Java, les objets : tout de suite !
Test, assertions, « vérification statique »

Cnam Paris
jean-michel Douin, douin@cnam.fr
Version du 13 février 2003

Notes de cours associées au chapitre 6
Avec des assertions, utilisation de Jass et ESC/Java
tutorial BlueJ
http://www.bluej.org/doc/documentation.html

Ce support accompagne, référence le livre de David J. Barnes & Michael Kölling
Objects First with Java A Practical Introduction using BlueJ
Pearson Education, 2003 ISBN 0-13-044929-6.

Sommaire

- **Test**
  - **lesquels ?**
  - **tests unitaires avec Bluej**
- **Evite le Débogage ?**
  - **Complémentaire**
- **Règles de conception des tests, non régression**
  - **ou comment faciliter les évolutions du logiciel**

  Assertions en java depuis 1.4

  http://java.sun.com/j2se/1.4/docs/guide/lang/assert.html

  Les extras

  JASS : http://semantik.informatik.uni-oldenburg.de/~jass/

  ESC/Java : http://research.compaq.com/SRC/esc/

  une présentation : http://jfod.cnam.fr/tp_cdi/douin/ESC_Java.pdf

- Tests lesquels ?

- Black Box Testing
- White Box Testing
- Unit Testing
- Integration Testing
- System Testing
- Regression Testing
- Acceptance Testing
- Clean Room Testing

Black Box Testing

- S ’assurer que le logiciel respecte les spécifications annoncées.
- Ces spécifications sont souvent décrites comme des commentaires associés aux sources.
- Les tests ne portent pas sur les choix d ’implantation, ne concernent que les méthodes publiques

White Box Testing

- **La conception de tests d ’un logiciel en ayant la connaissance de son implémentation**

  Tests des cas limites en fonction des types de données utilisés

  Numérique : nombre min, zéro, nombre max

  Dépassement de capacité d ’une structure de données

  État d ’une instance

Unit Testing

- **Tests individuels d ’une classe**

  Consignés, mais comment ?

- **Facilités avec BlueJ**

  Fenêtre(s) Inspector,

  mais éphémère, pas de traces des différents tests (BlueJ 1.4 ?)

  Quelles situations critiques ?

  Quel enchaînement des méthodes ?

Integration et System Testing

- **Intégration**

  Tests individuels de plusieurs classes effectués séparément

- **System**

  Test du programme dans son environnement, au sein d ’un système d ’exploitation

Regression Testing

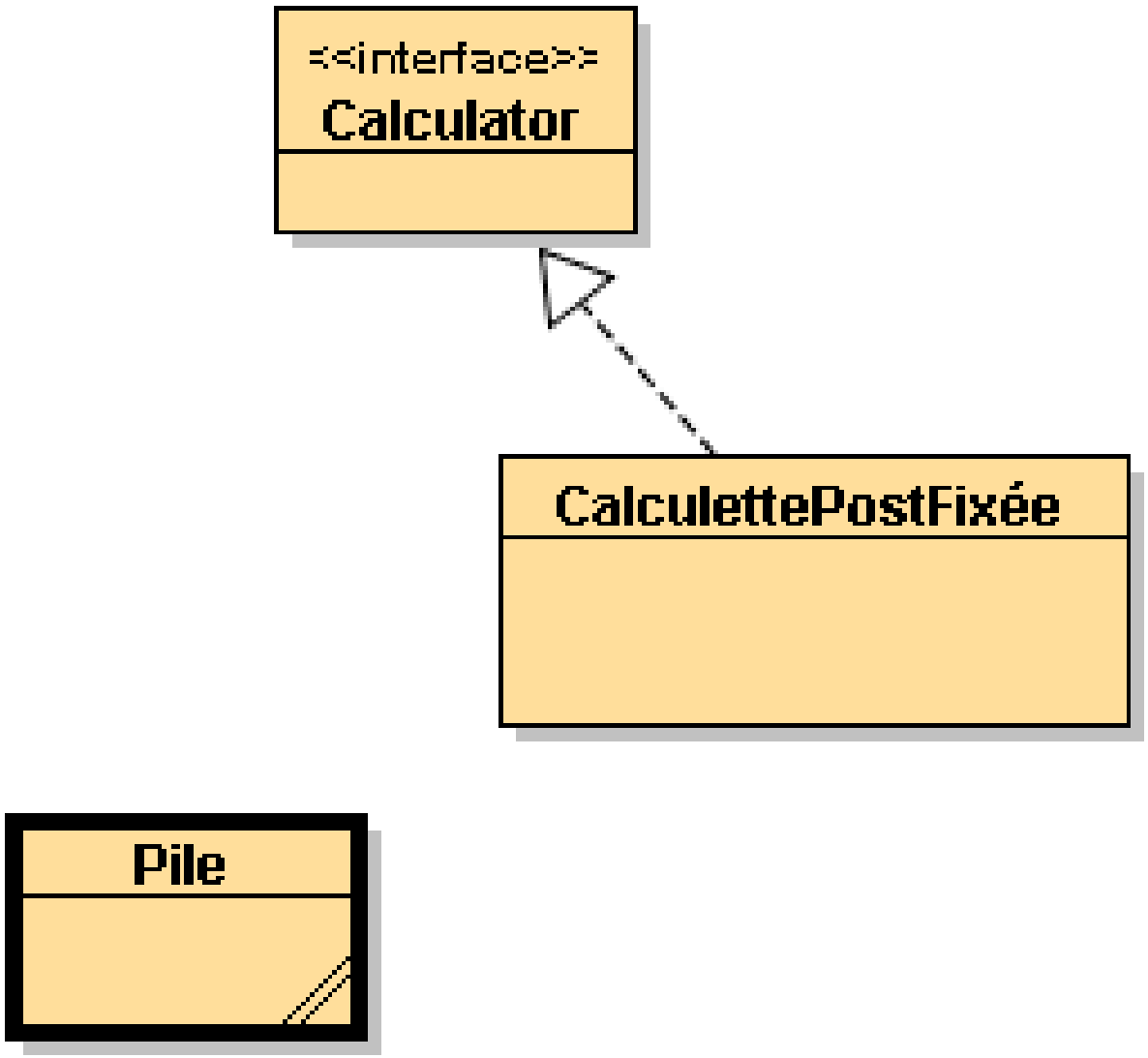
- Conserver les tests d ’un logiciel.
- Une modification du logiciel engendre de nouveau l ’application des tests et permet de vérifier l ’absence d ’une nouvelle erreur
- Chaque modification doit au moins respecter les tests de la version précédente
- Comment éviter qu ’un logiciel se dégrade ...

