Année universitaire 2017-2018
Université de Caen Basse-Normandie

Rapport TPA (Lecteur de musique augmenté)

ZAIZAFOUN Sami et JACQUELINE Martin

L2 Informatique

Table des matières

A Choix du projet

Nous avons choisi ce projet car nous avons trouvé le concept de créer notre propre lecteur de musique intéressant. Durant la première séance de TPA, nous avons trouvé que l'utilisation du *fxml* était plus simple et efficace pour créer l'interface graphique de notre programme. Donc nous avons repartie les taches entre nous. L'interface a été prise en œuvre par une personne et l'autre personne a pris en œuvre la manipulation de la lecture des différents fichiers medias et leur propriétés.

B Introduction

Le projet que nous avons sélectionné est un lecteur de musique, nous nous sommes lancés dans l'optique de faire un programme avec une interface utilisateur extrêmement claire.

Nous souhaitions en premier lieu ajouter une recherche de parole, permettant à l'utilisateur d'avoir, dès que sont appareil est connecté à internet, un affichage simple et lisible des paroles. Le programme se devait donc de ne pas être à une taille réduite. Nous avons par la suite ajouter une possible playlist alternative contenant les "favoris" à savoir les musiques jouées plus de x fois. Nous voulions que l'application ne soit pas trop imposante, ne prenant qu'une petite partie de l'écran. Nous avons donc passé quelques temps de recherche dès le début de la réflexion à chercher la taille permettant d'allier de manière optimale une bonne visibilité et lecture pour l'utilisateur et, pour autant la plus simple et petite possible.

Nous allons ci-dessous aborder la conception en elle-même du programme.

C Cahier des charges

fonction : lecteur de musique
- objectif : créer la base du programme, le lecteur de musique basique et fonctionnel
- description : création des fonctions permettant la lecture d'une musique et d'une interface permettant de les utiliser
- contraintes : gérer l'interface pour qu'elle permette le futur ajout des paroles et des favoris
- priorité : haute

fonction : ajout des paroles
- objectif : récupérer les paroles et les afficher
- description : créer les fonctions permettant la récupération des paroles et son affichage
- contraintes : avoir une récupération rapide, lisible (retour a la ligne, taille text, etc...)
- priorité : moyenne

fonction : ajout des favoris
- objectif : compter le nombre de lecture pour chaque musique et permettre l'ajout d'une musique dans les favoris
- description : créer les fonctions permettant le comtage du nombre de lecture de chaque musique pour l'ajouter au bout de x lancement et permettre à l'utilisateur d'ajouter et supprimer des musiques au favoris
- contraintes : n/a
- priorité : moyenne

D Schéma UML

Le *fxml* est un langage de balisage d'interface utilisateur basé sur *xml* créé par Oracle Corporation pour définir l'interface utilisateur d'une application JavaFX.
En implémentant la méthode *initialize(URL location, ResourceBundle resources)* qui est appelé pour initialiser un contrôleur après que son élément racine a été complètement traité, nous pouvons manipuler toutes les fonctionnalités de notre programme et du *fxml*.

Pour manipuler un programme qui utilise du *fxml*, il faut un main qui ouvre le fichier *fxml* et une classe qui sert comme contrôleur, dans notre cas, la classe *HomeController*.

