

# JUnit 3.8 / JUnit 4

P. Graffion

# Objectifs

- **Framework de tests écrit pour faciliter :**
- **l'écriture de tests**
  - tests unitaires
  - tests de recette
- **l'exécution de ces tests**
- **l'exploitation de ces tests**

# Origine

- Développement piloté par les tests  
**(Test Driven Development)**
- Framework Junit écrit en Java par :
  - Kent Beck (**Extreme programming**)
  - Erich Gamma (**Design patterns**)
  - (**[www.junit.org](http://www.junit.org)**)
- Désormais décliné dans tous les langages!

# Pourquoi tester?

- Pour gagner du temps!
- C'est plus efficace que de débugger
- Permet de valider le logiciel
- Permet de valider les remaniements de code
- Permet de détecter les régressions

**Test Driven Development (TDD) :**  
**on écrit les tests avant de coder l'application!**

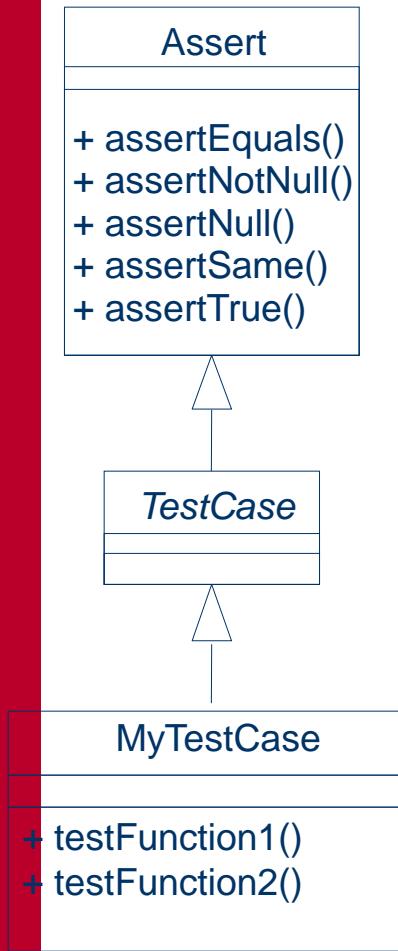
# JUnit 3.8

<http://junit.sourceforge.net/junit3.8.1>

<http://junit.sourceforge.net/junit3.8.1/javadoc>



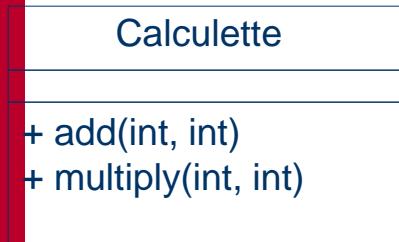
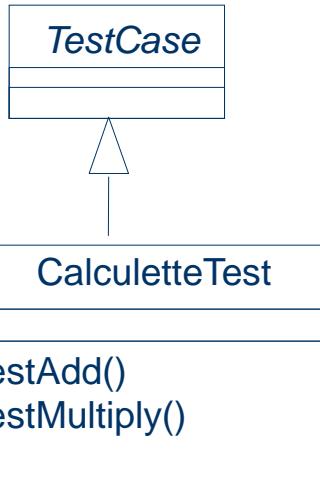
# TestCase



**Ecrire une classe de test consiste à :**

- **hériter de la classe TestCase**
- **implémenter plusieurs méthodes nommées `test<f>()`**
- **écrire des assertions :**
  - `assertTrue(1 > 0)`
  - `assertEquals(7, 3+4)`

# Premier exemple



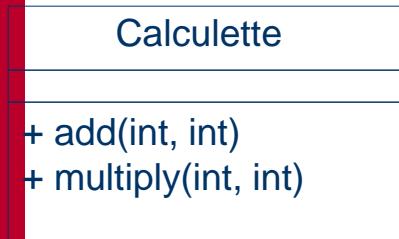
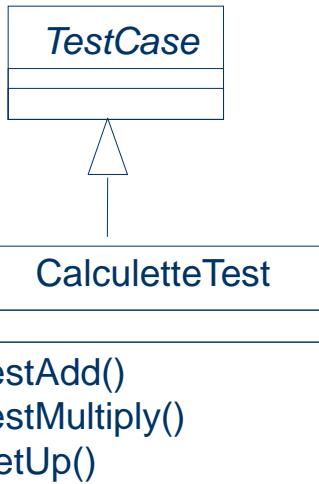
```
import junit.framework.TestCase;
public class CalculetteTest extends TestCase
{
    public void testAdd() {
        Calculette c = new Calculette();
        int expected = 7;
        int actual = c.add(3, 4);
        assertEquals(expected, actual);
        assertEquals(1, c.add(3, -2));
        ...
    }

    public void testMultiply() { ... }
}
```

# Fixture

- **Ensemble des objets utilisés dans plusieurs fonctions d'une même classe de test**
- **Chaque objet utilisé est alors**
  - déclaré en tant que variable d'instance de la classe de test
  - initialisé dans la méthode `setUp()`
  - éventuellement libéré dans la méthode `tearDown()`

# Exemple avec fixture



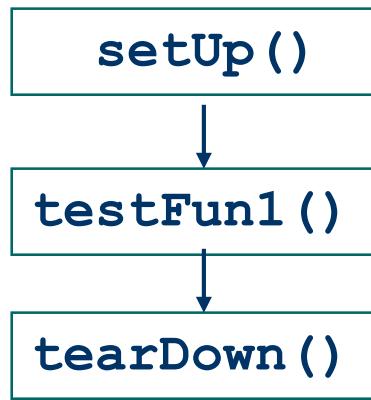
```
public class CalculetteTest extends TestCase
{
    private Calculette _calc;

    public void setUp() {
        _calc = new Calculette();
    }

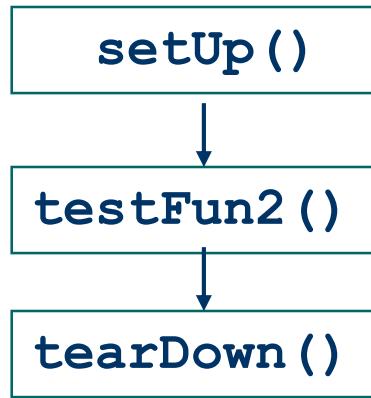
    public void testAdd() {
        int expected = 7;
        int actual = _calc.add(3, 4);
        assertEquals(expected, actual);
        assertEquals(-6, _calc.add(3, -2));
        assertEquals(-5, _calc.add(-3, -2));
    }

    public void tearDown() { _calc = null; }
}
```

# setUp et tearDown



- **setUp est exécutée avant chaque méthode testXXX**
- **tearDown est exécutée après chaque méthode testXXX**



# Exécuter un test isolément

- Définir un main utilisant un TestRunner

```
import junit.framework.TestCase;
public class CalculetteTest extends TestCase {
    ...
    public static void main(String[] args) {
        // affichage dans une vue textuelle
        junit.textui.TestRunner.run(CalculetteTest.class);
    }
}
```

- Il existe aussi des vues graphiques :
  - junit.swingui.TestRunner, ...

