# Contrats sur les classes : Assertions et exceptions

## Plan du cours

- Le type abstrait *Pile* et une structure de données qui le met en œuvre,
- Les assertions qui servent à la mise au point du type abstrait.
- Les exceptions qui servent à sécuriser son usage.

# Le TDA Pile

- le type défini est *Pile*
- il est paramétré par un autre type $T$

Opérations

- initialiser()
- empiler(T element)
- depiler() : T
- sommet() : T
- 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 $p.depiler() = t$

# Interface IPile

```
public interface IPile<T>
{
    void initialiser();
    void empiler(T t);
    T depiler();
    T sommet();
    boolean estVide();
    int nbElements();
}
```

# Classe Pile

```
public class Pile<T> implements IPile<T>{
   private ArrayList<T> elements;

   public Pile(){initialiser();}

   public T depiler() {
     T sommet = elements.get(elements.size()-1);
     elements.remove(sommet);
     return sommet;
   }

   public void empiler(T t) {elements.add(t);}

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

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

   public T sommet() {return elements.get(elements.size()-1);}
   // ....
```

# Classe Pile

```
public class Pile<T> implements IPile<T>{

   // (....)

   public String toString(){return "Pile = "+ elements;}

   public int nbElements(){return elements.size();}
}
```

# Main

```
public static void main(String[] a)
{
   Pile<String> p = new Pile<String>();
   System.out.println(p);
   p.empiler("a"); p.empiler("b"); p.empiler("c");
   System.out.println(p);
   p.depiler();
   System.out.println(p);
   p.depiler(); p.depiler();
   System.out.println(p);
 }
```

# Assertions

- Les assertions permettent de vérifier des propriétés sur une classe.
- Le programme est interrompu en cas de non respect de ces propriétés.
- Elles peuvent être activées ou inhibées et sont principalement une aide au débogage et vont servir à vérifier la plupart des axiomes (post-conditions).

