Atelier Java - séance 4

Objectifs :

→ Comprendre les intentions de l'API Stream
→ Écrire des codes permettant d'extraire les informations souhaitées grâce à l'API Stream.
→ Créer des streams de différentes façons : à partir d'une collection, depuis un fichier, à partir de rien.

Thèmes abordés :

- API Stream

Récupérez les sources de la séance 10 et importez le projet dans votre workspace.

Exercice 1 : expressions Lambda

Considérant la liste suivante

```
List<String> words = Arrays.asList("hi", "hello", "hola", "bye", "goodbye", "adios");
```

1. Ecrivez une méthode de classe `allMatches` (dans une classe `Utils` dans le package exercices) qui prend en paramètre une liste de String et un `Predicate` de String. Cette méthode renvoie une nouvelle liste de toutes les valeurs qui réussissent le test du prédicat. Testez avec plusieurs prédicats : les mots qui contiennent la lettre 'o', les mots de longueur supérieure à 4, …
2. Modifiez cette méthode en veillant à ce que n'importe quel type de listes puisse être transmis en paramètre avec évidemment un prédicat du même type. Testez avec une liste d'entiers :
   ```
   List<Integer> nums = Arrays.asList(1, 10, 100, 1000, 10000);
   List<Integer> bigNums = Utils.allMatches(nums, n -> n>500);
   ```
3. Ecrivez une méthode de classe `transformedList` qui prend en paramètre une liste de `String` et une `Function<String,String>` et renvoie une nouvelle liste qui contient les éléments de la liste reçue sur lesquels la fonction en paramètre a été appliquée. Testez avec plusieurs fonctions : mettre en majuscules, ajouter un ' !' à la fin de chaque mot, remplacer les 'i' par des 'e', …
4. Modifiez de façon similaire au point 2 cette méthode afin de permettre par exemple le code ci-dessous :
   ```
   List<Integer> wordLengths = Utils.transformedList(words, String::length);
   List<Double> inverses = transformedList(nums, i -> 1.0/i);
   ```

Exercice 2 : streams de transactions[1]

Complétez la méthode main de la classe `Exercice2Transactions` afin d'effectuer les opérations suivantes :

1. Lister toutes les transactions de 2011 en ordre croissant de valeur.
2. Lister les villes où travaillent les courtiers (traders).
3. Lister tous les courtiers de Cambridge par ordre alphabétique sur leur nom.
4. Construire une `String` contenant tous les noms des courtiers triés par ordre alphabétique et l'afficher.
5. Préciser (true-false) s'il y a un courtier qui travaille à Milan.
6. Afficher toutes les valeurs des transactions effectuées par des courtiers qui vivent à Cambridge.
7. Fournir la plus grande valeur de transaction.
8. Trouver la transaction dont la valeur est la plus faible.
9. Indiquer en une seule fois le nombre de transactions, le total des transactions, la valeur de la transaction minimum, …

[1] Ces exercices sont issus du livre Java 8 in action référencé en bibliographie de la théorie.

Exercice 3 : streams de String

Créez une classe contenant une méthode `main` à l'intérieur de laquelle vous allez créer une liste de `String`. Ensuite, effectuez les exercices suivants en une seule ligne :

1. Afficher chaque mot de la liste.
2. Produire une autre liste de String dans laquelle chaque mot se termine avec " !".
3. Placer tous les mots dans un tableau de String.
4. Produire une autre liste de String dans laquelle chaque mot est mis en majuscule et trié par ordre anti-alphabétique.
5. Fournir une String contenant tous les mots de longueur pair séparé par un espace.
6. Ecrire une méthode qui prend en paramètre une liste de `Strings` et un char. Cette méthode renvoie une `String` qui correspond à ce qui suit :
    a. le premier mot de la liste en argument ;
    b. mis en majuscule ;
    c. dont la longueur est inférieure à 4 ;
    d. contenant la lettre reçue en argument.

    Si aucun mot ne correspond, cette méthode renvoie "No match".

    Ici, c'est la méthode qui doit être écrite en une seule ligne!

