

Test automatisé ?

- ◆ Concevoir des programmes de test
- ◆ Appliquer les programmes de test pour
 - Vérifier le bon fonctionnement
 - des composants (tests unitaires)
 - du programme (test d'intégration)
 - Identifier les aspects défectueux du programme

Annotations JUnit4

- ◆ **@Test** précède toute méthode de test.
Les méthodes de test sont exécutées dans leur ordre d'apparition dans la classe de test.
- ◆ **@BeforeClass** précède une méthode statique à exécuter une fois avant la première méthode de test.
- ◆ **@Before** précède une méthode à exécuter avant chaque méthode de test.
- ◆ **@After** précède une méthode à exécuter après chaque méthode de test.
- ◆ **@AfterClass** précède une méthode statique à exécuter une fois après la dernière méthode de test.

Méthode de test JUnit4

- ◆ Une méthode de test comporte :
 - du code « standard »
 - une ou des méthodes d’assertion de la classe `org.junit.Assert`
 - `assertTrue(expression)`
 - `assertFalse(expression)`
 - `assertEquals(expressionAttendue, expression)`
 - ...

Méthode de test JUnit4 ...

- ◆ Une méthode de test peut aussi :
 - Attendre la levée d’une exception
`@Test(expected = UneClasseDException)`
 - Fixer une durée maximum d’exécution
`@Test(timeout = millis)`

Où sont les méthodes de test ?

- ◆ Une méthode de test trouve sa place dans une classe de test dont le code est précédé de 2 imports :

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

Classe de test JUnit 4

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

public class MaClasseDeTest {
    @BeforeClass
    public static void initTests() { ... }
    @Test
    public void methodeDeTest1 () { ... assert(...); }
    @Test
    public void methodeDeTest2 () { ... assert(...); }
}
```

Ordre d'exécution des tests

Les méthodes de test sont exécutées dans un ordre différent de celui de leur écriture sauf à préciser le contraire en annotant la classe de test

@FixMethodOrder(MethodSorters.NAME_ASCENDING)

Ce qui implique les imports suivants :

import org.junit.FixMethodOrder;

import org.junit.runners.MethodSorters;

Echec d'un test

- ◆ Un test (méthode annotée @Test) échoue si :
 - au moins une assertion échoue dans la méthode;
 - une exception attendue n'est pas levée;
 - le temps d'exécution dépasse un temps limite fixé.

JUnit 5 (voir <https://junit.org/junit5/docs/current/user-guide/>)

JUnit Platform	moteur de lancement des tests
+ JUnit Jupiter	tests et extensions en JUnit 5
+ JUnit Vintage	tests JUnit 3 et JUnit 4
<hr/>	
= JUnit 5	

Les classes JUnit 5 sont dans le paquet
org.junit.jupiter.api

Annotations JUnit 5

- ◆ **@Test** précède toute méthode de test.
Les méthodes de test sont exécutées dans leur ordre d'apparition dans la classe de test.
- ◆ **@DisplayName("nom à afficher")**
- ◆ **@BeforeAll** précède une méthode statique à exécuter une fois avant la première méthode de test.
- ◆ **@BeforeEach** précède une méthode à exécuter avant chaque méthode de test.
- ◆ **@AfterEach** précède une méthode à exécuter après chaque méthode de test.
- ◆ **@AfterAll** précède une méthode statique à exécuter une fois après la dernière méthode de test.

Méthode de test JUnit 5

- ◆ Une méthode de test comporte :
 - du code « standard »
 - une ou des méthodes d’assertion de la classe `org.junit.jupiter.api.Assertions`
 - `assertTrue(expression)`
 - `assertFalse(expression)`
 - `assertEquals(expressionAttendue, expression)`
 - ...

Méthode de test JUnit 5 ...

- ◆ Une méthode de test peut aussi :
 - Attendre la levée d’une exception


```
Exception e =
assertThrows(ArithmeticException.class, () ->
    calculator.divide(1, 0));
assertEquals("/ by zero", e.getMessage());
```
 - Fixer une durée maximum d’exécution


```
String res = assertTimeout(ofMinutes(2), () -> {
    return "a result";
});
assertEquals("a result", res);
```

méthode de la classe
`java.time.Duration`