



JUnit

Jean-Marc Farinone

**Maître de Conférences
Conservatoire National des Arts et Métiers
CNAM Paris (France)**

Plan



- JUnit = ?, les tests
- Installer JUnit
- Une exemple d'illustration
- Syntaxe
- Exercice
- Bibliographie

JUnit = ?

- Version 4.10 depuis le 10 mai 2011
- JUnit est un framework pour écrire et exécuter des tests. Il fait partie de l'architecture de tests xUnit
- site de référence : www.junit.org
- JUnit propose :
 - des assertions qui vérifient les résultats à attendre du code développé
 - un environnement d'exécution de tests
- JUnit a été initialement écrit par Erich Gamma et Kent Beck
- Peut être utilisé pour faire des tests unitaires mais aussi pour des "hiérarchies" de tests
- JUnit version 4.x utilise les annotations (donc Java 1.5) ce qui n'est pas le cas avec les versions 3.x

Quand tester ?

- Construire les tests pendant (ou avant) le code (cf eXtreme Programming). Voir à <http://junit.sourceforge.net/doc/testinfected/testing.htm> utilisant JUnit 3
 - "During Development- When you need to add new functionality to the system, write the tests first. Then, you will be done developing when the test runs.
 - During Debugging- When someone discovers a defect in your code, first write a test that will succeed if the code is working. Then debug until the test succeeds."

Comment installer et désinstaller JUnit ?

- Voir à :

`http://junit.sourceforge.net/doc/faq/faq.htm#starte
d_2` pour installation et

`http://junit.sourceforge.net/doc/faq/faq.htm#starte
d_3` pour désinstallation

- En gros, pour installer JUnit, il suffit de faire repérer `junitXXX.jar` par la variable `CLASSPATH` (et PAS le mettre dans `jre/lib/ext` grr!!)

Un exemple : Calculator, une classe à tester

```
package calc;

public class Calculator {
    private static int result; // Le "registre" de la calculette

    public void add(int n) { result = result + n; }

    public void subtract(int n) {
        result = result - 1; // Bug : devrait être result = result - n
    }

    public void multiply(int n) {} // Non implémenté

    public void divide(int n) { result = result / n; }

    public void squareRoot(int n) {
        for (; ; ) ; // Arg : une boucle infinie
    }

    public void square(int n) {
        result = n * n;
    }

    public void clear() { result = 0; }

    public int getResult() { return result; }
}
```

Une classe qui teste

Calculator (1/2)

```
package junit4;

import calc.Calculator;
import org.junit.Before;
import org.junit.Ignore;
import org.junit.Test;
import static org.junit.Assert.*;

public class CalculatorTest {
    private static Calculator calculator =
        new Calculator();

    @Before
    public void clearCalculator() {
        calculator.clear();
    }

    @Test
    public void add() {
        calculator.add(1);
        calculator.add(1);
        assertEquals(calculator.getResult(), 2);
    }
}
```

- utilisation de la syntaxe Java 1.5
 - import static
 - annotation
- utilisation des classes et méthodes du framework JUnit
- utilisation du package org.junit (pour JUnit 4)

Une classe qui teste

Calculator (2/2)

```
@Test
public void subtract() {
    calculator.add(10);
    calculator.subtract(2);
    assertEquals(calculator.getResult(), 8);
}

@Test
public void divide() {
    calculator.add(8);
    calculator.divide(2);
    assertEquals(calculator.getResult(), 5);
}

@Test(expected = ArithmeticException.class)
public void divideByZero() {
    calculator.divide(0);
}

@Ignore("test à ignorer pour l'instant")
@Test
public void multiply() {
    calculator.add(10);
    calculator.multiply(10);
    assertEquals(calculator.getResult(), 100);
}
```

- utilisation de `assertEquals()` du framework JUnit
- Test des exceptions qui doivent être levées

Méthodes `assertXXX()`

- Les diverses méthodes statiques `assertXXX()` de la classe `org.junit.Assert` du framework JUnit dont `assertEquals()` sont indiquées à :
http://junit.sourceforge.net/javadoc_40/org/junit/Assert.html

Compilation, exécution et démonstration

- démonstration dans le répertoire de ce cours

- compileAll.bat

```
set CLASSPATH=.;cheminAbsoluOuRelatifQuiMeneA\junitXXX.jar
javac -d ../classes calc/*.java junit4/*.java
```

- runCalculatorTest.bat

```
set CLASSPATH=.;cheminAbsoluOuRelatifQuiMeneA\junitXXX.jar
java -ea org.junit.runner.JUnitCore junit4.CalculatorTest
```

- L'exécution est lancée dans "l'environnement"

```
org.junit.runner.JUnitCore
```

Résultat du test

■ La sortie est :

```
JUnit version 4.4
..E.E.I
Time: 0,015
There were 2 failures:
1) subtract(junit4.CalculatorTest)
java.lang.AssertionError: expected:<9> but was:<8>
    at org.junit.Assert.fail(Assert.java:74)
...
    at junit4.CalculatorTest.subtract(CalculatorTest.java:29)
...
2) divide(junit4.CalculatorTest)
java.lang.AssertionError
    at junit4.CalculatorTest.divide(CalculatorTest.java:36)
...

FAILURES!!!
Tests run: 4, Failures: 2
```

Les méthodes pour JUnit 4.x

- Avec JUnit4.x, on utilise les annotations sur les méthodes (à la place des conventions de noms des méthodes) :
 - `@Test` indique une méthode de test (au lieu de préfixer les méthodes de test par `test` : JUnit 3.x)
 - `@Before` indique une méthode d'initialisation (au lieu d'une méthode `setUp()` JUnit 3.x)
 - `@After` indique une méthode à exécuter après une méthode de test (au lieu d'une méthode `tearDown()` JUnit 3.x)
- On importe le paquetage `org.junit`
- Plus besoin d'hériter de la classe `TestCase` (JUnit 3.x)
- Plus besoin de créer une classe `TestSuite` (JUnit 3.x)



Rappel Java 1.4 : les assertions

Les assertions (1/2)

- Source :

`http://java.sun.com/docs/books/jls/third_edition/html/statements.html#14.10 ou encore`

`http://java.sun.com/j2se/1.5.0/docs/guide/language/assert.html`

- De la forme :

`assert Expression1 : Expression2 ;`

où `Expression1` est un `boolean` ou un `Boolean`, `Expression2` est éventuellement absente

- Si `Expression1` est `true`, le reste de l'assertion est ignoré.