7. Compléter l'exercice précédent en veillant à afficher, à chaque fois qu'un mot est mis en majuscule, "TO UPPER " avec le mot en question. Ceci se fait en écrivant une fonction `Function<String,String>`.
8. Produire une String qui contiendra la concaténation de tous les mots du stream mis en majuscule. On vous demande de le faire en 2 versions différentes : l'une avec `map` et l'autre sans.
9. Fournir le nombre total de caractères des strings de la liste.
10. Fournir le nombre de mot qui contienne le caractère 'e'

Exercice 4 : streams infinis de nombres

Créez une classe contenant une méthode `main`

1. Dans le `main`, construire un tableau de 10 réels aléatoires à partir d'un `Stream` de double compris entre 0 (inclus) et 100 (exclus) en utilisant la fonction `doubles()` de la classe `Random`.
2. Calculer la somme des racines carrées des éléments de ce tableau. Dans la classe `DoubleStream`, la méthode `of` permet de fournir le Stream d'un tableau.
3. Ecrire deux méthodes similaires : l'une fournit un `DoubleStream` de réels aléatoires et l'autre un `Stream<Double>` de réels aléatoires. Toutes deux prennent en paramètre la valeur maximum des réels aléatoires. La méthode `generate` est sympa ! Faites les exercices suivants en invoquant ces deux méthodes :
    a. Afficher 5 réels aléatoires compris entre 0 et 100.
    b. Créer une liste de 10 réels aléatoires compris entre 0 et 100.
    c. Construire un tableau de 20 réels aléatoires compris entre 0 et 100.

Exercice 5 : streams avec les fichiers

On vous fournit un fichier CSV d'étudiants : etudiants.csv.

Créez une classe contenant une méthode `main`. En utilisant les streams et le `try-with-resource`, répondez aux points suivants :

1. Afficher la première ligne du csv
2. Construire une liste d'étudiants.

`Etudiant` contient 4 attributs (nom, prenom, email et matricule). Il possède 2 constructeurs : l'un avec un String en paramètre qui concatène toutes les propriétés avec des ; comme le csv (il splitte ceux-ci pour les affecter aux bonnes valeurs) et l'autre avec l'email et le matricule.

   a. Construire une liste d'Etudiant en utilisant le constructeur avec un seul paramètre.
   b. Construire ensuite une autre liste en utilisant le constructeur qui a les deux paramètres.

3. Imprimer les noms de moins de 6 caractères qui contiennent la voyelle 'l' ou'Y'. Il faut appeler une seule fois la fonction `filter`.
4. Définir une méthode de classe `isAeWord` qui renvoie vrai si le mot en paramètre contient la suite "ae". Avec cette méthode, afficher un prénom du fichier csv qui contient cette suite ou « none » s'il n'y en a pas.
5. Indiquer si tous les étudiants ont un email
6. Indiquer le matricule d'un étudiant dont le prénom est Kevin
7. Afficher le nombre d'étudiants du fichier csv.
8. Afficher tous les fichiers du projet Eclipse dans lequel vous travaillez. Regardez dans `Files.`

Exercice 6 : collection de transactions

Reprenons les streams de transactions comme à l'exercice 2. Complétez la classe `Exercice6Transactions` afin de construire, grâce aux streams, les collections suivantes :

1. `Map<Trader, List<Transaction>>` (transactions du trader)
2. `Map<Trader,Long>` (nombre de transactions de ce trader)
3. `Map<Trader,Transaction>` (transaction du trader dont la valeur est la plus élevée)
4. `Map<String,Map<Trader,List<Transaction>>>` (les transactions de chaque ville par trader)
5. `Map<TransactionsLevel,List<Transaction>>`

Considérant l'`enum TransactionsLevel {VeryHi, Hi, Lo, Me},` les transactions sont réparties selon les critères suivants :

- si valeur >= 1000, elle est `VeryHi` ;
- si 800 <= valeur <1000, elle est `Hi` ;
- si 600 <= valeur < 800, elle est `Me` ;
- sinon elle est `Lo`.

6. `Map<Boolean,List<Transaction>>` (les transactions des traders de Cambridge et les autres)