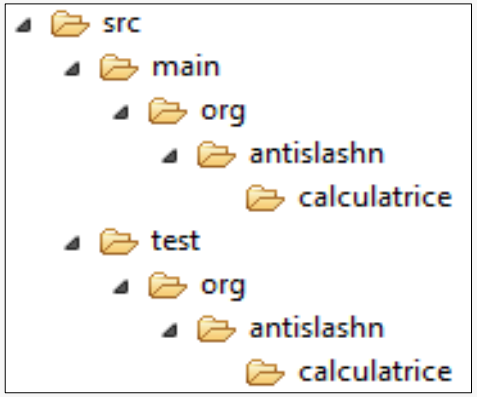
Java
JUnit

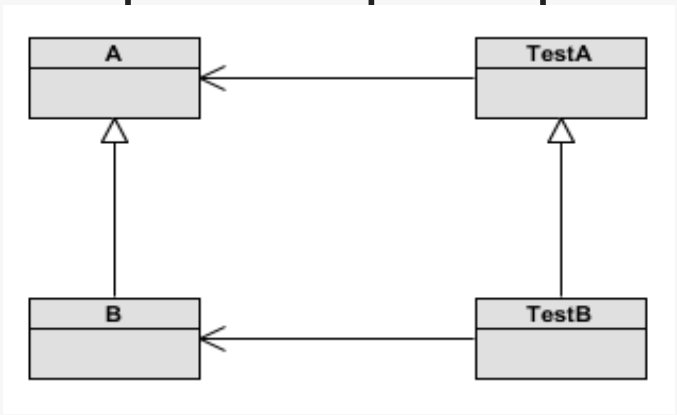
JUnit

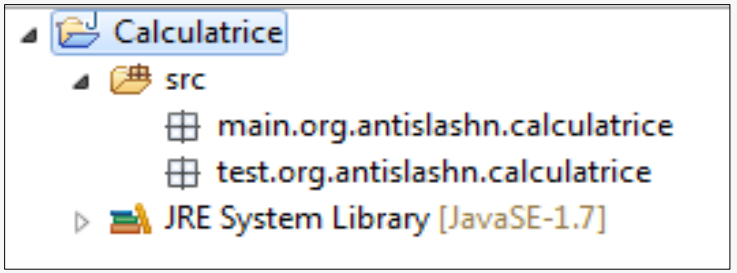
- Framework de test
  - imaginé par Kent Beck et Erich Gamma
  - framework de rédaction et exécution de tests unitaires
  - offre au développeur un environnement de développement de tests
- Chaque test unitaire est représenté par une classe
  - un test unitaire valide plusieurs méthodes d'une classe
  - un test est composé de plusieurs tests unitaires
- Site de référence : *ww.junit.org*

Structuration du code de test

- Symétrie entre le code et les tests
  - organisation des fichiers
  - organisation des packages
  - organisation des classes
    - une classe B héritant de A sera testée par sa classe de test qui ne testera que les parties spécifiques de B