Les assertions (2/2)

- Si `Expression1` est `false`,
- 1er cas : `Expression2` est absente
 - Une instance `AssertionError` est créée et cette exception `AssertionError` est levée
- 2ieme cas : `Expression2` est présente `Expression2` est évaluée, convertie en `String`, et constitue le message de l'exception `AssertionError` qui est construit et levée.

Utilisation des assertions



- Il faut lancer le code par :

```
java -ea ClassePrinc
```

pour utiliser les assertions sinon elles sont ignorées



Fin du rappel Java 1.4 : les assertions

Syntaxe JUnit 4 (1/2)



- Les méthodes qui testent sont annotées par `@Test`
- Les méthodes annotées par `@Before` sont lancées avant tout test (i.e. méthode annotée par `@Test`)
- Les méthodes annotées par `@After` sont lancées après tout test (i.e. méthode annotée par `@Test`)
- Une classe de test doit avoir au moins une méthode annotée `@Test`
- On peut utiliser les assertions auquel cas, exécuter le programme avec l'option `-ea` de `java`

Syntaxe JUnit 4 (2/2)

- L'annotation `@Test` peut avoir des paramètres indiquant l'exception qui doit être levée. Si cette exception n'est pas levée ou qu'une autre exception est levée, le test échoue. (cf. `test divideByZero()`)
- La méthode `multiply(int n)` de la classe `Calculator` n'est pas encore implémentée mais lorsqu'elle le sera on voudra la tester. En indiquant `@Ignore` avant (ou après) `@Test`, on précise que pour l'instant le test est ignoré. Lors de l'exécution des tests, les tests ignorés sont indiqués (ainsi on ne les oublie pas)

Fixture



- Les méthodes de test manipulent un (ou plusieurs) objets de la classe à tester. Deux méthodes de test utilisent des objets de la classe à tester (et d'autres classes) distincts
- Une fixture est un ensemble d'objets propre à l'exécution d'une méthode de test
- Ces objets sont, en général, construits dans les méthodes annotées `@Before`
- Les désallocations faites par une fixture sont en général faites dans les méthodes annotées `@After`

Rappel de l'exécution (1/2)

■ Dans le code ci contre, un ordre des appels peut être est :

setUp,

testEmptyCollection,

setUp,

testOneItemCollection

■ L'ordre de l'exécution des deux méthodes de tests peut changé

```
import org.junit.*;
import static org.junit.Assert.*;
import java.util.*;

public class SimpleTest {

    private Collection<Object> collection;

    @Before
    public void setUp() {
        collection = new ArrayList<Object>();
    }

    @Test
    public void testEmptyCollection() {
        assertTrue(collection.isEmpty());
    }

    @Test
    public void testOneItemCollection() {
        collection.add("itemA");
        assertEquals(1, collection.size());
    }
}
```

Rappel de l'exécution (2/2)



- "JUnit assumes that all test methods can be performed in an arbitrary order. Therefore tests should not depend other tests"

- source :

