Programmation JAVA
Les Assertions

Syntaxe

assert expression;

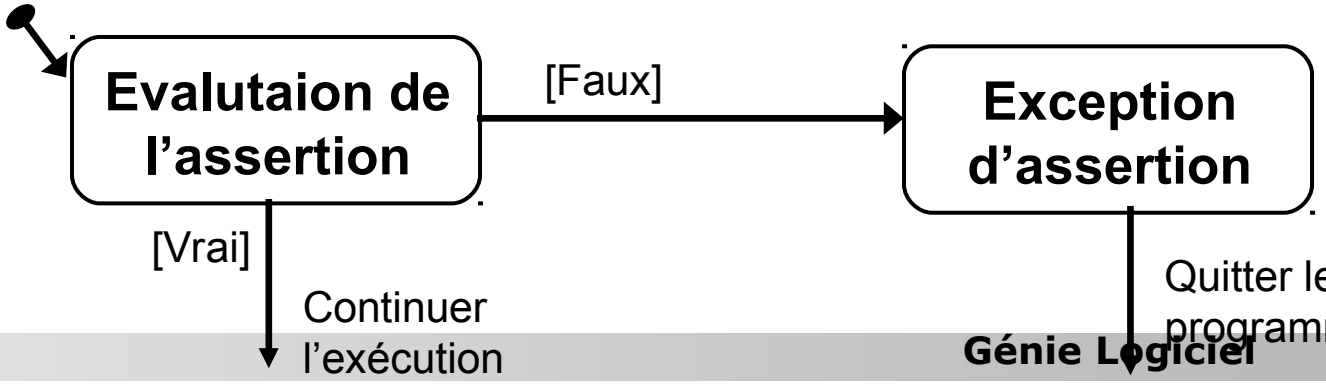
expression est une expression booléenne

assert $expression_1$: $expression_2$

$expression_1$: expression booléenne

$expression_2$: expression retournant une valeur. Ne peut être une méthode de type void

Exemple : assert x>y : "x : " + x + ", y : " +y ;



Compilation run-time

Les assertions peuvent être activées ou désactivées à la demande du programmeur.

Avantages

- Eviter les traitements lourds d'assertions coûteuses.
- Activer cette option en cas de doute.

