(String id, String nombre) {
        this.id = id;
        this.nombre = nombre;
    }

    ...

    /**
     * Compara este alumno con el alumno del parametro usando sus id
     * como cadenas usando el método compareTo.
     * @param alumno Alumno con el que se compara este alumno.
     * @return 0 si son iguales, positivo si el este alumno es mayor que
     * el alumno del parametro, positivo si este alumno es menor que el
     * alumno del parametro
     */
    @Override
    public int compareTo(Alumno alumno) {
        return id.compareTo(alumno.id);
    }

    @Override
    public String toString() {
```

```
        return id + ", " + nombre;
    }
}
```

ListaAsistencia.java

```
/*
 * ListaAsistencia.java
 *
 * @author mdomitsu
 */
package objetosNegocio;

import java.util.Set;
import java.util.TreeSet;

/**
 * esta clase representa una lista de alumnos implementada sobre un
 * conjunto ordenado.
 */
public class ListaAsistencia {

    private Materia materia;
    private Set<Alumno> listaAlumnos;

    /**
     * Constructor. Inicializa el atributo materia al valor de su parametro
     * @param materia
     */
    public ListaAsistencia(Materia materia) {
        this.materia = materia;

        // Crea un conjunto ordenado para almacenar alumnos de la materia
        listaAlumnos = new TreeSet();
    }

    /**
     * Regresa la materia de la lista de alumnos
     * @return La materia de la lista de alumnos
     */
    public Materia getMateria() {
        return materia;
    }

    /**
     * Regresa la lista de alumnos de la materia
     * @return La lista de alumnos de materia.
     */
    public Set<Alumno> getListaAlumnos() {
        return listaAlumnos;
    }

    /**
     * Agrega un alumno al conjunto ordenado que representa la lista de
     * alumnos de la materia
     * @param alumno a aggregar a la lista de alumnos de la materia
     */
```

```
    public void agregaAlumno(Alumno alumno) {
        listaAlumnos.add(alumno);
    }
}
```

Para probar las clases anteriores se tiene el siguiente programa de prueba:

Prueba.java

```
/*
 * Prueba.java.
 *
 * @author mdomitsu
 */

package pruebas;

import java.util.Set;
import objetosNegocio.Alumno;
import objetosNegocio.ListaAsistencia;
import objetosNegocio.Materia;

public class Prueba {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // Se crean cinco alumnos
        Alumno a1 = new Alumno("000123", "Juan Perez");
        Alumno a2 = new Alumno("000125", "Lourdes Serrano");
        Alumno a3 = new Alumno("000135", "Jose Sierra");
        Alumno a4 = new Alumno("000153", "Mario Lopez");
        Alumno a5 = new Alumno("000170", "Ana Benitez");

        // Se crean seis listas de asistencia
        ListaAsistencia la1 =
           new ListaAsistencia(new Materia("003753", "Integridad Personal"));
        ListaAsistencia la2 =
           new ListaAsistencia(new Materia("003768", "Calcculo I"));
        ListaAsistencia la3 =
           new ListaAsistencia(new Materia("003773", "Matematica Discreta"));
        ListaAsistencia la4 =
           new ListaAsistencia(new Materia("003774", "Programación I"));
        ListaAsistencia la5 =
           new ListaAsistencia(
              new Materia("003775", "Arquitectura de Computadoras"));
        ListaAsistencia la6 =
           new ListaAsistencia(new Materia("003865", "Vida Saludable"));

        // Se agregan alumnos a la clase "Integridad Personal"
        la1.agregaAlumno(a1);
        la1.agregaAlumno(a2);
        la1.agregaAlumno(a4);
        la1.agregaAlumno(a3);

        // Lista los alumnos de la clase "Integridad Personal"
```

```
        System.out.println(
                    "Lista de alumnos de la clase Integridad Personal");
        System.out.println(la1.getListaAlumnos());

        // Se agregan alumnos a la clase "Calculo I"
        la2.agregaAlumno(a2);
        la2.agregaAlumno(a4);
        la2.agregaAlumno(a3);

        // Lista los alumnos de la clase "Calculo I"
        System.out.println("Lista de alumnos de la clase Calculo I");
        System.out.println(la2.getListaAlumnos());

        // Se agregan alumnos a la clase "Matematica Discreta"
        la3.agregaAlumno(a1);
        la3.agregaAlumno(a5);
        la3.agregaAlumno(a4);

        // Lista los alumnos de la clase "Matematica Discreta"
        System.out.println(
                       "Lista de alumnos de la clase Matematica Discreta");
        System.out.println(la3.getListaAlumnos());

        // Se agregan alumnos a la clase "Programación I"
        la4.agregaAlumno(a4);
        la4.agregaAlumno(a3);
        la4.agregaAlumno(a5);

        // Lista los alumnos de la clase "Programación I"
        System.out.println("Lista de alumnos de la clase Programación I");
        System.out.println(la4.getListaAlumnos());

         // Se agregan alumnos a la clase "Arquitectura de Computadoras"
        la5.agregaAlumno(a2);
        la5.agregaAlumno(a1);
        la5.agregaAlumno(a4);
        la5.agregaAlumno(a5);

        // Lista los alumnos de la clase "Arquitectura de Computadoras"
        System.out.println(
             "Lista de alumnos de la clase Arquitectura de Computadoras");
        System.out.println(la5.getListaAlumnos());

         // Se agregan alumnos a la clase "Vida Saludable"
        la6.agregaAlumno(a2);
        la6.agregaAlumno(a1);
        la6.agregaAlumno(a4);
        la6.agregaAlumno(a5);
        la6.agregaAlumno(a3);

        // Lista los alumnos de la clase "Vida Saludable"
        System.out.println("Lista de alumnos de la clase Vida Saludable");
        System.out.println(la6.getListaAlumnos());

        ...
    }
}
```

La corrida de la clase prueba arroja el siguiente resultado. Note que aunque los alumnos se agregaron en desorden en el conjunto los alumnos se ordenan, como se ve en el listado:

