TD n° 1
Les tests unitaires

Le test unitaire a pour objet de démontrer qu'une partie modulaire autonome d'un programme est **fonctionnellement correcte**, et ce **indépendamment de son environnement**. On pourra donc utiliser en confiance cette "brique de base" en vue d'un assemblage qui constituera le programme.

Dans le cadre de la **programmation objet** cette partie modulaire sera très naturellement la **classe**.

1 Les tests unitaires en Java

2.1 Junit

Les principes que devraient suivre un framework de tests unitaires sont :

1. Indépendance des exécutions : Exécutions indépendantes des tests unitaires
2. Identification des erreurs : Détection d'erreurs test par test
3. Sélection des tests : Facilité pour identifier les tests à exécuter

Un framework Java permettant de tester unitairement les composants d'un logiciel écrit en Java :

- Junit pour Java

D'autres frameworks suivant la même philosophie sont disponibles pour d'autres langages. Ils constituent la famille des frameworks xUnit :

- NUnit pour .NET
- UNIT++ pour C++

2.2 Les cas de test

Un cas de test (test case) *est un ensemble de tests portant sur une classe du logiciel.*

Les tests sont regroupés dans une classe dérivant de :
junit.framework.TestCase
Chaque test est codé sous la forme d'une méthode :
1 - sans paramètre,
2 - sans valeur de retour,
3 - dont le nom est préfixé conventionnellement par "test"

2.3 Squelette d'un cas de test

Supposons que notre logiciel contienne une classe *Calculatrice* possédant une méthode *add* effectuant une addition.


```
public class Calculatrice {
   public double add(double number1, double number2) {
      return number1 + number2;
   }
}
```

Ecrivons un cas de test *CalculatriceTest* pour tester cette méthode, composé de l'ensemble des tests suivants :

- une addition (10, 10)
- une addition (10, 20)
- une addition (10, 30)

Avec Junit, le cas de test correspondant prend la forme suivante :

```
import static org.junit.Assert.*;
import org.junit.Test;
public class CalculatriceTest extends TestCase {

   @Test
   public void testAdd1() {
      Calculatrice calculatrice = new Calculatrice();
      double result = calculatrice.add(10, 10);
      assertEquals(20, result, 0);
   }

   public void testAdd2() {
      ...
   }

   public void testAdd3() {
     ...
   }
}
```

Principe de sélection : annotations indiquant les méthodes à exécuter. L'ordre d'exécution est arbitraire.

Il n'est pas possible de réutiliser une variable d'instance entre les méthodes de tests, car chaque méthode de test est exécutée dans une nouvelle instance de la classe de test.

Principe d'identification : les erreurs sont détectés par les assertions.

2 Les assertions et l'échec

Dans JUnit, les assertions sont des méthodes permettant de comparer une *valeur obtenue* lors du test avec *une valeur attendue*.

Les méthodes de test disponibles sont les suivantes :

Méthode de test	Description
fail(String)	Let the method fail, might be usable to check that a certain part of the code is not reached.
assertTrue(true); / assertTrue(false);	Will always be true / false. Can be used to predefine a test result if the test is not yet implemented.
assertsEquals([String message], expected, actual)	Tests if the values are the same. Note: for arrays the reference is checked not the content of the arrays.
assertsEquals([String message], expected,	Usage for float and double; the tolerance is the

actual, tolerance)	number of decimals which must be the same.
assertNull([message], object)	Checks if the object is null.
assertNotNull([message], object)	Checks if the object is not null.
assertSame([String], expected, actual)	Checks if both variables refer to the same object.
assertNotSame([String], expected, actual)	Checks that both variables refer not to the same object.
assertTrue([message], boolean condition)	Checks if the boolean condition is true.

