

**Chap 4**

# Tests unitaires : JUnit

SupNum, DS12

M. Demba

# Test ?

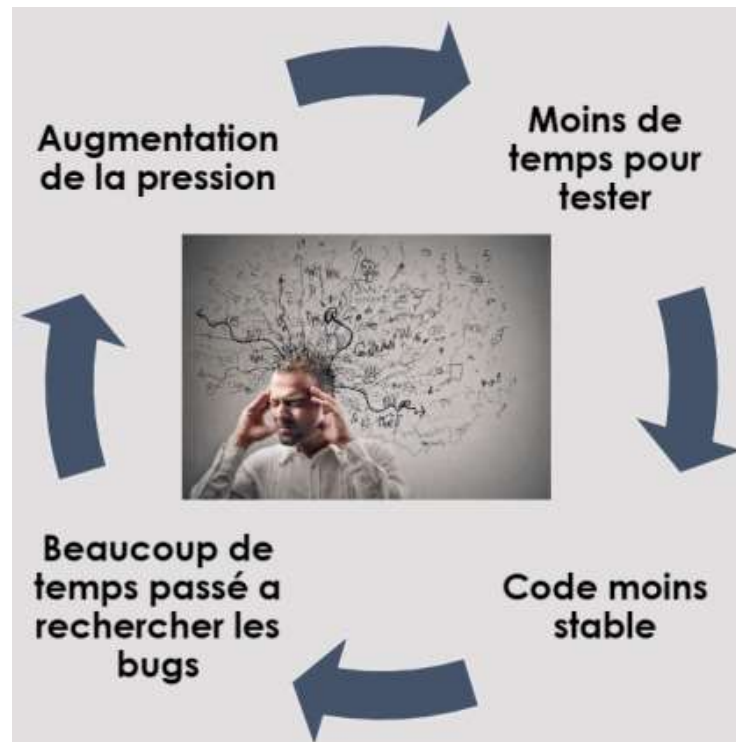
## Qu'est-ce que le test ?

- Procédure qui vise à identifier les comportements problématiques d'un programme afin d'en augmenter la qualité
- Vérifier par l'exécution
- Confronter la réalisation aux
  - Spécifications
  - Exigences
- Doit répondre à la question:  
La réalisation fait-elle bien
  - Ce qui devait être implémenté ?
  - Ce que voulait le client ?
- Permet de statuer sur le succès ou l'échec d'une vérification

## Pourquoi tester ?

- Pour gagner du temps
  - Gain du temps passé à debugger
- Pour valider le logiciel
- Pour valider les remaniements du code
- Pour détecter les régressions

## Paradoxe du test !



## « Testing is the process of executing a program with the intent of finding errors »

- ❑ Une expérience d'exécution, pour mettre en évidence un défaut ou une erreur
  - ✓ **Diagnostic** : quel est le problème
  - ✓ Besoin d'un **oracle**, qui indique si le résultat de l'expérience est conforme aux intentions
  - ✓ Localisation (si possible) : où est la cause du problème ?
    - ♣ Les tests doivent mettre en évidence des erreurs
    - ♣ On ne doit pas vouloir démontrer qu'un programme marche à l'aide de tests !
- ❑ Souvent négligé car :
  - les chefs de projet n'investissent pas pour un résultat négatif
  - les développeurs ne considèrent pas les tests comme un processus destructeur