```
Lista de alumnos de la clase Integridad Personal
[000123, Juan Perez, 000125, Lourdes Serrano, 000135, Jose Sierra,
000153, Mario Lopez]
Lista de alumnos de la clase Calculo I
[000125, Lourdes Serrano, 000135, Jose Sierra, 000153, Mario Lopez]
Lista de alumnos de la clase Matematica Discreta
[000123, Juan Perez, 000153, Mario Lopez, 000170, Ana Benitez]
Lista de alumnos de la clase Programación I
[000135, Jose Sierra, 000153, Mario Lopez, 000170, Ana Benitez]
Lista de alumnos de la clase Arquitectura de Computadoras
[000123, Juan Perez, 000125, Lourdes Serrano, 000153, Mario Lopez,
000170, Ana Benitez]
Lista de alumnos de la clase Vida Saludable
[000123, Juan Perez, 000125, Lourdes Serrano, 000135, Jose Sierra,
000153, Mario Lopez, 000170, Ana Benitez]
```

Mapas

Un Mapa es un objeto que asocia o mapea llaves a valores. Un mapa no puede tener llaves duplicadas. Cada llave puede mapear a cuando mucho un valor.

En la figura 6.8 se muestran las interfaces y clases que implementan los mapas.

Interfaz `Map<K, V>`

Esta interfaz se encuentre en el tope de la jerarquía de interfaces y las clases que implementan esas interfaces. Declara los métodos comunes a todas las implementaciones de mapas. En la tabla 6.16 se muestran los métodos declarados por la interfaz `Map<K, V>`.

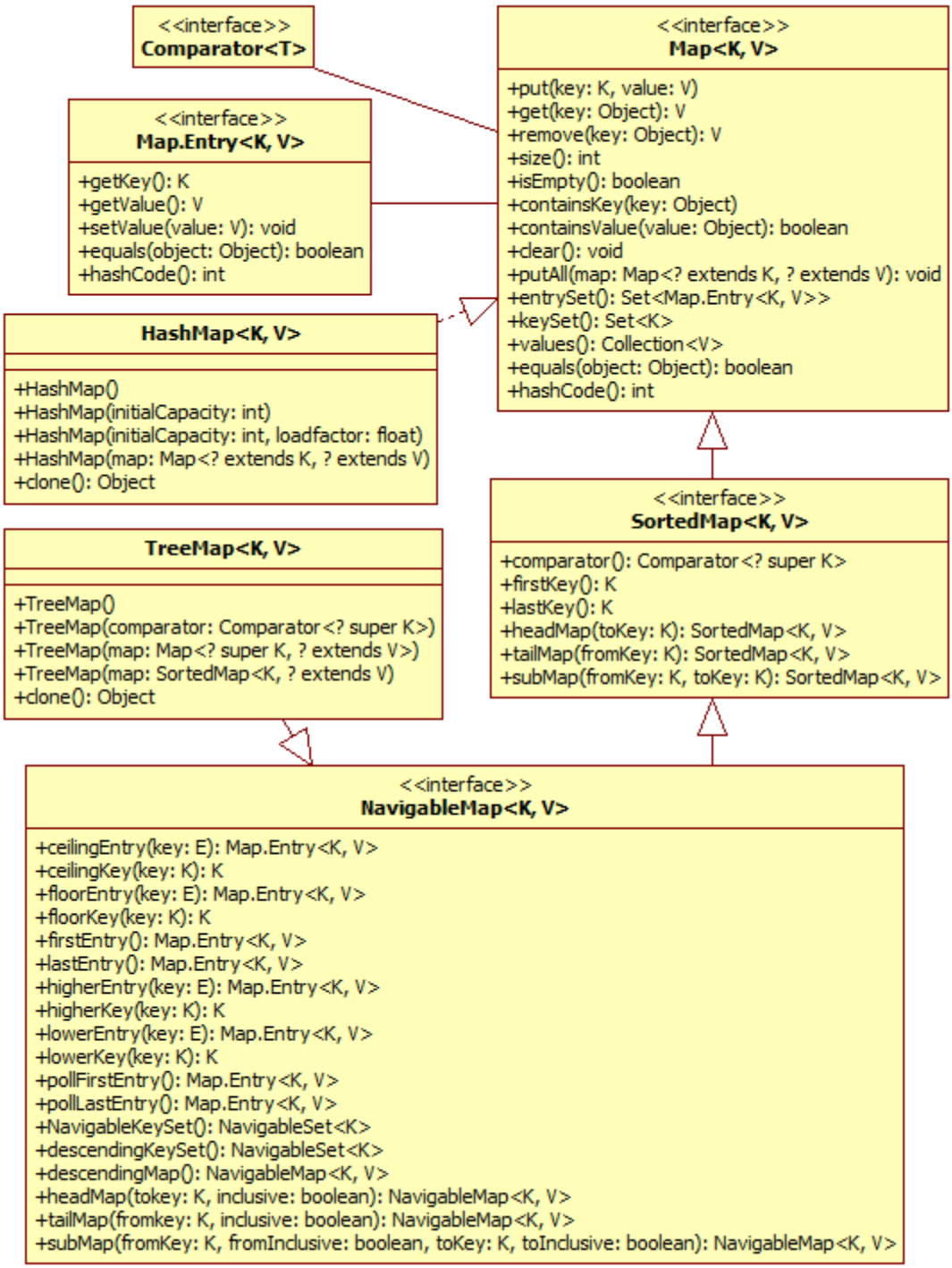
<<interface>>
Comparator<T>
<<interface>>
Map.Entry<K, V>
+getKey(): K
+getValue(): V
+setValue(value: V): void
+equals(object: Object): boolean
+hashCode(): int
<<interface>>
Map<K, V>
+put(key: K, value: V)
+get(key: Object): V
+remove(key: Object): V
+size(): int
+isEmpty(): boolean
+containsKey(key: Object)
+containsValue(value: Object): boolean
+clear(): void
+putAll(map: Map<? extends K, ? extends V): void
+entrySet(): Set<Map.Entry<K, V>>
+keySet(): Set<K>
+values(): Collection<V>
+equals(object: Object): boolean
+hashCode(): int
HashMap<K, V>
+HashMap()
+HashMap(initialCapacity: int)
+HashMap(initialCapacity: int, loadfactor: float)
+HashMap(map: Map<? extends K, ? extends V)
+clone(): Object
<<interface>>
SortedMap<K, V>
+comparator(): Comparator<? super K>
+firstKey(): K
+lastKey(): K
+headMap(toKey: K): SortedMap<K, V>
+tailMap(fromKey: K): SortedMap<K, V>
+subMap(fromKey: K, toKey: K): SortedMap<K, V>
TreeMap<K, V>
+TreeMap()
+TreeMap(comparator: Comparator<? super K>)
+TreeMap(map: Map<? super K, ? extends V>)
+TreeMap(map: SortedMap<K, ? extends V)
+clone(): Object
<<interface>>
NavigableMap<K, V>
+ceilingEntry(key: E): Map.Entry<K, V>
+ceilingKey(key: K): K
+floorEntry(key: E): Map.Entry<K, V>
+floorKey(key: K): K
+firstEntry(): Map.Entry<K, V>
+lastEntry(): Map.Entry<K, V>
+higherEntry(key: E): Map.Entry<K, V>
+higherKey(key: K): K
+lowerEntry(key: E): Map.Entry<K, V>
+lowerKey(key: K): K
+pollFirstEntry(): Map.Entry<K, V>
+pollLastEntry(): Map.Entry<K, V>
+NavigableKeySet(): NavigableSet<K>
+descendingKeySet(): NavigableSet<K>
+descendingMap(): NavigableMap<K, V>
+headMap(tokey: K, inclusive: boolean): NavigableMap<K, V>
+tailMap(fromkey: K, inclusive: boolean): NavigableMap<K, V>
+subMap(fromKey: K, fromInclusive: boolean, toKey: K, toInclusive: boolean): NavigableMap<K, V>

Figura 6.8

Tabla 6.16 Métodos de la Interfaz `Map<K, V>`

`V put(K key, V value)` Asocia el valor del parámetro con la llave del parámetro en este mapa. Si el mapa contenía previamente un mapeo para la llave, el valor que tenía asociado se reemplaza por el valor del parámetro. Regresa el valor previamente asociado con la llave del parámetro o `null` si no había un mapeo para la llave o si el valor asociado con la llave era null y el mapa acepta valores nulos. **Lanza**: `UnsupportedOperationException` - Si el mapa no soporta la operación `put()`. `ClassCastException` – Si la clase de la llave o del valor no permiten que sean almacenadas en este mapa. `NullPointerException` – Si la llave del parámetro o su valor son `null`. Y este mapa no soporta llaves o valores nulos. `IllegalArgumentException` –Si alguna propiedad de la llave o valor del parámetro les impide almacenarlos en el mapa.
`V get(Object key)` Regresa el valor mapeado con la llave del parámetro, o `null` si no había un mapeo para la llave. Si el mapa permite valores null, entonces si el método regresa null no necesariamente indica que el mapa no contiene un mapeo para la llave. **Lanza**: `ClassCastException` – Si la clase de la llave es de un tipo no apropiado para este mapa. `NullPointerException` – Si la llave del parámetro es `null` y este mapa no soporta llaves nulas.
`V` **`remove`**`(Object key)` Elimina el mapeo para la llave del parámetro, si existe. Regresa el valor previamente mapeado por la llave del parámetro, o `null` si no había un mapeo para la llave. **Lanza**: `UnsupportedOperationException` - Si el mapa no soporta la operación `remove()`. `ClassCastException` – Si la clase de la llave es de un tipo no apropiado para este mapa. `NullPointerException` – Si la llave del parámetro es null y este mapa no soporta llaves nulas.
`int` **`size`**`()` Regresa el número de mapeos llave – valor en este mapa. Si el mapa contiene más de de `Integer.MAX_VALUE` elementos, regresa `Integer.MAX_VALUE`. Regresa el número de mapeos llave – valor en este mapa.
`boolean` **`isEmpty`**`()` Regresa verdadero si este mapa no contiene mapeos llave – valor.
`boolean` **`containsKey`**`(Object key)` Regresa verdadero si este mapa no contiene mapeos llave – valor. **Lanza**: `ClassCastException` – Si la clase de la llave es de un tipo no apropiado para este mapa. `NullPointerException` – Si la llave del parámetro es `null` y este mapa no soporta llaves nulas.

Tabla 6.16 Métodos de la Interfaz `Map<K, V>`. Cont

`boolean` **`containsValue`**`(Object value)` Regresa verdadero si este mapa mapea el valor del parámetro a una o más llaves. **Lanza**: `ClassCastException` – Si la clase del valor es de un tipo no apropiado para este mapa. `NullPointerException` – Si el valor del parámetro es `null` y este mapa no soporta valores nulos.
`void` **`clear`**`()` Elimina todos los mapeos para este mapa. El mapa quedará vacío después de la operación. **Lanza**: `UnsupportedOperationException` - Si el mapa no soporta la operación `clear()`.
`void` **`putAll`**`(Map<? extends K,? extends V> map)` Copia todos los mapeos del mapa del parámetro a este mapa. **Lanza**: `UnsupportedOperationException` - Si el mapa no soporta la operación `putAll()`. `ClassCastException` – Si la clase de la llave o del valor son de un tipo no apropiado para este mapa. `NullPointerException` – Si el mapa del parámetro es null, o si este mapa no permite llaves o valores nulos y el mapa del parámetro contiene llaves o valores nulos. `IllegalArgumentException` –Si alguna propiedad de la llave o valor en el mapa del parámetro les impide almacenarlos en este mapa.
`Set<Map.Entry<K,V>>` **`entrySet`**`()` Regresa una vista como conjunto del contenido de este mapa. El conjunto está respaldado por el mapa, de tal manera que los cambios en el mapa se reflejan en el conjunto y viceversa.
`Set<K>` **`keySet`**`()` Regresa una vista como conjunto de las llaves contenidas en este mapa. El conjunto está respaldado por el mapa, de tal manera que los cambios en el mapa se reflejan en el conjunto y viceversa.
`Collection<V>` **`values`**`()` Regresa una vista como coleccion de los valores contenidos de este mapa. El conjunto está respaldado por el mapa, de tal manera que los cambios en el mapa se reflejan en el conjunto y viceversa.
`int` **`hashCode`**`()` Regresa el código Hash para este mapa.

Interfaz `Map.Entry<K, V>`

