Tests unitaires avec Junit

Christelle CAILLOUET
(christelle.caillouet@unice.fr)

Les tests unitaires

- Introduits en M112-M113 (**assert**)

- Etape du cycle de développement d'un logiciel
  - Phase de tests avant intégration

- Principes (appliqués à la POO) :
  - Développement des classes accompagné **en parallèle** de ses modules (*framework*) de tests unitaires
  - Ces modules de tests sont des classes
  - **Indépendance structurelle** des classes de tests et de la classe testée

Rôle des modules de tests unitaires

- Permettre de vérifier la conformité de la classe vis-à-vis de ses spécifications techniques détaillées

- Permettre d'exécuter au moins une fois et dans chacun des contextes opérationnels :
  - **Chaque constructeur**
  - **Chaque méthode publique**

- Cas particuliers et cas généraux

Limites

- Un module de test unitaire **ne prouve pas** que la classe cible fonctionne parfaitement
- Il permet seulement de vérifier que les résultats produits par les méthodes exécutées sont conformes à des résultats attendus et prédéfinis
- Le programmeur **a la charge** de définir de la manière la + complète possible :
  - **Tous les contextes d'exécution (cas de tests)**
  - **Tous les résultats attendus**

Frameworks de tests unitaires

- Liés au langage de développement (plugin IDE) :
  - Java : principalement **JUnit**
  - C++ : **Cutter**, Google propose **Google C++ Testing Framework**, la bibliothèque Boost comprend la **Boost Test Library**
  - Python : **unittest**, **PyUnit**
  - PHP : **PHPUnit**, **SimpleTest**, **Atoum**
  - Javascript : le framework jQuery utilise **qunit**, **Jarvis**, **jfUnit**, **google-js-test**

JUnit

http://junit.org

- Initialement développé par Erich Gamma et Kent Beck.
- **Outil de gestion** (bibliothèque) des tests unitaires pour les programmes Java :
  - Framework pour le développement des tests unitaires reposant sur des assertions qui testent les résultats attendus
  - Applications pour permettre l'exécution des tests et afficher les résultats
- Le but est d'automatiser les tests
- Il existe un plug-in Eclipse pour JUnit (intégré par défaut dans la plupart des IDEs)

JUnit offre :

- des primitives pour créer un test (assertions)
- des primitives pour gérer des suites de tests
- des facilités pour l'exécution des tests
- des statistiques sur l'exécution des tests
- une interface graphique pour la couverture des tests
- des points d'extensions pour des situations spécifiques

Avantages

- Ancien test de classe : méthode *main* qui contient les traitements de tests
  - Code pouvant être inclus dans la classe donc dépendent du code source
  - Exécution doit se faire manuellement

➔ La rédaction de cas de tests :
  - Effet immédiat pour détecter des bugs
  - Effet à long terme pour faciliter la détection d'effets de bords lors de modifications

- Les cas de tests sont regroupés dans des classes Java qui contiennent une ou plusieurs méthodes de tests.

- Ces cas de tests peuvent être regroupés sous la forme de suites de tests (`TestSuite` qui permet d'exécuter un certain nombre de `TestCase` déjà définis)

JUnit 5

- Historique :
  - JUnit 4 a été publié en 2005 pour permettre la prise en compte des annotations de Java 5.
  - JUnit 5, publié en 2017, utilise des fonctionnalités de Java 8 notamment les lambdas, les annotations répétées, …
- Réécriture intégrale du framework ayant plusieurs objectifs :
  - le support et l'utilisation des nouvelles fonctionnalités de Java 8
  - une nouvelle architecture reposant sur plusieurs modules
  - le support de différents types de tests (imbriqués, dynamiques, paramétrés)
  - un mécanisme d'extension qui permet l'ouverture vers des outils tiers ou des API
- Contrairement aux versions précédentes livrées en un seul jar, JUnit 5 est livré sous la forme de différents modules

Package JUnit

- Depuis JUnit 5, il y a 3 sous-projets :
  - JUnit Platform : API permettant aux outils de découvrir et exécuter des tests.
  - **JUnit Jupiter** : API reposant sur des annotations pour écrire des tests unitaires JUnit 5 et un TestEngine pour les exécuter : `junit-jupiter-api`
  - JUnit Vintage : TestEngine pour exécuter des tests JUnit 3 et 4 et ainsi assurer une compatibilité ascendante

- JUnit 5 compatible avec une version >= Java 8 : il n'est pas possible d'utiliser une version antérieure.