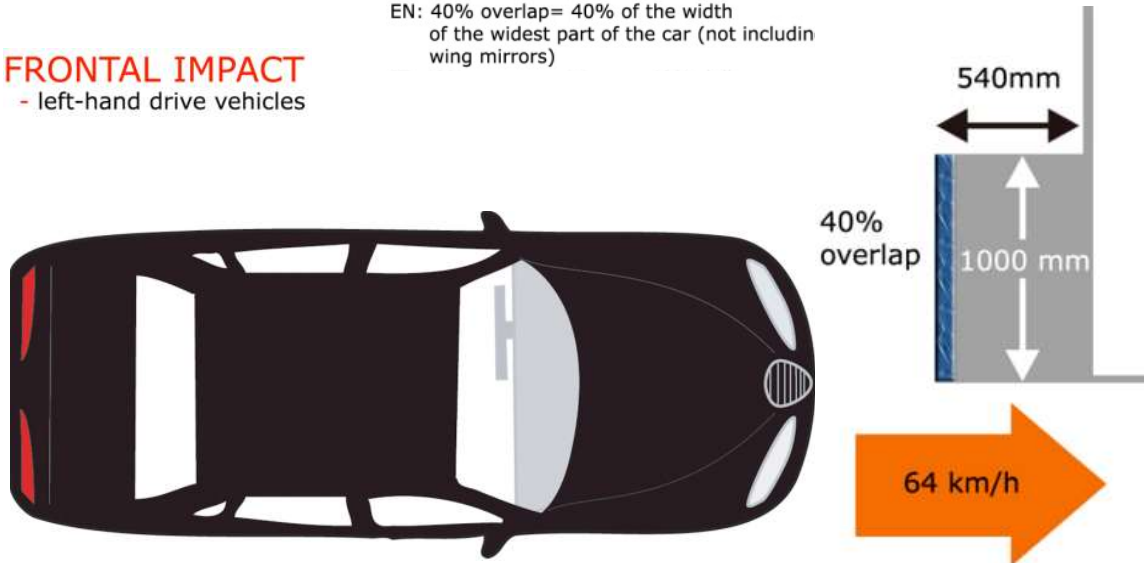


# Le test, c'est du sérieux !

## FRONTAL IMPACT

- left-hand drive vehicles

EN: 40% overlap= 40% of the width of the widest part of the car (not including wing mirrors)



Un test : des données de test



## Un test : des oracles

### ADULT OCCUPANT

Total 35 pts | 97%

#### FRONTAL IMPACT

15,4 pts



Driver



Passenger

#### SIDE IMPACT CAR

8 pts

#### SIDE IMPACT POLE

7,9 pts



Car



Pole

#### REAR IMPACT (WHIPLASH)

3,4 pts



|                                                                                                                       |          |
|-----------------------------------------------------------------------------------------------------------------------|----------|
| <span style="display:inline-block; width:15px; height:15px; background-color:green; border:1px solid black;"></span>  | GOOD     |
| <span style="display:inline-block; width:15px; height:15px; background-color:yellow; border:1px solid black;"></span> | ADEQUATE |
| <span style="display:inline-block; width:15px; height:15px; background-color:orange; border:1px solid black;"></span> | MARGINAL |
| <span style="display:inline-block; width:15px; height:15px; background-color:brown; border:1px solid black;"></span>  | WEAK     |
| <span style="display:inline-block; width:15px; height:15px; background-color:red; border:1px solid black;"></span>    | POOR     |

#### FRONTAL IMPACT

##### HEAD

|                          |        |
|--------------------------|--------|
| Driver airbag contact    | stable |
| Passenger airbag contact | stable |

##### CHEST

|                                   |        |
|-----------------------------------|--------|
| Passenger compartment             | stable |
| Windscreen Pillar rearward        | 4mm    |
| Steering wheel rearward           | none   |
| Steering wheel upward             | none   |
| Chest contact with steering wheel | none   |

##### UPPER LEGS, KNEES AND PELVIS

|                               |      |
|-------------------------------|------|
| Stiff structures in dashboard | none |
| Concentrated loads on knees   | none |

##### LOWER LEGS AND FEET

|                         |              |
|-------------------------|--------------|
| Footwell Collapse       | none         |
| Rearward pedal movement | brake - 11mm |
| Upward pedal movement   | none         |

#### SIDE IMPACT

|                         |     |
|-------------------------|-----|
| Head protection airbag  | Yes |
| Chest protection airbag | Yes |

#### WHIPLASH

|                      |                             |
|----------------------|-----------------------------|
| Seat description     | Standard cloth 6 way manual |
| Head restraint type  | Reactive                    |
| Geometric assessment | 1 pts                       |

##### TESTS

|                   |         |
|-------------------|---------|
| - High severity   | 2,5 pts |
| - Medium severity | 2,5 pts |
| - Low severity    | 2,4 pts |

Tester une **unité de code**:

- composant, module, classe, méthode

Vérifier un **comportement**:

- cas normaux
- cas limites
- cas anormaux

# JUnit

Un [framework de test unitaire pour Java](#)

Open Source: [www.junit.org](http://www.junit.org)

- K. Beck & E. Gamma (1998)

Intégré à la plupart des IDEs Java:

- BlueJ
- Eclipse
- NetBeans
- IntelliJ

**En 2013, une étude menée sur les projets hébergés sur GitHub montre que 31% des projets utilisent JUnit**

# Série xUnit

Port dans de nombreux langages:

- FlexUnit      ActionScript
- NUnit        C#
- HUnit        Haskell
- JUnit        JavaScript
- PHPUnit      PHP
- PyUnit      Python
- ...

