# Práctica 1

## MEMORIA

*Programació II*

**Eduardo Wass Rosado**
**NIUB 16447981**
*Universitat de Barcelona*

# Índice

# Introducción

## Descripción de la práctica

Se tiene que hacer un programa que organice nuestra biblioteca musical.

El programa se debe hacer en Java y orientado a objetos

El modelo de datos se compone de los archivos que corresponden a canciones, cada canción guardará (además de la ruta al archivo) una serie de atributos:

- Nombre
- Autor
- Disco
- Año
- Número de canción
- Discográfica
- Duración

Cada uno de estos archivos se organizará dentro de una lista. Para interactuar con esta lista dispondremos de un menú con las siguientes funciones:

- Añadir a lista
- Mostrar lista
- Eliminar archivo
- Guardar lista
- Cargar lista

En la **primera parte** de la práctica deberemos implementar todo nosotros mismos sin el uso de clases externas, la **segunda parte** consiste en adaptar el programa introduciendo la clase ArrayList nativa de Java.

## Objetivos

- Definiremos nuestras propias clases para el programa: **FitxerAudio** y **TaulaFitxers**
- La clase **FitxerAudio** heredará de la clase **File** de Java y además añadirá sus propios atributos y métodos
- La clase **TaulaFitxers** se debe implementar sin uso de librerías externas, tan solo con los conocimientos vistos en la asignatura de PRO1
- Se debe seguir el patrón de diseño **MVC**, aunque para esta primera práctica solo utilizamos: Modelo y Vista
- El código debe seguir los **patrones de estilo** dados por la guía proporcionada por SUN
- El código debe tener las secciones de comentarios pertinentes para la correcta generación de la documentación utilizando **JavaDoc**

# Análisis

## Previsión y organización

Para la correcta organización del trabajo es importante tener claro que la práctica se divide en dos objetivos, el primero de los cuales es el diseño e implementación del programa sin clases externas para nuestra estructura de datos, y el segundo la adaptación del mismo ayudándonos esta vez sí de clases específicas para el control de listas.

Así pues, centrándose en el primer objetivo, deberemos tener muy clara la estructura de datos a utilizar, por lo tanto es importante construir un primer esqueleto de las clases con todos sus atributos/métodos y tener claro que es lo que queremos que haga cada cosa.

## Guión de trabajo

- Generar la estructura/esqueleto de las clases:
    - **ReproductorUB1A (clase vista)**
        - **Atributos:**
            - objeto TaulaFitxers
        - **Funciones:**
            - main()
            - menu()
                - **funciones de cada una de las opciones**
    - **ReproductorUB1B (clase vista a rellenar más adelante)**
    - **FitxerAudio** (clase hereditaria de class File de java)
        - **Atributos:**
            - nombre cancion, autor, disco, etc.
        - **Funciones:**
            - public demanaDadesTeclat();
            - public boolean equals(Object fitxerAudio)
            - toString() -- output del objeto, mirar enunciado
            - equals() -- comprobar igualdad entre ficheros

- **TaulaFitxers** (lista compuesta de varios elementos de FitxerAudio)
  - **Atributos:**
    - private FitxerAudio[] taula;
    - private int tamany;
  - **Funciones:**
    - public int tamany(); (podemos suponer size=100)
    - public void afegirFitxer(FitxerAudio fitxer) ;
    - public void eliminarFitxer(FitxerAudio fixter) ;
    - public FitxerAudio getAt(int position) ;
    - públic void clear();
    - protected boolean isFull();
    - toString() -- output del objeto, mirar enunciado
    - equals() -- este si que probablemente haya que redefinirlo

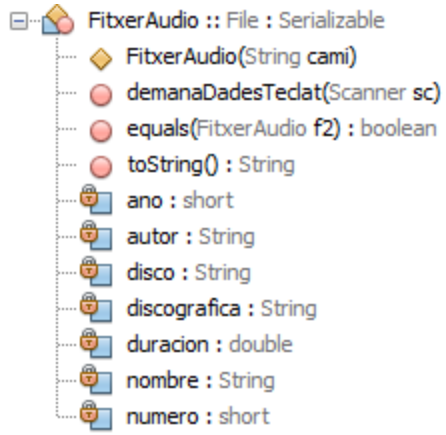
- Crear JavaDoc para los métodos y tener claro que hace cada funcion
- Implementar los métodos y probar las clases descritas en el punto anterior
- Implementación de la parte del menú (opciones de 1 a 3) de la aplicación. Permiten elegir entre las opciones e introducir la información por teclado
- Implementación de menú (opciones 4 y 5) guardar y cargar de un fichero (Streams)
- Implementar **ReproductorUB1B** donde reemplazaremos la clase TaulaFitxers por ArrayList

