TP3

Lectures préalables :	Thèmes du TP :
Java_II tutorial Interfaces	la classe Object **Vector<T>** **Stack<T>**

Une pile d'Objects

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

question1-1 Proposer en conséquence une nouvelle version de la classe Pile et de l'IHM demandées au TP précédent question3.ApplettePile ou ApplettePile2

un exemple de la "nouvelle" ApplettePile

question1-2) Soit cet extrait d'une classe d'utilisation d'une Pile (classe UneUtilisation), vérifier l'affichage produit

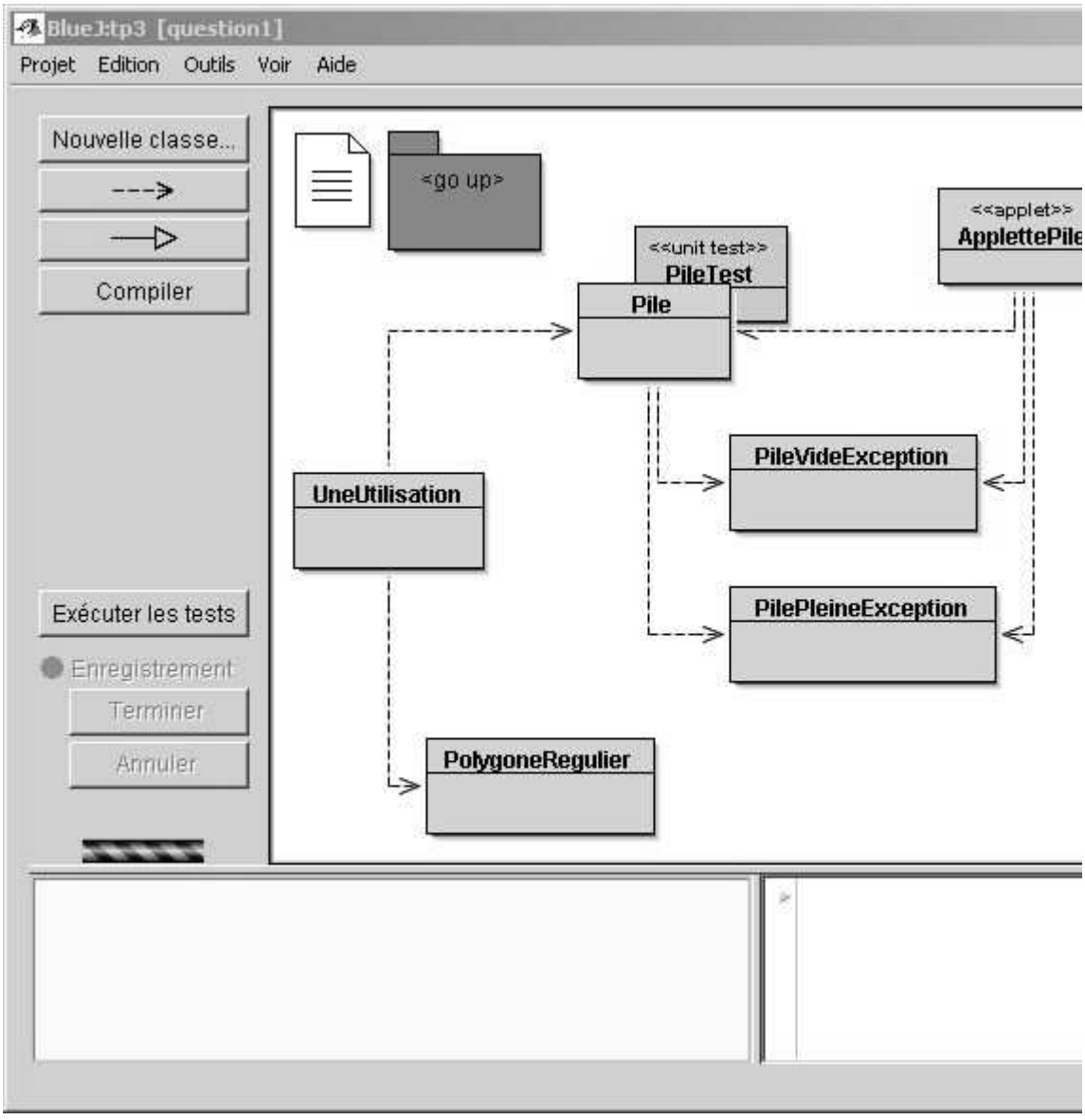
```
package question1;

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));
      // ...                          // erreur de compilation ? pourquoi ?
```

```
      String s = (String)p1.dépiler(); // erreur d'execution ? pourquoi ?
    }catch(Exception e ){
      e.printStackTrace();
    }


  }
}
```

