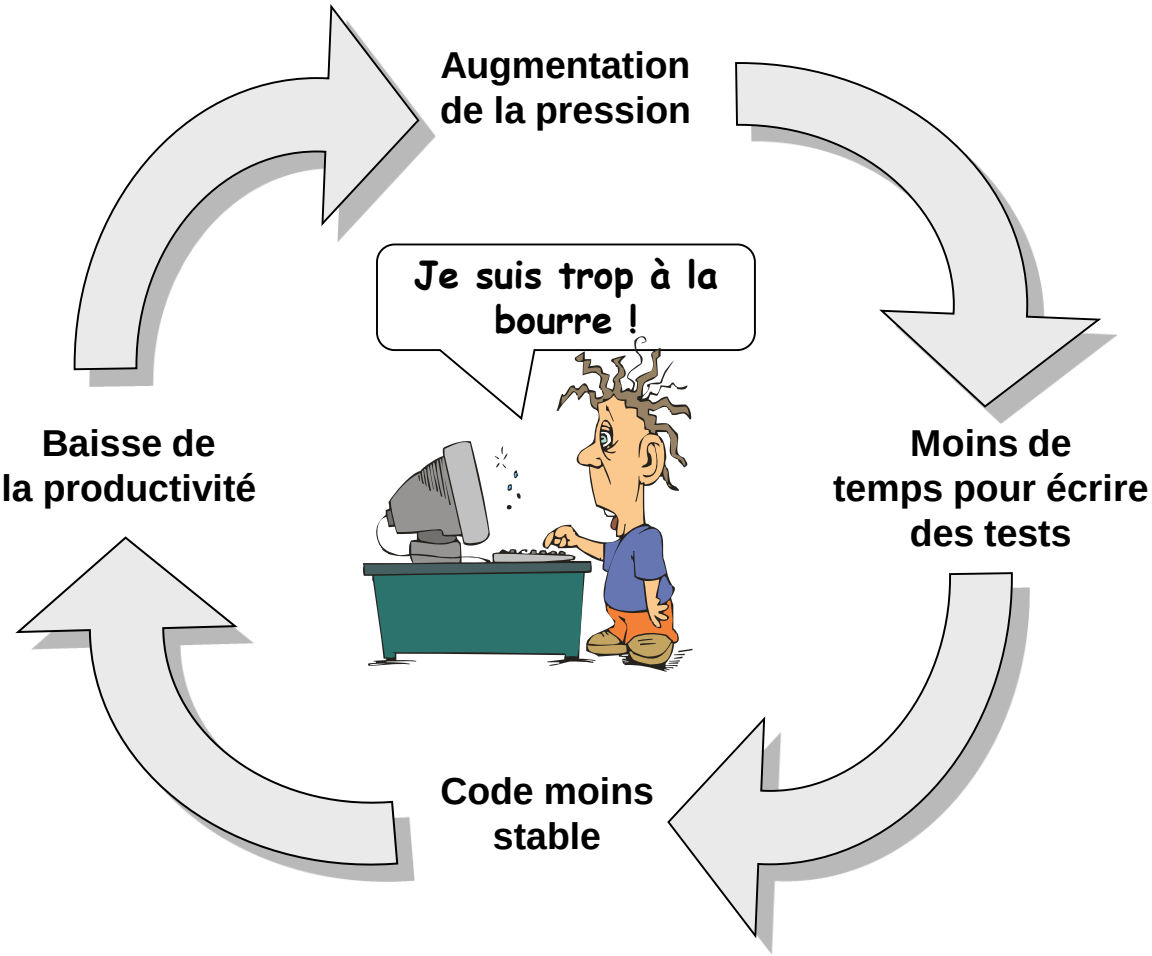
# Tests unitaires avec JUnit

*Les programmeurs aiment écrire des tests*

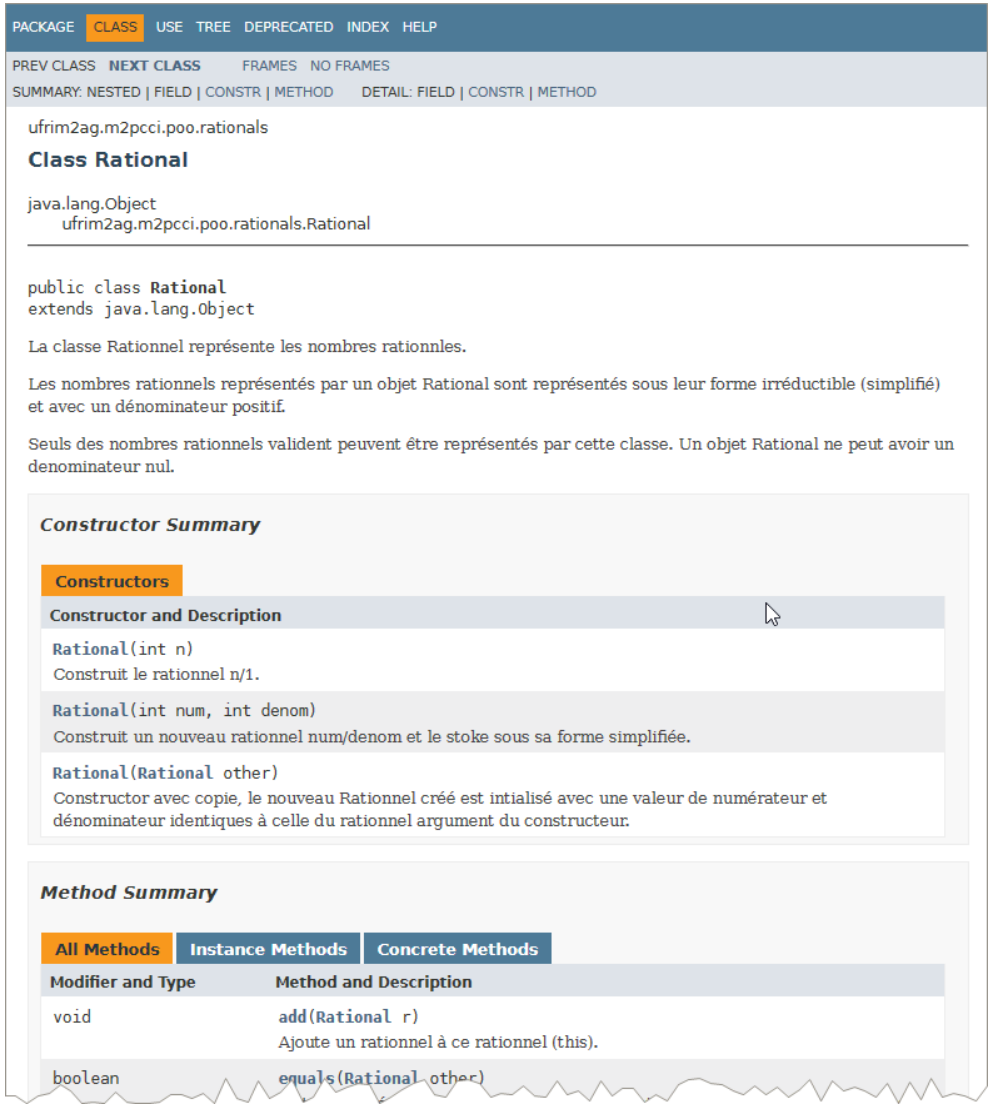
# Le problème du test

- Tous les programmeurs savent qu'ils doivent écrire des tests, peu le font..



# Test Unitaire d'une classe

- L'interface publique d'une classe définit "un contrat" entre celui qui fournit la classe et celui qui l'utilise.
- Ce contrat définit:
  - *les services proposés par la classe*
  - *la manière dont ces services doivent être utilisés*
- Tester une classe consiste à vérifier la validité de ce contrat
  - *il confronte la réalisation de la classe à sa spécification.*



PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

ufrim2ag.m2pcci.poo.rationals

**Class Rational**

java.lang.Object
ufrim2ag.m2pcci.poo.rationals.Rational

public class **Rational**
extends java.lang.Object

La classe Rationnel représente les nombres rationnles.

Les nombres rationnels représentés par un objet Rational sont représentés sous leur forme irréductible (simplifié) et avec un dénominateur positif.

Seuls des nombres rationnels valident peuvent être représentés par cette classe. Un objet Rational ne peut avoir un denominateur nul.

***Constructor Summary***

Constructors

| Constructor and Description |
|---|
| `Rational(int n)` Construit le rationnel n/1. |
| `Rational(int num, int denom)` Construit un nouveau rationnel num/denom et le stoke sous sa forme simplifiée. |
| `Rational(Rational other)` Constructor avec copie, le nouveau Rationnel créé est intialisé avec une valeur de numérateur et dénominateur identiques à celle du rationnel argument du constructeur. |