Acceptance et Clean Room Testing

- **Tests effectués par le client**

- **...**

Un exemple : une calculette à pile



L 'interface Calculator

```
/** Code 6.4 page 155 */
public interface Calculator{

  public int getDisplayValue();

  public void numberPressed(int number);

  public void plus();

  public void minus();

  public void equals();

  public void clear();

}
```

La calculette post fixée utilise une pile

- **3 + 2**

  devient

- **3 empiler(3)**
- **2 empiler(2)**
- **+ empiler(dépiler() + dépiler())**

- **Le résultat est au sommet de pile**

Une pile d ’entiers

- **La pile est bornée alors**
- Que se passe t-il lorsque l ’on tente d ’empiler sur une pile pleine ?
- **Et dans tous les cas**
- Que se passe t-il lorsque l ’on tente de dépiler une pile vide ?

Documentation utilisateur de la Pile

```
public class Pile{

/**  empile un entier au sommet de la pile
* si la est pile pleine, cette méthode n 'a aucun effet
public void empiler(int i);

/**  dépile un entier
* si la pile est vide, cette méthode retourne -1
*/
public int dépiler();
```

- */** cette méthode n 'a aucun effet ?,*
- */** cette méthode retourne -1,*

  si le dernier entier empilé était -1 ???

- **Une première solution les assertions**

Une première solution : les assertions

- **En attendant les exceptions**

- **usage de assert (1.4, BlueJ (Tools -> Preferences))**

assert ExpressionBooléenne;

assert ExpressionBooléenne : "Message à la console";

L 'expression booléenne doit être vraie, sinon le programme s 'arrête et le message est affiché

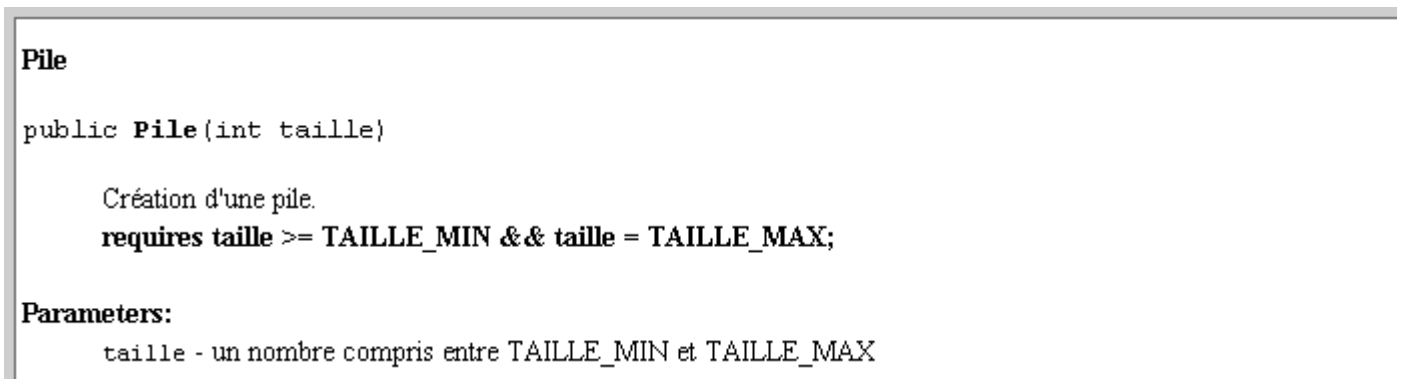
```
public int dépiler(){
  assert !EstVide():" dépiler sur une pile vide!!!  ";
  // le code ici
```

