Atelier Java - séance 4 et 5

Objectifs :

1. Comprendre les intentions de l'API Stream
2. Comprendre les expressions lambda et leurs interfaces fonctionnelles
3. Écrire des codes permettant d'extraire les informations souhaitées grâce à l'API Stream.
4. Créer des streams de différentes façons : à partir d'une collection, depuis un fichier, à partir de rien.

Thèmes abordés :

→ API Stream

Créez un nouveau module **workshop4** dans votre projet d'Ateliers Java. Récupérez les sources dans Moodle et importez-les dans votre nouveau module.

Exercice 1 : expressions Lambda

Dans cet exercice, vous utiliserez des `Predicate` et des `Function`. L'utilisation de ces objets est décrite dans la théorie à partir de la page 3 : "Les fonctions en Java". Vous devrez utiliser la méthode `test` des `Predicate` et la méthode `apply` des `Function`.

Remarque : ne passez **pas** par l'API Streams pour faire les points 1 à 4.

1. Dans la classe `Lambda` du package lambda, implémentez les méthodes `allMatches` et `transformAll` en respectant la javadoc fournie.
2. Modifiez la classe `TestLambda` en fournissant des expressions lambda qui correspondent aux indications en commentaires. Utilisez-la ensuite pour tester vos méthodes `allMatches` et `transformAll`.
3. Modifiez vos méthodes pour qu'elles puissent maintenant accepter des `List` de n'importe quel type (au lieu de uniquement des `Integers`).
4. Dans la classe `TestLambda`, décommentez la deuxième partie et complétez les expressions lambda.
5. Ajoutez deux méthodes à votre classe Lambda, appelées `filter` et `map`. Ces deux méthodes doivent faire exactement la même chose que `allMatches` et `transformAll`, mais **doivent** utiliser l'API Stream.

Exercice 2 : streams de Tweets

Complétez les méthodes de la classe `TestTweets` du package tweets en suivant les indications dans les TODOS. Vous devez à chaque fois afficher le résultat grâce à `System.out.println`.

Vous remarquerez les dates ne s'affichent de manière très agréable pour l'œil. Pour améliorer cela, nous allons utiliser un `DateTimeFormatter` dans le `toString()` de la classe `Tweet` pour modifier l'affichage de la date. Référez-vous à la théorie sur les dates p.2-3 : "Formatage des dates" pour plus d'informations.

Exercice 3 : streams de String

Complétez la classe `TestStrings` dans le package strings, en effectuant chaque exercice ci-dessous en une seule chaîne d'instructions :

1. Afficher chaque mot de la liste.
2. Produire une nouvelle liste de String dans laquelle on a ajouté "- " au début de chaque mot.
3. Placer tous les mots convertis en majuscules dans un tableau de String (pour savoir comment faire, consultez : https://stackoverflow.com/questions/23079003/how-to-convert-a-java-8-stream-to-an-array).
4. Produire une autre liste de String triée par ordre anti-alphabétique qui ne contient que les String originales dont la longueur est paire.

5. Fournir une String contenant tous les mots de longueur inférieure ou égale à 5, séparés par un espace, et en majuscule.
6. Compléter la méthode `premierMotDeLongueurContenant` qui prend en paramètre un int et un char. Cette méthode renvoie, sous forme de String, le premier mot de la liste de longueur égale à l'int reçu en paramètre, et qui contient le char reçu en paramètre. La String retournée **est convertie en majuscule**. Si aucun mot ne correspond, cette méthode renvoie "No match".
7. Produire une String qui contiendra la concaténation de tous les mots du stream en majuscule. On vous demande de le faire en 2 versions différentes : l'une avec map et l'autre sans. Utilisez la méthode `reduce`.
8. Fournir le nombre de mots qui se terminent par le caractère 's'.
9. Compléter la méthode `occurrences` qui retourne le nombre d'occurrences d'un caractère dans une String, puis fournir le nombre total de fois qu'apparaît le caractère 'e' dans la liste.