Cas de test (*test case*)

- test d'une méthode dans un contexte particulier

Suite de test (*test suite*)

- ensemble de cas de tests

Test unitaire (*unit test*)

- plus petite unité à tester

## Exhaustivité du test

- ✓ Impossible car
  - Nombre d'entrées valides démesurément grand, voire infini
  - Nombre d'entrées possibles non valides essentiellement infini
- ✓ Choix d'une stratégie
  - Test "boîte noire"
  - Test "boîte blanche"

## Tests "boîte blanche vs boîte noire "

### Tests "boîte noire "

- ✓ Principe
  - Tests "dirigés par les données"
  - Aucune connaissance du code

### Tests "boîte blanche"

- ✓ **Principe**
  - On connaît la structure interne du programme
  - Tenter un test exhaustif des chemins dans le code
- ✓ **Conséquence**
  - Test exhaustif impossible (pb des boucles)
  - On ne teste pas les cas oubliés par le développeur
  - Certains cas choisis pour les branches ne sont pas représentatifs de toutes les exécutions possibles

```
if (a && b){  
    System.out.print("c'est bon");  
}
```

# Principes du test de logiciel: The Art of Software Testing

## Glenford J. Myers

1. Une partie nécessaire d'un cas de test est la définition du résultat attendu

- ✓ L'humain voit plus ce qu'il voudrait voir que ce qu'il voit
- ✓ => Être précis dans les descriptions
- ✓ Un cas de test = description précise
  - Des données d'entrée
  - Du résultat attendu

2. Un programmeur devrait éviter de tester son propre code

- ✓ Analogie quand on se relie, on ne voit pas ses erreurs (syntaxe, grammaire) même évidentes. De même, on comprend forcément ce qu'on a voulu dire
- ✓ Test = activité destructrice.
  - On n'aime pas détruire ce qu'on a construit

# Principes du test de logiciel: The Art of Software Testing

## Glenford J. Myers

3. Une entreprise de développement de logiciels devrait éviter de tester ses propres programmes

- ✓ Une entreprise doit
  - Économiser du temps
  - Économiser de l'argent
- ✓ Or le test
  - Prend du temps
  - Est coûteux
- ✓ => L'entreprise n'est pas neutre dans le processus de test

4. Vérifier minutieusement les résultats de chaque test

On "loupe" des erreurs en regardant trop vite le résultats des tests

# Principes du test de logiciel: The Art of Software Testing

## Glenford J. Myers

5. Les cas de tests doivent être écrits pour des conditions d'entrées invalides ou inattendues, aussi bien que pour celles qui sont valides et attendues
  - ✓ On oublie de tester les cas "invalides"
  - ✓ La plupart des erreurs détectées après la sortie officielle du logiciel sont liées à des cas d'utilisation non prévues au départ
  - ✓ Les tests sur entrées invalides ont beaucoup plus de chance de détecter des erreurs
6. Examiner un programme pour voir s'il ne fait pas ce qui devrait ne constitue que la moitié de la bataille ; l'autre moitié consiste à voir si le programme fait ce qu'il n'est pas supposé faire.
7. Eviter de jeter les cas de tests, sauf si le programme est lui-même jeté.
  - ✓ Concevoir un cas de test prend du temps
    - Noter tous les tests que l'on fait/envisage
  - ✓ Garder les tests pour les prochaines versions et éviter la "régression"

## **Principes du test de logiciel: The Art of Software Testing**

### **Glenford J. Myers**

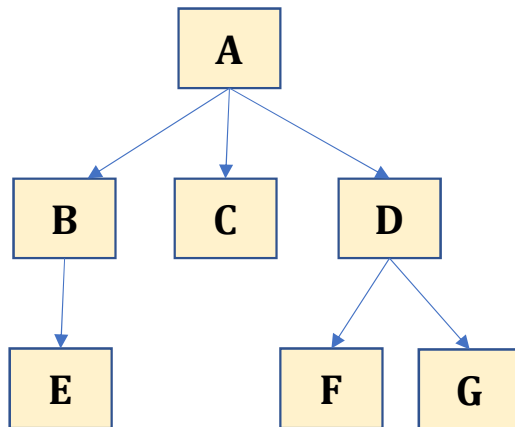
8. Ne pas planifier des tests en faisant implicitement l'hypothèse qu'aucune erreur ne sera trouvée
9. Le risque de trouver une erreur dans une partie du code est proportionnel au nombre d'erreurs déjà trouvées dans cette partie.
  - ✓ Les erreurs sont groupées en paquets
  - ✓ Explication possible :
    - Les erreurs sont groupées dans les parties les plus compliquées
    - Les parties simples sont exemptes d'erreurs
10. Trouver des bons cas de test est certainement plus créatif qu'écrire un programme