***Method Summary***

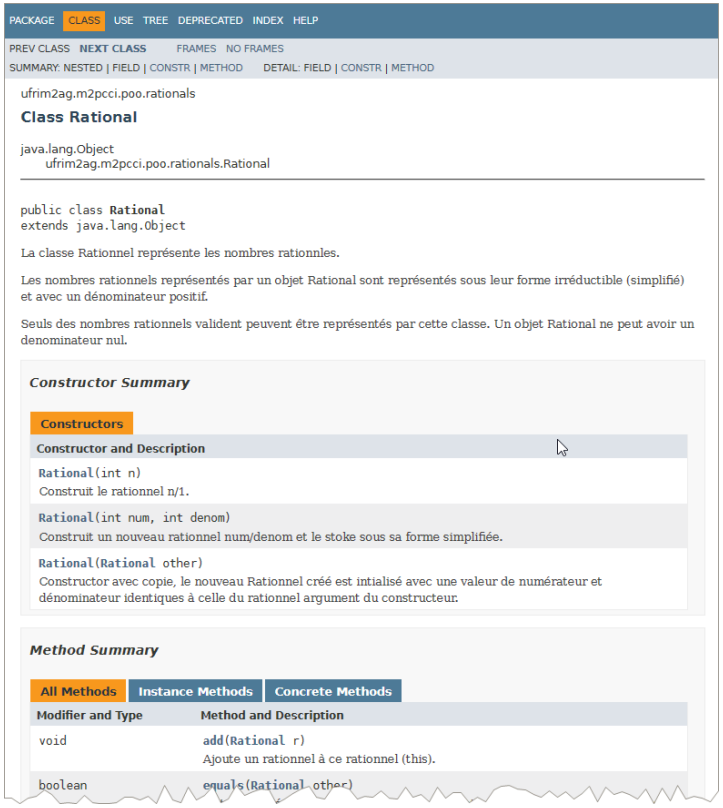
All Methods | Instance Methods | Concrete Methods

| Modifier and Type | Method and Description |
|---|---|
| void | `add(Rational r)` Ajoute un rationnel à ce rationnel (this). |
| boolean | `equals(Rational other)` |

# Comment tester une classe ?

- Traces (instruction **`print`**) dans les programmes

Spécification



PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

ufrim2ag.m2pcci.poo.rationals

**Class Rational**

java.lang.Object
ufrim2ag.m2pcci.poo.rationals.Rational

public class **Rational**
extends java.lang.Object

La classe Rationnel représente les nombres rationnles.

Les nombres rationnels représentés par un objet Rational sont représentés sous leur forme irréductible (simplifié) et avec un dénominateur positif.

Seuls des nombres rationnels valident peuvent être représentés par cette classe. Un objet Rational ne peut avoir un denominateur nul.

***Constructor Summary***

**Constructors**

| Constructor and Description |
|---|
| `Rational(int n)` Construit le rationnel n/1. |
| `Rational(int num, int denom)` Construit un nouveau rationnel num/denom et le stoke sous sa forme simplifiée. |
| `Rational(Rational other)` Constructor avec copie, le nouveau Rationnel créé est intialisé avec une valeur de numérateur et dénominateur identiques à celle du rationnel argument du constructeur. |

***Method Summary***

**All Methods** | **Instance Methods** | **Concrete Methods**

| Modifier and Type | Method and Description |
|---|---|
| void | `add(Rational r)` Ajoute un rationnel à ce rationnel (this). |
| boolean | `equals(Rational other)` |

Implémentation

Rational.java

TestRational.java

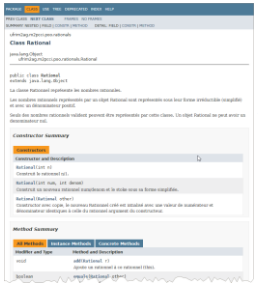
```java
package ufrim2ag.m2pcci.poo.rationals;

/**
 * Programme simple de test de la classe Rational
 * @author Philippe Genoud
 */
public class TestRational {

    public static void main(String[] args) {
        Rational r = new Rational(6, 4);
        System.out.println("r = " + r.toString());
        System.out.println("\tsous la forme réelle, r = " + r.toDouble());

        Rational s = new Rational(2);
        System.out.println("s = " + s);

        r.add(s);
        System.out.println("r + s = " + r);

        Rational t = new Rational(34, 8);
        System.out.println("t = " + t);

        System.out.print(s + " x " + t + " = " );
        s.mult(t);
        System.out.println(s);
    }
}
```

Le programme de test

# Comment tester une classe ?

- Traces (instruction **`print`**) dans les programmes

Spécification



Le programme de test

TestRational.java

Exécution

Trace d'exécution

```
run:
r = 3 / 2
        sous la forme réelle, r = 1.5
s = 2
r + s = 7 / 2
t = 17 / 4
2 x 17 / 4 = 17 / 2
BUILD SUCCESSFUL (total time: 0 seconds)
```

