NFP121, Cnam/Paris
JUnit tests

Cnam Paris
jean-michel Douin, douin au cnam point fr
version du 15 Octobre 2018

Notes sur l'usage de junit3 et un peu plus…

Le lecteur intéressé par un cours sur les tests pourra se référer à celui de l'université de Trente « Software Analysis and Testing » http://selab.fbk.eu/swat/program.ml?lang=en

Avertissement, 3 documents

1. **Présentation de JUnit, les bases**
   - **junit3 est choisi délibérément, (Android utilise junit3),**
   - **Junit4 = Junit3 + Annotations cf. http://cedric.cnam.fr/~farinone/SMB212/**

2. **Android et les tests unitaires**
   - **Android utilise junit3**
   - **Les tests unitaires d'une activité, d'un service**

3. **Assertions et programmation par contrats**
   - **Pré et post assertions, héritage d'assertions**
   - **Jass,**
   - **JML, cofoja par annotations**

- **Avertissement :** *Ces documents, dont certains sont (tjs) en cours d'élaboration, peuvent être lus séparément*

Avertissement, 3 documents en images

1. **Présentation de JUnit3**
   - **Un exemple d'application**

2. **Android et les tests unitaires**
   - **Le même exemple pour Android**

3. **Assertions et programmation par contrats**

Sommaire : Les fichiers

1) junit3, les bases des tests pour java

les exemples présentés: http://jfod.cnam.fr/NFP121/junit3_tests.jar

junit4 = Junit3 + Annotations cf. http://cedric.cnam.fr/~farinone/SMB212/

2) Android,

http://en cours, en cours, en cours

3) Java et la programmation par contrats

- ***http://en cours, en cours, en cours***
- ***www.cs.ucsb.edu/~bultan/courses/272-F08/lectures/Contract.ppt***
- **Implémentation avec le patron décorateur cf. NFP121 cours 8**

Sommaire 1)

- **Tests et tests unitaires**
  - **Outil : junit www.junit.org une présentation**
- **Tests d'une application**
  - **Une pile et son IHM**
    - **Tests unitaires de la pile**
    - **Tests de plusieurs implémentations de piles**
    - **Tests d'une IHM**
    - **Tests de sources java**
    - **Invariant et fonction d'abstraction comme tests**
  - **Tests en boîte noire**
  - **Tests en boîte blanche**
  - **Doublures d'objets, bouchons pour les tests**
- **Pertinence des tests ?**
  - **Outil Cobertura *cobertura.github.io/cobertura/***
- **Génération automatique des tests ?**
  - **Randoop**, *jWalk, muJava*

Sommaire 2)

- Test unitaires et Android
- Tests d'une Activity
  - Une pile et son IHM
    - Tests unitaires de la pile
    - Tests de plusieurs piles
    - Tests de l'IHM
      - Activity
      - Service
  - Tests en boîte noire
  - Tests en boîte blanche
  - Monkey tests stressants de l'interface
  - Doublures d'objets

Sommaire 3)

- Assertions en Java
  - Assert, AssertionError
- Programmation par contrats
  - Pré-post assertions, invariant de classe
  - Invariant et variant de boucle, invariant de classe, boucles
    - Un cours/TP de NFP121
- Assertions et héritage
  - Héritage et pré et post assertions
  - Usage de Jass : source java instrumenté
  - JML, (JMLUnit),
  - cofoja : usage des annotations
- Initiation à la preuve,
  - Ce programme respecte les assertions à l'aide d'un démonstrateur
  - ESC2Java

Sommaire 1 technique

- **Assertions JUnit3**
  - Assertions JUnit
  - Assertions JUnit et levée d'une exception
  - Assertions JUnit et tests d'une IHM
  - Assertions JUnit et tests des affichages sur une console

- **Outils utilisés**
  - Utilisation de junit

    Intégré à bluej (www.bluej.org) ou autres

  - En ligne de commande
  - Cobertura
    - https://github.com/cobertura/cobertura/wiki/Command-Line-Reference

    Ou comment obtenir une mesure de la pertinence d'un jeu de tests

Bibliographie utilisée

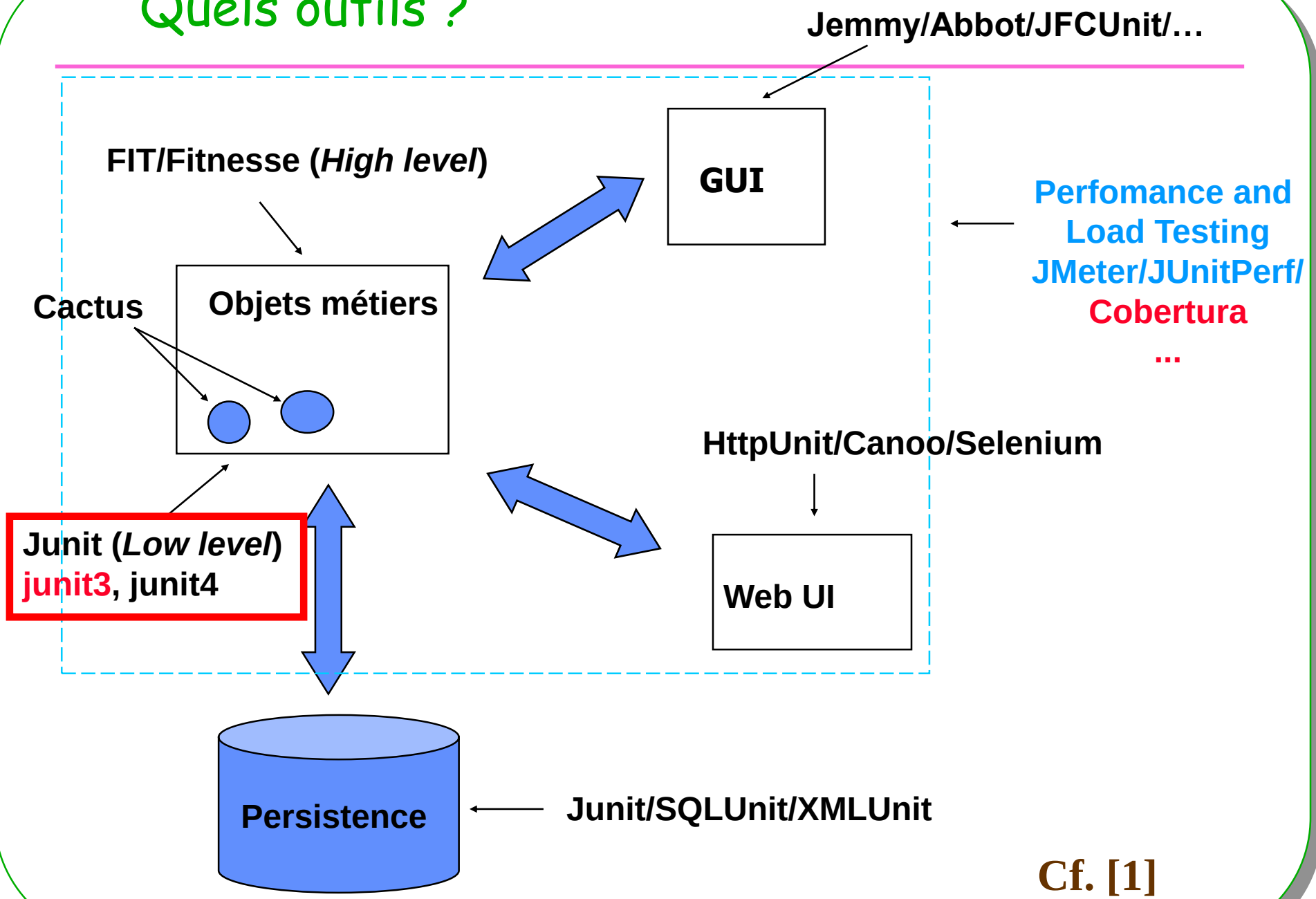
- [1] Le cours **Software Analysis and Testing** de *Paolo Tonella* , *Mariano Ceccato* *Alessandro Marchetto*, *Cu Duy Nguyen* **University of Trento**
  http://selab.fbk.eu/swat/program.ml?lang=en
  - Plusieurs figures de ce support sont extraites de cette référence
  - http://selab.fbk.eu/swat/slide/3_JUnit.ppt
    - **Lecture préalable indispensable**
- Les objets de type Mock
  - http://www.jmdoudoux.fr/java/dej/chap-objets-mock.htm
- Génération automatique de tests, un comparatif
  - http://lore.ua.ac.be/Teaching/OnderzoekStageMaster/Papers2009/NastassiaSmeetsOnderzoeksstage1.pdf

- Outils utilisés dans ce support:
- http://www.bluej.org
  - http://www.bluej.org/tutorial/testing-tutorial.pdf

- http://www.junit.org

- http://cobertura.github.io/cobertura/
- https://randoop.github.io/randoop/

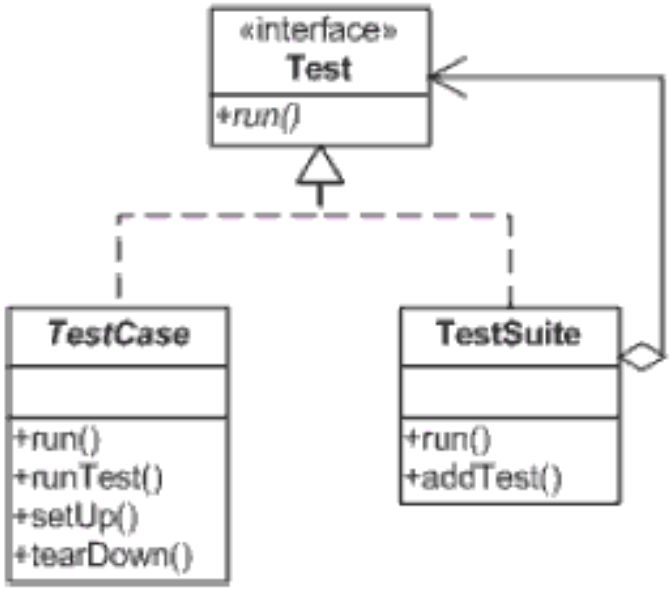
Quels outils ?

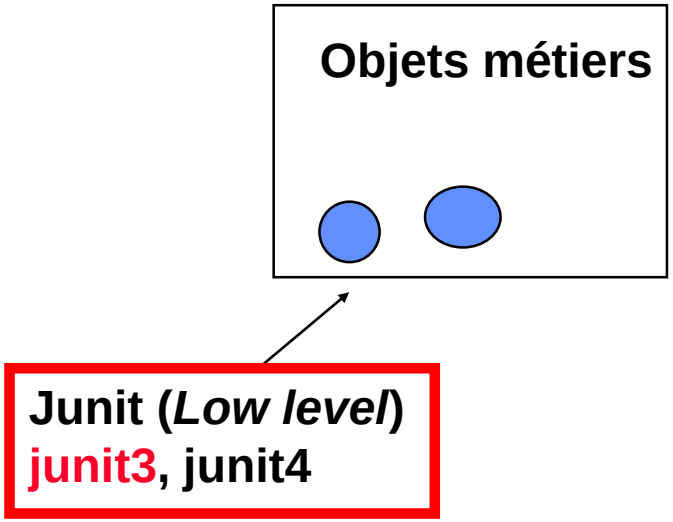


Cf. [1]

Tests et tests unitaires

- **Tests et tests unitaires**
  - **Outils : junit3 et junit4**



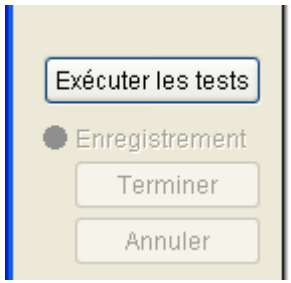


  - **Test Suite est un Test, il est constitué de TestCase et de TestSuite**
    - **Cf. Le patron Composite**
    - **Une feuille est une classe de test**
    - **Lecture associée : http://junit.sourceforge.net/doc/cookstour/cookstour.htm**

Un exemple simple 3 + 2 == 5 ?