## Check-list Flux d'exécution

- ✓ Les boucles se terminent-elles toujours ?
- ✓ Un programme/fonction se termine-t-il ?
- ✓ Est-il possible qu'on ne rentre jamais dans le corps d'une boucle ? Est-ce normal ?
- ✓ Quand on peut sortir d'une boucle soit après un nombre fini d'itération connu au départ, soit en cours de route, les 2 types de sortie sont-ils correctement gérés ?
- ✓ Les compteurs initialisés à zéro sont-ils correctement utilisés ?
- ✓ En cas de tests non-exhaustif, cela est-il correct ? le choix gardé par défaut se justifie-t-il ?
- ✓ Etc...

# Tests unitaires: Stratégies

- ✓ Commencer par des tests « boîte blanche »
- ✓ Compléter éventuellement par des tests « boîte noire »
- ✓ Choisir entre un *test incrémental* et un *test non-incrémental*



## Test incrémental ascendant

1. On teste E, C, F, G
2. On teste B, D en utilisant E, F, G
3. On teste A en utilisant B, C, D

## Test incrémental descendant

1. On teste A
2. On teste B, C, D en utilisant A
3. On teste E, F, G en utilisant B, D

## Tests unitaires non-incrémental

Les modules sont testés indépendamment les uns des autres

## Assertions JUnit

`fail(message)`: échoue toujours

`assertTrue(condition)`: échoue si la condition est fausse

`assertFalse(condition)`: échoue si la condition est vraie

`assertEquals(expected, actual)`: échoue si les deux objets ne sont pas égaux

Etc...

`@BeforeEach`, `@AfterEach` identifie une méthode à exécuter avant (setup) / après (tear down) chaque test.

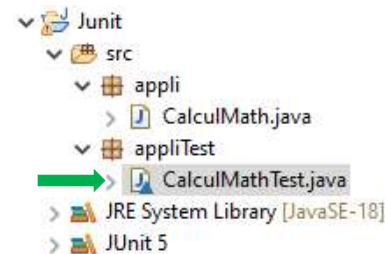
Chaque appel à une méthode de test sera précédé/succédé par l'appel à la méthode annotée par `@BeforeEach`/`@AfterEach`.

Les méthodes `setUp` et `tearDown` doivent être publiques, ne prendre aucun paramètre et ne rien renvoyer

## Factoriel(5) ?

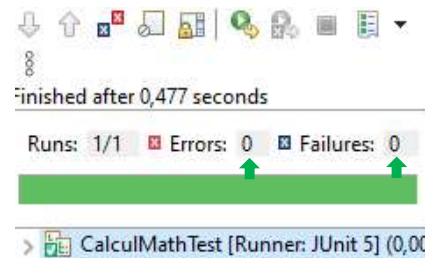
```
package appli;

public class CalculMath {
    public int fact(int val) {
        int res=val;
        for(int k=val-1;k>0;k--) {
            res*=k;
        }
        return res;
    }
}
```



```
1 package appliTest;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 import org.junit.jupiter.api.Test;
6
7 import appli.CalculMath;
8
9 class CalculMathTest {
10
11     CalculMath c=new CalculMath();
12
13     @Test
14     void testFact() {
15         int fact=120;
16         assertEquals(fact,c.fact(5));
17     }
18 }
```

## Execution



```
1 package appliTest;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 import org.junit.jupiter.api.Test;
6
7 import appli.CalculMath;
8
9 class CalculMathTest {
10
11     CalculMath c=new CalculMath();
12
13     @Test
14     void testFact() {
15         int fact=120;
16         assertEquals(fact,c.fact(5));
17     }
18 }
```

## Factoriel(0) ?

```
package appli;
```

```
public class CalculMath {  
    public int fact(int val) {  
        int res=val;  
        for(int k=val-1;k>0;k--) {  
            res*=k;  
        }  
        return res;  
    }  
}
```

