IFT1025 - TP1 (25%) - POOphonia

Échéancier et éléments fournis avec l'énoncé

Disponible dès le 7 février 2025. À rendre via StudiUM au plus tard le 9 mars 2025 à 23h59. Vous devrez remettre tous vos codes dans une archive `TP1_Member1_Member2.tar.gz` (voir détails plus bas). Vérifier les fichiers qui viennent avec l'énoncé. Vous trouverez des classes Java, des fichiers de commandes (`.txt`), des librairies (`.csv`) et des fichiers d'output (`.output`).

Lisez attentivement l'énoncé. Il y a des bits d'information un peu partout qui répondront à la plupart de vos questions. Si des questions persistent, n'hésitez pas à les poser sur le forum `TP1Q&R` sur le site StudiUM du cours.

Objectif

Concevez et implémentez un système de gestion de bibliothèque musicale en Java qui démontre les principes fondamentaux de la programmation orientée objet (POO) : Encapsulation, héritage et polymorphisme. Ce projet vous aidera à consolider votre compréhension des concepts de la POO dans une application pratique. De plus, il vous permettra de vous familiariser avec les "packages".

Aperçu du projet

Vous allez créer une application, **POOphonia**, permettant aux utilisateurs de gérer une collection d'éléments de musique (`MusicItem`). Le système prendra en charge différents éléments, tels que des chansons (`Song`), des albums (`Album`) et des podcasts (`Podcast`), mettant en évidence l'héritage et le polymorphisme. L'encapsulation sera démontrée par l'utilisation de champs privés et de méthodes publiques pour accéder et modifier les données.

Exigences

1. Encapsulation

Champs privés : Tous les attributs des classes doivent être privés pour protéger l'intégrité des données.

Getters et *Setters* publics : Fournir des méthodes publiques pour accéder aux champs privés et les modifier.

Validation des entrées : S'assurer que les données sont valides.

2. Héritage (modèle)

Vous aller créer un "**package**" pour les modèles (**package** `models`). Les modèles sont constitués de :

`MusicItem` : Une classe de base pour les éléments avec au minimum :

- Les attributs : `id` (**int**), `title` (**String**), `releaseYear` (**int**), `isPlaying` (**boolean**).
- Les méthodes : *Getters* et *setters*, `play`() (pour démarrer la lecture d'un élément), `pause`() (pour mettre sur pause la lecture d'un élément), `stop`() (pour arrêter la lecture d'un élément) et `toString`() (pour afficher les informations d'un élément).
- Le constructeur avec les paramètres `id`, `title` et `releaseYear` et qui met initialise `isPlaying` à **false**.
- Les méthodes abstraites : `getInfo`() (pour obtenir les informations spécifiques du type d'élément musical) et `toCSV` (pour transformer un élément en format CSV: "comma separated value").

et de classes dérivées de `MusicItem` :

- **`Song`** :
  - Champs supplémentaires : `artist` (**String**), `genre` (**String**), `duration` (**int** en secondes)
- **`Album`** :
  - Champs supplémentaires : `artist` (**String**), `numberOfTracks` (**int**), `label` (**String**)
- **`Podcast`** :
  - Champs supplémentaires : `host` (**String**), `episodeNumber` (**int**), `topic` (**String**)

3. Polymorphisme

Redéfinition de méthodes : Chaque classe dérivée doit redéfinir les méthodes nécessaires pour ajuster le comportement différent et détaillé spécifique aux éléments de musique (ex. : `toString`(); une chanson affiche l'id, le titre, l'année de sortie, l'artiste, le genre et la durée alors que pour un album on affiche l'id, le titre, l'année de sortie, le nombre de morceaux et la maison de disque.

La hiérarchie des classes et les spécifications fonctionnelles minimales sont montrées dans la **Figure 1**, à gauche.

