Cégep André-Laurendeau

TP final : Un lecteur et un serveur de fichiers média.

Algorithmique et programmation
420-201-AL
Date de remise : 30 mai 2016

1 Objectifs

Voici les objectifs de ce travail :
- Compréhension de la programmation orientée objet (polymorphisme, héritage,...)
- Gestion d'exceptions
- Utilisation des entrées/sorties en Java

2 Présentation du problème

Vous devez implanter une version minimaliste de iTunes ou autre utilitaire semblable et un système client-serveur de fichiers média (mp3 ou mp4). Une interface graphique de base vous est fournie à laquelle vous devez ajouter les fonctionalités suivantes :
- la gestion des fichiers à faire jouer.
- la communication avec le serveur qui permettra à l'utilisateur d'écouter des médias qu'il téléchargera.
- la gestion des librairies locales, du serveur, et de la liste de fichiers à jouer (normalement appelée la "playlist" dans le jargon des utilitaires comme iTunes).

2.1 Le rôle du serveur

Le serveur contient un ensemble de fichiers qu'il peut envoyer à un client. Le serveur peut effectuer les opérations suivantes :
- envoyer des fichiers médias (mp3 ou mp4) à un client
- recevoir des fichiers médias d'un client désirant partager des fichiers à d'autres clients.

2.2 L'interface graphique

2.2.1 Son rôle

L'interface graphique est une interface client. Elle effectue les opérations suivantes :
- recevoir et afficher les informations des fichiers présent sur le serveur
- afficher les informations sur la librairie média locale (celle que possède le client sur sa machine)
- jouer en boucle une "playlist" de fichiers médias (mp3 ou mp4)
- assurer les échanges de fichiers (upload/download) entre le client et le serveur.

2.2.2 Comment lancer le gui ?

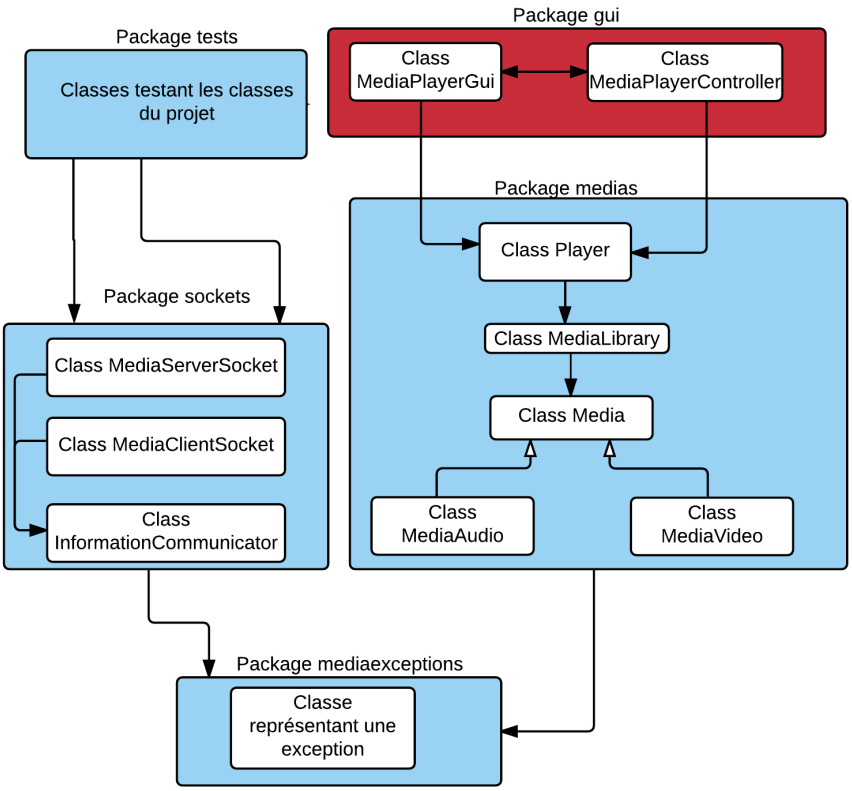
La fonction `main` lançant l'interface graphique est dans la classe `SuperMediaPlayer` du package `exe`. Voici comment la lancer à partir de la ligne de commande :

```
java exe.SuperMediaPlayer <fichierLibrairieLocale> [server address] [pPort]
```

où :
- `fichierLibrairieLocale` est le fichier de description de la librairie locale du client (voir section 6)
- `server addresse` est l'adresse ip du serveur. Cet argument est optionnel, si on souhaite lancer l'interface sans connexion au serveur.
- `pPort` est le port sur lequel on peut se connecter au serveur. Cet argument est optionnel, si on souhaite lancer l'interface sans connexion au serveur.

3 Les classes du projet

Voici le diagramme général de l'ensemble des classes formant le projet :



Les flèches pleines dans le diagramme indique qu'une classe en utilise une autre. Les flèches avec une pointe vide marque la relation d'héritage.

3.1 Les classes du package `medias`

Le package `medias` contient toutes les classes de base représentant chacun des fichiers des librairies locale et du serveur. Ce package contient aussi la classe `Player`, qui permet l'interaction avec l'interface graphique. Pour une description détaillée de chacune de ces classes, voir la section 4.1.

3.1.1 La classe `Media`

Cette classe représente un fichier média (audio ou vidéo). Elle contiendra toutes les informations décrivant le fichier :

— le titre ;
— l'année de création ;
— le nom du fichier .mp3 ou .mp4. avec son chemin ("path") complet sur le disque dur.

Cette classe contiendra aussi les fonctions nécessaires pour écrire les informations relatives à un média dans un flux (stream) donné.

3.1.2 La classe `MediaAudio`

Cette classe représente un fichier média audio. Elle hérite de la classe `Media`, donc elle contient les informations génériques à un fichier média en plus des spécificités reliées à un fichier audio :

— l'album ;
— l'artiste.

3.1.3 La classe `MediaVideo`

Cette classe représente un fichier média vidéo. Elle hérite de la classe `Media`, donc elle contient les informations génériques à un fichier média en plus des spécificités reliées à un fichier vidéo :

— la dimension des images ;
— la fréquence des images.

3.1.4 La classe `MediaLibrary`

Cette classe représente une librarie média dans le programme. L'interface graphique interagit avec trois différentes librairies :

— la librairie locale : celle décrivant l'ensemble des fichiers que possède l'utilisateur
— la librairie du serveur : celle décrivant l'ensemble des fichiers disponibles sur le serveur. Les fichiers présents sur le serveur peuvent être téléchargés pour être ajoutés à la liste courante ou la "playlist".
— la liste courante ou la "playlist" : celle décrivant l'ensemble des fichiers à jouer quand on lance la lecture à partir de l'interface graphique.

3.1.5 L'interface `MediaConstant`

Cette interface ne contient que des variables `static final`, donc des constantes, permettant l'identification les champs d'information des fichiers médias, les types de librairies (locale, sur le serveur ou la liste courante), les types de média (audio ou vidéo), ... Vous pouvez en ajouter si vous avez besoin d'autres constantes.
Vous devez utiliser ces constantes pour assurer une homogénéité dans le code.