La classe Pile

```
public class Pile{
  public final static int TAILLE_MIN =1;
  public final static int TAILLE_MAX = 1000;
  private int[] zone;
  private int   ptr;


  /** <b> requires !estPleine(); </b>   */
  public void empiler(int i){
    assert !estPleine();
    zone[ptr] = i; ptr++;
  }

  /** <b> requires !estVide(); </b>   */
  public int dépiler(){
    assert !estVide();
    ptr--;
    return zone[ptr];
  }
```

La classe Pile le constructeur

```
/** Création d ’une pile.
 * <br><b> requires taille >= TAILLE_MIN && taille <= TAILLE_MAX;</b>
 * @param taille un nombre compris entre TAILLE_MIN et TAILLE_MAX
 */
  public Pile(int taille){
    assert taille >= TAILLE_MIN && taille <= TAILLE_MAX;
    ptr = 0;
    zone = new int[taille];
  }
```



Pile

`public` **`Pile`**`(int taille)`

Création d'une pile.
requires taille >= TAILLE_MIN && taille = TAILLE_MAX;

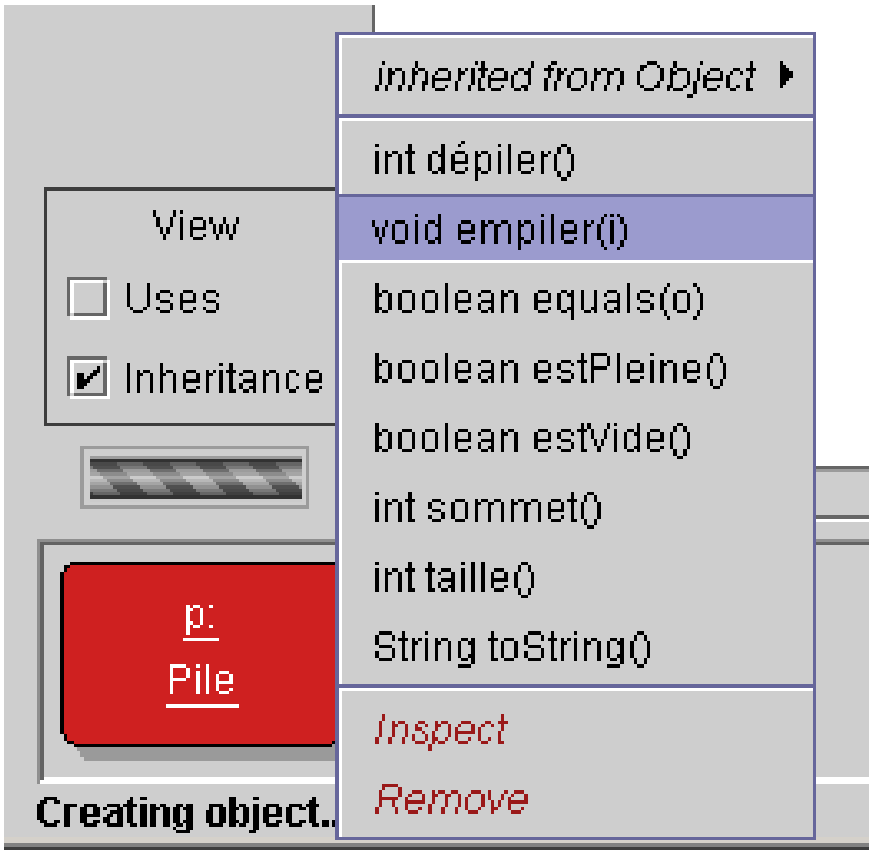
Parameters:

`taille` - un nombre compris entre TAILLE_MIN et TAILLE_MAX

La classe Pile, suite

```
public int sommet(){
  assert !estVide();
  return zone[ptr-1];
}

public boolean estVide(){return ptr==0; }


public boolean estPleine(){return ptr==zone.length;}

public boolean taille(){return ptr;}


public boolean equals(Object o){ ...


public String toString(){ ...
```

Test de la classe Pile avec BlueJ



Pile : WhiteBox Testing ?