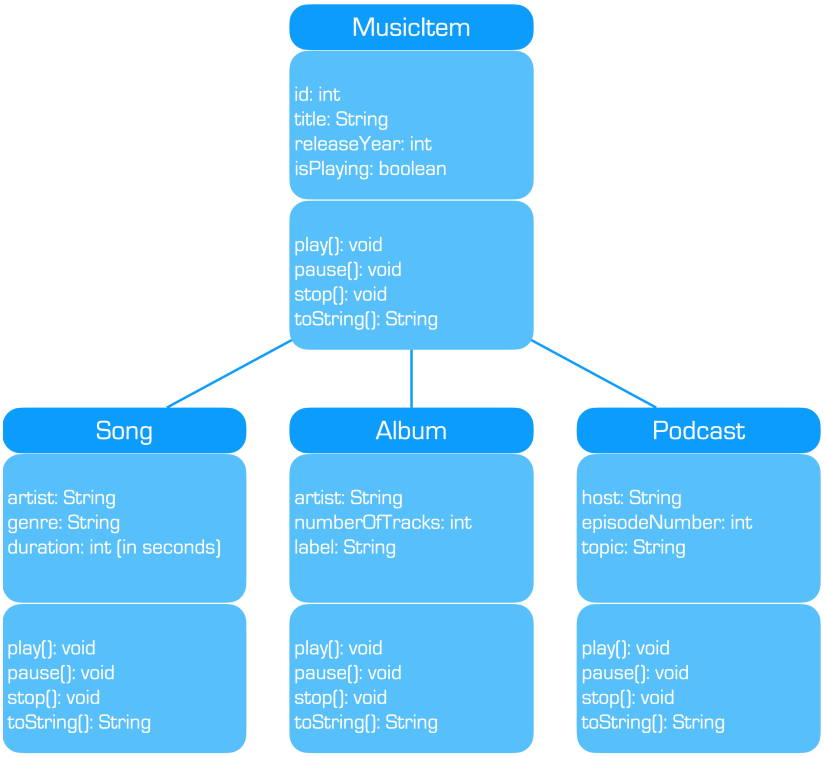
Librairies dans des fichiers en format CSV

`MusicItemFactory`

Vos modèles d'éléments de musique auront besoin d'une classe pour interfacer des éléments de musique stockés dans un fichier en format CSV. Le code de cette classe est donné dans le répertoire du TP1. Un exemple de librairie en format CSV est donné dans la **Figure 1**, en bas.

Services

Vous aller créer un "`package`" pour les services (`package services`). Les services sont constitués au minimum de :




```
song,1,Imagine,1971,John Lennon,Rock,183
album,2,Thriller,1982,Michael Jackson,Epic Records,9
podcast,3,Tech Talk,2023,Jane Doe,AI,42
song,4,Imagine,2004,A Perfect Circle,Alternative,288
```

Figure 1: Modèles de **POOphonia**. En haut) Hiérarchie de classes pour **POOphonia**. En bas) Exemple du fichier de la librairie **POOphonia** en format CSV : `POOphonia.csv`.

`MusicLibrary`

`MusicLibrary` aura besoin d'une liste pour gérer les instances d'éléments musicaux. Utilisez la classe `ArrayList` de Java.

- **private** ArrayList<MusicItem>[1] items;

`MusicLibrary` aura aussi besoin d'un constructeur et de certaines méthodes pour effectuer les commandes de l'application **POOphonia**, au minium :

- **public void** addItem(MusicItem item);
  - pour ajouter un élément de musique à la librairie.
- **public void** removeItem(**int** id);
  - pour retirer l'élément de musique avec l'id `id` de la librairie.
- **public void** listAllItems();
  - pour lister les éléments de musique présents dans la librairie.
- **public void** playItem(int id);
  - pour lancer/jouer l'élément musical `id`.
- **public void** pauseItem();
  - pour mettre sur pause l'élément musical qui joue.
- **public void** stopItem();
  - pour arrêter l'élément musical qui joue.
- **public void** clearAllItems();
  - pour vider la liste des éléments musicaux présents dans la librairie.

MusicLibraryFileHandler

Un service nécessaire est l'entrée/sortie des éléments de musique de et vers les fichiers en format CSV. `MusicLibraryFileHandler` est fourni avec l'énoncé du TP. Il est prêt à être utilisé par vos classes. Notez que le fichier de la libraire par défaut de **POOphonia**, `POOphonia.csv`, est situé dans le répertoire "`data`" au top niveau de vos `packages`. Notez également que plusieurs librairies peuvent exister, puisque **POOphonia** permet de lire et écrire des libraires en fournissant un nom dans les commandes `LOAD` et `SAVE` (voir plus bas). Le répertoire "`data`" est le seul utilisé pour stocker toutes les librairies.

CommandProcessor

Vous devrez créer une classe `CommandProcessor` pour effectuer et valider les commandes envoyées à **POOphonia**. Pour le TP, les commandes seront placées dans un fichier de commandes, "`commands.txt`" aussi placé dans le répertoire "`data`" (**Figure 2**).

