1. But

Le but de ce travail est de mettre en pratique les concepts d'expressions lambda, et de streams tout en respectant toutes les spécifications des méthodes à définir.

2. Description générale

Votre travail consiste à implémenter un petit module de recherche d'informations sur un répertoire des cours de finance donnés en ligne, en utilisant les streams et les expressions lambda.

Fichiers fournis :

1)	Classe `CoursFinance`	La classe qui modélise un cours de finance. À ne pas modifier.
2)	Classe `CoursFinanceInvalideException`	Exception levée dans le constructeur de la classe `CoursFinance` lorsqu'on tente de construire un cours avec des données invalides. À ne pas modifier.
3)	Fichier texte **repertoireCoursFinances.csv**	Exemple de fichier texte contenant une liste de cours de finance pouvant être passé en paramètre au constructeur de la classe `RepertoireCoursFinance`. **Référence** : les données de ce fichier ont été tirées (et adaptées) d'un fichier de données publié par Chase Willden sur le site Web de Data World dont le lien est le suivant : https://data.world/chasewillden/business-finance-courses-from-udemy

NOTE : Les deux classes fournies seront utilisées telles quelles dans les tests de votre programme. Il est donc important de ne pas les modifier.

3. Détails et contraintes d'Implémentation

Vous devez implémenter une classe nommée `RepertoireCoursFinance`. Cette classe modélise un ensemble de cours de finance, et fournit des méthodes permettant d'obtenir des informations ou statistiques sur ces cours. Cette classe doit contenir les membres décrits dans les sous-sections ci-dessous.

IMPORTANT *:* le nom et le type de l'attribut, les noms de méthodes, les types de retour ainsi que l'ordre et le type des paramètres mentionnés dans les sections suivantes doivent être respectés à la lettre sinon vous risquez de perdre plusieurs points dans les tests.

3.1 Constantes de classe

Vous pouvez déclarer autant de constantes de classe que vous le jugez pertinent.

3.2 Attribut d'instance

Nom	Type	Description
`listeDesCours`	`List<CoursFinance>`	La liste des cours du répertoire.

- Vous devez respecter le nom et le type donnés de l'attribut d'instance. Cet attribut doit être déclaré sans indication de droit d'accès ou de niveau de visibilité pour qu'il soit accessible par toutes les classes du même paquetage.
- Vous ne devez pas ajouter d'autres attributs d'instance ou de classe.

3.3 Constructeur

Paramètre	Type	Description
`cheminFichier`	`String`	Le chemin du fichier CSV qui contient la liste des cours du répertoire.

Ce constructeur lit chacun des cours contenus dans le fichier donné en paramètre, et construit la liste des cours (attribut `listeDesCours`). Le fichier est formaté de la façon suivante (supposez qu'il est valide c'est à dire formaté correctement, et qu'il n'y manque aucune donnée). Chaque ligne du fichier (sauf la première) contient les informations sur un cours. Chaque information correspond à une colonne, et les colonnes sont séparées entre elles par un point-virgule. Voici les entêtes des colonnes (se trouvant toujours sur la première ligne du fichier) que vous devrez utiliser pour construire les objets de type `CoursFinance` qui doivent être stockés dans la liste `listeDesCours`.

ID : Le numéro d'identification **unique** du cours

Titre : Le titre du cours

Prix : Le prix du cours.
- ✓ Si le cours est gratuit, le prix est égal à 0.

Nb. d'étudiant(e)s : Le nombre d'étudiant(e)s inscrit(e)s

Nb. d'évaluations : Le nombre d'évaluations pour ce cours

Évaluation : La note moyenne (sur 10) attribuée à ce cours

Niveaux d'expériences : Le niveau d'expérience de l'étudiant(e) requis pour suivre ce cours
- ✓ Peut prendre les valeurs : Junior, Intermediaire, Senior, ou Tous.

Durée : La durée, en minutes, de ce cours

Date et heure de publication : La date et l'heure de publication de ce cours
- ✓ Sous le format aaaa-mm-jj HH:mm (ex.: 2015-10-07 14:01)

IMPORTANT :

1. Tous les cours contenus dans le fichier passé en paramètre doivent être présents dans la liste `listeDesCours`.
2. L'ordre des cours, dans la liste `listeDesCours` créée, doit respecter l'ordre des cours dans le fichier donné en paramètre.
3. Si le fichier reçu en paramètre ne peut pas être lu (est `null`, est inexistant sur le disque, etc.) ou s'il existe, mais qu'il est vide, alors la liste des cours créée demeure vide (la longueur est égale à 0).