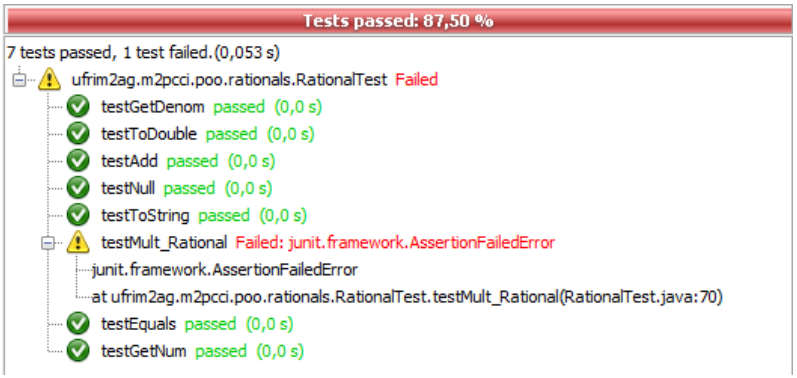
Rational.java

Implémentation

- **Inefficace**
  - *nécessitent un jugement humain*
  - *temps élevé, risques d'erreur*
    - *si une trace contient de nombreux print, perte de lisibilité (« Scroll Blindness »)*
  - *processus à renouveler chaque fois que la classe est modifiée et/ou qu'un bug est corrigé*


➔ **Automatiser les tests et les confier à un programme**

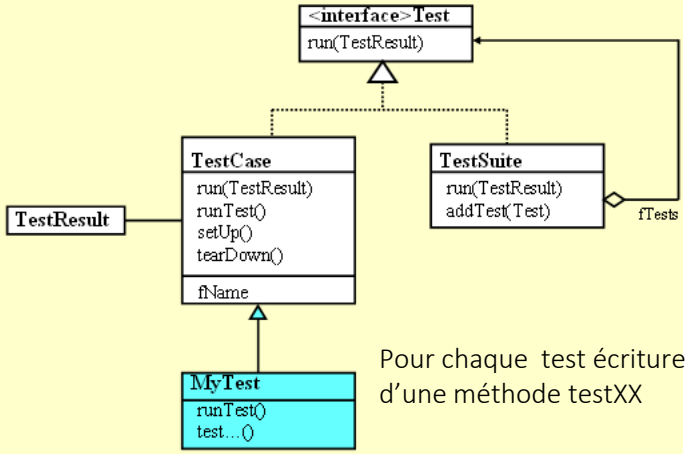
- **JUnit :** un framework open source (www.junit.org) pour le test unitaire de programmes Java qui permet **d'automatiser les tests.**
- **Facilite**
  - *écriture des programmes de tests unitaires*
  - *exécution des test unitaires*
  - *l'exploitation des résultats de test*




- Terminologie JUnit
  - ***Test unitaire (Unit test)*** *: test d'une classe*
  - ***Cas de test (Test case)*** *: teste les réponses d'une méthode à un ensemble particulier d'entrées*
  - ***Suite de tests (Test suite)*** *: une collection de cas de tests*
  - ***Testeur (Test runner)*** *: programme qui exécute des suites de tests et rapporte les résultats*

# Mise en œuvre de tests avec JUnit

- 2 versions
- JUnit 3.8
  - *Basé sur un framework de Classes et Interfaces*
  - *Ecriture de tests en écrivant des classes s'intégrant au framework*




- JUnit 4
  - *Basé sur des annotations (Java 5+)*
  - *Ces annotations permettent de marquer les classes et méthodes de test*
  - *Nouvelles fonctionnalités et plus de souplesse (plus de contraintes d'héritage)*

- Accroche une information à un élément (classe, méthode, attribut) Java
- Utilisée par un mécanisme tiers (compilateur java, environnement d'exécution...) ... et non par la classe où elle est définie

```
@Test
public void getSolde() {
    ...
 }
```

- bientôt JUnit 5
  - *1 trimestre 2017*

JUnit 5

*JUnit 5* is the next generation of JUnit. The goal is to create an up-to-date foundation for developer-side testing on the JVM. This includes focusing on Java 8 and above, as well as enabling many different styles of testing.

# Ecriture d'une classe de Test

- La classe à tester :

| Counter |
| --- |
| - count : int |
| + Counter()<br>+ Counter(int)<br>+ int increment()<br>+ int decrement<br>+ int getValue()<br>+ Counter add(Counter)<br>+ Counter sub(Counter) |

# JUnit 4 Ecriture d'une classe de Test

- La classe de test :

```
import org.junit.Test;
import org.junit.Assert; import static org.junit.Assert.*;
public class TestCounter {

    //…

    @Test
    public void testSimpleAdd() {

        Counter c1 = new Counter(10);
        Counter c2 = new Counter(12);

        Counter c3 = c1.add(c2);

        Assert.assertTrue(c3.getValue() ==
                          c1.getValue() +c2.getValue());
    }
}
```

Création des objets qui vont interagir lors du test

Code qui agit sur les objets impliqués dans le test

Vérification que le résultat obtenu correspond bien au résultat attendu.

static void assertTrue(boolean ***test***)

*méthode JUnit : vérifie que* test == true *et dans le cas contraire lance une exception (en fait une Error) de type* AssertionFailedError