# Résultat de l'exécution du test (mode texte)

```
$ set CLASSPATH=/java/junit3.8.2/junit.jar
$ javac CalculetteTest.java
$ java CalculetteTest
.F
Time: 0
There was 1 failure:
1)
testAdd(CalculetteTest) junit.framework.AssertionFailedError:
expected:<-6> but was:<1>
    at CalculetteTest.testAdd(CalculetteTest.java:15)
    at CalculetteTest.main(CalculetteTest.java:21)
```

FAILURES!!!

Tests run: 1,

Failures: 1,

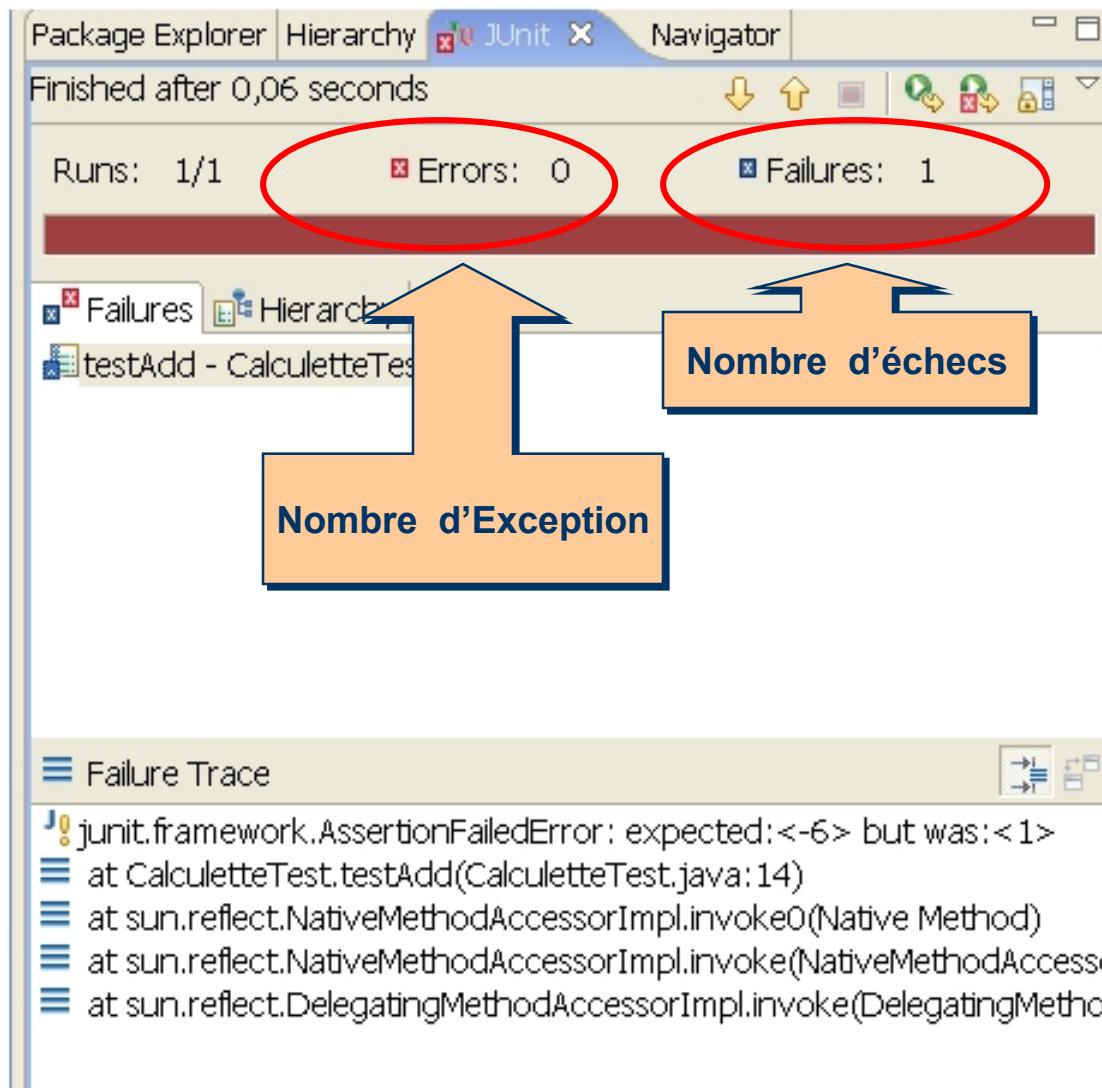
Errors: 0

Nombre de méthodes  
de test exécutées

Nombre d'échecs

Nombre d'Exception

# Résultat de l'exécution du test sous Eclipse



# Failure vs Exception



- "**Failure**" = assertion (`assertXXX`) qui échoue
- "**Error**" = Exception non prévue ou non attrapée

**Lors de la détection d'une "Failure" ou "Error" dans une méthode, JUnit :**

- interrompt l'exécution de cette méthode
- lance l'exécution de la méthode de test suivante

# Exemple avec Failures et Exceptions (1)

```
package junit38samples;
import junit.framework.TestCase; import java.io.*;
public class TestWithFailuresAndErrors extends TestCase {
    public void testFailure1() {
        assertEquals(1, 4 - 2); // échoue! => break
        assertEquals(1, 4 - 3);
    }
    public void testNullPointerExceptionThrown() {
        String s = null;
        assertEquals(0, s.length()); // Exception => break
    }
    public void testDivideByZero() {
        int i = 3 / 0; // Exception => break
        i = 5;
    }
    public void testException1() throws FileNotFoundException {
        FileReader fr = new FileReader("X.java"); // Exception => break
        assertNotNull(fr);
    }
    public void testException2() {
        try {
            FileReader fr = new FileReader("X.java");
            fail("exception expected"); // ne doit pas passer!
        } catch (FileNotFoundException e) { /* OK! */ }
        assertEquals(2, 1 + 1); // passe!
    }
    public static void main(String[] args) {
        junit.textui.TestRunner.run(TestWithFailuresAndErrors.class);
    }
}
```

# Exemple avec Failures et Exceptions (2)

```
$ set CLASSPATH=/java/junit3.8.1/junit.jar  
$ javac junit38samples/TestWithFailuresAndErrors.java  
$ java junit38samples.TestWithFailuresAndErrors  
.F.E.E.E.  
Time: 0  
There were 3 errors:  
1)  
testNullPointerExceptionThrown(junit38samples.TestWithFailur  
esAndErrors) java.lang.NullPointerException
```

[...]