NOTE : Pour obtenir un objet de type `LocalDateTime` à partir d'une date représentée sous forme de chaine de caractères dans le format `aaaa-mm-jj HH:mm`, utilisez la méthode de classe `parse` de la classe `LocalDateTime`, de cette manière :

```
String dateEnString = "2015-10-07 14:01";
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm");
LocalDateTime dateTime = LocalDateTime.parse(dateEnString, formatter);
```

CONTRAINTES D'IMPLÉMENTATION

- **Vous devez** utiliser un stream contenant les lignes du fichier donné en paramètre, et des expressions lambda comme paramètre(s) des méthodes appelées sur ce stream.
- **Vous devez** utiliser la méthode `map` de `Stream` pour transformer les `String` en `CoursFinance` dans le stream.

3.4 Méthodes d'instance publiques

Nom de la méthode : `obtenirNbCours`
Type de retour : `int`
Paramètre : aucun

Cette méthode retourne le nombre total de cours dans le répertoire de cours.

CONTRAINTE D'IMPLÉMENTATION

- **Vous devez** utiliser un stream et des expressions lambda comme paramètre(s) des méthodes appelées sur ce stream.

Nom de la méthode : `rechercherParTitre`
Type de retour : `List<CoursFinance>`

Paramètre	Type	Description
`chaineCaracteres`	`String`	La chaîne de caractères qui doit être contenue dans le titre des cours recherchés.

Cette méthode retourne une liste contenant tous les cours de ce répertoire qui contiennent, dans leur titre, la chaîne de caractères passée en paramètre (sans tenir compte de la casse). Si aucun cours n'est trouvé, cette méthode retourne une liste vide.

Exemple : En supposant que la chaîne de caractères passée en paramètre est "**account**", les cours ayant les titres suivants doivent être retournés par cette recherche :

- ✓ Introduction to **Account**ing : Mastering Financial Statements
- ✓ Introduction to Finance, **Account**ing, Modeling and Valuation
- ✓ **Account**ing & Financial Statement Analysis: Complete Training
- ✓ Three Steps Trading - Live Trading - Real **Account**
- ✓ Learn **Account**ing. Understand Business.
- ✓ Introduction to **Account**ing: The Language of Business
- ✓ Basic Excel for Basic Bookkeeping and **Account**ing
- ✓ Test your knowledge in Basics of **Account**ing
- ✓ Test your Knowledge in Cost **Account**ing
- ✓ Practical **Account**s APP Overview
- ✓ **Account**ing for Beginners : Learn Basics in under 1 Hour
- ✓ Practical **Account**s & Bookkeeping Automated Overview
- ✓ Live **Account** - ETF Trading System - Hacking The Stock Market
- ✓ Advanced **Accoun**t**ing** A Complete Study for CA / CMA / CFA / CS
- ✓ B Com **Account**ancy I (Paper ECO 02 IGNOU)
- ✓ Learn the basics of preparing **account**ing statements

Si le paramètre `chaineCaracteres` est `null` ou une chaîne de caractères vide, cette méthode retourne une liste vide. De plus, la liste des cours retournée doit être triée par titre (sans tenir compte de la casse).

CONTRAINTES D'IMPLÉMENTATION

- **Vous devez** utiliser un stream et des expressions lambda comme paramètre(s) des méthodes appelées sur ce stream.
- Pour obtenir une liste (`List<T>`) à partir d'un `stream<T>`, **utilisez** la méthode terminale `collect` sur ce stream, de cette manière : `stream.collect(Collectors.toList())`. Il vous faudra importer la classe `java.util.stream.Collectors`.

Nom de la méthode : `rechercherParEvaluation`
Type de retour : `List<CoursFinance>`

Paramètre	Type	Description
`eval`	`float`	L'évaluation des cours recherchés.
`plusPetite`	`boolean`	Si vrai, on recherche les cours ayant une évaluation plus petite que `eval`, sinon on recherche les cours ayant une évaluation plus grande ou égale à `eval`.

Cette méthode retourne une liste contenant tous les cours de ce répertoire qui possèdent une évaluation plus petite à `eval` si le paramètre `plusPetite` est vrai, sinon elle retourne une liste contenant tous les cours de ce répertoire qui possèdent une évaluation plus grande ou égale à `eval`. Si aucun cours n'est trouvé, cette méthode retourne une liste vide. De plus, la liste des cours retournée doit être triée par évaluation. Si plusieurs cours retournés ont la même évaluation, ceux-ci doivent être triés entre eux selon le niveau d'expérience (sans tenir compte de la casse), et si plusieurs cours ont la même évaluation, et le même niveau d'expérience, ceux-ci doivent être triés entre eux selon leur ID.