3.2 Les classes du package `sockets`

Ce package contient toutes les classes relatives à la communication client-serveur.

3.2.1 La classe `MediaServerSocket`

Cette classe contient les flux formatés et binaires liant le serveur à son client. Elle contient aussi toutes les fonctions reliées à chacune des actions du serveur. Il y aura au moins une fonction pour chacune des opérations suivantes :

— l'initialisation de connexions avec le client
— la communication avec le client

3.2.2 La classe `MediaClientSocket`

Cette classe contient les flux formatés et binaires liant le client au serveur. Elle contient aussi toutes les fonctions reliées à chacune des actions requêtes que le client peut faire dans ce projet. Il y aura au moins une fonction pour chacun des services fournis par le serveur, comme par exemple :

— envoi d'un fichier au serveur
— réception d'un fichier du serveur
— réception des informations pour chacun des médias contenus dans librairie média du serveur
— ...

3.2.3 La classe `InformationCommunicator`

Cette classe contiendra des fonctions statiques qui correspondent à des opérations de communication faites autant par le serveur que le client. Par exemple, envoyer un fichier .mp3 du côté serveur et du côté client correspond à la même opération pour le serveur et le client. Cette classe est présente pour éviter la duplication de code dans les deux classes serveur et client.

3.3 Les classes du package `mediaexception`

Vous n'avez, au minimum, qu'une classe de votre choix à ajouter dans cette section. Vous devrez choisir un type d'exception que vous allez créer pour ce projet. Par exemple, vous pourriez ajouter l'exception `BadFieldMediaException` qui serait lancée lorsqu'on essaie de donner une dimension d'images à un fichier audio qui ne contient pas d'image. Choisissez le cas d'erreur que vous voulez, et créez une classe enfant de la classe `Exception` de l'api de Java.

3.4 Les classes du package `tests`

Ce package contiendra toutes les classes tests que vous avez créées pour mener à bien votre projet. Il devrait y avoir minimalement une classe testant le contenu du package `media` et deux autres testant les communications client-serveur.

3.5 Les classes du package `exe`

Ce package contiendra toutes les classes exécutables, i.e. contenant une fonction `main`.

3.5.1 La classe `SuperMediaPlayer`

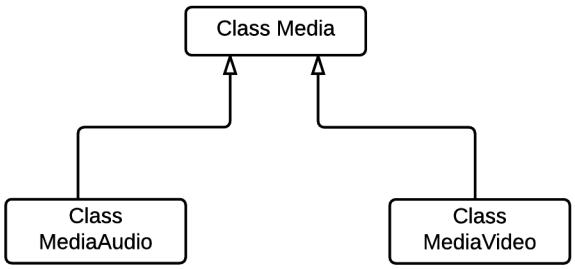
Cette classe permet de lancer l'interface graphique. Elle vous est fournie et vous n'avez pas à modifier quoique ce soit. Le lancement de cette classe est décrit en détail à la section 2.2

3.6 La classe `SuperMediaServer`

Cette classe permet de lancer le serveur responsable de recevoir des connexions avec le client. Basez-vous sur l'exemple de serveur fourni en classe pour compléter cette classe.

4 Partie 1 : Les classes du package `medias`

4.1 La hiérarchie de classes contenant `Media`, `MediaAudio` et `MediaVideo`



4.1.1 La classe `Media`

La classe `Media` devrait être **abstraite** puisqu'elle représente de façon générique un média. Les seules instances qu'on devrait pouvoir créer sont des classes `MediaAudio` et `MediaVideo`. À vous de décider quelles fonctions sont abstraites. **Expliquez votre choix en commentaires dans le code**. Il se peut qu'il manque des constructeurs, ou des fonctions, vous pourrez les ajouter en prenant soin de bien expliquer en commentaires leur rôle. Voici le diagramme de classe :

Classe Media
mFileName :String # mDuration :int # mYear :int # mTitle :String
+ Media(pFileName :String, pDuration :int, pYear :int,pTitle :String) + Media(pMedia :Media) + getFullArrayInfo : String[] + setFileName(pFileName : String) + getFileName() : String + getDuration() : int + getTitre() : String + getYear() : int + toString() : String

Voici une description du rôle de chacune des variables membres :

- `String mFileName` : une instance de la classe Media contient les informations relatives à un fichier contenu dans une librairie. La variable `mFileName` contient l'endroit où le fichier (`mp3` ou `mp4`) est localisé sur le disque dur. Elle contient donc le chemin complet vers le fichier relié à l'instance de la classe `Media`.
- `int mDuration` : c'est la durée, en secondes, de l'instance.
- `int mYear` : l'année de création de l'instance
- `int mTitle` : le titre de l'instance.

Voici une description de chacune des fonctions :

- `Media(String pFileName, int pDuration, int pYear, String pTitle)` : constructeur de la classe prenant en paramètre les valeurs des différentes variables membre de cette classe.
- `Media(Media pMedia)` : constructeur permettant de créer une copie de l'instance pMedia passée en paramètre. Ce constructeur pourrait vous être utile lorsque vous voudrez transférer des fichiers du serveur vers le client, ou du client vers le serveur.
- `String[] getFullArrayInfo` : cette fonction permet de retourner, sous la forme d'un tableau de chaînes de caractères, les différents champs définissant le média. Vous trouverez dans l'interface `MediaConstants` les valeurs suivantes :

```
int FILE_NAME = 5;
int DURATION = 2;
int YEAR = 1;
int TITLE = 0;
int ARTIST = 3;
int ALBUM = 4;
int FRAME_RATE = 3;
int DIMENSION = 4;
```

Ces constantes indiquent à quel index du tableau devrait se trouver chacun des champs décrivant un média. Ainsi, si on a un fichier audio, le tableau retourné devrait être le suivant :

mTitle	mYear	mDuration	mArtist	mAlbum	mFileName

et dans le cas d'un fichier vidéo, ce tableau devrait être :

mTitle	mYear	mDuration	mFrameRate	"mWidthXmHeight"	mFileName

Remarquez que dans les deux cas, les informations sont placés selon l'index défini dans la classe MediaConstant. **C'est absolument nécessaire que vous respectiez cet ordre, sinon les informations ne seront pas bien affichées dans l'interface graphique**. Utilisez les constantes pour déterminer les index des différentes valeurs plutôt que de "hardcoder" les index dans le code.

- `void setFileName(pFileName: String)` : cette fonction permet de relocaliser un média. Vous vous en servirez sûrement lorsque vous téléchargerez des fichiers de et vers le serveur. Le chemin sur le disque dur du serveur ne devrait pas être le même chez le client, et de même dans l'autre direction.
- `String getFileName()` : fonction d'accès au chemin du fichier relié à l'instance média.
- `int getDuration()` : fonction d'accès à la durée de l'instance
- `String getTitre()` : fonction d'accès au titre de l'instance
- `int getYear()` : fonction d'accès à l'année de création du média
- `String toString()` : fonction permettant de retourner une représentation sous forme de `String` de l'instance. Voir la section 6 pour savoir exactement quel sera le format de cette chaîne de caractères.