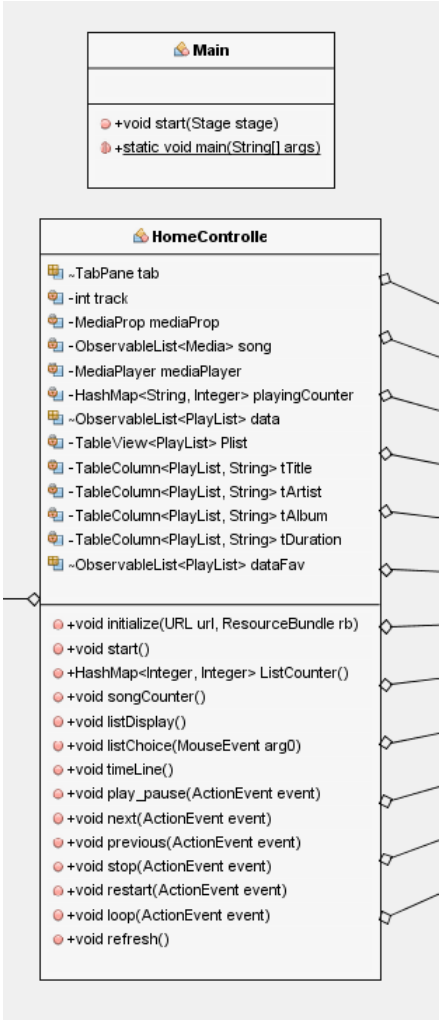


FIGURE 1 – Schéma UML des classes *Main* et *HomeController*

La classe *HomeController* fait appel à deux classes. La classe *MediaProp* et la classe *PlayList*. Ces deux classes seront expliquées dans la prochaine partie.