*L'exécuteur de tests JUnit attrape ces objets* `Errors` *et indique les tests qui ont échoué*

# JUnit 4 Les différentes méthodes assert

- `static void assertTrue(String message, boolean test)`
  - Le *message* optionnel est inclus dans l'Error

- `static void assertFalse(boolean test)`
  `static void assertFalse(String message, boolean test)`
  - *vérifie que* test == true

- `assertEquals(expected, actual)`
  `assertEquals(String message, expected, actual)`
  - *méthode largement surchargée:* arg1 *et* arg2 *doivent être tout deux des objets* ou bien *du même type primitif*
  - *Pour les objets, utilise la méthode equals (public boolean equals(Object o)) – sinon utilise* ==

- `assertSame(Object expected, Object actual)`
  `assertSame(String message, Object expected, Object actual)`
  - *Vérifie que* expected *et* actual *référencent le même objet (==)*

- `assertNotSame(Object expected, Object actual)`
  `assertNotSame(String message, Object expected, Object actual)`
  - *Vérifie que* expected *et* actual *ne référencent pas le même objet (==)*

# JUnit 4 Les différentes méthodes assert

- `assertNull(Object object)`
  `assertNull(String message, Object object)`
  - *Vérifie que* objet *est* null

- `assertNotNull(Object object)`
  `assertNotNull(String message, Object object)`
  - *Vérifie que* objet *n'est pas* null

- `fail()`
  `fail(String message)`
  - *Provoque l'échec du test et lance une* `AssertionFailedError`
  - *Utile lorsque les autres méthodes* `assert` *ne correspondent pas exactement à vos besoins ou pour tester que certaines exceptions sont bien lancées.*

```
try {
   // appel d'une méthode devant lancer une Exception
     ....
   // si l'exception n'a pas eu lieu on force le test  échouer
     fail("Did not throw an ExpectedException");
}
catch (ExpectedException e) { }
```

# JUnit 4 Ecriture d'une classe de Test

*Verifying that code completes normally is only part of programming.*
*Making sure the code behaves as expected in exceptional situations is part of the craft of programming too.*

JUnit Cookbook
Kent Beck, Erich Gamma

- Comment vérifier que des exceptions sont lancées comme prévu ?

```
@Test
public void testEmpty() {
      try {
         new ArrayList<Object>().get(0);
         fail("ArrayIndexOutOfBounds non lancée");
      }
      catch (ArrayIndexOutOfBounds e) {
         // OK exception lancée
      }
}
```

à la JUnit 3.8

Avec JUnit4 paramètre expected de l'annotation **@Test**

```
@Test (expected=IndexOutOfBoundsException.class)
public void testEmpty() {
   new ArrayList<Object>().get(0);
}
```

Le test échoue si l'exécution de la méthode dépasse le temps fixé par le paramètre **timeout**

```
@Test(timeout=10)
public void uneMéthode() {
   ...
}
```

- Plusieurs méthodes de test dans une même classe de Test

```
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import static org.junit.Assert.*;

public class TestCounter {
    //…

    @Test
    public void add() {

        Counter c1 = new Counter(10);
        Counter c2 = new Counter(12);
        Counter c3 = c1.add(c2);
        assertTrue(c3.getValue() ==
                    c1.getValue() + c2.getValue());
    }


    @Test
    public void sub() {

        Counter c1 = new Counter(10);
        Counter c2 = new Counter(12);
        Counter c3 = c1.sub(c2);
        assertTrue(c3.getValue() ==
                    c1.getValue() - c2.getValue());
    }

}
```

```
private Counter c2;
private Counter c1;
```

Code exécuté avant chaque méthode de test

```
@Before
public void setUp() {

    c1 = new Counter(10);
    c2 = new Counter(12);
}
```

Code exécuté après chaque méthode de test

```
@After
public void tearDown() {
    ...
    ...
}
```

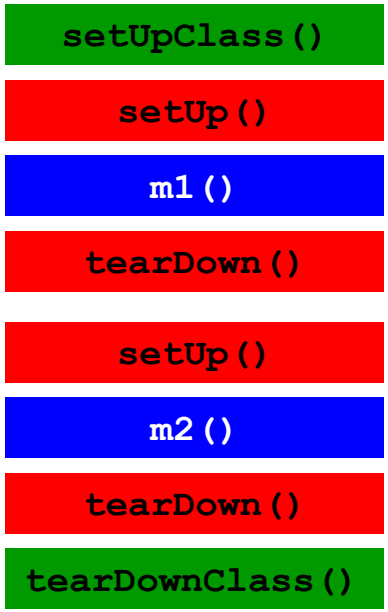
```
import org.junit.*;
import static org.junit.Assert.*;

public class TestCounter {

    @BeforeClass
    public static void setUpClass() {
        ...
    }

    @AfterClass
    public static void tearDownClass() {
        ...
    }

    @Before
    public void setUp() {
       ...
    }

    @After
    public void tearDown() {
       ...
    }

    @Test
    public void m1() {
       ...
    }
    @Test
    public void m2() {
       ...
    }
}
```