4.1.2 La classe `MediaAudio`

La classe `MediaAudio` est une sous-classe de la classe `Media`. Voici le diagramme de classe de cette classe :

Classe MediaAudio
mArtist :String # mAlbum :String
+ MediaAudio(pFileName :String, pDuration :int, pYear :int, pTitle :String, pArtist :String, pAlbum :String) + MediaAudio(pMedia :MediaAudio) + MediaAudio(pScan :Scanner) + getArtiste() :String + getAlbum() :String

Voici une description du rôle de chacune des variables membres :

- `String mArtiste` : l'artiste relié à l'instance
- `String mAlbum` : l'album relié à l'instance.

Voici une description de chacune des fonctions :

- `MediaAudio(String pFileName, int pDuration, int pYear, String pTitle,String pArtist, String pAlbum)` : constructeur de la classe prenant en paramètre les valeurs des différentes variables membre de cette classe.

— `MediaAudio(pMedia:MediaAudio)` : constructeur permettant de créer une copie de l'instance pMedia passée en paramètre. Ce constructeur pourrait vous être utile lorsque vous voudrez transférer des fichiers du serveur vers le client, ou du client vers le serveur.
— `MediaAudio(Scanner pScan)` : constructeur permettant de créer une instance en lisant les différentes valeurs des champs d'information dans un flux. Je vous suggère ici d'utiliser la classe Scanner, mais ce n'est pas obligatoire. Référez-vous à la section 6 pour arriver à compléter ce constructeur. Le format de description sous forme texte devrait vous aider à compléter le code de ce constructeur.
— `String getArtiste()` : fonction d'accès à l'artiste relié à l'instance.
— `String getAlbum()` : fonction d'accès à l'album relié à l'instance.

Attention !!! Vous devez penser à surdéfinir les fonctions héritées qui ne conviennent pas exactement à cette classe.

4.1.3 La classe `MediaVideo`

La classe `MediaVideo` est une sous-classe de la classe `Media`. Voici le diagramme de classe de cette classe :

Classe MediaVideo
mFrameRate :int # mWidth :int # mHeight :int
+ MediaVideo(pFileName :String, pDuration :int, pYear :int, pTitle :String, pFrameRate :int, pWidth :int, pHeight :int) + MediaVideo(pMedia :MediaVideo) + MediaVideo(Scanner pScan) + getFrameRate() :int + getHeight() :int + getWidth() :int

Voici une description du rôle de chacune des variables membres :
— `int mFrameRate` : le nombre d'images par seconde dans le vidéo
— `int mWidth` : la largeur des images du vidéo
— `int mHeight` : la hauteur des images du vidéo

Voici une description de chacune des fonctions :
— `MediaVideo(String pFileName, int pDuration, int pYear, String pTitle,int pFrameRate, int pWidth, int pHeight)` : constructeur de la classe prenant en paramètre les valeurs des différentes variables membre de cette classe.
— `MediaVideo(MediaVideo pMedia)` : constructeur permettant de créer une copie de l'instance pMedia passée en paramètre. Ce constructeur pourrait vous être utile lorsque vous transférerez des fichiers du serveur vers le client, ou du client vers le serveur.

— `MediaVideo(Scanner pScan)` : constructeur permettant de créer une instance en lisant les différents champs dans un flux. Je vous suggère ici d'utiliser la classe Scanner, mais ce n'est pas obligatoire, si vous préférez utiliser une autre classe de l'api de java. Référez-vous à la section 6 pour arriver à compléter ce constructeur. Le format de description sous forme texte devrait vous aider à compléter le code de ce constructeur.
— `String getFrameRate()` : fonction d'accès au nombre d'images/seconde dans le vidéo.
— `int getWidth()` : fonction d'accès à la largeur des images.
— `int getHeight()` : fonction d'accès à la largeur des images.

Attention ! ! ! Vous devez penser à surdéfinir les fonctions héritées qui ne conviennent pas exactement à cette classe.

4.2 La classe `MediaLibrary`

Cette classe représente une librairie de fichiers médias. Il existe trois librairies manipulées par l'interface graphique :

1. la librairie locale, celle qui se trouve sur l'ordinateur du client.
2. la librairie qui se trouve sur le serveur
3. la playlist qui est en fait la liste de fichiers à faire jouer par l'interface graphique.

Elle est en fait un groupement d'instances des classes `MediaAudio` et `MediaVideo`. Voici le diagramme de cette classe :

Classe MediaLibrary
mMediaList :Vector<Media>
+ MediaLibrary() + MediaLibrary(pScan :Scan) + remove(pIdx :int) + add(pMedia :Media) + get(pIdx :int) :Media + size() :int + getBasicFieldInfo() :String[] + getAudioFieldInfo() :String[] + getVideoFieldInfo() :String[] + getBasicLibraryInfo() :String[][] + getFullMediaInfo(pIdx :int) :String[] + toString() :String

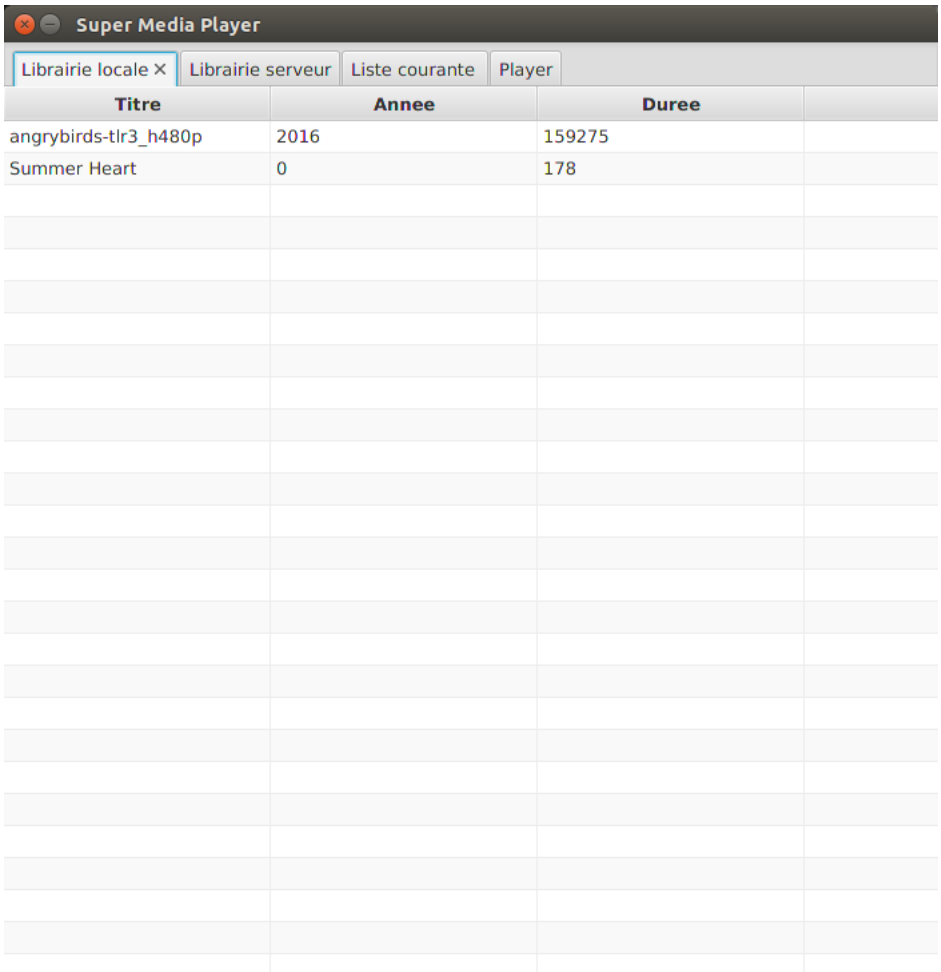
Voici une description du rôle de chacune des variables membres :

