UNED

ETS de Ingeniería Informática

Estrategias de Programación y Estructuras de Datos

Grado en Ingeniería Informática

Grado en Tecnologías de la Información

Práctica curso 2016-2017

Enunciado

Índice

1. Presentación del problema

Los reproductores de música nos permiten administrar nuestra colección de canciones ofreciéndonos algunas funcionalidades como la creación de listas de reproducción (que son agrupaciones de canciones bajo cierto criterio común, por ejemplo "*canciones para conducir*", "*canciones para hacer ejercicio*", etc.), buscar todas las canciones que tengamos de cierto artista (o de una época determinada, o de un estilo común…) y organizar la reproducción de las canciones.

Toda esta administración requiere la organización de la información de las canciones para poder realizar, entre otras, las siguientes acciones:

- Organizar y acceder a las canciones que tenemos en nuestra colección.
- Crear y gestionar varias listas de reproducción.
- Seleccionar las canciones que van a ser reproducidas.
- Consultar cuáles han sido las últimas canciones reproducidas.

En esta práctica nos ocuparemos de diseñar y programar una serie de componentes que nos van a permitir gestionar toda esta información. De las canciones nos interesarán sus metadatos (autor, título, álbum al que pertenece, año de publicación, etc.), pero no contemplaremos la reproducción en sí. Es decir, sólo vamos a manejar la información asociada a la canción, sin ocuparnos de que la canción llegue a sonar.

Los objetivos principales de esta práctica son:

- Comprender la implementación de Tipos Abstractos de Datos (TAD)
- Ejercitar el uso de TAD
- Implementar nuevos TAD a partir de otros conocidos

Como objetivos adicionales, podemos destacar:

- Familiarizarse con la definición de interfaces de TAD
- Implementar la solución de un problema completo utilizando TAD en Java
- Familiarizarse con la prueba de programas mediante la técnica de Juegos de Pruebas
- Familiarizarse con el uso de entornos de programación (IDE)

2. Diseño.

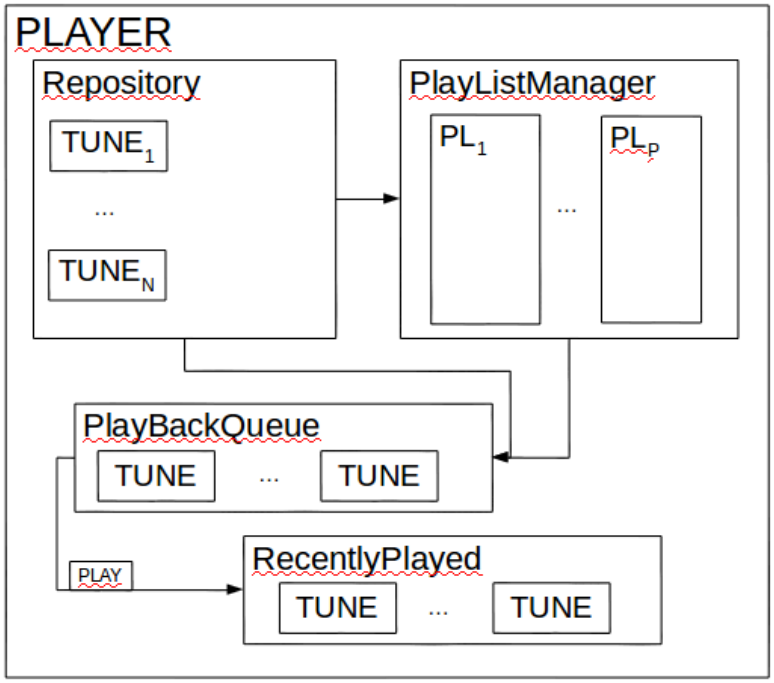
En nuestro reproductor vamos a tener cuatro componentes principales:

- **Repository**: contendrá **todas** las canciones de nuestra colección, que serán las únicas a las que tendrá acceso nuestro reproductor. Cada canción vendrá representada por sus metadatos (título, autor, género, álbum al que pertenece, año de publicación y duración). Por simplicidad no se podrá modificar el repositorio una vez construido, es decir, no se podrán añadir ni eliminar canciones del mismo.
- **PlayListManager**: es el componente encargado de la gestión de las diferentes listas de reproducción (cada una de las cuales será accesible mediante un identificador), que se construyen añadiendo canciones presentes en el repositorio. Cada lista individual podrá

almacenar un número no acotado de canciones del repositorio y se admite que una misma canción pueda aparecer más de una vez en una misma lista o en varias.

- **PlayBackQueue**: representa la cola de canciones que están a la espera de ser reproducidas. Se construye añadiendo canciones directamente desde el repositorio o bien a partir de alguna de las listas de reproducción. Así, por ejemplo, si disponemos de una lista de reproducción con "*canciones que le gustan al niño*" y otra con "*canciones que le gustan a la niña*", podremos añadir ambas listas a la cola de reproducción para que sean reproducidas. Por lo tanto, la cola de reproducción podrá almacenar un número no acotado de canciones, cada una de las cuales podrá aparecer más de una vez.
- **RecentlyPlayed**: nos permitirá almacenar las últimas canciones que hayan sido reproducidas. Si bien se puede reproducir un número no acotado de canciones, sólo se almacenará un número máximo de ellas. Es decir, cuando se reproduzca una canción, ésta deberá salir de **PlayBackQueue** y pasar a **RecentlyPlayed**. Esto puede provocar la eliminación en **RecentlyPlayed** de la canción menos reciente.

A continuación mostramos un esquema en el que se ven los cuatro componentes que acabamos de describir unidos mediante flechas, las cuales indican el flujo de la información sobre las canciones:



Así:

- En **Repository** tenemos toda nuestra colección de canciones ($TUNE_1$ a $TUNE_N$ en el esquema).
- En **PlayListManager** tendremos nuestras listas de reproducción (PL_1 a PL_P en el esquema) que se forman exclusivamente a partir de canciones existentes en el repositorio. Cada lista de reproducción será accesible mediante un identificador.
- En **PlayBackQueue** se encuentran las canciones que van a ser reproducidas. Se construye añadiendo canciones directamente desde el repositorio o bien desde una de las listas de reproducción existentes.
- En **RecentlyPlayed** almacenaremos las últimas canciones que han sido reproducidas, extrayéndolas de **PlayBackQueue** cada vez que se reproduzca una.