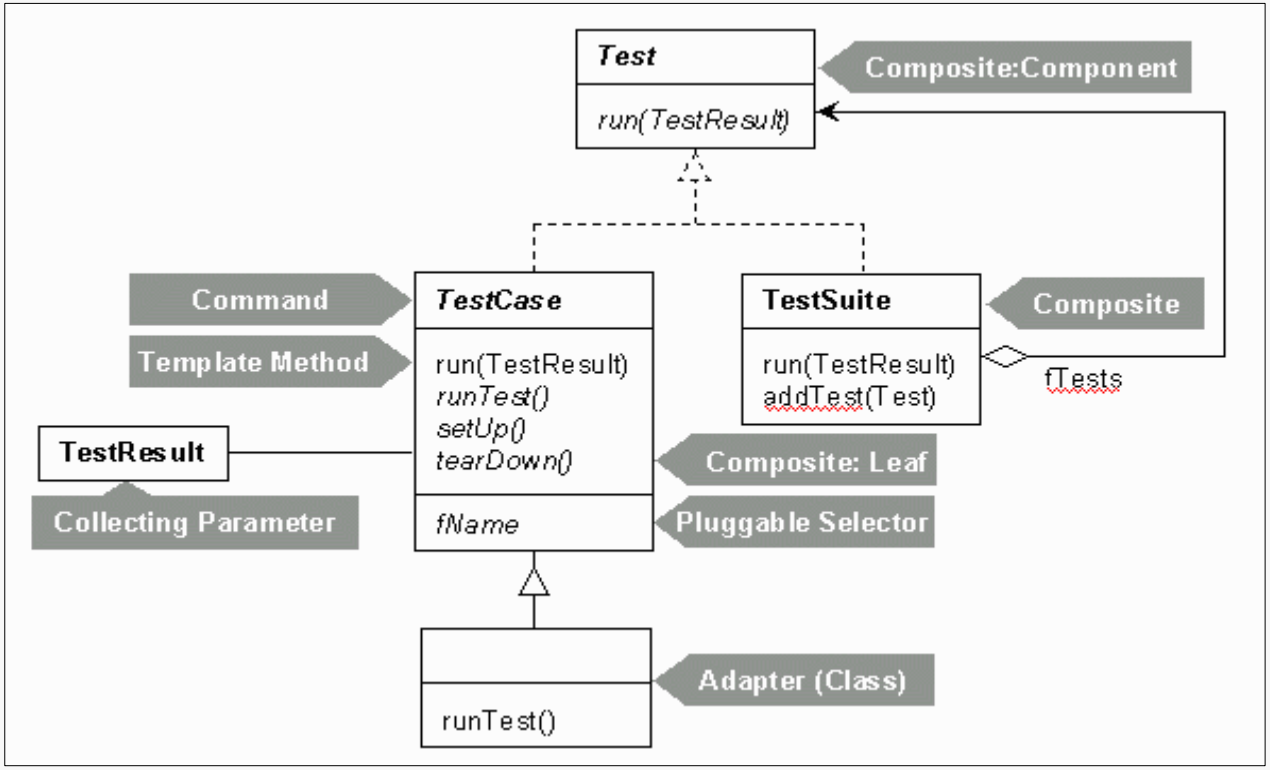






JUnit

- Architecture de JUnit
  - le développeur utilise `TestCase` et `TestSuite`
    - JUnit 4 permet d'utiliser les annotations, plutôt que l'héritage de ces classes

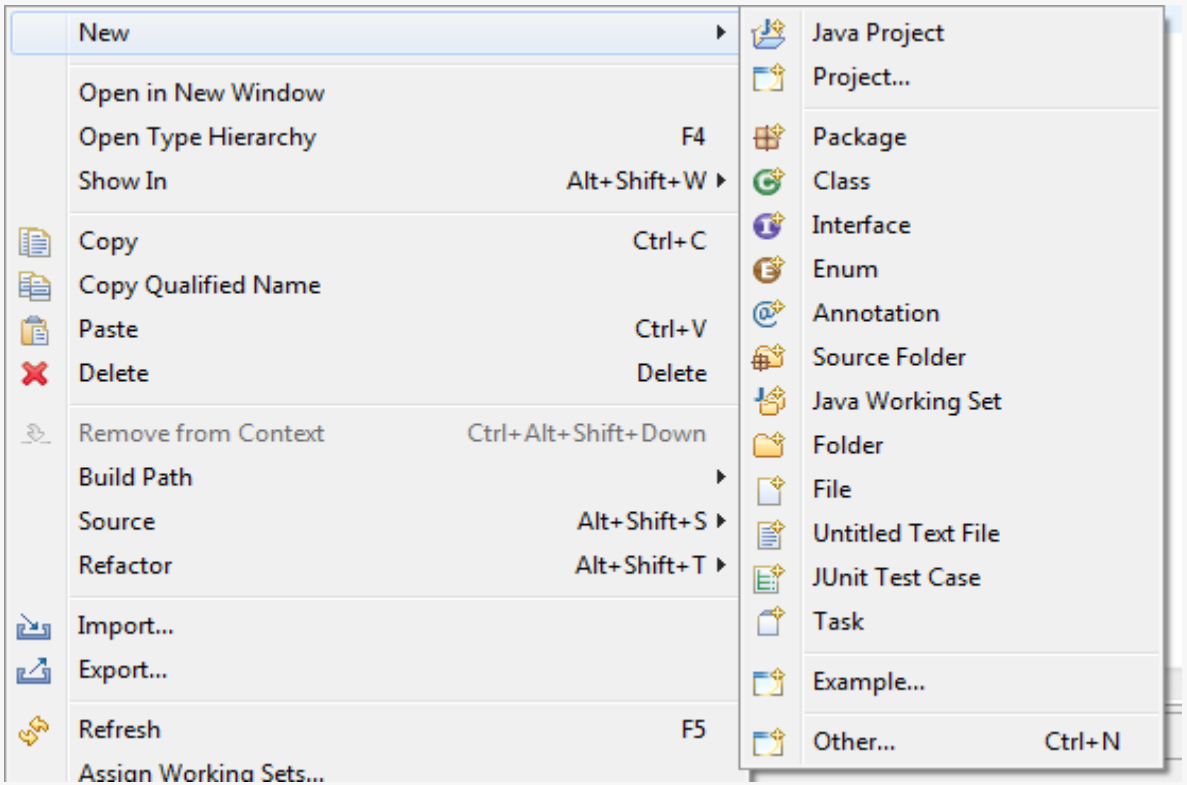


JUnit

- Pour déterminer les méthodes de test, JUnit3 recherche les méthodes `testXxxx` dans les classe dérivant de `TestCase`
- Par rapport à JUnit 3, JUnit 4 apporte
  - les annotations pour les tests
  - de nouvelles assertions
  - les suppositions
  - les tests paramétrés
  - les annotations pour les suites de tests
- Eclipse intègre JUnit