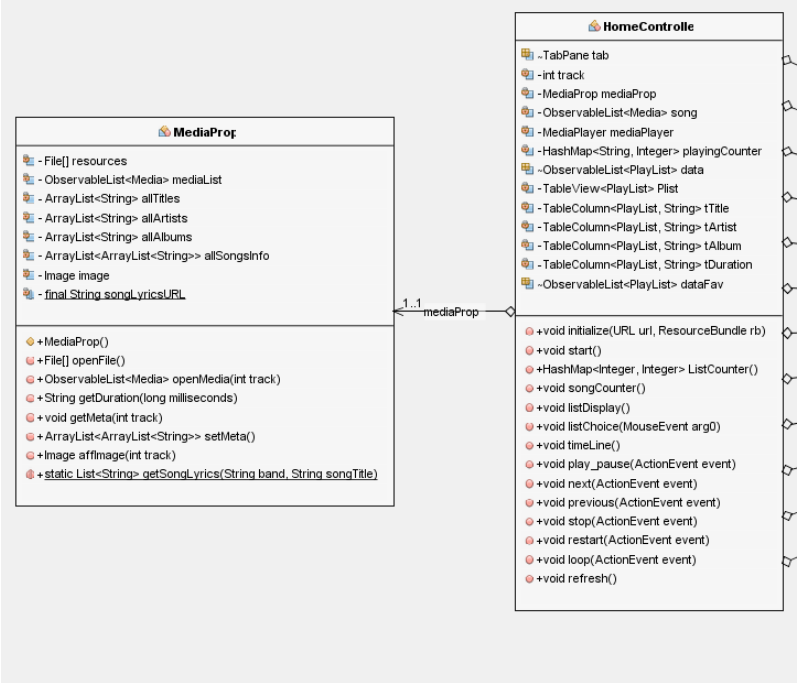


FIGURE 2 – Schéma UML des classes *HomeController* et *MediaProp*

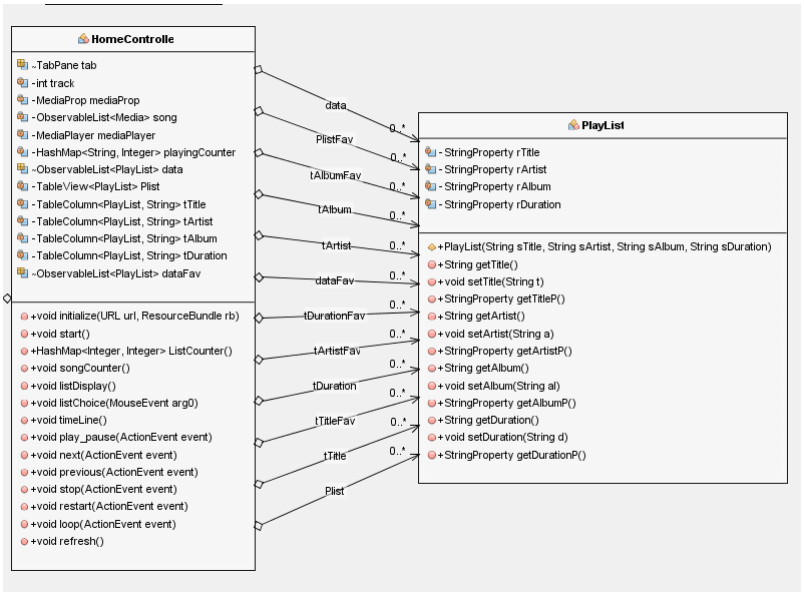


FIGURE 3 – Schéma UML des classes *HomeController* et *PlayList*

E Les classes

E.1 MediaProp

La classe *MediaProp* sert principalement à récupérer les musiques et leurs informations. Elle récupère donc en premier temps le dossier où les médias existent en retournant une liste appelée *resources* contenant tout les fichiers. Ensuite, elle récupère cette liste et elle transforme tous les fichiers en fichiers de type Media, les ajoute dans une liste puis les transforme en fichiers de type MediaPlayer, ceux-ci permettant de jouer une musique MP3.

Algorithm 1: ObservableList<Media> openMedia(int track)

Input: Integer track
Output: List of media files

```
1  mediaList ← []
2  resources ← [fichier musique]
3  fichier ← ""
4  for f in resources do
5  |   fichier ← resources[track]
6  |   resource ← path of fichier
7  |   media ← media(resource)
8  |   mediaPlayer ← MediaPlayer(media)
9  |   mediaList+ = [media, ]
10 |   track+ = 1
11 end
12 return mediaList
```

Ensuite, nous avons une méthode qui récupère la durée de chaque musique et l'enregistre dans une variable *totalTime* indépendante et variant de valeur pour chaque musique. Elle utilise alors une méthode asynchrone. Cette-dite méthode est asynchrone car elle utilise un *listener* qui détecte de nouvelles informations metadata dans une musique (les metadatas étant les informations de la musique comme l'artiste, le titre, l'album, etc..). Une fois qu'une nouvelle information est détectée, la méthode cherche une clé (titre, artiste, album) et récupère sa valeur, si elle existe, puis l'insère dans une liste spécifique. Ensuite, toutes les listes sont ajoutées dans une liste *allSongsInfo* qui est utilisée pour récupérer les informations quand besoin il y a. Cette liste est renvoyée dans la méthode *setMeta()*.

De même, nous devions récupérer une photo d'album. L'idée de départ fut de récupérer les photos depuis le web mais nous fûmes confrontés au problème de tailles d'image, celles-ci différaient, certaines étant alors peu correctement présentées, et ceux malgré l'utilisation d'une méthode servant à altérer la taille des images en question. Nous avons donc récupéré les images d'album nous intéressant et nous les avons déposés dans un dossier, la méthode vérifie ensuite si le fichier MediaPlayer allant être joué possède une photo d'album associée, dans le cas contraire elle charge alors une image neutre.

Pour ce qui est de la récupération des paroles, nous avons choisi d'utiliser le site web SongLyrics. Grâce à la librairie *Jsoup.jar*, nous avons réussi à nous connecter sur ce site en utilisant le nom de l'artiste (ou du groupe les cas échéant) et le nom de la musique en remplaçant tous les espaces par des "-" et les majuscules par des minuscules (l'url du site étant généré de cette manière). Dans le cas où le programme réussi à se connecter, il récupère alors le *div* contenant les paroles et l'ajoute à une liste appelée *lyrics*.

E.2 PlayList

La classe *PlayList* permet au programme de placer les valeurs titre, artiste, album et durée dans leurs colonnes respectives du *TableView* (une méthode *fxml* créant un tableau). Cette classe contient principalement des méthodes dites "getter" et des méthodes "setter" qui récupèrent les valeurs et les initialisent pour en avoir l'usage ailleurs.

E.3 HomeController

Nous allons maintenant aborder le cœur du programme, la classe *HomeController* ainsi que ses méthodes.