FAILURES!!!

Tests run: 5,

Failures: 1,

Errors: 3

Nombre de méthodes  
de test exécutées

Nombre d'échecs

Nombre d'Exception

# Exemple avec Failures et Exceptions (3)

```
public void testFailure1() {  
    assertEquals(1, 4 - 2); // échoue! => break  
    assertEquals(1, 4 - 3); // vrai mais pas atteint!  
    assertEquals(2, 4 - 3); // faux mais pas atteint!  
}
```



```
There was 1 failure:  
1)  
testFailure1(TestWithFailuresAndErrors)junit.framework.AssertionFailedError:  
    expected:<1> but was:<2>  
        at  
TestWithFailuresAndErrors.testFailure1(TestWithFailuresAndErrors.java:6)
```

# Exemple avec Failures et Exceptions (4)

```
public void testNullPointerExceptionThrown() {  
    String s = null;  
    assertEquals(0, s.length()); // Exception => break  
    assertEquals(1, 4 - 3); // vrai mais pas atteint!  
    assertEquals(2, 4 - 3); // faux mais pas atteint!  
}
```



There were 3 errors:

```
1) testNullPointerExceptionThrown(TestWithFailuresAndErrors) java.lang.NullPointerException  
   at  
TestWithFailuresAndErrors.testNullPointerExceptionThrown (TestWithFailuresAndErrors.java:11)
```

# Exemple avec Failures et Exceptions (5)

```
public void testDivideByZero() {  
    int i = 3 / 0; // Exception => break  
    i = 5;         // pas atteint!  
}
```



```
2) testDivideByZero(TestWithFailuresAndErrors)  
   java.lang.ArithmetricException: / by zero  
     at  
TestWithFailuresAndErrors.testDivideByZero (TestWithFailuresAndErrors.java:14)
```

# Exemple avec Failures et Exceptions (6)

```
public void testException1() throws FileNotFoundException
{
    FileReader fr = new FileReader("X.java");
    // (Exception => break)
    assertNotNull(fr);    // pas atteint!
}
```



3) testException1(TestWithFailuresAndErrors) java.io.FileNotFoundException:  
X.java (Le fichier spécifié est introuvable)  
at java.io.FileInputStream.open(Native Method)  
at java.io.FileInputStream.<init>(FileInputStream.java:106)  
at java.io.FileInputStream.<init>(FileInputStream.java:66)  
at java.io.FileReader.<init>(FileReader.java:41)  
at  
TestWithFailuresAndErrors.testException1(TestWithFailuresAndErrors.java:18)

# Exemple avec Failures et Exceptions (7)

```
public void testException2() {  
    try {  
        FileReader fr = new FileReader("X.java");  
        fail("exception expected");  
    } catch (FileNotFoundException e) {  
        // OK!  
    }  
    assertEquals(2, 1 + 1); // passe!  
}
```



OK!

# Tester une méthode levant une Exception (1)

```
public class Customer {  
    // ...  
  
    public void checkData throws CheckException {  
        if (getFirstname() == null  
            || "".equals(getFirstname()))  
            throw new CheckException("Invalid first name");  
        // ...  
    }  
}
```

## 2 cas à traiter

- **Customer valide => pas d'exception**
- **Customer invalide => exception**

# Tester une méthode levant une Exception (2)

## Cas 1 : customer valide => pas d'exception

```
public void testCreateValidCustomer() {  
    try {  
        Customer c = new Customer("by867", "Bill", "Gates");  
        c.checkData();  
    } catch (CheckException e) {  
        fail("Method should not fail! Custumer data is OK!");  
    }  
}
```

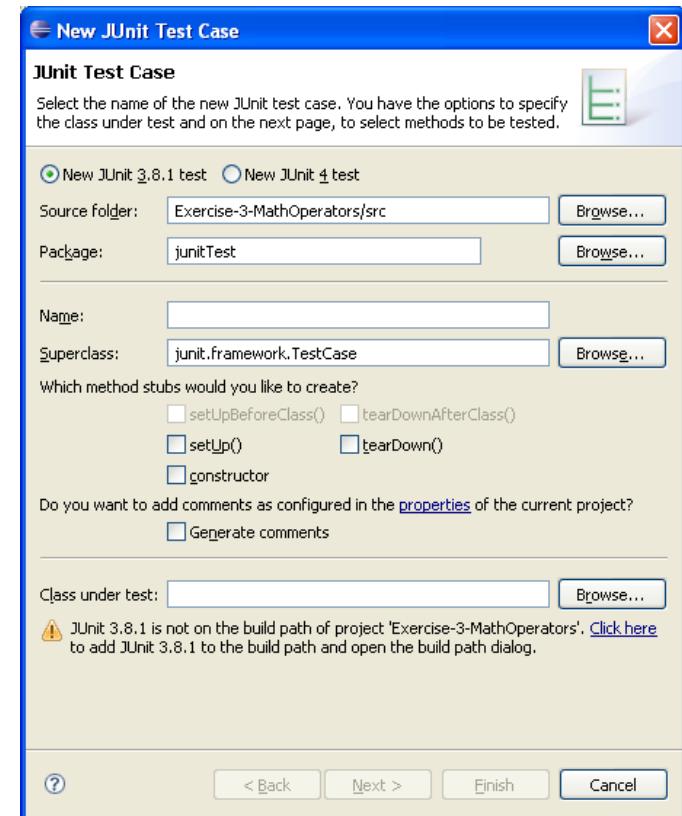
# Tester une méthode levant une Exception (3)

## Cas 2 : customer invalide => exception

```
public void testCreateCustomerWithInvalidValues() {  
    try {  
        Customer c = new Customer("jobs01", null, "Jobs");  
        c.checkData();  
        fail("Object with empty values should not be created");  
    } catch (CheckException e) {  
        assertEquals("Invalid first name", e.getMessage());  
    }  
}
```

