Les tests unitaires : junit

Table des matières

Présentation générale

le test unitaire

- intégration de tests unitaires dans le cycle de développement
- factorisation-capitalisation des tests
- structuration des tests en hiérarchie de « suites de tests »
- outil « open source » développé dans le cadre d'un projet autonome

les concepts

- un **TestCase** : un ensemble de méthodes de test (implémentées sous la forme ***testXXX()***)
- une **TestSuite** est une hiérarchie **TestCase** et de **TestSuite**. .

- une **TestSuite** invoque automatiquement tous les tests définis dans la hiérarchie **TestCase**.
- Chaque méthode de test s'exécute isolément dans sa "fixture" (espace propre)

Aspect programmatique (1)

la programmation d'un TestCase

- étendre la classe : **junit.framework.TestCase.**
- définir des méthodes public testXXX()
- tester la validité des résultats via une variante de **assert().**
  assertTrue(boolean res),
  assertFalse(boolean res)
  assertEquals([<type> attendu,<type> obtenu);
  fail();

  assertTrue(String msg , boolean res)
  assertFalse(String msg, boolean res)
  assertEquals(String msg, <type> attendu,<type> obtenu);
  fail(String);

assertEquals(boolean, boolean)
assertEquals(int, int)
assertEquals(long, long)
assertEquals(Object, Object)
……

```
package tests;
import mois.Mois;
import junit.framework.TestCase;

public class TestMoisIllegal extends TestCase {

    public void testMois1() {
        try {
            Mois mois = new Mois(2004, 12);
            fail("aurait du generer une IllegalArgumentException");
        } catch(IllegalArgumentException ex) {}
    }
}
```

```
package tests;
import mois.Mois;
import junit.framework.TestCase;

public class TestMois2 extends TestCase {

    public void testMois1() {
        Mois mois = new Mois(2, 2004);
        assertEquals(29, mois.nombreJours());
    }
    public void testMois2() {
        Mois mois = new Mois(2, 2005);
        boolean res = (28 == mois.nombreJours()) ;
        assertTrue("valeur attendue : 28", res);
    }
}
```

Aspect programmatique (2)

l'exécution d'un TestCase

- chaque test s'exécute dans sa "fixture" (espace propre) → le constructeur est invoqué pour chaque méthode de test
- possibilité de définir une méthode setUp et tearDown invoquée avant/après chaque méthode de test

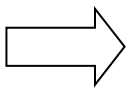
```
import junit.framework.TestCase;

public class JunitExemple0 extends TestCase {
    private int x;

    public JunitExemple0() { System.out.println("ctor"); }

    public void setUp() { System.out.println("setUp ");   }
    public void tearDown() { System.out.println("teardown"); }

    public void test0() { System.out.println("test0 : " + x++); }
    public void test1() { System.out.println("test1 : " + x++); }
    public void test2() { System.out.println("test2 : " + x++); }
    public void test3() {    System.out.println("test 3: " + x++); }
}
```



```
ctor
ctor
ctor
ctor
setUp
test0 : 0
teardown
setUp
test1 : 0
teardown
setUp
test2 : 0
teardown
setUp
test3 : 0
teardown
```

Aspect programmatique (3)

la programmation d'une TestSuite

- définir une méthode statique **suite()** créant une instance de **junit.framework.TestSuite.**
  - ajout implicite de toutes les méthodes de test

```
package tests;
import junit.framework.Test;
import junit.framework.TestSuite;

public class AllTests {
    public static Test suite() {
        TestSuite suite = new TestSuite("Test pour Mois");
        suite.addTestSuite(TestMois2.class);
        suite.addTestSuite(TestMoisIllegal.class);
        return suite;
    }
}
```

