TP5 : "Pile" économique en mémoire

Vous devez spécifier, implémenter et valider une `Pile` qui sécurise et économise la mémoire:

1. La méthode `empiler` réalloue de la place en cas de nécessiter (c'est-à-dire, lorsque la pile est pleine). Pour ce faire, vous utiliserez la méthode `modificationTailleZone`.
2. La fonction `depiler` réduit l'espace alloué à la pile lorsque la pile n'est pas assez remplie (il y a plus de 9 cases de libre). Pour ce faire, vous utiliserez la méthode `modificationTailleZone`.
3. Pas de pré-condition pour les méthodes publiques.

Installation

1. Ouvrez une "fenêtre de commandes" Windows (onglet Démarrer, puis Exécuter "cmd"). Dans la fenêtre de commandes, tapez la commande suivante ou faites un copier/coller:

```
> set Path=%Path%;\\info\Bibliotheque\MF-dev\TP-EscJava\windows\escjava\bin
```

2. Placez-vous dans le répertoire `<REPERTOIRE_TP>\Exercices\2.Pile\B`

```
> Z :
> CD <REPERTOIRE_TP>\Exercices\2.Pile\B
```

Pour utiliser ESC/Java, tapez la commande (Windows) suivante:

```
> escjava -loopSafe Pile.java
```

Le répertoire `\\info\Bibliotheque\MF-dev\TP-EscJava\Doc` contient de la documentation sur ESC/Java.

Invariant - Les invariants suivants devront être vérifiés à la fin de la séance de travail :

```
//@ invariant zone != null;
//@ invariant ptr <= zone.length & ptr >= 0;
//@ invariant 0 < zone.length ;
//@ invariant ptr >= zone.length -10;
```

Méthode `estVide`

1. Dans la spécification de la méthode `estVide`, trouvez la bonne valeur pour les ??? . Une fois les points d'interrogation remplacés, retirez les premiers commentaires des 2 lignes.

```
// //@ ensures ptr <= 0 ==> ????
// //@ ensures ptr > 0 ==> ????
```

2. Validez cette méthode : aucune mise en garde relative à la méthode `estVide` ne doit être générée par `escjava` pour cette méthode.

Méthode `copiePartiel`

3. Donnez la spécification complète de la méthode `copiePartiel`, en précisant notamment le contenu de la table **`t2`**. N'oubliez pas les invariants de boucle.

4. Validez cette méthode : aucune mise en garde relative à la méthode `copiePartiel` ne doit être générée par `escjava.`

Méthode `modificationTailleZone`

5. Donnez la spécification complète de la méthode `modificationTailleZone`, en précisant notamment le contenu de la table **`zone`**, ainsi que la valeur de sa taille.
6. Validez votre code : aucune mise en garde relative à la méthode `modificationTailleZone` ne doit être générée par `escjava.`

Méthode `empiler`

7. Donnez la spécification de `empiler`, en précisant notamment le contenu de la table **`zone`**, et la nouvelle valeur de **`ptr`**. La spécification doit être cohérente avec les invariants à vérifier.
8. Codez la méthode `empiler`. Le code doit être conforme à la spécification. (ESC/Java doit le vérifier!) et il doit compiler sans erreur avec la commande `javac Pile.java.`
   Vous utiliserez la méthode privée `modificationTailleZone.`

Méthode `depiler`

9. Donnez la spécification de `depiler`, en précisant notamment le contenu de la table **`zone`**, et la nouvelle valeur de `ptr`. La spécification doit être cohérente avec les invariants à vérifier.
10. Réécrivez le code de la méthode `depiler`. Le code doit être conforme à la spécification. (ESC/Java doit le vérifier!) et il doit compiler sans erreur avec la commande `javac Pile.java.`
    Vous utiliserez la méthode privée `modificationTailleZone.`

Invariant - suite -

11. Dé-commentez l'invariant en commentaire.
12. Validez votre code : aucune mise en garde ne doit être générée par `escjava`.

Méthode `Main`

Trouvez la bonne valeur pour les ??? . Une fois les points d'interrogation remplacés, retirez les premiers commentaires des lignes : `// /*@ loop_invariant ???;`
et des lignes `// //@ assert p.ptr == ???? ;`

13. Validez votre code : aucune mise en garde ne doit être générée par `escjava`.

```
public class Pile {
   private /*@ spec_public */ int[] zone;
   private /*@ spec_public */ int  ptr;

     //@ invariant zone != null;
     //@ invariant ptr <= zone.length & invariant ptr >= 0;
    //@ invariant 0 < zone.length ;
    // //@ invariant ptr >= zone.length -10;

   //@ ensures zone != null & ptr == 0;
    public Pile(){ zone = new int [5]; ptr = 0; }

  // //@ ensures ptr <= 0 ==> ????
  // //@ ensures ptr > 0 ==> ????
   public boolean estVide () {
      if (ptr <= 0) return true; return false; }

    private static void copiePartiel(int[] t1, int[] t2, int longueur) {
      int i = 0;
      while( i < longueur){ t2[i] = t1[i]; i++;}
    }


    private  void modificationTailleZone(int taille) {
      int[] zoneBis = new int[taille];
      if (zoneBis.length >= zone.length)
            copiePartiel(zone, zoneBis, zone.length);
      else
            copiePartiel(zone, zoneBis, zoneBis.length);
      zone = zoneBis;
     }


    public void empiler(int elt) {}

    // A REECRIRE
    public int depiler(){ return 0; }

    public static void main(String [] a){
      Pile p = new Pile();
      int res;
      int i = 0, j = 0;
      int B1 = 20, B2 = 15;
      /* /*@ loop_invariant i <= B1;
            loop_invariant i>=0;
            loop_invariant p.ptr == ????;
            loop_invariant (\forall int l; l >=0 & l < i ==> p.zone[l] == ????);
        */
      while (i < B1)  { p.empiler(i); i++; }
      /* /*@ assert p.ptr ==  ???? ;
            loop_invariant j<= B2;
            loop_invariant j>=0;
            loop_invariant p.ptr == ????;
            loop_invariant (\forall int l; l >=0 & l < p.ptr ==> p.zone[l] == ??);
         */
      while( j < B2) { res = p.depiler(); j++; }
      // //@ assert p.ptr == ???? ;
    }
}
```