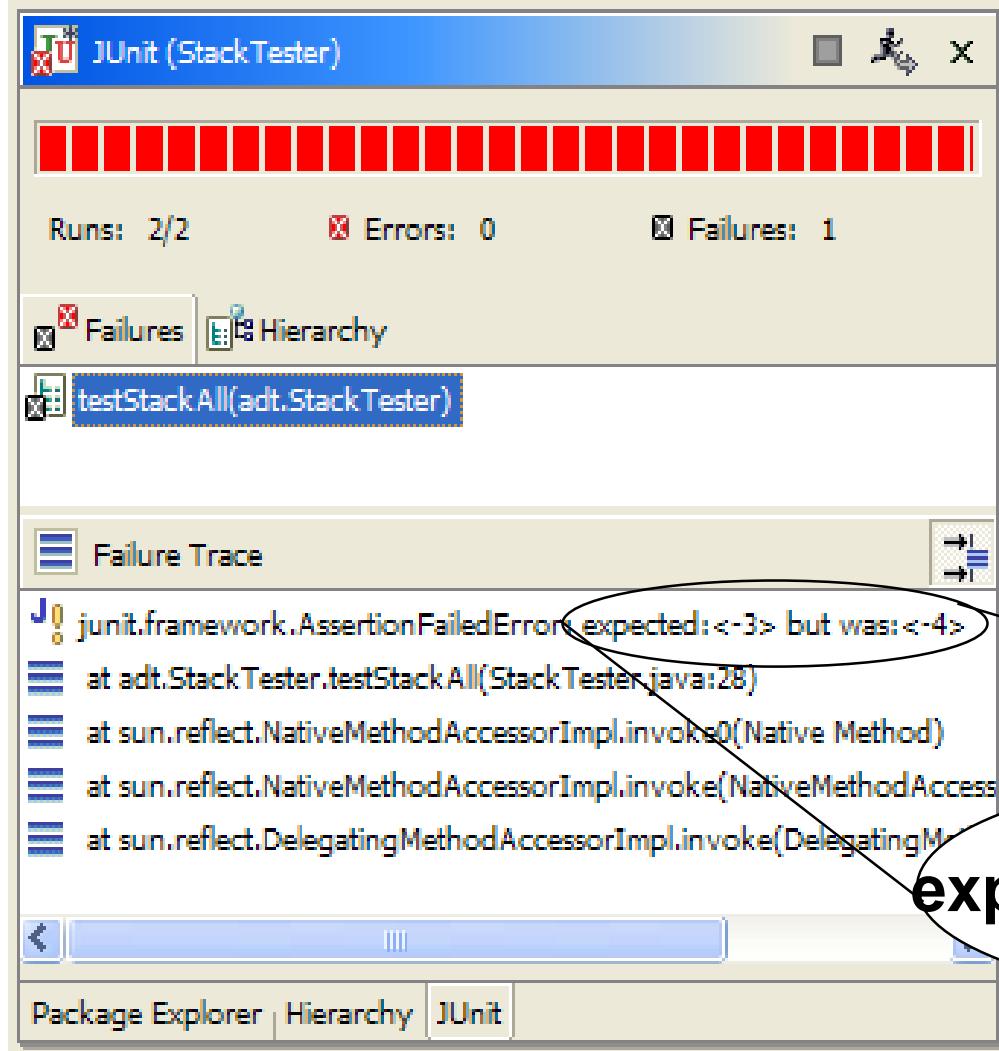
# JUnit dans Eclipse – créer un test

- File
  - New
    - Junit Test Case
      - Set the parameters:
        - Junit 3.x or 4.x
        - name of the class
        - etc.
      - Finish

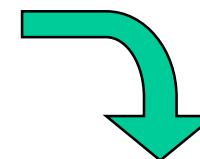


# JUnit dans Eclipse – De la barre rouge à verte

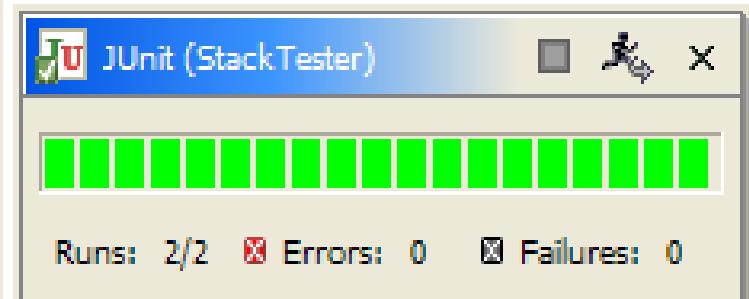
Echec



Mise au point itérative ...