En cuanto a la selección de canciones (para su inserción en una lista de reproducción o directamente en la cola de reproducción) se podrá realizar de dos formas diferentes:

- Indicando de forma explícita qué canciones se desea añadir.
- Mediante una serie de criterios de búsqueda, lo que añadirá todas las canciones del repositorio que los cumplan. Por ejemplo, todas las canciones de Leonard Cohen de los años 80.

Para completar el diseño, vamos a ver los diferentes TAD que vamos a considerar en nuestra práctica. Para cada uno de ellos veremos qué operaciones van a ofrecer (agrupándolas bajo las siglas **C.R.U.D.**: **CREATE** creación, **READ** consulta, **UPDATE** modificación y **DELETE** borrado) y cómo han de comportarse éstas (casos de uso).

2.1 Player.

El TAD `Player` representa un reproductor de música. Está compuesto por cuatro componentes:

- Un repositorio de canciones (ver `TuneCollection`).
- Un gestor de listas de reproducción (ver `PlayListManager`).
- Una cola de reproducción (ver `PlayBackQueue`).
- Un almacén con las últimas canciones reproducidas (ver `RecentlyPlayed`).

y ofrece las siguientes operaciones:

- **Operaciones de creación**:
  - Una operación de creación que creará un nuevo Reproductor de Música a partir de una colección de canciones.
- **Operaciones de consulta**:
  - Una operación que devuelva la secuencia de los identificadores de todas las listas de reproducción existentes en el reproductor.
  - Una operación que devuelva una secuencia con el contenido de una lista de reproducción concreta del reproductor a partir de su identificador.
  - Una operación que devuelva una secuencia con el contenido de la cola de reproducción.
  - Una operación que devuelva una secuencia con las últimas canciones reproducidas.
- **Operaciones de modificación**:
  - Una operación que permita crear una nueva lista de reproducción a partir de un identificador para la misma. Si ya existiera una lista de reproducción con dicho identificador, no se hará nada. De esta forma se garantiza la unicidad del identificador de cada lista de reproducción.
  - Una operación que permita eliminar una de las listas de reproducción existentes a partir de su identificador. Si no existe ninguna lista de reproducción con dicho identificador, no se hará nada.
  - Una operación que permita añadir una lista de canciones del repositorio, a partir de sus identificadores, a una lista de reproducción existente en el reproductor seleccionada mediante su identificador. Si no existe ninguna lista de reproducción con dicho identificador, no se hará nada.
  - Una operación que permita añadir todas las canciones del repositorio que cumplan unos criterios a una lista de reproducción existente en el reproductor seleccionada mediante su identificador. Si no existe ninguna lista de reproducción con dicho identificador, no se hará nada.

    - Una operación que permita eliminar una canción, a partir de su identificador, de una lista de reproducción existente en el reproductor seleccionada mediante su identificador. Si no existe ninguna lista de reproducción con dicho identificador, no se hará nada.
    - Una operación que permita añadir una lista de canciones del repositorio, a partir de sus identificadores, a la cola de reproducción.
    - Una operación que permita añadir todas las canciones del repositorio que cumplen unos criterios a la cola de reproducción.
    - Una operación que permita añadir una de las listas de reproducción existentes en el repositorio, seleccionada a partir de su identificador, a la cola de reproducción. Si no existe ninguna lista de reproducción con dicho identificador, no se hará nada.
    - Una operación que permita vaciar la cola de reproducción.
    - Una operación que reproduzca la siguiente canción disponible en la cola de reproducción. Consecuentemente, esta operación deberá guardar la canción reproducida en el almacén de últimas canciones reproducidas.
- **Operaciones de borrado**:
  No vamos a considerar que un reproductor de música pueda ser borrado, por lo que no contemplaremos este tipo de operaciones en este TAD.

Como se puede ver, las operaciones de este TAD son algunas de las operaciones típicas que un reproductor de música ofrece al usuario. Ese es uno de los motivos por el cual este TAD ofrece bastantes operaciones, pero la mayoría de ellas delegarán el trabajo en operaciones de alguno de los TAD que representan los componentes que forman nuestro reproductor.

2.2 TuneCollection.

El TAD `TuneCollection` representa una colección de canciones (ver clase `Tune`). Cada canción podrá referenciarse en él mediante un identificador entero y ofrece las siguientes operaciones:

- **Operaciones de creación**:
    - Una operación de creación que creará una nueva colección de canciones cargándolas a partir de un fichero.
- **Operaciones de consulta**:
    - Una operación que devuelva el tamaño de la colección de canciones cargada.
    - Una operación que dado un identificador de canción contenida en la colección, devuelva dicha canción.
- **Operaciones de modificación**:
  Por simplicidad, no vamos a considerar que una colección de canciones pueda modificarse una vez construida, por lo que no contemplaremos ese tipo de operaciones en esta clase.
- **Operaciones de borrado**:
  No vamos a considerar que este componente pueda ser borrado, por lo que no contemplaremos este tipo de operaciones en este TAD.

El Equipo Docente dará una implementación completa de este TAD.

2.3 Tune.

Este TAD representa representa una canción a través de los siguientes atributos:

- **Título**: una cadena de caracteres no vacía con el título de la canción.

- **Autor**: una cadena de caracteres no vacía con el autor de la canción.
- **Género**: una cadena de caracteres no vacía con el género de la canción.
- **Álbum**: una cadena de caracteres no vacía con el nombre del álbum al que pertenece la canción.
- **Año**: un entero positivo indicando el año de publicación de la canción.
- **Duración**: un entero positivo indicando la duración, en segundos, de la canción.

y ofrece las siguientes operaciones:

- **Operaciones de creación**:
  - Una operación de creación que creará una nueva canción a partir de los valores para sus atributos (título, autor, género, álbum, año y duración en segundos).
- **Operaciones de consulta**:
  - Una operación que, dados unos criterios de búsqueda (ver `Query`), indique si la canción los cumple todos o no.
- **Operaciones de modificación:**
  Por simplicidad no vamos a considerar que una canción pueda modificarse una vez construida, por lo que no contemplaremos ese tipo de operaciones en este TAD.
- **Operaciones de borrado:**
  Por simplicidad no vamos a considerar que una canción pueda ser borrada, por lo que no contemplaremos este tipo de operaciones en esta clase.