JUnit 4

- JUnit recherche les méthodes annotées par `@Test`
- Eclipse possède une assistant de création des tests
  - **File … New … JUnit Test Case**



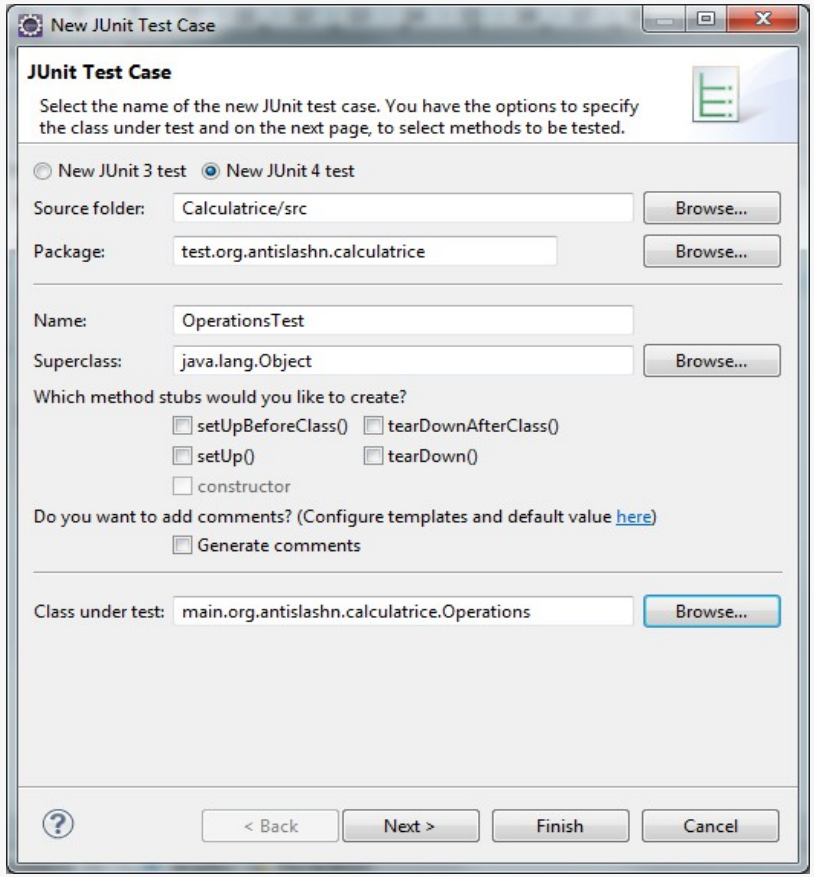
JUnit 4

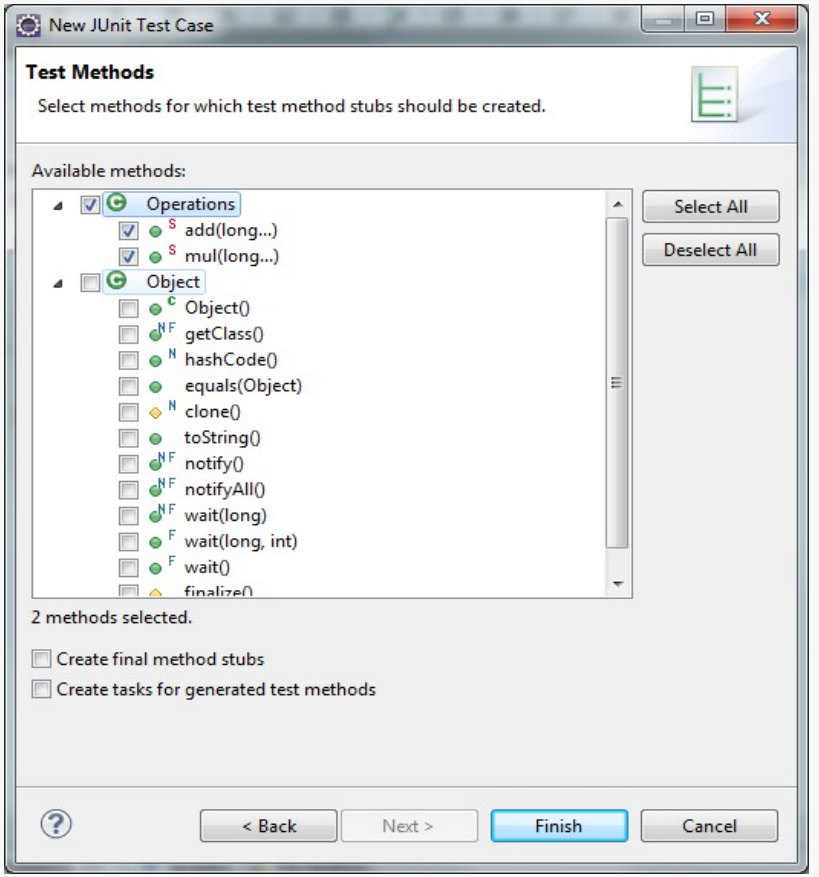
- Exemple de classe à tester
  - les méthodes ne sont pas codées

```java
package main.org.antislashn.calculatrice;

public class Operations {
    public static long add(final long ... numbers){
        return 0;
    }

    public static long mul(final long ... numbers){
        return 0;
    }
}
```