La classe *HomeController* est celle qui initialise le programme car tout ce qui est en rapport avec le *fxml* n'est définissable que dans cette classe. Pour bien comprendre cette classe, il faut que nous vous expliquions les fonctionnalités et les méthodes que nous avons choisi d'utiliser.

En premier temps, la méthode *listset()*. Cette méthode permet au *fxml* de lier les colonnes de *TableView* des musiques et des favoris avec les valeurs qui sont défini dans la classe *PlayList*. Une fois que ceci est fait, la méthode *listDisplay()* s'occupe de l'affichage de la playlist. Pour cela, nous avons créé trois *ArrayList* qui prenne les différentes listes créée dans la classe *MediaProp* pour récupérer les bons titres, artistes et albums. Pour rendre ce qui est dit plus compréhensible, voici le pseudo-code :

Algorithm 2: listDisplay()

```
1  meta ← mediaProp.setMeta() // mediaProp.setMeta() returns a list of lists
       that contain a media file's information
2  titleName ← meta.get(0) // gets the title list
3  artistName ← meta.get(1) // gets the artist list
4  albumName ← meta.get(2) // gets the album list
5  for i in range file.lengeth do
6  |  totalTime ← song.duration
7  |  ele = PlayList(title, artist, album, totalTime)) // calls the PlayList
   |     constructor while inserting the right information
8  |  data.add(ele) // adds the playlist into TableView
9  end
10 metaData() // Updates the displayed info
```

Pour les favoris nous avons dû créer deux méthode : *songCounter* et *listCounter*. La première est exécutée dès lors qu'une musique est lancée et vérifie si le titre de la musique qui vient d'être lancée est déjà présent dans la HashMap *playingCounter*, si tel est le cas, il augmentera de un l'entier associé à ce-dit titre, dans le cas contraire il initialisera la clé du titre actuel avec la valeur 1. La seconde méthode sert, elle, a faire une HashMap proche de celle que nous venions d'aborder, ceci étant que la clé n'est plus une String qui contient le titre de la chanson mais un entier, entier qui est la valeur associé a l'entier de *track* qui est le pointeur de *MediaPlayer* sur *File*. Cette fois ci, les musiques n'ayant pas encore été jouées voient leur valeur (associée à l'entier qui les représente) mise à 0, les autres sont au nombre de lancement incrémenté dans *songCounter*. Nous obtenons donc une HashMap qui contient, dans l'ordre des fichiers, le nombre de lancement de chaque musique, nous pouvons donc très simplement retrouver quelle musique ajouter aux favoris.

Pour rendre le programme plus facile à utiliser, nous avons créé une fonction qui permet aux utilisateurs de cliquer directement sur une musique dans la playlist pour la jouer. Cette fonctionnalité contient quelques paramètres. Un paramètre "double clique" qui permet aux utilisateurs de naviguer la playlist sans devoir commencer une musique, un paramètre qui empêche l'utilisateur d'avoir un choix nul et le paramètre qui récupère le nom de la musique et le transform en format Media pour le lancer. Ensuite, il met à jour les informations affichées et la photo de l'album.
Cette méthode est utilisée pour la playlist des favoris aussi.

Vient ensuite, la méthode *timeLine()*. Cette méthode a principalement trois fonctionnalités.
— Bouger la barre d'avancement avec la musique. Ceci est fait grâce à deux *listener*, un qui observe la valeur de la barre et l'autre qui observe le temps passé de la musique et le premier *listener* change la valeur du curseur par rapport à l'avancement de la musique.
— Initialiser un minuteur qui suit l'avancement de la musique.
— Un test boolean pour vérifier si l'utilisateur veut jouer la musique en boucle.

La classe contient aussi tout les boutons. Les fonctionnalités de ces boutons seront expliquées dans la partie Les boutons.

Une autre utilisation d'une barre d'avancement a été faite pour contrôler le volume dans la méthode *volume()*. Le principe est plus ou moins identique à celui de la méthode d'avant. Elle utilise un *listener* qui observe l'avancement du curseur et elle applique la valeur du curseur sur le volume.