Code exécuté **une seule fois** avant d'exécuter toutes les méthodes de test de la classe

Code exécuté **une seule fois** après avoir exécuté toutes les méthodes de test de la classe

Code exécuté avant chaque méthode de test de la classe

Code exécuté après chaque méthode de test de la classe



- en mode texte sur la console

```
$ set CLASSPATH=.:/java/junit4.10/junit-4.10.jar
$ javac CounterTest.java
$ java org.junit.runner.JUnitCore CounterTest
JUnit version 4.10
....E.
Time: 0,007
There was 1 failure:
1) testAdd(CounterTest)
java.lang.AssertionError: expected:<2> but was:<3>
            at org.junit.Assert.fail(Assert.java:93)
            at org.junit.Assert.failNotEquals(Assert.java:647)
            at org.junit.Assert.assertEquals(Assert.java:128)
            at org.junit.Assert.assertEquals(Assert.java:472)
            at org.junit.Assert.assertEquals(Assert.java:456)
            at CounterTest.testAdd(CounterTest.java:42)
            at sun.reflect.NativeMethodAccessorImpl.invo
            …
            at Essai.main(Essai.java:14)

FAILURES!!!
Tests run: 5,  Failures: 1
```

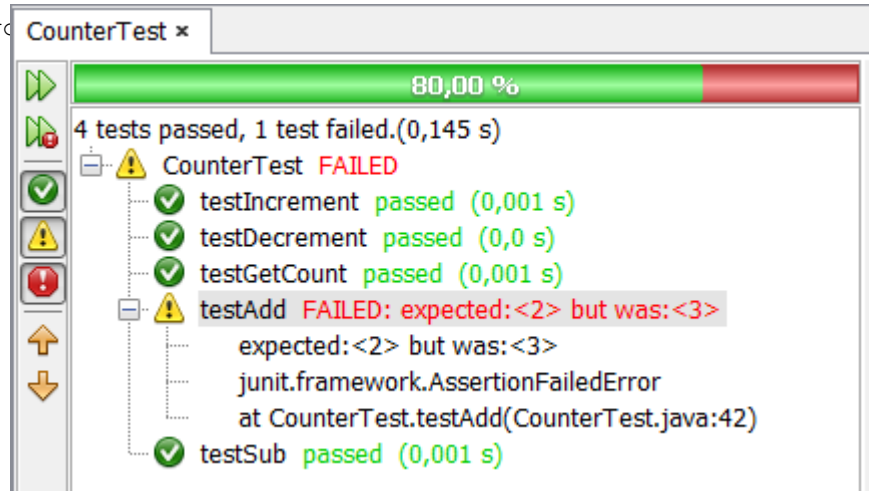
**Affiche sur la console le résultat de tous les tests contenus dans la classe CounterTest**

**JUnit 4 ne propose plus en standard de vue graphique (Green Bar)**

**nombre de méthodes de test exécutées**

**nombre de méthodes de test ayant échoué**
- **assertion non vérifiée**
- **méthode `fail` exécutée**
- **exception levée**