`http://www.vogella.de/articles/JUnit/article.html#junitclipse`

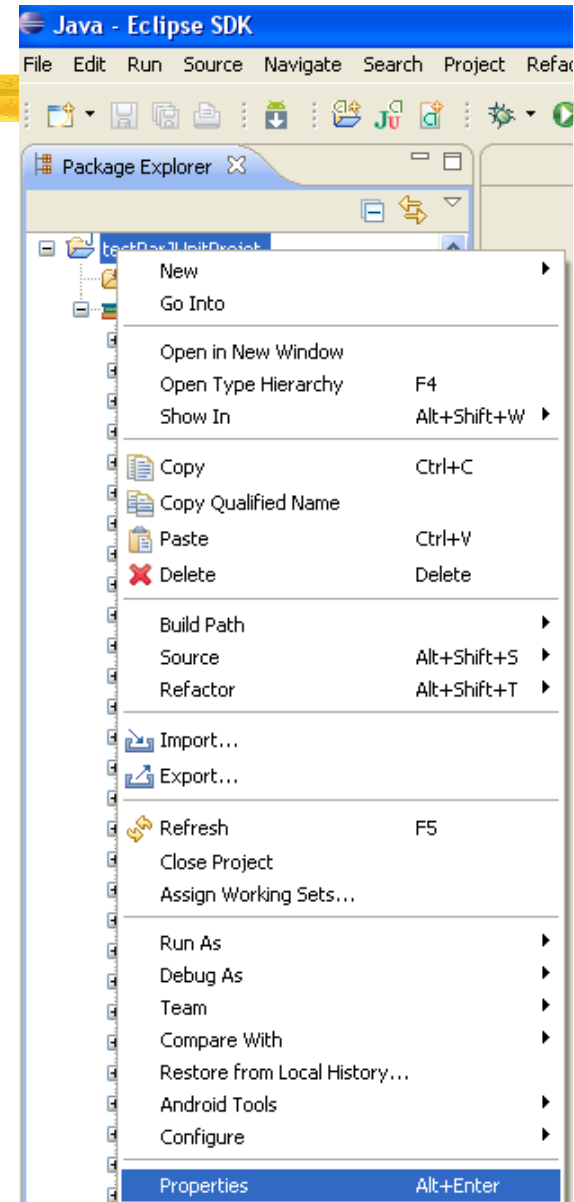
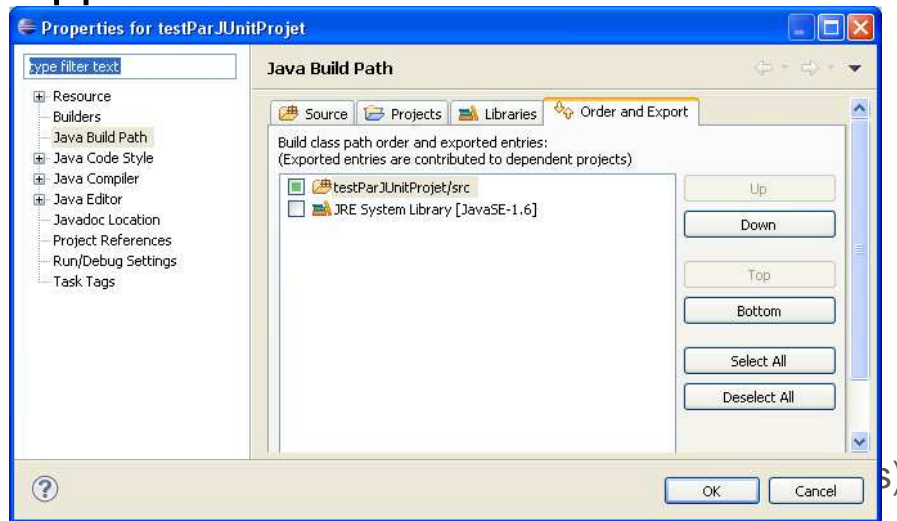
JUnit et Eclipse



- Plusieurs étapes :
 - Ajouter `junitXXX.jar` aux bibliothèques `.jar` déjà existantes (si ce n'est déjà fait) pour le projet
 - Ecrire une classe à tester
 - Ecrire un testeur de cette classe
 - Exécuter le test

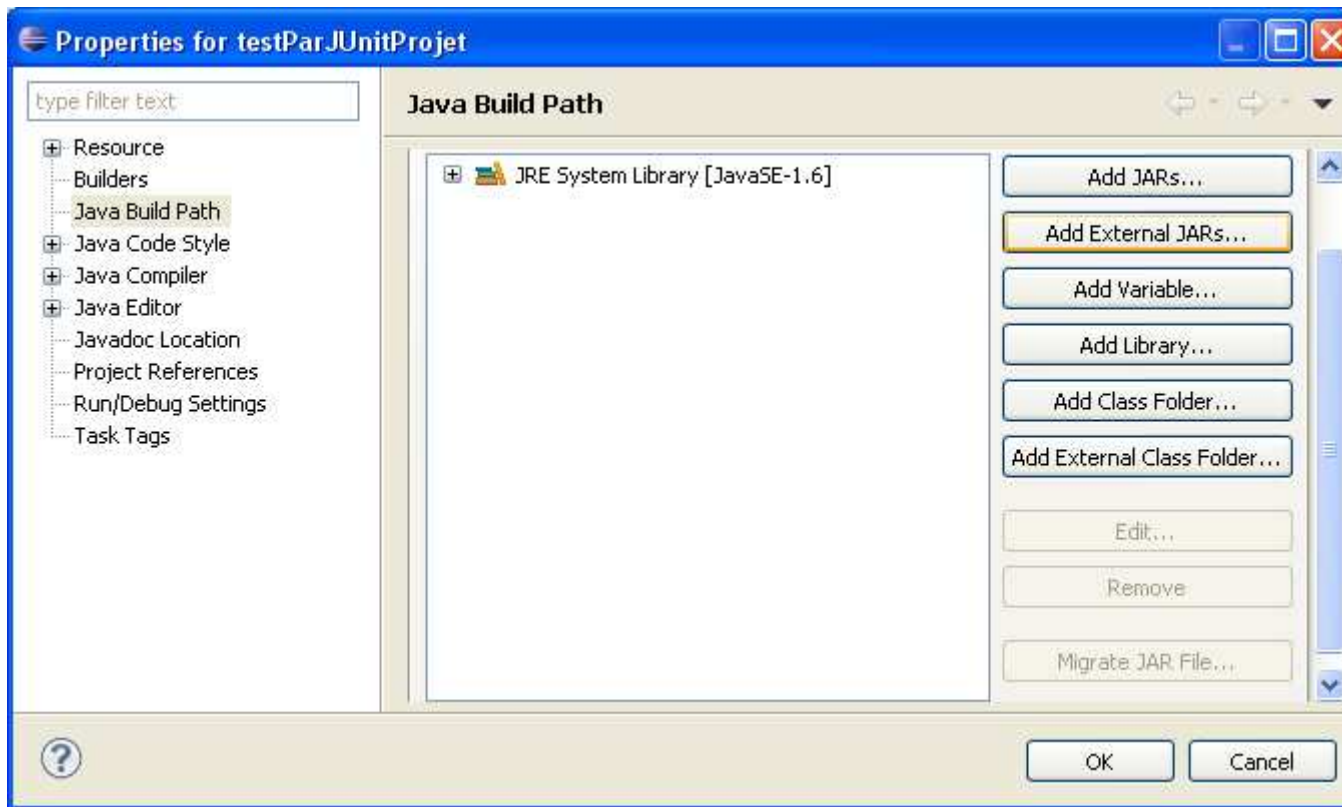
Ajouter junitXXX.jar au projet (1/4)

