Práctica de creación de una Playlist en Java (I)

Ingeniería de Sistemas Telemáticos.
Grado en Ing. en Tecnologías de la Telecomunicación, 2016-2017.
Escuela Técnica Superior de Ingeniería de Telecomunicación.
Universidad Rey Juan Carlos.

Fecha límite de entrega: **00:00 Lunes 27 de febrero, 2017**.

1. Objetivo de la práctica

El objetivo de la práctica es adquirir mayor destreza en la implementación de clases Java para POO. En particular:

- La definición de miembros de una clase (atributos y métodos), eligiendo adecuadamente los modificadores de acceso apropiados en cada caso.
- La creación de métodos constructores de clase.
- Sobreescribir y sobrecargar métodos en Java.
- Composición y herencia.
- Cómo sobreescribir algunos métodos importantes para garantizar el correcto funcionamiento de nuestras clases en Java: `toString()`, `equals()` y `hashCode()`.
- Probar mínimamente nuestro código dentro del método `main()` de la clase principal de nuestro paquete.

La temática de la práctica se centra en la creación de una implementación inicial y muy sencilla de una clase para modelar el comportamiento de una *playlist*, una lista de reproducción de canciones. Para ello, se proporciona una versión inicial de la clase `Cancion`, ya implementada, así como un esqueleto de una clase intermedia `Album`, que agrupa canciones, parcialmente implementada.

1.1. La clase `Cancion`

La clase `Cancion` se proporciona ya implementada, aunque **no se garantiza** que la implementación ofrecida en algunos de los métodos sea formalmente correcta, sino únicamente válida a efectos de proporcionar la funcionalidad deseada.

A medida que trabajemos en diferentes versiones de la práctica se comprenderá mejor por qué algunas funciones deben implementarse más correctamente de otra manera en Java, conforme vayamos introduciendo nuevos conceptos y convenciones de programación en este lenguaje.

Concretamente, esta versión de la clase `Cancion` proporciona:

- Atributos para almacenar el título, autor, duración y formato de codificacion de la canción. Este último se implementa mediante un tipo enumerado de datos que, al ser `public`, puede ser accedido por los usuarios de esta clase.
- Dos versiones del método constructor de la clase: uno por defecto y otro con argumentos de inicialización para todos los atributos.
- Métodos de acceso (*get...()*) y modificación (*set...()*) de los atributos.

- Un método `toString()`, que sobreescribe al de la clase `Object`, y proporciona una representación del contenido de la canción en formato `String`.

```
@Override
public String toString() {
        /**
          * Creación de una representación del contenido de la
          * Canción en formato String
          */
        int hours = duracion / 3600;
        int minutes = (duracion % 3600) / 60;
        int seconds = duracion % 60;

        String timeString;
        if (hours > 0) {
                timeString = String.format("%02d:%02d:%02d", hours, minutes, seconds);
        } else {
                timeString = String.format("%02d:%02d", minutes, seconds);
        }
        return String.join("\n", "----------",
                        "Título: " + titulo,
                        "Autor: " + autor,
                        "Duración: " + timeString,
                        "Formato: " + (formato == Codecs.MP3 ? "MP3" : "FLAC"),
                        "----------");
}
```

- Una <u>versión inicial</u> del método `equals()`, que sobreescribe al de la clase `Object`, y permite comparar dos canciones por su contenido.

```
@Override
public boolean equals(Object other) {
    /**
      * Implementación de un método de comparación del contenido
      * de dos canciones
      */
    if (other == null) return false;
    if (other == this) return true;
    if (!(other instanceof Cancion)) return false;

    Cancion otherCancion = (Cancion)other;
    if (this.titulo == otherCancion.titulo &&
                this.autor == otherCancion.autor &&
                this.duracion == otherCancion.duracion &&
                this.formato == otherCancion.formato) {
        return true;
    } else {
        return false;
    }
}
```

- Un método `hashCode()`, que sobreescribe al de la clase `Object`, y calcula un *hash* del contenido del objeto `Cancion` para comparación rápida entre dos objetos (será de utilidad al implementar una *hash table*, con la clase `HashTable`, que indexe objetos de este tipo).

```
@Override
public int hashCode(){
        /**
          * Desde Java 7, es obligatorio sobreescribir el método hashCode
          * para computar el código hash de los contenidos del objeto
          * Nota: el hashCode es el código que devuelve el método toString() de
          * la clase Object: NombreClaseObjeto<hashCodeObjeto>.
          * En el caso de Object, el código hash devuelto corresponde a la
          * dirección de memoria del objeto:
          * http://docs.oracle.com/javase/tutorial/java/IandI/objectclass.html
          */
        return Objects.hash(titulo, autor, duracion, formato);
}
```

- Un método `main()` para comprobar rápidamente la funcionalidad de los objetos de tipo `Cancion`).

1.2. La clase `Album`

La clase `Album`, utiliza la clase `Cancion` para crear objetos que puedan simular la funcionalidad de un álbum que incluye múltiples canciones. Esta clase está incompleta y hay que implementar las partes que faltan para completar su funcionalidad.