— `Vector<Media> mMediaList` : c'est un vecteur contenant l'ensemble des instances de la classe `Media` contenues dans la librairie.

Voici une description de chacune des fonctions :

- `MediaLibrary()` : constructeur vide de la classe créant une liste de médias vide.
- `MediaLibrary(Scan pScan)` : constructeur de la classe permettant d'initialiser la liste de médias à partir du contenu d'un flux. Voir la section 6 pour déterminer comment coder cette fonction.
- `remove(int pIdx)` : cette fonction permet d'effacer le fichier à l'index `pIdx`. Elle ne fait rien dans le cas où l'index est invalide.
- `add(Media pMedia)` : ajoute un nouveau média à la librairie.
- `Media get(int pIdx)` : fonction d'accès au média se trouvant à l'index `pIdx` de la librairie. Si l'index est invalide, la fonction retourne `null`.
- `size()` : fonction d'accès à la taille de la librairie (i.e. le nombre de médias se trouvant dans la librairie).

Les fonctions suivantes sont utilisées dans le cadre de l'affichage de l'interface graphique. Sur l'interface, on voit, pour chacun des médias présents dans chacune des librairies, des informations de base apparaître, soit dans l'image suivante :

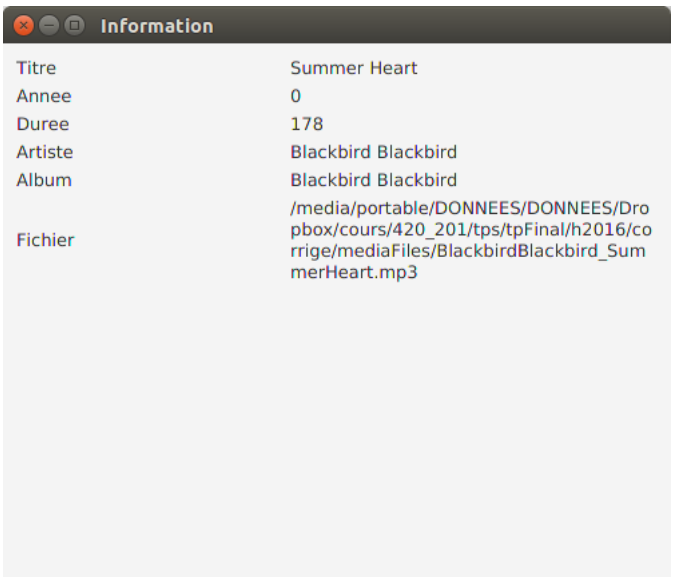


On voit, pour les deux fichiers de la librairie locale, les informations suivantes :

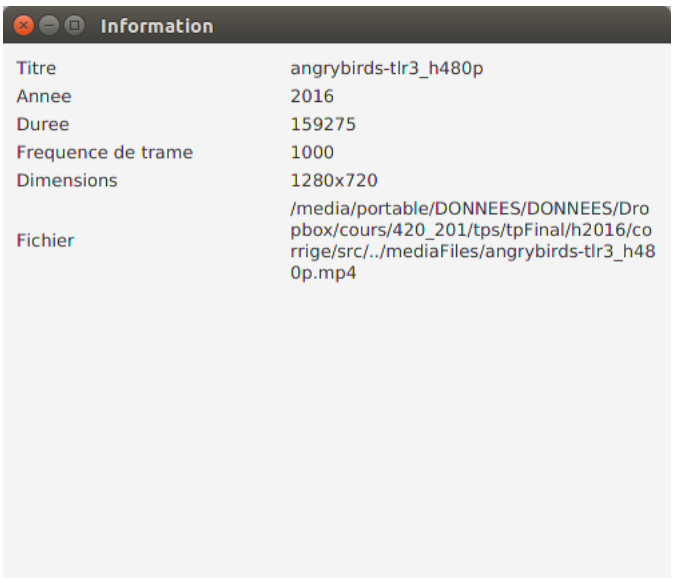
1. le titre

2. l'année
3. la durée

Ces informations sont les **informations de base** de chacun des fichiers média des librairies. Quand on fait un clique droit sur un des items d'une librairie, un menu contextuel apparaît. Si on choisit l'option `Information sur le média`, une boîte de dialogue apparaît, affichant les **informations complètes** du média. Par exemple, pour le cas d'un fichier audio, la boîte de dialogue serait la suivante :



Et pour un fichier vidéo, la boîte de dialogue serait plutôt la suivante :



— `String[] getBasicFieldInfo()` : retourne le nom des champs (le titre des colonnes) à afficher dans le cas des informations de base, affichées par défaut dans le gui. Les

champs seront retournés sous forme de tableau de chaînes de caractères. Ils seront les titres de chacune des colonnes définissant un média dans le gui. Il est préférable de respecter l'ordre des champs définis dans l'interface `MediaConstant`. Ces champs devraient être {Titre, Année, Durée}.

— `String[] getAudioFieldInfo()` : retourne le nom de tous les champs à afficher pour les informations détaillées d'un média audio. Il est préférable de respecter l'ordre des champs définis dans l'interface `MediaConstant`. Ces champs sont {Titre, Année, Durée, Artiste, Album, Fichier}.

— `String[] getVideoFieldInfo()` :retourne le nom de tous les champs à afficher pour les informations détaillées d'un média vidéo. Il est préférable de respecter l'ordre des champs définis dans l'interface `MediaConstant`. Ces champs sont {Titre, Année, Durée, Fréquence de trame, Dimensions, Fichier}.

