TD et TP nº 8 : Streams
(Correction)

Astuce : pour ces exercices, il est parfois possible d'utiliser les conversions automatiques de votre IDE. Dans IntellIJ IDEA, placez votre curseur sur le mot-clé `for` *ou sur une des opérations d'un stream, faites "alt + entrée", puis regardez les choix proposés.*
Ceci permet de gagner du temps et garantit des transformations correctes, mais ne permet pas toujours de voir toutes les transformations possibles. Il reste plus que conseillé de comprendre ce que fait le code à tranformer avant de le manipuler.

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

Expliquez ce que calculent les instructions suivantes puis traduisez-les en boucles `for` qui calculent la même valeur dans une variable.

1.
```
int x = appartements.stream()
   .filter(a -> a.nbPieces >= 3)
   .map(a -> a.prix)
   .reduce(Integer.MAX_VALUE, (a, b) -> (a<b)?a:b);
```

Ici, `appartements` est une liste d'`Appartement` où `Appartement` est la classe suivante :

```
public class Appartement {
   public int nbPieces;
   public int prix;
   public String lieu;
   ...
}
```

Correction :

```
int x = Integer.MAX_VALUE;
for (Appartement a : appartements)
   if (a.nbPieces >= 3 && a.prix < x)
      x = a.prix;
```

2.
```
List<PetitObjet> inventaire = maison.meubles.stream()
   .flatMap(m -> m.contenu.stream())
   .collect(Collectors.toList())
```

Où `maison` est de type `Maison` et l'on suppose définies les classes suivantes :

```
public class PetitObjet { }
public class Meuble { List<PetitObjet> contenu; }
public class Maison { List<Meuble> meubles; }
```

Correction :

```
List<PetitObjet> inventaire = new LinkedList<>();
for (var meuble : maison.meubles)
   for (var petitObjet : meuble.contenu)
      inventaire.add(petitObjet);
```

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. (inspiré du tutoriel d'Oracle)

```
for (Person p : roster) {
   if (p.getGender() == Person.Gender.FEMALE) {
      System.out.println(p.getName());
   }
}
```

On suppose ici les types `Person` et `Gender` déjà définis, ainsi que leurs membres utilisés dans l'exemple.
. Astuce : utilisez l'opération intermédiaire `filter` et l'opération terminale `forEach`.

Correction :

```
roster.stream()
      .filter(p -> p.getGender() == Person.Gender.FEMALE)
      .map(Person::getName)
      .forEach(System.out::println);
```

2.

```
int effectif = 0;
for (var c : universite.composantes)
   for (var f : c.filieres)
      for (var e : f.etudiants)
         effectif++;
```

Où `universite` est de type `Universite` et sont définies les classes suivantes :

```
public class Universite { List<Composante> composantes; }
public class Composante { List<Filiere> filieres; }
public class Filiere { List<Etudiant> etudiants; }
public class Etudiant { }
```

Correction :

```
universite.composantes.stream()
   .flatMap(c -> c.filieres.stream())
   .flatMap(f -> f.etudiants.stream())
   .count();
```

Exercice 3 : Requête avec des streams (partiellement inspiré du tutoriel d'Oracle)

On vous donne sur Moodle les deux classes `Album` et `tracks` 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).

Correction :

```
1   System.out.println(albums.stream().map(a -> a.name).reduce("",(s,t)-> s+
        "; "+t));
```

Comment supprimer ce point-virgule en trop ?

Correction :

```
System.out.println(albums.stream()
           .map(a -> a.name)
           .reduce("",(s,t)-> (s.equals("")?t:s+ "; "+t)));
```

2. Retournez une liste des albums ayant 4 tracks. (Utilisez `collect` à la fin).

Correction :

```
System.out.println(albums.stream()
           .filter(a -> a.tracks.size() == 4)
           .map(a -> a.name).collect(Collectors.toList()));
```

3. 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
```

Correction :

```
   System.out.println(albums.stream()
           .map(a -> a.name
               + a.tracks.stream().map(t -> t.name).reduce(":",(s,t)-> s+
                   "_"+t)
           )
           .reduce("",(s,t)-> s+ "\n"+t));
```

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));
```

Correction :

```
     List<Album> favs = albums.stream()
           .filter(a -> a.tracks.stream()
           .anyMatch(t -> t.rating >= 4))
           .sorted(Comparator.comparing(a -> a.name))
           .collect(Collectors.toList());
```