TP3 : "Pile"

En Informatique, une **pile** (en anglais stack) est une structure de données fondée sur le principe « dernier arrivé, premier sorti » (ou LIFO pour *Last In, First Out*), ce qui veut dire que les derniers éléments ajoutés à la pile seront les premiers à être récupérés. Le fonctionnement est similaire à celui d'une pile d'assiettes : on ajoute des assiettes sur la pile, et on les récupère dans l'ordre inverse, en commençant par la dernière ajoutée (extrait de la définition de **pile** de Wikipédia).

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\A`

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

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.

Expliquez les directives de ESC/Java suivantes :

```
spec_public
modifies
\old
\result
```

Invariant

1. Ajoutez des invariants bornant la valeur de `ptr` (il faut une borne inférieure aux valeurs de `ptr` et une borne supérieure aux valeurs de `ptr`).

Méthode `empiler`

1. Codez la méthode `empiler`. Le code de la méthode doit être conforme à la spécification. (ESC/Java doit le vérifier!) et il doit compiler sans erreur avec `javac Pile`.

Méthode `depiler`

2. Donnez la spécification de `depiler`, en précisant notamment les valeurs modifiées et la nouvelle valeur de `ptr`.

3. Validez votre méthode : aucune mise en garde ne doit être générée par `escjava`.

Méthode `Main`

4. Trouvez la bonne valeur pour les ??? . Une fois les points d'interrogation remplacés, retirez les premiers commentaires des lignes : `// //@ assert res == ???;`
5. Validez votre code (aucune mise en garde ne doit être générée).

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

    //@ invariant zone != null;

    //@ requires taille > 0;
    //@ ensures zone != null & ptr == 0 && zone.length == taille;
    public Pile(int taille){
           zone = new int[taille];
           ptr = 0;
    }

    /*@ requires ptr < zone.length;
        modifies ptr, zone[ptr-1];
        ensures \old(ptr) + 1 == ptr && zone[\old(ptr)] == elt;
    */
    public void empiler(int elt){ }

    public int depiler(){ ptr--; return zone[ptr]; }

    public static void main(String[]a){
           Pile p = new Pile(8);
           // //@ assert p.zone.length == ????;
           p.empiler(1); p.empiler(5);
           // //@ assert p.zone[0] == ????;
           // //@ assert p.zone[1] == ????;
           // //@ assert p.ptr == ????;
           int res = p.depiler();
           // //@ assert res == ????;
           // //@ assert p.zone[0] == ????;
           // //@ assert p.ptr == ????;
           p.empiler(3);
           // //@ assert p.zone[0] == ????;
           // //@ assert p.zone[1] == ????;
           // //@ assert p.ptr == ????;
           p.empiler(9);
           // //@ assert p.zone[0] == ????;
           // //@ assert p.zone[1] == ????;
           // //@ assert p.zone[2] == ????;
           // // @ assert p.ptr == ????;
           res = p.depiler();
           // //@ assert p.zone[0] == ????;
           // //@ assert p.zone[1] == ????;
           // @ assert p.ptr == ????;
           // //@ assert res == ????;
           p.empiler(7);
    }
}
```