Lambda, method reference et stream

Exercice 1 - Le compte est bon

Le but de cet exercice est de découvrir comment utiliser des lambdas et des streams.

1. On cherche à compter le nombre d'occurences d'un mot dans une liste.

```
var list = List.of("hello", "world", "hello", "lambda");
System.out.println(count(list, "hello"));  // 2
```

   Écrire le code de la méthode `count` sachant que le compteur est un entier long.

2. On cherche à écrire une méthode `count2` sémantiquement équivalente (qui fait la même chose) à `count` mais en utilisant l'API des **Stream**.
   Comment obtenir un `Stream` à partir d'un objet de type `List` ?
   L'idée, ici, est de filtrer le stream pour ne garder que les mots égaux au mot passé en paramètre puis de compter ceux-ci.
   Quel sont les méthodes permettant respectivement de filtrer un stream et de compter le nombre d'éléments ?
   La méthode qui permet de filtrer prend un objet de type `Predicate<T>` en paramètre. Dans notre cas, quel est le type correspondant à `T` ?
   Indiquer le code permettant de créer une lambda filtrant sur le mot passé en paramètre que l'on peut déclarer en tant que `Predicate`
   Écrire le code de la méthode `count2`.

Exercice 2 - En majuscule

Le but de cet exercice est de découvrir comment utiliser, en plus des lambdas et des streams, des " method references ".

1. On cherche à écrire une méthode prenant en paramètre une liste de chaînes de caractères et renvoyant une nouvelle liste contenant les chaînes de caractères en majuscules.

```
var list = List.of("hello", "world", "hello", "lambda");
System.out.println(upperCase(list));  // [HELLO, WORLD, HELLO, LAMBDA]
```

   Écrire la méthode `upperCase` (dans un premier temps) sans utiliser l'API des `Stream`.

2. On cherche maintenant à écrire une méthode `upperCase2` faisant la même chose mais avec un `Stream`.
   Comment peut-on utiliser la méthode `Stream.map` ici ?
   Pour stocker le résultat dans une nouvelle liste, l'idée est de créer la liste puis d'ajouter chaque mot dans la liste.

```
public static List<String> upperCase2(List<String> words) {
  var uppercases = new ArrayList<String>();
  ...
```

   pour demander l'ajout, on utilisera sur le stream la méthode `forEach`.
   Écrire le code de la méthode `upperCase2` en utilisant des lambdas.

3. En fait, au lieu d'utiliser des lambdas, il est possible dans cet exemple d'utiliser la syntaxe des références de méthodes avec l'opérateur `::` (coloncolon).
   Écrire une méthode `upperCase3` qui utilise la syntaxe des référence de méthodes.

4. En fait, au lieu d'utiliser `forEach`, il est mieux (pas d'effet de bord) d'utiliser la méthode `collect` avec comme **Collector** celui renvoyé par la méthode **Collectors.toList()**.
   Écrire une méthode `upperCase4` en utilisant le collector `Collectors.toList()`.

Exercice 3 - Comptons sur une réduction

Le but de cet exercice est de découvrir comment effectuer une réduction sur un stream.
Lors du premier exercice, nous avons utilisé la méthode `count` qui retourne un entier long. On souhaite maintenant écrire une nouvelle

méthode `count3` qui renvoie un entier sur 32 bits.
Pour cela, une fois les mots filtrés, nous allons transformer (avec `map`) chaque mot en 1 (le nombre) puis nous allons, avec la méthode `reduce`, faire l'agrégation des valeurs.

1. Expliquer pourquoi nous n'allons pas utiliser la méthode `map` mais la méthode `mapToLong` ?
2. Écrire le code de la méthode méthode `count3`.

Exercice 4 - Evaluation de vitesse

On cherche à savoir, parmi les 3 façons d'écrire `count`, quelle est la plus rapide.
Nous allons pour cela utiliser une liste définie par le code suivant

```
var list2 =
  new Random(0)
    .ints(1_000_000, 0, 100)
    .mapToObj(Integer::toString)
    .collect(Collectors.toList());
```

1. Expliquer ce que contient la variable locale `list2`.
2. On cherche à écrire une méthode `printAndTime` permettant de calculer le temps d'exécution de l'appel à la méthode `count` sur la `list2`.
   Pour calculer le temps d'exécution, on demande le temps avant puis le temps après et si l'on soustrait les deux temps, on trouve le temps d'exécution.

   ```
   var start = System.nanoTime();
   ... // faire le calcul
   var end = System.nanoTime();
   System.out.println("result " + result);
   System.out.println(" elapsed time " + (end - start));
   ```

   Écrire le code de `printAndTime`.
3. On souhaite également calculer le temps d'exécution avec d'autres méthodes, comme `count2` par exemple. Comment faire pour NE PAS dupliquer le code pour le calcul du temps d'exécution?
   Quelle **interface fonctionnelle** doit-t-on utiliser sachant que l'on va appeler `printAndTime` de la façon suivante?

   ```
   printAndTime(() -> count(list2, "33"));
   printAndTime(() -> count2(list2, "33"));
   printAndTime(() -> count3(list2, "33"));
   ```

   Écrire le nouveau code de `printAndTime`.
4. Expliquer les résultats obtenus.