JUnit 4

- L'assistant JUnit permet
  - de choisir la classe à tester , puis ses méthodes





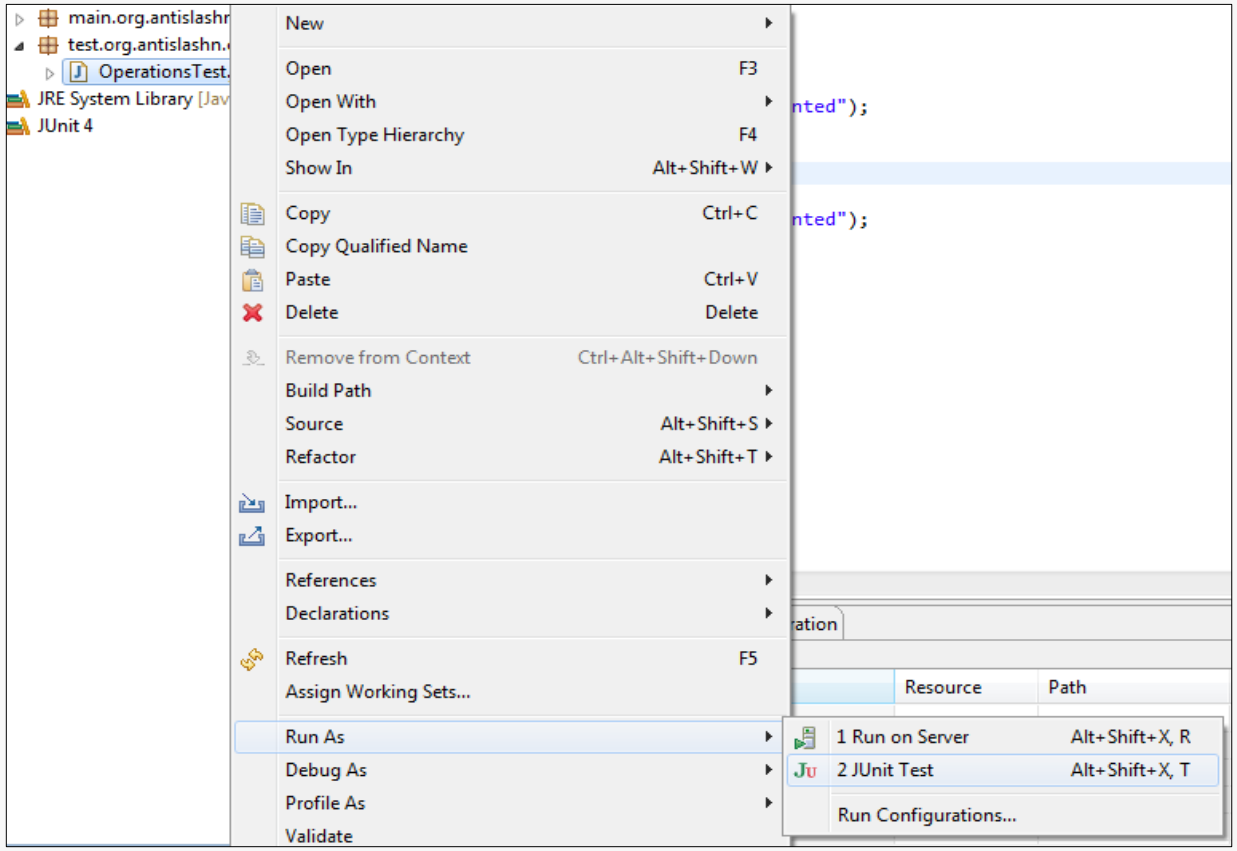
JUnit 4

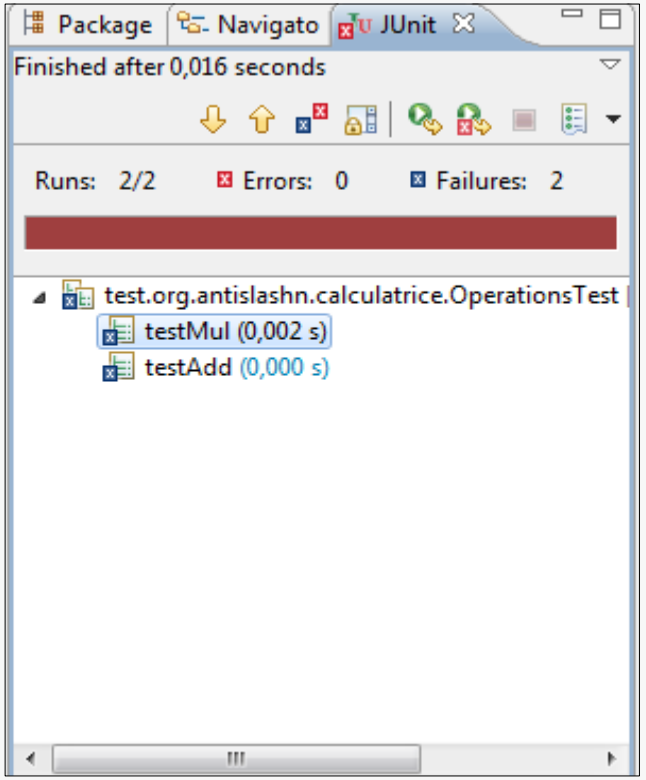
- Classe de tests générée par l'assistant
  - représente un cas de tests

```
package test.org.antislashn.calculatrice;

import static org.junit.Assert.*;

import org.junit.Test;

public class OperationsTest {

     @Test
     public void testAdd() {
          fail("Not yet implemented");
     }

     @Test
     public void testMul() {
          fail("Not yet implemented");
     }
}
```

