

Le Langage Java

1^{re} année

J. Beleho (bej) C. Leruste (clr) M. Codutti (mcd)
P. Bettens (pbt) F. Servais (srv) C. Leignel (clg)
D. Nabet (dna) J. Lechien (jlc)

Haute École de Bruxelles — École Supérieure d'Informatique

Année académique 2014 / 2015

Séance 8

Tester son code

- Présentation
- Les tests
- Plan de tests
- JUnit
- Conclusion

« The best performance improvement is the transition from the nonworking state to the working state. » J. Osterhout

Tests



Crédit photo

Présentation

Tous les programmes réels contiennent des **bugs** (erreurs, défauts)

- ▶ Parfois même **beaucoup**
- ▶ **Inacceptable**
 - **Inconfort** pour l'utilisateur
 - **Perte** de temps, d'argent, de données, de matériel
 - Voire **danger** pour la vie humaine

Présentation

Rappel des types d'erreurs

- ▶ À la **compilation**
- ▶ À l'**exécution**, le programme **s'arrête**
- ▶ À l'**exécution**, le programme fournit une **mauvaise réponse**

On pourrait aussi parler d'autres défauts

- ▶ Trop **lent**
- ▶ Trop gourmand en **mémoire**

« J'ai fait tourner le programme ;
il fonctionne très bien. »

Présentation

Pour produire un logiciel **sans bug** il faut

- ▶ Suivre une **méthodologie** éprouvée
 - Pour produire une première version avec peu de bugs (cf. *Analyse*)
- ▶ **Tester, tester** et ... **tester** encore !
 - Pour détecter ceux qui restent
 - Besoin d'outils pour nous aider
 - Le plus facile/rapide possible

Les tests


Il existe plusieurs sortes de tests : unitaires, d'intégration, fonctionnels, non-régression, ...

Nous nous intéressons aux tests **unitaires**

Pour en savoir plus : [http://fr.wikipedia.org/wiki/Test_\(informatique\)](http://fr.wikipedia.org/wiki/Test_(informatique))

Test unitaire :

Procédure permettant de tester le bon fonctionnement d'un module (une méthode)

A person dressed as a pirate, seen from the back, stands on a rocky shore looking out at the ocean. They are wearing a black pirate hat with a large red flower and white lace trim, and a red jacket with gold fringe. They are holding a piece of old, torn parchment that serves as a treasure map, which shows various islands, a compass rose, and a small skull and crossbones. The background is a vast body of water under a cloudy sky.

planning

Crédit photo

Plan de tests

- ▶ Planifier les tests
- ▶ Choisir les cas intéressants / judicieux
- ▶ Se baser sur les erreurs fréquentes

Expérience

Plan de tests - Exemple

Exemple

public static int max(**int** n1, **int** n2, **int** n3) ...

qui calcule la valeur maximale de trois nombres

- ▶ Que tester en plus du cas général?
 - Le maximum est la première/dernière valeur
 - Présence de nombres négatifs

Plan de tests - Exemple

Plan de tests de la méthode *max*

#	nombres	résultat	ce qui est testé
1	1, 3, 0	3	cas général
2	1, -3, -4	1	maximum au début
3	1, 3, 11	11	maximum à la fin
4	-1, -3, -4	-1	que des négatifs

Plan de tests

Quand tester ? Le plus **souvent** possible

- ▶ Erreur plus facile à identifier/corriger
- ▶ Idéalement après chaque méthode écrite

Que tester ? **Tout**

- ▶ Le nouveau code peut mettre en évidence un problème dans le code ancien (**régression**)

JUnit

Comment tester ?

- ▶ Pas à la main : intenable
- ▶ Besoin d'un outil **automatisé**



JUnit

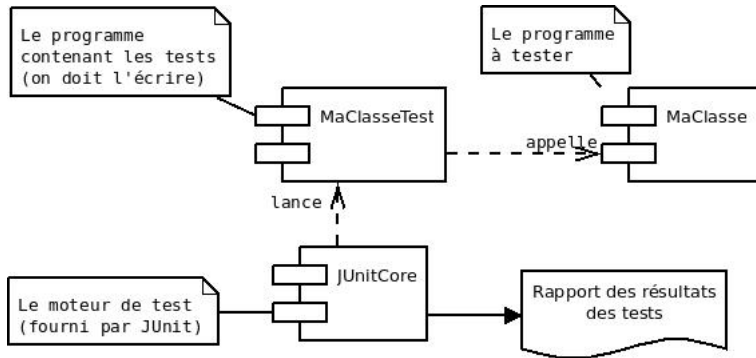
Crédit photo

JUnit - En pratique

La classe de test contient **une méthode** de test **par cas**

- ▶ Autonome (ne reçoit rien ne retourne rien)
- ▶ Contient des **affirmations**
 - Appel de la méthode à tester
 - **Comparaison** entre le résultat **attendu** et le résultat **obtenu**

JUnit



JUnit - En pratique

Exemple

```
@Test
public void max_cas1() {
    int n1 = 1, n2 = 3, n3 = 0;
    assertEquals( 3, MaClasse.max(n1, n2, n3) );
}
```

- ▶ Reconnu comme un test unitaire grâce à l'**annotation** `@Test`
- ▶ Pas de **static**
- ▶ `assertEquals` vérifie que les 2 valeurs sont identiques
- ▶ `assertTrue(val)`, `assertFalse(val)`, ...