---

[1] Le type `ArrayList` est l'équivalent de la liste en Python. Elle a un paramètre qui indique le type des éléments à gérer. Elle s'importe du package java.util : `import java.util.ArrayList;` Nous verrons les listes en détails dans le module 6. En attendant, vous pouvez consulter la documentation : `https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html`

```
# POOphonia Test Case - Command File
# This test case covers adding, removing, playing, and listing music items.

# Adding various items
ADD song,1,Imagine,1971,John Lennon,Rock,183
ADD song,2,Bohemian Rhapsody,1975,Queen,Rock,354
ADD album,3,Thriller,1982,Michael Jackson,Epic Records,9
ADD album,4,The Dark Side of the Moon,1973,Pink Floyd,Harvest Capitol,10
ADD podcast,5,Tech Talk,2023,Jane Doe,Technology,42
ADD podcast,6,Science Weekly,2022,John Smith,Science,24

# Listing all items (Expect: 6 items displayed)
LIST

# Searching for a specific song by ID (Expect: Imagine by John Lennon)
SEARCH 1

# Searching for a song that does not exist (Expect: Not found)
SEARCH 99

# Searching by title and artist (Expect: Bohemian Rhapsody by Queen)
SEARCH Bohemian Rhapsody by Queen

# Playing the searched song
PLAY

# Searching by title and artist (Expect: Thriller by Michael Jackson )
SEARCH Thriller by Michael Jackson

# Playing Bohemian Rhapsody continues...
PLAY

# Searching by title but wrong artist (Expect: Not found)
SEARCH Bohemian Rhapsody by The Beatles

# Playing a song by ID (Expect: Playing "Imagine" by John Lennon)
PLAY 1

# Playing a song by title and artist (Expect: Playing "Bohemian Rhapsody" by Queen)
PLAY Bohemian Rhapsody by Queen

# Playing a non-existent song (Expect: Not found)
PLAY Hey Jude by The Beatles

# Removing a song (Expect: Bohemian Rhapsody removed)
REMOVE 2

# Listing items after removal (Expect: 5 items remaining)
LIST

# Removing a non-existent item (Expect: No item found)
REMOVE 99

# Playing an item after removal (Expect: Not found)
PLAY 2

# Clearing the library before exiting
CLEAR

# Exit the program
EXIT
```

```
***** POOphonia: Welcome! *****
Library in file POOphonia loaded successfully.
Sourcing commands...
Song of 1971 Imagine by John Lennon added to the library successfully.
Library saved successfully to POOphonia.
Song of 1975 Bohemian Rhapsody by Queen added to the library successfully.
Library saved successfully to POOphonia.
Album Thriller of 1982 with 9 tracks by Michael Jackson added to the library successfully.
Library saved successfully to POOphonia.
Album The Dark Side of the Moon of 1973 with 10 tracks by Pink Floyd added to the library successfully.
Library saved successfully to POOphonia.
Podcast Tech Talk episode 42 of 2023 on Technology by Jane Doe added to the library successfully.
Library saved successfully to POOphonia.
Podcast Science Weekly episode 24 of 2022 on Science by John Smith added to the library successfully.
Library saved successfully to POOphonia.
Library:
Song [ID=1, Title=Imagine, Release Year=1971, Artist=John Lennon, Genre=Rock, Duration=183s]
Song [ID=2, Title=Bohemian Rhapsody, Release Year=1975, Artist=Queen, Genre=Rock, Duration=354s]
Album [ID=3, Title=Thriller, Release Year=1982, Artist=Michael Jackson, Tracks=9, Label=Epic Records]
Album [ID=4, Title=The Dark Side of the Moon, Release Year=1973, Artist=Pink Floyd, Tracks=10, Label=Harvest Capitol]
Podcast [ID=5, Title=Tech Talk, Release Year=2023, Host=Jane Doe, Episode=42, Topic=Technology]
Podcast [ID=6, Title=Science Weekly, Release Year=2022, Host=John Smith, Episode=24, Topic=Science]
Song of 1971 Imagine by John Lennon is ready to PLAY
SEARCH item ID 99 failed; no such item
Song of 1975 Bohemian Rhapsody by Queen is ready to PLAY
Playing Song of 1975 Bohemian Rhapsody by Queen.
Album [ID=3, Title=Thriller, Release Year=1982, Artist=Michael Jackson, Tracks=9, Label=Epic
Records]
Song of 1975 Bohemian Rhapsody by Queen is already playing.
SEARCH Bohemian Rhapsody by The Beatles failed; no item found.
Stopping Song of 1975 Bohemian Rhapsody by Queen.
Playing Song of 1971 Imagine by John Lennon.
PLAY item: Hey Jude by The Beatles failed; no such item.
Removed Song of 1975 Bohemian Rhapsody by Queen successfully.
Library saved successfully to POOphonia.
Library:
Song [ID=1, Title=Imagine, Release Year=1971, Artist=John Lennon, Genre=Rock, Duration=183s]
Album [ID=3, Title=Thriller, Release Year=1982, Artist=Michael Jackson, Tracks=9, Label=Epic
Records]
Album [ID=4, Title=The Dark Side of the Moon, Release Year=1973, Artist=Pink Floyd, Tracks=10,
Label=Harvest Capitol]
Podcast [ID=5, Title=Tech Talk, Release Year=2023, Host=Jane Doe, Episode=42, Topic=Technology]
Podcast [ID=6, Title=Science Weekly, Release Year=2022, Host=John Smith, Episode=24,
Topic=Science]
REMOVE item 99 failed; no such item.
PLAY item ID 2 failed; no such item.
Music library has been cleared successfully.
Library saved successfully to POOphonia.
***** POOphonia: Goodbye! *****
```