CONTRAINTES D'IMPLÉMENTATION

- **Vous devez** utiliser un stream et des expressions lambda comme paramètre(s) des méthodes appelées sur ce stream.
- Pour obtenir une liste (`List<T>`) à partir d'un `stream<T>`, **utilisez** la méthode terminale `collect` sur ce stream.

Nom de la méthode : `rechercherParNiveauExperience`
Type de retour : `List<CoursFinance>`

Paramètre	Type	Description
`niveauxExperience`	`List<String>`	On recherche les cours dont le niveau d'expérience requis est présent dans cette liste.

Cette méthode retourne une liste contenant tous les cours dont le niveau d'expérience requis est présent dans la liste `niveauxExperience` donnée en paramètre. Si aucun cours n'est trouvé, la méthode retourne une liste vide.

En ce qui concerne la sélection selon le niveau d'expérience, pour qu'un cours soit sélectionné, il faut que le niveau d'expérience associé soit exactement égal (mais sans tenir compte de la casse) à l'une des valeurs dans la liste `niveauxExperience` donnée. Si la liste `niveauxExperience` donnée en paramètre est `null` ou vide, cette méthode retourne une liste vide. De plus, la liste des cours retournée doit être triée par titre (sans tenir compte de la casse). Si plusieurs cours ont le même titre, ceux-ci doivent être triés entre eux selon leur ID.

CONTRAINTES D'IMPLÉMENTATION

- **Vous devez** utiliser un stream et des expressions lambda comme paramètre(s) des méthodes appelées sur ce stream.
- Pour obtenir une liste (`List<T>`) à partir d'un `stream<T>`, **utilisez** la méthode terminale `collect` sur ce stream.

Nom de la méthode : `rechercherParPeriode`
Type de retour : `List<CoursFinance>`

Paramètre	Type	Description
`dateHeureDebut`	`LocalDateTime`	La date et l'heure de publication des cours considérées comme la date / heure inférieure ou minimale des cours recherchés. Si cette date / heure est `null`, il n'y a pas de date inférieure ou minimale à considérer pour la recherche.
`dateHeureFin`	`LocalDateTime`	La date et l'heure de publication des cours considérées comme la date / heure supérieure ou maximale des cours recherchés. Si cette date / heure est `null`, il n'y a pas de date supérieure ou maximale à considérer pour la recherche.

Cette méthode retourne une liste contenant tous les cours dont la date et l'heure de publication se trouvent entre `dateHeureDebut` et `dateHeureFin` inclusivement. Si aucun cours n'est trouvé, cette méthode retourne une liste vide.

Précisions :

- Si `dateHeureDebut` n'est pas `null` et `dateHeureFin` n'est pas `null`, on recherche les cours dont la date / heure de publication est entre `dateHeureDebut` et `dateHeureFin` inclusivement.
- Si `dateHeureDebut` est `null` et `dateHeureFin` n'est pas `null`, on rechercher les cours dont la date / heure de publication est inférieure ou égale à `dateHeureFin`.
- Si `dateHeureDebut` n'est pas `null` et `dateHeureFin` est `null`, on recherche les cours dont la date / heure de publication est supérieure ou égale à `dateHeureDebut`.
- Si `dateHeureDebut` est égale à `dateHeureFin`, on recherche les cours dont la date / heure de publication est égale à `dateHeureDebut` (ou `dateHeureFin`).
- Cette méthode doit lever une `java.util.IllegalArgumentException` si `dateHeureDebut` et `dateHeureFin` sont toutes les deux `null` ou si `dateHeureDebut` est supérieure à `dateHeureFin`.

De plus, la liste retournée doit être triée par date / heure de publication. Si plusieurs cours ont la même date et l'heure de publication, ceux-ci doivent être triés entre eux selon leur ID.

NOTE : Pour tester si une `LocalDateTime` est inférieure, égale ou supérieure à une autre `LocalDateTime`, utilisez la méthode `compareTo` de la classe `LocalDateTime`. Notez qu'inférieure signifie antérieure et supérieure signifie postérieure.

CONTRAINTES D'IMPLÉMENTATION

- **Vous devez** utiliser un stream et des expressions lambda comme paramètre(s) des méthodes appelées sur ce stream.
- Pour obtenir une liste (`List<T>`) à partir d'un `stream<T>`, **utilisez** la méthode terminale `collect` sur ce stream.