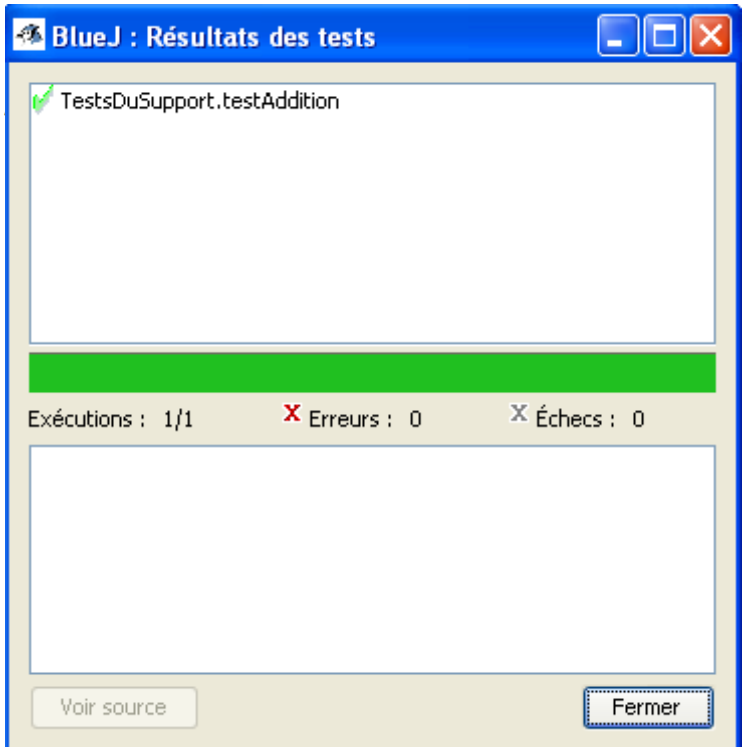
- **Un exemple simple de test unitaire**
  - de `static int plus(int x, int y){ return x+y;}`
  - Débute par l'écriture d'une assertion
    - Une assertion qui nous semble pertinente…
  - assertEquals(5, plus(3,2));
    - assertEquals(résultat attendu, le calcul effectué)
  - En Java avec Bluej et junit3

```
public class TestsDuSupport extends junit.framework.TestCase{

    public void testAddition(){
      assertEquals( 5, plus(3 + 2));
    }
```

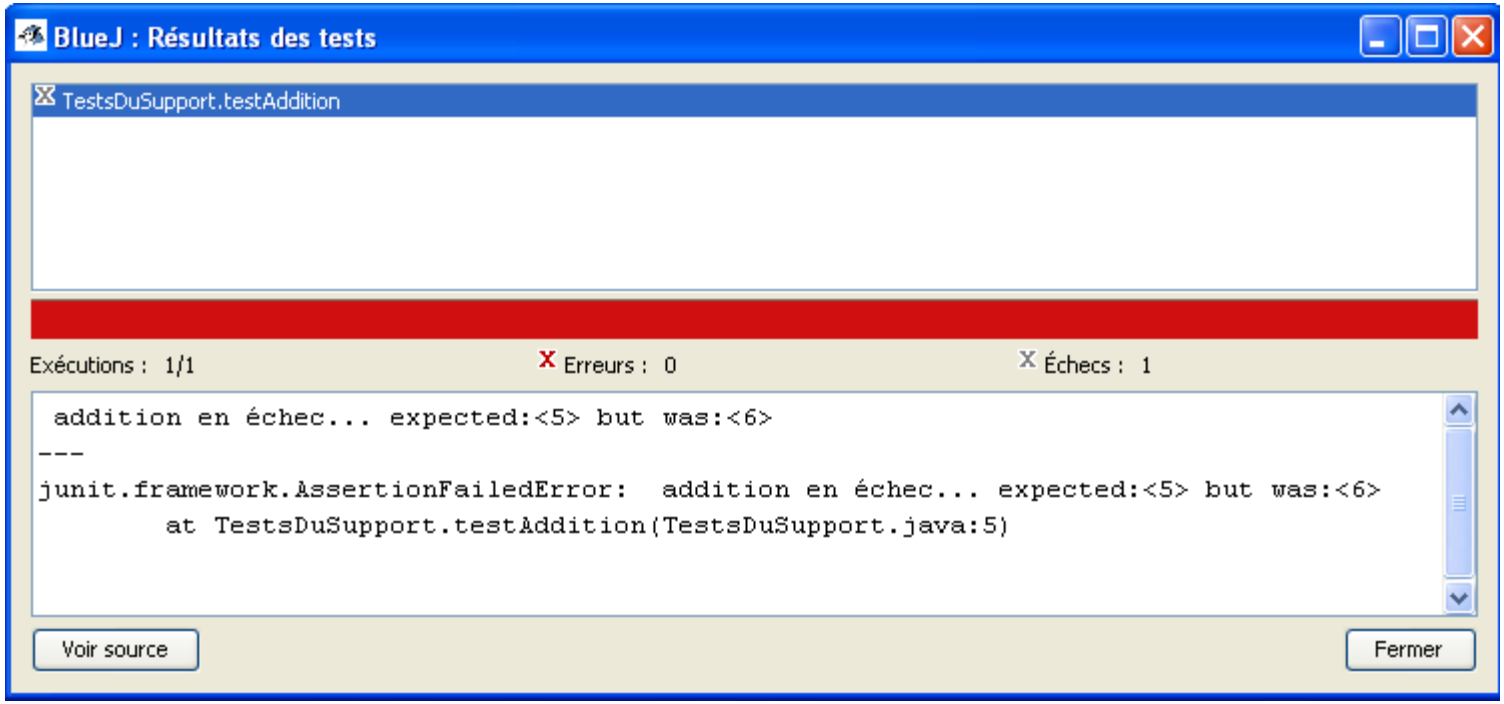


Depuis bluej : succès !

```
public class TestsDuSupport extends junit.framework.TestCase{

    public void testAddition(){
      assertEquals( 5, plus(3 + 2))
    }
```



Depuis bluej : échec ?




```
public class TestsDuSupport extends junit.framework.TestCase{


 public void testAddition(){
    assertEquals(" addition en échec... ", 5, plus(3, 3));
 }
```

Des assertions, méthodes de la classe Assert

- **Avec junit.framework.TestCase extends junit.framework.Assert**

- **assertEquals(attendu, effectif);**
- **assertEquals(" un commentaire ???" , attendu, effectif);**
- **…**
- **assertSame(" un commentaire ???" , attendu, effectif);**
- **…**
- **assertTrue(" un commentaire ???" , une expression booléenne);**
- **…**
- **assertFalse(" un commentaire ???" , une expression booléenne);**
- **…**
- **assertNotNull(" un commentaire ???" , une référence)**
- **…**
- **fail(" un commentaire ???" , une expression booléenne);**

  - **" un commentaire ??? " est optionnel**

junit.framework.Assert

- **En résumé, nous avons :**

fail
assertNotSame
assertSame
assertNull
assertNotNull
assertFalse
assertTrue
assertEquals

- **Liste obtenue par**


```
void fail()
void fail(String)
void assertNotSame(Object, Object)
void assertNotSame(String, Object, Object)
void assertSame(Object, Object)
void assertSame(String, Object, Object)
void assertNull(Object)
void assertNull(String, Object)
void assertNotNull(String, Object)
void assertNotNull(Object)
void assertFalse(String, boolean)
void assertFalse(boolean)
void assertTrue(boolean)
void assertTrue(String, boolean)
void assertEquals(byte, byte)
void assertEquals(boolean, boolean)
void assertEquals(String, boolean, boolean)
void assertEquals(long, long)
void assertEquals(String, long, long)
void assertEquals(float, float, float)
void assertEquals(String, float, float, float)
void assertEquals(double, double, double)
void assertEquals(String, double, double, double)
void assertEquals(String, String)
void assertEquals(String, String, String)
void assertEquals(Object, Object)
void assertEquals(String, Object, Object)
void assertEquals(String, char, char)
void assertEquals(char, char)
void assertEquals(String, short, short)
void assertEquals(short, short)
void assertEquals(String, int, int)
void assertEquals(int, int)
void assertEquals(String, byte, byte)
```

Vocable junit3

- *test fixture*
  - **un engagement, un contexte pour chaque test**
- *unit test*
  - **Un test unitaire d'une classe**
- *test case*
  - **Une méthode de tests, plusieurs assertions**
- *test suite*
  - **Une collection de *test case*, *Cf. patron Composite***
- *test runner*
  - **Exécution par introspection des méthodes de tests**
- *integration test*
  - **Une classe de tests de plusieurs classes**

Vocable et méthodes, avec exemples

- *test fixture*

```
protected void setUp() throws java.lang.Exception
protected void tearDown(){ throws java.lang.Exception
```

- *unit test*

```
public class IHMPileTest extends junit.framework.TestCase
```

- *test case*

```
public void testIHMConforme() throws Exception
```

- *test suite*

```
public static Test suite() {
  TestSuite suite= new TestSuite("Tests");
  suite.addTestSuite(TestsPresentsSurLeSupport.class);
  suite.add(TestPile.class);
  …
  return suite;
}
```