Vemos que la operación de consulta es la encargada de comprobar si la canción cumple o no con los criterios de búsqueda definidos por unos criterios de búsqueda que recibe como parámetro. A continuación vamos a definir cuándo una canción cumple los criterios de búsqueda.

En primer lugar, hay que hacer notar que se establecen todos los criterios de búsqueda simultáneamente, pero en muchas ocasiones no querremos buscar por todos ellos a la vez. Por ejemplo, si buscamos canciones de Leonard Cohen de los años 80, el título de la canción no es relevante. Por tanto estableceremos un valor por defecto para cada criterio (que no pueda aparecer en los atributos de ninguna canción), que indicará que dicho criterio no interviene en la búsqueda:

> Para los atributos representados mediante una cadena de caracteres, dicho valor por defecto será la cadena vacía (`""`), ya que ninguna canción puede tener un título, autor, género o álbum representados por la cadena vacía. Por otro lado, para los atributos representados mediante enteros será el `-1`, ya que los atributos numéricos (año y duración en segundos) de las canciones se han de representar mediante un entero positivo. De esta forma, si un criterio de búsqueda contiene el valor por defecto, significará que no tenemos que comprobar si la canción lo cumple o no.

Para los criterios que sí intervengan en la búsqueda:

- Para los criterios representados por una cadena de caracteres, la canción cumplirá el criterio sí y sólo si el valor del atributo correspondiente coincide (salvo mayúsculas y minúsculas) con el valor del criterio.
- Para los criterios representados por un entero:
  - Año mínimo: el año de publicación de la canción deberá ser mayor o igual que el del criterio.

- Año máximo: el año de publicación de la canción deberá ser menor o igual que el del criterio.
- Duración mínima: la duración de la canción deberá ser mayor o igual que la del criterio.
- Duración máxima: la duración de la canción deberá ser menor o igual que la del criterio.

Por ejemplo, para buscar todas las canciones de Beethoven, crearíamos los siguientes criterios de búsqueda:

- **Título**: `""`
- **Autor**: `"Beethoven"`
- **Género**: `""`
- **Álbum**: `""`
- **Año mínimo**: `-1`
- **Año máximo**: `-1`
- **Duración mínima**: `-1`
- **Duración máxima**: `-1`

Con esto estaremos indicando que sólo queremos comprobar si el autor de la canción coincide (salvo mayúsculas y minúsculas) con la cadena "`Beethoven`".

Si ahora quisiéramos buscar todas las canciones de los Rolling Stones entre 1980 y 1989 (ambos inclusive) que durasen al menos dos minutos (120 segundos), tendríamos que crear los siguientes criterios de búsqueda:

- **Título**: `""`
- **Autor**: `"Rolling Stones"`
- **Género**: `""`
- **Álbum**: `""`
- **Año mínimo**: `1980`
- **Año máximo**: `1989`
- **Duración mínima**: `120`
- **Duración máxima**: `-1`

2.4 PlayListManager.

El TAD `PlayListManager` será el encargado de gestionar las diferentes listas de reproducción que pueden definirse en el reproductor. Asociará cada lista de reproducción a un identificador único representado por una cadena de caracteres no vacía. Ofrece las siguientes operaciones:

- **Operaciones de creación**:
  - Una operación de creación que inicializará el componente sin ninguna lista de reproducción.
- **Operaciones de consulta**:
  - Una operación que compruebe si existe una lista de reproducción a partir de su identificador.
  - Una operación que devuelva una lista de reproducción existente a partir de su identificador.
  - Una operación que devuelva la lista de los identificadores de todas las listas de

reproducción existentes.

- **Operaciones de modificación**:
  - Operaciones de modificación:
  - Una operación que permita crear una nueva lista de reproducción a partir de un identificador que no pertenezca a otra lista de reproducción existente.
  - Una operación que permita eliminar una lista de reproducción existente a partir de su identificador.
- **Operaciones de borrado**:
  No vamos a considerar que este componente pueda ser borrado, por lo que no contemplaremos este tipo de operaciones en este TAD.

2.5 PlayList.

El TAD `PlayList` representa una lista de reproducción. Toda lista de reproducción deberá contener una colección (de tamaño no acotado) de identificadores de canciones presentes en el repositorio ordenados de forma implícita según estos vayan siendo introducidos en la lista de reproducción. Un mismo identificador podrá aparecer más de una vez. Las operaciones que ofrece son las siguientes:

- **Operaciones de creación**:
  - Una operación de creación que creará una nueva lista de reproducción.
- **Operaciones de consulta**:
  - Una operación que devuelva la secuencia de los identificadores de canciones almacenada en la lista de reproducción, en el mismo orden en el que se encuentran almacenados en la lista.
- **Operaciones de modificación:**
  - Una operación que, dada una lista de identificadores de canciones, añada dichos identificadores a la lista de reproducción.
  - Una operación que, dado un identificador de canción, elimine toda aparición de dicho identificador en la lista de reproducción.
- **Operaciones de borrado:**
  Una lista de reproducción puede ser eliminada, pero no se requiere de una operación específica a la hora de hacerlo, por lo que no contemplaremos este tipo de operaciones en este TAD.

2.6 Query.

El TAD `Query` representa unos criterios para realizar una búsqueda en el repositorio. Estos criterios comprenden:

- **Título**: título de la canción a buscar expresado en forma de cadena de caracteres.
- **Autor**: autor de la canción a buscar expresado en forma de cadena de caracteres.
- **Género**: género de la canción a buscar expresado en forma de cadena de caracteres.
- **Álbum**: álbum al que pertenece la canción a buscar expresado en forma de cadena de caracteres.
- **Año mínimo**: el primer año del intervalo en el que se publicó la canción a buscar, expresado mediante un entero.
- **Año máximo**: el último año del intervalo en el que se publicó la canción a buscar, expresado mediante un entero.