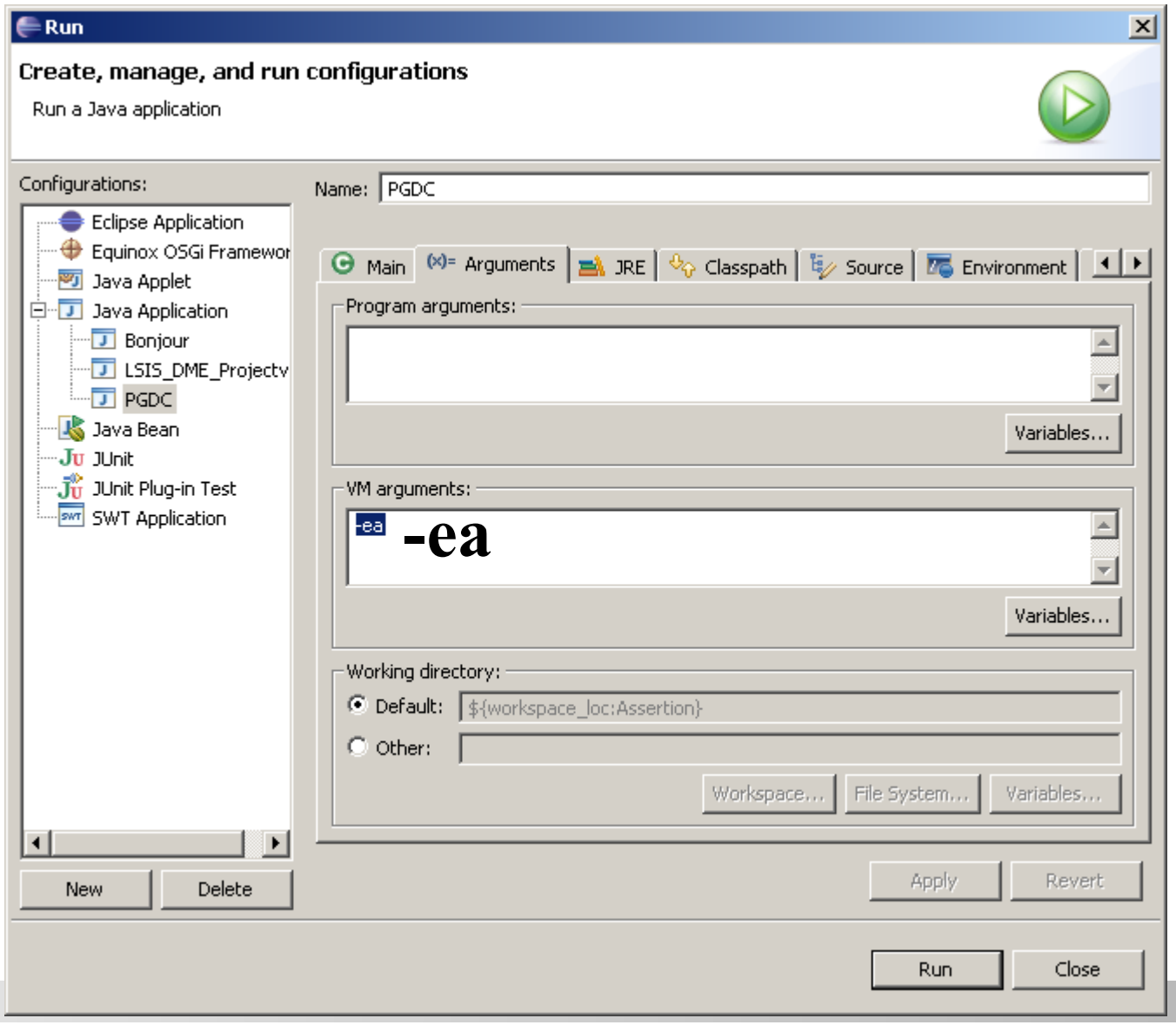
Autoriser et interdire les Assertions

Option	Description
-ea -da	Pour toutes les classes non-système
-ea<package>… -da<package>…	Pour toutes les classes non-système du package et ses sous-packages
-ea… -da…	Pour toutes les classes non-système du package par défaut
-ea <classe> -da <classe>	Pour la classe en question
-esa -dsa	Pour les classes système

Exemple :

java -ea Test
java -ea -da Bonjour Test

Comment ça marche sous Eclipse ?



Faire appel aux assertions dans le code

3 cas d'utilisation :

1- les invariants

2- les invariants pour le contrôle de flux

3- Pré & Post-conditions et invariants de classe

3 cas à éviter :

1- Ne pas utiliser les assertions pour vérifier les arguments d'une méthode **publique**

2- Ne pas remplacer les instructions conditionnelles par des assertions

3- les assertions ne doivent pas apportées des modifications sur les variables utilisées par le programme

Cas particulier : sauf celles utilisables par d'autres assertions

Assertion modifiant une variable

```
// action dans une assertion
assert names.remove(null);

// ré-écriture de l'instruction
booleen nullsRemoved =
  names.remove(null);
assert nullsremoved;
```

Sémantique

Les assertions doivent être sans effet sur les variables du programme. L'évaluation d'une assertion ne doit pas avoir de modification.

