Test unitaire avec JUnit

Introduction

Le but du test unitaire est de tester individuellement chaque fonction d'un programme pour vérifier si elle fait ce que le programmeur pense qu'elle devrait faire : il s'agit pour le programmeur de tester une fonction, *indépendamment du reste du programme*, afin de s'assurer qu'elle répond aux spécifications [1].

Junit est un outil de test unitaire pour le langage java ; il est possible de l'utiliser en ligne de commande ; il existe aussi un plugin qui permet de l'utiliser directement depuis eclipse.

Le test unitaire permet de tester différents aspects d'une fonction (ou d'une classe) ; nous n'en présentons qu'un seul ici.

Test de validité des résultats :
- exécuter une fonction sur des données choisies avec soin et comparer le résultat obtenu avec le résultat attendu : s'ils coïncident, le test est réussi, dans le cas contraire, le test a échoué.
- exécuter la fonction sur des valeurs hors spécification pour vérifier le bon fonctionnement des assertions.

Pour plus de détails, voir : `http://rpouiller.developpez.com/tutoriels/java/tests-unitaires-junit4` et `http://jmdoudoux.developpez.com/cours/developpons/eclipse/chap-eclipse-junit.php`.

1 Incorporer une classe de test existante à un projet existant

1. Copier la classe de test dans le répertoire de la classe à tester ;
2. Afficher les propriétés du projet : « Project / Properties » rubrique « Java Build Path » puis onglet « Libraries » et enfin « ClassPath »
   - cliquer « Add Library » ; choisir JUnit ; cliquez Next
   - Version de Junit : choisir JUnit4
   - « Finish » ; « OK »

2 Exécuter un programme de test

- sélectionner la classe de test
- clic-droit puis Run as Junit Test

JUnit affiche les tests passés avec succès et ceux qui ont échoué : il faut bien entendu corriger le programme testé pour supprimer toutes les erreurs de test.

Remarque : le test ne permet pas de prouver que la classe testée est correcte : il permet juste de détecter des erreurs.

3 Pour créer une nouvelle classe de test

1. sélectionner la classe à tester
2. File / New / Junit Test Case puis compléter selon les besoins :
   - JUnit 4
   - package
   - nom de la classe de test (par exemple : TestUnitaire*ClasseATester*)
   - nom de la classe à tester
   - Ignorer les cases setUpxxx et tearxxx
3. Next :
   - sélectionner les fonctions et méthodes à tester
4. Finish

Lors de la première exécution, eclipse demande s'il faut ajouter JUnit4 au « Build Path » : accepter.

1. fr.wikipedia.org/wiki/Test_unitaire

4 Écrire les méthodes de test

Chaque méthode de test doit tester un cas précis : un constructeur doit mettre une instance dans un état précis qu'on peut vérifier ; une méthode de calcul doit produire une valeur prévue, ...

La validité des résultats attendus se teste avec des assertions JUnit (paquetage org.junit.Assert), à ne pas confondre avec les assertions java.

Une méthode de test est précédée de l'annotation @Test ; si cette méthode doit tester une assertion, l'annotation sera : @Test(expected = AssertionError.class) ; si la méthode testée doit avoir une durée maximale, il faut l'indiquer dans l'annotation (unité = ms) : @Test(timeout=1000). Cette dernière annotation est utile pour passer un jeu de tests dans le cas où l'une des méthodes testée risque de boucler.

1. tester la validité d'une expression :
   - assertTrue(*expression*) : assertion vérifiée si l'expression vaut vrai, échec sinon ;

     exemple : Assert.assertTrue(r1.getNumerateur() == 1) ;
   - assertTrue(*message, expression*) : assertion vérifiée si l'expression vaut vrai, échec sinon ; en cas d'échec, le message est affiché ;

     exemple : Assert.assertTrue("numérateur attendu : 1 mais r1.getNumerateur() obtenu", r1.getNumerateur() == 1) ;
   - assertFalse(*expression*) : assertion vérifiée si l'expression vaut faux, échec sinon ;
   - assertArrayEquals(t1, t2) : assertion vérifiée si les deux tableaux t1 et t2 ont même taille et mêmes éléments ; disponible pour les tableaux d'entiers et de caractères.

2. tester si les assertions java sont vérifiées : il faut appeler la méthode à tester avec des valeurs hors spécification : le test réussit si l'assertion java est déclenchée.

Il existe plusieurs autres assertions propres à JUnit qui ne sont pas présentées ici.

Exemple : vérifier si l'algorithme de tri par insertion trie correctement un tableau de deux éléments ;

```
@Test(timeout=10)
public void tri_2elements()
{
  int [] tnombres = { 999, 111 };      // tableau à trier
  int [] resu     = { 111, 999 };      // résultat attendu
  // trier le tableau tnombres avec l'algorithme du tri par insertion
  TriDicho.triInsertion(tnombres, tnombres.length);
  // vérifier si le tableau trié est égal au résultat attendu
  Assert.assertArrayEquals(tnombres, resu);
}
```

Ce test réussit si l'assertion assertArrayEquals(tnombres, resu) est vérifiée **et** si le test ne dure pas plus de 10 ms ; si l'assertion n'est pas vérifiée **ou** si le test dure plus de 10ms, le test échoue.