JUnit 4

- Le test peut être directement lancé
  - le test ne passe pas encore

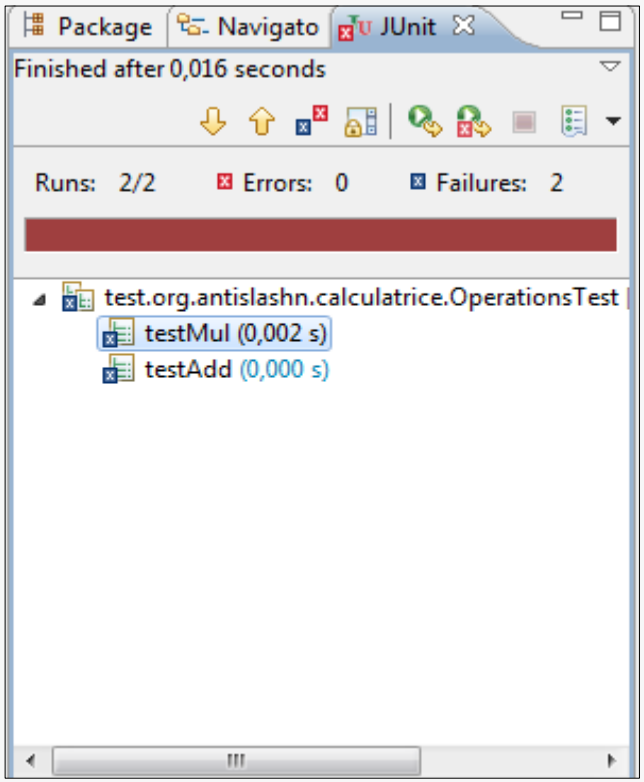




JUnit 4

- Changement du code de test
  - puis exécution du test

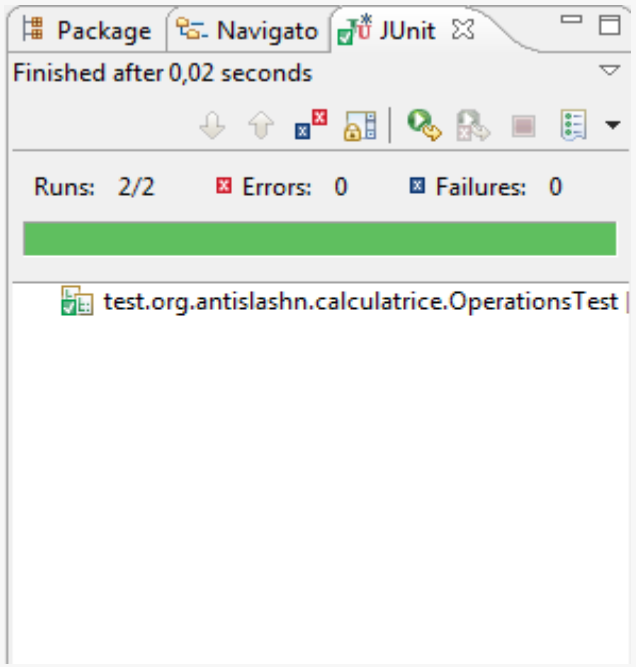
```
public class OperationsTest {

    @Test
    public void testAdd() {
        long resultat = Operations.add(10,20,30);
        Assert.assertEquals(resultat, 10+20+30);
    }

    @Test
    public void testMul() {
        long resultat = Operations.mul(10,20,30);
        Assert.assertEquals(resultat, 10*20*30);
    }
}
```



JUnit 4

- Écriture du code à tester
  - et exécution du test

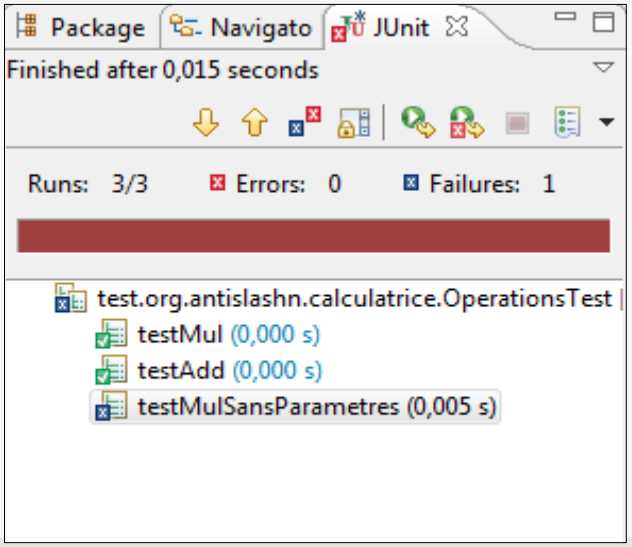
```java
public class Operations {
    public static long add(final long ... numbers){
        long result = 0;
        for(long n : numbers)
            result += n;
        return result;
    }

    public static long mul(final long ... numbers){