Une exception, lorsque l'assertion modifie certaines variables utilisées par d'autres assertions

Invariant interne (d'étape)

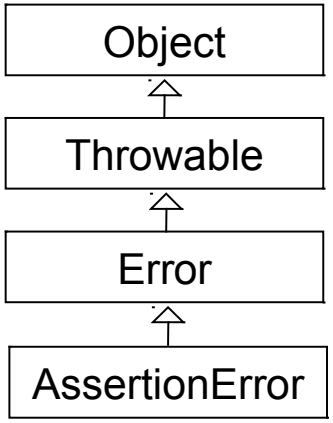
```
if (i%3==0) {
   ...
} else if (i%3==1) {
   ...
} else {
   // nous déduisons
   //i%3==2
   ...
}
```

```
if (i%3==0) {
   ...
} else if (i%3==1) {
   ...
} else {
   assert i%3==2;
   ...
}
```

Pb: cas où i<0

Invariant pour le contrôle de flux

```
void hasard(int piece) {
  switch (piece)
     case PILE : ...
                 break;
     case FACE : ...
                 break;
     case default : assert false;
     //← vérifier l'atteignabilité de ce
  // code
  ...
}// assert false ⇔ new AssertionError()
```



Préconditions

1er cas : traiter les préconditions dans le corps de la méthode

```
public float inverse (float x) {
// forcer le test de la précondition x !=0
   if (x==0)
    throw new
IllegalArgumentException("Division
          par zéro");
     return (1/x);
}
```

2ème cas : relâcher les préconditions dans la méthode appelante

```
private float inverse (float x) {
    assert x!=0;
    return (1/x);
}
```

Postconditions

Testables à la sortie de méthode publiques et non-publiques

```
int abs(int x) {
    // effectuer les traitements

     .....
    // fin des traitements
    assert x>=0;
    return x;
}
```

Exemple : Spécification formelle → Code

fonction Recherche (X : dans Tableau_d_entier ; Clé: dans Entier) **retour** Entier ;

Pré: il existe i dans *X'PREMIER…X'DERNIER* tel que X(i) = Clé

Post : **X"**(Recherche (X, Clé)) = Clé et X**"** = X

Erreur: Recherche (X,Clé) = *X'DERNIER* + 1

```
public int recherche(final int [] X, final int cle) {
   int indexcle = 0;
   class Xcopie{
   private int[] tab;
   public Xcopie() {
            tab = X.clone();
   }
   boolean nonModifieX(){ return Arrays.equals(X, tab);}
   int indexCleX(){
            int index = 0;
       while ((index < (tab).length) && (tab[index] != cle))
   index++;
       return index;
      }
   }
   Xcopie xcopie = new Xcopie();
   if (xcopie.indexCleX() == xcopie.tab.length)
   throw new IllegalArgumentException("Clé : " + cle + " est
   non présente dans X");
   // manipuler le tableau X d'entrée
   ....
   // fin de traitement de X
   assert  (xcopy.nonModifieX()) && (X[indexcle] == cle);
   return indexcle;
}
```

Invariant de classe

- Associé à toutes les instances de la classe
- Vérifiable pendant toute la durée de vie d'une instance.
- Pas de technique particulière pour la mise en œuvre

```
class TabOrd {
        private int [] tab;
        TabOrd(){
        tab = null;
        }
        TabOrd(int[] X){
        tab = X.clone();
        verifTab();
        }
        private void verifTab(){
        int i = 0;
        while(i++ < tab.length)
           assert tab[i] <= tab[i+1] :
         "tab[" + i + "]> tab[" +(i+1)+ "]";
        }
}
```

Questions/Réponses

1) Quelles sont les instructions déclenchant une erreur

(a) assert true : true; (b) assert true : false;

(c) assert false : true; (d) assert false : false;

2) Quelles des options sont valides

(a)-ae (b) -enableassertions (c) -disablesystemassertions

(d) -dea

3) Soit la méthode ci-dessous, quels appels déclencheront une erreur

```
static int inv(int value){
        assert value > -50 : value < 100;
        return 100/value;
}
```

(a) inv(-50); (b) inv(0); (c) inv(50); (d) inv(100); (e) inv(150);

Questions/Réponses

4) Quel est le résultat de cette exécution

```
public class T_Assertion{
  public static void assertBounds(int low, int high, int value){
       assert( value > low ? value < high : false)
              : (value < high ? ‘‘too low’’ : ‘‘too high’’);
  }
  public static void (String[] args){
       assertBounds(100, 200, 150);
  }
}
```