- **Duración mínima**: mínimo valor de la duración, en segundos, de la canción a buscar, expresado mediante un entero.
- **Duración máxima**: máximo valor de la duración, en segundos, de la canción a buscar, expresado mediante un entero.

Las operaciones que ofrece este TAD son las siguientes:

- **Operaciones de creación**:
  - Una operación de creación que creará un nuevo objeto conteniendo unos criterios de búsqueda a partir de dichos criterios: título, autor, género, álbum, año mínimo, año máximo, duración mínima y duración máxima.
- **Operaciones de consulta**:
  - Una operación que devuelva cada uno de los criterios definidos. Es decir, un total de ocho operaciones de consulta que permitan obtener los criterios de forma individual.
- **Operaciones de modificación:**
  No vamos a considerar que los criterios de búsqueda puedan ser modificados posteriormente a su creación, por lo que no contemplaremos este tipo de operaciones en esta clase.
- **Operaciones de borrado:**
  No vamos a considerar que los criterios de búsqueda puedan ser eliminados, por lo que no contemplaremos este tipo de operaciones en esta clase.

2.7 PlayBackQueue.

El TAD `PlayBackQueue` representa la cola de reproducción que contiene la colección (de tamaño no acotado) de los identificadores de las canciones presentes en el repositorio que van a ser reproducidas. Las canciones serán reproducidas en el mismo orden en el que se insertaron en la cola.

Ofrece las siguientes operaciones:

- **Operaciones de creación**:
  - Una operación de creación que creará una nueva cola de reproducción vacía, es decir, que no contendrá ningún identificador de canción.
- **Operaciones de consulta**:
  - Una operación que devuelva la secuencia con los identificadores de las canciones que están programadas para su reproducción en el mismo orden en el que éstos fueron introducidos.
  - Una operación que indique si la cola de reproducción está vacía o no.
  - Una operación que devuelva el identificador de la primera canción presente en una cola de reproducción no vacía.
- **Operaciones de modificación:**
  - Una operación que extraiga el identificador de la primera canción presente en una cola de reproducción no vacía.
  - Una operación que, dada una lista de identificadores de canciones, añada dichos identificadores a la cola de reproducción.
  - Una operación que vacíe por completo la cola de reproducción.
- **Operaciones de borrado:**
  No vamos a considerar que este componente pueda ser borrado, por lo que no contemplaremos este tipo de operaciones en este TAD.

2.8 RecentlyPlayed.

El TAD `RecentlyPlayed` nos permitirá gestionar los identificadores de las últimas canciones reproducidas por nuestro reproductor. Su tamaño estará acotado.

- **Operaciones de creación**:
  - Una operación de creación que creará un nuevo objeto para almacenar los identificadores de las últimas canciones que hayan sido reproducidas. Recibirá un parámetro que representará su tamaño máximo.
- **Operaciones de consulta**:
  - Una operación que devuelva la secuencia de los identificadores de las últimas canciones reproducidas que se encuentran almacenadas, en el orden inverso a su orden de reproducción.
- **Operaciones de modificación:**
  - Una operación que añade una nueva canción reproducida a la estructura. Esta operación deberá asegurarse de que nunca se almacenan más canciones reproducidas que las indicadas por el tamaño máximo indicado en el constructor.
- **Operaciones de borrado:**
  No vamos a considerar que este componente pueda ser borrado, por lo que no contemplaremos este tipo de operaciones en este TAD.

3. Interfaces de los TAD.

A continuación vamos a describir los TAD en forma de interfaces Java. Para cada una de las operaciones de modificación, acceso y borrado descritas en el apartado anterior, la interfaz prescribirá un método que deberá ser implementado por la clase que implemente el interfaz.

Añadiremos, en forma de anotaciones, la descripción de los parámetros de entrada y salida, precondiciones y postcondiciones cuando sea preciso.

