

Tests et qualité du code

JUnit 5

Youness LAGHOUAOUTA

Institut National des postes et télécommunications
laghouaouta@inpt.ac.ma

Fevrier 2022



1/13

Youness LAGHOUAOUTA Tests et qualité du code

Introduction

- JUnit est un framework de tests unitaires pour Java
- Généralisation des concepts à d'autres langages (XUnit)
- 3 versions de JUnit :
 - JUnit 3 : première version est désormais considérée comme obsolète
 - JUnit 4 : la version qui a vraiment vu l'explosion de JUnit. Très présente dans les projets industriels
 - JUnit 5 : dernière version de JUnit. Il est possible de lancer depuis JUnit 5 des tests JUnit 4

Framework

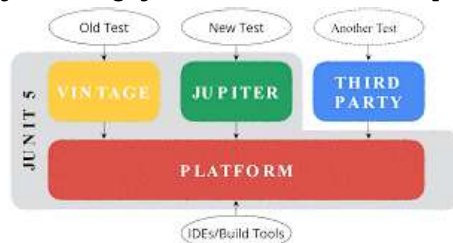
Un programme à "trous" qui offre la partie commune des traitements et chaque utilisateur le spécialise pour son cas particulier

2/13

Youness LAGHOUAOUTA Tests et qualité du code

Introduction

- Structure en 3 blocs :
 - JUnit Platform : le coeur du système ; cadre général permettant d'exécuter des tests
 - JUnit Jupiter : la bibliothèque nécessaire à l'exécution de tests JUnit 5
 - JUnit Vintage : pour exécuter des tests JUnit 3 et JUnit 4 depuis JUnit 5.
- Plusieurs IDE supportent JUnit 5 : Eclipse, IntelliJ IDEA
- Support native pour l'exécution des tests JUnit 5 avec Maven et Gradle
- Documentation :
'<https://junit.org/junit5/docs/current/api/index.html>



3/13

Youness LAGHOUAOUTA Tests et qualité du code

Premier exemple

- Classe à tester

```
public class Calculator {  
    float add(float a, float b) {  
        return a + b;  
    }  
  
    int divide(int a, int b) {  
        return a/b;  
    }  
}
```

- Classe de test

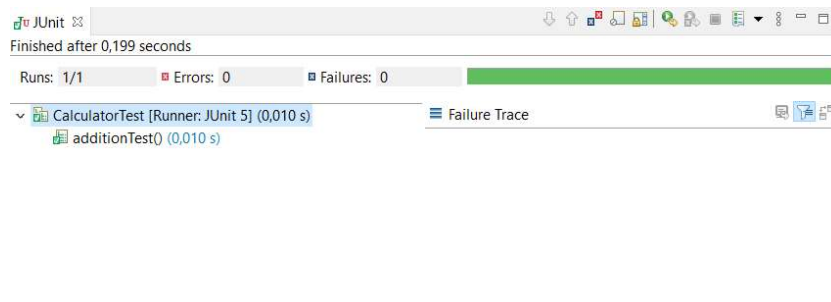
```
class CalculatorTest {  
  
    @Test  
    void additionTest() {  
        Calculator calc = new Calculator();  
        assertEquals(2, calc.add(1,1), "The output should be the sum of the two arguments");  
    }  
}
```

<https://stackabuse.com/unit-testing-in-java-with-junit-5/>

4/13

Youness LAGHOUAOUTA Tests et qualité du code

Premier exemple



Classe Assertions

Ensemble de méthodes statiques aidant à écrire des tests. Ces méthodes lèvent des `AssertionFailedError` (sous-type de `AssertionError`) en cas d'échec

- `assertTrue(boolean)` : le paramètre doit être vrai
- `assertEquals(Object attendu, Object effectif)` : `attendu.equals(effectif)` doit être vraie
- `assertEquals(typeDeBase attendu, typeDeBase effectif)` : `attendu == effectif` doit être vraie
- `assertEquals(typeReel attendu, typeReel effectif, typeReel delta)` : `abs(attendu - effectif) < delta` doit être vrai
- `assertSame(Object attendu, Object effectif)` : `attendu == effectif` doit être vraie
- `assertNull(Object objNull)` : `objNull == null` doit être vraie

