
# JMusicHub

Pour le 31 décembre 2020.



BELHADJ Saïd

ESIEA UFA 31

# I - Introduction.

Ce projet a été réalisé **seul et en totale autonomie** il contient 15 classes (enums, classe abstraite, interface, classe d'exception). Mon application présente les différentes thématiques suivantes :

- ✔️ Héritage.
- ✔️ Sérialisation.
- ✔️ Gestion des exceptions.
- ✔️ Les collections.
- ✔️ Classe abstraite et utilisation d'interface.
- ✔️ Les fichiers contiennent au moins 20 éléments, 3 albums et 2 playlists.

L'application **JMusicHub** possède les fonctionnalités suivantes :

- **Chanson**
  - Créer des chansons.
  - Afficher des chansons.
  - Supprimer des chansons.
  - Sauvegarder les chansons dans un fichier XML.
- **Livre audio**
  - Créer des livres audios
  - Afficher des livres audio rangés par auteur.
  - Sauvegarder les livres audios dans un fichier XML.
- **Album**
  - Créer des albums.
  - Afficher les albums rangés par date de sortie.
  - Afficher les chansons d'album rangées par genre.
  - Ajouter des chansons.
  - Supprimer des chansons.
  - Sauvegarder les albums dans un fichier XML.
- **Playlist**
  - Créer des playlists.
  - Afficher les playlists rangées par noms.
  - Ajouter des chansons et des livres audios.
  - Supprimer des chansons et des livres audios.
  - Sauvegarder les playlists dans un fichier XML.

# II - Explications des différents choix de conceptions utilisés au sein de l'application JMusicHub.

## Main :

La classe Main permet de gérer la **boucle principale** de notre programme, c'est aussi la classe qui va être **invoquée** par notre **compilateur** et afin de permettre aux utilisateurs de **gérer leurs éléments musicaux**.

## Command :

Cette classe a été créée en raison de **soucis de lisibilité** et afin de ne pas **surcharger** la classe Main. Cela m'a permis de débugger mon programme plus facilement. Cette classe contient toutes les fonctions qui interagissent avec l'utilisateur comme la demande de **saisie d'informations** par exemple.

## InputException :

C'est une classe d'exception qui **hérite** de classe JAVA **Exception** qui va permettre de lever des exceptions que l'utilisateur aura provoquées et de contrôler l'**intégralité** des saisis des utilisateurs. ***"Never trust user input" !***

## MusicalElement, Song, AudioBook :

Comme vous pouvez le deviner, MusicalElement est la **classe abstraite** qui conceptualise le comportement des classes **Song** et **AudioBook**, pourquoi une classe abstraite ? Premièrement afin d'éviter les répétitions de code comme vous pouvez le remarquer la classe Song et AudioBook présente beaucoup de similitude et abstraite car elle ne peut instancier un objet MusicalElement cela n'a pas de sens physique ou concret contrairement à une chanson et un livre audio.

## Category, Language, Genre:

Ici, j'ai choisi de représenter ces entités en tant que **Enum** car elles représentent une liste finie d'éléments contrairement aux chansons et livres audio.

### Album, Playlist :

Les albums contiennent exclusivement des chansons et les playlists contiennent des chansons et des livres audios.

### ListPlaylist, ListMusicalElement, ListAlbum:

Ces classes contiennent respectivement la liste totale des playlists, éléments musicaux et albums

### Serialization :

Cette classe me permet de **sérialiser** et de **désérialiser** les éléments suivants :

- Éléments musicaux (Chansons et Livres audio)
- Playlists
- Albums

en fichier *XML*.

## III - Les difficultés rencontrées et les solutions qui leur ont été apportées.

### Difficulté 1 :

Je suis resté bloqué pendant un certain temps sur la **sérialisation** des objets, les objets que je tentais de sérialiser ne s'enregistraient pas dans mon fichier XML car une des particularités de la sérialisation en fichier XML est que les objets à sérialiser ne doivent pas avoir de paramètres obligatoires.

Source : https://www.jmdoudoux.fr/java/dej/chap-serialisation.htm

### Solution 1 :

Pour régler ce problème j'ai dû enlever les paramètres des constructeurs de mes classes Album, Playlist, Musicalement, Song et AudioBook et réorganiser ma manière de créer mes objets dans mon programme plus particulièrement dans ma classe Command lors de la saisie des utilisateurs.

## Difficulté 2 :

La sérialisation de nouveaux objets écrasait les anciens objets.

## Solution 2 :

Afin de pallier ce problème j'ai créé des classes que je nomme personnellement buffer, qui auront pour rôle d'enregistrer les anciens objets contenus dans mes fichiers XML à chaque début de programme et les sauvegarder en cas de demande de l'utilisateur et en parallèle d'ajouter chaque nouvel élément créé de type Album, MusicalElement ou Playlist. Les classes en question sont : ListMusicalElement, ListPlaylist, ListAlbum.

## Difficulté 3 :

Trier mes albums par date de sortie.

## Solution 3 :

Après de nombreuses recherches j'ai fini par trouver une solution propre et gérée nativement par JAVA, l'interface Comparable, qui me permet de comparer différents objets selon un attribut pour ce faire on implémente et override la fonction qui s'occupe de comparer ces objets (compareTo()) afin de comparer et trier nos objets de type Album selon la date de sortie.

Source : https://stackoverflow.com/questions/5927109/sort-objects-in-arraylist-by-date

# IV - Diagramme de classe UML.

*ici les identifiants ont volontairement été omis.