Esta interfaz representa una entrada de un mapa (un par llave – valor). El método `entryset()` de la interfaz `Map<K, V)` regresa una vista de colección del mapa, cuyos elementos son de esta clase. La única forma de obtener una referencia a una entrada de un mapa es del iterador de esa vista de colección y esa entrada sólo es

válida durante la iteración. En la tabla 6.17 se muestran los métodos declarados por la interfaz `Map.Entry<K, E>`.

Tabla 6.17 Métodos de la Interfaz `Map.Entry<K, V>`

`K` **`getKey`**`()` Regresa la llave correspondiente a esta entrada. **Lanza**: `IllegalStateException` –Si esta entrada ha sido eliminada del mapa correspondiente.
`V` **`getValue`**`()` Regresa el valor correspondiente a esta entrada. **Lanza**: `IllegalStateException` –Si esta entrada ha sido eliminada del mapa correspondiente.
`V` **`setValue`**`(V value)` Reemplaza el valor corresponiente de esta entrada por el valor del parámetro. **Lanza**: `UnsupportedOperationException` - Si el mapa asociado no soporta la operación `put()`. `ClassCastException` – Si la clase del valor es de un tipo no apropiado para el mapa asociado. `NullPointerException` – Si el valor del parámetro es null y el mapa asociado no permite valores nulos. `IllegalArgumentException` –Si alguna propiedad del valor del parámetro le impide almacenarlo al mapa asociado. `IllegalStateException` –Si esta entrada ha sido eliminada del mapa correspondiente.
`boolean` **`equals`**`(Object o)` Compara el objeto del parámetro con esta entrada por igualdad. Regresa verdadero si el objeto del parámetro es una entrada de un mapa y las dos entradas representan el mismo mapeo. Regresa `true` si el objeto del parámetro y este mapa son iguales, `false` en caso contrario.
`int` **`hashCode`**`()` Regresa el código Hash para esta entrada de mapa.

Interfaz `SortedMap<K, V>`

La interfaz `SortedMap<K, V>` garantiza un ordenamiento total en sus llaves. El mapa está ordenado de acuerdo al ordenamiento natual de sus llaves, o por un comparador establecido por su constructor.

Todas las llaves que se inserten en un mapa ordenado deben implementar la interfaz `Comparable` (o ser aceptadas por comparador especificado). En la tabla 6.18 se muestran los métodos declarados por la interfaz `Sorted Map<K, V>`.

Tabla 6.18 Métodos de la Interfaz `SortedMap<K, V>`

`Comparator<? super E>` **`comparator`**`()` Regresa el comparador usado para ordenar las llaves de este mapa ordenado, o null si el mapa usa el ordenamiento natural para sus llaves.
`K` **`firstKey`**`()` Regresa la primera llave de este mapa ordenado. **Lanza**: `NoSuchElementException` – Si el mapa ordenado está vacío.
`K` **`lastKey`**`()` Regresa la última llave de este mapa ordenado. **Lanza**: `NoSuchElementException` – Si el mapa ordenado está vacío.
`SortedMap<K,V>` **`headMap`**`(K toKey)` Regresa una vista de este mapa cuyas llaves sean estrictamente menores que la llave `toKey.` El mapa regresado está respaldado por este mapa, de tal manera que los cambios en el mapa regresadose reflejan en este mapa y viceversa. **Lanza**: `ClassCastException` – Si la llave `toKey` no es compatible con el comparador del mapa (o si el mapa no tiene un comparador o la llave `toKey` no implementa la interfaz `Comparable`). `NullPointerException` – Si la llave `toKey` es null y el mapa asociado no permite llaves nulas. `IllegalArgumentException` – Si el mapa tiene una restricción de rango y la llave toKey cae fuera de los límites del rango.
`SortedMap<K,V>` **`tailMap`**`(K fromKey)` Regresa una vista de este mapa cuyas llaves sean mayores o iguales que la llave `toKey`. El mapa regresado está respaldado por este mapa, de tal manera que los cambios en el mapa regresadose reflejan en este mapa y viceversa. **Lanza**: `ClassCastException` – Si la llave `toKey` no es compatible con el comparador del mapa (o si el mapa no tiene un comparador o la llave `toKey` no implementa la interfaz `Comparable`). `NullPointerException` – Si la llave `toKey` es null y el mapa asociado no permite llaves nulas. `IllegalArgumentException` – Si el mapa tiene una restricción de rango y la llave `toKey` cae fuera de los límites del rango.

Tabla 6.18 Métodos de la Interfaz `SortedMap<K, V>`. Cont.

`SortedMap<K,V>` **`subMap`**`(K fromKey,K toKey)` Regresa una vista de este mapa cuyas llaves estén en el rango de la llave `fromKey`, inclusive a la llave `toKey`, `exclusive`. El mapa regresado está respaldado por este mapa, de tal manera que los cambios en el mapa regresadose reflejan en este mapa y viceversa. **Lanza**: `ClassCastException` – Si las llaves `fromKey` y `toKey` no son compatibles con el comparador del mapa (o si el mapa no tiene un comparador, no pueden compara usando el orden natural). `NullPointerException` – Si las llaves `fromKey` y `toKey` son null y el mapa asociado no permite llaves nulas. `IllegalArgumentException` –Si la llave `fromKey` es mayor que `toKey;` o el mapa tiene una restricción de rango y las llaves `fromKey` y `toKey` caen fuera de los límites del rango.

Interfaz `NavigableMap<K, V>`

La interfaz `NavigableMap<K, V>` extiende la interfaz `SortedMap<K, V>` con métodos de navegación que regresan las coincidencias más cercanas para una búsqueda dada. Un mapa navegable puede accederse ybrecorrerse en orden ascendente o descendente.

En la tabla 6.19 se muestran los métodos declarados por la interfaz `NavigableMap<K, V>`.

Tabla 6.19 Métodos de la Interfaz `NavigableMap<K, V>`

`Map.Entry<K, V>` **`ceilingEntry`**`(K key)` Regresa una entrada llave – valor asociada con la llave menor que sea mayor o igual que la llave del parámetro, o null si no existe tal llave. **Lanza**: `ClassCastException` – Si la llave del parámetro no puede compararse con las llaves del mapa. `NullPointerException` – Si la llave del parámetro es nula y el mapa no admite llaves nulas.
`K` **`ceilingKey`**`(K key)` Regresa la llave menor que sea mayor o igual que la llave del parámetro, o null si no existe tal llave. **Lanza**: `ClassCastException` – Si la llave del parámetro no puede compararse con las llaves del mapa. `NullPointerException` – Si la llave del parámetro es nula y el mapa no admite llaves nulas.

Tabla 6.19 Métodos de la Interfaz `NavigableMap<K, V>`. Cont.

`Map.Entry<K, V>` **`floorEntry`**`(K key)` Regresa una entrada llave – valor asociada con la llave mayor que sea menor o igual que la llave del parámetro, o null si no existe tal llave. **Lanza**: `ClassCastException` – Si la llave del parámetro no puede compararse con las llaves del mapa. `NullPointerException` – Si la llave del parámetro es nula y el mapa no admite llaves nulas.
`K` **`floorKey`**`(K key)` Regresa la llave mayor que sea menor o igual que la llave del parámetro, o null si no existe tal llave. **Lanza**: `ClassCastException` – Si la llave del parámetro no puede compararse con las llaves del mapa. `NullPointerException` – Si la llave del parámetro es nula y el mapa no admite llaves nulas.
`Map.Entry<K, V>` **`firstEntry`**`()` Regresa una entrada llave – valor asociada con la llave menor del mapa, o null si el mapa esta vacío.
`Map.Entry<K, V>` **`lastEntry`**`()` Regresa una entrada llave – valor asociada con la llave mayor del mapa, o null si el mapa esta vacío.
`Map.Entry<K, V>` **`higherEntry`**`(K key)` Regresa una entrada llave – valor asociada con la llave menor que sea mayor que la llave del parámetro, o null si no existe tal llave. **Lanza**: `ClassCastException` – Si la llave del parámetro no puede compararse con las llaves del mapa. `NullPointerException` – Si la llave del parámetro es nula y el mapa no admite llaves nulas.