— `String[][] getBasicLibraryInfo()` : retourne un tableau qui aura trois colonnes, soit les trois champs (titre, année, durée) constituant les informations de base de chacun des médias de la librairie. Ainsi, si votre librairie contient 5 médias, le tableau retourné sera sous la forme :

titre1	annee1	duree1
titre2	annee2	duree2
titre3	annee3	duree3
titre4	annee4	duree4
titre5	annee5	duree5

Le tableau retourné sera de dimensions `taille de la librairie X 3`

— `String[] getFullMediaInfo(pIdx:int)` : cette fonction retourne l'information complète reliée au média de l'index `pIdx`. `null` si `pIdx` est invalide. Pensez à utiliser la fonction `getFullMediaInfo` de la classe `Media`.

4.3 La classe `Player`

La classe `Player` est le coeur de votre application. Elle communique avec l'interface graphique, est responsable des communications avec le serveur,... Le squelette de cette classe est disponible dans le code de base fourni avec cet énoncé. Je vous suggère de la faire en deux parties :

1. assurer le maintien de la librairie locale et de la liste courante (playlist).
2. ajouter les communications avec le serveur lorsque les classes du package `sockets` seront complétées et testées.

Voici le diagramme de cette classe :

Classe Player
mLocalLib :MediaLibrary # mServerLib :MediaLibrary # mPlayList :MediaLibrary # mCurrentPlayed :int # mClient :MediaSocketClient # TMP_DIRECTORY :String
+ Player(pFileNameLocalLib :String) + Player(pFileNameLocalLib :String, pServerAddress :String, pServerPort :int) + deleteFromPlaylist(pIdx :int) + getCurrentFileNamePlayer() :String + getFileNameFromMedia(pTypeLib :int, pIdx :int) :String + next() + previous() + getCurrentPlayed() :int + setCurrentPlayed(pCurrentPlayed :int) + addToPlayList(pLibrarySource :int, pIdx :int) + addToServer(pIdx :int) + getSizeLibrary(pTypeLib :int) :int + getBasicLibraryFieldInfo() :String[] + getBasicLibraryInfo(pTypeLib :int) :String[][] + getFullFieldMediaInfo(pTypeLib :int, pIdx :int) :String[] + getFullMediaInfo(pTypeLib :int, pIdx :int) :String[] + close()

Voici une description pour chacune des variables de cette classe :

- `MediaLibrary mLocalLib` : la librairie locale que possède l'utilisateur du programme.
- `MediaLibrary mServerLib` : la librairie disponible sur le serveur.
- `MediaLibrary mPlayList` : la playlist qui se remplira au fur et à mesure de l'exécution de l'interface graphique. C'est par l'interface graphique que l'utilisateur peut ajouter des fichiers à la playlist.
- `int mCurrentPlayed` : le fichier qui est présentement en train de jouer. Cet information est reçue par l'interface graphique, via la fonction `setCurrentPlayed`.
- `MediaSocketClient mClient` : un objet qui permet une connexion et des interactions avec le serveur.
- `String TMP_DIRECTORY` : Ce répertoire sert à stocker temporairement les fichiers téléchargés du serveur. Lorsqu'un utilisateur ajoute un fichier du serveur à la liste courante (playlist), ce fichier est téléchargé et sauvegardé dans ce répertoire. Lorsque la connexion est terminée, le contenu de ce répertoire est effacé. Les clients ne sont pas propriétaires des fichiers du serveur !

Voici une description de chacune des fonctions :

- `Player(String pFileNameLocalLib)` : Constructeur de la classe prenant en paramètre un fichier texte décrivant la librairie local de l'utilisateur. Le format de ce fichier est décrit dans la section 6. Ce constructeur vous permet de créer une instance de la classe `Player` sans connexion avec un serveur. Voici quelques conseils pour bien compléter ce constructeur :
  - s'il n'y a pas de connexion au server, la librairie `mServerLib` sera vide.
  - par défaut, `mPlayList` est toujours vide à la création d'une nouvelle instance de la classe `Player`, et se remplira via l'interface graphique.
  - puisqu'aucun fichier média n'a été sélectionné encore pour jouer, la variable `mCurrentPlayed` aura pour valeur -1.
- `Player(String pFileNameLocalLib, String pServerAddress, int pServerPort)` : Constructeur de la classe permettant une connexion avec un serveur dont l'adresse ip est `pServerAddress` et le port est `pServerPort`. Le fichier de description de la librairie locale est `pFileNameLocalLib`. Voici les opérations à compléter dans ce constructeur :
  1. on charge le contenu du fichier `pFileNameLocalLib`, afin d'initialiser la variable `mLocalLib`.
  2. on tente une connexion avec le serveur à l'adresse `pServerAddress` et au port `pServerPort`.
     - Si la connexion réussit, on recevra les informations de chacun des fichiers médias disponibles sur le serveur. Voir la section 9 pour plus d'information.
     - Si la connexion ou la réception des informations échoue, la variable `mServerLib` sera une librairie vide.

     La variable `mClient` sera initialisée lors de la connexion, et sera `null` si la connexion échoue.
  3. la variable `mPlayList` sera une librairie vide à la création du "player".
  4. la variable `mCurrentPlayed` aura pour valeur -1 puisqu'aucun fichier n'est en lecture lors de la création du "player".
- `void deleteFromPlaylist(int pIdx)` : permet de retirer le média à l'index `pIdx`. Vous n'avez pas à gérer si l'index effacé est celui en cours de lecture, c'est géré au niveau du gui. Si `pIdx` est invalide, l'appel à cette fonction n'aura aucun impact.
- `String getCurrentFileNamePlayer()` : fonction d'accès au chemin du fichier (variable `mFileName` de la classe `Media`) du média en cours de lecture.
- `String getFileNameFromMedia(int pTypeLib, int pIdx)` : retourne le nom du fichier qui se trouve dans la librairie dont le type correspond à `pTypeLib`, à l'index `pIdx`. La valeur de `pTypeLib` correspondra à une des constantes de l'interface `MediaConstant` soient :
  - `MediaConstant.LOCAL`
  - `MediaConstant.SERVER`
  - `MediaConstant.PLAYLIST`

  Par exemple, si `pTypeLib = MediaConstant.LOCAL`, et `pIdx = 0`, la fonction retournera le nom du fichier qui se trouve à l'index 0 de la librairie `mLocalLib`. Cette fonction retourne `null` si `pIdx` ou `pTypeLib` est invalide.