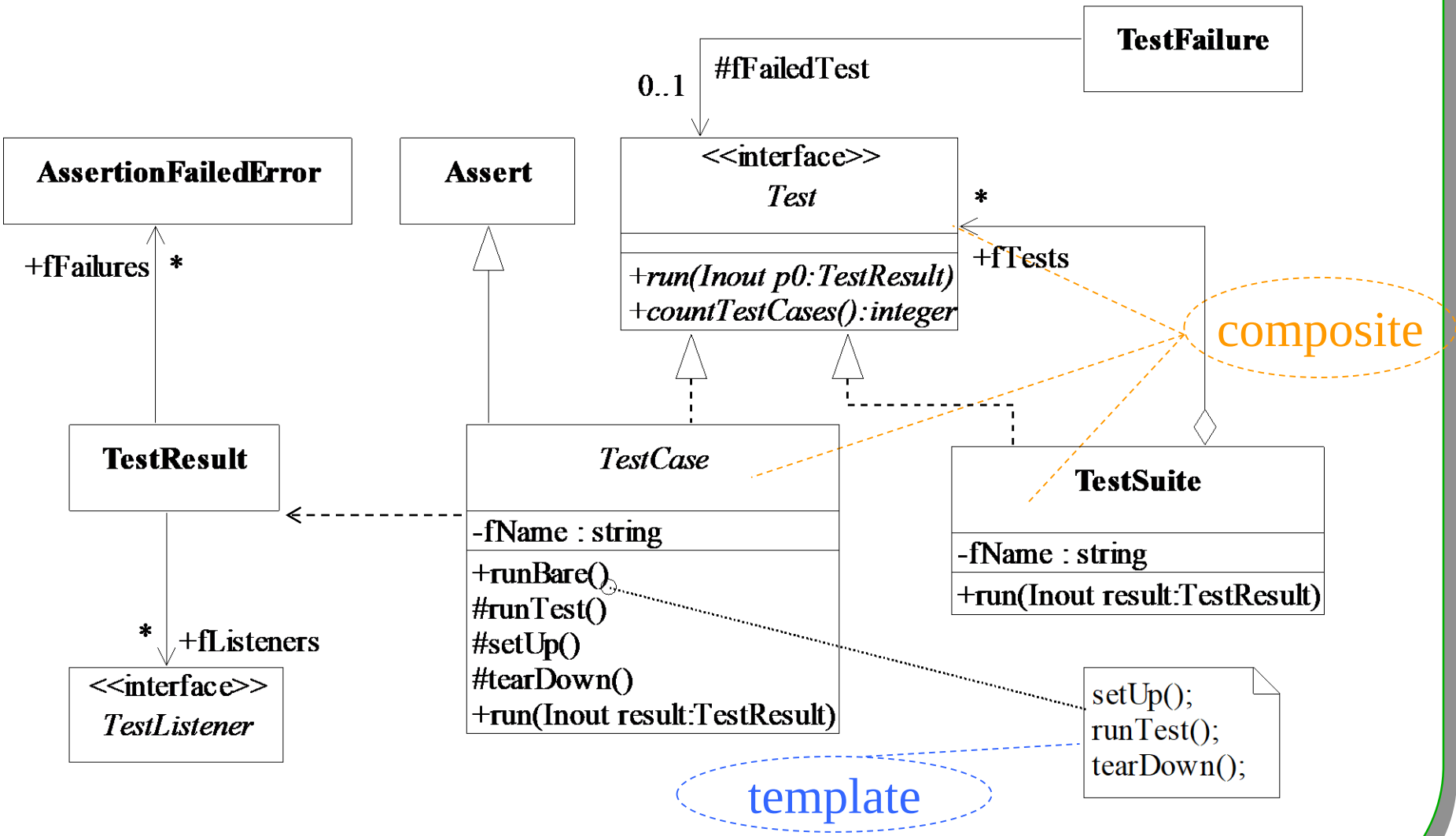
- *test runner*

```
junit.textui.TestRunner.run(suite());
```

- *integration test*

```
Une classe de tests de plusieurs classes
```

Junit est un *framework*,
une baie d'accueil de vos tests

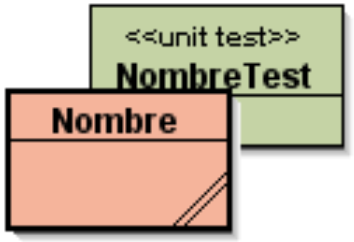


La suite…

- **Test unitaire d'une classe**
  - **Assertions simples sur les valeurs d'un accesseur.**
- **Test unitaire avec engagement**
  - **Les méthodes setUp et tearDown.**
- **Deux implémentations d'une même interface**
  - **Deux classes de tests unitaires quasi identiques.**
    - **Usage du patron fabrique**
- **Ce sont des tests en boîtes noires**
  - **Tests de l'extérieur du résultat des méthodes**
- **En boites blanches: par introspection**
  - **Accès aux attributs d'une classe (même privées).**

Test unitaire d'une classe, la classe Nombre

```
public class Nombre{
   private int valeur;

   public Nombre(int valeur){
      this.valeur = valeur;
   }

   public int valeur(){
      return this.valeur;
   }
}
```

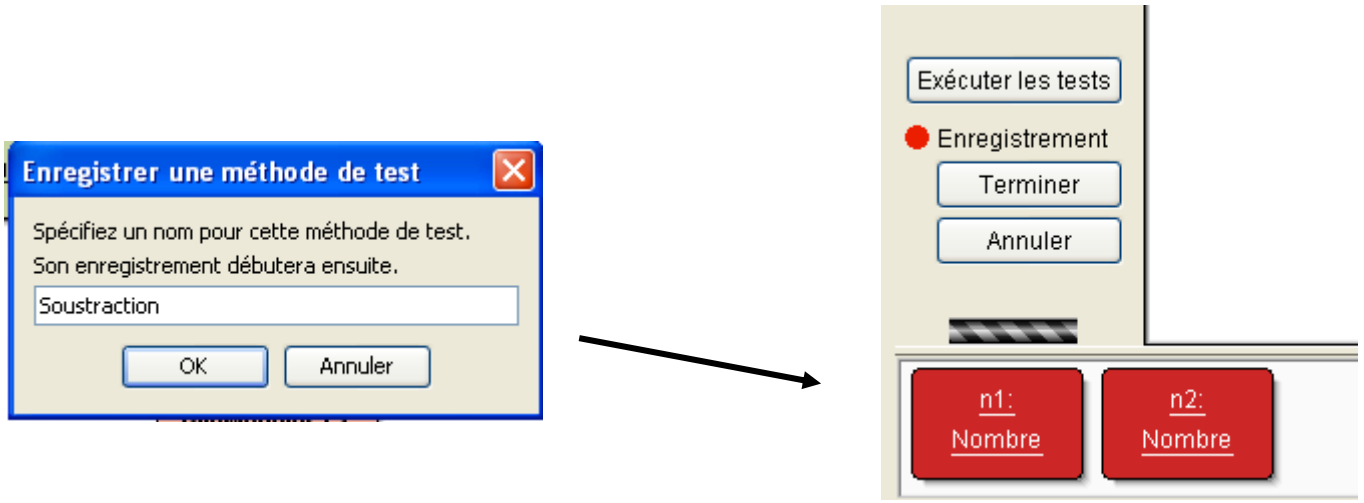


Attributs d'une classe de tests : Engagez vous !

```
public class NombreTest extends junit.framework.TestCase{
  private Nombre n1;
  private Nombre n2;
  /** Met en place les engagements.
    * Méthode appelée avant chaque appel de méthode de test.
    */
    protected void setUp(){
        n1 = new Nombre(3);
        n2 = new Nombre(2);
    }
    /**
     * Supprime les engagements.
     * Méthode appelée après chaque appel de méthode de test.
     */
    protected void tearDown(){
        //Libérez ici les ressources engagées par setUp()
    }

    public void testAddition(){
        assertEquals(" accesseur en défaut ???", 3, n1.valeur());
        assertEquals(5, n1.valeur() + n2.valeur());
        Nombre n3 = new Nombre(5);
        assertEquals(10, n1.valeur() + n2.valeur() + n3.valeur());
    }
```

Les engagements, vus par bluej

- **A chaque création d'un test, les engagements sont respectés**

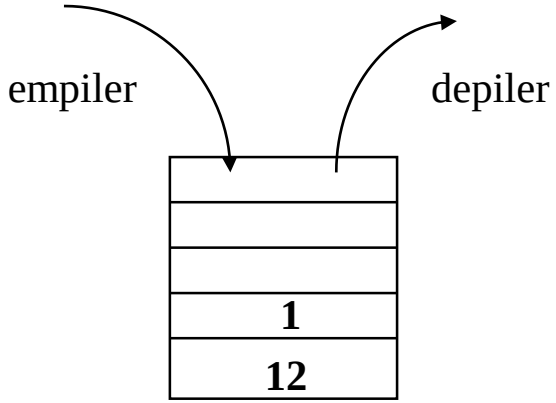


- **A l'exécution, nous avons**
  - **setUp();**
  - **testAddition();**
  - **tearDown();**

Discussion, Nombre et NombreTest

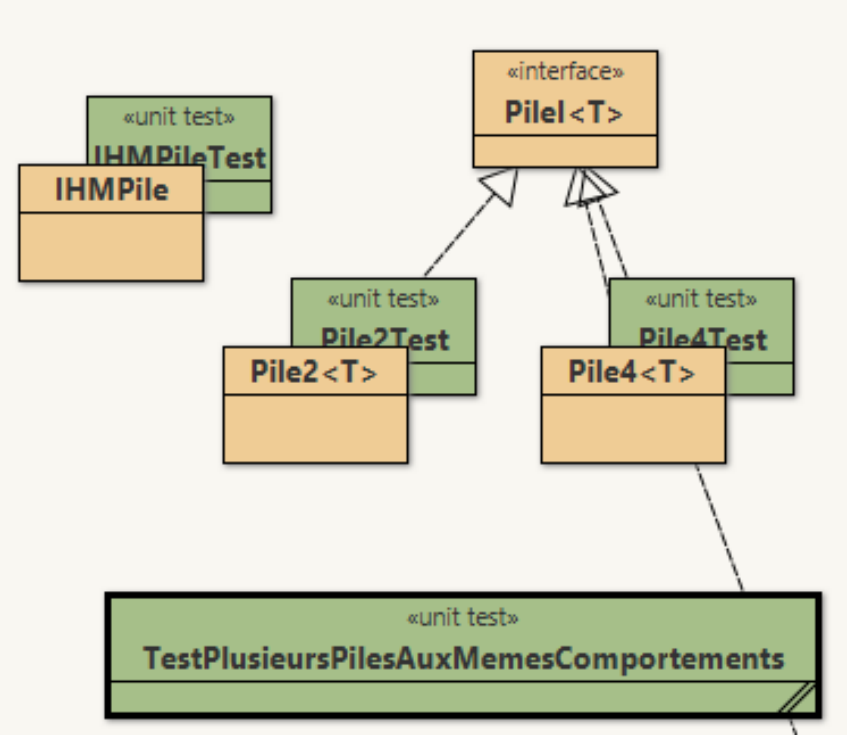
- **Quand faut-il développer la classe de test ?**
  - **Avant ou après le développement de la classe Nombre**
  - **Avant**
    - **Comportement attendu, la classe n'est pas écrite,**
    - **Seules les spécifications sont utilisées,**
    - **Développement initié par les tests**
  - **Après**
    - **Comportement existant**
    - **Compréhension du comportement**
  - A lire pour en savoir un peu plus
    - http://selab.fbk.eu/swat/slide/3_**JUnit.ppt**
    - **extreme programming test driven development**
  - **Discussion**

Tests d'une application complète



- **Une IHM**
- **Un modèle : la pile**

Une application : une Pile



- **Une interface PileI<T>**

- **Deux implémentations de cette interface Pile2<T> et Pile4<T>**
  - **Test Unitaire**
- **Une IHM IHMPile**
  - **Test Unitaire**
- **Une classe de tests de plusieurs piles avec le même comportement**
  - **Test d'intégration**