        long result = 0;
        for(long n : numbers)
            result *= n;
        return result;
    }
}
```



JUnit 4

- annotation `@Test`
  - paramètres optionnels
    - `expected`
      - vérifie qu'un `Throwable` a bien été déclenché
    - `timeout`
      - cause l'échec du test si le code testé ne répond pas dans la limite de temps
      - en millisecondes

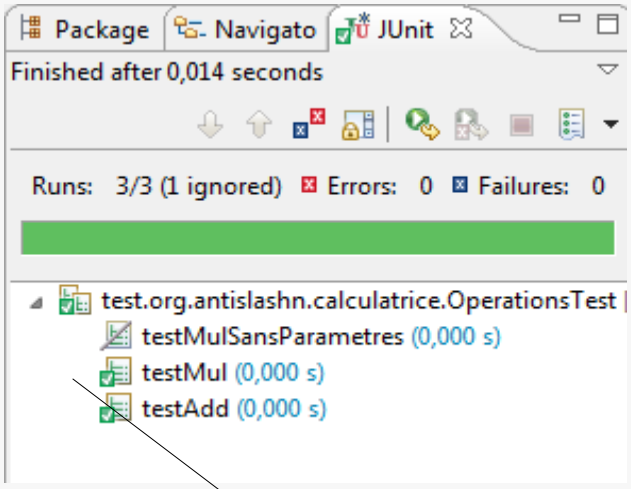
```
@Test(expected=IllegalArgumentException.class)
public void testMulSansParametres(){
    Operations.mul();
}
```



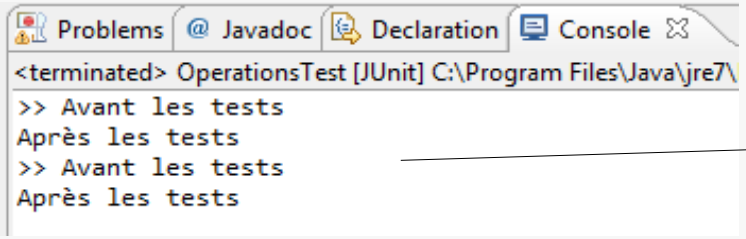
JUnit 4

- Annotations `@Before` et `@After`
  - permettent à des méthodes d'être exécutées avant et après chaque méthode de test

```
@Before
public void beforeTest(){
    System.out.println(">> Avant le test");
}

@After
public void afterTest(){
    System.out.println("Après le test");
}
```