- `void next()` : permet de changer l'index du fichier en cours de lecture pour aller au prochain dans la playlist. Si on est au dernier index, on recommence au début de la playlist.
- `void previous()` : permet de changer l'index du fichier en cours de lecture pour aller au précédent dans la playlist. Si on est au premier index, on recommence à la fin de la playlist.
- `int getCurrentPlayed()` : fonction d'accès à la variable `mCurrentPlayed`.
- `void setCurrentPlayed(int pCurrentPlayed)` : fonction permettant de changer la valeur de la variable `mCurrentPlayed`
- `void addToPlayList(int pLibrarySource, int pIdx)` : permet d'ajouter un média à la playlist courante. Le paramètre `pLibrarySource` indique de quelle librairie le média à ajouter provient, et le paramètre `pIdx` est l'index du média à ajouter. `pLibrarySource` peut prendre l'une ou l'autre des valeurs suivantes :
  - `MediaConstant.LOCAL` : on ajoute un média provenant de la librairie locale. Dans ce cas, il faudra simplement ajouter l'instance de la classe `Media` se trouvant à l'index `pIdx` de `mLocalLib` à la playlist.
  - `MediaConstant.SERVER` : on ajoute un média provenant du serveur à la playlist. Dans ce cas, il faudra :
    1. il faudra ajouter une copie de l'instance de la classe `Media` se trouvant à l'index `pIdx` de `mServerLib` à la playlist
    2. télécharger le fichier .mp3 ou .mp4 rattaché du serveur et sauvegarder le fichier reçu dans le répertoire `TMP_DIRECTORY`
    3. modifier le nom de fichier de l'instance de la classe `Media` ajoutée à la playlist. Le nouveau nom de fichier correspondra à son emplacement du côté client. Normalement, il devrait se trouver dans le répertoire `TMP_DIRECTORY`.

    Pour plus de détails sur le protocol de communication entre le client et le serveur, voir la section 9.
- `void addToServer(int pIdx)` : permet d'envoyer un fichier de la librairie locale au serveur. Le fichier qui sera envoyé sera celui rattaché à l'instance de la classe `Media` se trouvant à l'index `pIdx` de la librairie locale. Pour plus de détails sur le protocol de communication entre le client et le serveur, voir la section 9.
- `int getSizeLibrary(pTypeLib:int)` : permet d'obtenir la taille d'une des librairies du player. Le paramètre `pTypeLib` aura pour valeur :
  - `MediaConstant.LOCAL` : pour la taille de la librairie locale
  - `MediaConstant.SERVER` : pour la taille de la librairie du serveur
  - `MediaConstant.PLAYLIST` : pour la taille de la playlist.
- `String[] getBasicLibraryFieldInfo()` : retourne les champs d'informations de base pour les médias à afficher dans l'interface graphique. Cette fonction fait simplement appel à la fonction `getBasicFieldInfo` de la classe `MediaLibrary`
- `String[][] getBasicLibraryInfo(int pTypeLib)` : retourne les informations de base pour chacun des médias contenus dans la librairie de type `pTypeLib` qui aura pour valeur :

- `MediaConstant.LOCAL` : pour les informations des médias de la liste locale
- `MediaConstant.SERVER` : pour les informations des médias de la liste du serveur
- `MediaConstant.PLAYLIST` : pour les informations des médias composant la playlist.

Cette fonction fait appel à la fonction `getBasicLibraryInfo` de la classe `MediaLibrary`.

- `String[] getFullFieldMediaInfo(pTypeLib:int, pIdx:int)` : cette fonction permet d'accéder aux champs d'informations disponibles pour un média donné. Les paramètres `pTypeLib` et `pIdx` permettent respectivement de spécifier de quelle librairie (Media.LOCAL, Media.SERVER ou Media.PLAYLIST) et à quel index est le média donné. Si le média est de type audio, les champs seront ceux retournés par la fonction `getAudioFieldInfo()`, et s'il est un vidéo, les champs seront ceux retournés par la fonction `getVideoFieldInfo()` de la classe `MediaLibrary`.
- `String[] getFullMediaInfo(pTypeLib:int, pIdx:int)` : cette fonction permet d'accéder à l'ensemble des informations disponibles pour un média donné. Les paramètres `pTypeLib` et `pIdx` permettent respectivement de spécifier de quelle librairie (Media.LOCAL, Media.SERVER ou Media.PLAYLIST) et à quel index est le média donné. La fonction retourne le résultat de la fonction `getFullMediaInfo` de la classe `MediaLibrary`.
- `close()` : cette fonction est appelée lors de la fermeture du gui. Elle sert à nettoyer les ressources réservées par l'instance de la classe `Player`. Cette fonction effectue les opérations suivantes :
  1. ferme la connexion avec le serveur. Voir la section 9 pour plus d'information sur le protocole suggéré.
  2. efface tous les fichiers qui se trouvent dans le répertoire `TMP_DIRECTORY`, qui ont été téléchargés du serveur. Voir la section 7 pour avoir la marche à suivre.

4.4 Les tests à faire pour cette partie

Pour arriver à bien tester les classes `Media` et `MediaLibrary`, basez-vous sur la classe `TestPersonne` présentée en classe.

5 Partie 2 : Les classes du package `sockets`

Pour cette seconde partie, les indications suivantes sont à titre indicatif seulement. La structure suggérée a été testée et fonctionne bien. Elle vous est donnée de façon très générique, il vous faudra donc penser au modèle de conception. Pour vous aider dans cette tâche, inspirez-vous de l'exemple de client-serveur partageant un carnet d'adresse.

5.1 La classe MediaClientSocket

Cette classe représente un client permettant de se connecter à un serveur de fichier média. Elle devrait avoir les variables membres suivantes :

- une instance de la classe `Socket` qui est la connexion vers le serveur
- deux instances permettant d'écrire des données sur le socket :
  - vous utiliserez une instance de la classe `PrintWriter` pour envoyer des données formatées. Si vous avez utilisé une autre classe pour implanter les classes du package `medias`, utilisez la même classe.
  - vous utiliserez une instance de la classe `DataOutputStream` pour être en mesure de télécharger ("uploader") vers le serveur des fichiers médias qui doivent être envoyés sous forme binaire.
- deux instances permettant de lire des données sur le socket :
  - vous utiliserez une instance de la classe `Scanner` pour recevoir des données formatées. Si vous avez utilisé une autre classe pour implanter les classes du package `medias`, utilisez la même classe.
  - vous utiliserez une instance de la classe `DataInputStream` pour être en mesure de télécharger ("downloader") du serveur des fichiers médias qui doivent être reçus sous forme binaire.

Pour déterminer les fonctions qui seront nécessaires dans cette classe, référez vous à la section 9. Normalement, chacun des ordres à transmettre au serveur devrait correspondre à une fonction de votre classe.

5.2 La classe MediaServerSocket

Cette classe représente un serveur de partage de fichiers médias. Elle devrait avoir les variables membres suivantes :

- une instance de la classe `ServerSocket` qui est le socket permettant aux clients de se connecter.
- une instance de la classe `Socket` qui est le socket vers un client connecté.
- deux instances permettant d'écrire des données sur le socket client :
  - vous utiliserez une instance de la classe `PrintWriter` pour envoyer des données formatées. Si vous avez utilisé une autre classe pour implanter les classes du package `medias`, utilisez la même classe.
  - vous utiliserez une instance de la classe `DataOutputStream` pour être en mesure de télécharger ("uploader") vers le client des fichiers médias qui doivent être envoyés sous forme binaire.
- deux instances permettant de lire des données sur le socket :
  - vous utiliserez une instance de la classe `Scanner` pour recevoir des données formatées. Si vous avez utilisé une autre classe pour implanter les classes du package `medias`, utilisez la même classe.
  - vous utiliserez une instance de la classe `DataInputStream` pour être en mesure de télécharger ("downloader") du client des fichiers médias qui doivent être reçus sous forme binaire.
- Une instance de la classe `MediaLibrary` qui contient toutes les informations des médias que le serveur possède.