Une interface

```
4  public interface PileI<T>{
5
6    public final static int CAPACITE_PAR_DEFAUT=6;
7
8    // mutateurs
9    public void empiler(T t) throws PilePleineException;
10   public T depiler() throws PileVideException;
11
12   // accesseurs
13   public T sommet() throws PileVideException;
14   public int capacite();
15   public int taille();
16   public boolean estVide();
17   public boolean estPleine();
```

- **Ce que toute pile devra proposer**

Pile2Test, testEmpiler

```
public void testEmpilerCapaciteElement(){
  try{
    assertTrue(" La pile n'est pas vide ?", p.estVide());
    assertEquals(" La pile est vide, la taille <> 0 ?", 0, p.taille());
    p.empiler(3);
    assertFalse(" La pile serait-elle vide, après empiler ?", p.estVide());
    assertFalse(" La pile n'est pas pleine, après empiler ?", p.estPleine());
    assertEquals(" La taille de la pile <> 1 ?", p.taille(), 1);

    try{
      while(!p.estPleine()){
        p.empiler(3);
      }
      assertEquals(" La pile est pleine, la taille <> la capacité ?", p.taille(), p.capacite());
      p.empiler(3);
      fail("La pile est pleine, et empiler, une exception doit être levée !");
    }catch(Exception e){
      assertTrue(" Est-ce la \"bonne\" exception ?", e instanceof PilePleineException);
    }
  }catch(Exception e){
    fail("Exception inattendue ?" + e.getMessage());
  }
}
```

- **Tests de la méthode empiler …**
  - **Exhaustif ?**

Pile2Test, testDepiler

```
public void testDepiler(){
  try{
    assertTrue(" La pile n'est pas vide ?", p.estVide());
    assertEquals(" La pile est vide, la taille <> 0 ?", 0, p.taille());
    p.empiler(3);
    p.empiler(2);
    p.empiler(1);
    assertEquals(" depiler ne fournit pas la bonne valeur!",1, p.depiler().intValue());
    assertEquals(" depiler ne fournit pas la bonne valeur!",2, p.depiler().intValue());
    assertEquals(" depiler ne fournit pas la bonne valeur!",3, p.depiler().intValue());
    assertTrue(" La pile n'est pas vide ?", p.estVide());
    p.empiler(2);
    p.empiler(1);
    assertEquals(" depiler ne fournit pas la bonne valeur!",1, p.depiler().intValue());
    assertEquals(" depiler ne fournit pas la bonne valeur!",2, p.depiler().intValue());
    assertTrue(" La pile n'est pas vide ?", p.estVide());

    try{
        p.depiler();
        fail("La pile est vide, et depiler, une exception doit être levée !");
    }catch(Exception e){
      assertTrue(" Est-ce la \"bonne\" exception ?", e instanceof PileVideException);
    }
  }catch(Exception e){
    fail("Exception inattendue ?" + e.getMessage());
  }
```

- **La pile testée, p, est initialisée par un engagement, cf. setUp**

Engagement de Pile2Test

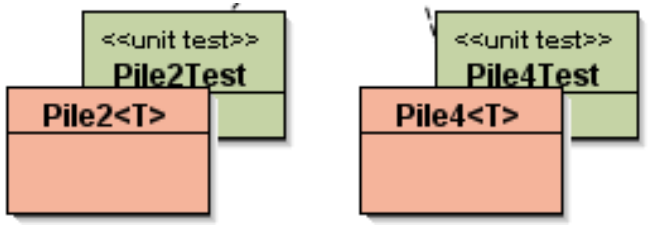
```
public class Pile2Test extends junit.framework.TestCase{

   private Pilel<Integer> p;


   protected void setUp(){
      this.p = new Pile2<Integer>();
   }

   protected void tearDown(){
   }
```

et toutes les initialisations de piles « locales »

Tests du constructeur

```
public void testConstructeurAvecTailleNulle(){
    try{
      PileI<Integer> p = new Pile2<Integer>(0);
      assertEquals("Capacité par défaut attendue…", PileI.CAPACITE_PAR_DEFAUT , p.capacite());
    }catch(Exception e){
      fail("Exception inattendue ! " + e.getMessage());
    }
  }


  public void testConstructeurAvecTaillePositive(){
    try{
      PileI<Integer> p = new Pile2<Integer>(10);
      assertEquals("Taille de 10, la capacité de 10 est attendue !", 10, p.capacite());
    }catch(Exception e){
      fail("Exception inattendue ! " + e.getMessage());
    }
  }


  public void testConstructeurAvecTailleNegative(){
    try{
       PileI<Integer> p = new Pile2<Integer>(-1);
      assertEquals("capacité par défaut attendue…", PileI.CAPACITE_PAR_DEFAUT, p.capacite());
    }catch(Exception e){
      fail("Exception inattendue ! " + e.getMessage());
    }
  }
```

Deux classes de test, presque identiques



- **Les deux classes de tests ont un contenu identique,**
  - **Seule change l'instance de la classe Pile2<T> ou Pile4<T> …**

Comment factoriser ?

- **Substituer new Pile2<Integer> par new Pile4<Integer>**

- **Un paramètre de la classe de tests ?**
  - **Un fichier de configuration ?**
  - **Injection de dépendance ?**
  - **Usage du patron Fabrique ?**

Le contenu diffère de peu, alors

- **Usage du patron fabrique, (ou usage de l'injection de dépendance)**

```
private PileI<Integer> p;


    private static class Fabrique<T>{
      PileI<T> creerPile(){
        return new Pile4<T>();
      }

      PileI<T> creerPile(int taille){
        return new Pile4<T>(taille);
      }
    }


    protected void setUp(){
       this.p = new Fabrique<Integer>().creerPile();
    }
```

Le contenu diffère de peu, alors

- **Usage du patron fabrique**

```
private PileI<Integer> p;


    private static class Fabrique<T>{
      PileI<T> creerPile(){
        return new Pile2<T>();
      }

      PileI<T> creerPile(int taille){
        return new Pile2<T>(taille);
      }
    }


    protected void setUp(){
       this.p = new Fabrique<Integer>().creerPile();
    }
```

Plusieurs implémentations de Piles

Ont-elles le même comportement ?

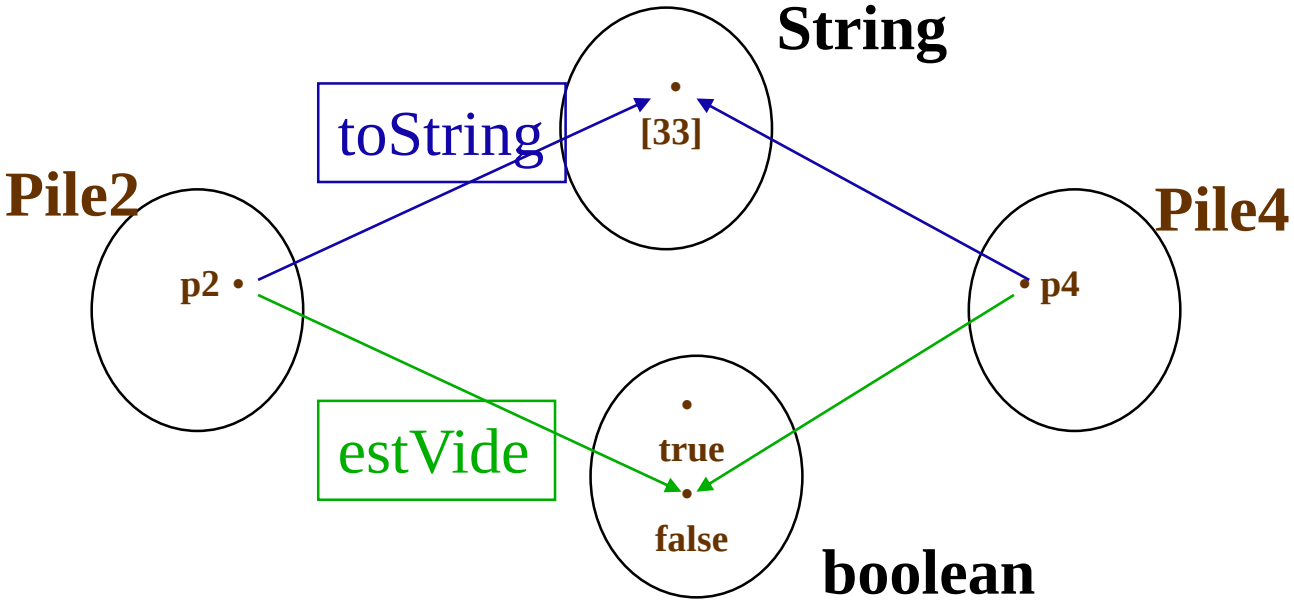
```
Pile2<Integer> p2 = new Pile2<Integer>();
Pile2<Integer> p4 = new Pile2<Integer>();
assertEquals(p2.estVide(), p4.estVide());
assertEquals(p2.estPleine(), p4.estPleine());
…
```

Même comportement, synthèse

- **Pour chaque méthode, le résultat doit être identique**

Deux exemples : estVide, toString

PileI<Integer> p2 = new Pile2<Integer>(); p2.empiler(33);
PileI<Integer> p4 = new Pile4<Integer>(); p4.empiler(33);



Les piles doivent avoir le même comportement


```
public void test_meme_comportement(){
 try{
   p2.empiler("aze");
   p1.empiler("aze");
    assertEquals(p1.capacite(), p2.capacite());

    assertEquals("[aze]", p1.toString());
   assertEquals(p1.toString(), p2.toString());

   assertEquals(p1.sommet(), p2.sommet());
    assertEquals(p1.estVide(), p2.estVide());
   assertEquals(p1.estPleine(), p2.estPleine());



    /// …
}catch(StackOverflowError e){
      fail(" récursivité ?? " + e.getClass().getName());
    }catch(NoSuchMethodError e){
       fail("exception inattendue ! " + e.getClass().getName());
     }catch(Exception e){
      fail("exception inattendue ! " + e.getClass().getName());
    }
   }
```

Avons-nous tout testé ?, sommes nous sûrs de ne pouvoir le faire ?

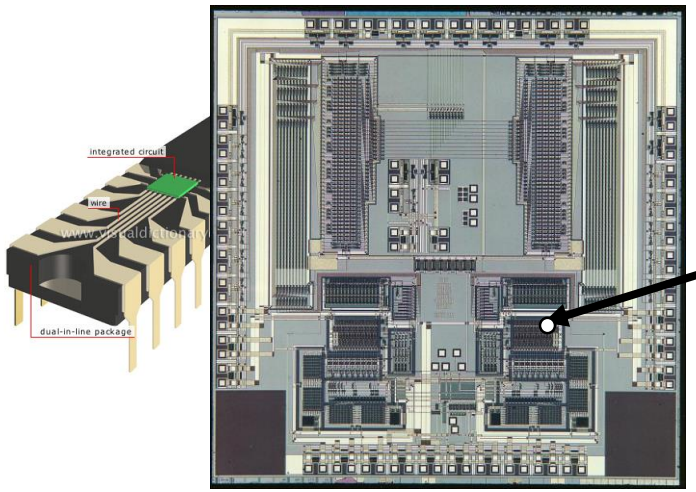
Boites noires, boites blanches ?