Figure 2. Exemple de fichier de commandes (en haut) et de son "output" (en bas). **POOphonia** interprète les commandes du fichier `commands.txt` et envoie sur le "stdout" les résultats des commandes, qui ont été redirigés vers le fichier `commands.output`. Notez qu'après chaque insertion d'un élément musical, **POOphonia** sauve la library. Notez aussi que **POOphonia** termine normalement même si la commande EXIT est omise.

Validation et exécution

La classe `CommandProcessor` doit valider et effectuer les instructions du fichier de commandes. Notez qu'une commande qui débute avec le caractère '#' n'est pas interprétée.

Les validations incluent : l'identificateur (`id`) qui doit être un entier positif unique à chaque élément de musique, l'année de sortie (`releaseYear`) doit être un entier positif entre 1850 et 2025 (inclusivement), la durée d'une chanson doit être entre 1 et 36 000 secondes (inclusivement), le nombre de morceaux sur un album doit être entre 1 et 100 (inclusivement) et le numéro d'épisode d'un podcast doit être entre 1 et 500 (inclusivement).

Les commandes incluent : `SOURCE`, `LOAD`, `SAVE`, `ADD`, `REMOVE`, `PLAY`, `PAUSE`, `STOP`, `CLEAR`, et `LIST`. La **Figure 3** donne les commandes et leur comportement.

Commmande	Comportement
`SOURCE |` `SOURCE <file>`	Exécute les commandes du fichier de commandes par défaut, `commands.txt`, ou du fichier passé en argument. Notez qu'au lancement de **POOphonia**, le fichier de commandes par défaut est lu automatiquement.
`LOAD |` `LOAD <lib>`	Charge les éléments musicaux de la librairie par défaut, ou provenant du fichier `<lib>` lorsque fourni.
`SAVE |` `SAVE <file>`	Sauve la librairie sur le fichier par défaut, ou dans le fichier `<file>` lorsque fourni. Notez qu'à la fin d'exécution de **POOphonia**, la librairie est automatiquement sauvée dans le fichier par défaut.
`ADD <item>`	Charge l'`item` (Song, Album ou Podcast) dans la librairie si toutes les parties sont valides.
`REMOVE <id>`	Retire l'élément musical (Song, Album ou Podcast) avec l'ID `<id>` de la librairie.

`SEARCH <id> |`
`<title> by`
`<artist>`
	Trouve et prépare un élément de musique (Song, Album ou Podcast) avec l'id `<id>` OU correspondant à un titre et artiste.
	Les informations de l'élément trouvé sont affichées. De plus, si aucun élément est en train de jouer, la commande met l'élément trouvé comme prochain à jouer.

`PLAY |`
`PLAY <id> |`
`PLAY <title>`
`by <artist>`
	Lance/joue le prochain élément musical (Song, Album ou Podcast) OU l'élément musical avec l'ID `<id>` OU l'élément musical avec titre `<title>` et artiste `<artist>`.
	Si un autre élément musical est en train de jouer, il est stopper avant que le nouvel élément soit lancer.

