Gestion des tests avec JUnit

JUnit

Outil de gestion des *tests unitaires* pour les programmes Java, JUnit fait partie d'un cadre plus général pour le test unitaire des programmes, le modèle de conception (*pattern*) XUnit (CUnit, ObjcUnit, etc.).

JUnit offre :

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

dans une archive Java `junit.jar` dont le principal paquetage est `junit.framework`.

Il existe un *plug-in* Eclipse pour JUnit.

`http://junit.org`

Principe du test avec JUnit

Rappels: test unitaire = test des parties (unités) du code
TestSuite = ensemble de TestCase

1. Pour chaque fichier Foo.java créer un fichier FooTest.java (dans le même repertoire) qui inclut (au moins) le paquetage `junit.framework.*`
2. Dans FooTest.java, pour chaque classe Foo de Foo.java écrire une classe FooTest qui hérite de TestCase
3. Dans FooTest définir les méthodes suivantes :
   - le constructeur qui initialise le nom de la suite de tests
   - setUp appelée avant chaque test
   - tearDown appelée après chaque test
   - une ou plusieurs méthodes dont le nom est prefixé par `test` et qui implementent les tests unitaires
   - `suite` qui appelle les tests unitaires
   - main qui appelle l'exécution de la suite

Ecrire les tests unitaires avec Junit

Dans les méthodes de test unitaire, les méthodes testées sont appelées et leur résultat est testé à l'aide d'assertions :

- assertEquals(a,b)
  teste si a est éqal à b (a et b sont soit des valeurs primitives, soit des objets possédant une méthode equals)
- assertTrue(a) et assertFalse(a)
  testent si a est vrai resp. faux, avec a une expression booléenne
- assertSame(a,b) et assertNotSame(a,b)
  testent si a et b référent au même objet ou non.
- assertNull(a) et assertNotNull(a)
  testent si a est null ou non, avec a un objet
- fail (message)
  si le test doit echouer (levée d'exception)

Exemple : Conversion binaire/entier

```
// File Binaire.java
import java.io.*;
public class Binaire {
  private String tab;
  public Binaire() { tab = new String(); }
  public Binaire(String b, boolean be) {
    tab = new String(b); if (be) revert(); }
  private void revert() {
    byte[] btab = tab.getBytes();
    for (int i = 0; i < (btab.length >> 1); i++) {
       byte tmp = btab[i]; btab[i] = btab[btab.length - i];
       btab[btab.length - i] = tmp;
    }
    tab = new String(btab);
  }
  public int getInt() {
    int nombre = 0;
    /* little endian */
    for (int i = tab.length()-1; i >= 0; i--) {
      nombre = (nombre << 1) +  (tab.charAt(i) - '0'); }
    return nombre; }
}
```

Exemple : Test Conversion binaire/entier

```
// Fichier BinaireTest.java
import junit.framework.*;
public class BinaireTest extends TestCase {
  private Binaire _bin;
  public BinaireTest(String name) { super(name); }
  protected void setUp() throws Exception {
    _bin = new Binaire(); }
  protected void tearDown() throws Exception {
    _bin = null; }
  public void testBinaire0() {
    assertEquals(_bin.getInt(),0); }
  public void testBinaire1() {
    _bin = new Binaire("01",false);
    assertEquals(_bin.getInt(),2);
  }
  public static Test suite() { // si
    return new TestSuite(BinaireTest.class); // or
    // TestSuite suite = new TestSuite();
    // suite.addTest(BinaireTest.suite());
    // return suite;
    }
```

Exécuter une suite de tests

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

1. En ligne de commande : (Junit est dans le CLASSPATH)

   ```
   java BinaryTest
   ```

2. En utilisant Eclipse :
   - ajouter JUnit dans les libraries du projet
   - exécuter BinaireTest.java comme test JUnit
   - Remarque : dans ce cas, les méthodes mai et suite ne sont pas nécessaires

Quelques règles de bonne conduite avec JUnit

- Ecrire les test en même temps que le code.
- Tester uniquement ce qui peut vraiment provoquer des erreurs.
- Exécuter ses tests aussi souvent que possible, idéalement après chaque changement de code.
- Ecrire un test pour tout bogue signalé (même s'il est corrigé).
- Ne pas tester plusieurs méthodes dans un même test : JUnit s'arrete à la première erreur.
- Attention, les méthodes privées ne peuvent pas être testées !