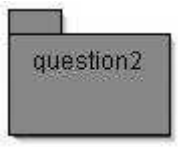
question1-3) Proposez une classe conséquente de tests unitaires de la classe Pile, lire ce tutoriel

avec bluej, menu Outils, item Préférences, onglet divers cochez "Montrer les outils de test" , puis clic droit classe Pile, item Créer classe de test

Vous devriez obtenir cette classe en arrière plan de la classe Pile comme suit :



ensuite, en travail personnel et à l'aide du tutoriel, enregistrez plusieurs méthodes de test, et vérifiez le bon fonctionnement.



La classe Pile implémente l'**`interface PileI`**, modifier la classe Pile en conséquence

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

public interface PileI{

  public final static int TAILLE_PAR_DEFAUT = 10;

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

  public Object sommet() throws PileVideException;
  public int capacité();
  public int taille();
  public boolean estVide();
  public boolean estPleine();
  public String toString();
}
```

---

question2-1) Proposez **deux autres** implémentations de l'interface **`PileI`**,

1. *utiliser les classes prédéfinies* ***`java.util.Stack<T>`*** *(**Pile2**.java), et* ***`java.util.Vector<T>`*** *(**Pile3**.java)*
1. *pour l'utilisateur* ***ce sont toujours des piles bornées***
1. *Vos deux nouvelles classes "Pile" seront composées d'une instance d'une classe prédéfinie comme le suggère les extraits de code suivants :*

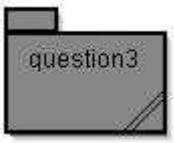
```
import java.util.Stack;
public class Pile2 implements PileI{
 private Stack<Object> stk; // La classe Pile2 est implémentée p
 ...
}
```

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

private Vector<Object> v;

...

}
```

Remarque

1. 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 l' ApplettePile avec une implémentation de la classe Pile par délégation à la classe java.util.Stack<Object>

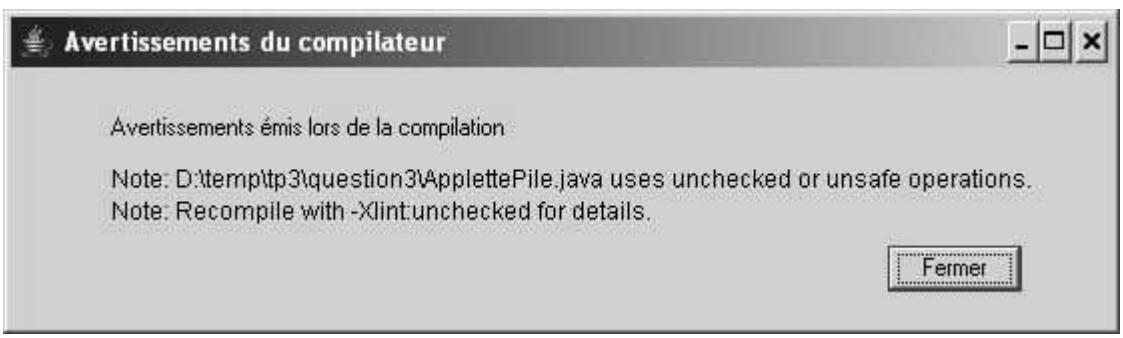


Généricité : un premier usage

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 TAILLE_PAR_DEFAUT = 10;

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

  public T sommet() throws PileVideException;
  public int capacité();
  public int taille();
  public boolean estVide();
  public boolean estPleine();
  public String toString();

}
```

question3-2) vérifiez le source de la classe Pile2<T> et **modifier** l'IHM ApplettePile, 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**.

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

Peut-on/doit-on imposer une implémentation particulière pour les piles contenues dans p2 ?

```
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();
    }
```

N'oubliez pas de faire un submit avant de quitter la salle !