TD et TP n° 7 : Streams

Très important : L'exercice noté de la semaine prochaine (semaine du 15 novembre) portera sur le TP6 : Exo 1, 2 et 3. Pensez à vous inscrire à votre groupe si ce n'est déjà fait, sinon vous ne pourrez pas lire le sujet.

Priorité : Si vous ne les avez pas finis, faites les exercices 1 à 3 du TP6.

Important : Il est **vivement conseillé** de consulter la page : `https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/stream/Stream.html`.
Il est de façon générale toujours conseillé d'aller voir l'API et/ou le tutorial de java lorsque l'on veut des informations précises et complètes. Les forums permettent assez souvent de dépanner, mais ne sont pas toujours assez précis ni exhaustifs.

Exercice 1 : Conversions de *pipelines* de *streams* en boucles `for`.

Pour cet exercice et les suivants, utilisez les fichiers fournis. Essayez de deviner ce que calculent les instructions avant de les testez.

1. Expliquez ce que calculent les instructions suivantes puis traduisez-les en boucles `for` qui calculent la même valeur dans une variable.
   (a)
   ```
   int x = appartements.stream()
      .filter(a -> a.nbPieces >= 3)
      .map(a -> a.prix)
      .reduce(Integer.MAX_VALUE, (a, b) -> (a<b)?a:b);
   ```
   (Les attributs des classes fournies ne sont publics que dans le but de ne pas vous encombrer avec des appels de méthodes, mais, bien sûr, il ne devraient pas l'être.)
   (b) Que se passe-t'il si, par exemple, la liste est vide ? Que faudrait-il faire pour que le résultat ait un sens quoiqu'il arrive ?
2. Même exercice avec :
   ```
   List<PetitObjet> inventaire = maison.meubles.stream()
      .flatMap(m -> m.contenu.stream())
      .collect(Collectors.toList())
   ```

Exercice 2 : Conversions de boucles `for` en *pipelines* de *streams*.

Écrivez les blocs `for` suivants comme *pipelines* de *streams* (en utilisant des lambda-expressions).

1.
   ```
   for (Appartement a : appartements) {
      if (a.prix<=200_000) {
         System.out.println(a.lieu);
      }
   }
   ```
   Où `Appartement` est la classe vue dans l'exercice 1. Utilisez l'opération intermédiaire `filter` et l'opération terminale `forEach`.
2.
   ```
   int pdsTotal = 0;
   for(Meuble m: maison.meubles)
      for(PetitObjet obj : m.contenu)
        pdsTotal += obj.poids;
   System.out.println(pdsTotal);
   ```

Exercice 3 : Requête avec des streams

On vous donne sur Moodle les deux classes `Album` et `Track` ainsi qu'un début de main où est initialisée une liste d'albums.
Essayez de répondre à ces questions en utilisant les streams.

1. Retournez une chaîne de caractères contenant la liste des titres des albums séparés par un point virgule. Utilisez entre autre `reduce`. (S'il y a un point virgule au début, ce n'est pas grave, on verra ça plus tard).

   (facultatif) Comment supprimer ce point-virgule en trop ?
2. Retournez une liste des albums ayant 4 tracks. (Utilisez `collect` à la fin).
3. (facultatif) On veut maintenant sous forme de chaîne de caractères, la liste des albums avec pour chaque la liste de leur tracks. (Il faut utiliser deux niveaux de streams).

   L'affichage de la chaîne devra donner ceci :

```
Latin Music 1:_lorem_ipsum_sit
Latin Music 2:_dolor_amet_adispising
More Latin Music:_elit_lorem_tempor_aliqua
Latin Music For Ever:_elit_enim_amet_aliqua
Common Latin Music:_minim_veniam_laboris
```

4. Convertissez le code suivant en une instruction qui utilise les lambda-expressions et les opérations d'agrégation au lieu de boucles **`for`** imbriquées. Astuce : construisez un *pipeline* invoquant les opérations `filter`, `anyMatch`, `sorted` et `collect` dans cet ordre.

```
      List<Album> favs = new ArrayList<>();
      for (Album a : albums) {
         boolean hasFavorite = false;
         for (Track t : a.tracks) {
            if (t.rating >= 4) {
               hasFavorite = true;
               break;
            }
         }
         if (hasFavorite)
            favs.add(a);
      }
      Collections.sort(favs, Comparator.comparing(a -> a.name));
```