*Le test ne passe pas*

## Execution

Finished after 0,245 seconds

Runs: 1/1 Errors: 0 Failures: 1

CalculMathTest [Runner: JUnit 5] (0,05 s)

testFact() (0,053 s)

```
1 package appliTest;  
2  
3 import static org.junit.jupiter.api.Assertions.*;  
4  
5 import org.junit.jupiter.api.Test;  
6  
7 import appli.CalculMath;  
8  
9 class CalculMathTest {  
10     CalculMath c=new CalculMath();  
11  
12  
13 @Test  
14 void testFact() {  
15     int fact=1;  
16     assertEquals(fact,c.fact(0));  
17 }  
18 }
```

## Correction

```
package appli;
```

```
public class CalculMath {  
    public int fact(int val) {  
        int res=1;  
        for(int k=val;k>0;k--) {  
            res*=k;  
        }  
        return res;  
    }  
}
```

*Le test passe*

Finished after 0,226 seconds

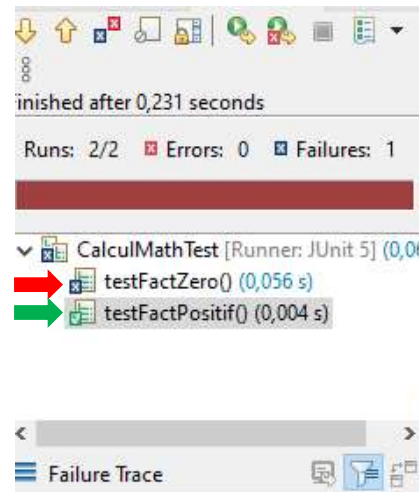
Runs: 1/1 Errors: 0 Failures: 0

CalculMathTest [Runner: JUnit 5] (0,03 s)

```
1 package appliTest;  
2  
3 import static org.junit.jupiter.api.Assertions.*;  
4  
5 import org.junit.jupiter.api.Test;  
6  
7 import appli.CalculMath;  
8  
9 class CalculMathTest {  
10     CalculMath c=new CalculMath();  
11  
12  
13 @Test  
14 void testFact() {  
15     int fact=1;  
16     assertEquals(fact,c.fact(0));  
17 }  
18 }
```

```
package appli;
```

```
public class CalculMath {
    public int fact(int val) {
        int res=val;
        for(int k=val-1;k>0;k--) {
            res*=k;
        }
        return res;
    }
}
```



```
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
import appli.CalculMath;

class CalculMathTest {

    CalculMath c=new CalculMath();

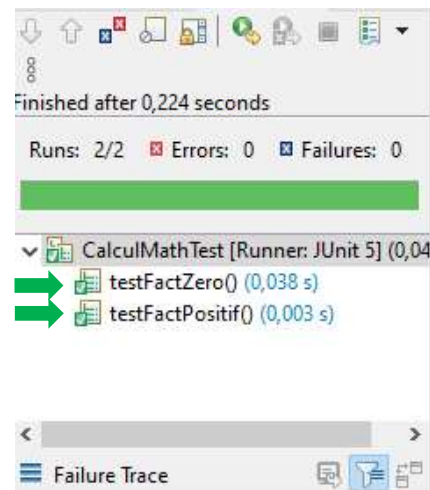
    @Test
    void testFactPositif() {    // pour les valeurs >0
        int fact=120;
        assertEquals(fact,c.fact(5));
    }

    @Test
    void testFactZero() {    // pour zero
        int fact=1;
        assertEquals(fact,c.fact(0));
    }
}
```

2 fonctions à tester

```
package appli;
```

```
public class CalculMath {
    public int fact(int val) {
        int res=1;
        for(int k=val;k>0;k--) {
            res*=k;
        }
        return res;
    }
}
```



```
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
import appli.CalculMath;

class CalculMathTest {

    CalculMath c=new CalculMath();

    @Test
    void testFactPositif() {    // pour les valeurs >0
        int fact=120;
        assertEquals(fact,c.fact(5));
    }

    @Test
    void testFactZero() {    // pour zero
        int fact=1;
        assertEquals(fact,c.fact(0));
    }
}
```