- **Tests des méthodes publiques et uniquement**

Une pile
dans sa boite noire

```
assertEquals(3, p1.sommet());
assertFalse(p1.estVide());
assertTrue(p1.estPleine());
```

Boites blanches … et introspection



Une pile décortiquée

L'attribut **nombre**

!

- **Tests de la Présence d'un attribut, d'une méthode**
  - **Lecture d'un attribut privé**
  - **Appel d'une méthode, vérification**
  - **…**
- **Réservés aux créateurs de la classe ? Abscons ?**
- **Discussion … utile/inutile**

Par introspection …

```
public class ParIntrospection extends junit.framework.TestCase{

  private PileI<Integer> p4;

  protected void setUp() throws Exception{
    this.p4 = new Pile4<Integer>();
  }

  public void testAttributNombre() throws Exception{
    p4.empiler(3);
    Integer v = getFieldValue("nombre");
    assertEquals(p4.taille(),v.intValue());
  }

  // accès aux champs privés d'une instance quelconque …
  private <T> T getFieldValue(String nom) throws Exception{
    Field f = Pile4.class.getDeclaredField(nom);
    f.setAccessible(true);
    return (T) f.get(p4);
  }
}
```

Pause, détente : Liskov

- **Documentation d'une classe ?**
  - **Pour les utilisateurs : javadoc**
  - **Pour les implémenteurs ? les développeurs ? les successeurs**
    - **Liskov dans son livre propose deux notions**
      - **Fonction d'abstraction**

      et

      - **Invariant de représentation**
    - **Comme commentaires dans le code ou bien en méthodes**

Page 99

Aids to Understanding Implementations

In this section, we discuss two pieces of information, the abstraction function and the representation invariant, that are particularly useful in understanding an implementation of a data abstraction.

The *abstraction function* captures the designer's intent in choosing a particular representation. It is the first thing you decide on when inventing the rep: what instance variables to use and how they relate to the abstract object they are intended to represent. The abstraction function simply describes this decision.

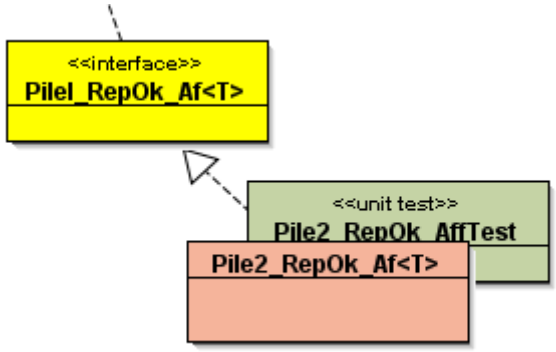
The *rep invariant* is invented as you investigate how to implement the constructors and methods. It captures the common assumptions on which these implementations are based; in doing so, it allows you to consider the implementation of each operation in isolation of the others.

The abstraction function and rep invariant together provide valuable documentation, both to the original implementor and to others who read the code. They capture the reason why the code is the way it is: for example, why the implementation of `choose` can return the *zero*th element of `els` (since the elements of `els` represent the elements of the set), or why `size` can simply return the size of `els` (because there are no duplicates in `els`).

Because they are so useful, both the abstraction function and rep invariant should be included as comments in the code. This section describes how to define them and also how to provide them as methods.

repOk et af de Liskov

- **Invariant de représentation et fonction d'abstraction**
  - Une aide à la compréhension de l'implémentation
  - La pile2 est revisitée



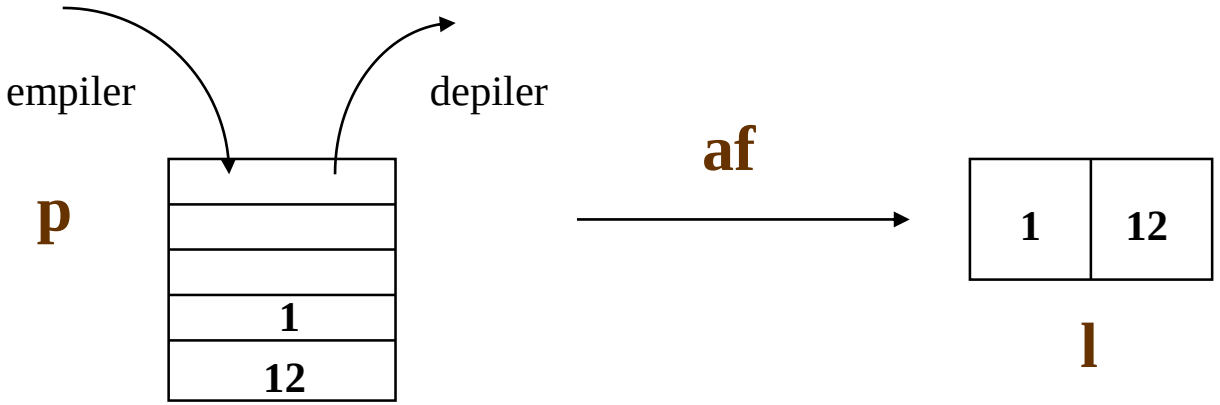

```
public interface PileI_RepOk_Af<T> extends PileI<T>{
  // documentation pour implémenteurs
  // invariant de représentation ... cf. Liskov chap. 5
  public boolean repOk();
  // fonction d'abstraction ... cf. Liskov chap. 5
  public List<T> af();
}
```

Invariant de représentation, fonction d'abstraction

- **Invariant de représentation, repOk**
  - Si la pile a été intialisée correctement
  - Alors pour tous les objets présents, ils doivent être différents de null
    - L'invariant est vrai avant et après l'appel de chaque méthode

    et après l'appel du constructeur

- **Fonction d'abstraction, af**
  - Toute Pile p a une représentation en Liste l



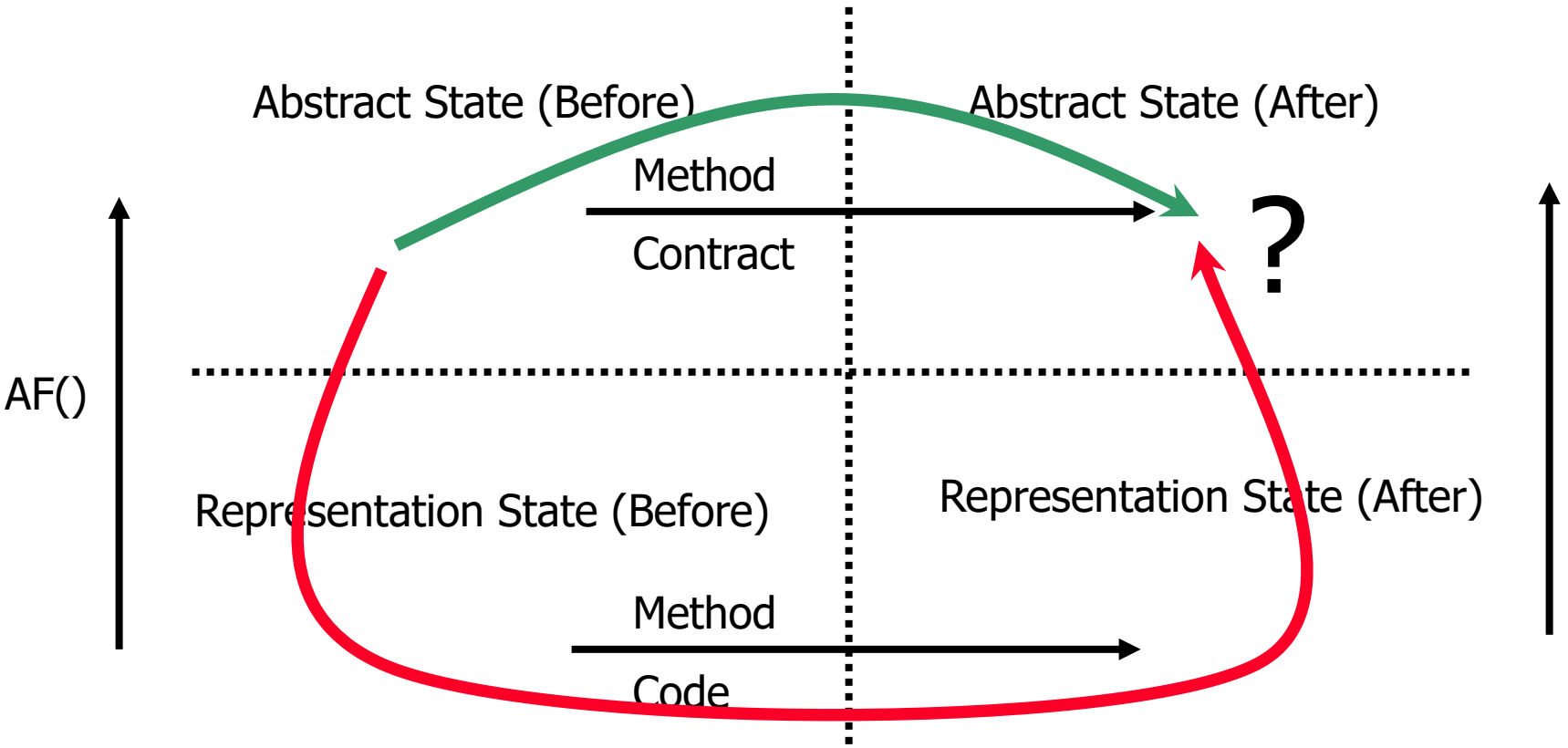
repOk et af d'une Pile2

```
public class Pile2_RepOk_Af<T> implements PileI_RepOk_Af<T>{
  private Stack<T> stk;
  private int      capacité;

// Si la pile a été intialisée correctement
//  alors pour tous les objets présents, ils doivent être différents de null
  public boolean repOk(){
    if (stk!=null && capacité > 0){
      for(Object o : stk)
        if(o==null) return false;
      return true;
    }
    return false;
  }

// de Pile en List, à chaque pile lui correspond une liste
 public List<T> af(){
    List<T> liste = new ArrayList<T>();
    for(int i = stk.size()-1; i>=0;i--){
      liste.add(stk.elementAt(i));
    }
    return liste;
  }
```

Usage de repOk dans les tests unitaires

```
public void testEmpiler(){
   try{
     assertTrue("repOk en échec !!!", p.repOk());

     p.empiler(3);

     assertTrue("repOk en échec !!!", p.repOk());


     assertTrue("repOk en échec !!!", p.repOk());

       while(!p.estPleine()){
         p.empiler(3);
         assertTrue("repOk en échec !!!", p.repOk());


       }

     assertTrue("repOk en échec !!!", p.repOk());
```

