Travail pratique #3

Ce travail doit être fait individuellement.

Aucun retard permis

Notions mises en pratique : respect d'un ensemble de spécifications, expressions lambda, et streams.

1. Description générale

Votre travail consiste à implémenter un petit module de recherche d'informations sur les films d'une collection donnée, en utilisant les streams et les expressions lambda.

Fichiers fournis :

1)	Classe `Film`	Classe qui modélise un film. À ne pas modifier.
2)	Classe `FilmInvalideException`	Exception levée dans le constructeur de la classe `Film` lorsqu'on tente de construire un film avec des données invalides. À ne pas modifier.
3)	Fichier texte **statistiquesFilms.csv**	Exemple de fichier texte contenant une liste de films pouvant être passé en paramètre au constructeur de la classe `CollectionFilms`. **Référence** : les données de ce fichier ont été tirées (et adaptées) d'un fichier de données publié par Yashwanth Sharaff sur https://data.world/yasharaff/movies-gross.

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

1.1 DÉTAILS ET CONTRAINTES D'IMPLÉMENTATION

Vous devez implémenter une classe nommée `CollectionFilms`. Cette classe modélise une collection (liste) de films, et fournit des méthodes permettant d'obtenir des informations ou statistiques concernant ces films. 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.**

1.1.1 Constantes de classe

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

1.1.2 Attribut d'instance

Nom de l'attribut	type	Description
`films`	`List<Film>`	La liste des films dans cette collection

- Vous DEVEZ respecter le nom et le type donnés de l'attribut d'instance.
- Vous ne DEVEZ PAS ajouter d'autres attributs d'instance ou de classe.

1.1.3 Constructeur

Paramètres	type	Description
`cheminFic`	`String`	Le chemin du fichier CSV qui contient la liste des films que contiendra cette collection.

Ce constructeur lit chacun des films contenus dans le fichier donné en paramètre, et construit la liste des films (attribut `films`). Le fichier est formaté de la façon suivante (supposez qu'il est valide c.-à-d. formaté correctement, et qu'il n'y manque aucune donnée) : chaque ligne du fichier (sauf la première) contient les informations sur un film. Chaque information correspond à une colonne, et chaque colonne est séparée 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 `Films` qui doivent être stockés dans la liste `films`.

Film ID : Numéro d'identification **unique** du film

Titre : Titre du film

Classement MPAA : Classement MPAA pour ce film.
- Peut prendre les valeurs : G, PG, PG-13, R, ou NC-17

Budget : Le budget alloué pour faire ce film

Recettes : Les recettes produites par ce film

Date de sortie : La date de sortie de ce film
- Sous le format `aaaa-mm-jj` (ex.: 1989-10-12)

Genre : Le genre de ce film
- Peut prendre les valeurs : Romance, Comedie, Crime, Guerre, Drame, Famille, Action, Animation, Science Fiction, Aventure, Thriller, Western, Horreur, Mistere, Histoire, et Fantastique.

Duree : La durée, en minutes, de ce film

Evaluation : Note moyenne (sur 10) attribuée à ce film

Nombre d'evaluations : Le nombre d'évaluations sur lequel on a calculé l'évaluation précédente (note moyenne).

Le fichier **statistiquesFilms.csv**, donné avec l'énoncé de ce TP, est un exemple de fichier valide. Notez que dans un fichier valide, tous les films ont un **Film ID** différent, mais il peut cependant y avoir des doublons. **Deux films sont considérés comme des doublons s'ils ont le même titre (titres égaux, sans tenir compte de la casse), la même date de sortie, et la même durée (même si leurs IDs sont différents)**, en conformité à la méthode `equals` de la classe `Film`. De plus, notez que les films, dans un fichier valide, ne sont pas nécessairement triés.

IMPORTANT :
1. Tous les films contenus dans le fichier passé en paramètre doivent être présents dans la liste `films` créée (même les doublons, s'il y a lieu).
2. L'ordre des films, dans la liste `films` créée, doit respecter l'ordre des films 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 collection créée demeure vide (sa liste `films` a une longueur égale à 0).

NOTE :
Pour obtenir un objet de type `LocalDate` à partir d'une date représentée sous forme de chaine de caractères dans le format `aaaa-mm-jj`, utilisez la méthode de classe `parse` de la classe `LocalDate`, de cette manière :
`LocalDate d = LocalDate.parse("1998-05-23").`

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 `Film` dans le stream.

1.1.4 Méthodes d'instance publiques

Nom méthode : `getNbrFilmsDistincts`
Type retour : `int`
Paramètre : aucun

Cette méthode retourne le nombre total de films **distincts** (sans compter les doublons) dans la collection.

CONTRAINTE D'IMPLÉMENTATION

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

Nom méthode : `rechercherParTitre`
Type retour : `List<Film>`

Paramètre	type	Description
`expression`	`String`	Expression qui doit être contenue dans le titre des films qu'on recherche.