Les annotations sont à placer avant les méthodes d'une classe de tests unitaires

| Annotation  | Description                                   |
|-------------|-----------------------------------------------|
| @Test       | méthode de test                               |
| @BeforeEach | méthode exécutée <i>avant chaque test</i>     |
| @AfterEach  | méthode exécutée <i>après chaque test</i>     |
| @BeforeAll  | méthode exécutée <i>avant le premier test</i> |
| @AfterAll   | méthode exécutée <i>après le dernier test</i> |
| @Disabled   | méthode qui ne sera pas lancée comme test     |

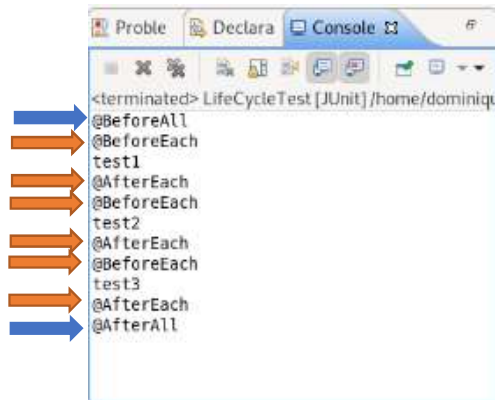
### Ordre d'exécution

- 1 La méthode annotée @BeforeAll
- 2 Pour chaque méthode annotée @Test (ordre indéterminé)
  - 1 Les méthodes annotées @Before (ordre indéterminé)
  - 2 La méthode annotée @Test
  - 3 Les méthodes annotées @After (ordre indéterminé)
- 3 La méthode annotée @AfterAll

## Les annotations @BeforeEach, @AfterEach, @BeforeAll et @AfterAll

La méthode annotée *@BeforeEach/@AfterEach*, si elle est définie, est invoquée *avant/après* l'exécution de chaque méthode de test

Voici les résultats produits par cette classe de test



```
class LifeCycleTest {  
  
    @BeforeAll  
    static void setUpBeforeClass() throws Exception {  
        System.out.println( "@BeforeAll" );  
    }  
  
    @BeforeEach  
    void setUp() throws Exception {  
        System.out.println( "@BeforeEach" );  
    }  
  
    @Test  
    public void test1() {  
        System.out.println( "test1" );  
    }  
  
    @Test  
    public void test2() {  
        System.out.println( "test2" );  
    }  
  
    @Test  
    public void test3() {  
        System.out.println( "test3" );  
    }  
  
    @AfterEach  
    void tearDown() throws Exception {  
        System.out.println( "@AfterEach" );  
    }  
  
    @AfterAll  
    static void tearDownAfterClass() throws Exception {  
        System.out.println( "@AfterAll" );  
    }  
  
}
```

Une classe qui permet de manipuler un compteur.

```
package Counter;

public class Compteur {
    int valeur;

    Compteur(){
        valeur=0;
    }
    Compteur(int val){
        valeur=val;
    }

    int Incrementer() {
        valeur++;
        return valeur;
    }

    int decremener() {
        valeur--;
        return valeur;
    }
    int getValeur() {
        return valeur;
    }
}
```

la classe dédiée pour tester  
la classe compteur?



```
package Counter;

public class Compteur {
    Compteur(){ }
    Compteur(int val){ }
    int Incrementer() { }
    int decremener() { }
    int getValeur() { }
}
```



```
import org.junit.jupiter.api.Test;
import static org.junit.Assert.*;

public class CompteurTest {
}
```

```
package TestCounter;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
import Counter.Compteur;
```

```
class CompteurTest {
```

```
    @Test
```

```
    void testIncrement() {
```

```
        Compteur c1=new Compteur();
```

```
        c1.Incrementer();
```

```
        int attendue=1;
```

```
        assertTrue(attendue==c1.getValeur());
```

```
    }
```

```
    @Test
```

```
    void testDecrement() {
```

```
        Compteur c2=new Compteur(5);
```

```
        c2.Incrementer();
```

```
        c2.decrementer();
```

```
        int attendue=5;
```

```
        assertTrue(attendue==c2.getValeur());
```

```
    }
```

```
}
```

Partie commune  
de chaque test

```
package TestCounter;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import Counter.Compteur;
```

```
class CompteurTest2 {
```

```
    private Compteur c1,c2;
```

```
    private int attendue;
```

```
    @BeforeEach
```

```
    void setUp() throws Exception {
```

```
        c1=new Compteur();
```

```
        c2=new Compteur(5);
```

```
    }
```

```
    @Test
```

```
    void testIncrement() {
```

```
        c1.Incrementer();
```

```
        attendue=1;
```

```
        assertTrue(attendue==c1.getValeur());
```

```
    }
```

```
    @Test
```

```
    void testDecrement() {
```

```
        c2.Incrementer();
```

```
        c2.decrementer();
```

```
        attendue=5;
```

```
        assertTrue(attendue==c2.getValeur());
```

```
    }
```

```
}
```