- Il faut d'abord, si vous ne l'avez pas fait par ailleurs, ajouter le junitXXX.jar comme bibliothèque de classes additionnelle à votre projet
- Pour cela, sélectionner le projet, cliquez droit, puis cliquez Properties. Sélectionner Java Build Path. Apparaît la fenêtre :



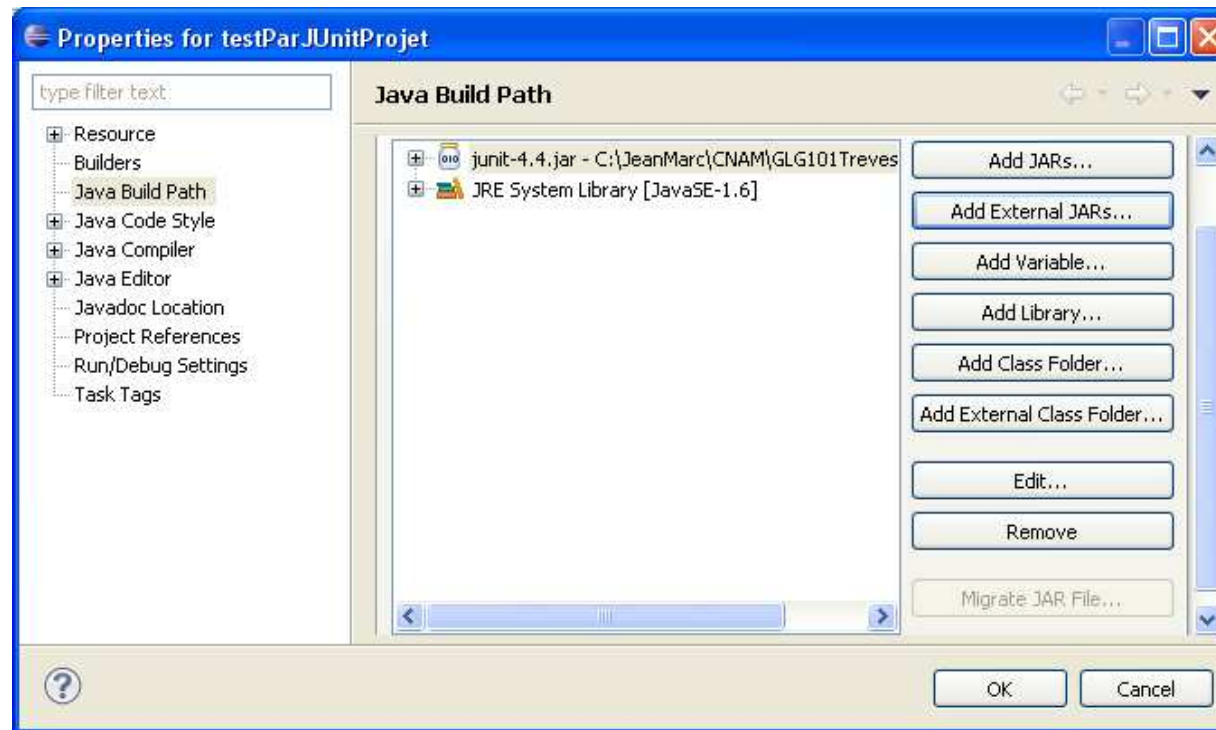
Ajouter junitXXX.jar au projet (2/4)

- Sélectionnez l'onglet "Libraries",
- Cliquez le bouton "Add External JARs..."



Ajouter junitXXX.jar au projet (3/4)

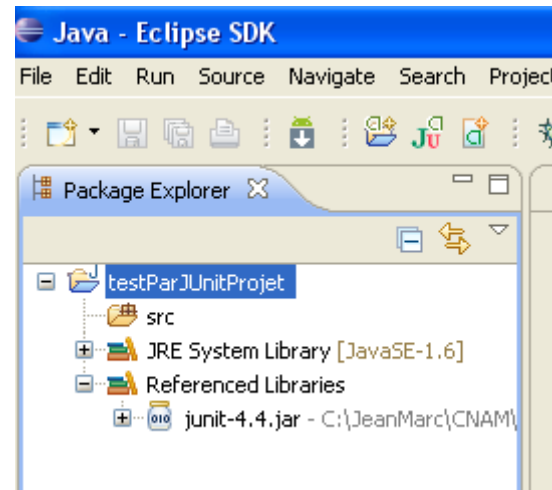
- Parcourez votre système de fichiers pour trouver le junitXXX.jar.
- Après l'avoir sélectionné, il apparaît comme bibliothèque supplémentaire :



- C'est fini !

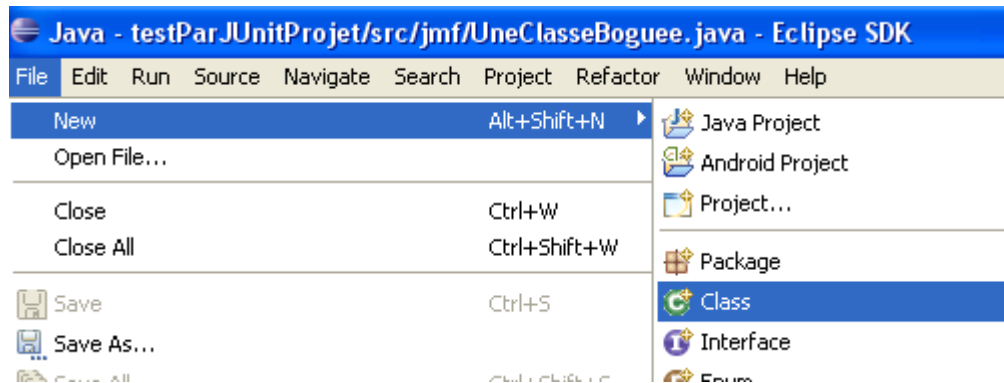
Ajouter junitXXX.jar au projet (4/4)

- Vérification : Dans "Referenced Libraries" du "Package Explorer" apparaît le junitXXX.jar.