`K` **`higherKey`**`(K key)` Regresa la llave menor que sea mayor que la llave del parámetro, o null si no existe tal llave. **Lanza**: `ClassCastException` – Si la llave del parámetro no puede compararse con las llaves del mapa. `NullPointerException` – Si la llave del parámetro es nula y el mapa no admite llaves nulas.
`Map.Entry<K, V>` **`lowerEntry`**`(K key)` Regresa una entrada llave – valor asociada con la llave menor que sea mayor que la llave del parámetro, o null si no existe tal llave. **Lanza**: `ClassCastException` – Si la llave del parámetro no puede compararse con las llaves del mapa. `NullPointerException` – Si la llave del parámetro es nula y el mapa no admite llaves nulas.

Tabla 6.19 Métodos de la Interfaz `NavigableMap<K, V>`. Cont.

`K` **`lowerKey`**`(K key)` Regresa la llave mayor que sea menor que la llave del parámetro, o null si no existe tal llave. **Lanza**: `ClassCastException` – Si la llave del parámetro no puede compararse con las llaves del mapa. `NullPointerException` – Si la llave del parámetro es nula y el mapa no admite llaves nulas.
`Map.Entry<K, V>` **`pollFirst`** `Entry()` Elimina y regresa la entrada llave – valor asociada con la primer (la menor) llave de este mapa, o null si el mapa está vacío. Regresa la entrada llave – valor asociada con la primer (la menor) llave de este mapa, o null si el mapa está vacío.
`Map.Entry<K, V>` **`pollLast`** `Entry()` Elimina y regresa la entrada llave – valor asociada con la última (la mayor) llave de este mapa, o null si el mapa está vacío. Regresa la entrada llave – valor asociada con la última (la mayor) llave de este mapa, o null si el mapa está vacío.
`NavigableSet<K>` **`NavigableKeySet`**`()` Regresa una vista del tipo NavigableSet en orden ascendente de las llaves de este mapa. El conjunto navegable regresado está respaldado por este mapa. Los cambios en el conjunto se reflejan en el mapa regresado y viceversa.
`NavigableSet<K>` **`descendingKeySet`**`()` Regresa una vista del tipo NavigableSet en orden invertido de las llaves de este mapa. El conjunto navegable regresado está respaldado por este mapa. Los cambios en el conjunto se reflejan en el mapa regresado y viceversa.
`NavigableMap<K, V>` **`descendingMap`**`()` Regresa una vista en orden invertido de las entradas de este mapa. El mapa descendente está respaldado por este mapa. Los cambios en el mapa se reflejan en el mapa descendente y viceversa.
`NavigableMap<K, V>` **`headMap`**`(K toKey, boolean inclusive)` Regresa una vista de la porción de este mapa cuyas llaves son menores (o iguales si el parámetro `inclusive` es verdadero) que la llave del parámetro. El mapa regresado está respaldado por este mapa. Los cambios en el mapa se reflejan en el mapa regresado y viceversa. **Lanza**: `ClassCastException` – Si la llave `toKey` no es compatible con el comparador de este mapa (o si el mapa no tiene comparador, si la llave `toKey` no implementa la interfaz comparable). `NullPointerException` – Si la llave `toKey` es nula y el mapa no admite llaves nulas. `IllegalArgumentException` – Si este mapa tiene un rango restringido y la llave `toKey` cae fuero del rengo del parámetro.

Tabla 6.19 Métodos de la Interfaz `NavigableMap<K, V>`. Cont.

`NavigableMap<K, V>` **`tailMap`**`(K fromKey, boolean inclusive)` Regresa una vista de la porción de este mapa cuyas llaves son mayores (o iguales si el parámetro `inclusive` es verdadero) que la llave del parámetro. El mapa regresado está respaldado por este mapa. Los cambios en el mapa se reflejan en el mapa regresado y viceversa. **Lanza**: `ClassCastException` – Si la llave `fromKey` no es compatible con el comparador de este mapa (o si el mapa no tiene comparador, si la llave `fromKey` no implementa la interfaz comparable). `NullPointerException` – Si la llave `fromKey` es nula y el mapa no admite llaves nulas. `IllegalArgumentException` – Si este mapa tiene un rango restringido y la llave `fromKey` cae fuero del rengo del parámetro.

`NavigableMap<E>` **`subMap`**`(K fromKey, boolean fromInclusive,`
`K toKey, boolean toInclusive)`
 	
Regresa una vista de la porción de este mapa cuyas llaves están en el rango de `fromKey` a `toKey`, si `fromKey` y `toKey` son iguales, el mapa regresado es vacío a menos que `fromInclusive` y `toInclusive` sean ambas verdaderas. El mapa regresado está respaldado por este mapa. Los cambios en el mapa se reflejan en el mapa regresado y viceversa.
 	
Lanza:
`ClassCastException` – Si las llaves `fromKey` y `toKey` no pueden compararse entre sí usando el comparador del mapa (o si el mapa no tiene comparador o, usando el ordenamiento natural).
`NullPointerException` – Si `fromKey` o `toKey` son nulas y el mapa no admite llaves nulas.
`IllegalArgumentException` – Si `fromKey` es mayor que `toKey`, o si este mapa tiene un rango restringido y `fromKey` o `toKey` caen fuera del rengo del parámetro.

Clase `HashMap<K, V>`

Esta clase implementa la interfaz `Map`, sobre una Tabla Hash. No hay garantías sobre el orden de iteración del conjunto; en particular no hay garantía que el orden permanecera constante con el tiempo. Esta clase permite llaves y valores nulos.

En la tabla 6.20 se muestran los métodos exclusivos de la clase `HashSet<E>`.

Tabla 6.20 Métodos exclusivos de la Clase `HashMap<E>`

`HashMap``()` Construye un mapa vacío con una capacidad inicial de 16 y un factor de carga de 0.75.
`HashMap``(Map<? extends K,? extends V> map)` Construye un mapa que contiene los mismos mapeos que el mapa del parámetro. El factor de carga es de 0.75. El mapa tiene una capacidad inicial a suficiente para contener los mapeos del mapa del parámetro. Lanza: `NullPointerException` – Si el mapa del parámetro es nulo.

Tabla 6.20 Métodos exclusivos de la Clase `HashMap<E>`

`HashSet``(int` *`inicialCapacity`*`, float loadfactor)` Construye un mapa vacío con la capacidad inicial y factor de carga dados por los parámetros. Lanza: `IllegalArgumentException` – Si el valor de la capacidad inicial es negativo o el factor de carga sea no positivo.
`HashSet``(int` *`inicialCapacity`*`)` Construye un mapa vacío con la capacidad inicial dada por el parámetro y un factor de carga de 0.75. Lanza: `IllegalArgumentException` – Si el valor de la capacidad inicial es negativo.
`Object clone()` Regresa una copia del conjunto. Las llaves y valores del mapa no son copiados.

Clase `TreeMap<K, V>`

Esta clase implementa la interfaz `NavigableMap`, sobre un árbol Rojo - Negro. Esta clase garantiza que el mapa estará ordenado en orden ascendente, de acuerdo al orden natural de sus elementos o por un comparador establecido en el constructor. En la tabla 6.21 se muestran los métodos exclusivos de la interfaz `TreeMap<K, V>`.

Tabla 6.21 Métodos exclusivos de la Clase `TreeMap<K, V>`

`TreeMap``()` Construye un nuevo mapa vacío ordenado de acuerdo al orden natural de sus llaves.
`TreeMap``(Map<? extends K,? extends V> m)` Construye un nuevo mapa con los mapeos del mapa del parámetro, ordenados de acuerdo al orden natural de sus llaves. Lanza: `ClassCastException` – Si las llaves del mapa del parámetro no son comparables o no son mutuamente comparables. `NullPointerException` – Si el mapa del parámetro es nula.
`TreeMap``(Comparator<? super K> comparator)` Construye un nuevo mapa vacío, ordenado de acuerdo al comparador del parámetro.
`TreeSet``(SortedMap<K,? extends V> map)` Construye un nuevo mapa con los mapeos del mapa ordenado del parámetro, ordenados de acuerdo al mismo orden. Lanza: `NullPointerException` – Si conjunto ordenado del parámetro es nulo.
`Object clone()` Regresa una copia del conjunto. Las llaves y valores del mapa no son copiados.

Ejemplos Sobre Mapas

El siguiente programa cuenta el número de veces que cada palabra aparece en una lista de cadenas.