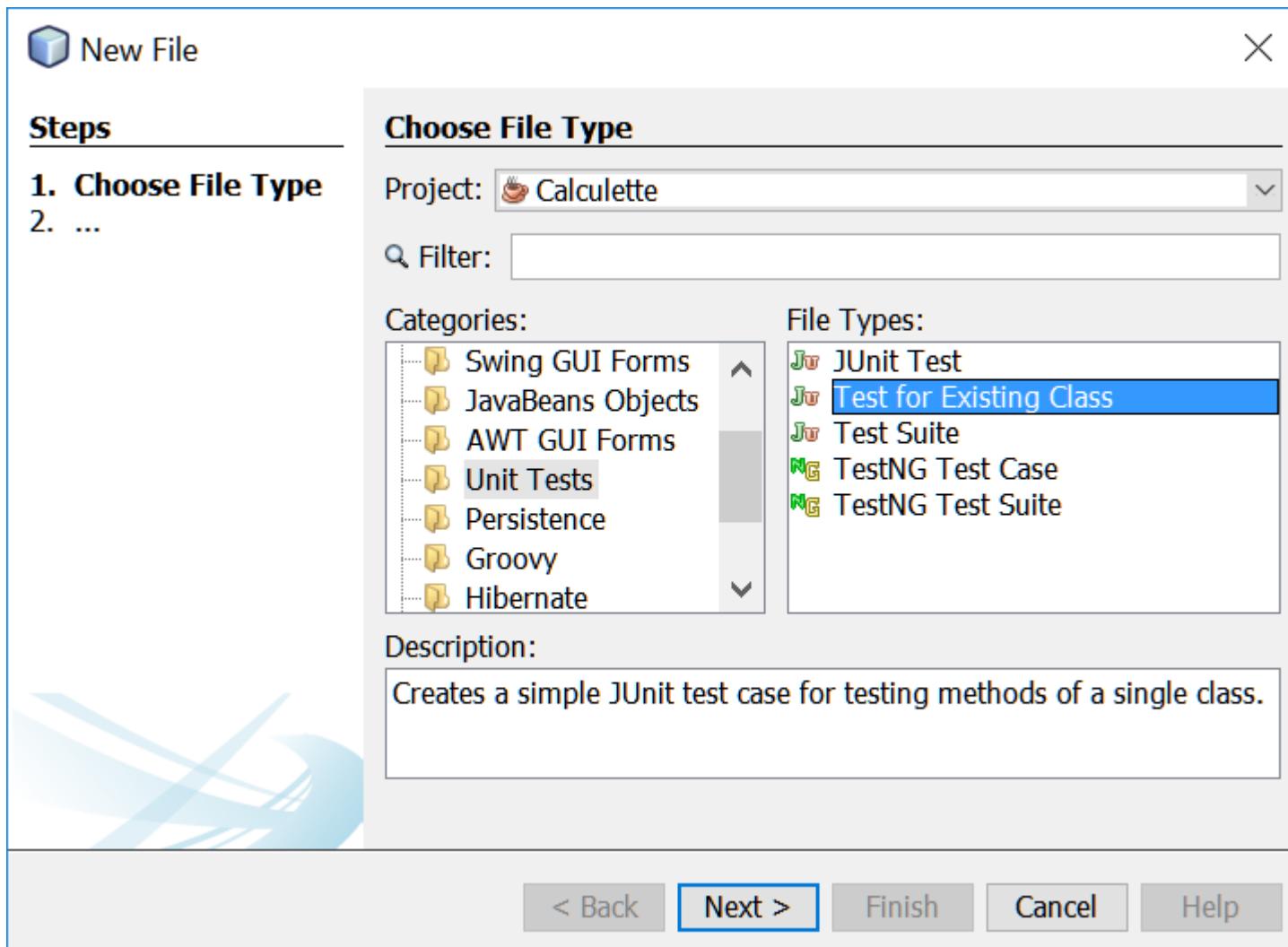


Succès

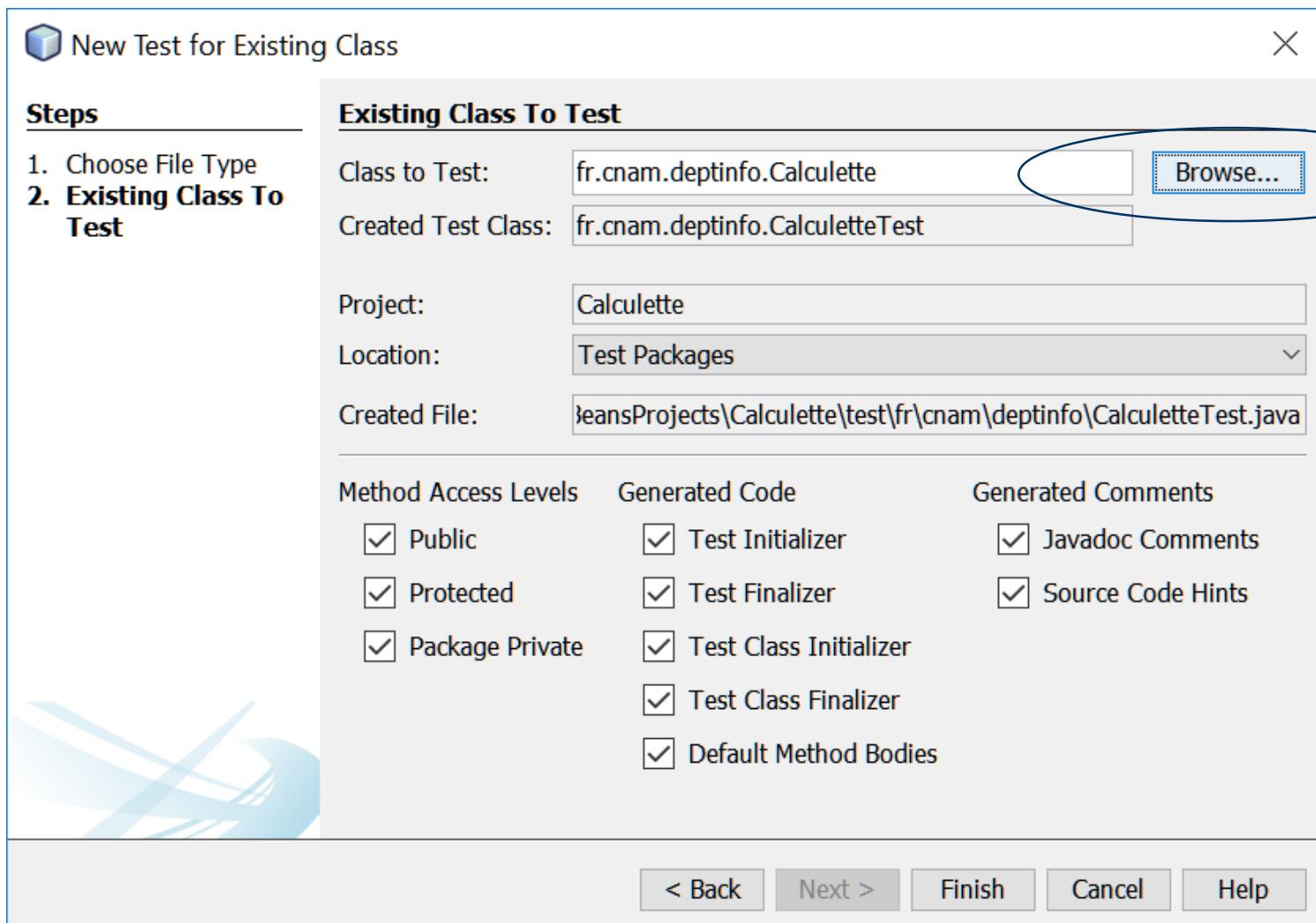


**expected <-3> but was <-4>**

# JUnit dans Netbeans – créer un test



# JUnit dans Netbeans – créer un test



# JUnit dans Netbeans – créer un test

```
package fr.cnam.deptinfo;

/*
 * @author Frederic
 */
public class Calculette {
    public int add(int a, int b) {
        return a+b;
    }

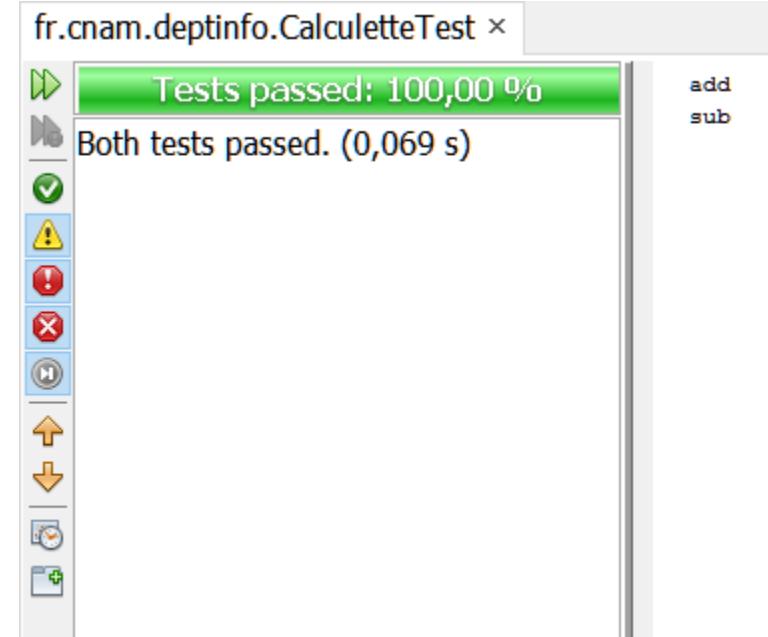
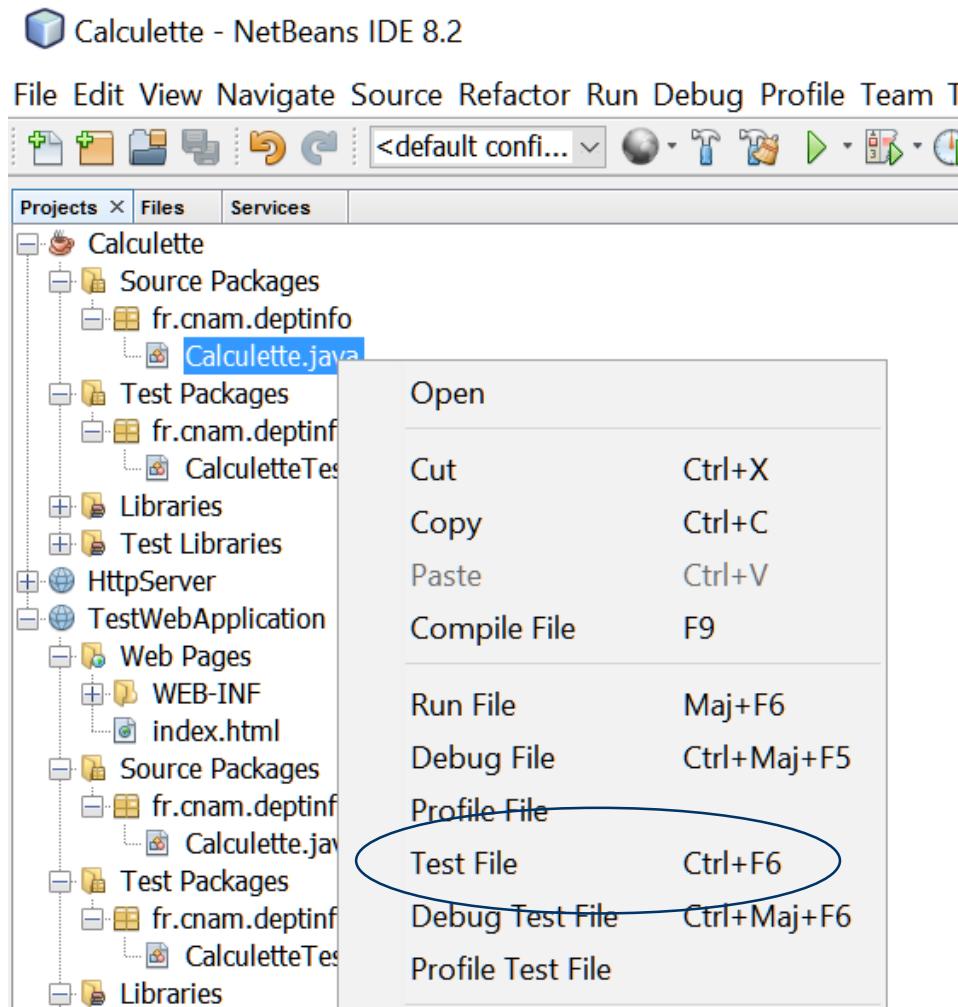
    public int sub(int a, int b) {
        return a-b;
    }
}
```

```
public class CalculetteTest {

    /**
     * Test of add method, of class Calculette.
     */
    @Test