# Desarrollo

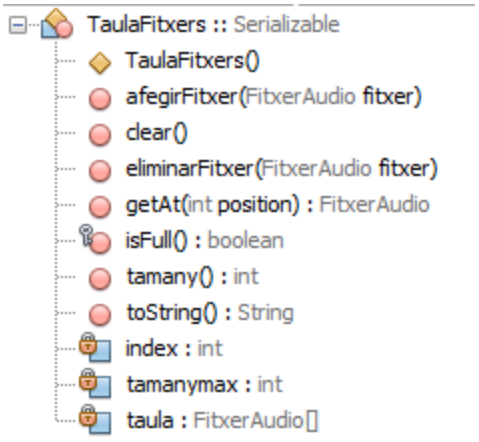
## Métodos y funciones

Finalmente, tras la implementación la estructura del programa ha quedado de la siguiente manera:

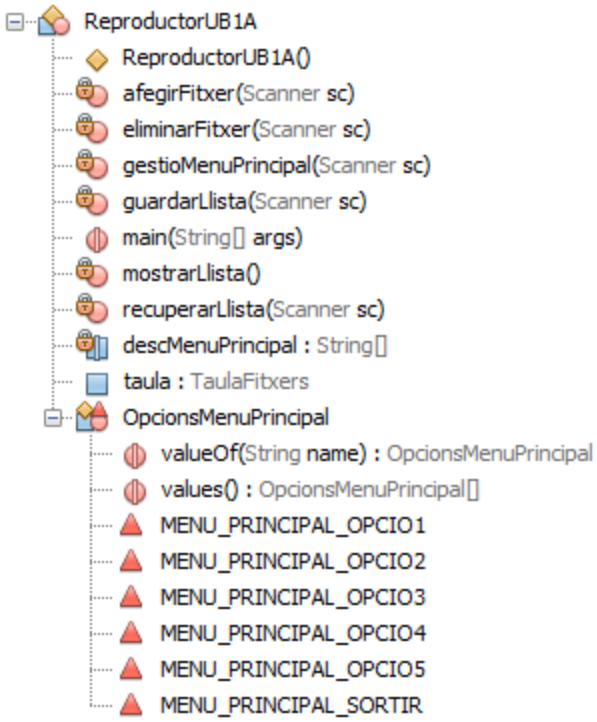
### FitxerAudio



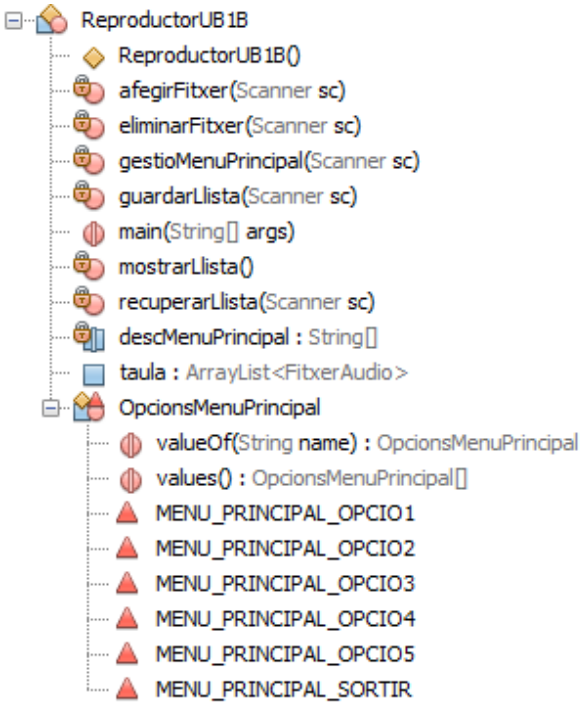

### TaulaFitxers




### ReproductorUB1A




### ReproductorUB1B



## Implementación del Objetivo B

Para la segunda parte de la práctica se nos pedía sustituir nuestra clase **TaulaFitxers** por la clase de Java **ArrayList** y observar las diferencias.

A parte de la declaración (ArrayList <FitxerAudio> taula), la diferencia más evidente y que se aprecia enseguida es que nos ahorramos gran cantidad de trabajo de implementación de las funciones de operación de la lista, ya que para la clase ArrayList ya están implementadas.

Así pues, podemos reemplazar nuestras funciones: **tamany(), afegirFitxer(), eliminarFitxer(), getAt()**... por **size(), add(), remove(), get()** … que ya están implementadas y conservar la misma funcionalidad, sin tener que ponernos a picar el código de lo que hace cada una.

## Problemas y soluciones

A continuación explicaré para mi caso particular, los problemas con los que me encontré y como decidí solucionarlos:

- **Incorrecta definición de estructura/esqueleto del proyecto y sus clases métodos:**
  El problema fue causado por un mal entendimiento del enunciado, por suerte me dí cuenta a tiempo pudiendo redefinir la estructura del proyecto antes de ponerme a implementar nada.
- **Tipos de datos incorrectos:**
  A mitad del proyecto decidí cambiar el tipo de datos de alguno de los atributos ya que durante alguna fase de prueba me dí cuenta de que no eran los adecuados y esto me obligó a refactorizar algún atributo.
- **Función eliminarFitxer():**
  Creo que esta fue la función que más me dió que pensar, ya que cada vez que eliminaba un fichero el array quedaba descolocado y había que reordenarlo asegurándose que no nos dejábamos residuos en el camino. Al final opté por shiftar los elementos a la derecha del eliminado y revisar que no quedarán datos incongruentes después de la operación.