*Test Driven Development*

## La couverture du code

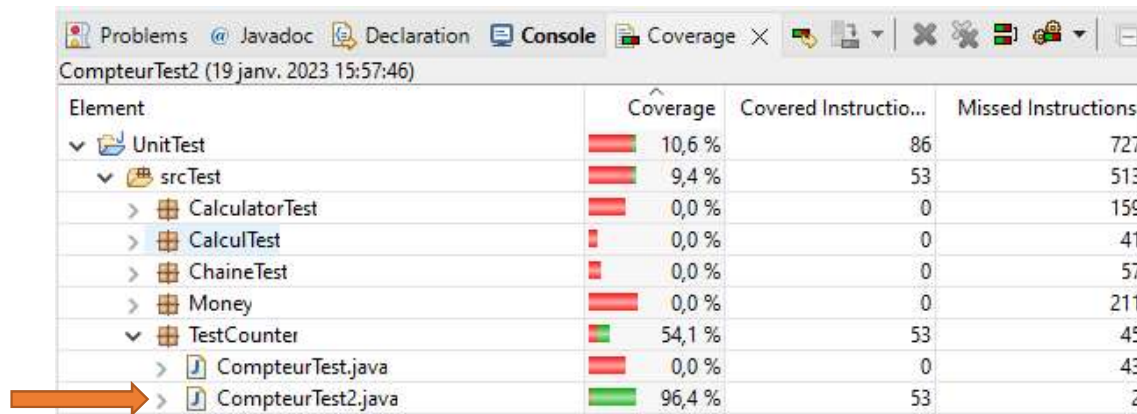
The screenshot shows an IDE interface with the following components:

- Menu Bar:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help. An orange arrow points to the **Search** menu.
- Toolbar:** Contains icons for file operations, running, and debugging.
- Package Explorer:** Shows the project structure with 'JUnit' selected.
- JUnit Console:** Displays the test results: 'Finished after 0,471 seconds', 'Runs: 2/2', 'Errors: 0', and 'Failures: 0'. A green progress bar indicates 100% success.
- Code Editor:** Shows the source code for `CompteurTest2.java`. The code is as follows:

```
1 import static org.junit.jupiter.api.Assertions.*;
2 import org.junit.jupiter.api.BeforeEach;
3 import org.junit.jupiter.api.Test;
4 import Counter.Compteur;
5
6
7 class CompteurTest2 {
8
9     private Compteur c1,c2;
10    private int attendue;
11
12    @BeforeEach
13    void setUp() throws Exception {
14        c1=new Compteur();
15        c2=new Compteur(5);
16    }
17
18
19    @Test
20    void testIncrement() {
21        c1.Incrementer();
22        attendue=1;
23
24        assertTrue(attendue==c1.getValeur());
25    }
26
27    @Test
28    void testDecrement() {
29        c2.Incrementer();
30        c2.decrementer();
31        attendue=5;
32
33        assertTrue(attendue==c2.getValeur());
34    }
35 }
```
- Failure Trace:** An empty panel at the bottom left.

## Statistique sur les tests

Et voici le rapport de couverture de code pour mes packages :



| Element            | Coverage | Covered Instructio... | Missed Instructions |
|--------------------|----------|-----------------------|---------------------|
| UnitTest           | 10,6 %   | 86                    | 727                 |
| srcTest            | 9,4 %    | 53                    | 513                 |
| CalculatorTest     | 0,0 %    | 0                     | 159                 |
| CalculTest         | 0,0 %    | 0                     | 41                  |
| ChaineTest         | 0,0 %    | 0                     | 57                  |
| Money              | 0,0 %    | 0                     | 211                 |
| TestCounter        | 54,1 %   | 53                    | 45                  |
| CompteurTest.java  | 0,0 %    | 0                     | 43                  |
| CompteurTest2.java | 96,4 %   | 53                    | 2                   |

Généralement on cherche à atteindre au moins 80%, c'est le chiffre qui est généralement accepté. Malheureusement, les 80% testés sont souvent les 80% les plus simples à tester et les bugs les plus graves se cachent bien sûr dans les 20% restants.