Inspirez-vous de l'exemple présenté en classe pour déterminer les fonctions nécessaires au fonctionnement du serveur.

5.3 La classe `InformationCommunicator`

Cette classe vous est suggérée afin d'éviter de dupliquer du code dans les classes `MediaClientSocket` et `MediaServerSocket`. En effet, plusieurs opérations sont faites autant du côté client que du côté serveur, et correspondent au même code. La classe `InformationCommunicator` servirait d'utilitaire pour les classes clients et serveurs, et ne contiendrait que des fonctions `static` pour éviter d'avoir à instancier des objets de la classe. Voici quelques exemples de fonctions qui pourraient s'y trouver :

- `sendFile` : fonction permettant d'envoyer un fichier .mp3 ou .mp4 au serveur ou au client.
- `receiveFile` : fonction permettant de recevoir un fichier .mp3 ou .mp4 au serveur ou au client.

Ces deux opérations sont faites du côté serveur et du côté client, il est donc préférable qu'une seule fonction remplisse ce rôle. Cela rendra plus clair votre code et plus propre, en plus de vous aider à débugger.

En principe, aussitôt que vous êtes en train de refaire une même opération dans le client et le serveur, cette opération devrait devenir une fonction de la classe `InformationCommunicator` et sera appelée par le client et le serveur.

Une pénalité sera appliquée si le même code est dupliqué du côté client et du côté serveur.

5.4 La classe `SuperMediaServer` à ajouter au package `exe`

En vous basant sur l'exemple vu en classe, vous devrez founir une classe contenant une fonction `main` qui lancera le serveur et gérera les connexions de clients. Le serveur devrait se lancer comme suit :

```
java SuperMediaServer <port> <fichier des informations de la librairie>
<répertoire dans lequel sauvegarder les fichiers téléchargés>
```

où :

- `port` : est le port sur lequel on peut se connecter au serveur.
- `fichier des informations de la librairie` : fichier texte contenant les informations reliés à chacun des médias disponibles sur le serveur.
- `répertoire dans lequel sauvegarder les fichiers téléchargés` : répertoire dans lequel on sauvegardera les fichiers envoyés par le client.

5.5 Tests à faire pour les classes du package `sockets`

Pour arriver à bien tester, surtout ne vous lancez pas dans l'incorporation des sockets avec le gui. Faites vous d'abord un serveur/client en mode terminal, tel que présenté dans

l'exemple de carnet d'adresse présenté en classe. Vous n'avez pas à faire un client qui fournit un menu, mais seulement un client qui teste chacun des services du serveur. Par exemple, votre test, du côté client pourrait compléter les opérations suivantes :

1. Connexion au serveur
2. Réception de la librairie de média du serveur
3. Envoi d'un fichier au serveur
4. Réception d'un fichier du serveur
5. Quitte le serveur

Si toutes ces opérations fonctionnent, vous devriez être en mesure de plus facilement intégrer les sockets à l'interface graphique.

6 Format de représentation des médias sous forme texte

Afin de pouvoir créer des instances de la classe `MediaLibrary`, on utilise un fichier texte décrivant l'ensemble de son contenu. Chacun des médias présent dans la librairie sera décrit dans ce fichier texte, dans un format bien précis. Chacune des descriptions sera espacée d'une ligne vide pour indiquer la fin des informations sur un média. Ainsi, pour chacun des médias, on aura dans le fichier de description de la librairie :

```
0 (pour un fichier vidéo) ou 1 (pour un fichier audio)
FILE_NAME
<chemin vers le fichier rattaché au média>
DURATION
<durée du média>
YEAR
<année de création du média>
TITLE
<titre du média>
FRAME_RATE
<nombre d'images secondes>:seulement pour les vidéos
HEIGHT
<hauteur des images>:seulement pour les vidéos
WIDTH
<largeur des images>:seulement pour les vidéos
ARTIST
<nom de l'artiste>:seulement pour les fichiers audios
ALBUM
<nom de l'album>:seulement pour les fichiers audios
```

Voici un exemple pour deux médias, un vidéo et un audio.

```
0
FILE_NAME
/media/portable/DONNEES/DONNEES/Dropbox/cours/420_201/tps/tpFinal/h2016/corrige/src/../
mediaFiles/angrybirds-tlr3_h480p.mp4
DURATION
159275
YEAR
2016
TITLE
angrybirds-tlr3_h480p
FRAME_RATE
1000
HEIGHT
720
WIDTH
1280

1
FILE_NAME
/media/portable/DONNEES/DONNEES/Dropbox/cours/420_201/tps/tpFinal/h2016/corrige/mediaFil
BlackbirdBlackbird_SummerHeart.mp3
DURATION
178
YEAR
0
TITLE
Summer Heart
ARTIST
Blackbird Blackbird
ALBUM
Blackbird Blackbird
```

Le premier bloc correspond à une vidéo. La première ligne qui commence à le décrire contient la valeur 0 ou (MediaConstant.VIDEO) pour indiquer que le média décrit est une vidéo. Ensuite, on donne le nom du champ et sa valeur, et ce pour toutes les informations disponibles pour ce média.
À la fin de la description du média contenu dans le fichier `angrybirds-tlr3_h480p.mp4`, il y a une ligne vide, et la description du second fichier commence après cette ligne vide.
La description du second média commence par la valeur 1 (ou MediaConstant.AUDIO) pour indiquer que le média décrit est un fichier audio. Ensuite, on donne le nom du champ et la valeur attachée pour chacune des informations disponibles pour ce média.
Ce format de représentation des données vous est imposé puisque c'est le format qui sera

utilisé par l'utilitaire `LibraryDescriptorGenerator`. Ce dernier vous permettra d'extraire les informations relatives à un ensemble de fichiers médias et sauvegardera les informations extraites sous ce format. Voir section 8 pour savoir comment utiliser cet outil qui vous est fourni.
Ce même format devrait être utilisé quand le serveur échange des informations sur sa librairie avec le client. Voir section 9.

6.1 Conseils pour vous faciliter la tâche

— la fonction `toString()` des classes `MediaAudio` et `MediaVideo` devrait respecter ce format. Ainsi, pour la classe `MediaAudio`, on devrait retourner une chaîne de caractères qui aurait le format :
`FILE_NAME`
`mFileName`
`DURATION`
`mDuration`
`YEAR`
`mYear`
`TITLE`
`mTitle`
`ARTIST`
`mArtist`
`ALBUM`
`mAlbum`
Le même principe devrait s'appliquer à la classe `MediaVideo`

— la fonction `toString()` de la classe `MediaLibrary` devrait respecter ce format. Ainsi, cette fonction devrait retourner une chaîne de caractères correspondant au format :
`Pour chaque média m de la librairie:`
`"MediaConstant.VIDEO ou MediaConstant.AUDIO`
`m.toString()`
`<ligne vide>"`

7 Comment effacer les fichiers contenus dans un répertoire ?

Voici un bout de code permettant d'effacer les fichiers contenus dans le répertoire `"repertoire"`.

```
File dir = new File(TMP_DIRECTORY);
for(File file: dir.listFiles())
        file.delete();
```

À vous de fouiller l'api pour déterminer les packages à importer et les exceptions qui pourraient être lancées.