- **1) Ajout de la méthode whiteBoxTesting à la classe Pile**
  selon ce schéma

```
private final static boolean TESTING_ON = true;

public final static boolean whiteBoxTesting(){
  if(TESTING_ON){ // compilation conditionnelle
    // ici les tests pertinents …

    return true;
  }else{
    return false;
  }
}
```

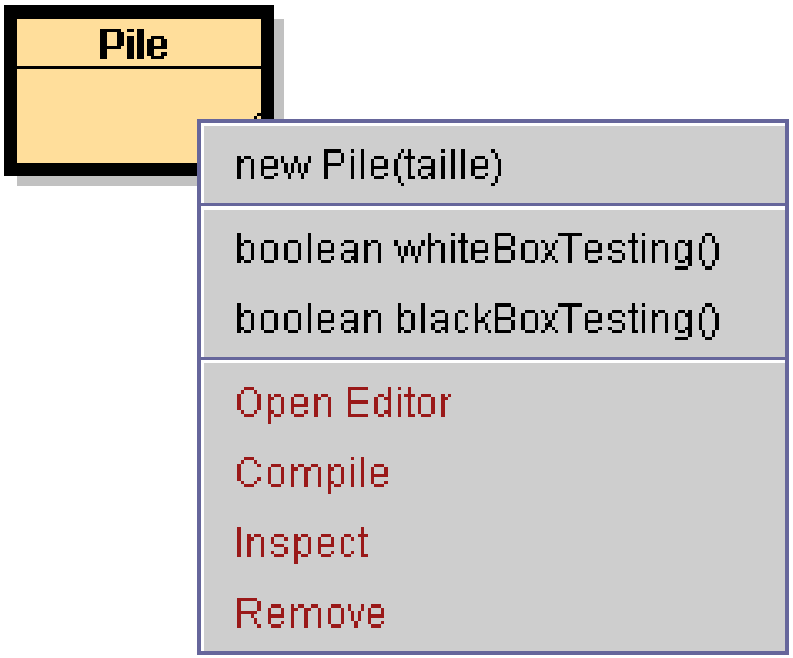
ce schéma pourquoi ?

si le booléen TESTING_ON est false, le code n 'est pas généré cette méthode retourne false (compilation conditionnelle en java (assert fonctionne de la même façon))

la méthode de classe whiteBoxTesting, assert

```
public static boolean whiteBoxTesting(){
  if(TESTING_ON){ // compilation conditionnelle
    Pile p  = new Pile(10); assert p.ptr==0;
                            assert p.zone!=null && p.zone.length==10;
    int indice = p.ptr;
    p.empiler(3);          assert indice+1 == p.ptr && p.zone[indice] == 3;
                           assert p.ptr == 1;
    indice = p.ptr;
    int res = p.dépiler();         assert indice-1 == p.ptr;
                                   assert res == p.zone[p.ptr];
      boolean état = p.estVide(); assert p.ptr == 0;
      while(!p.estPleine())
        p.empiler(res++);
                                   assert p.ptr==p.zone.length;

      return true;
    }else{
      return false;
    }
  }
```

whiteBoxTesting avec BlueJ



Traces d ’exécution avec BlueJ

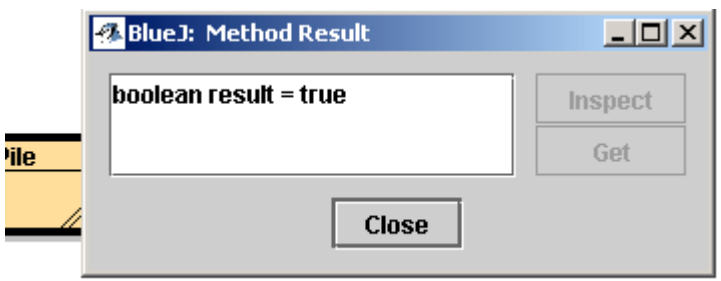
assertError : un exemple, l ’élément en p.zone[indice] est différent de 3

```
88
89  public static boolean whiteBoxTesting(){
90    if(TESTING_ON){ // compilation conditionnelle
91      Pile p  = new Pile(10);     assert p.ptr==0 && p.zone != null && p.zone.length == 10;
92      int indice = p.ptr;
93      p.empiler(3);               assert indice+1 == p.ptr && p.zone[indice] == 4 && p.taille() == 1;
94      indice = p.ptr;
95      int res = p.dépiler();      assert indice-1 == p.ptr && res == p.zone[p.ptr];
96      boolean état = p.estVide(); assert p.ptr == 0;
97      while(!p.estPleine())
```

```
AssertionError:
null
```

Avec éventuellement un Message

Le test est passé



whiteBoxTesting, &=, Code 6.3 page 154

```
public static boolean whiteBoxTesting(){// sans assert, avec &=
  if(TESTING_ON){
    boolean passed = true;
    Pile p  = new Pile(10); passed &= p.ptr==0;
                            passed &= p.zone!=null && p.zone.length==10;
    int indice = p.ptr;
    p.empiler(3);      passed &= indice+1 == p.ptr && p.zone[indice] == 3;
                       passed &= p.ptr == 1;
    indice = p.ptr;
    int res = p.dépiler();         passed &= indice-1 == p.ptr;
                                   passed &= res == p.zone[p.ptr];
      boolean état = p.estVide(); passed &= p.ptr == 0;
      while(!p.estPleine())
        p.empiler(res++);
                                   passed &= p.ptr==p.zone.length;

      return passed;
    }else{
      return false;
    }
  }
```

Discussion

- **&=**

ou

- **assert**

BlackBox Testing ?