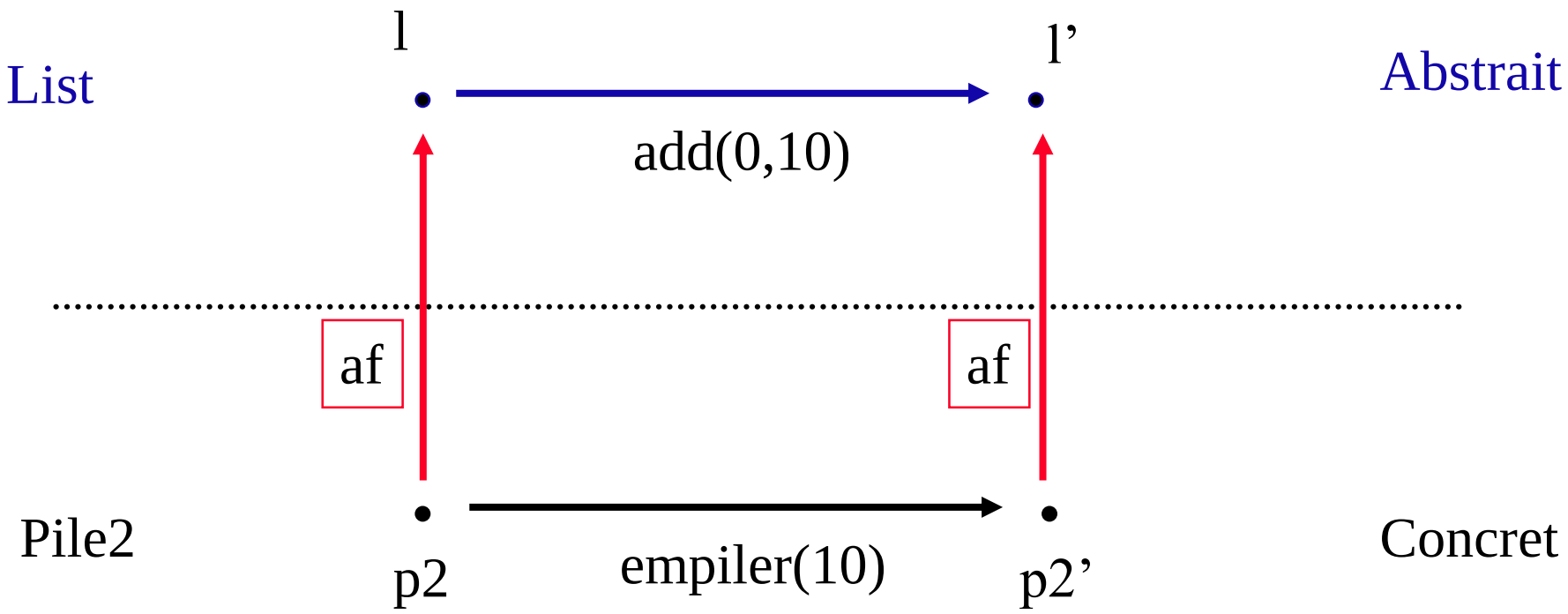
Fonction d 'abstraction: une explication s'impose



- Source : http://cs.gmu.edu/~pammann/619/ppt/DataAbstraction2.ppt

p2.af().add(0,10).equals(p2.empiler(10).af()) ?

- **Pour chaque opération :**
  - **nous devons vérifier que le graphe commute …**



- **assertEquals(**p2.af().add(0,10), p2.empiler(10).af()**)**

Les tests unitaires deviennent

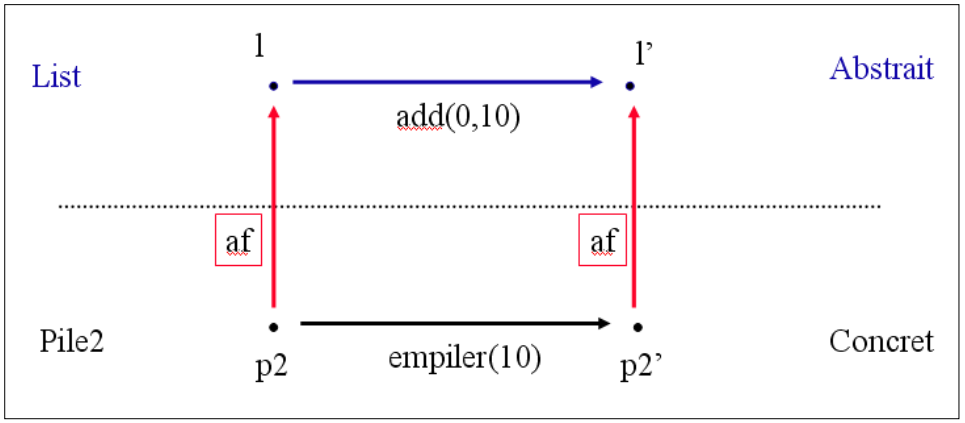
Hyp. : La classe Pile2 est le résultat d'un raffinement de la Liste …

```
public void testRafinementEmpiler() throws Exception{
    List<Integer> l1 = p.af();
    l1.add(0,33);
    p.empiler(33);
    assertTrue(l1.equals(p.af()));

    List<Integer> l2 = p.af();
    l2.add(0,45);
    p.empiler(45);
    assertTrue(l2.equals(p.af()));
    // toString
    assertTrue(l2.toString().equals(p.af().toString()));
  }
```

Exercice de style ?, utilise/inutile ? Discussions

Raffinement et génie logiciel

- **Comment raffiner une classe par une autre ?**
- **Substitution d'une classe par une autre ?**
- **Choisir une implémentation plus performante**
  - **En empreinte mémoire**
  - **En temps d'exécution**
- **…**
- **Discussion …**



Sommaire, suite, quelques techniques

- **Tests d'un affichage**
- **Test de la levée ou non d'une exception**
- **Tests d'une IHM**
  - **JFrame**
- **Simuler le comportement d'une classe :**
  - **En attendant qu'elle soit développée, livrée, testée,…**
  - **Une solution : Les doublures d'objets, les Mock**

Tests d'un affichage sur la console

```
3  public class Main{
4
5    public static void main(String[] args){
6      for(String str : args){
7        try{
8          int opr = Integer.parseInt(str);
9          System.out.println(Integer.toString(opr+1));
10       }catch(Exception e){
11         System.out.println("erreur pour : " + e.getMessage());
12       }
13     }
14   }
15 }
```

- **La sortie obtenue est-elle conforme ?**

```
public void testSystemOut() throws Exception{

    String[] console = mainExec("Main",new String[]{"20","xxx","30","50"});

    assertEquals(" le nombre de lignes attendu n'est pas correct ...", console.length, 4);

    assertEquals(" sortie non conforme ???", "21", console[0]);
    assertTrue(" sortie non conforme ???", console[1].contains("For input string: \"xxx\""));
    assertEquals(" sortie non conforme ???", "31", console[2]);
    assertEquals(" sortie non conforme ???", "51", console[3]);

}
```

Tests d'un affichage sur la console, un utilitaire

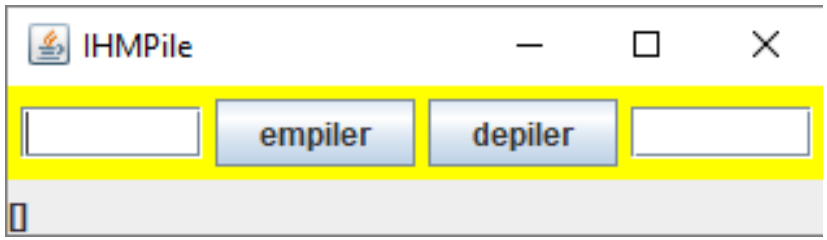
```
/** Obtention de l'affichage produit par l'exécution de la méthode main d'une classe.
    * @param className le nom de la classe
    * @param args les arguments de la ligne de commande
    * @return le texte en tableau de lignes
    * @throws une exception est levée si la classe est inconnue
    */
    public static String[] mainExec(String className, String[] args)
                                                              throws Exception{
      java.io.PrintStream out = System.out;
      String[] consoleOut = null;
      try{
        java.io.ByteArrayOutputStream baos = new java.io.ByteArrayOutputStream();
        java.io.PrintStream ps = new java.io.PrintStream(baos);
        Class<?> c = Class.forName(className);
        System.setOut(ps);
        c.getMethod("main",String[].class).invoke(null, new Object[]{args});
        consoleOut = baos.toString().split(System.getProperty("line.separator"));
      }catch(java.lang.reflect.InvocationTargetException e){
           int line = e.getTargetException().getStackTrace()[0].getLineNumber();
           String fileName = e.getTargetException().getStackTrace()[0].getFileName();
           String msg = "Exception " + e.getCause().toString() + " at line " +
                                                          line + " of " + fileName;
           consoleOut = new String[]{msg};
      }finally{
        System.setOut(out);
      }
      return consoleOut;
    }
```

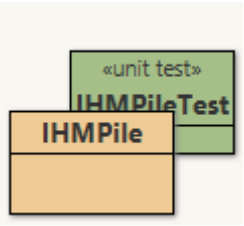
À recopier telle quelle dans votre classe de tests

Test d'une exception

```
assertTrue(" La pile n'est pas vide ?", p.estVide());

try{
  int s = p.sommet();
  fail("une exception doit être levée !");
}catch(Exception e){
    assertTrue(" Est-ce la \"bonne\" exception ?",
                                e instanceof PileVideException);
 }
```

Test d'une IHM (ici une instance de JFrame)





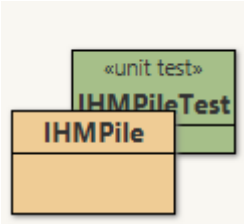
- **Cette IHM, est une boite noire …**
  - **Simulation des clics de l'utilisateur,**
  - **vérification des affichages produits dans les composants graphiques**

1) Obtention de l'IHM
2) Est-il conforme (à l'énoncé)
3) Click() et résultats attendus

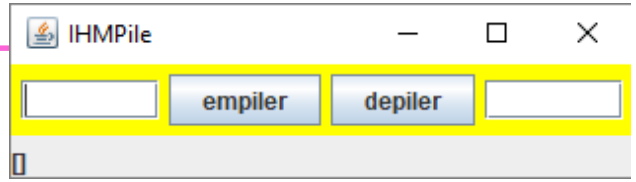
Test d'une IHM (une instance de Jframe)

- **1) Obtention des composants graphiques d'une JFrame**

```
protected void setUp() throws java.lang.Exception{

    f = new IHMPile()
    f.pack();
    f.setVisible(true);
```



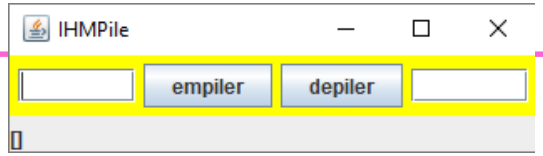
Test d'une IHM



- **2) L'IHM est-elle conforme ?**

```
public void testIHMConforme() throws Exception{ // juste le bon IHM
   try{
      Container panel = f.getContentPane();
      Component[] components = panel.getComponents();
      assertEquals(components.length, 2);

      // la bonne IHM
      assertTrue(" Est-ce la bonne IHM ?", components[0] instanceof JPanel);
      assertTrue(" IHM ?",components[1] instanceof JLabel);

      Component[] subComponents = ((JPanel)components[0]).getComponents();
      assertTrue(subComponents[0] instanceof JTextField);
      assertTrue(subComponents[1] instanceof JButton);
       assertTrue(subComponents[2] instanceof JButton);
      assertTrue(subComponents[3] instanceof JTextField);


     }catch(Exception e){
     fail("exception inattendue ! " + e.getClass().getName());
   }
  }
```

