## Forme générale d'une classe JUnit

```
import org.junit.*;
import static org.junit.Assert.*;

// le nom de la classe de test doit etre NomClasseATesterTest
public class XXXTest {

    // definition des acteurs necessaires aux tests
    // sous forme d'attributs

    /** Documenter cette methode pour savoir comment sont
     *  initialises les acteurs
     */
    @Before public void setUp() {
        // initialisation des acteurs
        // methode executee avant chaque test
    }

    @After public void tearDown() {
        // nettoyage eventuel des tests
        // methode executee apres chaque test
    }

    /** Ne pas oublier de documenter les methodes de test
     *  pour savoir ce que fait le test !
     *  Choisir un nom de methode explicite.
     */
    @Test public void testFooMethode() {
        // eventuellement, assertions a verifier avant d'effectuer
        // une action sur les acteurs
        // initialisation de variables locales

        // action(s) sur les acteurs

        // assertions a verifier apres l'action
    }

    ...
}
```

## Lancer un test

```
java org.junit.runner.JUnitCore ClasseTest
```

## Vérifier qu'une exception est levée

Le test suivant réussira si une exception de type **package.**ExceptionAttendue est levée durant l'exécution du test :

```
@Test(expected=package.ExceptionAttendue.class)
public void testFoo() throws ExceptionAttendue {
   ...
}
```

## Faire une suite de tests

Écrire une classe **de test** lançant les classes de test Classe1Test, Classe2Test et Classe3Test :

```
import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses( { Classe1.class, Classe2.class, Classe3.class })
public class TestSuite {}
```

## Assertions disponibles

Dans ce qui suit :
– exp est la valeur attendue
– actual est la valeur obtenue lors du test

| | |
|---|---|
| `assertEquals(String exp, String actual)` | comparerdeux chaînes de caractères |
| `assertEquals(double exp, double actual, double delta)` | comparer deux réels avec une précision `delta` |
| `assertEquals(int expected, int actual)` | comparer deux entiers |
| `assertFalse(boolean condition)` | `condition` doit s'évaluer à **`false`** |
| `assertTrue(boolean condition)` | `condition` doit s'évaluer à **`true`** |
| `assertNotNull(Object ref)` | la référence `ref` ne doit pas être **`null`** |
| `assertNull(Object ref)` | la référence `ref` doit être **`null`** |
| `assertSame(Object exp, Object actual)` | exp et actual sont deux références vers le même objet |
| `assertNotSame(Object exp, Object actual)` | exp et actual sont deux références vers deux objets différents |
| `assertEquals(Object exp, Object actual)` | exp et actual sont deux références vers des objets égaux au sens de `equals` |
| `assertNotEquals(Object exp, Object actual)` | exp et actual sont deux références vers deux objets différents au sens de `equals` |
| `assertEquals(T[] exp, T[] actual)` | exp et actual sont deux références vers des tableaux de type `T` égaux |

Toutes ces méthodes ont une version surchargée possédant en plus un argument de type `String` en première position servant de message d'erreur en cas de falsification de l'assertion.

## Références


[1] *JUnit*. URL : http://www.junit.org.

[2] T. Husted et V. Massol. *JUnit in action*. Manning Publications, 2003.

[3] A. Hunt et D. Thomas. *Pragmatic Unit Testing in Java with Junit*. The Pragmatic Bookshelf, 2003.