  - possibilité d'ajout explicite de certaines méthodes de TestCase

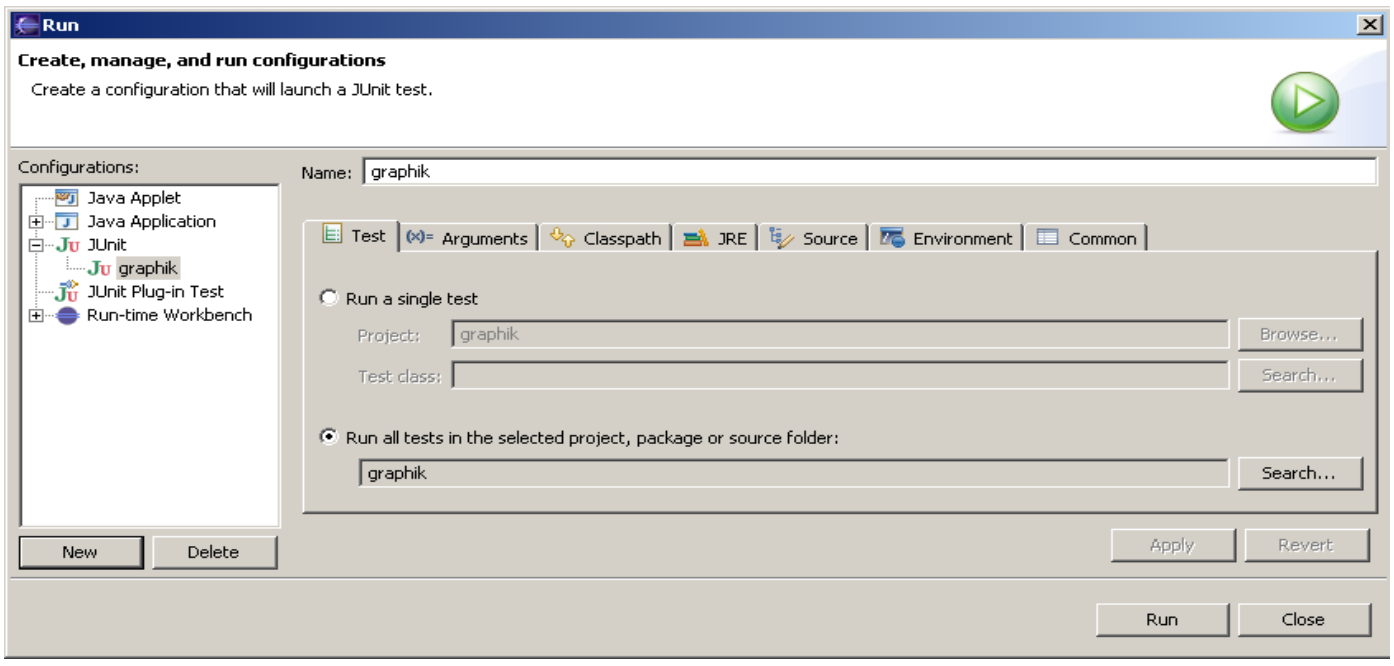
l'organisation des tests

- créer une suite pour chaque package (mettre les tests dans le même package que le code à tester)
- Mettre le code de tests dans une arborescence de répertoires distincte de celle du code à tester
- mettre les tests dans le process de build

Le lancement d'une suite de tests (1)