Utilisation

- Avec JUnit, l'unité de test est une classe dédiée qui regroupe des cas de tests (`TestCase`). Ces cas de tests exécutent les tâches suivantes :
  1. création d'une instance de la classe et de tout autre objet nécessaire aux tests
  2. appel de la méthode à tester avec les paramètres du cas de tests
  3. comparaison du résultat attendu avec le résultat obtenu : en cas d'échec, une exception est levée

Les annotations

- Introduites à partir de Java 5, permet de spécifier une information au compilateur :
  - `@Override` (déjà vu) demande au compilateur de vérifier que l'on redéfini bien une méthode héritée

- JUnit depuis la version 4, inclut l'utilisation des annotations pour les tests :
  - `@Test` avant une méthode, indique au compilateur que la méthode est un test
  - `@BeforeEach` et `@AfterEach` permettent d'indiquer une méthode qui sera exécutée avant **chaque** test et un méthode qui sera exécutée après **chaque** test.

```java
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

public class TestAvantApres {

    @BeforeEach
    public void avantTest() {
        System.out.println("--------------------");
        System.out.println("Avant Test");
    }

    @AfterEach
    public void apresTest() {
        System.out.println("Après Test");
        System.out.println("--------------------");
    }

    @Test
    public void premierTest() {
        System.out.println("Premier Test");
    }

    @Test
    public void deuxiemeTest() {
        System.out.println("Deuxième Test");
    }

    @Test
    public void troisiemeTest() {
        System.out.println("Troisième Test");
    }
}
```

- Affichera :

```
------------------------
Avant Test
Premier Test
Après Test
------------------------
------------------------
Avant Test
Deuxième Test
Après Test
------------------------
------------------------
Avant Test
Troisième Test
Après Test
------------------------
```

Autres annotations

- `@BeforeAll` et `@AfterAll` permettent d'indiquer une méthode qui sera exécutée avant **l'ensemble** des tests d'un cas de tests et une méthode qui sera exécutée après **l'ensemble** des tests d'un cas de tests.

- Toutes les annotations sont situées dans le package `org.junit.jupiter.api` du module `junit-jupiter-api`

https://junit.org/junit5/docs/current/api/org/junit/jupiter/api/package-summary.html

Annotations spéciales

- La méthode annotée est un cas de test paramétré : `@ParameterizedTest`
- La méthode annotée est un cas de test répété `@RepeatedTest`
- Ignorer un cas de test (ou désactiver un test temporaire) :

```
@Disabled
@Test
public void ignore() {
    Assertions.fail("Echec ignoré");
}
```

La classe Assertions

Package org.junit.jupiter.api

Class Assertions

java.lang.Object
    org.junit.jupiter.api.Assertions

```
@API(status=STABLE,
     since="5.0")
public class Assertions
extends Object
```

Assertions is a collection of utility methods that support asserting conditions in tests.

Unless otherwise noted, a *failed* assertion will throw an AssertionFailedError or a subclass thereof.

https://junit.org/junit5/docs/current/api/org/junit/jupiter/api/Assertions.html

L'assertion est la + petite unité de test dont le résultat de l'expression booléenne indique un succès ou une erreur

- Les `TestCase` utilisent des assertions sous la forme de méthodes nommées `assertXXX()` proposées par le framework.

- Il existe de nombreuses méthodes de ce type qui sont référencées dans la classe `Assertions` :

Méthodes assert...()

Méthode	Rôle
`assertEquals()`	Vérifier l'égalité de deux valeurs de type primitif ou objet (en utilisant la méthode equals()). Il existe de nombreuses surcharges de cette méthode pour chaque type primitif, pour un objet de type Object et pour un objet de type String
`assertFalse()`	Vérifier que la valeur fournie en paramètre est fausse
`assertNull()`	Vérifier que l'objet fourni en paramètre soit null
`assertNotNull()`	Vérifier que l'objet fourni en paramètre ne soit pas null
`assertSame()`	Vérifier que les deux objets fournis en paramètre font référence à la même entité Exemples identiques : assertSame(obj1, obj2, "Les deux objets sont identiques"); assertTrue(obj1 == obj2, "Les deux objets sont identiques ");
`assertNotSame()`	Vérifier que les deux objets fournis en paramètre ne font pas référence à la même entité
`assertTrue()`	Vérifier que la valeur fournie en paramètre est vraie