JUnit - En pratique

Comment **lancer** les tests ?

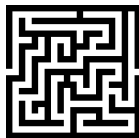
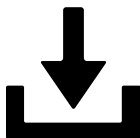
```
java org.junit.runner.JUnitCore mon.paquet.MaClasseTest
```

Résultats :

- ▶ Nombre de tests effectués / réussis
- ▶ Détail sur les tests ratés
 - Nom du test
 - Résultat obtenu comparé au résultat attendu

JUnit - En pratique

Où se trouve la classe `org.junit.runner.JUnitcore`?



JUnit - Exemple

Exemple

```
package be.heb.esi.java1;  
public class Outil {  
    public static int max(int n1, int n2, int n3) {  
        int max = 0;  
        if (n1 > max){  
            max = n1;  
        }  
        if (n2 > max){  
            max = n2;  
        }  
        if (n3 > max){  
            max = n3;  
        }  
        return max;  
    }  
}
```

JUnit - Exemple

Exemple (suite)

```
package be.heb.esi.java1 ;

import org.junit .Test ;
import static org.junit .Assert .* ;

public class OutilTest {
    @Test public void max_cas1() {
        assertEquals (3, Outil.max(1, 3, 0));
    }

    @Test public void max_cas4() {
        assertEquals (-1, Outil.max(-1, -3, -4));
    }

    // + plus les cas 2, 3 et 5
}
```

JUnit - Exemples

```
$ javac Outil*.java  
$ java org.junit.runner.JUnitCore  
    be.heb.esi.java1.OutilTest
```



```
JUnit version 4.11
...E
Time: 0,013
There was 1 failure:
1) max_cas4(be.heb.esi.java1.OutilTest)
java.lang.AssertionError: expected:<-1> but was:<0>
    at org.junit.Assert.fail(Assert.java:88)
    at org.junit.Assert.failNotEquals(Assert.java:743)
    at org.junit.Assert.assertEquals(Assert.java:118)
    at org.junit.Assert.assertEquals(Assert.java:555)
    at org.junit.Assert.assertEquals(Assert.java:542)
    at be.heb.esi.java1.OutilTest.max_cas4(OutilTest.java:13)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:483)
    at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:47)
```

java.lang.AssertionError: expected:<-1> but was:<0>

```
    at org.junit.runners.ParentRunner$5.run(ParentRunner.java:236)
    at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:63)
    at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:236)
    at org.junit.runners.ParentRunner.access$000(ParentRunner.java:53)
    at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:229)
    at org.junit.runners.ParentRunner.run(ParentRunner.java:309)
    at org.junit.runners.Suite.runChild(Suite.java:127)
    at org.junit.runners.Suite.runChild(Suite.java:26)
    at org.junit.runners.ParentRunner$3.run(ParentRunner.java:238)
    at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:63)
    at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:236)
    at org.junit.runners.ParentRunner.access$000(ParentRunner.java:53)
    at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:229)
    at org.junit.runners.ParentRunner.run(ParentRunner.java:309)
    at org.junit.runner.JUnit4Core.run(JUnit4Core.java:160)
    at org.junit.runner.JUnit4Core.run(JUnit4Core.java:138)
    at org.junit.runner.JUnit4Core.run(JUnit4Core.java:117)
    at org.junit.runner.JUnit4Core.runMain(JUnit4Core.java:96)
    at org.junit.runner.JUnit4Core.runMainAndExit(JUnit4Core.java:47)
    at org.junit.runner.JUnit4Core.main(JUnit4Core.java:40)
```

FAILURES!!!
Tests run: 2, Failures: 1

Que faire pour tester qu'une méthode lance bien une exception ?

JUnit - Tester les exceptions

Imaginons une méthode `sqrt` pour calculer la racine carrée d'un entier

- Hypothèse : elle doit lancer une exception en cas de paramètre négatif

```
public static int sqrt(int val) {  
    if (val < 0) {  
        throw new IllegalArgumentException(  
            "Pas de racine carrée pour un entier négatif");  
    }  
    // suite : calcul de la racine carrée  
}
```

JUnit - Tester les exceptions

Utilisation de l'**annotation**

```
@Test(expected=IllegalArgumentException.class)
public void sqrt_cas_négatif() {
    sqrt(-1);
}
```

- Le test est réussi si la méthode lance l'**exception** indiquée

« Faut-il *tout* tester ? »

Conclusion

« *Any program feature without an automated test simply doesn't exist.* »

Kent Beck in *Extreme Programming Explained*.

- ▶ Le site de JUnit : <http://www.junit.org>
- ▶ La Javadoc de JUnit :
<http://junit.sourceforge.net/javadoc>

Crédits

Ces slides sont le support pour la présentation orale de l'unité d'enseignement **DEV1-JAV** à HEB-ÉSI

Crédits

Les distributions Ubuntu et/ou debian
du système d'exploitation **GNU Linux**.

LaTeX/Beamer comme système d'édition.

Git et GitHub pour la gestion des versions et le suivi.

GNU make, rubber, pdfnup, ... pour les petites tâches.

Images et icônes

deviantart, flickr, The Noun Project ↴

