المدرسة الوطنية للعلوم التطبيقية -أكادير
ⵜⴰⵙⴷⴰⵡⵜ ⵜⴰⵏⴰⵎⵓⵔⵜ ⵏ ⵜⵣⵎⵔⵉⵜ ⵜⵉⵏⵣⵡⴰⵢⵏ – ⴰⴳⴰⴷⵉⵔ.
ÉCOLE NATIONALE DES SCIENCES APPLIQUÉES - AGADIR

# Test unitaire
# Junit

Mohamed El yaakoubi

*Tester, c'est exécuter le programme dans l'intention d'y trouver des anomalies ou des défauts.*

G. Myers (The Art of Software testing)

- JUnit est un framework open source pour le développement et l'exécution de tests unitaires automatisables.

JUnit

- Le principal intérêt est de s'assurer que le code répond toujours aux besoins même après d'éventuelles modifications.

- Plus généralement, ce type de tests est appelé tests unitaires de non régression.

- Pour tester une classe, on peut éventuellement créer une méthode main() dans chaque classe qu'on veut tester qui va contenir les traitements de tests.

- L'utilisation d'un framework pour le test comme Junit permet de séparer le code de la classe, du code qui permet de la tester.

- Test Driven Development (TDD) : on écrit les tests avant de coder l'application!

- Le testeur doit identifier les cas de test
- Un ensemble de cas de test est appelé jeux de test
- Le but est d'automatiser les tests
- Les tests sont exprimés dans des classes de test sous la forme de cas de tests avec leurs résultats attendus.
- Junit exécute ces tests et les comparent avec ces résultats.
- Junit repose sur des assertions qui testent les résultats attendus

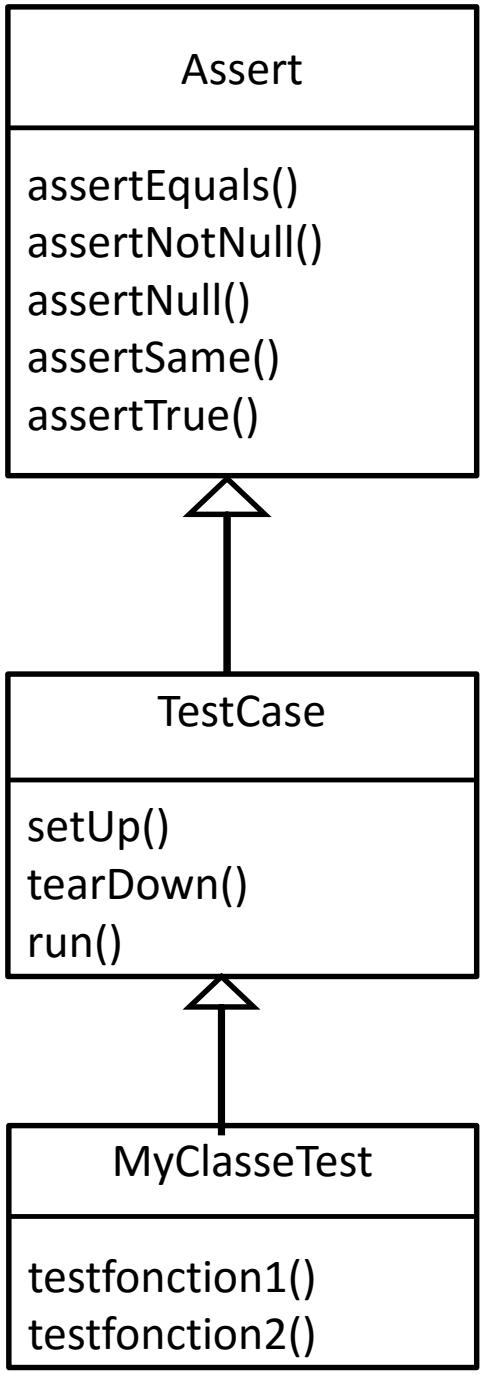
  Exemple : **`assertEquals(2,cl.somme(1,1));`**
- L'ordre -> (valeur attendue, valeur obtenue) est important pour obtenir un message d'erreur fiable en cas d'échec du cas de test.

- Le testeur est dans le but de générer un jeux de test satisfaisant a deux grandes stratégies

    - **Couvrir le fonctionnel (ou tests boîte noire)**

    - **Couvrir le structurel (ou tests boîte blanche)**

# *Junit3*




- Ecrire une classe de test consiste à :
  - hériter de la classe TestCase
  - implémenter plusieurs méthodes nommées testXXX()
  - écrire des assertions assetXXX :
    - assertTrue(1 > 0)
    - assertEquals(7, 3+4)

# Exemple

```
public class Calcul {
public int Add() { … }
public int Multiply() { … }
}
```

Classe à tester

Classe de test

```
import junit.framework.TestCase;
public class CalculTest extends TestCase
{
public void testAdd() {
Calcul cl = new Calcul();
int expected = 7;
int actual = cl.add(3, 4);
assertEquals(expected, actual);
assertEquals(1, cl.add(3, -2));
}
public void testMultiply() { … }
}
```

# Méthode contenant les assertions de tests

Chacune de ces méthodes doit avoir les caractéristiques suivantes :