**De nombreux IDE intègrent JUnit : NetBeans, Eclipse…**



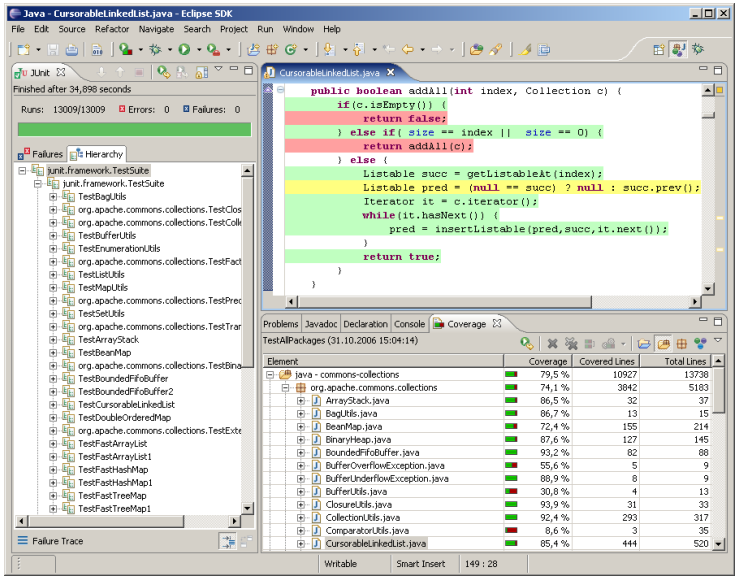
# Couverture de code

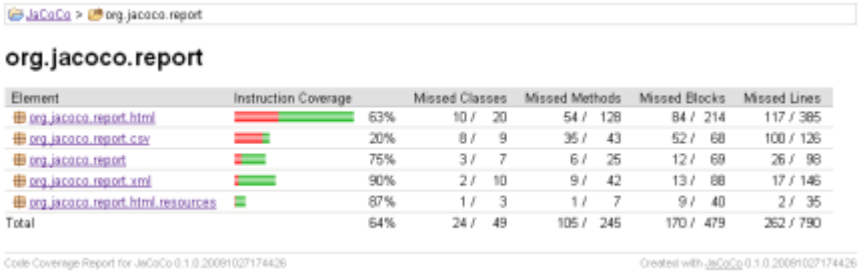
- comment mesurer la qualité des test effectués ?
- un indicateur de la qualité des tests effectués peut être la **couverture de code** (en anglais : **code coverage**)
  - *une mesure utilisée en génie logiciel pour décrire le taux de code source testé d'un programme.*
  - *nombreuses méthodes pour mesurer la couverture de code2. Les principales sont :*
    - *Couverture des fonctions (Function Coverage) - Chaque fonction dans le programme a-t-elle été appelée ?*
    - *Couverture des instructions (Statement Coverage) - Chaque ligne du code a-t-elle été exécutée et vérifiée ?*
    - *Couverture des points de tests (Condition Coverage) - Chaque point d'évaluation (tel que le test d'une variable) a-t-il été exécuté et vérifié ? (Le point de test teste-t-il ce qu'il faut ?)*
    - *Couverture des chemins d'exécution (Path Coverage) - Chaque parcours possible (par exemple les 2 cas vrai et faux d'un test) a-t-il été exécuté et vérifié ?*

# Couverture de code

- Très souvent en complément d'un framework de tests unitaires on utilise des outils de couverture de code
  - *JaCoCo Java Code Coverage Library* http://www.eclemma.org/jacoco/
    - Intégration dans les IDE

**EclEmma for Eclipse** http://www.eclemma.org/




**TikiOne JaCoCoverage plugin for Netbeans**
https://github.com/jonathanlermitage/tikione-jacocoverage

  - *d'autres outils Java*
    - Cobertura http://cobertura.github.io/cobertura/
    - Clover https://www.atlassian.com/software/clover
    - ....

# JUnit 4 Exécuter un ensemble de tests

- définition d'une suite de tests

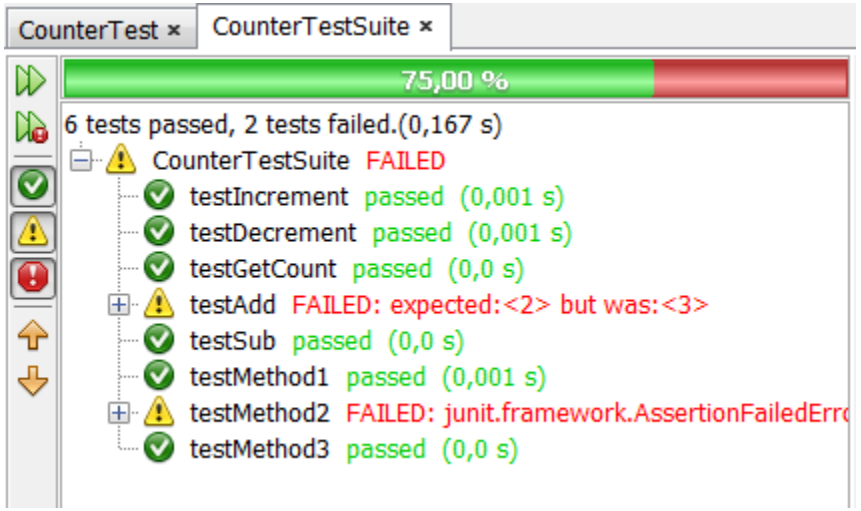
**CounterTestSuite.java**

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

@RunWith(Suite.class)
@Suite.SuiteClasses({CounterTest1.class, CounterTest2.class})
public class CounterTestSuite {
}
```

**classes dont tous les test sont à exécuter**
**ces classes peuvent elles-mêmes être des suites de de tests**

```
$ java org.junit.runner.JUnitCore CounterTestSuite
JUnit version 4.10
....E...E.
Time: 0,01
There were 2 failures:
1) testAdd(CounterTest1)
java.lang.AssertionError: expected:<2> but was:<3>
              at org.junit.Assert.fail(Assert.java:93)
              ...
              at CounterTest1.testAdd(CounterTest1.java:42)
              ...
              at Essai.main(Essai.java:14)
2) testMethod2(CounterTest2)
java.lang.AssertionError
              at org.junit.Assert.fail(Assert.java:92)
              at org.junit.Assert.assertTrue(Assert.java:43)
              at org.junit.Assert.assertTrue(Assert.java:54)
              at CounterTest2.testMethod2(CounterTest2.java:40)
              ...
              at Essai.main(Essai.java:14)

