Trabajo Práctico Especial

Programación Orientada a Objetos

Universidad Nacional del Centro de la Provincia de Buenos Aires

Horquin, Enzo Nicolás
Viñals, Tomás

Ayudante Asignado: Mauricio Arroqui

9 de Mayo del 2016

1. Clases Implementadas

1.1 ArchivoMusica

La clase *ArchivoMusica* fue implementada para poder abstraer el comportamiento básico de las clases Pista, Playlist y PlaylistAutomática. Se decidió que esta clase sea abstracta y “padre” de las tres clases mencionadas anteriormente, ya que una playlist puede tener Pistas y a su vez otras Playlist contenidas.

1.1.1 Métodos

public abstract int getCantidadElementos()
public abstract int getDuracion()
public abstract void Imprimir()
public abstract Playlist buscar(Filtro F)

1.2 Pista

Esta clase representa a cualquier canción que se desee instanciar para poder ser agregada a cualquier playlist. Esta clase, hereda de la clase *ArchivoMusica*.

1.2.1 Atributos privados

Para la clase Pista, se utilizaron los siguientes atributos:

- int id.
- String Titulo.
- int duracion.
- String Artista.
- String TituloAlbum.
- int Año.
- String Genero.
- String Comentarios.

1.2.2 Métodos

A continuación se explicará cada uno de los métodos más importantes de la clase Pista.

- **public Playlist buscar(Filtro F)**

  Este método se encarga, dependiendo si cumple o no el requisito de filtrado de búsqueda, de agregar la Pista a una Playlist resultado.

- **public int getDuracion()**

  Retorna la duración de la Pista.

1.3 Playlist

Esta clase es la que le permite al usuario poder crear y administrar cualquier *lista de reproducción*. Hereda de la clase *ArchivoMusica,* dado que comparte ciertos métodos con la clase Pista, y porque dentro de una playlist puede haber otra playlist contenida.

1.3.1 Atributos privados

- List<ArchivoMusica> listaPlaylist.
- String Titulo.
- int duracion.

1.3.2 Métodos públicos

- **public Playlist buscar(Filtro F)**

Por cada elemento de la listaPlaylist, se hará un llamado al método buscar. Si ese elemento *ArchivoMusica* cumple con el requisito del filtro, se lo agrega a una Playlist resultado que será la retornada por el método.

- **public int getDuracion()**

Retorna la duración de la Playlist.

- **public int getCantidadElementos()**

Retorna la cantidad de elementos que pertenecen a la Playlist.

- **public void copiarPlaylist(Playlist Pl)**

Copia todos los elementos pertenecientes a la instancia del objeto “this” a la playlist Pl.

- **public void cambiarOrden(Pista P1, Pista P2)**

Dadas dos Pistas, P1 y P2, este método intercambia el orden dentro de la Playlist.

1.3.3 Clases Privadas dentro de Playlist

Para poder recorrer una Playlist que contiene a todas las Pistas instanciadas en tiempo de ejecución, se tuvo que crear una clase privada *IteradorPlaylist* que implemente a la clase *Iterator*, parametrizada en Pista.

1.3.3.1 Métodos Públicos

- **public boolean hasNext()**

Retorna true o false, dependiendo si la Playlist posee o no un elemento en la posición siguiente del iterador.

- **public Pista next()**

Retorna la Pista en la cual está posicionada el iterador.

1.4 PlaylistAutomatica

Esta clase se implementó para tratar un caso especial de Playlist, cuyo contenido se actualiza dinámicamente, es decir, en cada acceso a la misma.

1.4.1 Atributos Privados

-String Titulo.
-Filtro Fil.
-Playlist Colección.

1.4.2 Métodos Públicos

- **public Playlist actualizarPlaylistAutomatica(Filtro Fil,Playlist Col)**

Este método es el que estará actualizando la Playlist cada vez que se quiera acceder a la PlaylistAutomatica. Dependiendo del estado actual de la playlist *Col*, se retornará una Playlist que cumpla los requisitos del Filtro *Fil*.