Pour les booléens, les assertions suivantes sont disponibles :

```
assertEquals (boolean attendu,boolean obtenu) ;
assertFalse (boolean obtenu) ;
assertTrue (boolean obtenu) ;
```

Pour les objets, les assertions suivantes sont disponibles, quel que soit leur type :

```
assertEquals (Object attendu,Object obtenu) ;
assertSame (Object attendu,Object obtenu) ;
assertNotSame (Object attendu,Object obtenu) ;
assertNull (Object obtenu) ;
assertNotNull (Object obtenu) ;
```

Il existe une variante pour chaque assertion prenant une chaîne de caractères en premier paramètre. Cette chaîne de caractères contient le message à afficher si le test échoue.

```
assertXxxx(message, ...);
```

Dans le cas des types correspondant à des nombres réels (float, double), un paramètre supplémentaire, le delta, est nécessaire, car les comparaisons ne peuvent être tout à fait exactes du fait des arrondis.

```
assertEquals(expected, actual, delta);
```

3 Les annotations

Les annotations disponibles dans JUnit sont les suivantes :

Annotation	Description
@Test	Annotation @Test identifies that this method is a test method.
@Before	Will perform the method() before each test. This method can prepare the test environment, e.g. read input data, initialize the class).
@After	La méthode est exécutée après chaque test.
@BeforeClass	Will perform the method before the start of all tests. This can be used to perform time intensive activities for example to connect to a database.
@AfterClass	Will perform the method after all tests have finished. This can be used to perform clean-up activities for example to disconnect to a database.
@Ignore	Will ignore the test method, e.g. useful if the underlying code has been changed and the test has not yet been adapted or if the runtime of this test is just too long to be included.

@Test(expected=IllegalArgumentException.class)	Tests if the method throws the named exception.
@Test(timeout=100)	Fails if the method takes longer than 100 milliseconds.

SetUp() et tearDown() : Les méthodes setUp() et tearDown() peuvent être rajoutées à une classe de test, dans ce cas elles sont exécutées avant et après, respectivement, chaque exécution de cas de test. Ce comportement est équivalent à celui induit par @Before et @After.

Test des exceptions : Pour tester qu'une exception a été correctement levée, on peut également ajouter une méthode de cette forme :

```
public void testException() {
        try {
            Object o = ...;
            fail("devrait lever une exception TYPEDELEXCETIONICI");
        }
         catch (TYPEDELEXCETIONICI succes) {
        }
}
```

Les temporisations

Exemple :

```
public class TestDefaultController
{
[...]
   @Test(timeout=130)
   public void testNomMethode()
   {
    ....
   }
```

Saut de tests

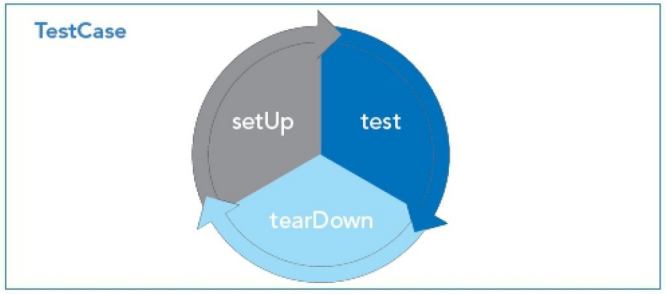
Exemple :

```
@Test(timeout=130)
@Ignore(value="Ignorer pour l'instant jusqu'à définition d'une limite")
public void testNomMethode()
   {
   ....
   }
```

4 Cycle de vie d'un cas de test Junit

Le cycle de vie d'un cas de test dans le framework Junit est le suivant :

1. Execute public void setUp().
2. Call a test-prefixed method.
3. Execute public void tearDown().
4. Repeat these steps for each test method.



5 Les suites de tests

Les suites permettent de regrouper les cas de tests dans un ou plusieurs ensembles permettant de simplifier le *lancement de ces tests de manière collective*.

De tels ensembles sont appelés des **suites de tests**.

La syntaxe d'une *suite de tests* est la suivante :

```
@RunWith(value=org.junit.runners.Suite.class)
@SuiteClasses(value={TestCaseA.class, TestCaseB.class, ...})
        public class MaTestSuite {
        }
```

La syntaxe d'une *suite de suites de tests* est la suivante :

```
@RunWith(value=Suite.class)
@SuiteClasses(value = {TestCaseA.class})
        public class TestSuiteA {
        }
@RunWith(value=Suite.class)
@SuiteClasses(value = {TestCaseB.class})
        public class TestSuiteB {
        }

@RunWith(value = Suite.class)
@SuiteClasses(value = {TestSuiteA.class, TestSuiteB.class})
        public class MaTestSuiteSuite{
        }
```

Vous pouvez exécutez n'importe quelle *classe de test ou suite de test ou suite de suite de test*.