Uniquement les méthodes publiques de la classe Pile

```
public static boolean blackBoxTesting(){
  if(TESTING_ON){
     Pile p  = new Pile(3);    assert p.estVide() && p.taille()==0;
    Pile p1 = new Pile(20);    assert p1.estVide() && p1.taille() == 0;
    Pile p4 = new Pile(Pile.TAILLE_MIN); assert p4.estVide();
    Pile p5 = new Pile(Pile.TAILLE_MAX);

   //Pile p6 = new Pile(0); // déjà dans le contrat de Pile
```

blackBoxTesting suite

```
// méthodes empiler, dépiler, sommet, estVide et estPleine
p.empiler(5); assert p.sommet()==5 :"erreur sommet() ou empiler()";
              assert p.taille() == 1 :"erreur taille()";
              assert !p.estVide() :"erreur estVide()";
              assert !p.estPleine() :"erreur estPleine()";

    int res = p.dépiler(); assert res == 5 :"erreur dépiler";
                           assert p.estVide() && !p.estPleine();

      int donnée = 0;
      assert p.estVide();
      p.empiler(donnée++);   assert !p.estVide() && !p.estPleine();
      p.empiler(donnée++);   assert !p.estVide() && !p.estPleine();
      p.empiler(donnée++);   assert !p.estVide() && p.estPleine();
      assert p.sommet() == donnée-1;
```

blackBoxTesting suite

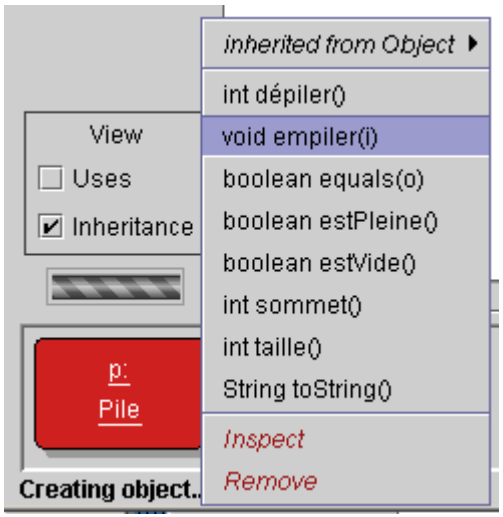
```
assert p.toString().equals("[4,3,2,1,0]") : "erreur
  toString()";
p1 = new Pile(10); Pile p2 = new Pile(10);
while(!p.estVide()){
  int i = p.dépiler();
  p1.empiler(i);
  p2.empiler(i);
}
assert p.toString().equals("[]") : "erreur toString()";
assert !p1.estVide() && !p1.estPleine();
assert !p2.estVide() && !p2.estPleine();
assert p1.equals(p2);
p1.dépiler();    assert !p1.equals(p2);
p1.empiler(10);  assert !p1.equals(p2);
p1.dépiler();p2.dépiler(); assert p1.equals(p2);
assert p1.toString().equals(p2.toString());
assert !p1.equals(new String("une pile"));
/* … */
```

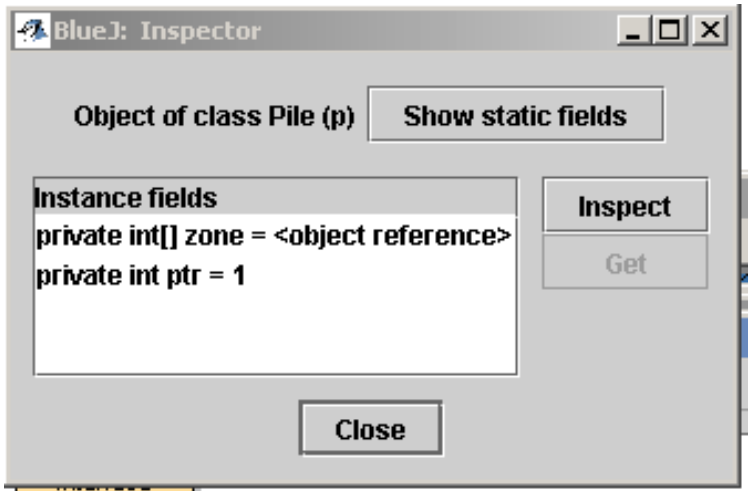
Test Exhaustif ?

