2. Feuille de TD

semaine 13 :

Un tableau de statistiques

On vous demande de définir un tableau enrichi de diverses statistiques. Pour cela, vous définirez une classe **Statistiques** qui hérite d'un ArrayList d'entiers et qui possède trois attributs de type entier : une variable *nbAjoutTotal*, une variable *nbNegatifs* et une variable *somme*.

- L'attribut **nbAjouTotal** permettra de fournir le nombre d'ajout (add) qui aura été fait dans le tableau.
- L'attribut **nbNegatifs** permettra de connaître le nombre d'entiers négatifs présents dans le tableau.
- Et enfin **somme** contiendra la somme des éléments du tableau.

La classe **Statistiques** doit pouvoir s'utiliser exactement de la même façon que la classe **ArrayList** seul l'affichage changera puisqu'il devra fournir les diverses statistiques.

Ainsi le main suivant fournira les affichages mis en commentaires :

```
public class Exec {
   public static void main(String [] args) {
       Statistiques stat = new Statistiques();
       stat.add(4);
       stat.add(-5);
       stat.add(3);
       stat.add(-3); // [4, -5, 3, -3]
       stat.remove(2); // supprime l'élément en position 2
       stat.set(2, -6);
       System.out.println(stat);
       // [4, -5, -6] nbAjout 4 nbNégatifs 2 somme -7
       List<Integer> l = new ArrayList<>();
       l.add(4);
       l.add(-5);
       l.add(3);
       l.add(-3);
       l.remove(2);
       l.set(2, -6);
       System.out.println(l); // [4, -5, -6]

   }
}
```

Votre classe **Statistiques** doit donc contenir un *constructeur* qui initialise les attributs.

Elle doit redéfinir la méthode *add* (son profil doit être celui de l'ArrayList soit **public boolean add(Integer v)**) qui permet de mettre à jour les attributs statistiques.

Les deux méthodes *remove* et *set* doivent également être redéfinies puisque les attributs **nbNegatifs** et **somme** seront mofifiés.

Profil de *remove* : **public Integer remove(int index)**

Profil de *set* : **public Integer set(int index, Integer nv)**

Transformation de code

1. Ré-écrivez le code suivant de façon à ne pas utiliser l'héritage.

```
import java.util.ArrayList;
public class ClavierTelephone extends ArrayList<String>
{
  ClavierTelephone()
  {
      this.add(" ");   // touche 0
      this.add("");    // touche 1
      this.add("abc"); // touche 2
      this.add("def"); // touche 3
      this.add("ghi");
      this.add("ghi");
      this.add("jkl");
      this.add("jkl");
      this.add("mno");
      this.add("pqrs");
      this.add("tuv");
      this.add("wxyz");
  }

  char getCharactere(int touche, int nbAppuis)
  {
      return get(touche).charAt(nbAppuis);
  }
}
```

2. Re-écrivez le code suivant de façon à utiliser l'héritage.

```
import java.util.Map;
import java.util.HashMap;

public class AnnuaireTelephonique{
   Map<String, String> contenu;
   // à chaque nom de personne est associé un numéro de téléphone
   // Par exemple : 'john' -> "02 38 38 12 12"
   AnnuaireTelephonique(){
       contenu = new HashMap<String, String>();
   }

   void ajouteContact(String nom, String numero)
   {
       this.contenu.put(nom, numero);
   }

   String get(String nom)
   {
       return this.contenu.get(nom);
   }
}
```

3. Modifiez le code précédent pour qu'une exception soit levée dans le code la méthode ajouteContact dans le cas où le numéro passé en argument n'a pas la forme "XX XX XX XX XX". Dans un premier temps, faites simple en considérant qu'un numéro n'a pas la bonne forme s'il ne fait pas 14 caractères.

Des Notes

Informations

La classe `Hashset` permet d'implémenter des ensembles. Elle possède entre autres les méthodes suivantes :

- `add(E e)` : ajoute un élément à l'ensemble s'il n'est pas déjà présent.
- `contains(Object o)` : retourne vrai si l'élément spécifié est présent dans l'ensemble.
- `remove(Object o)` : supprime de l'ensemble l'élément spécifié.
- `size()` : retourne le nombre d'éléments de l'ensemble.

Dans cet exercice on veut modéliser une partition de musique. Aucune implémentation n'est demandée. Certaines méthodes peuvent ne pas avoir de code. Pour cela on suppose que l'on a la classe `Note` suivante :

```java
public class Note {
   public Note(int frequence, int duree);
   public int getDuree();
   public int getFrequence();
   public void jouer();
   public int hashCode(); // ceci permet de les mettre dans un HashSet.
}
```

Remarque : une note de musique est plus que cela, cette modélisation est bien incomplète !

Un **accord** est un ensemble de notes ; il a une durée égale à la durée de la plus longue de ses notes ; je le trouve harmonieux si toutes les fréquences des notes sont des puissances de 2.

1. Créez une classe `Accord` qui hérite de `HashSet<Note>`. Ajoutez une méthode `duree`, une méthode `estHarmonieux` et une méthode `jouer` (qui peut n'être qu'un affichage dans le terminal dans notre exemple).
2. Dans un exécutable, créez un accord de trois notes.
3. Une partition est la donnée d'une suite de notes et d'accords. On voudrait donc qu'une partition soit l'héritier d'un ArrayList de notes et d'accords. Quel est le problème ? Pour le résoudre, proposez une interface qu'Accord et Note pourraient implémenter.
4. Ceci implique de devoir toucher le code de `Note`. Malheureusement, on ne le peut pas : `Note` nous a été donnée compilée ! Essayez de résoudre le problème en créant une classe `NotePerso` qui hérite de `Note` et qui implémente votre interface.
5. Écrivez une classe `Partition` ; donnez lui une méthode `jouer()`. Dans un exécutable, créez une Partition et *jouez*-là..