Test d'une IHM



- **3) Un clic sur l'un des boutons**

```
private void empiler(String str) throws Exception{
  Container panel = f.getContentPane();
  Component[] components = panel.getComponents();
  Component[] subComponents = ((JPanel)components[0]).getComponents();

   ((JTextField)subComponents[0]).setText(str);
   ((JTextField)subComponents[0]).postActionEvent();

   JButton btnEmpiler = (JButton)subComponents[1];
   btnEmpiler.doClick();
}
```

- **Note** : *postActionEvent déclenche les listeners*

Test d'une IHM

- **4) Vérification du contenu de l'un des champs**

```
private String pileToString() throws Exception{
  Container panel = f.getContentPane();
  Component[] components = panel.getComponents();
  JLabel sortie = (JLabel) components[1];
  return sortie.getText();
 }
```

```
public void testEmpiler() throws Exception{
  try{
    empiler("100");
    empiler("101");
    String sortie = pileToString();
    assertEquals(" La sortie conforme ???","[101, 100]", sortie);
  }catch(Exception e){
    fail("exception inattendue ! " + e.getClass().getName());
  }
}
```

SwingUtilities

- **Si cela le nécessite …**

```
SwingUtilities.invokeLater(new Runnable() {
        public void run() {
           popup.doClick();
        }
      });
```

Attend que tous les évènements de l'AWT soient terminés

Les doublures

- **Simuler le comportement d'une classe :**
  - **En attendant qu'elle soit développée, livrée, testée,…**
  - **Une solution : Les doublures d'objets, les Mock**

- **A lire : http://www.jmdoudoux.fr/java/dej/chap-objets-mock.htm**

Les doublures :à terminer

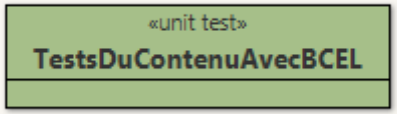
- **À terminer**

- **cf. mockito**

Vérification de sources java

- Comment vérifier le nombre de méthodes d'une classe ?
- Appels ou non de méthodes comme indiqué dans l'énoncé ?
- Absence ou présence d'instructions telles que for / while … etc
- Une proposition : usage de bcel, une bibliothèque d'analyse du.class
  - https://commons.apache.org/proper/commons-bcel/
  - Sources complets cf.

Analyse de sources Java : exemples d'assertions



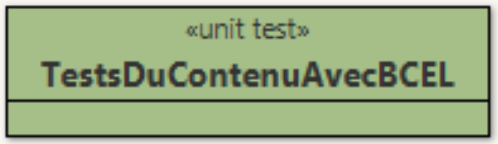
- **Les attendus d'un TP… par exemple…**

- **Énoncé : Développement d'une classe de tests**
  - **Avec les appels de empty et pop (java.util.Stack)**
  - **Avec au moins 4 méthodes**
  - **Et plusieurs appels de assertEquals …**

```
public void test1(){
        Stack<Integer> stk = new Stack<Integer>();
        boolean b = stk.empty();
        stk.push(33);
        int elt = stk.pop();
    }

public void test2(){ assertEquals(3,3); }
```

Le test du test

```
public void test_appels_de_methodes_dans_test1(){
  List<String> resultat = byteCodeDeLaMethode(bytecode,"test1()V" );
  int empty=0, pop=0;
  for(String s : resultat){
    if(s.contains("java.util.Stack.empty()Z"))
      empty++;

    if(s.contains("java.util.Stack.pop()Ljava/lang/Object;"))
      pop++;
  }

  assertTrue("Au moins un appel de empty est requis ?",empty>0);
  assertTrue("Au moins un appel de pop est requis ?",pop>0);
}
```



Sommaire suite

- **Exécutions en ligne de commande**
  - **Une série de tests en ligne de commande**
  - **Mesure de la pertinence d'un test ? Ou de plusieurs classes de tests ?**
  - **Délai de garde ?**
    - **Le test ne s'arrête pas …**
  - **Sécurité ?**
    - **Le test, ou la méthode testée ne doivent pas supprimer tous les fichiers de votre répertoire…**

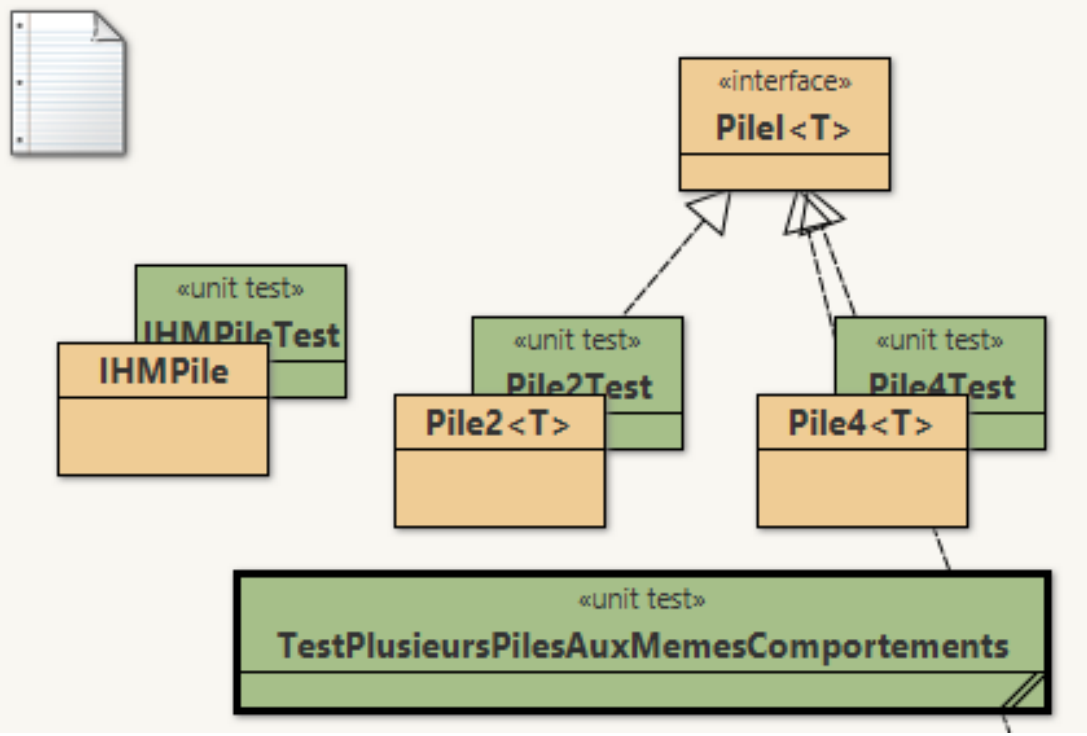
La suite : JUnit en ligne de commande

- **En ligne de commande**
- **Pertinence d'un jeu de tests ?**
  - **Couverture du code : utilisation de l'outil Cobertura**
- **Délai de garde,**
  - **L'une des méthodes testées devient sans fin…**
- **Stratégie de sécurité**
  - **Installation d'une stratégie de sécurité**
    - **Pas de Runtime.exec(" rmdir –s ");**

Exécution en ligne de commande

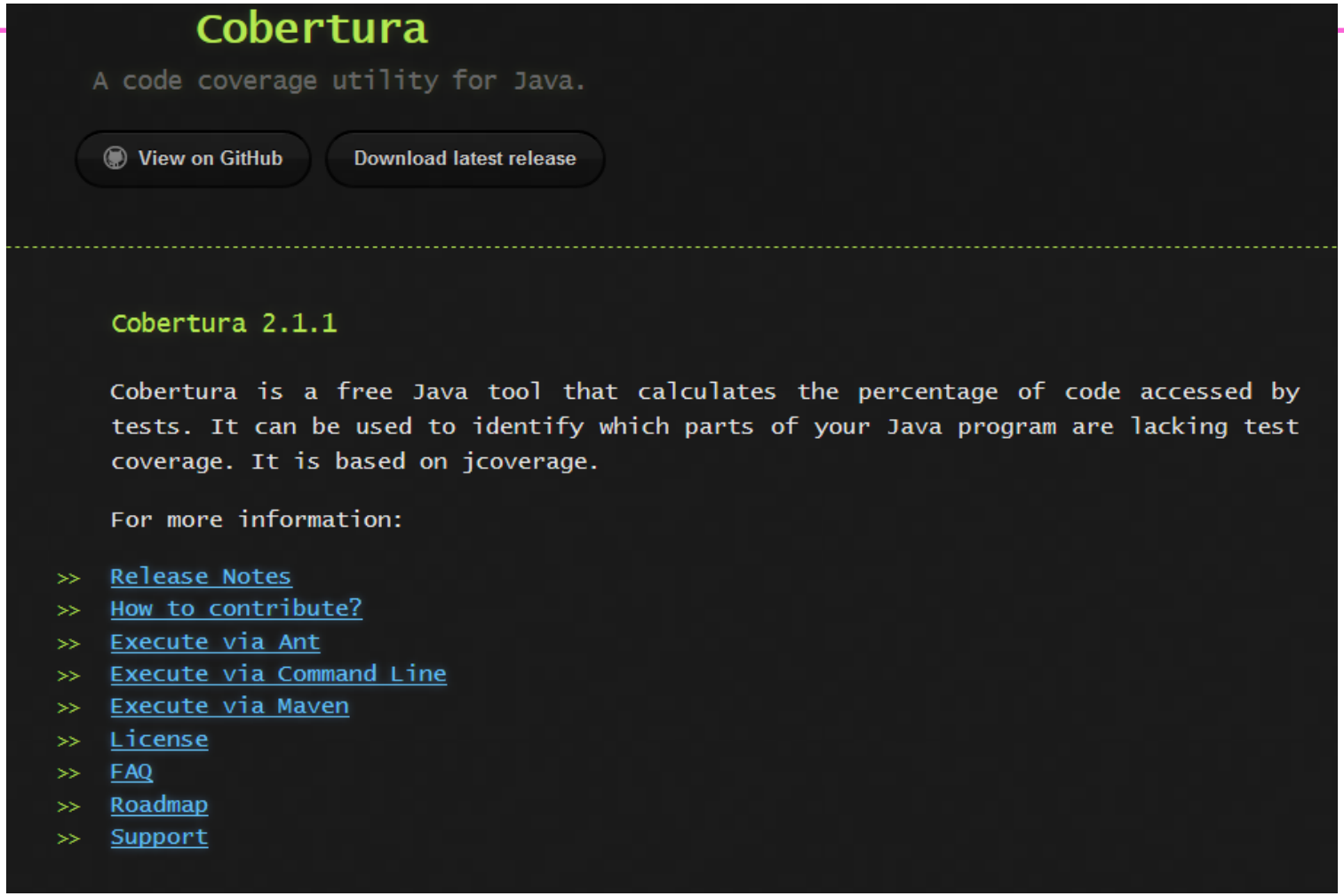
```
public class TestsTextUI extends junit.framework.TestCase{


  public static Test suite() {
    TestSuite suite= new TestSuite("Tests");
    suite.addTestSuite(Pile2Test.class);
    suite.addTestSuite(Pile4Test.class);
    suite.addTestSuite(TestPlusieursPilesAuxMemesComportements.class);
     suite.addTestSuite(IHMPileTest.class);

    return suite;
    }

  public static void main(String[] args){
    junit.textui.TestRunner.run(suite());
}
```