```
/*
 * CuentaPalabras.java
 *
 * @author mdomitsu
 */

package objetosNegocio;

import java.util.List;
import java.util.Map;
import java.util.TreeMap;

/**
 * Esta clase contabiliza las palabras diferentes del texto contenido
 * en una lista de cadenas
 */
public class CuentaPalabras {
    Map<String, Integer> mapaPalabras;
    List<String> texto;

    /**
     * Inicializa la clase con el texto contenido en una lista de cadenas
     * @param texto Lista de cadenas con el texto a procesar
     */
    public CuentaPalabras(List<String> texto) {
        this.texto = texto;
        mapaPalabras = new TreeMap();
    }

    /**
     * Regresa las palabras contenidas en el texto y su cuenta
     * @return Las palabras contenidas en el texto y su cuenta
     */
    public Map<String, Integer> getMapaPalabras() {
        return mapaPalabras;
    }

    /**
     * Este método procesa las cadenas de la lista
     */
    public void procesaTexto() {
        for(String linea: texto) {
            procesaLinea(linea);
        }
    }

    /**
     * Este metodo procesa una cadena encontrando la cuenta de cada palabra
     * diferente
     * @param linea Cadena a procesar
```

```
     */
    public void procesaLinea(String linea) {
        int pos1 = 0;
        int pos2;
        boolean continua = true;
        String palabra;

        // Mientras no se haya llegado al final de la cadena
        while(continua) {
            // Determina donde termina la palabra
            pos2 = linea.indexOf(' ', pos1);

            // Obtiene la palabra
            if(pos2 < 0) {
                palabra = linea.substring(pos1);
                continua = false;
            }
            else palabra = linea.substring(pos1, pos2);