Exemple : une classe Java

```
public class MaClasse{
    public static int calculer(int a, int b) {
        int res = a + b;
        if (a == 0)
            res = b * 2;

        if (b == 0)
            res = a * a;

        return res;
    }
}
```

Exemple : écrire une classe de tests

Import les méthodes
statiques de la classe Assertions

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

public class MaClasseTest {
    @Test
    public void testCalculer() throws Exception {
        assertEquals(2,MaClasse.calculer(1,1));
    }
}
```

Import de l'annotation @Test

Exemple : lancer les tests (JUnit 4)

```
C:\testjunit>javac MaClasse.java

C:\testjunit>javac -cp .;"C:\Program Files\JUnit\junit.jar";"C:\Program Files\JU
nit\hamcrest-core.jar" MaClasseTest.java

C:\testjunit>java -cp .;"C:\Program Files\JUnit\junit.jar";"C:\Program Files\JUn
it\hamcrest-core.jar" org.junit.runner.JUnitCore MaClasseTest
JUnit version 4.12
.
Time: 0,006

OK (1 test)
```

Erreurs

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

public class MaClasseTest {
    @Test
    public void testCalculer()
            throws Exception {
        assertEquals(4,
            MaClasse.calculer(1,1));
    }
}
```

```
C:\testjunit>javac -cp .;"C:\Program Files\JUnit\junit.jar";"C:\Program Files\JU
nit\hamcrest-core.jar" MaClasseTest.java

C:\testjunit>java -cp .;"C:\Program Files\JUnit\junit.jar";"C:\Program Files\JUn
it\hamcrest-core.jar" org.junit.runner.JUnitCore MaClasseTest
JUnit version 4.12
.E
Time: 0,015
There was 1 failure:
1) testCalculer(MaClasseTest)
java.lang.AssertionError: expected:<4> but was:<2>
        at org.junit.Assert.fail(Assert.java:88)
        at org.junit.Assert.failNotEquals(Assert.java:834)
        at org.junit.Assert.assertEquals(Assert.java:645)
        at org.junit.Assert.assertEquals(Assert.java:631)
        at MaClasseTest.testCalculer(MaClasseTest.java:7)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)
        at java.lang.reflect.Method.invoke(Unknown Source)
        at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(Framework
Method.java:50)
        at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCal
lable.java:12)
        at org.junit.runners.model.FrameworkMethod.invokeExplosively(FrameworkMe
thod.java:47)
        at org.junit.internal.runners.statements.InvokeMethod.evaluate(InvokeMet
hod.java:17)
        at org.junit.runners.ParentRunner.runLeaf(ParentRunner.java:325)
        at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRun
ner.java:78)
        at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRun
ner.java:57)
        at org.junit.runners.ParentRunner$3.run(ParentRunner.java:290)
        at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:71)
        at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:288)
        at org.junit.runners.ParentRunner.access$000(ParentRunner.java:58)
        at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:268)
        at org.junit.runners.ParentRunner.run(ParentRunner.java:363)
        at org.junit.runners.Suite.runChild(Suite.java:128)
        at org.junit.runners.Suite.runChild(Suite.java:27)
        at org.junit.runners.ParentRunner$3.run(ParentRunner.java:290)
        at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:71)
        at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:288)
        at org.junit.runners.ParentRunner.access$000(ParentRunner.java:58)
        at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:268)
        at org.junit.runners.ParentRunner.run(ParentRunner.java:363)
        at org.junit.runner.JUnitCore.run(JUnitCore.java:137)
        at org.junit.runner.JUnitCore.run(JUnitCore.java:115)
        at org.junit.runner.JUnitCore.runMain(JUnitCore.java:77)
        at org.junit.runner.JUnitCore.main(JUnitCore.java:36)

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

Règles à respecter

- Le `TestCase` d'une classe `MaClasse` s'appelle par convention `MaClasseTest`

- Chacune des méthodes de test de la classe de tests doit avoir les caractéristiques suivantes :
  - elle doit être déclarée public
  - elle ne doit renvoyer aucune valeur
  - elle ne doit pas posséder de paramètres.

Autres assertions

- Assertions pour les tableaux
  `assertArray`
  - Par exemple pour l'égalité : `assertArrayEquals`