    public void testAdd() {
        System.out.println("add");
        int a = 3;
        int b = 2;
        Calculette instance = new Calculette();
        int expResult = 5;
        int result = instance.add(a, b);
        assertEquals(expResult, result);
    }

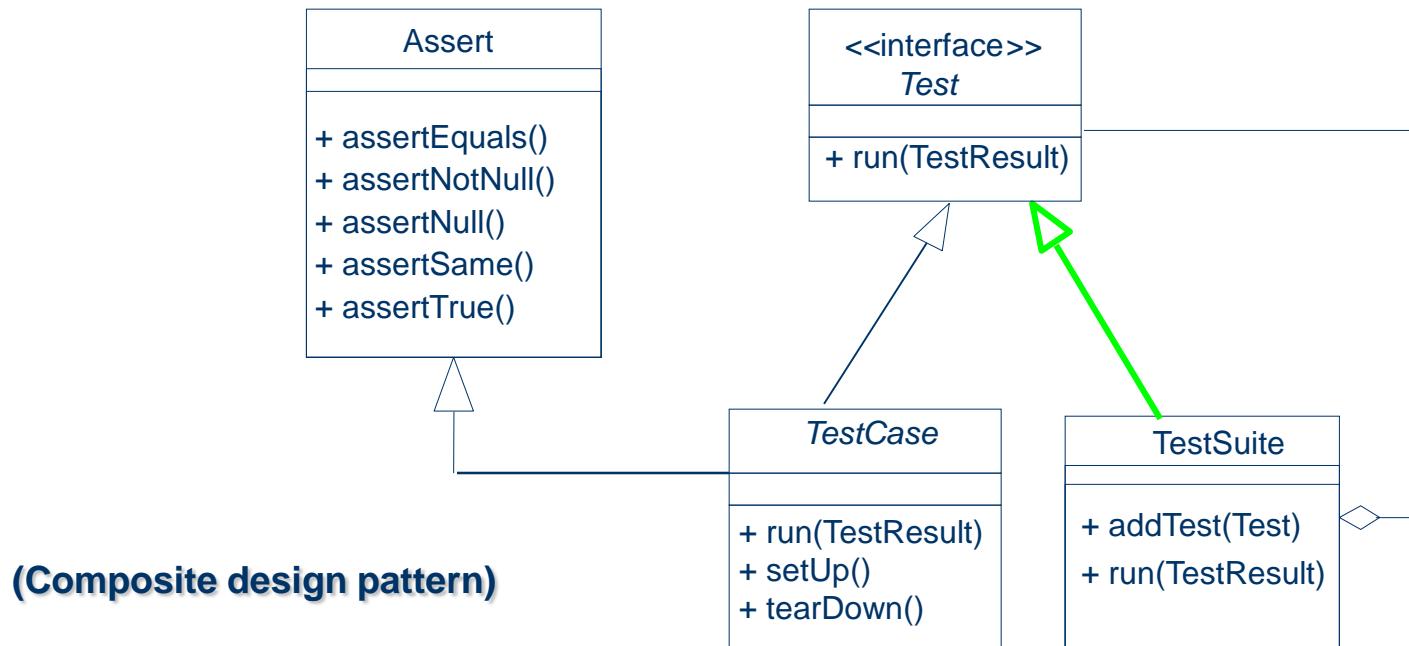
    /**
     * Test of sub method, of class Calculette.
     */
    @Test
    public void testSub() {
        System.out.println("sub");
        int a = 10;
        int b = 2;
        Calculette instance = new Calculette();
        int expResult = 8;
        int result = instance.sub(a, b);
        assertEquals(expResult, result);
    }
}
```