            // Si la palabra es nueva
            if(!mapaPalabras.containsKey(palabra)) {
                mapaPalabras.put(palabra, 1);
            }
            // Si la palabra esta repetida
            else {
                int n = mapaPalabras.get(palabra);
                mapaPalabras.put(palabra, n+1);
            }

            pos1 = pos2+1;
        }
    }
}
```

Para probar los métos de la clase `CuentaPalabras` se tiene la siguiente clase de prueba:

```
/*
 * Prueba.java
 */

package pruebas;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import objetosNegocio.CuentaPalabras;

/**
 *
 * @author mdomitsu
 */
public class Prueba {

    /**
     * @param args the command line arguments
```

```
     */
    public static void main(String[] args) {
        List<String> texto = new ArrayList<String>();

        texto.add("el caballo corre por el campo");
        texto.add("mi mama me mima");

        CuentaPalabras cuentaPalabras = new CuentaPalabras(texto);

        cuentaPalabras.procesaTexto();

        Map<String, Integer> mapaPalabras = cuentaPalabras.getMapaPalabras();

        System.out.println("Palabras encontradas: ");
        System.out.println(mapaPalabras);
    }
}
```

La corrida del programa genera la siguiente salida:

```
Palabras encontradas:
{caballo=1, campo=1, corre=1, el=2, mama=1, me=1, mi=1, mima=1, por=1}
```

Clase `Collections`

Esta clase contiene sólo métodos estáticos que operan sobre o regresan collecciones. Contienen algoritmos polimórficos que operan sobre colecciones, clases que encapsulan colecciones, que regresan una colección nueva basada en una colección dada y otros métodos adicionales. Todos los métodos de esta clase lanzan una excepción del tipo `NullPointerException` si la colección o clase de sus parámetros son nulas. Algunos de los métodos de esta clase semuestran en el diagrama de clases de la figura 6.9 y su descripción se encuentra en la tabla 6.15.

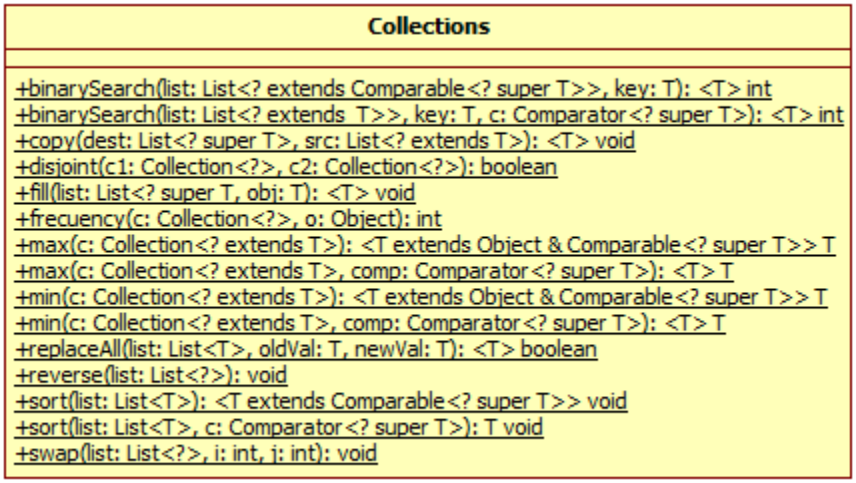

Figura 6.9

Tabla 6.22 Métodos de la Clase `Collections`

`public static <T> int`
`  `**`binarySearch`**`(List<? extends Comparable<? super T>> list, T key)`
`public static <T> int`
`  `**`binarySearch`**`(List<? extends T> list, T key, Comparator<? super T> c)`
 	
Busca en la lista del parámetro `list` al elemento del parámetro `key` usando el algoritmo de búsqueda binaria. La lista debe estar previamente ordenada en orden ascendente de acuerdo con el ordenamiento natural de sus elementos, en el primer método o de acuerdo al comparador del parámetro c del segundo método. Si la lista no está ordenada, los resultados no están definidos. Si la lista contiene elementos repetidos iguales al elemento del parámetro, no hay garantía de cual será encontrado.
 	
Regresan el índice del elemento buscado, si existe en la lista; de otro modo el valor (`–(punto de inserción) -1`). El punto de inserción se define como el punto en el cual el elemento buscado se insertaría en la lista.
 	
Lanza:
`ClassCastException` - Si la lista contiene elementos que no son mutuamente comparables (por ejemplo, cadenas con enteros), o el elemento buscado no es mutaumente comparable con los elementos de la lista.

`public static <T> void `**`copy`**`(List<? super T> dest, List<? extends T> src)` Copia todos los elementos de la lista dada por el parámetro `src` a la lista dada por el parámetro `dest`. La lista destino debe ser al menos tan grande como la lista origen. Si es mayor el resto de los elementos no seven afectados. **Lanza**: `IndexOutOfBoundsException –` Si la lista destino no es lo suficientemente grande para contener la lista origen. `UnsupportedOperationException –` Si el iterador de lista no soporta la operación `set()`.
`public static boolean `**`disjoint`**`(Collection<?> c1, Collection<?> c2)` Regresa verdadero si si las dos colecciones de sus parámetros c1 y c2 no tienen elementos en común.
`public static <T> void `**`fill`**`(List<? super T> list, T obj)` Reemplaza todos los elementos de la lista del parámetro `list` con el valor del parámetro `obj`. **Lanza**: `UnsupportedOperationException –` Si el iterador de lista no soporta la operación `set()`.
`public static int `**`frequency`**`(Collection<?> c, Object o)` Regresa el número de elementos de la colección del parámetro c que son iguales al objeto del parámetro o.

Tabla 6.22 Métodos de la Clase `Collections`. Cont.

`public static <T extends Object & Comparable<? super T>> T`
`  **max**(Collection<? extends T> coll)`
`public static <T> T **max**(Collection<? extends T> coll,`
`  Comparator<? super T> comp)`
 	
Regresan el elemento máximo de la colección dada por el parámetro `coll`, de acuerdo con el ordenamiento natural de sus elementos, en el primer método o de acuerdo al comparador del parámetro `comp` del segundo método. En el primer método método, todos los elementos de lista deben implementar la interfaz `Comparable` y ser mutuamente comparables. En el segundo método todos los elementos de la lista deben ser mutuamente comparables usando el comparador del parámetro `comp`.
 	
Lanza:
`ClassCastException` - Si la lista contiene elementos que no son mutuamente comparables.
`NoSuchElementException` – Si la colección está vacía.

`public static <T extends Object & Comparable<? super T>> T`
`  **min**(Collection<? extends T> coll)`
`public static <T> T **min**(Collection<? extends T> coll,`
`  Comparator<? super T> comp)`
 	
Regresan el elemento minimo de la colección dada por el parámetro `coll`, de acuerdo con el ordenamiento natural de sus elementos, en el primer método o de acuerdo al comparador del parámetro `comp` del segundo método. En el primer método método, todos los elementos de lista deben implementar la interfaz `Comparable` y ser mutuamente comparables. En el segundo método todos los elementos de la lista deben ser mutuamente comparables usando el comparador del parámetro `comp`.
 	
Lanza:
`ClassCastException` - Si la lista contiene elementos que no son mutuamente comparables.
`NoSuchElementException` – Si la colección está vacía.

`public static <T> boolean **replaceAll**(List<T> list, T oldVal, T newVal)` Reemplaza todas las ocurrencias del elemento dado por el parámetro `oldVal` por el elemento dado por el parámetro `newVal` de la lista del parámetro `list`. **Regresa**: `true` si se hizo algún reemplazo, `false` en caso contrario. **Lanza**: `UnsupportedOperationException` – Si el iterador de lista no soporta la operación `set()`.
`public static void **reverse**(List<?> list)` Invierte el orden delos elementos de la lista. **Lanza**: `UnsupportedOperationException` – Si el iterador de lista no soporta la operación `set()`.

Tabla 6.22 Métodos de la Clase `Collections`. Cont.

`public static <T extends Comparable<? super T>> void` **`sort`**`(List<T> list)`
`public static <T> void` **`sort`**`(List<T> list, Comparator<? super T> c)`
 	
Ordena la lista del parámetro `list` en orden ascendente, de acuerdo con el ordenamiento natural de sus elementos, en el primer método o de acuerdo al comparador del parámetro `c` del segundo método. En el primer método método, todos los elementos de lista deben implementar la interfaz `Comparable` y ser mutuamente comparables. En el segundo método todos los elementos de la lista deben ser mutuamente comparables usando el comparador del parámetro `c`.
 	
Lanza:
`ClassCastException` - Si la lista contiene elementos que no son mutuamente comparables (por ejemplo, cadenas con enteros), o el elemento buscado no es mutaumente comparable con los elementos de la lista.
`UnsupportedOperationException` – Si el iterador de lista no soporta la operación `set()`.

`public static void` **`swap`**`(List<?> list, int i, int j)` Intercambia los elementos cuyas posiciones están dadas por los parámetros `i` y `j` de la lista dada por el parámetro `list`. **Lanza**: `IndexOutOfBoundsException` – Si alguno de los parámetros `i` o `j` están fuera de rango.

Ejemplos Sobre la Clase Collections

En el siguiente ejemplo se ilustrará el uso de de los métodos `sort()` y `reverse()` para ordenar una lista de canciones por algunos de sus atributos en orden ascendente y descendente. Para poder ordenar las canciones de la lista se requiere poder comparar las canciones de acuerdo a un atributo. Para ello crearemos una serie de clases que implementen la interfaz `Comparator`, una para cada uno de los atributos por la que deseemos ordenar la lista de canciones. Algunas de esas clases comparan instancias de la clase `Medio` que es la clase padre de las clases `Cancion` y `Pelicula` por lo que pueden usarse para ordenar listas de canciones y de películas.

ComparaMediosClave.java