```
@Test
void verifierEgaliteTableaux() {
  Assertions.assertArrayEquals(new int[] { 1, 2, 3 }, new int[] { 1, 2, 3 },
    "Egalite des tableaux");
}
```

- Assertion pour les exceptions
  `assertThrows`

```
@Test
public void constructorShouldThrowIAEForInvalidInput() {
    assertThrows(IllegalArgumentException.class, () -> {
        new Money(-5,"USD");
    });
}
```

Tester la classe String

```
public class StringTest {
    @Test
    public void testConcatenation() {
        String foo = "abc";
        String bar = "def";
        assertEquals("abcdef", foo + bar);
    }

    @Test
    public void testStartsWith() {
        String foo = "abc";
        assertTrue(foo.startsWith("ab"));
    }
}
```

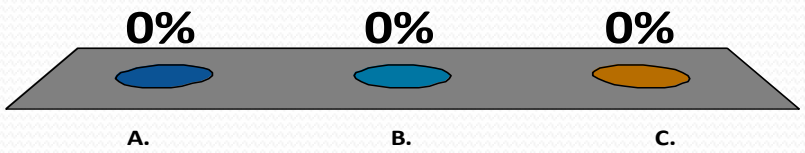
Que renvoie le `TestCase` suivant ?

```
public class StringTest {
    @Test
    public void testConcatenation() {
        String foo = "abc";
        String bar = "def";
        assertEquals("abcdef", foo + bar);
    }

    @Test
    public void testStartsWith() {
        String foo = "abc";
        assertTrue(foo.startsWith("ab"));
    }
}
```

30

A. Aucune erreur
B. 1 erreur
C. 2 erreurs



Tester la classe String

```
public class StringTest {

    @Test
    public void testConcatenation() {
        String foo = "abc";
        String bar = "def";
        assertEquals("abcdef", foo + bar);
    }

    @Test
    public void testStartsWith() {
        String foo = "abc";
        assertTrue(foo.startsWith("ab"));
    }
}
```

```
C:\testjunit>javac -cp .;"C:\Program Files\JUnit\junit.jar";"C:\Program Files\JU
nit\hamcrest-core.jar" StringTest.java

C:\testjunit>java -cp .;"C:\Program Files\JUnit\junit.jar";"C:\Program Files\JUn
it\hamcrest-core.jar" org.junit.runner.JUnitCore StringTest
JUnit version 4.12
..
Time: 0,009

OK (2 tests)
```

Tester une nouvelle classe

```
public class Personne {
    private String nom;
    private String prenom;

    public Personne() {    }

    public Personne(String nom, String prenom) {
        this.nom = nom;
        this.prenom = prenom;
    }

    public String getNom() {
        return nom;
    }

    public void setNom(String nom) {
this.nom = nom;
    }

    public String getPrenom() {
        return prenom;
    }

    public void setPrenom(String prenom) {
        this.prenom = prenom;
    }
}
```

La classe de tests de Personne

```
import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

public class PersonneTest {

  private Personne personne;

  @BeforeEach
  public void initialiser() throws Exception {
    personne = new Personne("nom1","prenom1");
  }

  @AfterEach
  public void nettoyer() throws Exception {
    personne = null;
  }

  @Test
  public void personne() {
    assertNotNull(personne, "L'instance n'est pas
créée");
  }

  @Test
  public void getNom() {
    assertEquals("nom1", personne.getNom(), "Le
nom est incorrect");
  }

  @Test
  public void getPrenom() {
    assertEquals("prenom1", personne.getPrenom(),
"Le prenom est incorrect");
  }

  @Test
  public void setPrenom() {
    personne.setPrenom("prenom2");
    assertEquals("prenom2", personne.getPrenom(),
"Le prenom est incorrect");
  }
}
```

JUnit dans Eclipse

https://www.eclipse.org/eclipse/news/4.7.1a/#junit-5-support

- File > New > JUnit Test Case
- Séparer le code source Java (dans src) et les modules de tests unitaires JUnit (dans tests)