F Manuel d'utilisation

F.1 Lancement

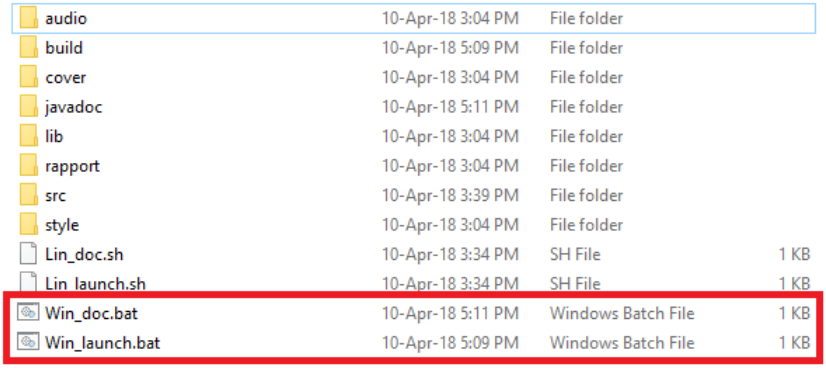
Le programme se lance préférablement sur *Windows* de deux manière :

1. Depuis un CMD en suivant les commandes :

```
javac -d build -cp "lib/jsoup.jar;." src/*.java
java -cp "build;lib/jsoup.jar;." src.Main

##JavaDoc
javadoc -charset utf8 -cp "lib/jsoup.jar" src/*.java -d javadoc
```

2. Depuis un *.bat* exécutable.



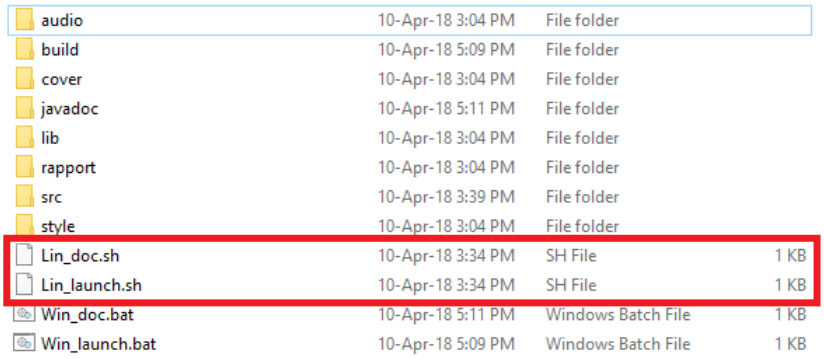
il se lance aussi sur *Linux* (sans autorisation de récupérer les paroles en ligne) de deux manière :

1. Depuis un terminal en suivant les commandes :

```
javac -d build -cp "lib/jsoup.jar:." src/*.java
java -cp "build:lib/jsoup.jar:." src.Main

##JavaDoc
javadoc -charset utf8 -cp "lib/jsoup.jar" src/*.java -d javadoc
```

2. Depuis un *.sh* exécutable.



En suivant les commandes :

