Exercices sur les itérateurs et les lambdas

1 Feu Tricolore

Un feu tricolore se compose de trois couleurs (on peut utiliser la classe `Color` en Java, avec des valeurs définies sur le modèle de la constante `Color`.red). Itérer sur le feu tricolore consiste à parcourir (sans fin a priori) les trois couleurs. Ecrire une classe pour représenter les feux tricolores, un itérateur de feu tricolore et un programme montrant une utilisation. L'opération `remove` n'est pas supportée, ce que vous représenterez par le signalement de l'exception `UnsupportedOperationException`.

2 Itérer sur un objet complexe

Une feuille de salaire comprend les éléments suivants qui peuvent être représentés par des attributs dans une classe `FeuilleSalaire` :

- employeur
- salarié
- convention collective
- nombre d'heures payées
- prélèvements fiscaux : CRDS, CSG, cotisations salariales
- net à payer

Définir une classe `FeuilleSalaire` itérable. Définir un itérateur de feuille de salaire qui retourne successivement chacun des champs de la feuille de salaire. Cela peut servir par exemple pour écrire une méthode d'affichage ou pour récupérer toutes les informations qui sont des nombres (définir de telles opérations en utilisant l'itérateur). L'opération `remove` n'est pas supportée, ce que vous représenterez par le signalement de l'exception `UnsupportedOperationException`.

Puis écrivez un programme mettant en œuvre une feuille de salaire et un itérateur.

3 Premières utilisations de lambdas

Définissez une classe `CollectionFeuillesSalaire` avec une liste de feuilles de salaire `listeFeuilles` en attribut.

Question 1 *Ecrivez une méthode `print` qui affiche successivement toutes les feuilles de salaire de `listeFeuilles` sans utiliser de boucle `for` mais en utilisant `stream()` et `forEach()`.*

Question 2 *Même question mais en utilisant cette fois la méthode forEach maintenant disponible (depuis Java 8) pour les listes, et une lambda.*

Question 3 *Ecrivez pour la classe `CollectionFeuillesSalaire` trois méthodes permettant de trier `listeFeuilles` par ordre de nombre d'heures payées croissant : sans lambdas ni classes anonymes, sans lambdas et avec classes anonymes, puis enfin avec lambdas. Testez.*

Question 4 *Ecrivez une méthode qui retourne les feuilles de salaire des salariés d'une l'entreprise donnée. Testez. Pour récupérer une liste à partir d'un stream, on utilisera : `stream.collect(Collectors.toList());`. Testez.*

Question 5 *Ecrivez une méthode qui retourne la liste des entreprises des feuilles de salaire de `listeFeuilles` dont le nom contient par une chaîne de caractère donnée. Testez.*

4 Encore des lambdas

On dispose d'une classe `Etudiant` minimale (un nom, un âge, une modalité de candidature codée par une énumération à 3 valeurs : `eCandidat`, `CampusFrance`, `autre`, et les accesseurs en lecture associés, ainsi qu'un constructeur paramétré). On écrit une classe `Promotion` qui dispose d'une liste d'étudiants. Les trois questions suivantes sont à réaliser sans écrire de boucle for ou while.

Question 6 *Ecrivez dans la classe* `Promotion` *une méthode qui retourne l'âge minimum des étudiants de la promotion recrutés via eCandidat.*

Question 7 *Ecrivez dans la classe* `Promotion` *une méthode qui retourne le nombre d'étudiants recrutés via une modalité de candidature donnée.*

Question 8 *Ecrivez dans la classe* `Promotion` *une méthode qui retourne la liste des étudiants d'âge maximal.*

5 Toujours des lambdas

Question 9 *Etudiez la méthode Stream.generate dont la signature est :*
`static <T> Stream<T> generate(Supplier<T> s)` *et la descrition est : Returns an infinite sequential unordered stream where each element is generated by the provided Supplier.*

Supplier est une interface fonctionnelle :

```
@FunctionalInterface
public interface Supplier<T>{
T get() ; // Gets a result
}
```

Question 10 *Utilisez cette méthode pour créer une méthode retournant le flux d'entiers correspondant à la suite de fibonacci.*

Question 11 *En utilisant la méthode* `limit` *sur les Streams et votre méthode précédente, écrivez une méthode qui affiche les n premiers entiers de la suite de Fibonacci, et une autre qui affiche les n premiers entiers pairs de la suite de Fibonacci.*

6 Itérateur sur une suite réelle

Nous souhaitons représenter des suites réelles, et plus précisément nous nous limitons ici aux suites constantes à partir d'un certain rang. On rappelle qu'une suite réelle *est une application u d'une partie de $\mathcal{N}$ dans $\mathcal{R}$. On considère ici les suites définies sur $\mathcal{N}$.*

Une suite est constante à partir d'un certain rang *s'il existe un entier naturel n et un réel a tels que pour tout entier naturel $n' \geq n$, $u(n') = a$.*

Question 12 *Ecrivez une classe* `SuiteConstanteRang` *pour représenter les suites constantes à partir d'un certain rang. On pourra stocker les $n-1$ premiers éléments dans une* `ArrayList`*.*

Question 13 *Ecrivez une classe* `IterateurSuiteConstanteRang` *qui permette d'itérer sur une suite constante à partir d'un certain rang. Un élément a toujours un successeur. L'opération* `remove` *ne pourra enlever que des éléments entre les rangs 1 et $n-1$.*

Question 14 *Rendez la classe* `SuiteConstanteRang` *itérable.*

Question 15 *Que va écrire le programme suivant et comment l'interprétez-vous ? Est-ce que cela mérite une modification de ce qui a été précédemment écrit ?*

```
public static void main(String[] argv){
        ArrayList<Double> a = new ArrayList<Double>();
        a.add(3.0); a.add(4.0); a.add(5.0); a.add(6.0); a.add(7.0);
        SuiteConstanteRang suite = new SuiteConstanteRang(a,8);
        Iterator<Double> it = suite.iterator();
        for (int i=0; i<14; i++)  System.out.println(it.next());
        for (double e : suite) System.out.println(e);
}
```

7 Généralisations

(1) Tentez de généraliser l'itérateur sur une feuille de salaire à un itérateur sur un objet quelconque. Analysez les difficultés que vous rencontrez.
(2) Peut-on associer plusieurs types d'itérateurs à une classe d'objets itérables ? Par exemple, essayez d'ajouter à votre précédent programme un itérateur de suite constante qui ne retourne que les valeurs de rang pairs ($u(0)$*,* $u(2)$*,* $u(4)$*, ...) et un itérateur qui ne retourne que les valeurs de rang impairs (*$u(1)$*,* $u(3)$*,* $u(5)$*, ...) . Comment les utiliser conjointement dans un programme ?*