- **public int getDuracion()**

Primero se hará uso del metodo *actualizarPlaylistAutomatica* y se retornará la duración de la Playlist resultante.

- **public int getCantidadElementos()**

Se invocará al método *actualizarPlaylistAutomatica* y se retornará la cantidad de elementos que tiene la Playlist.

- **public Playlist buscar(Filtro F)**

Se usará el método *buscar* de Playlist con el filtro *F,* a la playlist resultante de *actualizarPlaylistAutomatica.*

1.4 Filtro

Para poder realizar las búsquedas requeridas por el usuario y para poder crear las nuevas playlist se creó un sistema de filtros de acuerdo a las necesidades del usuario.

1.4.1 Atributos privados

Esta clase no posee atributos por eso se decidió que sea una interfaz, ya que para este trabajo teníamos dos tipos de filtros y no compartían los mismos atributos.

1.4.2 Métodos públicos

- public boolean cumpleRequisito(Pista p)

Este método es el que se encarga de chequear si una pista dada cumple con el filtro creado por el usuario, al ser una interfaz se implementará en las clases que la implementen.

1.5 FiltroString

Es la primera subclase de filtros de las dos detectadas, de ella heredarán todas las clases que sean filtros por alguna palabra, ya sea album,genero,artista etc.

1.5.1 Atributos privados

-String umbral

1.5.2 Métodos públicos

- **public abstract String getAtributo(Pista P)**

Este método es implementado en los filtros que lo heredan.

- **public boolean cumpleRequisito(Pista P)**

Este es el método heredado de la clase interfaz y en esta clase es implementado para saber si una pista determinada cumple con el filtro creado, chequeando que el valor deseado y el umbral sean iguales

1.6 FiltroEntero

La segunda subclase de filtros, de ella heredan en este caso los filtros por duración y por año

1.6.1 Atributos privados

-Integer umbral
-Integer valor

1.6.2 Metodos publicos

- **Public abstrac boolean cumpleRequisito(Pista P)**

En el caso de los filtros numéricos decidimos que implementar este método en los filtros “finales”

1.7 FIltroAlbum, FiltroGenero, FiltroArtista

Estas 3 subclases heredan de FiltroString, e implementan el unico metodo abstracto que heredaron.

1.7.1 Métodos Públicos

- **public String getAtributo(Pista P)**

Devuelve el atributo de la pista que se pasa por parámetro para poder compararlo con el filtro creado, en este caso será el género, el artista o el álbum.

1.8 FiltroDuracion, FiltroAño

Estas dos subclases heredan de FiltroEntero, e implementan el método para chequear si cumplen un requisito o no. Además para poder comprar los valores se implementó el compareTo con el objetivo de no tener que implementar una clase por cada caso (menor, mayor o igual).

1.8.1 Métodos Públicos

- **public int compareTo(Pista P)**

Este método se utiliza para comparar el valor deseado (dependiendo el filtro) con el umbral que se le pasa por parámetro. Este método devuelve un 1 si el valor es más grande que el umbral, 0 si son iguales y -1 si el valor es menor que el umbral.

- **public boolean cumpleRequisito(Pista P)**

Este método chequea si una pista dada cumple un requisito deseado mediante la utilización del compareTo y un valor que el usuario pasa como parámetro para ver si desea filtrar por menor, igual o mayor(-1,0,1).

1.9 FIltroAnd,FiltroOr,FiltroNot

Estos filtros fueron creados para poder tener un filtro que contemple dos condiciones o que no cumpla una condición. Las tres clases implementan a la interfaz Filtro.

1.9.2 Metodos Publicos

- public boolean cumpleRequisito(Pista P)

Dependiendo el caso chequea que una pista cunmpla dos filtros, cumpla uno de los dos, o no cumpla un filtro.

2. Diagrama de Clases