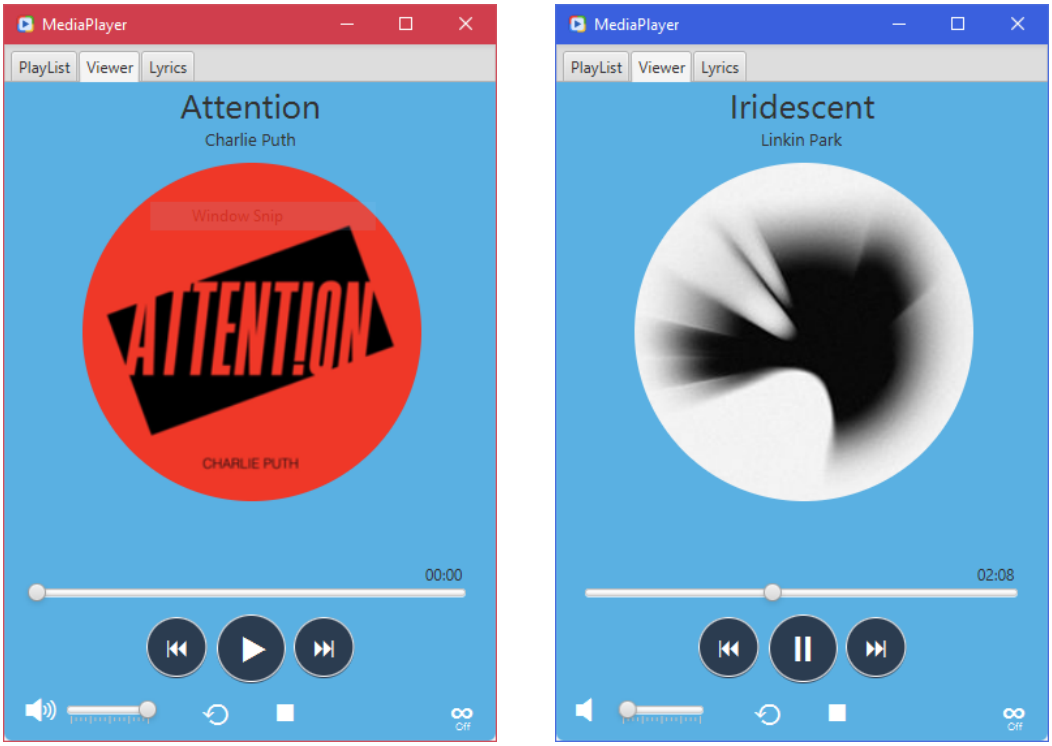
```
chmod +x Lin_launch.sh
sh Lin_launch.sh

## JavaDoc
chmod +x Lin_doc.sh
sh Lin_doc.sh
```

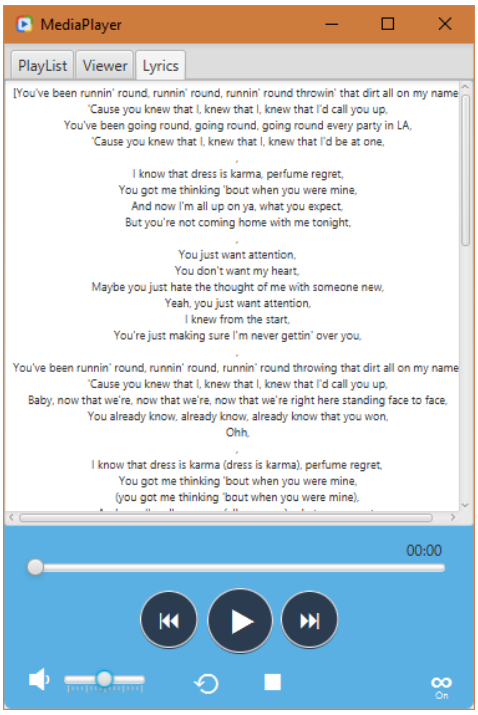
F.2 Utilisation

Nous avons fait en sorte que le programme soit facile à utiliser. L'interface contient trois onglets :

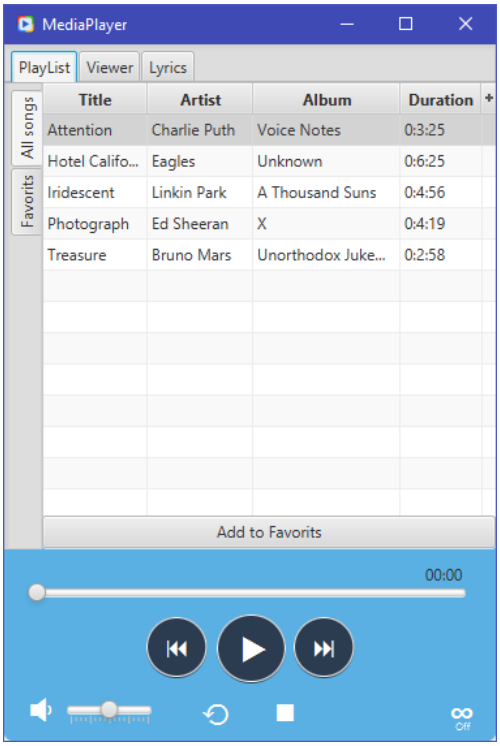
1. L'onglet *Viewer* qui affiche la musique en cours d'être jouée avec le titre, le nom de l'artiste et la photo de l'album en s'actualisant avec chaque changement de musique.