2 tests sont exécutés



2 exécutions `@Before` et `@After`

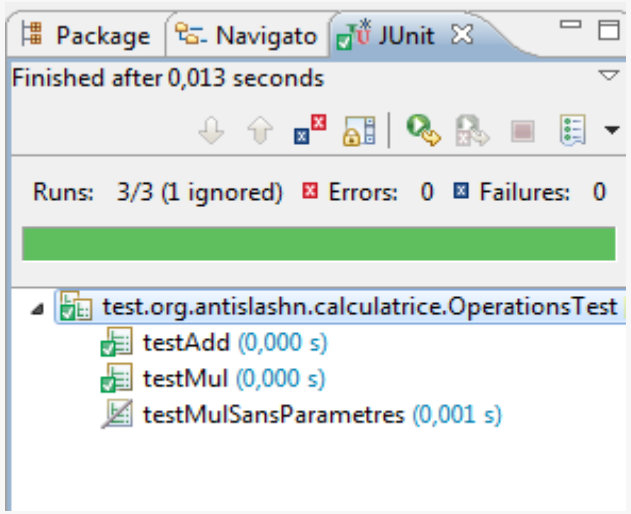
JUnit 4

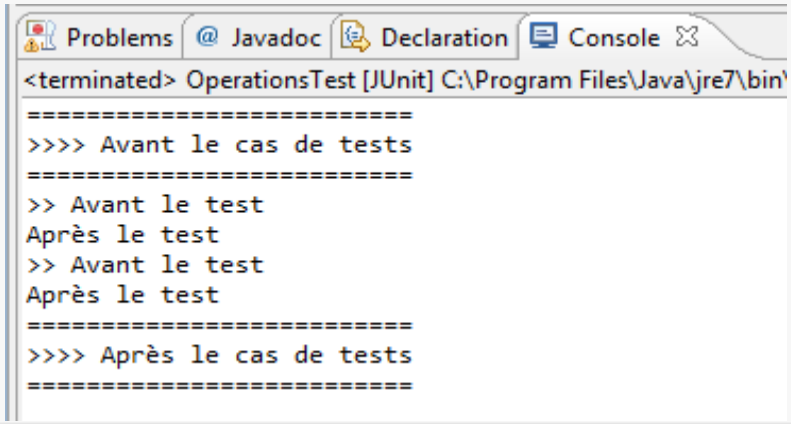
- Annotations `@BeforeClass` et `@AfterClass`
  - permettent d'indiquer des méthodes qui seront exécutées avant et après l'ensemble des méthodes de test
    - ces méthodes doivent être statiques
    - JUnit garantit que toutes les méthodes `@AfterClass` seront exécutées, même si une exception est levée dans une méthode `@BeforeClass`
    - les méthodes `@AfterClass` déclarées dans les classes de base seront exécutées

JUnit 4

- Annotations `@BeforeClass` et `@AfterClass`

```java
@BeforeClass
public static void beforeTestCase(){
    System.out.println("==========================");
    System.out.println(">>>> Avant le cas de tests");
    System.out.println("==========================");
}

@AfterClass
public static  void afterTestCas(){
    System.out.println("==========================");
    System.out.println(">>>> Après le cas de tests");
    System.out.println("==========================");
}
```





JUnit 4

- Méthodes d'assertion de base
  - les assertions sont des méthodes statiques de la classe `Assert`
  - `assertEquals` : vérifie que deux objets sont égaux
    - par appel de la méthode `equals(Object)`
  - `assertFalse` : vérifie que l'assertion est fausse
  - `assertNotNull` : vérifie que l'objet n'est pas nul
  - `assertNotSame` : vérifie que deux références sont différentes
  - `assertNull` : vérifie que l'objet est nul
  - `assertSame` : vérifie que deux références sont les mêmes
  - `assertTrue` : vérifie que l'assertion est vraie
  - `fail` : provoque l'échec du test