Nom de la méthode : `rechercherParNbEtudiants`
Type de retour : `String[]`

Paramètre	Type	Description
`n`	`int`	Si `n` est positif, on recherche les `n` cours ayant le plus grand nombre d'étudiants inscrits. Si `n` est négatif, on recherche les `|n|` cours ayant le plus petit nombre d'étudiants inscrits.

Cette méthode retourne un tableau de longueur minimale contenant les `n` **titres** de cours qui ont le plus grand nombre d'étudiants inscrits ou le plus petit nombre d'étudiants inscrits. Un tableau de longueur minimale signifie que sa longueur est égale au nombre d'éléments qu'il contient. En d'autres mots, si la méthode retourne `x` titres de cours, le tableau doit être de longueur `x`.

Si `n > 0`, la méthode doit retourner un tableau contenant les `n` titres de cours les **PLUS populaires**, et le tableau retourné doit être trié en ordre décroissant du nombre d'étudiants inscrits. Si deux cours ont le même nombre d'étudiants inscrits, ils doivent être triés entre eux selon leur ID (en ordre croissant).

Si `n < 0`, la méthode doit retourner un tableau contenant les `n` titres de cours les **MOINS populaires**, et le tableau retourné doit être trié en ordre croissant du nombre d'étudiants inscrits. Si deux cours ont le même nombre d'étudiants inscrits, ils doivent être triés entre eux selon leur ID (en ordre croissant).

Si `n = 0`, le tableau retourné doit être de longueur `0`.

Notez que si la valeur absolue de `n` est plus grande que le nombre de cours contenu dans ce répertoire, la méthode retourne un tableau contenant TOUS les titres des cours de ce répertoire.

CONTRAINTES D'IMPLÉMENTATION

- **Vous devez** utiliser un stream et des expressions lambda comme paramètre(s) des méthodes appelées sur ce stream.
- **Vous devez** utiliser la méthode `toArray` de Stream pour obtenir le tableau à retourner.

NOTE : La méthode `limit` de l'interface `Stream` pourrait être fort utile ici.

Nom de la méthode : `rechercherParPourcentageEvaluation`
Type de retour : `String[]`

Paramètre	Type	Description
`n`	`int`	Si `n` est positif, on recherche les `n` cours ayant le plus grand pourcentage d'évaluation. Si `n` est négatif, on recherche les `|n|` cours ayant le plus petit pourcentage d'évaluation.

Cette méthode retourne un tableau de longueur minimale contenant les `n` **titres** de cours qui ont le plus grand pourcentage d'évaluation ou le plus petit pourcentage d'évaluation. Un tableau de longueur minimale signifie que sa longueur est égale au nombre d'éléments qu'il contient. En d'autres mots, si la méthode retourne `x` titres de cours, le tableau doit être de longueur `x`. Notez bien qu'on calcule ici le pourcentage d'évaluation d'un cours selon la formule suivante :

```
Pourcentage d'évaluation = ((float)Nb. d'évaluations / Nb. D'étudiant(e)s) * 100
```

Si `n > 0`, la méthode doit retourner un tableau contenant les `n` titres de cours les **PLUS évalués**, et le tableau retourné doit être trié en ordre décroissant des pourcentages d'évaluations. Si deux cours ont le même pourcentage d'évaluation, ils doivent être triés entre eux selon leur ID (en ordre croissant).

Si `n < 0`, la méthode doit retourner un tableau contenant les `n` titres de cours les **MOINS évalués**, et le tableau retourné doit être trié en ordre croissant des pourcentages. Si deux cours ont le même pourcentage d'évaluation, ils doivent être triés entre eux selon leur ID (en ordre croissant).

Si `n = 0`, le tableau retourné doit être de longueur 0.

Notez que si la valeur absolue de `n` est plus grande que le nombre de cours contenu dans ce répertoire, la méthode retourne un tableau contenant TOUS les titres des cours de ce répertoire.

CONTRAINTES D'IMPLÉMENTATION

- **Vous devez** utiliser un stream et des expressions lambda comme paramètre(s) des méthodes appelées sur ce stream.
- **Vous devez** utiliser la méthode `toArray` de Stream pour obtenir le tableau à retourner.

NOTE : La méthode `limit` de l'interface `Stream` pourrait être fort utile ici.

Nom de la méthode : `rechercherParPrix`
Type de retour : `List<CoursFinance>`

Paramètre	Type	Description
`prix`	`float`	Le prix des cours recherchés.
`estPlusGrand`	`boolean`	Si vrai, on recherche les cours dont le prix est plus grand que `prix`, sinon on recherche les cours dont le prix est plus petit ou égal à `prix`.

