dimanche 1 novembre 2020

Documentation JUnit

Présentation

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

Il permet de s'assurer que le test réponde tout le temps malgré des modifications.

Il permet de séparer le code de la classe du code qui permet de la tester.

Écriture des classes test

Pour compiler la classe il faut le compiler avec le fichier junit.jar qui doit être dans le classpath.

Pour la classe test, par convention le nom de la classe test est composé du nom de la classe suivi de Test. Depuis JUnit 4 ce n'est plus obligatoire mais c'est une convention (remplacer par @Test). A partir de JUnit 4 on peut ignorer une classe test avec @Ignore.

IMPORTANT : lors de l'écriture de cas de tests, chaque cas doit être indépendant les uns des autres.

JUnit fonctionne par introspection donc JUnit ne garantit pas l'ordre d'exécution.

Définir une classe test

Pour écrire un test, il faut écrire qui étende la classe junit.framework.Text(Nom de la classe)

Exemple :

```
1  import junit.framework.*;
2
3  public class MaClasseTest extends TestCase {
```

Cette classe doit être déclarée **public**, ne doit renvoyer **aucune** valeur et ne doit pas posséder de **paramètres**.

⚠️ JUnit ira automatique chercher les méthodes qui respectent cette convention.

JUnit recherche obligatoirement les qui débutent par test, par conséquent chaque classe doit **au moins avoir une méthode de test** (avec les caractéristiques vient ci-dessus).

Dès qu'un test échoue, l'exécution de la méthode correspondante est interrompue, passe a la méthode suivante et remonte l'erreur.

Pour tester sur JUnit, on utilise des méthodes nommées assertXXX().

Voici quelques exemples :

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("Les deux objets sont identiques", obj1, obj2); 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

Exemple :

```
1  import junit.framework.*;
2
3  public class MaClasse2Test extends TestCase {
4
5    public void testCalculer() throws Exception {
6      MaClasse2 mc = new MaClasse2(1,1);
7      assertEquals(2,mc.calculer());
8    }
9  }
```

La méthode **fail()**, permet de forcer le cas de tests a échouer.

Initialisation des cas de tests

Il peut être fréquent que les cas de tests utilisent une instance d'un même ou jet ou nécessitent l'usage d'une classe pour l'accès à une base de donnée par exemple.

Pour réaliser ces operations de creation et destruction d'objet, nous pouvons utiliser **setUp() et tearDown()** qui sont appelés respectivement avant et après l'appel de chaque méthode contenant un cas de test.

*Utilisation de **setUP()** et **tearDown()*** :

- La méthode **setup()** est invoquée systématiquement avant l'appel de chaque méthode de tests. Sa mise en œuvre n'est donc requise que si toutes les méthodes de tests ont besoin de créer une instance d'un meme type.
- La méthode **tearDown()** peut permettre par exemple libérer une connexion a une base de données initialisée dans la méthode **setUp()** (dans l'exemple il remet a null une instance crée).

A partir de JUnit 4, on peut les remplacer respectivement par @Before et @After.

L'attribut @Test(excepted="...") de l'annotation Test attend comme valeur

la valeur de la classe de l'exception qui devrait être levée.

Exemple :

```
1  package com.jmdoudoux.test.junit;
2
3  import junit.framework.TestCase;
4
5  public class PersonneTest extends TestCase {
6
7    private Personne personne;
8
9    public PersonneTest(String name) {
10     super(name);
11   }
12
13   protected void setUp() throws Exception {
14     super.setUp();
15     personne = new Personne("nom1","prenom1"
16   }
17
18   protected void tearDown() throws Exception
19     super.tearDown();
20     personne = null;
21   }
22
23   public void testPersonne() {
24     assertNotNull("L'instance n'est pas créé
25   }
26
27   public void testGetNom() {
28     assertEquals("Le nom est incorrect", "no
29   }
30
31   public void testSetNom() {
32     personne.setNom("nom2");
33     assertEquals("Le nom est incorrect", "no
34   }
35
36   public void testGetPrenom() {
37     assertEquals("Le prenom est incorrect",
38   }
39
40   public void testSetPrenom() {
41     personne.setPrenom("prenom2");
42     assertEquals("Le prenom est incorrect",
43   }
44 }
```

Héritage d'une classe mère :

Il est possible de définir une classe mère a d'autres classes de tests notamment en fournissant des fonctionnalités communes. Ceci peut être pratique lorsqu'on souhaite que certaines initialisassions soit systématiquement exécutés.