Créer une classe à tester

- Construire un projet puis une classe dans eclipse (File | New | Class)



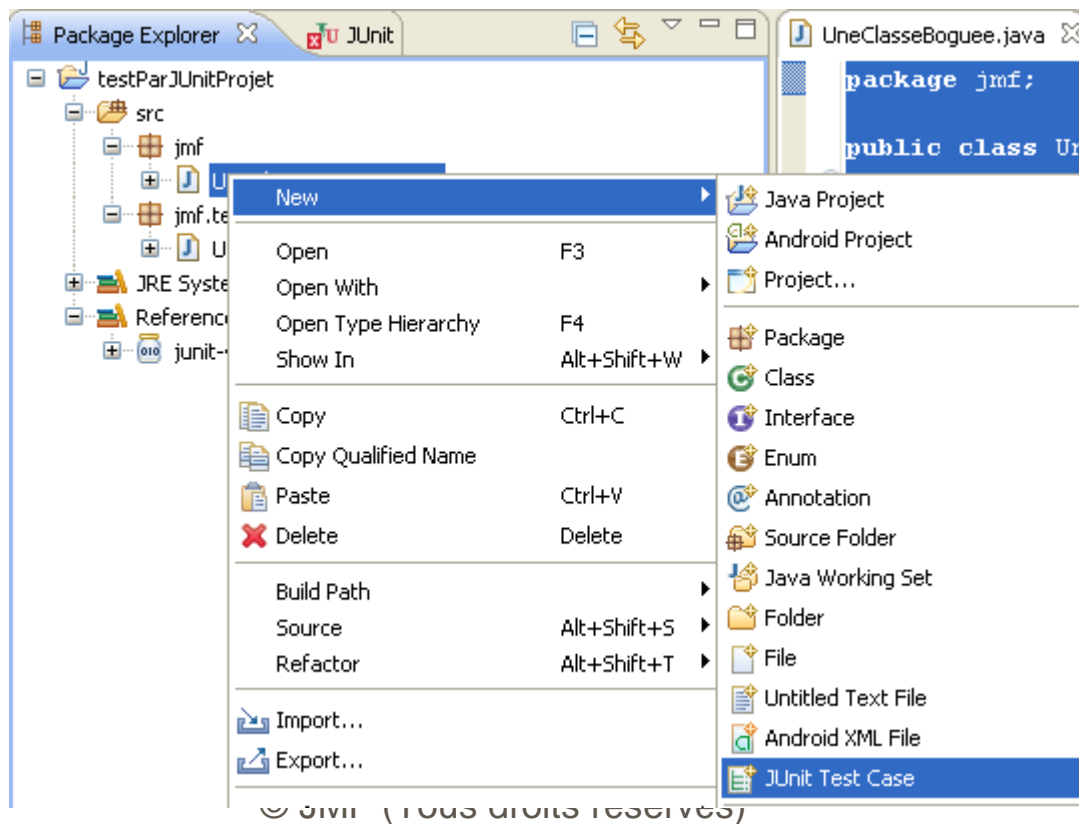
- Ecrire cette classe : par exemple :

```
package jmf;

public class UneClasseBogquee {
    public int multiply(int x, int y) {
        return x / y;
    }
}
```

Créer une classe de test (1/4)

- Sélectionner la classe à tester.
- Cliquez droit et sélectionner New | JUnit Test Case



Créer une classe de test (2/4)

- Dans la fenêtre "New JUnit Test Case", indiquer un nom de paquetage et un nom de classe de test. Cliquez "Next >"

New JUnit Test Case

JUnit Test Case

Select the name of the new JUnit test case. You have the options to specify the class under test and on the next page, to select methods to be tested.

New JUnit 3 test New JUnit 4 test

Source folder: testParJUnitProjet/src Browse...

Package: jmf Browse...

Name: UneClasseBogqueeTest

Superclass: java.lang.Object Browse...

Which method stubs would you like to create?

setUpBeforeClass() tearDownAfterClass()
 setUp() tearDown()
 constructor

Do you want to add comments? (Configure templates and default value [here](#))