# Resultados

## Pruebas realizadas

Para el testeo de la aplicación me hice un archivo ejemplo de input para poder introducir un montón de datos de archivos sin tener que hacerlo a mano cada vez.

Posteriormente probé cada una de las funciones repetidas veces y probando casos extremos hasta cerciorarme de que el funcionamiento era el correcto.

## Ejemplos de Output

### Reproductor UB1A:

[0] | Titol: nom | Autor: autor | Disc: disc | Pista: 1 | Discogràfica: discografica | Duració: 1.1 | Any: 1 | Fitxer: <ruta> |
[1] | Titol: nom2 | Autor: autor2 | Disc: disc2 | Pista: 12 | Discogràfica: discografica2 | Duració: 1.2 | Any: 12 | Fitxer: <ruta2> |
[2] | Titol: nom3 | Autor: autor3 | Disc: disc3 | Pista: 13 | Discogràfica: discografica3 | Duració: 1.3 | Any: 13 | Fitxer: <ruta3> |
[3] | Titol: nom4 | Autor: autor4 | Disc: disc4 | Pista: 14 | Discogràfica: discografica4 | Duració: 1.4 | Any: 14 | Fitxer: <ruta4> |
[4] | Titol: nom5 | Autor: autor5 | Disc: disc5 | Pista: 15 | Discogràfica: discografica5 | Duració: 1.5 | Any: 15 | Fitxer: <ruta5> |
[5] | Titol: nom6 | Autor: autor6 | Disc: disc6 | Pista: 16 | Discogràfica: discografica6 | Duració: 1.6 | Any: 16 | Fitxer: <ruta6> |

### ReproductorUB1B:

[0] | Titol: nom | Autor: autor | Disc: disc | Pista: 1 | Discogràfica: discografica | Duració: 1.1 | Any: 1 | Fitxer: <ruta> |
[1] | Titol: nom2 | Autor: autor2 | Disc: disc2 | Pista: 12 | Discogràfica: discografica2 | Duració: 1.2 | Any: 12 | Fitxer: <ruta2> |
[2] | Titol: nom3 | Autor: autor3 | Disc: disc3 | Pista: 13 | Discogràfica: discografica3 | Duració: 1.3 | Any: 13 | Fitxer: <ruta3> |
[3] | Titol: nom4 | Autor: autor4 | Disc: disc4 | Pista: 14 | Discogràfica: discografica4 | Duració: 1.4 | Any: 14 | Fitxer: <ruta4> |
[4] | Titol: nom5 | Autor: autor5 | Disc: disc5 | Pista: 15 | Discogràfica: discografica5 | Duració: 1.5 | Any: 15 | Fitxer: <ruta5> |
[5] | Titol: nom6 | Autor: autor6 | Disc: disc6 | Pista: 16 | Discogràfica: discografica6 | Duració: 1.6 | Any: 16 | Fitxer: <ruta6> |

Y la implementación de cada uno de ellos:

**Reproductor UB1A:**

```
for(int i=0;i<this.tamany();i++){
    retorn += "\n["+i+"] | ";
    retorn += this.taula[i];
}
return retorn;
```

**ReproductorUB1B:**

```
int i = 0;
String retorn = "";
for (FitxerAudio f : this.taula){
    retorn += "\n["+i+"] | ";
    retorn += f;
    i++;
}
System.out.println(retorn);
```

El primero está implementado dentro de la clase, mientras que el segundo lo he hecho dentro de la misma función que se llama desde el menú, el output es el mismo ya que el output natural de la clase ArrayList es muy similar.

## Responde a pregunta:

> *Respondre a la següent pregunta: segons la implementació de la part B, si tenim dos fitxers d'àudio corresponents al mateix fitxer, quan cridem al mètode per eliminar un d'aquests fitxers eliminarà el altre també o no?*

La respuesta es no, ya que nosotros le estamos pasando una posición del Array a eliminar, por lo tanto el método se limita a eliminar cualquier fichero que esté en esa posición independientemente de otros factores.

## Conclusiones

Creo que la conclusión importante de esta práctica es que (suponiendo que el rendimiento no sea un factor súper determinante para nuestra aplicación) siempre que podamos reaprovechar librerías que ya estén creadas debemos hacerlo. No tiene sentido reinventar la rueda si disponemos de código que han hecho otras personas y el mismo puede agilizar nuestro desarrollo.

También que es importante entender bien el concepto de clases/objetos y utilizar sus métodos y atributos de manera correcta, ya que esto nos ayudará a crear un código limpio, entendible y reutilizable, y realizar modificaciones como cambiar la estructura de datos principal (como el caso de esta práctica) no se convierta en una tarea horrible.