3.1 Interfaz PlayerIF

```
/* Representación de un reproductor de música                              */
public interface PlayerIF {

  /* Devuelve los identificadores de todas las listas de reproducción      */
  /* existentes                                                            */
  /* @returns -una lista con los identificadores de todas las listas de    */
  /*          reproducción (no importa el orden)                           */
  public ListIF<String> getPlayListIDs();

  /* Devuelve el contenido de una lista de reproducción                    */
  /* @param   -una cadena de caracteres no vacía con el identificador de la */
  /*          lista de reproducción de la que se quiere obtener su contenido */
  /* @return  -si en el reproductor existe una lista de reproducción con ese */
  /*          identificador, se devolverá una lista con su contenido       */
  /*          -en caso contrario, se devolverá una lista vacía             */
  public ListIF<Integer> getPlayListContent(String playListID);

  /* Devuelve los identificadores de las canciones contenidas en la cola de */
  /* reproducción                                                          */
  /* @return  una lista con los identificadores de las canciones que están */
  /*          en la cola de reproducción (ha de conservar el orden en el que */
```

```
/*          se introdujeron las canciones)                                  */
public ListIF<Integer> getPlayBackQueue();

/* Devuelve los identificadores de las últimas canciones reproducidas que  */
/* están almacenadas en RecentlyPlayed                                     */
/* @return  una lista con los identificadores de las últimas canciones     */
/*          reproducidas (en el orden inverso al que se reprodujeron)      */
public ListIF<Integer> getRecentlyPlayed();

/* Crea una nueva lista de reproducción a partir de su identificador       */
/* @param   -una cadena de caracteres no vacía con el identificador de la  */
/*          nueva lista de reproducción                                    */
/* @pos     -si no existe una lista de reproducción con ese identificador, */
/*          se crea                                                        */
/*          -en caso contrario, no se hace nada                            */
public void createPlayList(String playListID);

/* Elimina una lista de reproducción del reproductor a partir de su        */
/* identificador                                                           */
/* @param   -una cadena de caracteres no vacía con el identificador de la  */
/*          lista de reproducción a eliminar                               */
/* @pos     -si existe una lista de reproducción con ese identificador, se */
/*          elimina                                                        */
/*          -en caso contrario, no se hace nada                            */
public void removePlayList(String playListID);

/* Añade una lista de identificadores de canciones del repositorio a una   */
/* lista de reproducción                                                   */
/* @param   -una cadena de caracteres no vacía con el identificador de la  */
/*          lista de reproducción a la que se van a añadir las canciones   */
/*          -una lista de identificadores de canciones contenidas en el    */
/*          repositorio                                                    */
/* @pre     -todos los elementos de la lista son identificadores de        */
/*          canciones que existen dentro del repositorio                   */
/* @pos     -si existe una lista de reproducción con ese identificador, se */
/*          añaden a ella los identificadores contenidos en la lista       */
/*          -en caso contrario, no se hace nada                            */
public void addListOfTunesToPlayList(String playListID,ListIF<Integer> lT);

/* Añade los identificadores de todas las canciones del repositorio que    */
/* cumplan los criterios indicados a una lista de reproducción             */
/* @param   -una cadena de caracteres no vacía con el identificador de la  */
/*          lista de reproducción a la que se van a añadir las canciones   */
/*          -una cadena de caracteres con el título de la canción buscada  */
/*          -una cadena de caracteres con el autor de la canción buscada   */
/*          -una cadena de caracteres con el género de la canción buscada  */
/*          -una cadena de caracteres con el álbum al que pertenece la     */
/*          canción buscada                                                */
/*          -un entero con el primer año del intervalo en el que se        */
/*          publicó la canción a buscar                                    */
/*          -un entero con el último año del intervalo en el que se        */
/*          publicó la canción a buscar                                    */
/*          -un entero con la duración mínima de la canción a buscar       */
/*          -un entero con la duración máxima de la canción a buscar       */
/* @pos     -si existe una lista de reproducción con se identificador, se  */
/*          añaden a ella los identificadores de todas las canciones del   */
/*          repositorio que cumplan todos los criterios indicados          */
```

```
/*          -en caso contrario, no se hace nada                              */
public void addSearchToPlayList(String playListID,
                                String t, String a, String g, String al,
                                int min_y, int max_y,
                                int min_d, int max_d);

/* Elimina una canción de una lista de reproducción                         */
/* @param   -una cadena de caracteres no vacía con el identificador de la   */
/*          lista de reproducción de la que se quiere eliminar la canción   */
/*          -un entero con el identificador de la canción del repositorio   */
/*          que se quiere eliminar de dicha lista                           */
/* @pos     -si existe una lista de reproducción con se identificador, se   */
/*          elimina de dicha lista todas las apariciones del identificador  */
/*          de la canción del repositorio pasada como parámetro             */
/*          -en caso contrario, no se hace nada                             */
public void removeTuneFromPlayList(String playListID,int tuneID);

/* Añade una lista de identificadores de canciones del repositorio a la     */
/* cola de reproducción                                                     */
/* @param   -una lista de identificadores de canciones contenidas en el     */
/*          repositorio                                                     */
/* @pre     -todos los elementos de la lista son identificadores de         */
/*          canciones que existen dentro del repositorio                    */
/* @pos     se añaden a la cola de reproducción los identificadores de las  */
/*          canciones contenidos en la lista                                */
public void addListOfTunesToPlayBackQueue(ListIF<Integer> lT);

/* Añade los identificadores de todas las canciones del repositorio que     */
/* cumplan los criterios indicados a la cola de reproducción                */
/* @param   -una cadena de caracteres con el título de la canción buscada   */
/*          -una cadena de caracteres con el autor de la canción buscada    */
/*          -una cadena de caracteres con el género de la canción buscada   */
/*          -una cadena de caracteres con el álbum al que pertenece la      */
/*          canción buscada                                                 */
/*          -un entero con el primer año del intervalo en el que se creó    */
/*          la canción a buscar                                             */
/*          -un entero con el último año del intervalo en el que se creó    */
/*          la canción a buscar                                             */
/*          -un entero con la duración mínima de la canción a buscar        */
/*          -un entero con la duración máxima de la canción a buscar        */
/* @pos     se añaden a la cola de reproducción los identificadores de      */
/*          todas las canciones del repositorio que cumplan todos los       */
/*          criterios indicados                                             */
public void addSearchToPlayBackQueue(String t, String a, String g, String al,
                                     int min_y, int max_y,
                                     int min_d, int max_d);

/* Añade el contenido de una lista de reproducción a la cola de             */
/* reproducción                                                             */
/* @param   -una cadena de caracteres no vacía con el identificador de la   */
/*          lista de reproducción cuyo contenido se desea añadir a la cola  */
/*          de reproducción                                                 */
/* @pos     -si existe una lista de reproducción con se identificador, se   */
/*          añade su contenido a la cola de reproducción                    */
/*          -en caso contrario, no se hace nada                             */
public void addPlayListToPlayBackQueue(String playListID);
```

```
  /* Vacía la cola de reproducción                                          */
  /* @pos     -la cola de reproducción se vacía                             */
  public void clearPlayBackQueue();

  /* Reproduce la siguiente canción en la cola de reproducción              */
  /* @pos     -si la cola de reproducción no es vacía, se elimina de ella el */
  /*          primer elemento, pasando éste a la estructura que almacena las */
  /*          últimas canciones reproducidas, sin sobrepasar su tamaño      */
  /*          máximo                                                        */
  /*          -en caso contrario, no se hace nada                           */
  public void play();
}
```

3.2 Interfaz TuneCollectionIF

```
/* Representación del repositorio de canciones                             */
public interface TuneCollectionIF {

  /* Devuelve el tamaño de la colección de canciones                        */
  /* @return  -devuelve un entero con el número de canciones               */
  public int size();

  /* Devuelve una canción a partir de su identificador                     */
  /* @param   -un entero con el identificador de la canción a recuperar    */
  /* @pre     0<=ID<this.size()                                            */
  /* @return  -un objeto TuneIF con la canción con el identificador recibido */
  public TuneIF getTune(int ID);
}
```