```
/*
 * ComparaMediosClave.java
 */

package comparadores;

import java.util.Comparator;
import objetosNegocio.Medio;

/**
 * Esta clase compara dos medios por su clave
 * @author mdomitsu
 */
public class ComparaMediosClave implements Comparator<Medio> {
    /**
     * Este método compara dos medios por su clave
     * @param m1 Medio 1 a comparar
```

```
     * @param m2 Medio 2 a comparar
     * @return Regresa un entero negativo, cero, o un entero positivo
     * si el medio m1 es menor que, igual a, o mayor que el medio m2
     */
  public int compare(Medio m1, Medio m2) {
      return m1.getClave().compareTo(m2.getClave());
  }
}
```

ComparaMediosTitulo.java

```
/*
 * ComparaMediosTitulo.java
 */

package comparadores;

import java.util.Comparator;
import objetosNegocio.Medio;

/**
 * Esta clase compara dos medios por su titulo
 * @author mdomitsu
 */
public class ComparaMediosTitulo implements Comparator<Medio> {
    /**
     * Este método compara dos medios por su titulo
     * @param m1 Medio 1 a comparar
     * @param m2 Medio 2 a comparar
     * @return Regresa un entero negativo, cero, o un entero positivo
     * si el medio m1 es menor que, igual a, o mayor que el medio m2
     */
  public int compare(Medio m1, Medio m2) {
      return m1.getTitulo().compareTo(m2.getTitulo());
  }
}
```

ComparaMediosGenero.java

```
/*
 * ComparaMediosGenero.java
 */

package comparadores;

import java.util.Comparator;
import objetosNegocio.Medio;

/**
 * Esta clase compara dos medios por su genero
 * @author mdomitsu
 */
public class ComparaMediosGenero implements Comparator<Medio> {
    /**
     * Este método compara dos medios por su genero
     * @param m1 Medio 1 a comparar
```

```
     * @param m2 Medio 2 a comparar
     * @return Regresa un entero negativo, cero, o un entero positivo
     * si el medio m1 es menor que, igual a, o mayor que el medio m2
     */
  public int compare(Medio m1, Medio m2) {
      return
m1.getGenero().getCveGenero().compareTo(m2.getGenero().getCveGenero());
  }
}
```

ComparaMediosFecha.java

```
/*
 * ComparaMediosFecha.java
 */

package comparadores;

import java.util.Comparator;
import objetosNegocio.Medio;

/**
 * Esta clase compara dos medios por su fecha
 * @author mdomitsu
 */
public class ComparaMediosFecha implements Comparator<Medio> {
    /**
     * Este método compara dos medios por su fecha
     * @param m1 Medio 1 a comparar
     * @param m2 Medio 2 a comparar
     * @return Regresa un entero negativo, cero, o un entero positivo
     * si el medio m1 es menor que, igual a, o mayor que el medio m2
     */
  public int compare(Medio m1, Medio m2) {
      return m1.getFecha().compareTo(m2.getFecha());
  }
}
```

ComparaCancionesInterprete.java

```
/*
 * ComparaCancionesInterprete.java
 */

package comparadores;

import java.util.Comparator;
import objetosNegocio.Cancion;

/**
 * Esta clase compara dos canciones por su interprete
 * @author mdomitsu
 */
public class ComparaCancionesInterprete implements Comparator<Cancion> {
    /**
     * Este método compara dos canciones por su interprete
     * @param c1 Cancion 1 a comparar
     * @param c2 Cancion 2 a comparar
```

```
     * @return Regresa un entero negativo, cero, o un entero positivo
     * si la cancion c1 es menor que, igual a, o mayor que la canción c2
     */
  public int compare(Cancion c1, Cancion c2) {
      return c1.getInterprete().compareTo(c2.getInterprete());
  }
}
```

ComparaCancionesAlbum.java

```
/*
 * ComparaCancionesAlbum.java
 */

package comparadores;

import java.util.Comparator;
import objetosNegocio.Cancion;

/**
 * Esta clase compara dos canciones por su album
 * @author mdomitsu
 */
public class ComparaCancionesAlbum implements Comparator<Cancion> {
    /**
     * Este método compara dos canciones por su album
     * @param c1 Cancion 1 a comparar
     * @param c2 Cancion 2 a comparar
     * @return Regresa un entero negativo, cero, o un entero positivo
     * si la cancion c1 es menor que, igual a, o mayor que la canción c2
     */
  public int compare(Cancion c1, Cancion c2) {
      return c1.getAlbum().compareTo(c2.getAlbum());
  }
}
```

En el siguiente fragmento de código muestra como ordenar la lista de canciones por sus diferentes atributos usando los comparadores anteriores.

Prueba4.java