Los miembros incluidos y sugeridos son:

- Atributos para almacenar el título, autor, grupo, duración total y la lista de canciones del álbum.
- Un contructor por defecto (**a completar**) y un constructor con argumentos de inicialización ya implementado.
- Marca de tarea (**TODO**) **para implementar** los métodos de **asignación** y **actualización** de valores para los cuatro primeros atributos: título, autor, grupo y duración total.
- Una implementación del método `getTrackList()`, que devuelve la lista de canciones del álbum, en formato `ArrayList<Cancion>`.
- Una implementación del método `getTrack(int posicion)` que devuelve la canción que esté en la posición que se indique del álbum.
- Dos métodos `addTrack(...)` que añaden una canción al final de la lista de canciones (primer método) o en la posición que se indica como primer argumento (segundo método). **No están implementados**.
- Una implementación del método `deleteLastTrack()`, que elimina la última canción de la lista de canciones.
- Un método `deleteTrack(int posicion)`, que borra la canción en la posición indicada de la lista de canciones del `Album`. **No está implementado**.
- Un método `clearAlbum()`, que borra todas las canciones del `Album`. **No está implementado**.

Además, para completar la clase `Album` totalmente, se deberían proporcionar otros tres métodos básicos:

- Un método `toString()`, para representar en formato `String` el contenido del álbum (título, autor, grupo, duración total y datos de cada una de las canciones en la lista del álbum).
- Un método `equals()`, para comparar por su contenido dos objetos de tipo `Album`.
- Un método `hashCode()`, para comparar por el *hash* de su contenido dos objetos de tipo `Album` (análogamente a como se ha realizado en el método `hashCode()` de la clase `Cancion`).

1.3. La clase `Playlist`

La clase `Playlist` no está implementada en absoluto, por lo que se tendrá que crear para proporcionar una lista de objetos de tipo `Album` que contiene cada uno una lista de canciones (objetos de tipo `Cancion`).

Los atributos de esta clase serán:

- `String nombre`: nombre de la `Playlist`.
- `int numAlbumes`: número de álbumes incluidos en la `PlayList`.
- `int numCanciones`: número total de canciones en la `Playlist` (suma del número de canciones en cada `Album` incluido en la `Playlist`.

- `int duracion total`: duración total en segundos de las canciones en la `Playlist`.
- `ArrayList<Album>albumList`: lista con todos los álbumes incluidos en la `Playlist`.

La funcionalidad básica inicial que se pide implementar es:

- Creación de objetos de tipo `Playlist` mediante un **constructor por defecto** y **otro constuctor** que reciba argumentos para la inicializació de los valores de todos los atributos del objeto.
- Métodos de acceso y actualización de los valores de los cuatro primeros atributos de los objetos de tipo `Playlist`.
- Un método `addAlbum(Album unAlbum)`, que añada un `Album` al final de la `Playlist`; otro método `addAlbum(int posicion, unAlbum)`, que añada un `Album` en la posición de la lista de álbumes de la `Playlist` que indique el primer argumento.
- Un método `removeAlbum()`, que elimine el `Album` en la última posición de la `Playlist`; otro método `removeAlbum(int posicion)`, que elimine el `Album` que esté en la posición de la lista de álbumes de la `Playlist` que indique el argumento.
- Un método `clear()`, que borre todos los álbumes de la lista de álbumes de la `Playlist`.

Adicionalmente, se puede intentar implementar los siguientes métodos que serán necesarios más adelante:

- Un método `toString()`, para representar en formato `String` el contenido de la `Playlist` (nombre, número de álbumes, número total de canciones, duración total y datos de cada uno de los álbumes en la `Playlist`).
- Un método `equals()`, para comparar por su contenido dos objetos de tipo `Playlist`.
- Un método `hashCode()`, para comparar por el *hash* de su contenido dos objetos de tipo `Playlist` (análogamente a como se ha realizado en el método `hashCode()` de la clase `Cancion`).

2. Enunciado de la práctica

Calificación máxima: **10 puntos**.

Se pide completar las clases `Album` y `Playlist` para conseguir las funcionalidades indicadas anteriormente, y que se resumen a continuación:

Clase `Album`

Funcionalidad mínima (*3 puntos*).

- Constructor por defecto.
- Métodos de acceso y actualización de los cuatro primeros atributos.
- Dos métodos `addTrack(...)` para añadir canciones al final del `Album` o en la posición indicada, respectivamente.
- Un método `deleteTrack(int posicion)` para borrar la `Cancion` en esa posición de la lista del `Album`.
- Un método `clearAlbum()` para borrar todas las canciones del `Album`.

Funcionalidad adicional (*2 puntos*).

- Método `toString()`.
- Método `equals()`.
- Método `hashCode()`.

Clase `Playlist`

Funcionalidad mínima (*3 puntos*).

Atributos de objetos de esta clase:

- `String nombre`.
- `int numAlbumes`.
- `int numCanciones`.
- `int duracion total`.
- `ArrayList<Album>albumList`.

Métodos a implementar:

- Constructor por defecto y otro con argumentos de inicialización.
- Métodos de acceso y actualización de los cuatro primeros atributos.
- Dos métodos `addAlbum(...)` para añadir álbumes al final de la `PlayList` o en la posición indicada, respectivamente.
- Dos métodos `removeAlbum(...)` para borrar el `Album` al final de la `PlayList` o en la posición indicada, respectivamente.
- Un método `clear()` para borrar todas las canciones del `Album`.

Funcionalidad adicional (*2 puntos*).

Adicionalmente, se puede intentar implementar los métodos:

- Método `toString()`.
- Método `equals()`.
- Método `hashCode()`.

Validación con `main()`

La práctica incluirá la validación de la funcionalidad de las clases anteriormente descritas mediante la creación de objetos de dichas clases en el método `main()` (dentro de cada clase, o bien aparte), de forma que se llame a los diferentes métodos implementados en cada clase a través de esos objetos.