- elle doit être déclarée public
- elle ne doit renvoyer aucune valeur
- elle ne doit pas posséder de paramètres

Une méthode peut contenir une ou plusieurs assertions de tests

# Fixture

Ensemble des objets utilisés dans plusieurs fonctions d'une même classe de test

Chaque objet utilisé est alors

- déclaré en tant que variable d'instance de la classe de test
- initialisé dans la méthode setUp()
- éventuellement libéré dans la méthode tearDown()

# Exemple avec Fixture

```
import junit.framework.TestCase;

public class CalculTest extends TestCase {

Calcul cl;

protected void setUp() throws Exception {
cl = new Calcul();
}

protected void tearDown() throws Exception {
cl = null;
}

public void testAdd() {
assertEquals(2,cl.calculer(1,1));
}

}
```

# Exécuter un test isolément

Définir un main utilisant un TestRunner

```
import junit.framework.TestCase;
public class CalculeTest extends TestCase {
public static void main(String[] args){
// affichage dans une vue textuelle
junit.textui.TestRunner.run(CalculTest.class);
}
}
```

Il existe aussi des vues graphiques :

- `junit.swingui.TestRunner, …`

# Tester une exception

Classe à tester

```
public class Calcul {

public double devide(double x,double y)
throws IllegalArgumentException{

if(y==0) throw new IllegalArgumentException();

return x/y;
}
```

# Tester une exception

Classe de test

```
import junit.framework.TestCase;
public class CalculTest extends TestCase {

Calcul cl;

protected void setUp() throws Exception {
cl = new Calcul();
}

protected void tearDown() throws Exception {
cl = null;
}

public void testDevide(){
try {
cl.devide(1,0);
fail("Exception de type IllegalArgumentException aurait du être levée");
}
catch(IllegalArgumentException ise){}
}
}
```

# Faire échouer un test

- La méthode fail() permet de forcer le cas de test à échouer. Une version surchargée permet de préciser un message qui sera affiché.

# Suite de tests

```
import junit.framework.*;

public class ExecuterLesTests {

public static Test suite() {
TestSuite suite = new TestSuite("Tous les tests");
suite.addTestSuite(MaClasseTest1.class);
suite.addTestSuite(MaClasseTest2.class);
return suite;
}

public static void main(String args[]) {
junit.textui.TestRunner.run(suite());
}
}
```

# *Junit4*

- JUnit version 4 est une évolution majeure depuis les quelques années d'utilisation de la version 3.8.

- Un des grands bénéfices de cette version est l'utilisation des annotations.

- La définition des cas de tests et des tests ne se fait donc plus sur des conventions de nommage et sur l'introspection mais sur l'utilisation d'annotations ce qui facilite la rédaction des cas de tests.

- JUnit 4 requiert une version 5 ou ultérieure de Java.

- Le nom du package des classes de JUnit est différent entre la version 3 et 4
    - les classes de Junit 3 sont dans le package junit.framework
    - les classes de Junit 4 sont dans le package org.junit

## La définition d'une classe de tests

```
import org.junit.*;
import static org.junit.Assert.*;
public class CalculTest {}
```

## La définition d'un cas de test

```
@Test
public void testAdd() {
assertTrue(2==cl.calculer(1,1))
;
}
```

Exemple avec Fixture

```
import static org.junit.Assert.*;
import org.junit.After;
import org.junit.Before;
import org.junit.Ignore;
import org.junit.Test;

public class CalculTest {
Calcul cl;
@Before
public void setUp() {
cl = new Calcul();
}

@After
public void tearDown(){
cl = null;
}

@Test
public void testAdd() {
assertEquals(2,cl.calculer(1,1));
}
}
```

# Tester une exception

```
@Test(expected=IllegalArgumentException.class)
public void testDevide(){
cl.devide(1,0);
fail("Une exception IllegalArgumentException aurait du être levée");
}
```

## Ignorer un cas de test

```
@Ignore("Not ready to run")
@Test
public void ignore() {
        fail("Echec ignoré");
}
```

## La limitation du temps d'exécution d'un cas de test

```
@Test(timeout=1000)
public void testCalcul()  {

……

}
```

| Méthode | Rôle |
| --- | --- |
| assertEquals() | Vérifier l'égalité de deux valeurs de type primitif ou objet |
| 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<br>Exemples identiques :<br>assertSame("Les deux objets sont identiques", obj1, obj2);<br>assertTrue("Les deux objets sont identiques ", obj1 == obj2); |
| 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 |

# Suite de tests

```java
import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses({
CalculTest.class,CalculTest.class })
public class AllTests {

}
```

Le Test avec spring et configuration xml

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations="classpath:springContext.xml")
public class DaoTest1 {

@Autowired
private Idao dao;
@Test
public void testAjouter() {
….
}
@Test
public void testDelete() {
….
}
@Test
public void testFindById() {
….
}
@Test
public void testGetAll() {
….
}
}
```

## Le Test avec spring et configuration par annotations

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes={AppConfig.class})
public class DaoTest1 {


@Autowired
private Idao dao;
@Test
public void testAjouter() {
….
}
@Test
public void testDelete() {
….
}
@Test
public void testFindById() {
….
}
@Test
public void testGetAll() {
….
}
}
```