Il est possible aussi de faire des initialisassions dans le constructeurs de la classe mere et d'invoquer ce constructeur dans les constructeurs des classes filles. @Before sera invoquée avant celle de la classe fille et @After sera invoquée avant celle de la classe mère.

Comment exécuter les tests :

JUnit propose trois applications nommées **TestRunner** pour exécuter les tests dans le terminal ou l'application graphique.

- une application console : junit.textui.TestRunner qui est très rapide et adaptée à une intégration dans un processus de générations automatiques.
- une application graphique avec une interface Swing : junit.swingui.TestRunner
- une application graphique avec une interface AWT : junit.awtui.TestRunner

Les entités suivantes doivent être dans la bibliothèque de classe :

- Le fichier junit.jar
- Les classes a tester et les classes des cas de tests
- Les classes et bibliothèques dont toutes ces classes dépendant.

Après un test, il y'a 3 états différents :

1. Échoué : exception de type AssertionFailedError est écrite
2. Erreur : exception non émise par le framework et non capturée a été levée dans les traitements
3. Succès

Sur JUnit 4 il n'y a plus de différence entre échoué et erreur.

L'échec d'un cas test seulement entraine l'échec de tous les tests.

Les cas d'échecs sont des tests contenants des bugs, le test est mal défini.

Comment exécuter un test :

Les applications graphiques AWT et Swing permettant l'exécution et l'affichage des résultats des cas de tests ne sont plus fournies avec JUnit 4.

JUnit 4 :

Compilation : javac -cp junit-4.12.jar;. UserDAOTest.java ProductDAOTest.java

Exécution : java -cp junit-4.12.jar;hamcrest-core-1.3.jar;. org.junit.runner.JUnitCore UserDAOTest ProductDAOTest

JUnit 3 :

Compilation :

```
javac -cp <junit-jar-file>;. TestClass1.java
TestClass2.java
```

Exécution :

java -cp <junit-jar>;<hamcrest-jar>;. org.junit.runner.JUnitCore TestClass1 TestClass2

Annexe :

Comparaison JUnit 3 et JUnit 4 :

JUnit 3 :

```
1  package com.jmdoudoux.test.junit;
2
3  import junit.framework.TestCase;
4
5  public class PersonneTest extends TestCase {
6
7    private Personne personne;
8
9    public PersonneTest(String name) {
10     super(name);
11   }
12
13   protected void setUp() throws Exception {
14     super.setUp();
15     personne = new Personne("nom1","prenom1"
16   }
17
18   protected void tearDown() throws Exceptior
19     super.tearDown();
20     personne = null;
21   }
22
23   public void testPersonne() {
24     assertNotNull("L'instance n'est pas créé
25   }
26
27   public void testGetNom() {
28     assertEquals("Le nom est incorrect", "nc
29   }
30
31   public void testSetNom() {
32     personne.setNom("nom2");
33     assertEquals("Le nom est incorrect", "nc
34   }
35
36   public void testGetPrenom() {
37     assertEquals("Le prenom est incorrect",
38   }
39
40   public void testSetPrenom() {
41     personne.setPrenom("prenom2");
42     assertEquals("Le prenom est incorrect",
43   }
44 }
```

JUnit 4 :

```
1  package com.jmdoudoux.test.junit4;
2
3  import static org.junit.Assert.assertEquals;
4  import static org.junit.Assert.assertNotNull
5
6  import org.junit.After;
7  import org.junit.Before;
8  import org.junit.Test;
9
10 public class PersonneTest {
11
12   private Personne personne;
13
14   @Before
15   public void initialiser() throws Exception
16     personne = new Personne("nom1","prenom1"
17   }
18
19   @After
20   public void nettoyer() throws Exception {
21     personne = null;
22   }
23
24   @Test
25   public void personne() {
26     assertNotNull("L'instance n'est pas créé
27   }
28
29   @Test
30   public void getNom() {
31     assertEquals("Le nom est incorrect", "no
32   }
33
34   @Test(expected=IllegalArgumentException.cl
35   public void setNom() {
36     personne.setNom("nom2");
37     assertEquals("Le nom est incorrect", "no
38     personne.setNom(null);
39   }
40
41   @Test
42   public void getPrenom() {
43     assertEquals("Le prenom est incorrect",
44   }
45
46   @Test
47   public void setPrenom() {
48     personne.setPrenom("prenom2");
49     assertEquals("Le prenom est incorrect",
50   }
51 }
```