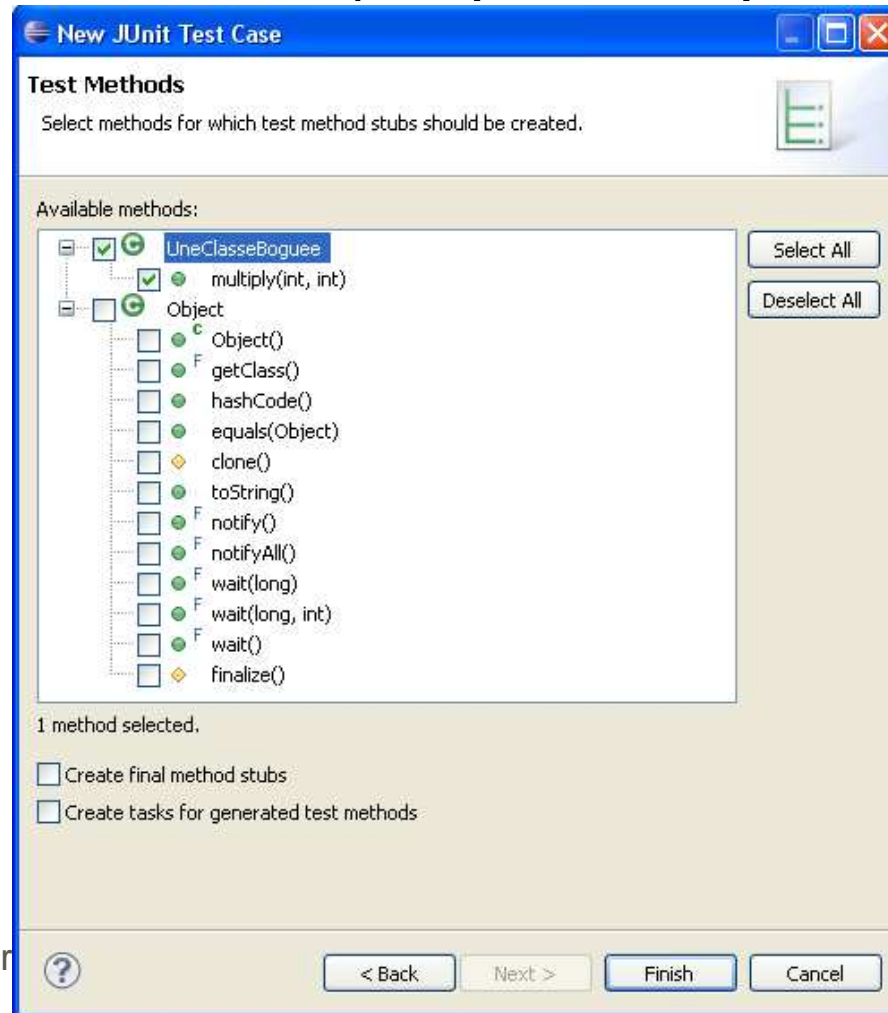
Generate comments

Class under test: jmf.UneClasseBogquee Browse...

< Back Next > Finish Cancel

Créer une classe de test (3/4)

- Dans la fenêtre de sous titre Test Methods, indiquez (en cochant) les méthodes à tester. Cliquez Finish



Créer une classe de test (4/4)

- La classe de test apparaît. Compléter la par du code approprié :

```
package jmf.test;

import static org.junit.Assert.assertEquals;
import jmf.UnclasseBogquee;

import org.junit.Test;

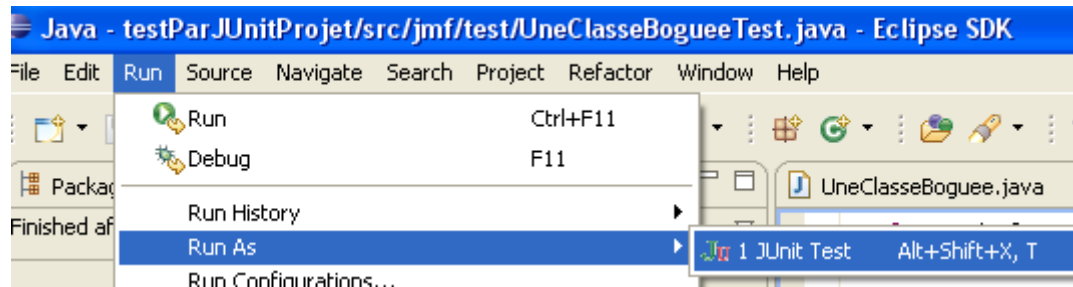
public class UnclasseBogqueeTest {

    @Test
    public void testMultiply() {
        UnclasseBogquee tester = new UnclasseBogquee();
        assertEquals("Result", 50, tester.multiply(10, 5));
    }

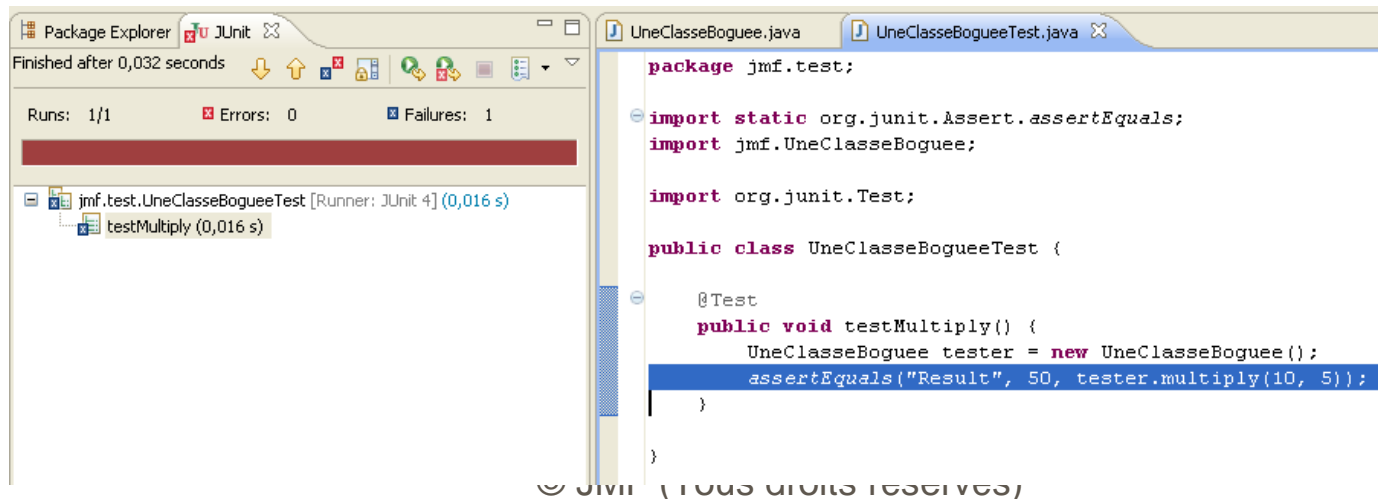
}
```


