Travail Pratique 3

Préparé par
Pierre Poulin (modifié par Karine Filiatreault)

1 Résumé

Pour ce travail pratique, nous allons concevoir une petite application de style « *media player* » avec quelques règles particulières.

2 Conditions de réalisation

Valeur de la note finale	Type	Durée	Nombre de remises
20 %	Équipe de 2 ou Individuel	2 semaines	1

3 Spécifications

Cette section vise à vous guider durant la réalisation de ce travail. Réalisez chaque étape dans l'ordre et passez à l'étape suivante uniquement lorsque vous avez terminé l'étape actuelle.

3.1 Récupération du projet

1 Récupérez le projet fourni avec l'énoncé. Prenez quelques instants pour comprendre son contenu.

3.2 Création des classes correspondant au diagramme de classes

Vous trouverez en annexe le diagramme de classes associé à ce travail. Prenez-en connaissance avant de le coder. En particulier, le type énuméré possède des méthodes que vous ne devez pas coder. Assurez-vous de les identifier avant de débuter la codification.

Pour toutes les classes que vous coderez, vous devez vous assurer que chaque méthode valide adéquatement les paramètres reçus et lance les exceptions appropriées lorsque nécessaire.

Portez également attention à la visibilité des méthodes.

Pensez à utiliser LINQ lorsque cela est utile.

Particularités à respecter pour chacune des classes à coder :

Paquet **`comparer`**

`IMediaComparer`

Interface permettant de comparer deux médias. Le critère de comparaison est précisé dans les sous-classes qui implantent cette interface. La méthode doit retourner -1 si le premier paramètre est plus petit que le second, 1 si le premier est plus grand que le second ou 0 si les deux sont égaux.

`NoSortComparer`, `TitleAscComparer`, `TitleDescComparer`, `YearAscComparer`, `YearDescComparer`

Ces classes implantent l'interface **`IMediaComparer`** et ont comme objectif de permettre la comparaison de deux médias selon le critère spécifié dans le nom de chaque classe.

Paquet **`media`**

`IPlayable`

Interface offrant les fonctionnalités de base d'un média : la lecture et l'arrêt.

`Media`

Classe abstraite représentant le concept général d'un média (musique ou vidéo).

`Music` et `Video`

Classes concrètes héritant de **`Media`**. Pour les besoins de ce travail, vous aurez juste le son lorsque vous démarrerez une vidéo. Vous pouvez vous amuser à utiliser un *player* externe pour les vidéos (comme VLC) mais ce n'est pas demandé.

`MediaPlayer`

Cette classe représente le concept d'un gestionnaire de médias. Certaines méthodes sont fournies mais possèdent du code en commentaires. Vous ne devriez pas avoir à les modifier autrement qu'en retirant le commentaire lorsque les classes **`Music`** et **`Video`** auront été codées.

Cette classe possède une liste des médias disponibles. Ces médias seront chargés lors de l'appel à la méthode **`LoadMedias`**.

Cette classe possède aussi un attribut de type **`Playlist`**. La liste de lecture (vide au démarrage de l'application) contient un sous-ensemble des médias disponibles.

`Playlist`

Cette classe représente le concept d'une liste de lecture. La liste de lecture contient une collection de médias (un sous-ensemble de tous les médias présents dans le **`MediaPlayer`**).

Il est possible d'ajouter et de retirer un média dans la liste de lecture.

La méthode **`FilterUnusedMedias`** a pour objectif concret de déterminer les médias qui ne sont pas déjà dans la liste de lecture à partir d'une liste reçue en paramètre. Elle est utilisée dans le contexte où il n'est pas permis d'ajouter deux fois le même média dans la liste de lecture. On veut donc éviter de proposer des choix invalides à l'utilisateur.

Les méthodes **`PlayNext`** et **`PlayPrevious`** doivent, comme leurs noms l'indiquent, jouer respectivement les médias suivants et précédents dans la liste de lecture. Si jamais la fin de la liste est atteinte avant l'appel à **`PlayNext`** il faut jouer le premier média. Si le début de la liste est atteint avant l'appel à **`PlayPrevious`**, il faut jouer le dernier média de la liste.

Notes importantes pour les exceptions :

- Tenter d'enlever un média dont l'index est invalide doit lancer un **`ArgumentOutOfRangeException`**.
- Tenter de retirer un média qui n'est pas présent dans la liste doit lancer un **`ArgumentException`**.
- Tenter de jouer un média dans une liste de lecture vide doit lancer un **`InvalidOperationException`**.

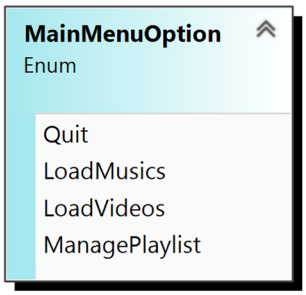
3.3 Codification des fonctionnalités utilisant les classes du diagramme de classes

Menu principal

1 Ajoutez à votre `Main` la possibilité pour l'utilisateur de choisir ce qu'il souhaite faire parmi quatre options : quitter, charger le contenu musical, charger le contenu vidéo et gérer la liste de lecture courante:

```
Your options are:

 0 Quit
 1 Load all musics
 2 Load all videos
 3 Manage playlist
Enter option:
```

Vous devez utiliser le type énuméré `MainMenuOption` pour la gestion du choix de l'utilisateur (pas de `int` ni de `String` sauf pour lire l'entrée à la console évidemment).



