TP3

Lectures préalables :	Thèmes du TP :

• tutorial
  ○ Getting Started
  ○ Learning the Java Language
	• Une Pile d'entiers
	• Plusieurs classes
	• Tableaux
	• Variables d'instance

- **Visualisez le sujet en ouvrant `index.html` du répertoire qui a été créé à l'ouverture de `tp3.jar` par BlueJ; vous aurez ainsi accès aux applettes et pourrez expérimenter les comportements qui sont attendus.**
- **Soumettez chaque question à l'outil d'évaluation junit3.**

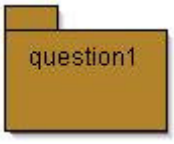
.1) Une pile d'"Object"s

Les éléments de la classe Pile sont maintenant des instances de la classe `java.lang.Object` (*classe racine de toute classe Java).*

Donc private Object[] zone; remplace private int[] zone;

Modifiez le code de la classe pile de la question 2 du TP précédent *(avec ou sans la désactivation des boutons)* pour obtenir une nouvelle version de la classe Pile et des IHM (AppletPile et AppletPile2)

un exemple de la "nouvelle" ApplettePile *(Essayez d'empiler des nombres, des mots, ...)*



.2) Développez à nouveau une classe "UneUtilisation", vérifiez l'affichage produit

```
package question1;
```

```
public class UneUtilisation{

  public static void main(String[] args) throws Exception{
    Pile p1 = new Pile(6);
    Pile p2 = new Pile(10);
    // p1 est ici une pile de polygones réguliers PolygoneRegulier.java
    p1.empiler(new PolygoneRegulier(4,100));
    p1.empiler(new PolygoneRegulier(5,100));
    p1.empiler("polygone");
    p1.empiler(new Integer(100));
    System.out.println(" la pile p1 = " + p1); // Quel est le résultat ?

    p2.empiler( new Integer(1000));
    p1.empiler(p2);
    System.out.println(" la p1 = " + p1); // Quel est le résultat ?

    try{
      p1.empiler(new Boolean(true));      // Quel est le résultat ?
      // ....
      String s = (String)p1.dépiler();
    }catch(Exception e ){
      e.printStackTrace();
    }


  }
}
```

Soumettez cette première question à l'outil d'évaluation pour ne pas laisser passer de problèmes avant d'aborder la question suivante.

.1) Spécification : Interface

Soit l'interface spécifiant le comportement d'une pile (fichier PileI) :

interface PileI

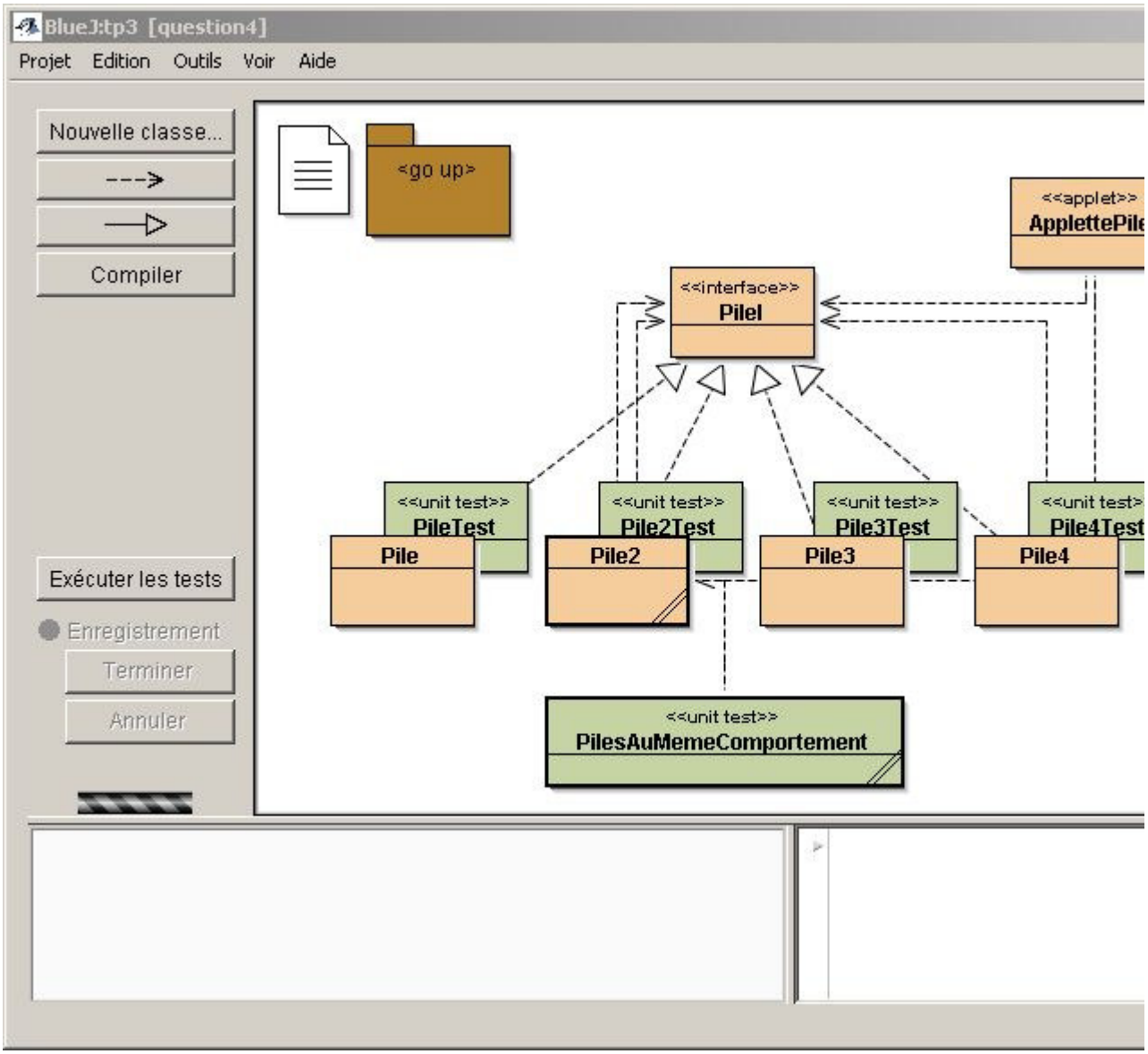
```
package question2;
import question1.PilePleineException;
import question1.PileVideException;

public interface PileI{

  public final static int CAPACITE_PAR_DEFAUT = 6;

  public void empiler(Object o) throws PilePleineException;
  public Object dépiler() throws PileVideException;

  public Object sommet() throws PileVideException;
  public int capacité(); // nb de cases
  public int taille();   // nb d'éléments présents
  public boolean estVide();
  public boolean estPleine();
  public boolean equals(Object o);
  public int hashCode();
  public Object clone() throws CloneNotSupportedException;
  public String toString();
}
```