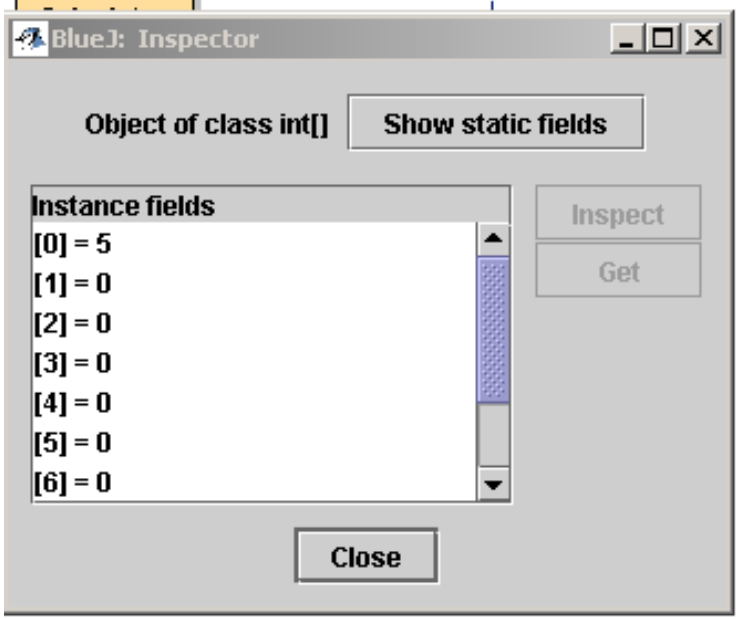
- **Non, dans la plupart des cas**
  Imaginer un compilateur Java, les tests sont infinis !!
- **Quelles méthodes ?**
- **Couverture de tests, noter ce que les tests utilisent comme code**
- **Les tests sont insuffisants, ils permettent d ’augmenter la confiance en ce logiciel, c ’est tout**

Unit Testing et BlueJ

p.empiler(5); (BlackBox)



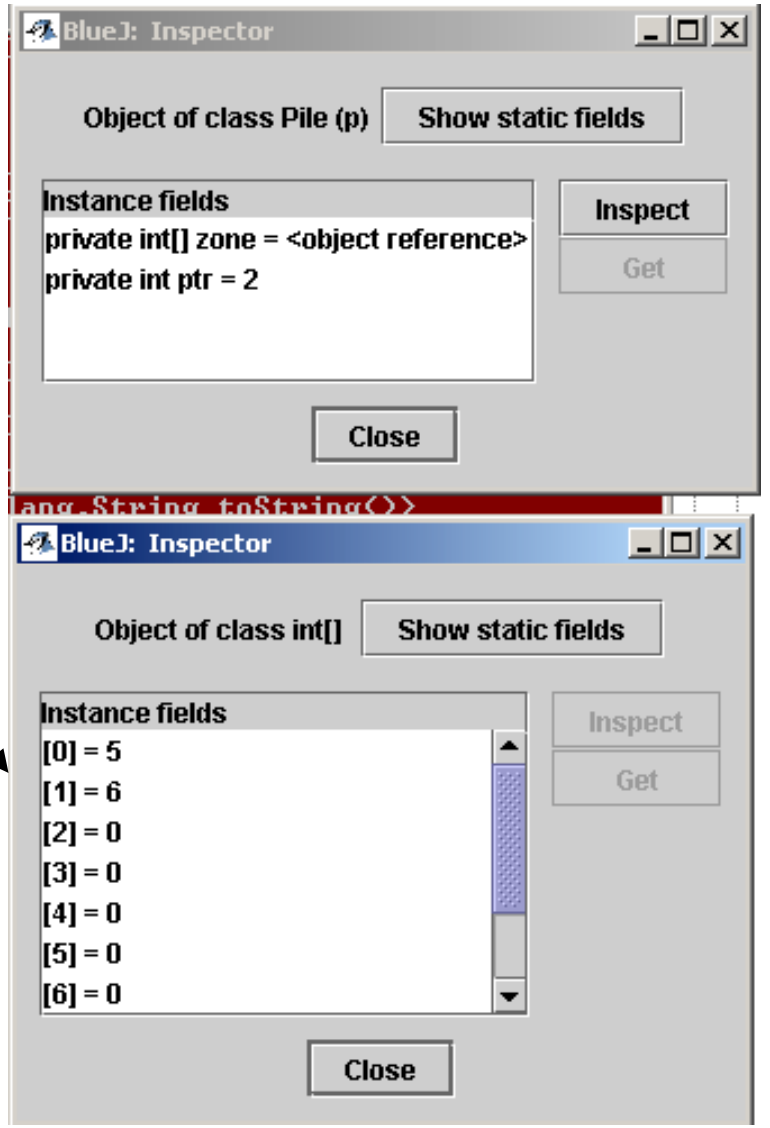




2 inspecteurs :
whiteBox

Unit Testing et BlueJ

p.empiler(6);