Gestion de la liste de lecture

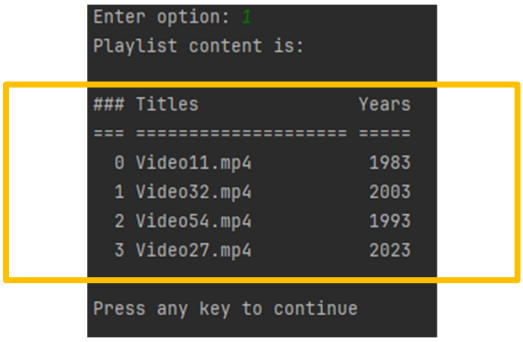
Lorsque l'utilisateur choisit l'option 3 dans le menu principal, un autre menu lui est alors proposé permettant de gérer la liste de lecture courante :

```
Enter option: 3
Your options are:

0 Quit
1 Print playlist
2 Add media to playlist
3 Remove media from playlist
4 Sort playlist by title (ascending)
5 Sort playlist by title (descending)
6 Sort playlist by Year (ascending)
7 Sort playlist by Year (descending)
8 Start the playlist (play)
Enter option: 2
```

Ce menu permet de nouvelles possibilités :

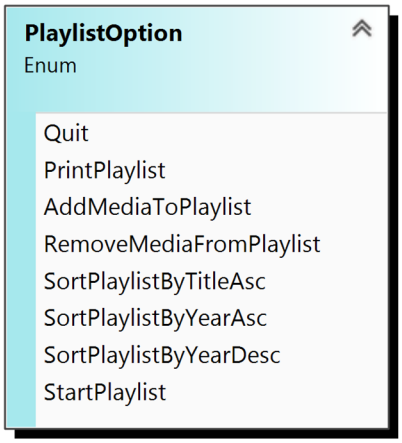
2 Écrivez le code nécessaire pour proposer ces possibilités à l'utilisateur. N'oubliez pas de faire en sorte que l'utilisateur ne puisse pas quitter ce menu s'il n'entre pas 0 pour quitter.

3 Ajoutez le code requis pour afficher la liste de lecture

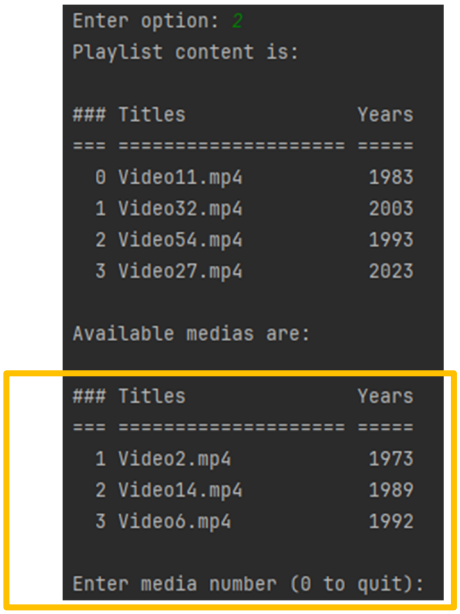


Notez que le contenu encadré doit être réalisé en appelant la méthode **`ToString`** de la classe **`Playlist`**.

Pour les différentes options offertes dans le menu de gestion de la liste de lecture, vous devez utiliser le type énuméré **`PlaylistOption`** :

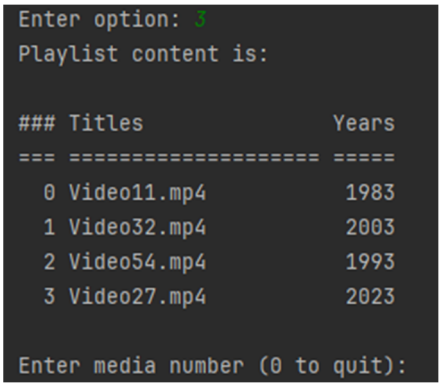


4 Ajoutez le code requis pour permettre l'ajout de médias dans la liste de lecture. L'ajout de médias se fait en deux étapes. La première étape affiche la liste de lecture actuelle. Vous pouvez évidemment réutiliser une partie du code écrit au point précédent.



La seconde étape énumère à l'utilisateur les médias disponibles car il n'est pas permis d'ajouter deux fois le même média dans la liste de lecture. Pour générer le contenu de la seconde liste, vous devez utiliser la méthode **`FilterUnusedMedias`** de la classe **`PlayList`**. L'utilisateur doit finalement saisir le numéro du média qu'il souhaite ajouter.

5 Ajoutez le code requis pour permettre le retrait de médias de la liste de lecture. Le retrait des médias débute par l'affichage de la liste de lecture initiale. L'utilisateur est ensuite amené à saisir le numéro du média qu'il souhaite retirer de la liste de lecture.