Alternativement vous pouvez créer une suite de tests par programme :


```
public class AllTests extends TestSuite {
  static public Test suite() {
    TestSuite suite = new TestSuite();
    suite.addTestSuite(SimpleTest.class);
    return suite;
  }
}
```

Vous pouvez également exécuter les tests en ligne de commande :

java junit.textui.TestRunner nomMonTest

Création et exécution d'une suite de tests

- Sélectionner le répertoire qui accueillera la classe de la suite de test (ex : testsunit/)
- Faire File > New > Other... > Java > JUnit > JUnit Test Suite.
- Renseigner le paquetage, le dossier de création, le nom de la suite et sélectionner les cas de test à inclure dans la suite
- Faire un clic-droit sur la suite puis sélectionner Run As ou Run ...
- Si tout se passe bien, vous devrez obtenir une barre verte

6 Application

- Créer et exécuter des tests d'acceptations selon les exigences suivantes. Modifier le programme si nécessaire.

  1. Créer une classe de tests DistributeurCafeTest
     - ajouterInventaireExceptionTest : levée d'une exception
     - ajouterInventaireTest : aucune exception n'est levée
     - preparerCafeTest : retourne la monnaie de l'achat
  2. Créer RecetteTest and InventaireTest également.
  4. Créer une suite de tests pour lancer l'ensemble des tests
  5. Créer un ensemble de cas tests suffisant pour détecter 5 bugs.
  6. Une fois détectés, corrigez-les

  Ajout d'une recette : trois peuvent être rajoutés de noms différents
  Supprimer la recette : si la recette existe dans la liste des recettes
  Editer la recette : si la recette existe déjà. Le nom de la recette ne peut être changé
  Ajout inventaire : l'inventaire est ajouté à la quantité existante. Un inventaire diminue uniquement lors de l'achat d'une boisson
  achat boisson : si la boisson existe et si l'acheteur a suffisamment d'argent, il peut prendre une boisson et récupérer sa monnaie

7 Exécuter les tests par programme

La classe *org.junit.runner.JUnitCore* fournit la méthode runClasses() qui vous permet d'exécuter un ou plusieurs cas de tests. Comme résultat vous recevez un objet dont le type est org.junit.runner.Result. Cet objet peut être utilisé pour réupérer des informations sur les tests.

Créez dans le dossier de vos tests une nouvelle classs MyTestRunner comme ceci.
Cette classe exécutera votre classe de tests et écrira les éventuels messages d'échecs à la console.

```
package de.vogella.junit.first;

import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;

public class MyTestRunner {
        public static void main(String[] args) {
          Result result = JUnitCore.runClasses(MyClassTest.class);
          for (Failure failure : result.getFailures()) {
                  System.out.println(failure.toString());
          }
        }
}
```

8 Tests paramétrables

Exemple :

```
@RunWith(value=Parameterized.class)
public class ParameterizedTest {
    private double expected;
    private double valueOne;
    private double valueTwo;
    @Parameters
    public static Collection<Integer[]> getTestParameters() {
      return Arrays.asList(new Integer[][] {
          {2, 1, 1},  //expected, valueOne, valueTwo
          {3, 2, 1},  //expected, valueOne, valueTwo
          {4, 3, 1},  //expected, valueOne, valueTwo
      });
    }
    public ParameterizedTest(double expected,
       double valueOne, double valueTwo) {
       this.expected = expected;
       this.valueOne = valueOne;
       this.valueTwo = valueTwo;
    }
    @Test
    public void sum() {
       Calculatrice calc = new calculatrice();
       assertEquals(expected, calc.add(valueOne, valueTwo), 0);
    }
}
```

9 Extensions Junit

Add-on	URL	Use
DbUnit	http://dbunit.sourceforge.net/	Provides functionality relevant to database testing including data loading and deleting, validation of data inserted, updated or removed from a database, etc.
HttpUnit	http://httpunit.sourceforge.net/	Impersonates a browser for web based testing. Emulation of form submission, JavaScript, basic http authentication, cookies and page redirection are all supported.
EJB3Unit	http://ejb3unit.sourceforge.net/	Provides necessary features and mock objects to be able to test EJB 3 objects out of container.
JUnitPerf	http://clarkware.com/software/JUnitPerf.html	Extension for creating performance and load tests with JUnit.