JUnit 4

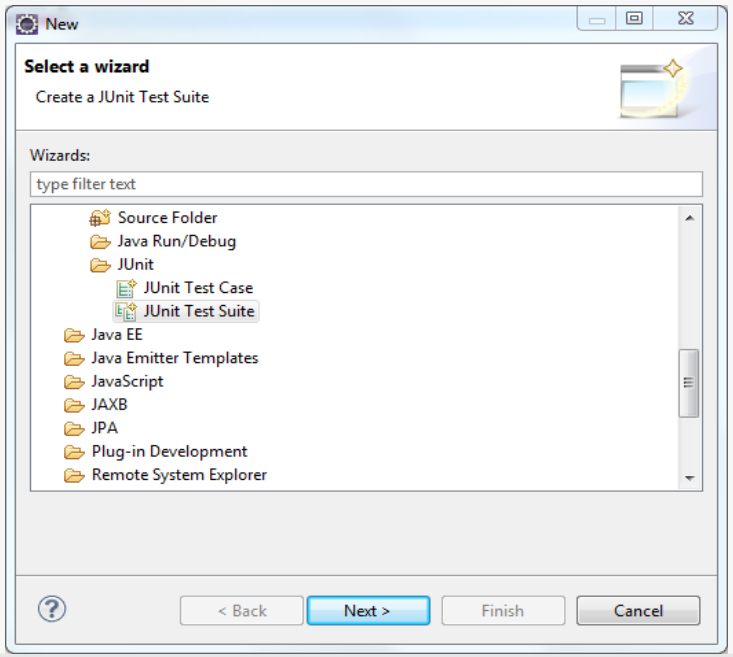
- Assertions d'égalité pour les tableaux
  - méthode : `assertArrayEquals`
  - différentes signatures permettent la comparaison de tableaux de types : `byte`, `char`, `short`, `int`, `long`, `Object`
  - certaines signatures permettent :
    - d'avoir des messages (`String`) en cas d'erreur
    - de lever une exception `ArrayComparisionFailure`
- Assertion d'égalité `assertEquals` sur double
  - avec prise en compte d'un delta maximum
  - avec message d'erreur

JUnit 4 – les suites de tests

- Une suite de tests permet d'exécuter plusieurs cas de test au sein d'une même exécution
  - la classe de suite est annotée par `@RunWith`
    - attribut ayant la valeur `Suite.class`

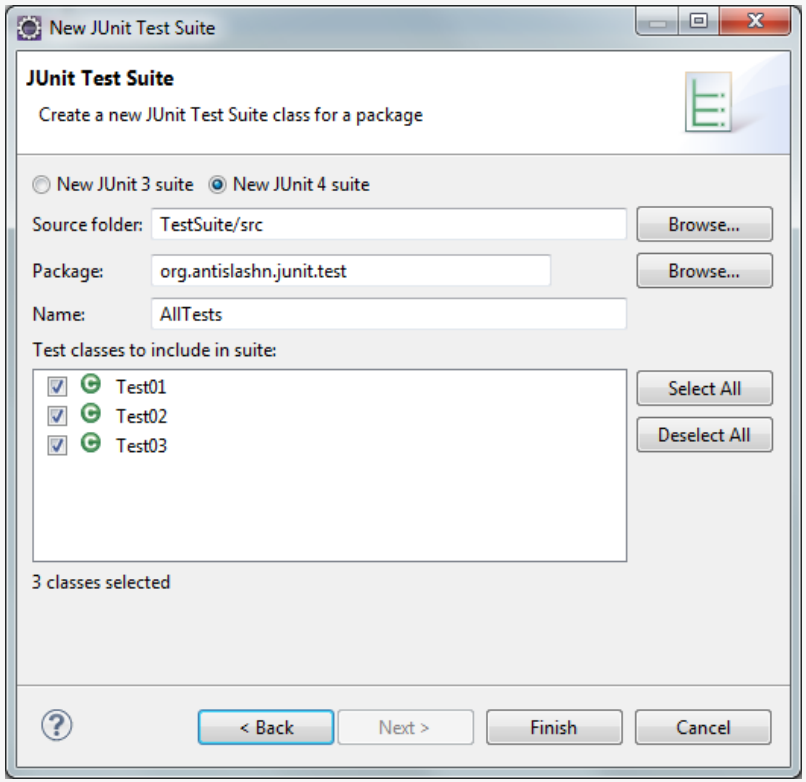
JUnit 4 – les suites de tests

- Les suites de tests unitaires
  - création sous Eclipse
    - dans le projet contenant les cas de tests à exécuter
      - **New → Other**
      - dans l'assistant ouvrir le dossier Java, puis JUnit
        - choisir JUnit Test Suite



JUnit 4 – les suites de tests

- Dans l'assistant de JUnit Test Suite
  - choisir les cas de tests à effectuer
  - une classe de base est créée, qui est exécutable par JUnit




```java
@RunWith(Suite.class)
@SuiteClasses({ Test01.class, Test02.class, Test03.class })
public class AllTests {

}
```

JUnit 4

- JUnit est en perpétuelle évolution
  - assertion définie par un contrat
  - supposition sur une condition de test
  - nouvelles fonctionnalités expérimentales
    - tests combinatoires
      - sur des jeux de données A : {A1,A2} et B : {B1,B2} pouvoir tester une méthode avec toutes le combinaisons A x B
- Il existe des extensions à JUnit
  - XMLUnit : tests sur documents XML
  - JUniExt : annotations `@Prerequisites`, ...