3.3 Interfaz TuneIF

```
/* Representación de una canción                                           */
public interface TuneIF {

  /* Dado un objeto QueryIF conteniendo unos criterios de búsqueda, devuelve */
  /* un valor de verdad indicando si la canción los cumple o no los cumple  */
  /* @param   -un objeto QueryIF con unos criterios de búsqueda             */
  /* @return  -si la canción cumple TODOS los criterios, devolverá verdadero */
  /*          -si la canción incumple algún criterio, devolverá falso       */
  public boolean match(QueryIF q);
}
```

3.4 Interfaz PlayListManagerIF

```
/* Representación del gestor de listas de reproducción                     */
public interface PlayListManagerIF {

  /* Comprueba si existe una lista de reproducción dado su identificador   */
  /* @param   -una cadena de caracteres no vacía con el identificador de la */
  /*          lista de reproducción que queremos saber si existe o no       */
  /* @return  -un valor booleano indicando si existe o no una lista de      */
  /*          reproducción asociada al identificador recibido como parámetro */
  public boolean contains(String playListID);

  /* Devuelve la lista de reproducción asociada a un identificador         */
  /* @param   -una cadena de caracteres no vacía con el identificador de la */
```

```
  /*          lista de reproducción que queremos recuperar                    */
  /* @pre     -existe una lista de reproducción asociada al identificador     */
  /*          que se recibe como parámetro                                    */
  /* @return  -la lista de reproducción asociada al identificador recibido   */
  /*          como parámetro                                                  */
  public PlayListIF getPlayList(String playListID);

  /* Devuelve una lista con todos los identificadores de las listas de       */
  /* reproducción existentes                                                 */
  /* @return  -una lista de cadenas de caracteres (todas no vacías) que son  */
  /*          los identificadores de todas las listas de reproducción        */
  /*          existentes                                                     */
  public ListIF<String> getIDs();

  /* Crea una nueva lista de reproducción vacía y la asocia a un nuevo       */
  /* identificador                                                           */
  /* @param   -una cadena de caracteres no vacía con el identificador de la  */
  /*          lista de reproducción que queremos crear                       */
  /* @pre     -no existe ninguna lista de reproducción asociada al           */
  /*          identificador recibido como parámetro                          */
  public void createPlayList(String playListID);

  /* Elimina una lista de reproducción asociada a un identificador           */
  /* @param   -una cadena de caracteres no vacía con el identificador de la  */
  /*          lista de reproducción que queremos eliminar                    */
  /* @pre     -existe una lista de reproducción asociada al identificador    */
  /*          recibido como parámetro                                        */
  public void removePlayList(String playListID);
}
```

3.5 Interfaz PlayListIF

```
/* Representación de una lista de reproducción                              */
public interface PlayListIF {

  /* Devuelve la lista de identificadores de canciones de la lista de        */
  /* reproducción                                                            */
  /* @return  -una lista de enteros con los identificadores de las canciones */
  /*          contenidas en la lista de reproducción                         */
  public ListIF<Integer> getPlayList();

  /* Añade una lista de identificadores de canciones a la lista de           */
  /* reproducción                                                            */
  /* @param   -una lista de enteros con los identificadores de las canciones */
  /*          que se quiere añadir a la lista de reproducción                */
  /* @pre     -todos los elementos de la lista son identificadores de        */
  /*          canciones que existen dentro del repositorio                   */
  /* @pos     -el contenido de la lista recibida como parámetro se concatena */
  /*          al contenido existente en la lista de reproducción             */
  public void addListOfTunes(ListIF<Integer> lT);

  /* Elimina todas las apariciones de un identificador de canción de la      */
  /* lista de reproducción                                                   */
  /* @param   -un entero representando el identificador de la canción que se */
  /*          desea eliminar de la lista de reproducción                     */
  /* @pos     -del contenido de la lista de reproducción se han eliminado    */
```

```
  /*          todas las apariciones del identificador recibido como          */
  /*          parámetro. El resto de identificadores conserva su orden       */
  /*          relativo                                                       */
  public void removeTune(int tuneID);
}
```

3.6 Interfaz QueryIF

```
/* Representación de los criterios de búsqueda                               */
public interface QueryIF {

  /* Devuelve el criterio título                                            */
  /* @return  -una cadena de caracteres con el título de la canción buscada  */
  public String getTitle();

  /* Devuelve el criterio autor                                             */
  /* @return  -una cadena de caracteres con el autor de la canción buscada   */
  public String getAuthor();

  /* Devuelve el criterio género                                            */
  /* @return  -una cadena de caracteres con el género de la canción buscada  */
  public String getGenre();

  /* Devuelve el criterio álbum                                             */
  /* @return  --una cadena de caracteres con el álbum al que pertenece la    */
  /*          canción buscada                                               */
  public String getAlbum();

  /* Devuelve el criterio año mínimo                                        */
  /* @return  -un entero con el primer año del intervalo en el que se grabó  */
  /*          la canción a buscar                                           */
  public int getMin_year();

  /* Devuelve el criterio año máximo                                        */
  /* @return  -un entero con el último año del intervalo en el que se grabó  */
  /*          la canción a buscar                                           */
  public int getMax_year();

  /* Devuelve el criterio duración mínima                                   */
  /* @return  -un entero con la duración mínima de la canción a buscar       */
  public int getMin_duration();

  /* Devuelve el criterio duración máxima                                   */
  /* @return  -un entero con la duración máxima de la canción a buscar       */
  public int getMax_duration();
}
```