# JUnit dans Netbeans – créer un test



# TestSuite

Une TestSuite est une composition de Test



# Construction d'une TestSuite

On définit dans les classes de Test la méthode  
**public static TestSuite suite()**

2 possibilités :

- Ajouter les méthodes à tester explicitement

`public TestSuite()`

construit une TestSuite vide

`public void addTest(Test test)`

- Utiliser l'introspection

`public TestSuite(java.lang.Class theClass)`

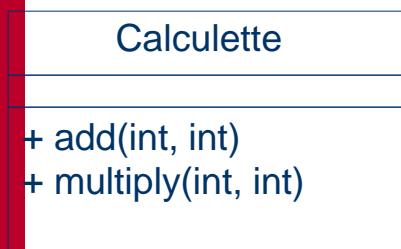
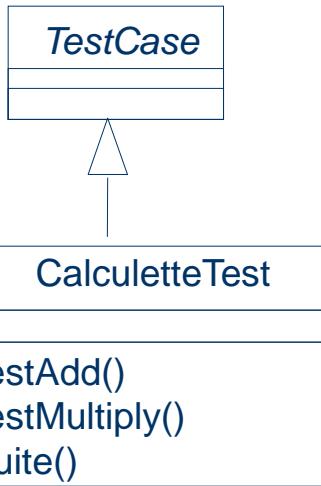
Adds all the methods starting with "test" as test cases to the suite.

# Construction d'une TestSuite explicitement

```
public class CalculetteTest extends TestCase {  
  
    public void testAdd() {  
        assertEquals(7, _calc.add(3, 4));  
    }  
    // ...  
    public CalculetteTest(String name) {  
        super(name);  
    }  
    public static Test suite() {  
        TestSuite suite = new TestSuite();  
        suite.addTest(new CalculetteTest("testAdd"));  
        suite.addTest(new CalculetteTest("testMultiply"));  
        // ...  
        return suite;  
    }  
}
```

**Chaque méthode à tester doit être ajoutée explicitement.**

# Construction d'une TestSuite par Introspection



```
public class CalculetteTest extends TestCase
{
    // ...
    public void testAdd() {
        int expected = 7;
        assertEquals(expected, _calc.add(3, 4));
    }
    public void testMultiply() {
        assertEquals(12, _calc.multiply(3, 4));
    }
    // ...
    public static Test suite() {
        return new
            TestSuite(CalculetteTest.class);
    }
}
```

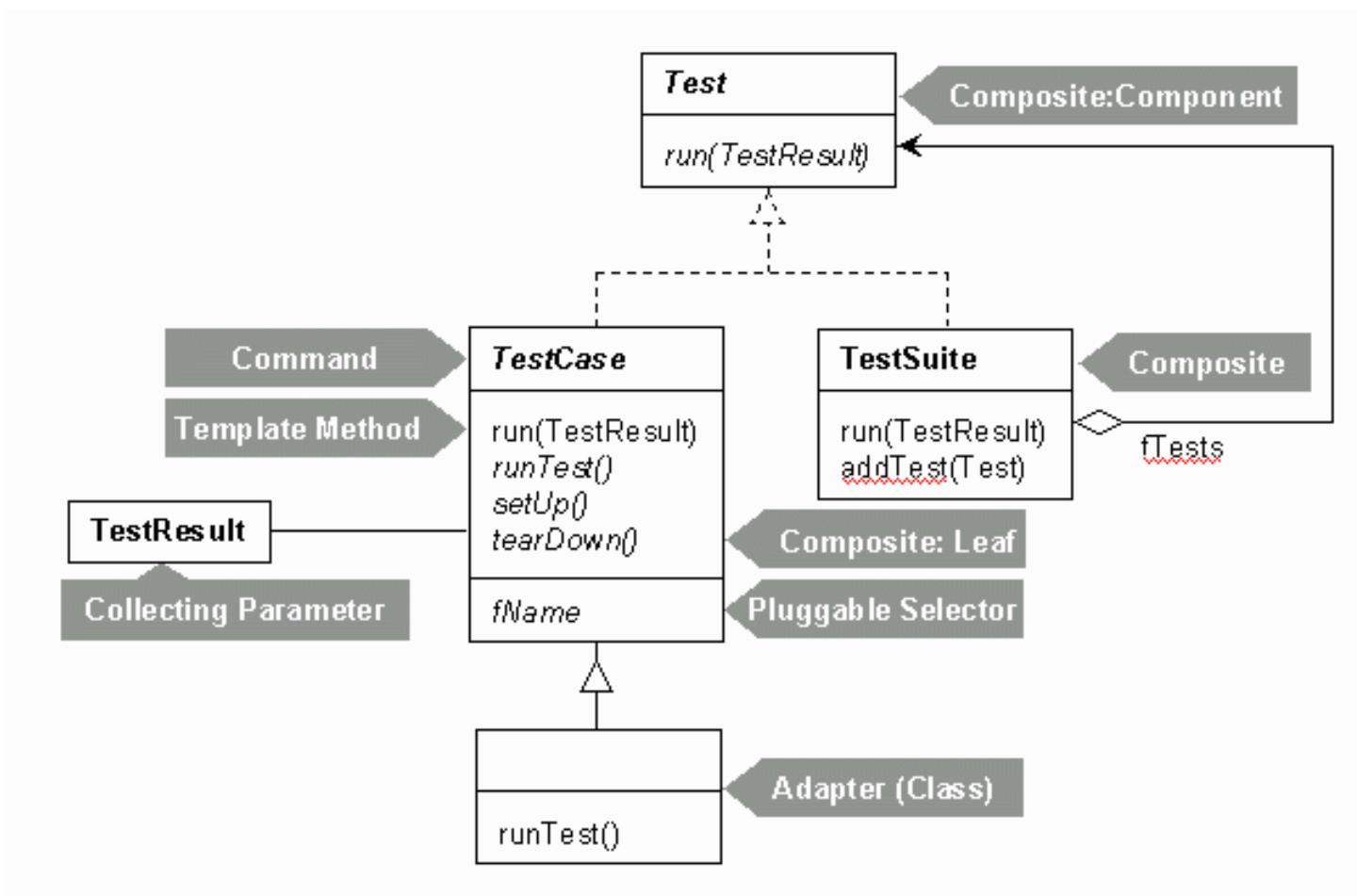
The code snippet shows the implementation of the "CalculetteTest" class. It extends the "TestCase" class. It contains several test methods: "testAdd()", "testMultiply()", and "suite()". The "testAdd()" method uses the "assertEquals" assertion to check if the result of calling "add(3, 4)" on the instance variable "\_calc" equals 7. The "testMultiply()" method does the same for the "multiply" method. The "suite()" method returns a new "TestSuite" object containing the "CalculetteTest" class.