`PAUSE`	Met sur pause l'élément musical qui était en train de jouer
`STOP`	Arrête l'élément musical qui était en train de jouer.
`CLEAR`	Retire tous les éléments musicaux de la librairie :
`LIST`	Énumère les éléments musicaux de la librairie.

Figure 3. Liste des commandes et comportements.

Messages

Pour simplifier la gestion des messages envoyés par les unités de services et leur signification, la **Figure 4** présente une énumération complète des situations et messages à envoyer via la méthode `send` de la classe `Message` (dans le `package ui`).

Commande	Situation	Message associé
`SOURCE | SOURCE <file>`	The command file is already being processed, infinite loop must be avoided.	Currently sourcing <filename>; SOURCE ignored.
	The specified command file does not exist.	Sourcing <filename> failed; file not found.
	A valid command file starts processing.	Sourcing <filename>...
	There is an error while reading the command file.	Error reading command file: <filename>
`ADD <item>`	The ADD command has missing parameters.	Invalid ADD command: <commandLine>
	The ADD command has an incorrect number of elements.	Wrong number of elements: <commandLine>
	The type of music item is invalid (not “song”, “album”, or “podcast”).	Wrong music item: <commandLine>
	The id is not a valid number or is less than 1.	Invalid music ID: <commandLine>
	The release year is incorrectly formatted or outside a reasonable range.	Invalid release year: <commandLine>
	The duration (for songs) is negative or longer than 10 hours.	Invalid duration: <commandLine>
	The number of tracks (for albums) is less than 1 or more than 100.	Invalid number of tracks: <commandLine>
	The episode number (for podcasts) is less than 1 or more than 500.	Invalid episode number: <commandLine>
	The item creation failed.	ADD item <itemData> failed; no such item.
	The given ID is already in use by another item.	ADD <item> failed; ID already used.
	The item is already in the library (same title, artist, etc.).	ADD <item> failed; item already in library.
	The item is successfully added to the library.	<item_info> added to the library successfully.
`REMOVE <id>`	The REMOVE command is missing an ID parameter.	Invalid REMOVE command: <commandLine>
	The ID is not a valid number.	Invalid ID for REMOVE command: <idStr>
	The item with the specified ID does not exist.	No item found with ID: <id>
	The specified ID does not match any item in the library.	REMOVE item <id> failed; no such item.
	The item is successfully removed from the library.	Removed <item_info> successfully.
`SEARCH <id> | <title> by <artist>`	The SEARCH command is missing parameters.	Invalid SEARCH command: <commandLine>
	The ID is invalid or the item is not found.	SEARCH item ID <id> failed; no such item
	The title and artist combination is invalid or not found.	SEARCH <title> by <artist> failed; no item found.
	The search format is incorrect.	Invalid SEARCH format. Use 'SEARCH <id>' or 'SEARCH <title> by <artist>'
Search prépare le prochain élément	There is no currently playing item	<item_info> is ready to PLAY.
	Another item is already playing	<item.toString()>
`PLAY | PLAY <id> | PLAY <title> by <artist>`	The PLAY command is missing parameters.	Invalid PLAY command: <commandLine>
	The ID is invalid or the item does not exist.	Invalid ID for PLAY command: <idStr>

	The title and artist combination does not match any item.	PLAY item: <title> by <artist> failed; no such item.
	The command format is incorrect.	Invalid PLAY format. Use 'PLAY', 'PLAY <id>' or 'PLAY <title> by <artist>'
	The requested item ID does not exist.	PLAY item ID <id> failed; no such item.
	The requested item is already playing.	<item_info> is already playing.
	A new item starts playing.	Playing <item_info>.
	There is no item to play when calling playItem() without an argument.	No item to PLAY.
`PAUSE & STOP`	The PAUSE command has extra parameters.	Invalid PAUSE command: <commandLine>
	There is no item currently playing.	No item to PAUSE.
	The item is already paused.	<item_info>; is already on pause.
	The playing item is successfully paused.	Pausing <item_info>.
	The STOP command has extra parameters.	Invalid STOP command: <commandLine>
	There is no item currently playing.	No item to STOP.
	The playing item is successfully stopped.	Stopping <item_info>.
`LIST`	The LIST command has extra parameters.	Invalid LIST command: <commandLine>
	The library is empty.	The library is empty.
	The library contains items.	Library: (followed by item details)
	The library is empty.	Music library is empty.
	The library contains items.	<item.toString()> (for each item in the library)
`LOAD/SAVE | LOAD/SAVE <lib>`	The LOAD command runs without specifying a file.	Loading from default library file.
	The LOAD command runs with a specified file.	Loading from file: <filename>
	The SAVE command runs without specifying a file.	Saving to default library file.
	The SAVE command runs with a specified file.	Saving to file: <filename>
	The library is loaded from a file.	(Handled by MusicLibraryFileHandler)
	The library is saved to a file.	(Handled by MusicLibraryFileHandler)
`CLEAR & EXIT`	The CLEAR command has extra parameters.	Invalid CLEAR command: <commandLine>
	The EXIT command has extra parameters.	Invalid EXIT command: <commandLine>
	The EXIT command stops execution.	Exiting program...
	The library is already empty.	Music library is already empty.
	The library is successfully cleared.	Music library has been cleared successfully.
`Invalid commands`	An unknown command is entered.	Unknown command: <commandLine>