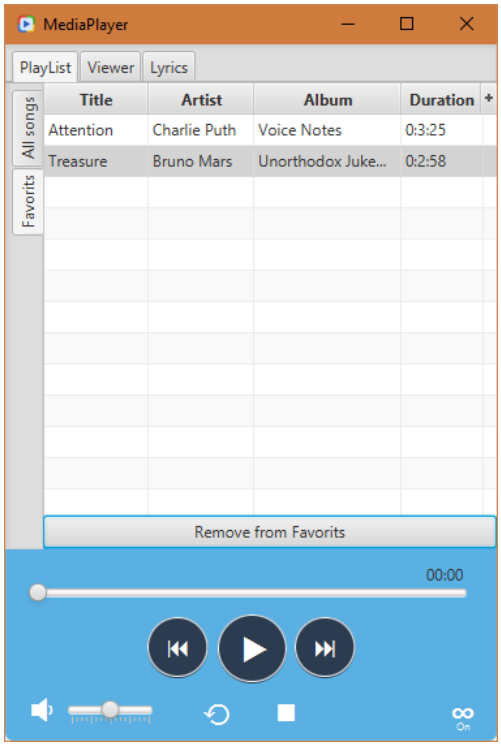


2. L'onglet *Lyrics* qui affiche les paroles de la musique en cours d'être jouée.



3. L'onglet *PlayList* qui affiche toutes les musique et la playlist des favoris.

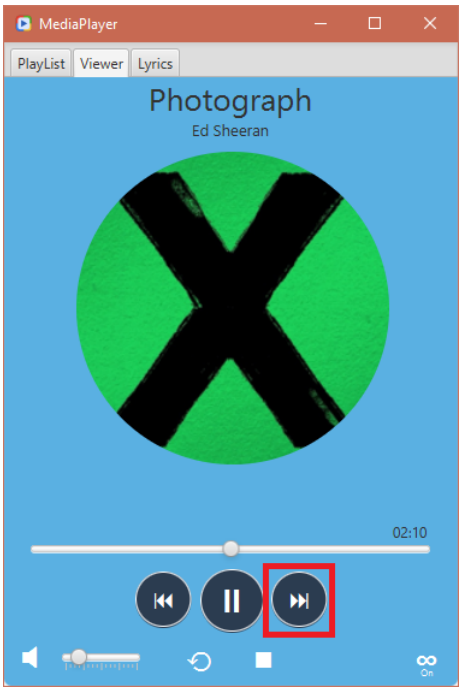




F.3 Les boutons

1. Bouton Play/Pause, joue ou pause la musique en actualisant le symbol.

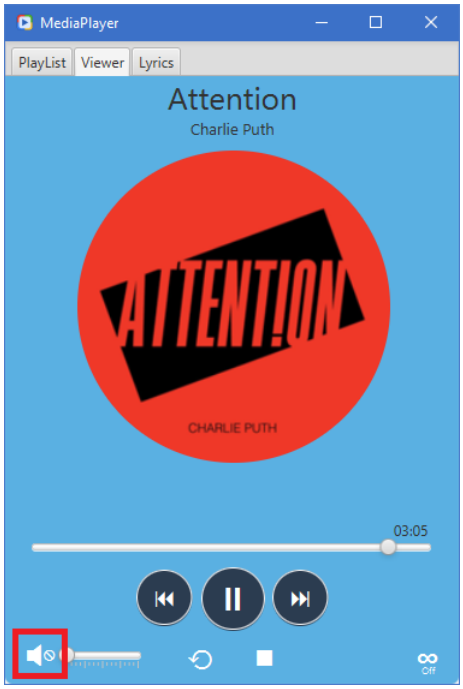
2. Boutons Next/Previous, passe à la musique suivante ou précédente en actualisant les informations.