le lancement dans Eclipse

- **par l'intermédiaire de Run**

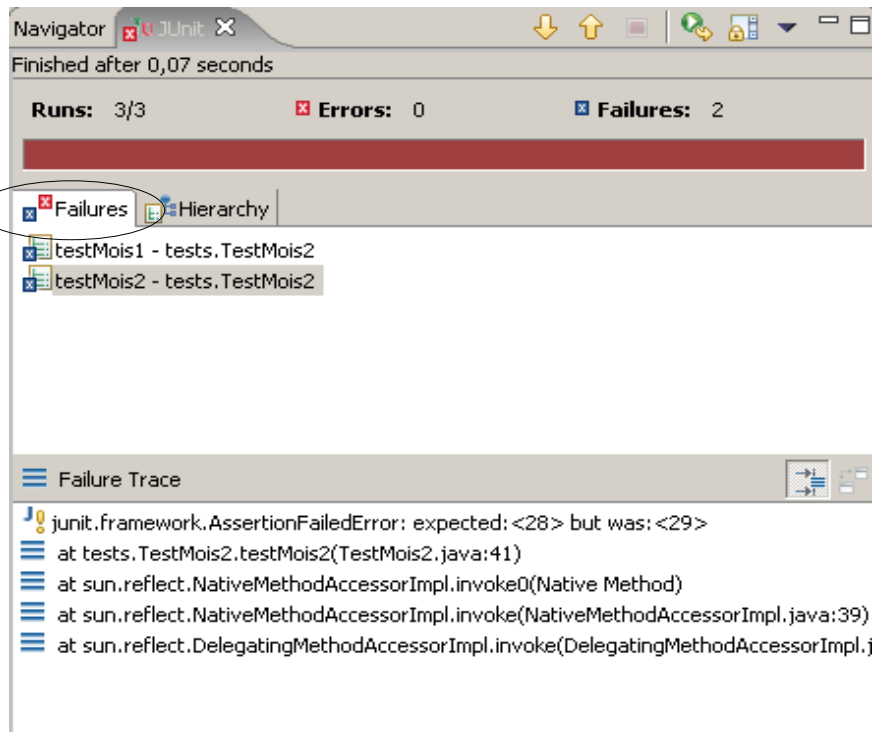


La vue junit (1)

l'onglet Failures

- visualise l'ensemble des tests en erreur [indication du résultat]
  bleu: erreur sur assert
  rouge : erreur avec sortie sur exception

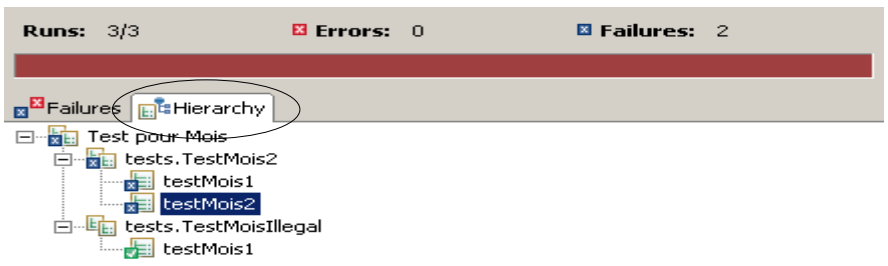
- pile des appels au moment de l'erreur
- (double) click sur la frame active l'éditeur de la classe correspondante avec positionnement sur la ligne.



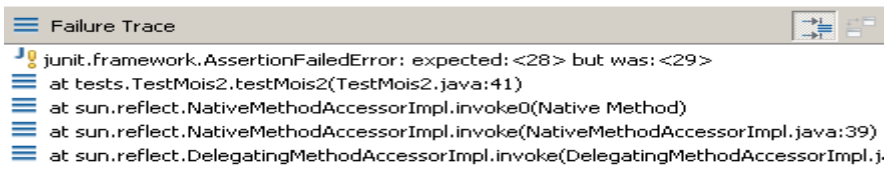
La vue junit (2)

l'onglet Hierarchy

- visualise l'ensemble des tests [indication du résultat]
  vert : OK
  bleu: erreur sur assert
  rouge : erreur avec sortie sur exception



- pile des appels au moment de l'erreur
- double click sur la frame active l'éditeur de la classe correspondante avec positionnement sur la ligne.



Le lancement d'une suite de tests (2)

le lancement hors Eclipse

- une interface textuelle : affiche OK ou failure
  correspond à junit.textui.TestRunner.run(AllTests.class);
- une interface graphique : affiche une barre de progression verte ou rouge
  correspond à java junit.swingui.TestRunner.run(AllTests.class);

```
package tests;

public class TextTests {

    public static void main(String[] args)  {
        if (args.length != 0 & ‘’t’’.equals(args[0])) junit.textui.TestRunner.run(AllTests.class);
        else junit.swingui.TestRunner.run(AllTests.class);
    }
}
```