6 Ajoutez le code requis pour permettre le tri de la liste de lecture selon les différents critères proposés dans le menu. Il est à noter qu'une seule instruction est requise pour cela. Elle consiste à appeler la méthode **`Sort`** de **`Playlist`** en lui passant l'instance concrète appropriée d'un **`IMediaComparer`**.
Lorsqu'un ordre de tri a été sélectionné, il doit toujours être respecté notamment lors de l'ajout d'un nouveau média dans la liste de lecture. Par exemple, si l'affichage d'une liste de lecture donne

```
Playlist content is:

### Titles                Years
=== ==================== =====
  0 Song2.mp3             1973
  1 Song11.mp3            1983
  2 Song14.mp3            1989
  3 Song54.mp3            1993
  4 Song32.mp3            2003

Press any key to continue
```

et que le média #1 est ajouté

```
Available medias are:

### Titles                Years
=== ==================== =====
  1 Song6.mp3             1992
  2 Song27.mp3            2023
```

le prochain affichage de la liste de lecture devrait donner

```
Playlist content is:

### Titles                Years
=== ==================== =====
  0 Song2.mp3             1973
  1 Song11.mp3            1983
  2 Song14.mp3            1989
  3 Song6.mp3             1992
  4 Song54.mp3            1993
  5 Song32.mp3            2003

Press any key to continue
```

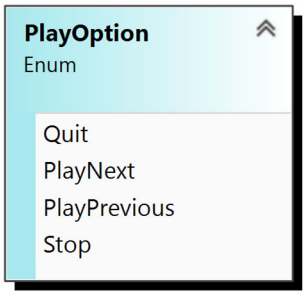
Écoute de la liste de lecture

Lorsque l'utilisateur choisit l'option 8 pour démarrer l'écoute de la liste, la première chanson (selon le tri choisi) commence à jouer. Un nouveau menu s'affiche aussi permettant d'avancer et de reculer dans la liste.

> **Information**
>
> Lorsqu'on démarre une nouvelle chanson, on doit préalablement arrêter la chanson qui jouait si on ne veut pas entendre les 2 chansons en même temps 😊

```
0 Quit
1 Print playlist
2 Add media to playlist
3 Remove media from playlist
4 Sort playlist by title (ascending)
5 Sort playlist by title (descending)
6 Sort playlist by Year (ascending)
7 Sort playlist by Year (descending)
8 Start the playlist (play)
Enter option: 8

sample-6s.mp3 is playing...

Your options are:

0 Quit
1 Play next
2 Play previous
3 Stop
If you quit the song will stop.

Enter option: 1

Stopping music sample-6s.mp3

sample.mp3 is playing...

Your options are:

0 Quit
1 Play next
2 Play previous
3 Stop
If you quit the song will stop.
```

Pour les différentes options offertes dans le menu de gestion de la liste de lecture, vous devez utiliser le type énuméré `PlayOption` :



Pour pouvoir utiliser la classe `WindowsMediaPlayer`, vous devez :

1. Ajouter `using WMPLib` pour avoir accès à la classe `WindowsMediaPlayer`
2. Appeler ensuite la méthode appropriée (`play()` ou `stop()`) sous l'attribut `controls`

Information

Pour ne pas avoir à vous casser la tête avec les chemins d'accès aux fichiers pour les chansons et vidéos (non fournis) qui seront accessibles à votre lecteur, je vous suggère de simplement les mettre dans le {dossier de votre tp}/TP3/bin/Debug/net8.0. De cette façon vous y aurez accès directement dans le code sans avoir à spécifier de *path*. Idem pour les fichiers `songs.music` et `videos.video` qui sont fournis.

4 Tests unitaires

Vous devez produire tous les tests unitaires pour la classe `Playlist`. Vous devez tester adéquatement toutes les méthodes sauf les méthodes `PlayNext`, `PlayPrevious`, `Sort` et `ToString`.

Pour la correction des tests unitaires, vous serez évalués(ées) de deux manières :

- Écriture de vos tests, notamment le niveau de couverture du code. Inspirez-vous des situations décrites dans la grille d'évaluation pour identifier tous les cas pertinents à tester.
- Exécution des tests du professeur.

Évidemment, pour que les tests du professeur puissent fonctionner, il faut que les signatures de méthodes soient similaires. Pour s'en assurer, un fichier vous est fourni. Il ne contient aucun test mais possède une méthode privée fournie qui fait des appels de méthodes. Vous devez vous assurer que cette méthode compile. Elle ne sert à rien d'autre. Vous pourrez enlever le commentaire au moment d'écrire les tests. En principe, si vous respectez les signatures fournies dans le diagramme de classes vous ne devriez pas avoir de problème à ce que les tests du professeur compilent.

N'oubliez pas de tester avec différentes valeurs afin que les exceptions appropriées soient générées.

5 Évaluation

Vous trouverez dans le fichier ci-joint la grille d'évaluation utilisée pour la correction de ce travail pratique.

6 Modalités de remise

Remettez votre projet sur LÉA, dans la section travaux, à l'intérieur d'une archive *Zip*. Supprimez les dossiers inutiles comme vu précédemment.

Annexe 1

Diagramme de classes du projet