Exécuter le test

- Lancer l'exécution du test par Run | Run As | JUnit Test



- Le résultat du test apparaît dans l'onglet JUnit



Exercice sur JUnit 4

- Ecrire des tests pour des classes modélisant des sommes d'argent et un porte feuille

JUnit 3.x

- La version JUnit 3.x est encore beaucoup utilisée. Elle diffère de JUnit 4.x dans sa syntaxe, peu dans ses concepts
- On charge JUnit 3.x à partir de `http://sourceforge.net/projects/junit/files/junit/`

Contraintes syntaxiques de JUnit 3.x

- Les classes amenées par JUnit 3.X sont dans le paquetage `junit.framework` (au lieu de `org.junit` en JUnit 4.x)
- Une classe de test pour JUnit 3.x doit dériver de `junit.framework.TestCase` (pas de telle contrainte en 4.x)
- Il ne peut y avoir qu'une seule méthode d'initialisation et celle-ci est la méthode `protected void setUp() throws java.lang.Exception` (plusieurs possibles en 4.x préfixées par `@Before`)
- Il ne peut y avoir qu'une seule méthode de traitement de fin et celle-ci est la méthode `protected void tearDown() throws java.lang.Exception` (plusieurs possibles en 4.x préfixées par `@Before`)

Les méthodes de test en JUnit 3.x

- Les méthodes sont des méthodes de test si et seulement si leur nom commence par `test`
- Plus précisément : "The method name has to be prefixed with 'test', it must return `void`, and it must have no parameters (e.g. `public void testDivide()`). A test method that doesn't follow this naming convention is simply ignored by the framework and no exception is thrown, indicating a mistake has been made."
- source : Antonio Goncalves à <http://www.devx.com/Java/Article/31983/1954?pf=true>

Tester la calculette en JUnit

3.x (1/2)

```
package junit3;

import calc.Calculator;
import junit.framework.TestCase;

public class CalculatorTest extends TestCase {
    private static Calculator calculator = new Calculator();

    @Override
    protected void setUp() {
        calculator.clear();
    }

    public void testAdd() {
        calculator.add(1);
        calculator.add(1);
        assertEquals(calculator.getResult(), 2);
    }

    public void testSubstract() {
        calculator.add(10);
        calculator.substract(2);
        assertEquals(calculator.getResult(), 8);
    }
}
```

Tester la calculette en JUnit

3.x (2/2)

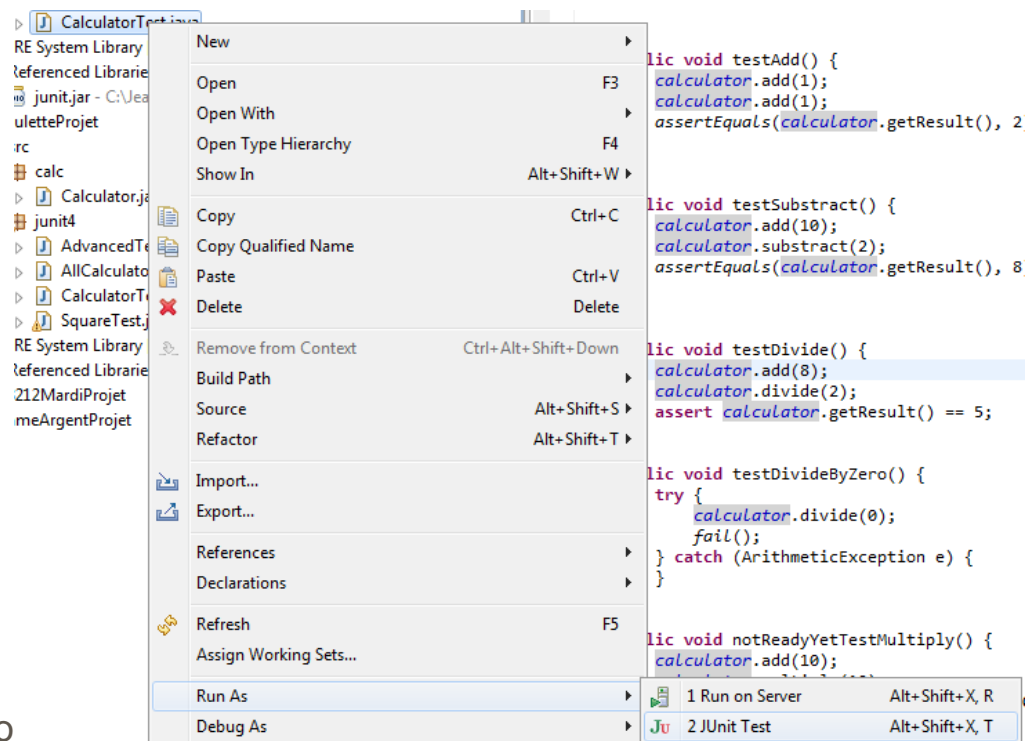
```
public void testDivide() {
    calculator.add(8);
    calculator.divide(2);
    assert calculator.getResult() == 5;
}

public void testDivideByZero() {
    try {
        calculator.divide(0);
        fail();
    } catch (ArithmeticException e) {
    }
}

public void notReadyYetTestMultiply() {
    calculator.add(10);
    calculator.multiply(10);
    assertEquals(calculator.getResult(), 100);
}
}
```

