TD et TP de Compléments en Programmation Orientée Objet n° 4 : Polymorphisme, Interface

Exercice 1 : Générateurs de nombres

Attention : pour faire cet exercice, il faut savoir implémenter une interface (`Generateur`) et comprendre le polymorphisme par sous-typage (toute méthode retournant un `Generateur` a le droit de retourner une instance de classe implémentant `Generateur`). Ces notions ne seront révisées qu'au prochain cours mais devraient déjà avoir été abordées en L2.

On définit l'interface générateur :

```
1  interface Generateur { int suivant(); }
```

Objectif : écrire la classe-bibliothèque `GenLib`, permettant de créer des générateurs de toute sorte (entiers au hasard, suites arithmétiques, suites géométriques, fibonacci, etc.), sans pour autant fournir d'autre classe publique que `GenLib` elle-même dans l'API de cette bibliothèque.

Méthode : utiliser le schéma suivant : `GenLib` (classe non instanciable) contient une série de fabriques statiques permettant de créer les générateurs. Chaque appel à une fabrique instancie une classe imbriquée en utilisant les paramètres passés.

Attention : la classe `GenLib` n'implémente pas elle-même l'interface `Generateur` (ça n'aurait pas de sens, puisqu'elle n'est pas instanciable). Ses méthodes ne renvoient pas de `int` !

Exemple d'utilisation : pour afficher les 10 premiers termes de la suite de Fibonacci

```
1  Generateur fib = GenLib.nouveauGenerateurFibonacci();
2  for (int i = 0; i < 10; i++) System.out.println(fib.suivant());
```

Questions :

1. Des 4 genres de classes imbriquées vues en cours (membre statique, membre non statique, locale, anonyme), l'un ne peut pas être utilisé ici, lequel ?
2. Programmez les méthodes statiques permettant de créer les générateurs suivants :
   — générateur d'entiers aléatoires (compris entre 0 et $m-1$, m étant un paramètre)
   — suite arithmétique : $0, r, 2r, 3r, \ldots$ (r étant un paramètre)
   — suite géométrique : $1, r, r^2, r^3, \ldots$ (r étant un paramètre)
   — suite de Fibonnacci : $1, 1, 2, 3, 5, 8, 13, \ldots$
   Variez les techniques : montrez un exemple pour chaque genre de classe imbriquée.
3. Écrivez un `main()` qui demande à l'utilisateur de choisir entre les différents types de suite (et éventuellement d'entrer un paramètre), puis instancie le générateur de suite correspondant et en affiche ses 10 premiers termes.
4. Réécrivez votre code à la mode Java 8 : là où c'est possible, remplacez une de vos classes imbriquées par une lambda-expression.

Exercice 2 : Tris

Le tri à bulles est un algorithme classique permettant de trier un tableau. Il peut s'écrire de la façon suivante en Java :

```java
static void triBulles(int tab[]) {
    boolean change = false;
    do {
        change = false;
        for (int i=0; i<tab.length - 1; i++) {
        if (tab[i] > tab [i+1]) {
            int tmp = tab[i+1];
            tab[i+1] = tab[i];
            tab[i] = tmp;
            change = true;
        }
        }
    } while (change);
}
```

Cette implémentation du tri à bulles permet de trier un tableau d'entiers. Maintenant on veut pouvoir utiliser le tri à bulles sur tout autre type de données représentant une suite (séquence) d'objets comparables. Pour cela, on considère les interfaces suivantes :

```java
public interface Comparable {
    public Object value(); // renvoie le contenu
    public boolean estPlusGrand(Comparable i);
}

public interface Sequencable {
    public int longeur(); // Renvoie la longueur de la sequence
    public Comparable get(int i); // Renvoie le ieme objet de la sequence
    public void echange(int i, int j); // Echange le ieme object avec le jieme objet
}
```

1. Écrivez une méthode `affiche()` dans l'interface `Sequencable` permettant d'afficher les éléments de la séquence du premier au dernier. (Utilisez la fonction `toString()` de Object.)
2. Écrivez une méthode `triBulle` dans l'interface `Sequencable` qui effectue un tri à bulles sur la séquence.
3. Écrivez une classe `MotComparable` représentant un mot et implémentant l'interface `Comparable` de tel sorte que `estPlusGrand(Comparable i)` :
   — quitte sur une exception (`throw new IllegalArgumentException();`) si `i.value()` n'est pas un sous-type de `String`,
   — retourne vrai si le contenu est plus grand lexicographiquement que `i.value()`, faux sinon.

   N'oubliez pas les constructeurs () et la méthode `toString()`.
4. Écrivez une classe `SequenceMots` qui représente une séquence de `MotComparable` et qui implémente `Sequencable`.

   Écrivez un constructeur prenant un tableau de `String`.
5. Testez votre code.

   Vous pouvez passer en paramètre un tableau de chaînes aléatoires générées avec l'instruction `Integer.toString((int)(Math.random()*50000))` (ou utilisez un des générateurs de l'exercice précédent).