Figure 4. Listes des commandes, situations et messages.

Lignes directrices pour l'implémentation

1. Configuration :

- Créez un nouveau projet Java.
- Créez les répertoires `src/main/java` et `data` au top niveau de votre projet. Dans le répertoire `src/main/java`, créez les répertoires pour vos packages : `models`, `services` et `ui`. Notez que la classe `MusicItemFactory` du `package models`, la classe `MusicLibraryFileHandler` du `package services` et les 2 classes `Message` et **POOphonia** du `package ui` ("user interface") sont fournies avec l'énoncé.
- Un fichier `makefile` est aussi fourni avec l'énoncé. Copiez le au top niveau de votre projet. Vous pourrez utiliser ce fichier pour compiler et exécuter vos classes Java. Tapez `make help` au top niveau de votre projet pour voir les options du `makefile`.
- Le fichier `pom.xml` est fourni pour ceux qui voudrait travailler avec Maven.
- Si vous utilisez un IDE pour faire le TP, assurez vous quand même de respecter la hiérarchie des répertoires. Elle devra être respectée lorsque vous soumettrez les classes de votre TP dans StudiUM.

2. Conception des classes :

- Commencez par concevoir la classe de base `MusicItem` et ses sous-classes.
- Assurez vous que chaque classe encapsule correctement ses données.

3. Gestion de la bibliothèque :

- Implémentez la classe `MusicLibrary` pour gérer les opérations.

4. Interface des commandes :

- Développez le service de lecture et gestion des fichiers de commandes, `CommandProcessor`.

5. Tests :

- Utilisez les cas de test fournis (ou qui seront fournis) avec l'énoncé. Créez vous même différents cas de test pour vérifier que toutes les fonctionnalités de votre programme fonctionnent correctement.
- Bien que certains cas limites sont inclus dans les cas de test fournis (ou qui seront fournis) avec l'énoncé, testez les cas limites de manière rigoureuse.

6. Documentation :

- Assurez-vous que les commentaires dans le code améliorent la clarté.

Livrables

Code source :

Tous les fichiers .java organisés dans vos packages devront être soumis via StudiUM avant la date limite (9 mars 2025 à 23h59 ; sinon une pénalité de 20% par jour ouvrable dès leur première minute sera appliquée à votre note). Utilisez le `makefile` fourni avec l'énoncé pour créer une archive dont le nom sera `TP1_Member1_Member2.tar.gz`. Renommez le fichier en remplaçant `Member1` et `Member2` par les noms de familles des 2 membres de l'équipe. Si vous faites le TP seul, mettez votre nom à `Member1` et retirer `_Member2` du nom du fichier.

Critères d'évaluation

Principes de la POO (40%)

- Bonne utilisation de l'encapsulation, de l'héritage et du polymorphisme.

Fonctionnalité (50%)

- L'application respecte toutes les exigences spécifiées et passe les cas de test fournis avec l'énoncé (10%) et d'autres cas non fournis avec l'énoncé (40%).

Qualité du code (10%)

- Code propre, lisible et bien documenté avec une utilisation appropriée des commentaires.

Conseils pour réussir

- Commencez tôt : Débutez par la planification et la conception des classes pour bien comprendre le système.
- Développement incrémental : Construisez et testez votre application par petites étapes gérables.
- Demandez de l'aide en cas de blocage : N'hésitez pas à consulter vos camarades, vos enseignants ou des ressources en ligne si vous rencontrez des difficultés.
- Testez minutieusement : Vérifiez que toutes les fonctionnalités fonctionnent comme prévu et gérez correctement les entrées inattendues.

Bonne chance et bon codage ! 🎧🎶