Exécution du test de la calculette en JUnit 3.x

- Sous Eclipse, il faut évidemment ajouter le `.jar` JUnit 3.x à l'environnement du projet : sélectionner le projet, clic droit, Properties, Java Build Path, onglet Librairies, bouton Add External JARs...
- Ce `.jar` s'appelle souvent `junit.jar`
- En ligne de commande repérer ce `junit.jar` avec la variable `CLASSPATH`
- Sélectionner la classe de test, clic droit puis Run As | 2 JUnit Test
- On obtient évidemment un résultat similaire à JUnit 4



Exercice sur JUnit 3

- Ecrire des tests pour une classe modélisant les polynomes du second degré
- Etudier le fonctionnement de JUnit

Retour sur JUnit 4 (1/2)

- Voici une nouvelle classe de test pour la calculette :

```
package junit4;

import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;

import calc.Calculator;

public class AdvancedTest {

    private static Calculator calculator;

    @BeforeClass
    public static void switchOnCalculator() {
        System.out.println("Switch on calculator");
        calculator = new Calculator();
        calculator.clear();
    }

    @AfterClass
    public static void switchOffCalculator() {
        System.out.println("Fin test calculette");
        calculator = null;
    }

    @Before
    public void clearCalculator() {
        System.out.println("Clear calculator");
        calculator.clear();
    }
}
```

Retour sur JUnit 4 (2/2)

```
@Test(timeout = 1000)
public void squareRoot() {
    calculator.squareRoot(2);
}

@Test
public void square2() {
    calculator.square(2);
    assertEquals(4, calculator.getResult());
}

@Test
public void square4() {
    calculator.square(4);
    assertEquals(16, calculator.getResult());
}

@Test
public void square5() {
    calculator.square(5);
    assertEquals(25, calculator.getResult());
}
}
```

Nouvelles annotations en JUnit 4

- `@BeforeClass` indique une méthode qui est exécutée une et une seule fois juste après le chargement de la classe de test et avant toute méthode de test (annoté `@Test`). Il y a au plus une telle méthode dans une classe de test. Cette méthode doit être `static public void`
- `@AfterClass` indique une méthode qui est exécutée une et une seule fois juste après toutes les méthodes de test (annoté `@Test`). Il y a au plus une telle méthode dans une classe de test. Cette méthode doit être `static public void`
- `@Test(timeout = 1000)` indique une méthode de test qui doit être exécutée en au plus 1000 millisecondes

Tests paramétrés en JUnit 4

(1/3)

- On a utilisé 3 méthodes de test pour tester les carrés de 2, 4 et 5
- On a une syntaxe plus efficace pour lancer une succession de test avec des paramètres distincts, chaque test devant renvoyer un résultat indiqué

```
package junit4;
import static org.junit.Assert.assertEquals;

import java.util.Arrays;
import java.util.Collection;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;
import org.junit.runners.Parameterized.Parameters;

import calc.Calculator;

@RunWith(Parameterized.class)
public class SquareTest {
    private static Calculator calculator = new Calculator();
    private int param;
    private int result;
```

Tests paramétrés en JUnit 4

(2/3)

```
@Parameters
public static Collection data() {
    return Arrays.asList(new Object[][]{
        {0, 0},
        {1, 1},
        {2, 4},
        {4, 16}, // OK : 42 = 16
        {5, 25},
        {6, 36},
        {7, 48} // NOK : 72 = 49 not 48
    });
}

public SquareTest(int param, int result) {
    this.param = param;
    this.result = result;
}

@Test
public void square() {
    calculator.square(param);
    assertEquals(result, calculator.getResult());
}
}
```


- Il faut d'abord indiquer que la classe utilise l'environnement d'exécution avec tenant compte de paramètres :

```
@RunWith(Parameterized.class)
```

- Pour indiquer les paramètres, la classe doit avoir une méthode

Tests paramétrés en JUnit 4

(3/3)



- Pour indiquer les paramètres, la classe doit avoir une méthode annotée `@Parameters`. Cette méthode doit être signée `public static Collection`, donc retourner une `Collection`
- La classe doit avoir un constructeur `public` qui prend les paramètres et le résultat pour chaque exécution de la méthode de test annotée `@Test`

Suite de tests en JUnit 4

- En général, tous les tests ne sont pas mis dans une seule classe
- On peut écrire une classe qui indique de faire les tests de plusieurs classes de tests

- Par exemple :

```
package junit4;

import org.junit.runner.RunWith;
import org.junit.runners.Suite;

@RunWith(Suite.class)
@Suite.SuiteClasses({
    CalculatorTest.class,
    AdvancedTest.class,
    SquareTest.class
})
public class AllCalculatorTests {
}
```

- Il faut, pour cela, écrire une classe (AllCalculatorTests) annotée par `@RunWith(Suite.class)` indiquant qu'elle lance une suite de test suivi de l'annotation `@Suite.SuiteClasses({ CalculatorTest.class, AdvancedTest.class, SquareTest.class })` indiquant les classes de test à utiliser

Bibliographie

- Le site originel de JUnit : <http://www.junit.org/>
- L'API JUnit
http://junit.sourceforge.net/javadoc_40/index.html
- La FAQ pour JUnit
<http://junit.sourceforge.net/doc/faq/faq.htm>
- Des tutoriaux pour JUnit version 4 (donc avec les annotations) :
<http://www.junit.org/taxonomy/term/12>,
<http://www.devx.com/Java/Article/31983> de Antonio Goncalves (qui compare JUnit 3 et 4). Son article complet à <http://www.devx.com/Java/Article/31983/1954?pf=true>.
Cet article a beaucoup inspiré ce support de cours
- Cours de Pascal Graffion (merci Pascal)
- Présentation de JUnit par Wikipedia :
<http://en.wikipedia.org/wiki/JUnit>



Fin