Exercice 4 : streams infinis de nombres

La classe `TestStreamDeNombres` du package infinis contient un tableau de doubles. Ce tableau est initialisé dans la `main`. Grâce à la classe `Random`, dont la méthode `doubles()` retourne un stream de doubles.

1. Grâce à la classe `DoubleStream`, vous pouvez créer un stream de `double` à partir d'un tableau via la fonction `of()`. Utilisez-la pour calculer la moyenne des racine carrées des réels du tableau.
2. Créez deux méthodes similaires : la première retourne un `DoubleStream` de réels aléatoires et la seconde, un `Stream<Double>` de réels aléatoires. Les réels contenus dans les streams doivent être strictement inférieurs à une valeur maximum donnée en paramètre. Utilisez la méthode `generate()` des classes `Stream` et `DoubleStream` pour y parvenir.
   Effectuez les exercices suivants avec chacune de vos deux méthodes :
   - Créer une liste de 10 réels aléatoires compris entre 0 et 100.
   - Construire un tableau de 20 réels aléatoires compris entre 0 et 100.
3. Dans la main, utilisez la méthode `generate()` pour afficher une liste de 20 nombres entiers pairs compris entre 0 et 100. N'utilisez pas d'autres méthode de l'API stream (excepté `limit`).

Exercice 5 : streams avec les fichiers

La boîte StreamingVF vous fournit un fichier contenant les informations de tous ses employés afin de procéder à des opérations et des vérifications. Le fichier est un csv : **streamingvf.csv**.

Ce fichier représente les employés par 5 attributs : un id de 5 chiffres, un nom en majuscules, un prénom, une adresse mail et une indication de si la personne travaille à plein temps (TP) ou à mi-temps (MT).

Afin d'utiliser correctement un fichier de ressources comme celui-ci avec IntelliJ, il faut l'importer dans un dossier spécifiquement destiné aux ressources (ce n'est pas obligatoire mais vivement recommandé). Sur la racine de votre module, clic droit >> new >> directory. Appelez ce dossier *resources*. Effectuez un clic droit sur ce nouveau dossier et sélectionnez Mark Directory as >> Resources Root Déposez ensuite le csv dans le dossier. De cette manière, le chemin d'accès à votre fichier sera toujours "*./resources/streamingvf.csv*".

Attention, si vous utilisez les modules d'IntelliJ, vous devrez préciser le nom du dossier contenant le module au début du chemin : "*./<module>/resources/streamingvf.csv*".

Dans le package `employee`, deux classes sont à votre disposition : `Employee`, qui représente un employé (attention aux attributs de la classe `Employee`, TP devient `true` et MT `false` dans l'attribut `fullTime`), et `EmployeeManagement`.

Complétez la classe `EmployeeManagement` avec les streams et le `try-with-resource` (https://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html) pour effectuer les opérations suivantes :

1. Afficher la première ligne du csv. Vous devrez au préalable compléter le `Supplier` en début de classe.

Pour la suite, vous devrez construire des Stream d'employés. Le constructeur d'`Employee` prend une String en paramètre et la splitte là où se trouvent des ";" (un fichier csv sépare ses colonnes par des ";"), afin d'initialiser ses attributs correctement. Vous utiliserez cette fonction pour les exercices suivants au lieu du `try-with-resource`.

2. Construire une liste avec les noms de famille de plus de 8 caractères qui contiennent la lettre 'O' ou 'K'. Il faut compléter le prédicat et n'appeler qu'une seule fois la fonction `filter`.
3. Contruire une BiFunction (une Function avec deux paramètres) qui compte les occurrences d'un character passé en paramètre dans le prénom de l'employé passé en paramètre. Utiliser cette BiFunction pour créer une liste du compte d'occurrences du caractère 'e' dans les prénoms de chaque employé.
4. Indiquer si tous les employés ont un email se terminant par @streamingvf.be.
5. Indiquer le prénom d'un employé dont le nom de famille fait plus de 14 caractères ou « None » s'il n'en existe pas.
6. (Optionnel) Retourner le nombre d'employé à mi-temps du fichier csv.
7. (Optionnel) Construire une Map<Boolean,List<Integer>> regroupant les id des employés selon qu'ils travaillent à plein temps (`true`) ou à mi-temps (`false`).
8. (Optionnel) Complétez la fonction `withLines(Consumer<Stream<String>> consumer)` qui s'occupe d'encapsuler le try-with-resource et son catch et fournira le stream de lignes au consumer qui lui est fourni. Complétez ensuite la fonction `printLongestName()` qui utilise `withLines` pour imprimer le plus long nom de famille du fichier.