Cette méthode retourne une liste contenant tous les cours de ce répertoire qui possèdent un prix plus grand que `prix` si le paramètre `estPlusGrand` est vrai, sinon elle retourne une liste contenant tous les cours de ce répertoire qui possèdent un prix plus petit ou égal au paramètre `prix`. Si aucun cours n'est trouvé, cette méthode retourne une liste vide. De plus, la liste des cours retournée doit être triée par prix. Si plusieurs cours retournés ont le même prix, ceux-ci doivent être triés entre eux selon leur ID.

CONTRAINTES D'IMPLÉMENTATION

- **Vous devez** utiliser un stream et des expressions lambda comme paramètre(s) des méthodes appelées sur ce stream.
- Pour obtenir une liste (`List<T>`) à partir d'un `stream<T>`, **utilisez** la méthode terminale `collect` sur ce stream.

4. Précisions sur les spécifications

- À moins d'indication contraire, lorsqu'on demande de trier, c'est toujours en ordre croissant.
- Les recherches et tris effectué(e)s sur les chaines de caractères ne doivent jamais tenir compte de la casse.
- Réutilisez votre code autant que possible. Vous pouvez (et devriez) faire des **méthodes privées** pour bien **structurer/modulariser votre code** (séparation fonctionnelle).
- Toute méthode qui n'est pas décrite dans cet énoncé doit être `private`.
- **Le non-respect des contraintes d'implémentation peut engendrer des pénalités sévères, car ce sont ces contraintes qui assurent que vous utilisiez la matière qu'on veut noter.**
- N'oubliez pas d'écrire les commentaires de documentation (**Javadoc**) pour documenter toutes vos méthodes et votre classe.
- Utilisez des constantes autant que possible. Celles-ci doivent être déclarées et initialisées au niveau de la classe (constantes de classe) : `public` (ou `private`) `static final`… Aussi, si vous utilisez un comparateur (`Comparator`) plusieurs fois, par exemple, faites-en une constante !
- Toute **variable globale est interdite** (sauf l'attribut d'instance, et les constantes de classe).
- Il ne doit y avoir aucun affichage dans vos méthodes (pas de `System.out`).

5. Travail demandé

5.1. Organisation de votre projet

Vous devez créer votre projet nommé `tp3`, et ajouter le paquetage : `ca.uqam.h2024.inf2120.grpe30.tp3`.

Vous devez ajouter les classes `CoursFinance`, `CoursFinanceInvalideException`, `RepertoireCoursFinance` dans le paquetage : `ca.uqam.h2024.inf2120.grpe30.tp3`. Assurez-vous d'ajouter le fichier `repertoireCoursFinances.csv` dans votre projet.

Vous devez implémenter la classe `RepertoireCoursFinance` et définir toutes les méthodes décrites. Votre implémentation doit respecter intégralement les spécifications détaillées dans les sections précédentes.

Si quelque chose est ambiguë, obscure, s'il manque de l'information, si vous ne comprenez pas les spécifications, si vous avez des doutes… vous avez la responsabilité de vous informer auprès de votre enseignant.

5.2. Ce que vous devez remettre

La date de remise du travail pratique 3 est le **jeudi 25 avril 2024** avant **23H55**. Via Moodle, vous devez remettre une copie électronique de votre projet `tp3` compressée (.zip) qui contient les classes : `CoursFinance`, `CoursFinanceInvalideException`, `RepertoireCoursFinance`, et le fichier `repertoireCoursFinances.csv`.

5.3. Pénalités

Pour tout travail remis en retard, les pénalités seront appliquées selon la formule suivante : `Nombre de points de pénalité = m / 144`, où m est le nombre de minutes de retard par rapport à l'heure de remise. Aucun travail ne sera accepté après cinq (5) jours de retard.

5.4. Plagiat

Le travail est strictement individuel. Le règlement sur le plagiat sera appliqué sans exception. Vous devez ainsi vous assurer de ne pas échanger du code Java avec des collègues.

6. Pondération

6.1. Code Source (40%)

Le respect des spécifications des méthodes à implémenter, l'utilisation des expressions lambda, et des streams, les commentaires Javadoc des méthodes, et les bonnes structures de contrôle non redondantes seront les éléments à considérer lors de l'évaluation.

6.2 Exécution (60%)

L'exactitude des résultats ou des statistiques retournés par les méthodes sera l'élément principal à considérer lors de l'évaluation de cette partie. La note zéro (0) sera attribuée à ce niveau pour tout programme Java qui ne compile pas.