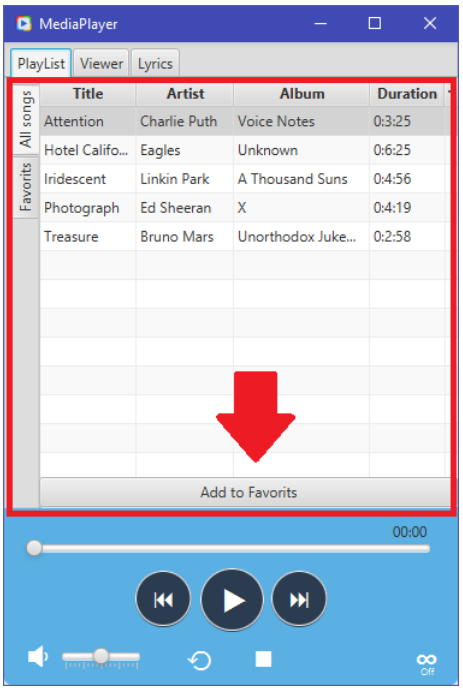
3. Boutons Restart/Stop, recommence ou arrête la musque.

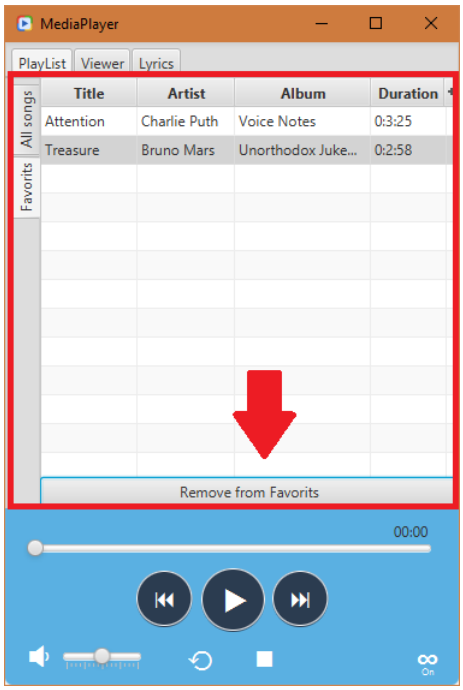
4. Bouton Loop, met la musique en boucle.

5. Barre d'avancement/Volume. La barre d'avancement suit la musique en actualisant la durée passée et la barre volume modifie le son avec la possibilité de cliquer su l'icon pour muer le volume.



6. Finalement, les boutons Add/Remove, permettent d'ajouter ou enlever une musique de la playlist favoris.





G Conclusion

Après cette description des diverses fonctions de notre programme, nous résumerons divers constats. Premièrement les améliorations possibles mais que nous n'avons pu créer du à des problèmes, que ça soit des informations difficilement trouvables, un manque de temps et une complexité de conception utilisant des choses que nous n'avons pas réussi à utiliser/comprendre.

— une visualisation animée du son en cours
— la création de multiples playlists
— la gestion de fichiers ayant d'autres extensions

Malgré cela nous avons pus nous améliorer dans bien des domaines, en voiçi quelques uns :

— l'utilisation du fxml pour une interface
— la gestion de fichier
— l'utilisation des librairies externes pour la lecture de fichier audio
— la gestion de fonctions asynchrones et synchrones
— la gestion des sliders et autres interactions avec des interfaces, que ce soit en fxml(javafx) ou en swing

H Bibliographie

La plus part de nos recherches ont été faites sur :
— Stack Overflow.
— Oracle.
— Des question posées durant les heures de TP.

I Fin

Nous vous remercions d'avoir lu ce rapport jusqu'au bout et de vous être penché sur ce programme.
Cordialement, Mr. ZAIZAFOUN et Mr. JACQUELINE.