Exercice 6 : répartitions de Tweets

Reprenons les Tweets de l'exercice 2. Complétez la classe `TweetDistribution` afin de construire, grâce aux streams, les collections suivantes :

1. `Map<User, List<Tweet>>` (les tweets de l'auteur)
2. `Map<User,Long>` (nombre de tweets de cet auteur)
3. `Map<User,Integer>` (nombre de total de retweets pour chaque auteur)
4. `Map<Boolean,List<Tweet>>` (les tweets des users à Bruxelles et les autres)

5. `Map<Visibility,List<Tweet>>` (les tweets, catégorisés selon le nombre de retweet)

Pour le point 5, utilisez l'enum `Visibility {`***`FAMOUS, POPULAR, NORMAL, UNKNOWN`***`}`. Les tweets sont classés selon le nombre de retweets :
- si retweets >= 2000, `FAMOUS` ;
- si 800 <= valeur <2000, `POPULAR` ;
- si 100 <= valeur < 800, `NORMAL` ;
- si <100, `UNKNOWN`.

Aidez-vous de la théorie p.17 : "Regrouper", et p.18 : "Partitionner"

Exercice 7 : Streams parallèles (Optionnel)

Un stream parallèle va traiter en parallèle les différents éléments de la collection qu'il représente.

L'interface DelayedOperations contient des opérations qui prennent un certain temps à s'exécuter :

- `fastMult2(Integer value)` qui retourne `2*value` après avoir attendu 1 ms.
- `slowMult2(Integer value)` qui retourne `2*value` après avoir attendu 10 ms.
- `ultraSlowMult2(Integer value)` qui retourne `2*value` après avoir attendu 100 ms.
- `randomlySlowMult2(Integer value)` qui retourne `2*value` après avoir attendu 500 ms si `value` est un multiple de 10, et après 1 ms sinon.

Elle contient également la fonction `runAndRecordTime(Runnabe process)` qui prend un processus en paramètre, l'exécute et retourne son temps d'exécution (référez-vous aux commentaires et à l'exemple qui se trouve dans la classe `ParallelStreams` pour voir comment l'utiliser).

Complétez la classe ParallelStreams :

1. Initialisez la liste `numbers` à l'aide d'un `IntStream` pour obtenir une liste d'entier de 1 à 100.
2. Complétez la fonction `serialMap()` qui retourne le temps nécessaire à appliquer chacune des 3 transformations, l'une après l'autre, à un stream issu de `numbers`.
3. Complétez la fonction `parallelMap()` qui retourne le temps nécessaire à appliquer chacune des 3 transformations en parallèle à un stream issu de `numbers`. Comparez le résultat avec celui de `serialMap`. À quoi correspond le gain de temps ?
4. Complétez les fonctions `serialFilteredBefore()`, `serialFilteredAfter()`, `parallelFilteredBefore()` et `parallelFilteredAfter()`, qui vont filtrer un stream issu de `numbers` en ne gardant que les nombres pairs avant (resp. après) d'y appliquer `ultraSlowMult2()` et retourner le temps d'exécution. Comparez les résultats et repérez l'influence de la mise en parallèle.
5. Complétez la fonction `randomMap()` qui retourne le temps nécessaire à appliquer `randomlySlowMult2` à un stream issu de `numbers`. Combien de temps devrait prendre cette méthode en théorie ? Quelle est la réalité ?