Les classes de tests



- **Mesures de la pertinence de classes de tests ?**

Cobertura : a free Java code coverage tool



http://cobertura.github.io/cobertura/

Il y en a d'autres… cf. Emma, Sonar…

Cobertura en ligne de commande, cf. cobertura.bat

```
rem création des classes intrumentées
call ./cobertura-2.1.1/cobertura-instrument.bat --basedir . --destination instr_pile -cp .;./junit.jar

rem exécution de la version intrumentée, génération des traces (*.ser)
java -cp ./cobertura-2.1.1/cobertura-2.1.1.jar;./cobertura-2.1.1/lib/slf4j-api-1.7.5.jar;./instr_pile;./junit.jar;. -Dnet.sourceforge.cobertura.datafile=cobertura1.ser junit.textui.TestRunner Pile2Test
java -cp ./cobertura-2.1.1/cobertura-2.1.1.jar;./cobertura-2.1.1/lib/slf4j-api-1.7.5.jar;./instr_pile;./junit.jar;. -Dnet.sourceforge.cobertura.datafile=cobertura2.ser junit.textui.TestRunner Pile4Test

rem concaténation des traces obtenues
call ./cobertura-2.1.1/cobertura-merge.bat --datafile cobertura.ser cobertura1.ser cobertura2.ser

rem génération des résultats, un rapport HTML
call ./cobertura-2.1.1/cobertura-report.bat --format html --datafile cobertura.ser --destination coverage .

rem Quelles sont les classes au faible taux de couverture ?
call ./cobertura-2.1.1/cobertura-check.bat --branch 50 --totalline 70

rem exécution du navigateur par défaut
start ./coverage/index.html
```

https://github.com/cobertura/cobertura/wiki/Command-Line-Reference

Cobertura

IHMPile	94 %	34/36
IHMPileTest	90 %	92/102
Main	0 %	0/5
Nombre	0 %	0/4
NombreTest	0 %	0/11
ParIntrospection	100 %	10/10
Pile2	100 %	29/29
Pile2Test	0 %	0/193
Pile2Test$1	N/A	N/A
Pile2Test$Fabrique	100 %	3/3
Pile2_RepOk_Af	57 %	22/38
Pile2_RepOk_AffTest	87 %	34/39
Pile2_RepOk_AffTest$1	N/A	N/A
Pile2_RepOk_AffTest$Fabrique	0 %	0/3
Pile4	86 %	38/44
Pile4$Maillon	100 %	6/6
Pile4Test	0 %	0/193

- **Couverture de tests faible pour certaines classes**
  - **Les tests sont donc à affiner …**

Pile4 en détail

Coverage Report - Pile4

Classes in this File	Line Coverage	Branch Coverage
Pile4	86 % 38/44	89 % 25/28
Pile4$Maillon	100 % 6/6	N/A N/A

```
1
2       public class Pile4<T> implements PileI<T>{
3         private Maillon stk;
4         /** la capacit? de la pile */
5         private int   capacite;
6         /** le nombre */
7         private int nombre;
8
9         /** Classe interne "statique" contenant chaque ?l?ment de la chaine
10         */
11        private class Maillon{
12          private T element;
13          private Maillon suivant;
14
15 191      public Maillon(T element, Maillon suivant){
16 191        this.element = element;
17 191        this.suivant = suivant;
18 191      }
19          public Maillon suivant(){
20 155        return this.suivant;
21        }
22        public T element(){
23 190        return this.element;
```

- **Sur ce test : bon taux couverture de cette classe**
  - **Attention : nous ne montrons pas l'absence d'erreurs**
    - **Tout au plus, notre taux de confiance croît …**

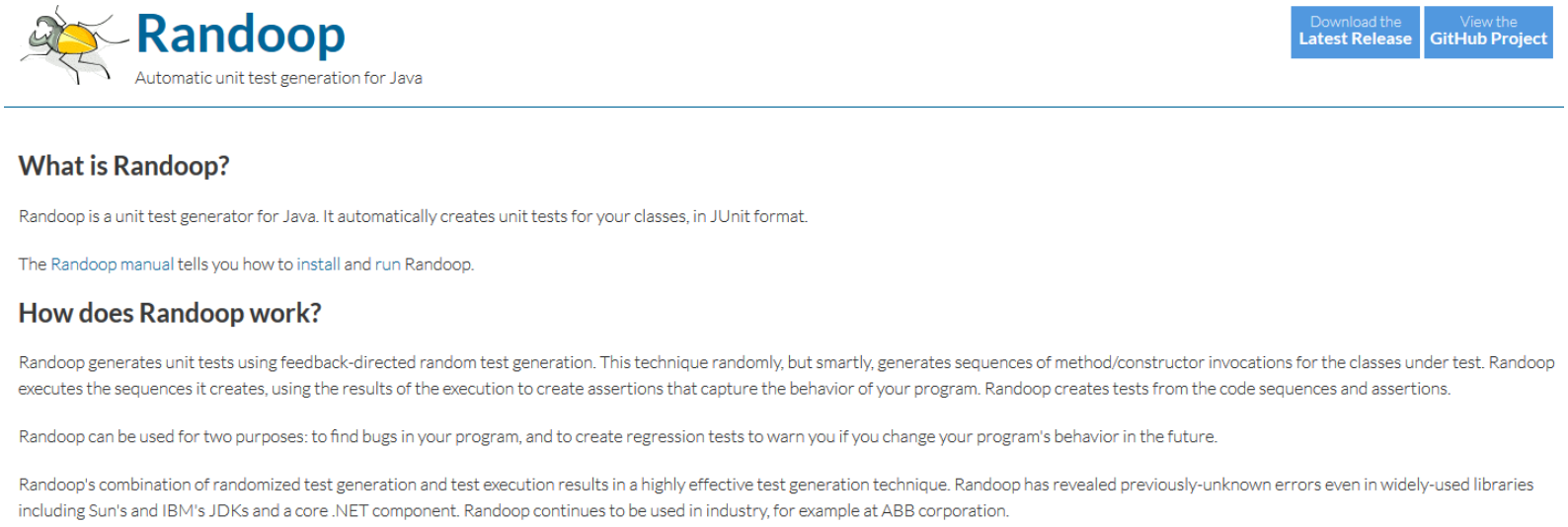
- **Conclusion hâtive ?**
  - **Taux de couverture : difficile de s'en passer, même si**

Pile4 une méthode oubliée ?

```
91    public boolean equals(Object o){
92    if (o instanceof Pile4){
93      Pile4 p = (Pile4)o;
94      if(p.capacite()==this.capacite() && p.taille()==this.taille()){
95        Maillon mp = p.stk;
96        for(Maillon m = stk; m!=null;m=m.suivant(),mp=mp.suivant())
97          if(!m.element().equals(mp.element())) return false;
98        return true;
99      }
100   }
101   return false;
102 }
103
104
105 private void traversée(Pile4 p, Maillon m){
106   try{
107     if(m!=null){
108       traversée(p,m.suivant());
109       p.empiler(m.element());
110     }
111   }catch(PilePleineException e){
112   }
113 }
114
```

- **??? De nouveaux Tests sont-ils à faire ???**

Génération automatique des tests …



- **Usage de Randoop**
  - **Cf. randoop.bat**
  - **Attention dans le projet bluej associé à ce support, les fichiers générés dépassent les 20000 lignes, ne tentez pas de les ouvrir depuis bluej (prenez notepad++)**
    - **Fichiers RegressionTestsX.java**
      - **Par défaut en JUnit4 transformés en JUnit3 pour ce support**

Cobertura avec Randoop

Coverage Report - Pile2

Classes in this File	Line Coverage	Branch Coverage
Pile2	96 % 28/29	90 % 18/20

```
1
2       import java.util.Stack;
3
4       public class Pile2<T> implements PileI<T>{
5         /** par d@l@gation : utilisation de la class Stack */
6         private Stack<T> stk;
7         /** la capcit@ de la pile */
8         private int   capacit@;
9
10        /** Cr@ation d'une pile.
11         * @param taille la taille de la pile, la taille doit @tre > 0
12         */
13 2583    public Pile2(int taille){
14 2583      if(taille<=0) taille = CAPACITE_PAR_DEFAUT;
15 2583      this.stk = new Stack<T>();
16 2583      this.capacit@ = taille;
17 2583    }
18
19        public Pile2(){
20 2101      this(PileI.CAPACITE_PAR_DEFAUT);
21 2101    }
22
23
24        public void empiler(T o) throws PilePleineException{
25 2349      if(estPleine()) throw new PilePleineException();
26 2349      stk.push(o);
27 2349    }
28
```

- **2583 appels du constructeur Pile2… discussions**

Un appel récursif, sans fin …

comment est-ce possible ?

```
public static Test suite() {
    TestSuite suite= new TestSuite("Tests");
    suite.addTestSuite(TestsPresentsSurLeSupport.class);
    suite.addTestSuite(TestPileI.class);
    suite.addTestSuite(Pile2Test.class);
    suite.addTestSuite(Pile4Test.class);
    suite.addTestSuite(TestPlusieursPilesAuxMemesComportements.class);
    suite.addTestSuite(IHMPileTest.class);

    return suite;
    }
```

```
public static void main(String[] args){
    WatchDog wd = new WatchDog(10*1000L);
   junit.textui.TestRunner.run(suite());
    wd.interrupt();
}
```

Classe WatchDog, prête à l'emploi

```
private static class WatchDog extends Thread{

   private long delay;

   public WatchDog(long delay){
     this.setPriority(Thread.MAX_PRIORITY);
     this.delay = delay;
     this.start();
   }

   public void run(){
     try{
       Thread.sleep(delay);
       Runtime.getRuntime().exit(1); // radical
     }catch(Exception e){
     }
   }
 }
```

Un peu de sécurité, au cas où

- **Interdire les Runtime.exec, par exemple**

- **java–cp .;junit.jar TestsTextUI**

```
public static void main(String[] args){
    System.setSecurityManager(new LocalSecurityManager());
    junit.textui.TestRunner.run(suite());
}
```

Security Suite, vérification, un test

```
public class TestRuntimeExec extends
  junit.framework.TestCase{


  public void testSecuriteEnPlace(){
    try{
      Process p = Runtime.getRuntime().exec("dir");
      fail("un \"security manager\" est-il en place ? ");
    }catch(Exception e){
      assertTrue(e instanceof SecurityException);
    }
  }
}
```

LocalSecurityManager
Au sein de TestsTextUI

```
private static class LocalSecurityManager
                              extends SecurityManager {

 public void    checkExec(String cmd) {
   throw new SecurityException("No exec permission : ");}

 public void    checkExit(int status) {
    throw new SecurityException("No Exit permission");}

 public void checkPermission(java.security.Permission perm) { }

 public void checkPermission(java.security.Permission perm,
                             Object context) { }
```

...

Conclusion