Introduction à JUnit

Emmanuel Coquery, Marie Lefevre, Lionel Médini

13 novembre 2012

1 Préambule

Ce TP se fait en binôme. Un compte-rendu par binôme est à rendre à la fin de la séance. Les éléments à mentionner dans le compte-rendu sont précisés dans le sujet : lisez donc bien le sujet. Le code n'est pas à rendre.

2 Découverte de JUnit

2.1 Introduction

JUnit est un framework de tests unitaire pour Java créé par Kent Beck et Erich Gamma. Il permet de simplifier considérablement l'écriture des tests en offrant un ensemble d'outils. Il permet notamment d'écrire des jeux de tests facilement réutilisables, et de regrouper les tests en fonction des besoins, des objets, etc. JUnit est aussi un framework extensible : par exemple, DBUnit propose des outils supplémentaires pour les tests spécifiques aux bases de données.

Utiliser des tests unitaires permet non seulement de tester un programme et donc d'en assurer la qualité, mais également d'éviter les régressions dans le code. Avec JUnit, on définit des tests réutilisables que l'on peut exécuter à volonté. Lorsque le code a progressé, on peut relancer une série de tests. Si les tests se passent toujours bien, alors on peut considérer que le code n'a pas régressé.

Un des intérêts de JUnit est qu'il permet d'écrire les tests avant le code. En procédant ainsi, au début, le programme n'est pas livrable car il ne passe aucun test. Plus le code progresse et plus le nombre de tests passés augmente.

Remarque 1. La version actuelle de JUnit est la version 4. Il existe une différence fondamentale entre JUnit 3 et JUnit 4 : l'introduction des annotations. Lorsque vous réaliserez les exercices de ce TP, faites bien attention à la version de JUnit que vous utilisez.

Remarque 2. JUnit est le framework le plus utilisé pour les tests unitaires en Java... mais il n'est pas le seul !

Pour la plupart des langages de programmation, il existe de framework xUnit ou équivalent :

- CUnit pour C ;
- cppunit, GoogleTest, BoostTest pour C++ ;
- JUnit et TestNG pour Java ;
- JSUnit pour JavaScript ;
- PHPUnit et SimpleTest pour PHP ;
- ...

Références utiles pour le TP et sources.

- http://www.junit.org/
- http://junit.sourceforge.net/
- https://github.com/KentBeck/junit/downloads

2.2 Exercices

2.2.1 Mettre en place des tests unitaires avec Nebeans

Suivez le tutoriel disponible à cette adresse : http://netbeans.org/kb/docs/java/junit-intro.html.
Notez dans le compte-rendu ce que vous retenez de ce tutoriel (1/2 page max).

2.2.2 Exercice 1

Dans cet exercice, nous allons chercher à comprendre un premier exemple de code utilisant la librairie JUnit. Observez le code suivant.

```
import junit.framework.*;
public class SimpleTest extends TestCase {
        public SimpleTest(String name) {
                super(name);
        }

        public void testSimpleTest() {
                int answer = 2;
                assertEquals((1+1), answer);
        }
}
```

- Créez une classe SimpleTest.java contenant le code présenté ci-dessus.
- Au besoin, adaptez cette classe pour la rendre conforme au mode de fonctionnement de JUnit 4.
- Exécutez le test.
- Introduisez une erreur dans votre classe (par exemple, en changeant la valeur de la variable answer) et relancez le test. Que se passe-t-il ? Expliquez-le dans votre compte-rendu.

2.2.3 Externalisation des tests

Dans cet exercice, nous allons voir comment définir une classe test qui regroupe plusieurs tests.
Voici le code de la classe à tester.

```
// A Class that adds up a string based on the ASCII values of its
// characters and then returns the binary representation of the sum.
public class BinString {
        public BinString() {}

                public String convert(String s) {
                        return binarise(sum(s));
                }

                public int sum(String s) {
                        if(s=="") return 0;
                        if(s.length()==1) return ((int)(s.charAt(0)));
                        return ((int)(s.charAt(0)))+sum(s.substring(1));
                }

                public String binarise(int x) {
                        if(x==0) return "";
                        if(x%2==1) return "1"+binarise(x/2);
                        return "0"+binarise(x/2);
                }
}
```

Voici maintenant le code de la classe test.

```
import junit.framework.*;
public class BinStringTest extends TestCase {
  private BinString binString;

        public BinStringTest(String name) {
                super(name);
        }

        protected void setUp() {
                binString = new BinString();
        }

        public void testSumFunction() {
                int expected = 0;
                assertEquals(expected, binString.sum(""));
                expected = 100;
                assertEquals(expected, binString.sum("d"));
```

```
                expected = 265;
                assertEquals(expected, binString.sum("Add"));
        }

        public void testBinariseFunction() {
                String expected = "101";
                assertEquals(expected, binString.binarise(5));
                expected = "11111100";
                assertEquals(expected, binString.binarise(252));
        }

        public void testTotalConversion() {
                String expected = "1000001";
                assertEquals(expected, binString.convert("A"));
        }
}
```

- En vous aidant de la javadoc de Junit, expliquez dans votre compte-rendu ce que fait la classe BinStringTest.
- Adaptez la classe test pour la faire fonctionner avec JUnit 4.
- Lancez les tests et commentez le résultat dans votre compte-rendu.
- En vous aidant des résultats des tests, corrigez les erreurs dans la classe principale.
- Relancez les tests jusqu'à ce que le programme fonctionne.
- Notez dans votre compte-rendu le processus suivi pour aboutir.

3 Problème

3.1 Phase 1. Choix du sujet

Chaque binôme choisit un sujet dans la liste ci-dessous.

- Écrire un programme permettant de calculer toutes les racines carrées des nombres compris entre A et B, A et B étant deux nombres entiers tels que A < B.
- Écrire un programme permettant d'afficher une matrice de taille MxN remplie par des nombres aléatoires compris entre A et B. Les valeurs M, N, A et B doivent être passées en paramètres.

3.2 Phase 2. Pour le sujet choisi...

- Écrivez le squelette de la classe principale, et commentez-le.
- Écrivez l'intégralité de la classe test. Cette classe doit comprendre :
  - Des assertions comme vu précédemment.
  - Des tests vérifiant que les exceptions sont bien levées quand elles doivent l'être (par exemple : de mauvais paramètres sont passés dans l'appel).
  - Des tests vérifiant que les boucles s'effectuent dans des temps raisonnables.
- Formatez correctement les sorties de vos tests en utilisant les annotations @Before et @After.

3.3 Phase 3. Échange des sujets

- Échangez vos classes avec un binôme ayant traité un sujet différent du vôtre.
- Implémentez la classe principale.
- Appliquez les tests.
- Itérez jusqu'à obtention d'un programme fonctionnel et satisfaisant l'ensemble des tests.

4 Et maintenant, passage à l'échelle...

Envoyez votre compte-rendu à votre intervenant de TP et libérez-vous de votre binôme. Retrouvez vos collègues du projet MultiMIF.

Dans le cadre de ce projet, vous avez rendu votre document de fin d'étude préliminaire. Ce document contient, entre autres, la planification de la première itération de votre conception logicielle. Cette planification décrit le code qui doit être produit et par qui pour la fin de l'itération. Définissez donc chacun les tests relatifs au code que vous devrez fournir en utilisant JUnit.