HAI401I - Exercices sur :
les assertions et les exceptions en Java

1 Piles et piles bornées

Observer la description du type abstrait de données *Pile* :
- le type défini est *Pile*
- Opérations
  - initialiser()
  - empiler(Object element)
  - depiler() : Object
  - sommet() : Object
  - estVide() : boolean
- Préconditions
  - précondition (p.depiler()) : $\neg\ p.estVide()$
  - précondition (p.sommet()) : $\neg\ p.estVide()$
- Axiomes
  - Juste après l'instruction $p.empiler(t)$, on a $p.sommet() = t$
  - Juste après $p.initialiser()$, on a $p.estVide() = true$
  - Juste après $p.empiler(t)$, on a $p.estVide() = faux$
  - Juste après $p.empiler(t)$, on a $p.depiler() = t$

Une interface `IPile` représentant le type abstrait *Pile*, une classe `Pile` l'implémentant avec une `ArrayList` pour stocker les éléments et une classe exception `PileVideException` sont présentées ci-dessous.

```
public class PileVideException extends Exception {
     public PileVideException() { }
     public PileVideException(String message) { super(message); }
}

public interface IPile
{
     void initialiser();
     void empiler(Object t) throws PileVideException;
     Object depiler()throws PileVideException;;
     Object sommet() throws PileVideException;
     boolean estVide();
     int taille();
}

public class Pile implements IPile{
     // structure de stockage interne des éléments
     private ArrayList<Object> elements;

     // Mise en oeuvre des opérations
     public Pile(){initialiser();}

     public Object depiler() throws PileVideException{
          if (this.estVide())
               throw new PileVideException("en dépilant");
          Object sommet = elements.get(elements.size()-1);
```

```
        elements.remove(sommet);
        return sommet;
    }

    public void empiler(Object t) throws PileVideException {
        elements.add(t);
        assert this.sommet()==t : "dernier empile ="+this.sommet();
    }

    public boolean estVide() {return elements.isEmpty();}

    public void initialiser() {elements = new ArrayList<T>();}

    public Object sommet() throws PileVideException{
        if (this.estVide())
            throw new PileVideException("en dépilant");
        return elements.get(elements.size()-1);
    }

    public int taille(){return elements.size();}
    public String toString(){return "Pile = "+ elements;}
}
```

QUESTION 1 *Vous pouvez observer la manière dont la spécification est mise en œuvre, en particulier concernant les axiomes. La déclaration* `public void empiler(Object t) throws PileVideException` *peut paraître surprenante puisque dans la définition du type abstrait l'opération* `empiler` *n'a pas de pré-condition. Voyez-vous la raison pratique qui l'explique ? Peut-on écrire l'assertion autrement pour éviter la déclaration* `throws PileVideException` *?*

QUESTION 2 *Ajouter des assertions pour contrôler que :*

- *Après avoir construit la pile, celle-ci est vide.*
- *À la fin de l'opération* `depiler`*, le nombre d'éléments de la pile a diminué de 1.*
- *À la fin de l'opération* `empiler`*, le nombre d'éléments de la pile a augmenté de 1.*

QUESTION 3 *On désire créer un nouveau type de pile, représentant les piles bornées, c'est-à-dire des piles dont le nombre d'éléments doit rester inférieur à une certaine limite (taille maximale). Cette taille maximale peut évoluer pendant la vie de la pile.*

- *Proposez une description du type abstrait (en indiquant en quoi il diffère du type pile).*
- *Ecrivez les classes d'exceptions représentant les erreurs qui peuvent se produire sur les piles bornées : la pile est pleine et on ne peut plus ajouter d'éléments ; la taille maximale ne peut pas être négative ; la taille maximale est diminuée mais le nombre d'éléments déjà stockés est supérieur à cette nouvelle taille maximale.*
- *Proposez une interface pour représenter le type* `Pile bornée`*.*
- *Proposez une classe pour l'implémenter. Les méthodes comporteront des assertions et des signalements d'exceptions aux endroits nécessaires. Par défaut, la taille maximale est de 10 éléments.*
- *Ecrivez un programme utilisant cette classe. Est-ce qu'une pile bornée peut remplacer une pile : lors de la compilation (substituabilité syntaxique) ? lors de l'exécution (substituabilité comportementale) ?*

2 Invariant de boucle dans le tri par insertion

On vous donne ci-dessous le code d'une classe contenant une méthode réalisant le tri par insertion.

```
public class TriInsertion {
   public static void triInsertion(int tableau[]) {
      int taille = tableau.length;
      // Pour chaque élément du tableau
      for (int i = 1; i < taille; i++){
         int index = tableau[i];
         // On cherche sa place entre 0 et i-1
         // on décale vers la droite les éléments qui le précèdent
         // dans le tableau mais sont plus grand que lui
         int j = i-1;
         while(j >= 0 && tableau[j] > index)  {
            tableau[j+1] = tableau[j];
            j--;
         }
         // puis on le positionne
         tableau[j+1] = index;
         System.out.println("Etat du tableau à l'étape "+ i);
         System.out.println(Arrays.toString(tableau));
         }
      }

   public static void main(String str[]){
      int[] tab = {2,13,14,3,5,4,112,37,9,11,18,1};
      System.out.println("Etat du tableau avant le tri");
      System.out.println(Arrays.toString(tab));
      //tri par insertion
      triInsertion(tab);
      System.out.println("Etat du tableau après le tri");
      System.out.println(Arrays.toString(tab));
   }
}
```

Question 4

- *Quel invariant est vrai avant la boucle et à la fin de chaque étape ?*
- *Vérifiez cet invariant par une assertion.*
- *Vérifiez qu'à la fin de l'opération* `TriInsertion`*, le tableau est trié.*
- *Quelle précondition de l'opération* `TriInsertion` *pourriez-vous proposer ? Mettez-la en place avec un signalement d'exception.*

3 Invariants de classe

En utilisant des assertions et des exceptions, développez une classe représentant une personne avec les attributs et contraintes suivantes :

- une année de naissance qui doit être supérieure à 1900.
- un statut (vivant ou décédé).
- la qualité d'être majeur (attribut dérivé).
- un père et une mère qui sont des personnes différentes de la personne.
- le père et la mère ne peuvent avoir eu leur enfant qu'après l'âge de 16 ans.