Cette méthode retourne une liste contenant tous les films **distincts** (sans doublons) de cette collection qui contiennent, dans leur titre, l'expression donnée en paramètre (sans tenir compte de la casse). Si aucun film n'est trouvé, cette méthode retourne une liste vide. Exemple :

En supposant que l'expression donnée en paramètre est **bat**, les films ayant les titres suivants seraient retournés par cette recherche :

- LE **BAT**EAU-MOUCHE
- Les **Bat**tements d'ailes d'un papillon
- A**bat**s et autres mets délicieux.

Si le paramètre `expression` est `null` ou est égal à la chaine vide, cette méthode retourne une liste vide.

De plus, la liste des films retournée doit être triée par titre (sans tenir compte de la casse). Si plusieurs films 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, de cette manière : `stream.collect(Collectors.toList())`. Il vous faudra importer la classe `java.util.stream.Collectors;`

Nom méthode : `rechercherParEvaluation`
Type retour : `List<Film>`

Paramètre	type	Description
`evaluationMinimum`	`double`	L'évaluation minimum des films recherchés.

Cette méthode retourne une liste contenant tous les films **distincts** (sans doublons) de cette collection qui possèdent une évaluation plus grande ou égale à `evaluationMinimum`. Si aucun film n'est trouvé, cette méthode retourne une liste vide. De plus, la liste des films retournée doit être triée par évaluation. Si plusieurs des films retournés ont la même évaluation, ceux-ci doivent être triés entre eux selon leur genre (sans tenir compte de la casse), et si plusieurs films ont la même évaluation, et le même genre, 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 méthode : `rechercherParGenres`
Type retour : `List<Film>`

Paramètre	type	Description
`genres`	`List<String>`	On recherche les films dont le genre est présent dans cette liste.
`evaluationMinimum`	`double`	L'évaluation minimum des films recherchés.

Cette méthode retourne une liste contenant tous les films **distincts** (sans doublons) dont le genre est présent dans la liste `genres` donnée en paramètre ET dont l'évaluation est plus grande ou égale à l'`évaluationMinimum` donné en paramètre. Si aucun film n'est trouvé, la méthode retourne une liste vide.

En ce qui concerne la sélection selon le genre, pour qu'un film soit sélectionné, il faut que son genre associé soit exactement égal (mais sans tenir compte de la casse) à l'une des valeurs dans la liste `genres` donnée. Si la liste `genres` donnée en paramètre est `null` ou vide, cette méthode retourne une liste vide. De plus, la liste des films retournée doit être triée par titre (sans tenir compte de la casse). Si plusieurs films 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 méthode : `rechercherParPeriode`
Type retour : `List<Film>`

Paramètre	type	Description
`dateDebut`	`LocalDate`	La date de sortie inférieure minimale des films recherchés. Si cette date est `null`, il n'y a pas de date minimale à considérer pour la recherche.
`dateFin`	`LocalDate`	La date de sortie supérieure maximale des films recherchés. Si cette date est `null`, il n'y a pas de date maximale à considérer pour la recherche.

Cette méthode retourne une liste contenant tous les films **distincts** (sans doublons) dont la date de sortie se trouve entre `dateDebut` et `dateFin` inclusivement. Si aucun film n'est trouvé, cette méthode retourne une liste vide.

Précisions :

- Si `dateDebut` n'est pas `null` et `dateFin` n'est pas `null`, on recherche les films dont la date de sortie est entre `dateDebut` et `dateFin` inclusivement.
- Si `dateDebut` est `null` et `dateFin` n'est pas `null`, on rechercher les films dont la date de sortie est inférieure ou égale à `dateFin`.
- Si `dateDebut` n'est pas `null` et `dateFin` est `null`, on recherche les films dont la date de sortie est supérieure ou égale à `dateDebut`.
- Si `dateDebut` est égale à `dateFin`, on recherche les films dont la date de sortie est égale à `dateDebut` (ou `dateFin`).
- Cette méthode doit lever une `java.util.NoSuchElementException` si `dateDebut` et `dateFin` sont toutes les deux `null` ou si `dateDebut` est supérieure à `dateFin`.

De plus, la liste retournée doit être triée par dates de sortie. Si plusieurs films ont la même date de sortie, ceux-ci doivent être triés entre eux selon leur ID.

NOTE :
Pour tester si une `LocalDate` est inférieure, égale ou supérieure à une autre `LocalDate`, utilisez la méthode `compareTo` de la classe `LocalDate`. Note que 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 méthode : `rechercherParProfit`
Type retour : `String[]`

Paramètre	type	Description
`n`	`int`	Si `n` est positif, on recherche les `n` films ayant fait le plus de profit. Si `n` est négatif, on recherche les `|n|` films ayant fait le moins de profit.

Cette méthode retourne un tableau de longueur minimale contenant les `n` **titres** de film **distincts** (sans doublons) qui ont généré le plus de profit ou le moins de profit. 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 film,

le tableau doit être de longueur `x`. Notez qu'on calcule ici le profit d'un film en faisant la différence entre ses recettes et son budget (`recettes - budget`).