FAILURES!!!
Tests run: 8,  Failures: 2
```

**Exécution de la suite dans NetBeans**



# JUnit Testing tips (JUnit primer)

- Code a little, test a little, code a little, test a little . . .
- Run your tests as often as possible, at least as often as you run the compiler ☺
- Begin by writing tests for the areas of the code that you're the most worried about . . .write tests that have the highest possible return on your testing investment
- When you need to add new functionality to the system, write the tests first
- If you find yourself debugging using **`System.out.println()`** , write a test case instead
- When a bug is reported, write a test case to expose the bug
- Don't deliver code that doesn't pass all the tests

- **Integration continue** – Pendant le développement, le programme ***marche toujours*** – peut être qu'il ne fait pas tout ce qui est requis mais ce qu'il fait il le fait bien.

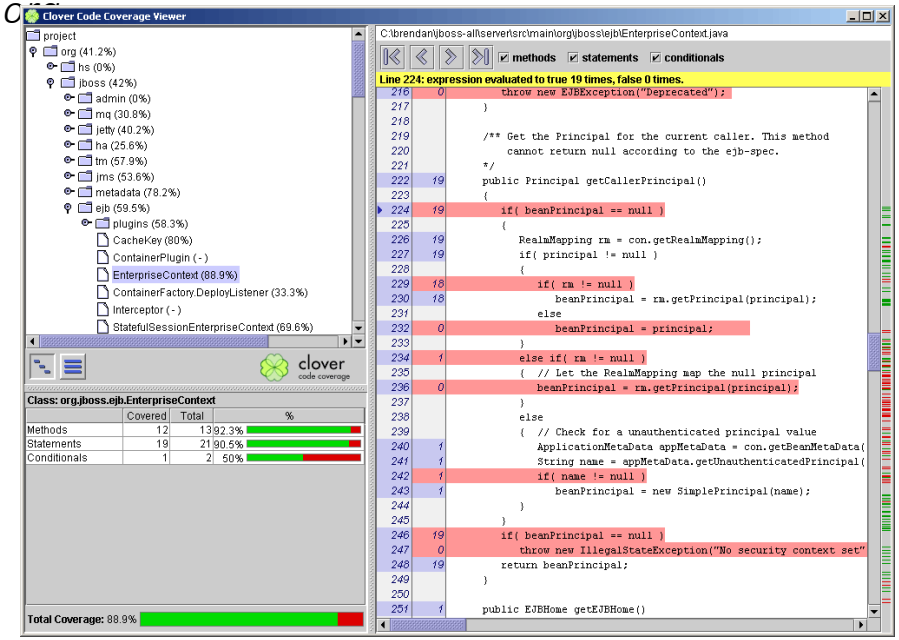
www.extremeprogramming.org

- TDD Test Driven Development :pratique qui se rattache à Xtreme Programming

*"Any program feature without an automated test simply doesn't exist."* **from Extreme Programming Explained, Kent Beck**

# JUnit et les autres

- xUnit famille de frameworks pour le test unitaire automatisés.
  - *Disponibles pour de nombreux langages et environements:*
    - *JUnit. (Java) http://www.junit.org.*
    - *TestNG (java) http://testng.org/doc/index.html*
      *http://kaczanowscy.pl/tomek/sites/default/files/testng_vs_junit.txt.slidy_.html#%281%29*
    - *cppUnit. (C++).*
    - *nUnit. (.NET)*
    - *dbUnit. (base de données testing). http://www.dbunit.org*
    - *HTTPUnit. (sites web) http://www.httpunit.org*
    - *…*
- D'autres outils
  - *Cactus*
    - *http://jakarta.apache.org/cactus*
  - *Clover, JaCoCo, …: couverture de code*
  - *JMeter (mesure performance appli web)*
    - *http://jakarta.apache.org/jmeter*
  - *SeleniumHQ (applications Web)*
    - *http://seleniumhq.org/…*
  - *…*



# Références

- **Le site JUnit** : www.junit.org
- **Jump Into JUnit 4**, Andrew Glover, http://www-128.ibm.com/developerworks/edu/j-dw-java-junit4.html
- **Get Acquainted with the New Advanced Features of JUnit 4**, Antonio Goncalves http://www.devx.com/Java/Article/31983/1954?pf=true
- www.extremeprogramming.org
- **eXtreme Programming Explained: Embrace Change & Test Driven Design**
  - *Kent Beck*
- **Java Tools for Extreme Programming**
  - *Rick Hightower and Nick Lesiecki (Wiley, 2001)*