# Assertions : activation sous eclipse

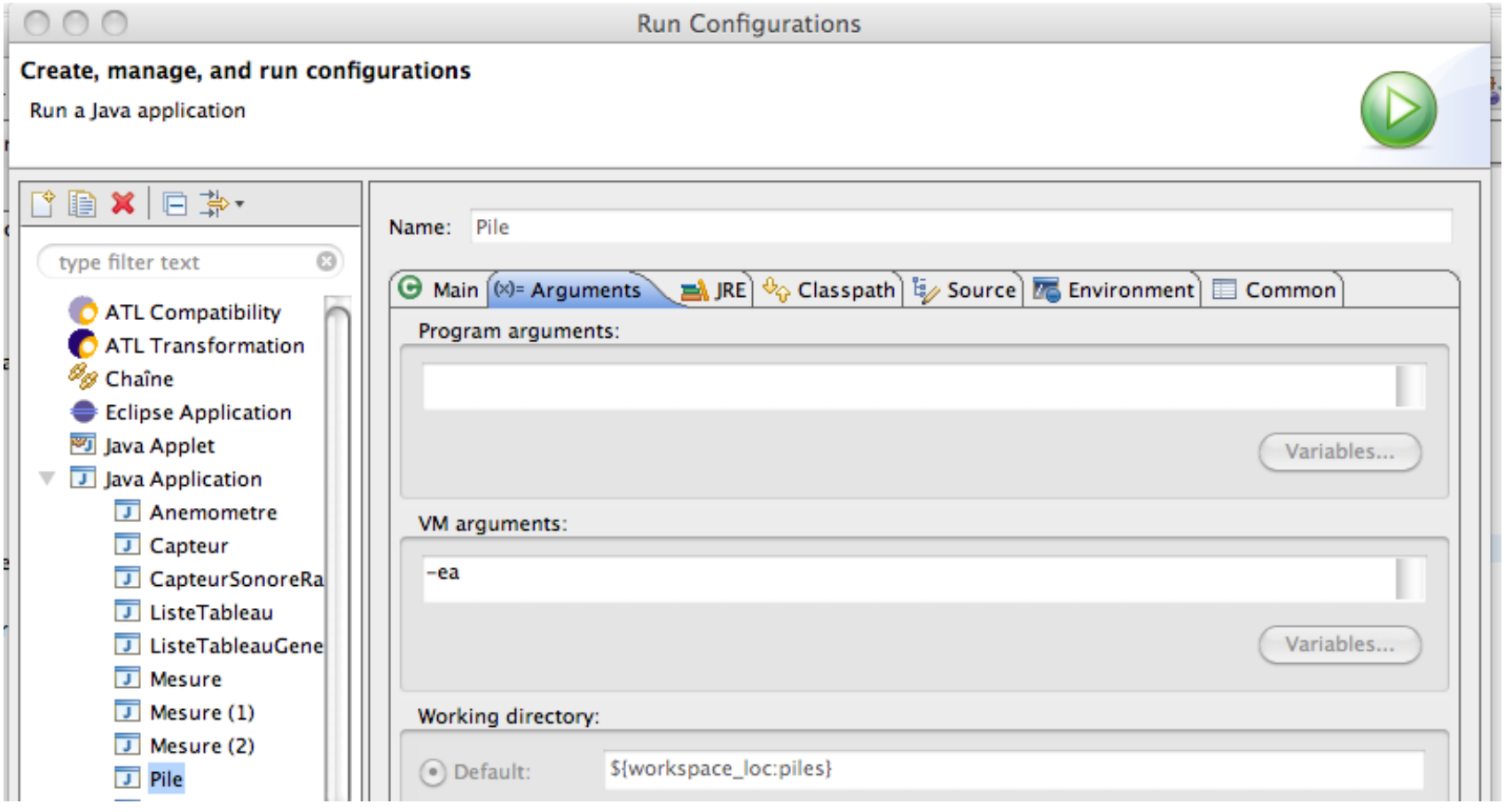


FIGURE – Configuration sous eclipse pour activer les assertions

# Assertions : activation en ligne de commande (pour repl.it)

- -ea (enable assertions)
- -da (disable assertions)
- Par exemple `java -ea maclasse`

# Assertions : 2 syntaxes

```
assert condition ;
assert condition : objet ;
```

# Assertion : empiler

```
    public void empiler(T t) {
    elements.add(t);
    assert this.sommet()==t;
}
```

# Assertion : empiler erronée

```
public void empiler(T t) {
   elements.add(t); elements.add(null);
   assert this.sommet()==t ;
}
```

L'exécution du *main* précédent s'arrête et on obtient le message :

```
Exception in thread "main" java.lang.AssertionError
at pile.Pile.empiler(Pile.java:20)
at pile.Pile.main(Pile.java:48)
```

# Assertion : affichage d'un message

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

Cette fois l'erreur affichée sera la suivante :

```
Exception in thread "main" java.lang.AssertionError:
dernier empile =null
at pile.Pile.empiler(Pile.java:20)
at pile.Pile.main(Pile.java:48)
```

# Exceptions

- signaler mais aussi rattraper des erreurs pouvant se produire pendant l'exécution du programme
- éviter des arrêts brutaux du programme
- revenir dans un fonctionnement normal
- stopper le programme proprement, après avoir éventuellement sauvegardé des données, fermé des fichiers, etc.

# Exceptions

Les exceptions sont des objets qui représentent une erreur à l'exécution :

- une méthode appelée hors de son domaine de définition, par exemple dépiler une pile vide, ou donner une mauvaise valeur de paramètre, cela correspond souvent aux préconditions des opérations des types abstraits,
- une erreur de saisie (comme saisir "aa" quand on attend un entier, ou une date mal formée),
- un problème matériel (plus de place lors de l'écriture d'un fichier, ce qui empêchera le respect d'une post-condition du fait de l'environnement d'exécution),
- certains événements de l'exécution qui vont se produire car le programme est mal prouvé ou mal testé (sortir des bornes d'un tableau, une division par zéro).

# Définition d'une classe représentant une exception

Exemple : la pile est vide quand on veut dépiler ou consulter le sommet.