```
/*
 * Prueba.java
 */
package pruebas;

import comparadores.ComparaCancionesAlbum;
import comparadores.ComparaCancionesInterprete;
import comparadores.ComparaMediosClave;
import comparadores.ComparaMediosFecha;
import comparadores.ComparaMediosGenero;
import comparadores.ComparaMediosTitulo;
import excepciones.PersistenciaException;
import interfaces.IPersistencia;
import java.util.Collections;
import java.util.List;
import objetosNegocio.Cancion;
```

```
import objetosNegocio.Genero;
import objetosServicio.Fecha;
import persistencia.PersistenciaListas;

/**
 *
 * @author mdomitsu
 */
public class Prueba4 {

  /**
   * @param args the command line arguments
   */
  public static void main(String[] args) {
    Prueba4 prueba4 = new Prueba4();

    // Crean la fachada de los objetos que permiten almacenar las
    // canciones y peliculas en arreglos
    IPersistencia fachada = new PersistenciaListas();
    List<Cancion> listaCanciones = null;
    ...

    try {
      // Obtiene la lista de canciones de la BD
      listaCanciones = fachada.consultaCanciones();

      // Despliega la lista de canciones
      System.out.println("Lista de canciones:");
      System.out.println(listaCanciones);

      // Ordena la lista de canciones por su clave
      Collections.sort(listaCanciones, new ComparaMediosClave());

      // Despliega la lista de canciones ordenada por su clave
      System.out.println("Lista de canciones ordenadas por clave:");
      System.out.println(listaCanciones);

      // Ordena la lista de canciones por su titulo
      Collections.sort(listaCanciones, new ComparaMediosTitulo());

      // Despliega la lista de canciones ordenada por su titulo
      System.out.println("Lista de canciones ordenadas por titulo:");
      System.out.println(listaCanciones);

      // Ordena la lista de canciones por su titulo en orden inverso
      Collections.reverse(listaCanciones);

      // Despliega la lista de canciones ordenada por su titulo en orden
      // inverso
      System.out.println(
        "Lista de canciones ordenadas por titulo en orden inverso:");
      System.out.println(listaCanciones);

      // Ordena la lista de canciones por su genero
      Collections.sort(listaCanciones, new ComparaMediosGenero());

      // Despliega la lista de canciones ordenada por su genero
```

```
      System.out.println("Lista de canciones ordenadas por genero:");
      System.out.println(listaCanciones);

      // Ordena la lista de canciones por su fecha
      Collections.sort(listaCanciones, new ComparaMediosFecha());

      // Despliega la lista de canciones ordenada por su fecha
      System.out.println("Lista de canciones ordenadas por fecha:");
      System.out.println(listaCanciones);

      // Ordena la lista de canciones por su interprete
      Collections.sort(listaCanciones, new ComparaCancionesInterprete());

      // Despliega la lista de canciones ordenada por su interprete
      System.out.println("Lista de canciones ordenadas por interprete:");
      System.out.println(listaCanciones);

      // Ordena la lista de canciones por su album
      Collections.sort(listaCanciones, new ComparaCancionesAlbum());

      // Despliega la lista de canciones ordenada por su album
      System.out.println("Lista de canciones ordenadas por album:");
      System.out.println(listaCanciones);
    } catch (PersistenciaException fe) {
      // Muestra el mensaje de error amistoso
      System.out.println("Error: " + fe.getMessage());
    }
  }
}
```

El fragmento de código anterior prodce la siguiente salida:

```
...

Lista de canciones:
[CBB0001, The long and winding road, Balada, 3, 24/3/1970, The Beatles, John
Lennon, Let it be, CSD0002, Garota de Ipanema, Bossanova, 3, 1/12/1970, Los
Indios Tabajaras, Antonio Carlos Jobim, Bossanova Jazz Vol. 1, CSB0003,
Desafinado, Bossanova, 3, 3/12/1980, Joao Gilberto, Joao Gilberto, Bossanova
Jazz Vol. 1]
Lista de canciones ordenadas por clave:
[CBB0001, The long and winding road, Balada, 3, 24/3/1970, The Beatles, John
Lennon, Let it be, CSB0003, Desafinado, Bossanova, 3, 3/12/1980, Joao
Gilberto, Joao Gilberto, Bossanova Jazz Vol. 1, CSD0002, Garota de Ipanema,
Bossanova, 3, 1/12/1970, Los Indios Tabajaras, Antonio Carlos Jobim,
Bossanova Jazz Vol. 1]
Lista de canciones ordenadas por titulo:
[CSB0003, Desafinado, Bossanova, 3, 3/12/1980, Joao Gilberto, Joao Gilberto,
Bossanova Jazz Vol. 1, CSD0002, Garota de Ipanema, Bossanova, 3, 1/12/1970,
Los Indios Tabajaras, Antonio Carlos Jobim, Bossanova Jazz Vol. 1, CBB0001,
The long and winding road, Balada, 3, 24/3/1970, The Beatles, John Lennon,
Let it be]
Lista de canciones ordenadas por titulo en orden inverso:
[CBB0001, The long and winding road, Balada, 3, 24/3/1970, The Beatles, John
Lennon, Let it be, CSD0002, Garota de Ipanema, Bossanova, 3, 1/12/1970, Los
Indios Tabajaras, Antonio Carlos Jobim, Bossanova Jazz Vol. 1, CSB0003,
```

```
Desafinado, Bossanova, 3, 3/12/1980, Joao Gilberto, Joao Gilberto, Bossanova
Jazz Vol. 1]
Lista de canciones ordenadas por genero:
[CBB0001, The long and winding road, Balada, 3, 24/3/1970, The Beatles, John
Lennon, Let it be, CSD0002, Garota de Ipanema, Bossanova, 3, 1/12/1970, Los
Indios Tabajaras, Antonio Carlos Jobim, Bossanova Jazz Vol. 1, CSB0003,
Desafinado, Bossanova, 3, 3/12/1980, Joao Gilberto, Joao Gilberto, Bossanova
Jazz Vol. 1]
Lista de canciones ordenadas por fecha:
[CBB0001, The long and winding road, Balada, 3, 24/3/1970, The Beatles, John
Lennon, Let it be, CSD0002, Garota de Ipanema, Bossanova, 3, 1/12/1970, Los
Indios Tabajaras, Antonio Carlos Jobim, Bossanova Jazz Vol. 1, CSB0003,
Desafinado, Bossanova, 3, 3/12/1980, Joao Gilberto, Joao Gilberto, Bossanova
Jazz Vol. 1]
Lista de canciones ordenadas por interprete:
[CSB0003, Desafinado, Bossanova, 3, 3/12/1980, Joao Gilberto, Joao Gilberto,
Bossanova Jazz Vol. 1, CSD0002, Garota de Ipanema, Bossanova, 3, 1/12/1970,
Los Indios Tabajaras, Antonio Carlos Jobim, Bossanova Jazz Vol. 1, CBB0001,
The long and winding road, Balada, 3, 24/3/1970, The Beatles, John Lennon,
Let it be]
Lista de canciones ordenadas por album:
[CSB0003, Desafinado, Bossanova, 3, 3/12/1980, Joao Gilberto, Joao Gilberto,
Bossanova Jazz Vol. 1, CSD0002, Garota de Ipanema, Bossanova, 3, 1/12/1970,
Los Indios Tabajaras, Antonio Carlos Jobim, Bossanova Jazz Vol. 1, CBB0001,
The long and winding road, Balada, 3, 24/3/1970, The Beatles, John Lennon,
Let it be]
...
```