**Chaque méthode (sans paramètre, qui retourne void et) dont le nom est préfixé par test est ajoutée à la suite.**

# Exécuter un ensemble de tests

```
public class AllTest extends TestCase {  
    ...  
    public static TestSuite suite() {  
        final TestSuite suite = new TestSuite();  
        // Domain test  
        suite.addTest(CategoryTest.suite());  
        suite.addTest(CustomerTest.suite());  
        // Service test  
        suite.addTest(CustomerServiceTest.suite());  
        // Web test  
        suite.addTest(CreateCustomerServletTest.suite());  
        return suite;  
    }  
  
    public static void main(String[] args) {  
        junit.textui.TestRunner.run(suite());  
    }  
}
```

# JUnit 3.8 et Design patterns



(<http://junit.sourceforge.net/junit3.8.1/doc/cookstour/cookstour.htm>)

## JUnit 4

<http://www.devx.com/Java/Article/31983/1954?pf=true>

<http://junit.org>



# Caractéristiques de JUnit4

## JUnit 4

- **assure une compatibilité ascendante et descendante**
  - JUnit 4 peut exécuter directement des tests JUnit 3
  - Des tests JUnit 4 peuvent être exécuter dans des environnements JUnit 3 (grâce à JUnit4TestAdapter)
- **utilise les annotations Java 5 à la place de l'héritage et des conventions de nommage**
- **permet à une classe de test d'hériter d'une autre classe**

# Principales différences

## Avec JUnit4 :

- On utilise les annotations sur les méthodes à la place des conventions de noms des méthodes :
  - **@Test** indique une méthode de test
  - **@Before** indique une méthode setUp
  - **@After** indique une méthode tearDown
- Plus besoin d'hériter de TestCase
- Plus besoin de créer une TestSuite
- On importe org.junit.\*

# Exemple avec fixture

```
import  
    junit.framework.TestCase;  
  
public class CalculetteTest  
extends TestCase {  
private Calculette _calc;  
  
public void setUp()  
{_calc = new Calculette();}  
  
public void testadd() {  
int actual= _calc.add(3, 4);  
assertEquals(7,actual);  
}  
}
```

JUnit 3.8

```
import org.junit.Before;  
import org.junit.Test;  
import static  
    org.junit.Assert.*;  
  
public class CalculetteTest  
{  
private Calculette _calc;  
  
@Before public void setUp1()  
{_calc = new Calculette();}  
  
@Test public void add() {  
int actual= _calc.add(3, 4);  
assertEquals(7,actual);  
}  
}
```

JUnit 4

# Exécuter un test isolément en mode texte (1)

Il n'est pas nécessaire de définir un main grâce à JUnitCore

```
$ set CLASSPATH=/java/junit4.4/junit-4.4.jar  
$ javac junit4samples/CalcTest.java  
$ java org.junit.runner.JUnitCore junit4samples.CalcTest  
JUnit version 4.4  
.E  
Time: 0,01  
There was 1 failure:  
1) testAdd(junit4samples.CalcTest)  
java.lang.AssertionError: expected:<-6> but was:<1>  
    at org.junit.Assert.fail(Assert.java:74)  
        [...]  
FAILURES!!!  
Tests run: 1,
```

Nombre de méthodes de test exécutées

Nombre d'échecs

Plus de nombre d'Exception!!

# Exécuter un test isolément en mode texte (2)

On peut aussi définir un main utilisant la façade JUnitCore

```
package junit4samples;
import org.junit.Test;
import static org.junit.Assert.*;
public class CalcTest {
    // ...
    @Test public void add() {
        assertEquals(-6, _calc.add(3, -2));
    }
    public static void main(String[] args) {
        org.junit.runner.JUnitCore.main(
            "junit4samples.CalcTest");
    }
}
```

Note: Il n'existe plus de vue graphique (Green Bar) fournie par JUnit 4!

# Exécuter un test isolément en mode texte (3)

L'output est identique au précédent

```
$ set CLASSPATH=/java/junit4.4/junit-4.4.jar  
$ javac junit4samples/CalcTest.java  
$ java junit4samples.CalcTest  
JUnit version 4.4  
.E  
Time: 0,01  
There was 1 failure:  
1) testAdd(junit4samples.CalcTest)  
java.lang.AssertionError: expected:<-6> but was:<1>  
    at org.junit.Assert.fail(Assert.java:74)  
        [...]  
FAILURES!!!  
Tests run: 1,
```

Nombre de méthodes de test exécutées

Nombre d'échecs

Plus de nombre d'Exception!!

# JUnit 4 ne distingue plus Failure et Exception

## JUnit version 3.8

FAILURES!!!

Tests run: 5,

Failures: 1

Errors: 3

Nombre de méthodes  
de test exécutées

Nombre d'échecs

Nombre d'Exception

## Une Exception non prévue se traduit par une failure

## JUnit version 4.4

FAILURES!!!

Tests run: 5,

Failures: 4

Nombre de méthodes  
de test exécutées

Nombre d'échecs

Rien!

# Exécuter un ensemble de tests

On annote une nouvelle classe avec `@RunWith` et `@Suite`

```
package junit4samples;
import org.junit.runner.RunWith;
import org.junit.runners.Suite;

@RunWith(Suite.class)
@Suite.SuiteClasses({
    Calc1Test.class,
    Calc2Test.class
})
public class AllJ4CalcTests {
    public static void main(String[] args) {
        org.junit.runner.JUnitCore.main(
            "junit4samples.AllJ4CalcTests");
    }
}
```

# Paramètres optionnels de @Test

```
package junit4samples;  
import java.util.ArrayList;  
import org.junit.Test;  
  
public class TestParametersTest {  
  
    @Test(expected=IndexOutOfBoundsException.class)  
    public void outOfBounds() {  
        new ArrayList<Object>().get(1);  
    }  
  
    @Test(timeout=100)  
    public void foreverLoop()  
    {  
        for (;;) ;  
    }  
}
```

**@Test(expected=IndexOutOfBoundsException.class)**

**@Test(timeout=100)**

Ce test doit échouer s'il dure plus de 100 ms

Cette méthode doit lever une exception de ce type sinon le test échoue