Si `n > 0`, la méthode doit retourner un tableau contenant les `n` titres de film les **PLUS profitables**, et le tableau retourné doit être trié en ordre décroisssant des profits. Si deux films ont le même profit, 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 film les **MOINS profitables**, et le tableau retourné doit être trié en ordre croisssant des profits. Si deux films ont le même profit, 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 films distincts contenu dans cette collection, la méthode retourne un tableau contenant TOUS les titres des films disctincts de cette collection.

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…

2. Précisions sur les spécifications

- À moins d'indication contraire, lorsqu'on demande de trier, c'est toujours en ordre croissant.
- La plupart des méthodes demandées doivent retourner une liste (ou un tableau) de films distincts (sans les doublons). Il est important que le filtre des doublons s'effectue sur la collection initiale complète, non triée, et qu'ainsi, les doublons retirés sont bien ceux qui viennent après dans la collection. Par exemple, soit la collection qui contient les films suivants : `[ f1, f2, f3, f1, f1, f3, f4]`. Il est important que ce soit les doublons bleus qui soient retirés de la collection, et non les doublon oranges.
- Deux films sont considérés comme des doublons s'ils ont le même titre (titres égaux, sans tenir compte de la casse), la même date de sortie, et la même durée (même si leurs IDs sont différents), en conformité à la méthode `equals` de la classe `Film`.
- 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).
- Votre classe doit se trouver dans le **paquetage par défaut**.
- Il ne doit y avoir aucun affichage dans vos méthodes (pas de `System.out`)

- Vous DEVEZ respecter **le style Java**.
- Votre code doit compiler et s'exécuter avec le **JDK 8.**
- Vous devez respecter le **principe d'encapsulation** des données.

Le non-respect de toute spécification ou consigne se trouvant dans l'énoncé de ce TP est susceptible d'engendrer une perte de points.

Si quelque chose est ambigu, 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 enseignante.

3. Détails sur la correction

3.1 LA QUALITÉ DU CODE (40 POINTS)

Concernant les critères de correction du code, lisez attentivement le document "**Critères généraux de correction du code Java"**. De plus, votre code sera noté sur le respect des conventions de style Java vues en classe dont un résumé se trouve dans le document **"Conventions de style Java"**. Ces deux documents peuvent être téléchargés dans la section TRAVAUX PRATIQUES (ET BOITES DE REMISE) de la page Moodle du cours.

Votre code sera corrigé sur la totalité ou une partie des critères de correction indiqués ci-dessus, ainsi que sur le respect des spécifications/consignes mentionnées dans ce document.

3.2 L'EXÉCUTION (60 POINTS)

Un travail qui ne compile pas se verra attribuer la note 0 pour l'exécution.

Votre code sera testé en tout ou en partie. Les points seront calculés sur les tests effectués. Ceci signifie que si votre code fonctionne dans un cas x et que ce cas x n'est pas testé, vous n'obtiendrez pas de points pour ce cas. Il est donc dans votre intérêt de vous assurer du bon fonctionnement de votre programme dans tous les cas possible..

4. Date et modalités de remise

4.1 REMISE

Date de remise : AU PLUS TARD le **24 avril 2022** à 23h59. **AUCUN RETARD ACCEPTÉ**

Le fichier à remettre : `CollectionFilms.java`
(**NE PAS remettre votre fichier dans une archive zip, rar…**).

Remise via Moodle uniquement.

Vous devez remettre (téléverser) votre fichier sur le site du cours (Moodle). Vous trouverez la boîte de remise dans la section **TRAVAUX PRATIQUES (ET BOITES DE REMISE)**.

Assurez-vous de remettre votre travail avant la date limite, sur Moodle, car après, vous n'aurez plus accès à la boite de remise, et vous ne pourrez plus remettre votre travail.

4.2 POLITIQUE CONCERNANT LES RETARDS

AUCUN RETARD PERMIS

4.3 REMARQUES GÉNÉRALES

- Aucun travail reçu par couriel ne sera accepté. PLUS PRÉCISÉMENT, **un travail remis par courriel sera considéré comme étant non remis**.
- **Vous avez la responsabilité de conserver des copies de sauvegarde** de votre travail (sur disque externe, Dropbox, Google Drive, etc.). La perte d'un travail due à un vol, un accident, un bris… n'est pas une raison valable pour obtenir une extension pour la remise de votre travail.
- **N'oubliez pas d'écrire votre nom complet et votre code permanent (entre autres) dans l'entête de votre classe à remettre).**

N'attendez pas à la dernière minute pour commencer le travail, vous allez fort probablement rencontrer des problèmes inattendus !

Le règlement sur le plagiat sera appliqué sans exception. Vous devez ainsi vous assurer de ne pas échanger du code avec des collègues ni de laisser sans surveillance votre travail au laboratoire. Vous devez également récupérer rapidement toutes vos impressions de programme au laboratoire.

BON TRAVAIL