La calculette post fixée utilise une pile

```
public class CalculettePostFixée implements Calculator
  private Pile pile;
/* le code ici */

 public final static boolean whiteBoxTesting(){
  if(TESTING_ON){
    CalculettePostFixée calc = new CalculettePostFixée();
    assert calc.pile != null && calc.pile.taille() == 0;
    calc.numberPressed(15);   assert calc.pile.sommet()==15;
    calc.numberPressed(-20);  assert calc.pile.sommet()==-20;
    calc.plus();              assert calc.pile.sommet()==-5;
      /* ... à compléter */
      return true;
    }else{
      return false;
    }
  }
}
```

Calculette PostFixée

```
public final static boolean blackBoxTesting(){
   if(TESTING_ON){
     Calculator calc = new CalculettePostFixée();

     // 23 + 17, 23 17 +
     calc.numberPressed(23);
     calc.numberPressed(17);
     calc.plus();
     calc.equals();
     assert calc.getDisplayValue() == 40;

     // 7 -  à compléter
      return true;
   }else{
     return false;
   }

  }
```

Integration testing

Toutes les classes peuvent être réunies,
par une séquence analogue à,

La classe Pile
assert Pile.whiteBoxTesting();
assert Pile.blackBoxTesting();

La classe CalculettePostFixée
assert CalculettePostFixée.whiteBoxTesting();
assert CalculettePostFixée.blackBoxTesting();

La classe ...

Test de non-régression

- à Chaque modification du source

- nous devons avoir :

```
assert Pile.whiteBoxTesting();
assert Pile.blackBoxTesting();

assert CalculettePostFixée.whiteBoxTesting();
assert CalculettePostFixée.blackBoxTesting();
```

les test unitaires peuvent être consignés ...

Les extras

- **Les extras ne sont pas dans le livre, mais ils méritent le détour !**

  JASS

  http://semantik.informatik.uni-oldenburg.de/~jass/

  ESC/Java

  http://research.compaq.com/SRC/esc/

  une présentation : http://jfod.cnam.fr/tp_cdi/douin/ESC_Java.pdf

- **Assertions et Jass :**
  Vérification de propriétés, à l 'exécution
- **ESC/Java**
  Vérification des propriétés à l 'analyse du source

Jass Oldenburg

- **Assertions vérifiées à l ’exécution**

  installées dans le source Java sous forme de commentaires

  / require une expression à valeur booléenne **/**

  / ensure **/**

  invariant de classe

  / invariant**

TicketMachine page 35, code 2.8

```
public class TicketMachine implements Cloneable{
…
/**
 * Create a machine that issues tickets of the given price.
 */
public TicketMachine(int ticketCost){
  /** require ticketCost >0; **/
  price = ticketCost;
  balance = 0;
  total = 0;
  /** ensure balance == 0 && total == 0 && price == ticketCost; **/
}
 /**
  * Return The price of a ticket.
  */
public int getPrice(){
  return price;
 /** ensure Result == price; changeonly{}; **/
}
/**
 * Return The amount of money already inserted for the next ticket.
 */
 public int getBalance(){
   return balance;
 /** ensure Result == balance; changeonly{}; **/
}
```

ESC/Java Compaq

- **Analyse du Source**
- **Aide à la détection d ’erreurs**
  - **Sources Java / requires, ensures, invariant**
- **Deux exemples**
  - **TicketMachine page 35 code 2.8**
  - **La Pile d ’entiers**

TicketMachine page 35, code 2.8

```
public class TicketMachine{
……
//@ invariant balance >= 0 && price > 0 && total >= 0;

/**
 * Create a machine that issues tickets of the given price.
 */
 //@ requires ticketCost >0
 //@ ensures balance == 0 && total == 0 && price == ticketCost
 public TicketMachine(int ticketCost){ …


/**
 * @Return The price of a ticket.
 */
//@ ensures \result == price;
public int getPrice(){…
```

TicketMachine page 35, code 2.8

```
/**
 * Return The amount of money already inserted for the
 * next ticket.
 */
//@ ensures \result == balance;
public int getBalance()

/**
 * Receive an amount of money in cents from a customer.
 * Check that the amount is sensible.
 */
//@ modifies balance;
/*@ ensures (amount>0 && balance == \old(balance) + amount) ||
            (amount <=0 && balance == \old(balance));
*/
public void insertMoney(int amount)
```

TicketMachine page 35, code 2.8

```
/**
 * Print a ticket if enough money has been inserted, and
 * reduce the current balance by the ticket price. Print
 * an error message if more money is required.
 */
//@ modifies total, balance;
/*@ ensures balance>=price && total == \old(total)+price &&
                               balance == \old(balance)-price ||
            balance<price;
*/
public void printTicket(){  }
/**
 * Return the money in the balance.
 * The balance is cleared.
 */
//@ modifies balance;
//@ ensures \result == \old(balance) && balance ==0;
public int refundBalance()
```