3.7 Interfaz PlayBackQueueIF

```
/* Representación de la cola de reproducción                                 */
public interface PlayBackQueueIF {

  /* Devuelve una lista con los identificadores de las canciones contenidas  */
  /* en la cola de reproducción                                             */
  /* @return  -una lista de enteros con los identificadores de las canciones */
  /*          que están en la cola de reproducción, conservando el orden en  */
  /*          el que fueron originalmente introducidos                      */
  public ListIF<Integer> getContent();
```

```
  /* Devuelve un booleano indicando si la cola de reproducción es vacía o no */
  /* @return  -devuelve un valor booleano que indica si la cola de           */
  /*          reproducción está vacía o no                                   */
  public boolean isEmpty();

  /* Devuelve un entero con el identificador de la primera canción que está  */
  /* en la cola de reproducción                                              */
  /* @pre     -la cola de reproducción no está vacía                         */
  /* @return  -devuelve el identificador de la primera canción en la cola de */
  /*          reproducción                                                   */
  public int getFirstTune();

  /* Extrae la primera canción que se encuentre en la cola de reproducción   */
  /* @pre     -la cola de reproducción no está vacía                         */
  /* @pos     -elimina de la cola de reproducción el primer identificador    */
  public void extractFirstTune();

  /* Añade una lista de identificadores de canciones a la cola de            */
  /* reproducción                                                            */
  /* @param   -una lista de enteros con los identificadores de las canciones */
  /*          que se desea añadir a la lista de reproducción                 */
  /* @pre     -todos los elementos de la lista son identificadores de        */
  /*          canciones que existen dentro del repositorio                   */
  /* @pos     -añade todos los identificadores presentes en la lista al      */
  /*          final de la cola de reproducción                               */
  public void addTunes(ListIF<Integer> lT);

  /* Vacía el contenido de la cola de reproducción                           */
  /* @pos     -la cola de reproducción queda vacía, sin identificadores      */
  public void clear();
}
```

3.8 Interfaz RecentlyPlayedIF

```
/* Representación del almacén de las últimas canciones reproducidas         */
public interface RecentlyPlayedIF {

  /* Devuelve los identificadores de las últimas canciones reproducidas en  */
  /* el orden inverso al que fueron reproducidas                             */
  /* @return  una lista con los identificadores de las últimas canciones     */
  /*          reproducidas (en el orden inverso al que se reprodujeron)      */
  public ListIF<Integer> getContent();

  /* Añade la última canción reproducida                                     */
  /* @param   -un entero con el identificador de la última canción           */
  /*          reproducida                                                    */
  /* @pos     -se añade el identificador a la estructura que almacena las    */
  /*          últimas canciones reproducidas, garantizándose que no se       */
  /*          almacenan más canciones que las marcadas por el valor máximo   */
  /*          permitido indicado en el constructor                           */
  public void addTune(int tuneID);
}
```

4. Implementación.

Se deberá realizar un programa en Java que contenga clases que implementen las interfaces descritas en el apartado anterior. Todas ellas se implementarán en un único paquete llamado:

`es.uned.lsi.eped.pract2016_2017`

Para la implementación de las estructuras de datos **se deberán utilizar las interfaces proporcionadas por el Equipo Docente** de la asignatura. El Equipo Docente, a través del Curso Virtual, proporcionará la implementación de una clase principal, los mecanismos de Entrada/Salida y una clase `TuneCollection` (que implementará el interfaz `TuneCollectionIF`).

Para cada clase, el trabajo de implementación que deberá realizar el estudiante consistirá en:

1. Elegir una estructura de datos adecuada para los objetos de la clase y plasmarla en los atributos de la clase.
2. Implementar todas las operaciones públicas de la clase cumpliendo lo indicado en el interfaz.

4.1 Constructores para las implementaciones de PlayerIF y TuneIF

Dado que parte de la implementación vendrá dada por el Equipo Docente, será necesario que algunos constructores respeten los perfiles previstos en dicha implementación. A continuación detallamos los constructores que deben implementarse obligatoriamente (lo cual no significa que no puedan existir otros constructores si se considera oportuno).

Constructor obligatorio para la implementación de PlayerIF

```
/* Constructor de la clase que implementa PlayerIF                        */
/* @param   -una colección de canciones como un objeto TuneCollectionIF    */
/*          -un entero representando el número máximo de canciones        */
/*          reproducidas que se pueden almacenar                          */
Player(TuneCollectionIF T,int maxRecentlyPlayed){}
```

Constructor obligatorio para la implementación de TuneIF

```
/* Constructor de la clase que implementa TuneIF                          */
/* @param   -una cadena de caracteres con el título de la canción         */
/*          -una cadena de caracteres con el autor de la canción          */
/*          -una cadena de caracteres con el género de la canción         */
/*          -una cadena de caracteres con el álbum al que pertenece       */
/*          -un entero con el año de publicación de la canción            */
/*          -un entero con la duración en segundos de la canción          */
/* @pre     -t != “” && a != “” && g != “” && al != “” && y > 0 && d > 0   */
Tune(String t, String a, String g, String al, int y, int d){}
```

4.2 Ejecución del programa.

En esta sección vamos a describir cómo ha de ser el funcionamiento del programa. En primer lugar detallaremos los parámetros de entrada que el programa recibe por línea de comandos y luego se indicará la estructura de los ficheros de datos y operaciones.

Dado que la lectura de parámetros de entrada y la carga de datos desde ficheros no son objeto de estudio en esta asignatura, todo esto se dará programado por parte del Equipo Docente.

Parámetros de entrada.

El programa recibirá tres parámetros de entrada que determinarán su comportamiento. El orden y significado de los parámetros será el siguiente:

1. Fichero de datos. Consistirá en una cadena con el nombre del fichero de datos con el que se construirá el repositorio de canciones.
2. Fichero de operaciones. Consistirá en una cadena con el nombre del fichero de operaciones que se desean realizar sobre el reproductor.
3. Tamaño máximo de RecentlyPlayed. Consistirá en un entero que representará el número máximo de canciones recientemente reproducidas que se deberá almacenar.

Estructura del fichero de datos.

El fichero de datos contendrá todos los datos necesarios para construir el repositorio de canciones. Está codificado en **UTF-8** en forma de fichero **TSV** (Tab-Separated Values), esto permite que pueda ser cargado en una hoja de cálculo (como Excel, Libre Office o similares) para su visualización.

La primera línea del fichero contiene los nombres (en inglés) de los atributos que se almacenan para una canción en el siguiente orden: Título, Autor, Género, Álbum, Año y Duración en segundos. De esta forma, si cargamos el fichero en una hoja de cálculo se podrá operar con ellos más fácilmente.

Cada una de las demás líneas del fichero contiene los valores de los atributos de una canción que se cargará en el repositorio (en el mismo orden que el indicado en la primera línea del fichero.

Estructura del fichero de operaciones.