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

# Déclaration et signalement

```
public interface IPile<T>
{
....
   T depiler() throws PileVideException;
...
}

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

# Programme présentant une erreur

```
public static void main(String[] a)
{
    Pile<String> p = new Pile<String>();
    System.out.println(p);
    p.empiler("a"); p.empiler("b"); p.empiler("c");
    System.out.println(p);
    p.depiler();
    System.out.println(p);
    p.depiler(); p.depiler();  p.depiler();
    System.out.println(p);
}
```

# Rattraper une erreur : Bloc Try/catch

```
try{BLOC-0}
catch (Class-1 e1){BLOC-1}
...
catch(Class-n en){BLOC-n}
finally{BLOC-n+1}
```

D'autres versions plus complexes existent (voir documentation Java)

# Rattraper une erreur : Bloc Try/catch

```
public static void main(String[] a)
{
     try{
         Pile<String> p = new Pile<String>();
         System.out.println(p);
         p.empiler("a"); p.empiler("b"); p.empiler("c");
         System.out.println(p);
         p.depiler();
         System.out.println(p);
         p.depiler(); p.depiler();  p.depiler();
         System.out.println(p);
    }
    catch (PileVideException p){System.out.println(p.getMessage());}
    System.out.println("Fin de la méthode main");
}
```

# Exemple plus complexe

```
public class E1 extends Exception {}
public class E2 extends Exception {}
public class E3 extends E2 {}
public class E4 extends Exception {}
public class TestException {

    public void meth1() throws E1, E2, E3, E4
    {  ..........}

    public void meth2() throws E4
    {
        try{
            System.out.print("(1)");
            meth1();
        }
        catch(E1 e){System.out.print("(2)");}
        catch(E2 e){System.out.print("(3)");}
        finally {System.out.print("(4)");}
        System.out.println("(5)");
    }
```

# Résultat

- s'il n'y a rien de signalé dans meth1 : (1)(4)(5)
- avec throw new E1() ; dans meth1 : (1)(2)(4)(5)
- avec throw new E2() ; dans meth1 : (1)(3)(4)(5)
- avec throw new E3() ; dans meth1 : (1)(3)(4)(5)
- avec throw new E4() ; dans meth1 : (1)(4)Exception in thread "main"
  exerciceCours2011.E4
  at exerciceCours2011.TestException.meth1(TestException.java :7)
  at exerciceCours2011.TestException.meth2(TestException.java :14)
  at exerciceCours2011.TestException.main(TestException.java :25)

# Exceptions en Java

- Throwable
  - Exception
    - *RunTimeException* (*NullPointerException*, *ArrayIndexOutOfBoundsException*, *ClassCastException*, etc.). On peut chercher à les capturer pour sécuriser le programme ; exceptions non vérifiées (non déclarées dans les signatures)
    - ... toutes les autres (comme PileVideException)
  - Error correspondent à des erreurs de la machine virtuelle sur lesquelles nous ne travaillerons pas dans le cadre de ce cours.

# Recommandations : Eviter de ...

- créer des effets de bord, c'est-à-dire de modifier l'état du programme (par exemple ne pas empiler ou dépiler dans une assertion).

# Recommandations : Assertions pour ...

- invariant d'algorithme, par ex. à la fin de l'itération i dans la recherche du minimum la variable min contient la plus petite valeur rencontrée entre 0 et i
- la plupart des postconditions des opérations et des axiomes des types abstraits de données, par ex. à la fin d'une fonction de tri le tableau est trié
- invariants de classe, par ex. une voiture a 4 roues

# Recommandations : Exceptions pour ...

- vérifier les paramètres d'une méthode car ces paramètres viennent d'ailleurs (de l'extérieur de la méthode) et peuvent vraisemblablement être faux (donc il vaut mieux ne pas interrompre brutalement l'exécution, et plutôt traiter le problème, avec des exceptions notamment),
- la plupart des préconditions des opérations,
- vérifier les résultats d'interactions avec un utilisateur (elles vont souvent comporter des erreurs, qu'il faut traiter),