8 Utilisation du programme `LibraryDescriptorGenerator`

Afin de pouvoir générer votre propre librairie locale, votre gentil professeur vous fournit un outil qui effectue les tâches suivantes :

1. extrait les informations relatives à tous les fichiers se trouvant dans un répertoire donné qui ont comme extension : ".mp3" et ".mp4"
2. génère un fichier texte dont le format respecte celui décrit dans la section 6.

Pour arriver à l'utiliser, dézippez l'archive `LibraryDescriptionGenerator.zip` . Placez votre terminal dans le répertoire contenant les fichiers de l'archive. L'utilitaire est dans le fichier `LibraryDescriptionGenerator.jar` et se lance de la façon suivante :

```
java -jar LibraryDescriptionGenerator.jar <repertoire contenant vos fichiers médias> <fichier de sortie>
```

Ainsi, si on lance le programme de la façon suivante :

```
java -jar LibraryDescriptionGenerator.jar ../mediaFiles/  libLocal.txt
```

le programme générera le fichier `libLocal.txt` qui contiendra les informations relatives à chacun des fichiers `.mp3` et `.mp4` du répertoire `../mediaFiles/`. Attention ! Il se peut que certains fichiers ne soient pas lisibles pour l'outil que je vous fournis. Faites-moi savoir, en m'envoyant le fichier en question, et j'essaierai de voir ce que je peux faire. Il existe plusieurs types de standards pour les informations relatives à des fichiers audio et vidéo, et la librairie utilisée pour faire cette outil n'est pas très bien mise à jour. Il se peut que certains fichiers ne soient pas compatibles. Dans le pire des cas, vous pourrez éditer partiellement le fichier de sortie pour qu'il convienne à vos fichiers.

9 Protocole de communication entre le serveur et le client

Un protocole de communication vous est très très fortement suggéré ici. Il a été testé et sa faisabilité a été testée. Voici, pour chacun des services proposés par le serveur, le protocol suggéré :

9.1 Envoi des informations de la librairie de médias du serveur

Client	Serveur
Envoi de la constante `ProtocolConstant.LIB_INFO`	
	Reçoit la demande `ProtocolConstant.LIB_INFO`
	Écrit sur le socket client la librairie (MediaLibrary) du serveur, en respectant le format décrit dans la section 6
Tant que le client ne reçoit pas la constante `ProtocolContant.DONE`, il lit les informations reçues du serveur	
	Envoie la constante `ProtocolContant.DONE` pour indiquer que toutes les informations ont été envoyées.

9.2 Envoi d'un fichier du client vers le serveur

Client	Serveur
Envoi de la constante `ProtocolConstant.UPLOAD`	
	Reçoit la demande `ProtocolConstant.UPLOAD`
Envoi les informations reliées au média qu'on s'apprête à télécharger sur le serveur	
	Reçoit les informations reliées au média qu'on recevra
Envoi de la taille du fichier `nbBytes` pour que le serveur sache combien de bytes il téléchargera	
	Réception de la valeur de `nbBytes` du fichier
Envoi du fichier .mp3 ou .mp4 en utilisant un flux de binaires	
	Lecture de `nbBytes` sur le socket et stockage du fichier reçu dans le répertoire contenant la librairie

Voici un bout de code permettant d'obtenir la taille d'un fichier :

```
File f = new File("chanson.mp3");
long taille = f.length();
```

9.3 Envoi d'un fichier du serveur vers le client

Client	**Serveur**
Envoi de la constante `ProtocolConstant.REQ_FILE`	
	Reçoit la demande `ProtocolConstant.REQ_FILE`
Envoi le nom du fichier (variable `mFileName` de la classe `Media`) désiré par le client	
	Réception du nom du fichier demandé
	Envoi de la taille du fichier `nbBytes` pour que le serveur sache combien de bytes il téléchargera
Réception de la valeur de `nbBytes` du fichier	
	Envoi du fichier .mp3 ou .mp4 en utilisant un flux de binaires
Lecture de `nbBytes` sur le socket et stockage du fichier reçu dans le répertoire temporaire du client.	

9.4 Le client quitte

Client	**Serveur**
Envoi de la constante `ProtocolConstant.CLOSE`	
	Reçoit la demande `ProtocolConstant.CLOSE`
	Le serveur ferme les flux vers le client et attend une autre connexion
Le client ferme les flux vers le serveur et efface les fichiers qui ont été téléchargés durant la connexion	

10 Liste des fichiers fournis

Voici la liste des fichiers fournis :

1. le code de base dans l'archive `src_fournie.zip`
2. l'utilitaire `LibraryDescriptionGenerator.zip`
3. le fichier `libLocal.txt` qui est un exemple de fichier représentant une librairie locale. Puisque les chemins des fichiers médias ne correspondront pas au même que sur votre machine, vous devrez regénérer votre propre fichier, avec l'outil `LibraryDescriptionGenerator.jar`

4. les archives `mediaFilesLocal.zip` et `mediaFilesServer.zip` qui sont des répertoires contenant des fichiers audio ou vidéo. Vous pouvez utiliser ces fichiers pour faire vos tests.

11 Barème de correction

Voici le barème de correction :

Qualité du code	/20
encapsulation	/5
clarté du code(noms des variables, ...)	/5
utilisation des mots-clés `static`, `abstract`, `final`,...	/5
bonne gestion des exceptions (message approprié, ...)	/5
conception et clareté du code dans les différents packages du projet	/10
Fonctionnement	**/85**
Partie 1 : Le package media	**/45**
La hiérarchie de classe `Media`	/15
La classe `MediaLibrary`	/15
La classe `Player` en incluant la gestion des communications client-serveur	/15
Partie 2 : Les communications client-serveur	**/35**
L'exécutable permettant de lancer le serveur	/10
Envoi de fichiers du client vers le serveur et vice-versa	/10
Envoi des informations de la librairie du serveur vers le client	/10
Bonne fermeture des connexions	/5
Les tests des classes des packages `medias` et `sockets`	/20
La classe ajoutée au package `mediaexceptions`	**/5**

Un programme qui ne compile pas ne sera pas recevable et méritera la note 0, à moins que ce ne soit pour une peccadille. Dans le cas d'une petite erreur, une pénalité de 25% sera appliquée.

12 Remise

La remise sera faite dans l'environnement LEA, dans la section prévu à cet effet.

12.1 Politique de retards

Aucun retard accepté pour ce travail, à moins d'une circonstance exceptionnelle.