El fichero de operaciones contiene una única operación por línea. Estas operaciones se corresponden con las operaciones públicas del interfaz `PlayerIF`. Por tanto, se realizarán operaciones de modificación del estado del reproductor (creación, modificación y borrado de listas de reproducción, modificación del estado de la cola de reproducción, reproducción de canciones) y operaciones que permitan consultar dicho estado tras las modificaciones realizadas (identificadores de las listas de reproducción existentes, contenido de alguna lista de reproducción, contenido de la cola de reproducción y contenido de las últimas canciones reproducidas).

Este fichero también estará codificado en **UTF-8** y será un fichero **TSV** (al igual que el fichero de datos). El primer campo se corresponde con la operación a realizar, que se identificará mediante una cadena de caracteres consistente en el nombre del método al que se desea llamar. Si la operación necesita de parámetros, éstos se situarán en campos sucesivos en el orden en el que el método los espera.

Salida del programa.

La salida del programa se realizará por la salida estándar de Java y consistirá en:

- La primera línea indicará cuántas canciones se han cargado en el repositorio.
- Para cada operación presente en el fichero de operaciones se volcará su llamada.
- Si la operación realizada es una de las cuatro operaciones de consulta, en la línea siguiente se volcará el resultado de la misma en forma de secuencia de valores separados por comas.

Ejecución y juegos de prueba.

Para la ejecución del programa se deberá abrir una consola y ejecutar:

```
java -jar eped2017.jar <datos> <operaciones> <tamaño>
```

- `<datos>` fichero de datos para construir el repositorio de canciones.
- `<operaciones>` fichero de operaciones con operaciones del reproductor.
- `<tamaño>` número máximo de canciones recientemente reproducidas que se pueden almacenar

El Equipo Docente proporcionará, a través del curso virtual, juegos de prueba consistentes en un fichero de datos, un fichero de operaciones, un tamaño máximo y una salida esperada para comprobar el correcto funcionamiento del programa.

El fichero de datos que proporciona el Equipo Docente ha sido generado automáticamente seleccionando al azar discos contenidos en la base de datos de discos de audio **FreeDB** (versión libre de **CDDB** surgida como respuesta a la transformación en comercial de esta). El resultado de esta selección aleatoria ha sido posteriormente limpiado a mano para eliminar posibles errores (principalmente debidos a la codificación de los ficheros de la **FreeDB**).

El fichero de operaciones consistirá en operaciones de modificación del estado del reproductor entre las que se intercalarán operaciones de consulta de dicho estado. Para superar el juego de pruebas será necesario que el resultado de las operaciones de consulta coincida con el esperado. **Si el juego de pruebas no se supera, la práctica se considerará suspensa.**

Junto con los juegos de prueba se entregará a los estudiantes la documentación necesaria para usarlos.

5. Preguntas teóricas

En este apartado se proponen algunas preguntas teóricas sobre aspectos de la práctica.

1. **Representación de los TAD** (*2 puntos*): para cada uno de los TAD que deben ser implementados, justificar la elección de la Estructura de Datos escogida para representar los elementos del TAD y comentar, al menos, otra posible Estructura (de las estudiadas en la asignatura) que pudiera usarse indicando los pros y los contras de dicha elección.
2. **Estudio teórico del coste** (*2 puntos*): para cada uno de los métodos del TAD `PlayerIF`, justificar:
   a) Tamaño del problema.
   b) Coste asintótico temporal en el caso peor en relación al tamaño del problema.
3. **Supuestos adicionales**: ante los siguientes supuestos adicionales se deberá justificar la respuesta a las preguntas propuestas:
   a) (*1 punto*) Supongamos que se añade una operación en `PlayListIF` que permita insertar una lista de identificadores de canciones en una posición determinada de la lista de reproducción. ¿Es necesario modificar la elección de la Estructura de Datos escogida para representar los elementos de este TAD?
   b) (*1 punto*) Supongamos que se añade una operación en `PlayBackQueueIF` que permita eliminar todas las apariciones de una canción (representada mediante su identificador) de la cola de reproducción. ¿Sería aconsejable cambiar la Estructura de Datos empleada para representar los elementos de este TAD?

6. Documentación, evaluación y plazos de entrega.

La práctica supone un 20% de la calificación de la asignatura, y es necesario aprobarla (5 puntos) para superar la asignatura. Además será necesario obtener, al menos, un 4 sobre 10 en el examen presencial para que la calificación de la práctica sea tenida en cuenta de cara a la calificación final de la asignatura.

Los estudiantes deberán asistir a una sesión obligatoria de prácticas con su tutor. Estas sesiones son organizadas por los Centros Asociados teniendo en cuenta sus recursos y el número de estudiantes matriculados, por lo que en cada Centro las fechas serán diferentes. Los estudiantes deberán, por tanto, dirigirse a su tutor para conocer las fechas de celebración de estas sesiones.

De igual modo, el plazo y forma de entrega son establecidos por los tutores de forma independiente en cada Centro Asociado, por lo que deberán ser consultados también con ellos.

La documentación que debe entregar cada estudiante a su tutor consiste en:

- Implementación en Java de la práctica, de la cual se deberá aportar tanto el código fuente como el programa compilado. Este programa deberá superar el juego de pruebas para que la práctica se considere aprobada.
- Memoria de la práctica, en la que se responderá a las cuestiones teóricas propuestas en el apartado anterior.

Adicionalmente, se deberá entregar una copia de esta documentación al Equipo Docente a través del curso virtual. Esta entrega servirá para que el Equipo Docente tenga acceso a todas las prácticas en caso de necesidad, pero no se enviará a los tutores para su corrección. Se avisará de los plazos de entrega de esta copia de la práctica a través del Tablón de Anuncios del curso virtual.

La forma de evaluar la práctica será:

1. Ejecución de la práctica sobre un juego de pruebas privado (sólo accesible por los tutores):
   a) Si la práctica no supera el juego de pruebas → calificación de la práctica: 0 (suspenso).
   b) Si la práctica supera el juego de pruebas → 4 puntos y se evalúa la memoria.
2. Evaluación de la memoria: se evaluará la respuesta a las preguntas teóricas propuestas en el apartado anterior atendiendo a la puntuación allí indicada.