## Un autre exemple

```
package Calculator;

public interface IntCalculator {
    int multiply(int a, int b);
    int divide(int a, int b);
    int add(int a, int b);
    int subtract(int a, int b);
}
```

```
package Calculator;
public class CalculatorImp implements IntCalculator {
```

```
    public int multiply(int a, int b) {
        return a*b;
    }
```

```
    public int add(int a, int b) {
        if(b>0) {
            for(int i=0;i<b;i++)
                a++;
        }else if(b<0){
            for(int i=0;i<-b;i++)
                a--;
        }
        return a;
    }
}
```

On a utilisé des méthodes non classiques ou trop simples pour calculer l'addition et la division et on veut les tester

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

```
    public int divide(int a, int b) {
        if(b==0)
            throw new ArithmeticException();
        boolean signe=false;
        int res=0;
        if(a<0) {
            signe=true;
            a=-a;
        }
        if(a<b) {
            res=0;
        }
        if(b<0) {
            signe=!signe;
            b=-b;
        }
        while(a>0) {
            a=subtract(a,b);
            res++;
        }
        if(signe)
            res=-res;

        return res;
    }
}
```

## La classe test

```
package CalculatorTest;

import static org.junit.Assert.assertTrue;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import Calculator.CalculatorImp;

class CalculatorImpTest {

    private CalculatorImp calc ;

    @BeforeEach
    void setUp() {
        calc = new CalculatorImp();
    }

    @Test
    void testMultiply() {
        int a, b, res;
        a = 5;
        b = 5;
        res = a * b;
        assertTrue("Le test passe",res==calc.multiply(a, b));
    }
}
```

```
@Test
void testDivide() {

    int a, b, res;
    a = 10;
    b = 2;
    res = 5;
    assertTrue(res==calc.divide(a, b));

    a = 0;
    b = 10;
    res = 0;
    assertTrue(res==calc.divide(a, b));

    a = -50;
    b = -2;
    res = 25;
    assertTrue(res==calc.divide(a, b));

    a = -21;
    b = 7;
    res = -3;
    assertTrue(res==calc.divide(a, b));

    a = 10;
    b = -2;
    res = -5;
    assertTrue(res==calc.divide(a, b));
}
```

## Résultat et statistique

```
@Test
void testAdd() {

    int a, b, res;
    a = 5;
    b = -5;
    res = a + b;
    assertTrue("a et b positif", calc.add(a, b) == res);

    a = -5;
    b = 10;
    res = a + b;
    assertTrue("a et b positif", calc.add(a, b) == res);
}
```

```
@Test
void testSubtract() {

    int a, b, res;
    a = 5;
    b = 5;
    res = a - b;
    assertTrue(res==calc.subtract(a, b));
}
```

```
9
10 class CalculatorImpTest {
11     private CalculatorImp calc ;
12     @BeforeEach
13     void setUp() {
14         calc = new CalculatorImp();
15     }
16     @Test
17     void testMultiply() {
18         int a, b, res;
19         a = 5;
20         b = 5;
21         res = a * b;
22         assertTrue("le test passe",res==calc.multiply(a, b));
23     }
24     @Test
25     void testDivide() {
26         int a, b, res;
27         a = 10;
28         b = 10;
29         res = 1;
30
31         assertTrue(res==calc.divide(a, b));
32
33         a = 0;
34         b = 10;
35         res = 0;
36         assertTrue(res==calc.divide(a, b));
37
38         a = -50;
39         b = -2;
40         res = 25;
41         assertTrue(res==calc.divide(a, b));
42     }
```

| CalculatorImp (19 janv. 2023 17:36:18) |          |                       |                     |
|----------------------------------------|----------|-----------------------|---------------------|
| Element                                | Coverage | Covered Instructio... | Missed Instructions |
| CalculatorTest                         | 94,9 %   | 168                   | 9                   |
| CalculatorImpTest.java                 | 94,9 %   | 168                   | 9                   |
| CalculatorImpTest                      | 94,9 %   | 168                   | 9                   |
| testDivide()                           | 94,2 %   | 81                    | 5                   |
| testSubtract()                         | 95,0 %   | 19                    | 1                   |
| testAdd()                              | 95,1 %   | 39                    | 2                   |
| testMultiply()                         | 95,2 %   | 20                    | 1                   |
| setUp()                                | 100,0 %  | 6                     | 0                   |