Classe Assertions

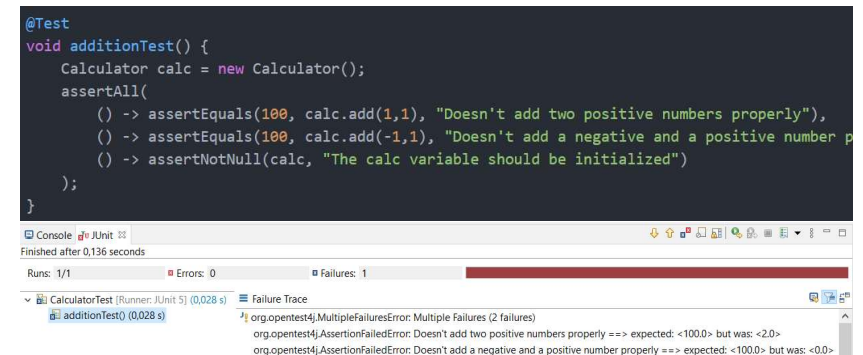
- `assertArrayEquals(Type[] attendu, Type[] effectif)` : égal "profond" sur les tableaux
- `assertIterableEquals(Iterable<?> attendu, Iterable<?> effectif)` : égal "profond" sur les itérables
- `assertArrayEquals(TypeReel[] attendu, TypeReel[] effectif, TypeReel delta)` : égal "profond" sur les tableaux de réels
- `assertThrows(Class<T> typeException, Executable exécutable)` : exécutable doit lever une exception du type spécifié
- `assertFalse(boolean)`, `assertNotEquals(...)`, `assertNotSame(Object attendu, Object effectif)` ...

Message d'échec

La plupart des méthodes sont surchargées en ajoutant aux paramètres de base une chaîne de caractères correspondant au message à afficher en cas d'échec

Détecter plusieurs anomalies

- Problème : un échec interrompt l'exécution de la méthode de test et empêche de procéder à plusieurs vérifications
- Solution : utilisation de la méthode `assertAll(...)`



<https://stackabuse.com/unit-testing-in-java-with-junit-5/>

Quelques annotations

- **@BeforeAll** : Initialisation avant l'exécution de tous les tests
- **@AfterAll** : Finalisation à la fin de l'exécution de tous les tests
- **@BeforeEach** : Code à exécuter avant chaque test
- **@AfterEach** : Code à exécuter après chaque test
- **@DisplayName(nomAAfficher)** : Affichage du nom du test
- **@Tag(étiquette)** : Annotation des méthodes de test avec une ou plusieurs étiquettes
- **@Disabled** : Désactivation d'une méthode de test

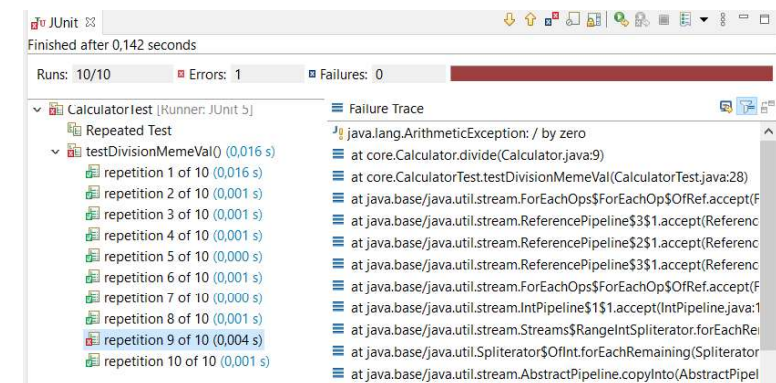
Exemple

Tests répétés

- L'annotation **@RepeatedTest** permet d'exécuter plusieurs fois une méthode de test
- Exemple :

```
@RepeatedTest(10)
void testDivisionMemeVal() {
    Random rand = new Random();
    int randVal = rand.nextInt(100);
    Calculator calc = new Calculator();
    assertEquals(1, calc.divide(randVal, randVal));
}
```

Tests répétés

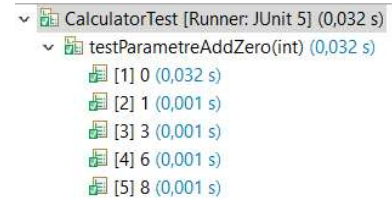


Exemple personnalisation du nom du test

```
@DisplayName("Test répété")
@RepeatedTest(value = 10, name = "{displayName} num : {currentRepetition}")
```

- L'annotation **@ParameterizedTest** permet de paramétrer des méthodes de test pour les faire exécuter avec des valeurs différentes
- Exemple :

```
@ParameterizedTest
@ValueSource(ints = {0, 1, 3, 6, 8})
void testParametreAddZero(int n) {
    Calculator calc = new Calculator();
    assertEquals(n, calc.add(0,n));
}
```



```
CalculatorTest [Runner: JUnit 5] (0,032 s)
  testParametreAddZero(int) (0,032 s)
    [1] 0 (0,032 s)
    [2] 1 (0,001 s)
    [3] 3 (0,001 s)
    [4] 6 (0,001 s)
    [5] 8 (0,001 s)
```