Modifiez la classe Pile pour qu'elle implémente maintenant l'**`interface PileI`** :

Attention *: 6 nouvelles méthodes sont spécifiées dans PileI (capacité, taille, sommet, equals, clone, et hashCode), et donc toute implémentation de cette interface doit implanter aussi ces 6 méthodes.*
Deux d'entre-elles vous sont fournies, mais cette note intitulée " How to avoid traps and correctly override methods from java.lang.Object " peut vous être utile pour comprendre.

---

question2 .2) Implémenter une interface : PLUSIEURS implantations UN SEUL comportement



Proposez**, 3 autres** implémentations des Piles (donc implémentant l'interface **`PileI`**), *attention, pour l'utilisateur* ***ici les piles doivent toujours être bornées***

- Pile2 : utilise la classe prédéfinie **`java.util.Stack<Object>`** (**Pile2**.java),
- Pile3 : utilise la classe prédéfinie **`java.util.Vector<Object>`** (**Pile3**.java)

- Pile4 : non utilisée dans ce tp

Remarques :

- Pile2 et Pile3 seront composées d'une instance d'une classe prédéfinie comme le suggère les extraits de code suivants :

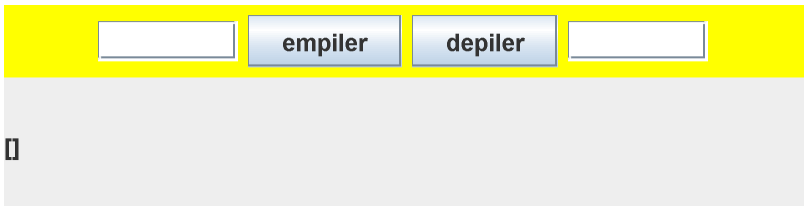
```
import java.util.Stack;
public class Pile2 implements PileI,Cloneable{
 private Stack<Object> stk; // La classe Pile2 est implémentée
  ...
}
```

```
import java.util.Vector;
public class Pile3 implements PileI,Cloneable{

private Vector<Object> v;

...

}
```

> Cette approche est nommée par Mark Grand le "Pattern delegation". Elle consiste à définir une donnée d'instance d'une classe, ensuite utilisée dans les méthodes.
> Ce pattern réalise une interface entre le client de la classe et l'implémentation effective par une classe de service. Il permet un couplage moins fort entre les deux classes que si l'une héritait de l'autre.

---

ci dessous "ApplettePile" avec une implémentation de la classe Pile par délégation à la classe java.util.Stack<Object>



.3) Proposez une classe de tests unitaires à chaque implémentation,

---

.4) Complétez la classe de Tests Unitaires nommée "PilesAuMemeComportement" , qui "vérifie" que toutes les piles ont bien le même comportement.

Cette question supplémentaire n'a pas à être soumise à évaluation par junit3 ni à être "rendue".

question3-1) *Généricité : un premier usage*

L'interface pileI est désormais paramétrée par le type des éléments

```
package question3;

import question1.PilePleineException;
import question1.PileVideException;

public interface PileI<T>{

  public final static int CAPACITE_PAR_DEFAUT = 6;

  public void empiler(T o) throws PilePleineException;
  public T dépiler() throws PileVideException;

  public T sommet() throws PileVideException;

.../...
} // PileI
```

question3-2) vérifiez le source de la classe Pile2<T> et **modifier** l'IHM AppletePile, ce type de message obtenu à la compilation **ne doit plus apparaître**

question3-3) Dans la méthode main de la classe UneUtilisation, proposez **toutes** les déclarations correctes afin que la compilation de cette classe s'effectue **sans aucun message d'erreur ou alertes** (sauf pour la méthode `clone`).

i.e. proposez les déclarations telles que p1 contient des éléments "de type PolygoneRegulier", p2 que des Piles de Polygone régulier , etc ...

```
import question1.PolygoneRegulier;

public class UneUtilisation{

  public static void main(String[] args) throws Exception{
    PileI ... p1 = new Pile2 ... (6);
    PileI ... p2 = new Pile2 ... (10);
    etc ...
```

Vérifiez ensuite que ces lignes extraites de la question 1 ne se compilent plus !

```
     try{
      p1.empiler(new Boolean(true));

      String s = (String)p1.dépiler();
    }catch(Exception e ){
      e.printStackTrace();
    }
```