Analyse statique du source passed

```
d:/esc/escjava TicketMachine.java
ESC/Java version 1.2.4, 27 September 2001
TicketMachine ...

TicketMachine: TicketMachine(int) ...
    [0.161 s]  passed

TicketMachine: getPrice() ...
    [0.04 s]  passed

TicketMachine: getBalance() ...
    [0.13 s]  passed

TicketMachine: insertMoney(int) ...
    [0.06 s]  passed

TicketMachine: printTicket() ...
    [0.11 s]  passed

TicketMachine: refundBalance() ...
    [0.02 s]  passed
```

La Pile d ’entiers en ESC

```
public class Pile{
  public final static int TAILLE_MIN =1;
  public final static int TAILLE_MAX = 1000;
  private /*@ spec_public */ int[] zone;
  private /*@ spec_public */ int   ptr;

  //@ invariant ptr >= 0 && ptr <= zone.length;
  //@ invariant zone != null;

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

Pile et ESC/Java, empiler, depiler, estVide, estPleine

```
//@ requires ptr < zone.length;
//@ modifies ptr, zone[ptr-1];
//@ ensures \old(ptr) + 1 == ptr && zone[\old(ptr)] == i;
public void empiler(int i){
  zone[ptr] = i; ptr++;
}
//@ requires ptr > 0;
//@ modifies ptr;
//@ ensures \old(ptr)-1==ptr && \result ==zone[ptr];
 public int depiler(){  ptr--; return zone[ptr];  }

//@ ensures \result == (ptr==0);
public boolean estVide(){    return ptr==0;  }
//@ ensures \result == (ptr==zone.length);
public boolean estPleine(){    return ptr==zone.length;  }
```

Pile et ESC/Java, `taille,equals, toString`

```
// pas de requires ni also_requires ici, … à voir ...
  public boolean equals(Object o){
    if(o instanceof Pile){
    //@ assume o!=null && \typeof(o) == \type(Pile);
     Pile p = (Pile) o;
     return this.zone.length == p.zone.length &&
            this.ptr == p.ptr &&
            this.memeContenu(p);
   }else{
     return false;    }  }


  //@ requires p != null && p.zone != null;
  //@ requires p.ptr == ptr;
  /*@ ensures \result ==
        (\forall int i;(i>=0 && i<ptr) ==> zone[i] == p.zone[i]);*/
private boolean memeContenu(Pile p){
```

ESC/Java whiteBox Testing

```
 //@invariant TESTING_ON == true;
private final static boolean TESTING_ON = true;

public static boolean whiteBoxTesting(){
  if(TESTING_ON){ // compilation conditionnelle
   Pile p  = new Pile(10); //@assert p.ptr==0 && p.zone != null;
                           //@assert p.zone.length == 10;
   int indice = p.ptr;
   p.empiler(3); //@assert indice+1 == p.ptr && p.zone[indice] == 4;
                 //@assert p.ptr == 1;
  indice = p.ptr;
  int res = p.depiler(); //@assert indice-1 == p.ptr && res == p.zone[p.ptr];
  boolean etat = p.estVide(); //@assert p.ptr == 0;
  while(!p.estPleine())
     p.empiler(res++);
                              //@assert p.ptr==p.zone.length;
   return true;
   }else{
  return false;    }
```

whiteBox Testing une trace ESC

```
Pile: whiteBoxTesting() ...
-----------------------------------------------------------------
Pile.java:99: Warning: Possible assertion failure (Assert)
      p.empiler(3);   //@assert indice+1 == p.ptr && p.zon ...
                                          ^
Execution trace information:
    Executed then branch in "Pile.java", line 96, col 18.
```

Analyse statique et erreur détectée
voir le transparent 23

BlackBox Testing

- **Les expressions ESC/Java ne peuvent comporter des appels de méthodes**

Résumé, synthèse

- **Tests unitaires : voir www.junit.org**
- **Voir JML + JUnit**

- **La nouvelle version de BlueJ ?**

- **De nouveaux abordés au chapitre 12**
  - **JASS (avec héritage)**
  - **Programmation par contrat**
  - **Comparaison entre Pile et java.util.Stack ? Comment ?**
  - **Conception des tests à l 'aide de Pattern (Décorateur)**
  - **…**

Documentation, avec JML

- **jmldoc Pile.java**

Specifications: spec_public

Constructor Detail

Pile

```
public Pile(int taille)
```

Specifications:

requires taille >= 1&&taille <= 1000;
ensures this.ptr == 0&&this.zone.length == taille;

Method Detail

empiler

```
public void empiler(int i)
```

Specifications:

requires this.ptr < this.zone.length;
assignable ptr, zone[this.ptr-1];
ensures \old(this.ptr)+1 == this.ptr&&this.zone[\old(this.ptr)] == i;