VERSION ENSEIGNANT

Eclipse – JUnit : test automatique

	test unitaire	**test perso**
reproductible	oui	non
compréhensible	oui	non
documenté	oui	non
conclusion	bon pour le service	à bannir

1 Tutoriel

Nous allons voir comment écrire notre premier test unitaire pas à pas. Pour ce faire, nous utiliserons le projet de la calculatrice comme exemple.

Dans ce tutoriel, nous allons suivre les étapes suivantes :

1. Créer le projet de calculatrice.
2. Ecrire la classe d'addition.
3. Créer le répertoire des tests dans lequel seront classés tous les tests. La façon la plus classique consiste à créer un dossier de tests symétrique au répertoire des sources pour y classer les tests selon la même organisation de paquet.
4. Ecrire la classe de tests voulue.

Eclipse-JUnit

— Cliquez sur File – New – Java Project

— Nommez votre projet calculatrice en laissant les options par défaut puis cliquez sur OK.

— Ajoutez un package math dans le répertoire src.

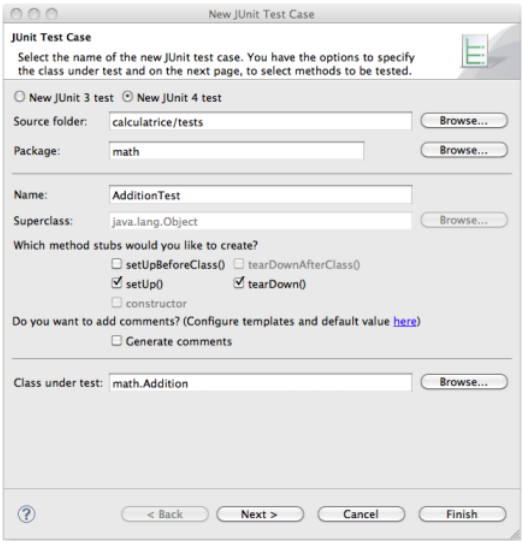
— Ajoutez la classe Addition dans le répertoire src.

```
1  package math;
2
3  class Addition {
4     public Long calculer(Long a, Long b) {
5        return a+b;
6     }
7     public Character lireSymbole() {
8        return '-';
9     }
10 }
```

— Ajoutez un nouveau dossier de sources nommé tests au même niveau d'arborescence que src.

— Dans l'explorateur de paquets, faites un clic droit sur la classe Addition.

— Dans le menu contextuel, cliquez sur New – JUnit Test Case. Un panneau s'affiche alors :



Dans ce panneau :

— Sélectionnez le bouton radio New JUnit 4 test ou plus.

— Changez le dossier Source folder pour tests.

— Nommez la classe AdditionTest.

— Cochez les cases setUp() et tearDown().

— Dans le champ Class under test, saisissez math.Addition.

— Enfin cliquez sur Finish.

Eclipse va remarquer que la bibliothèque de JUnit est absente du projet et vous propose d'ajouter automatiquement cette dernière au projet.

Eclipse va maintenant créer automatiquement le squelette de la classe de test :

```
1  package math;
2
3  import org.junit.After;
4  import org.junit.Before;
5
6  public class AdditionTest {
7
8      @Before
9      public void setUp() throws Exception {
10     }
11
12     @After
13     public void tearDown() throws Exception {
14     }
15
16 }
```

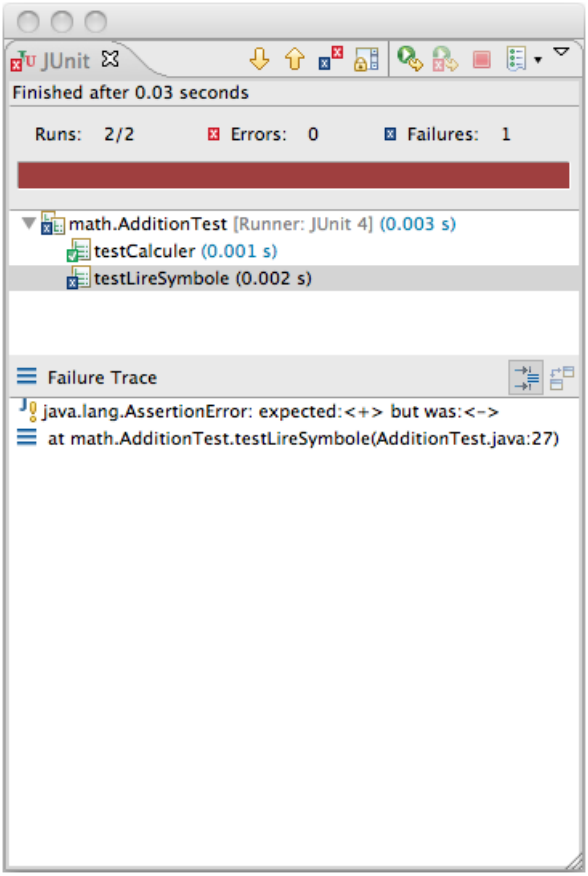
Il ne reste plus alors qu'à remplir cette dernière.

```java
package math;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import static org.junit.Assert.*;

public class AdditionTest {
   protected Addition op;

   @Before
   public void setUp() {
      op = new Addition();
   }

   @After
   public void tearDown() {
   }

   @Test
   public void testCalculer() throws Exception {
      assertEquals(new Long(4),
                   op.calculer(new Long(1), new Long(3)));
   }

   @Test
   public void testLireSymbole() throws Exception {
      assertEquals((Character)'+', op.lireSymbole());
   }
}
```

— Dans l'explorateur de paquets, faites un clic droit sur la classe AdditionTest.

— Dans le menu contextuel, cliquez sur Run As – JUnit test.

— Enfin, le premier rapport de tests s'affiche !



La barre de progression est rouge, indiquant qu'au moins un test est en échec. Le rapport d'erreur permet de visualiser les tests en échec et d'afficher la cause et l'origine du problème. Dans ce cas, une erreur s'est glissée sur le symbole de l'addition !

2 Exercices

Exercice 1. Répéter l'expérience avec l'opérateur de division.

Exercice 2. Date.

Les fichiers incomplets (et même partiellement erronés !) Date.java et DateTest.java sont fournis (voir ENT).

1. Créer un projet Date avec deux dossiers sources, un pour Date.java et l'autre pour les tests (DateTest.java).
2. Compléter les tests unitaires DateTest pour couvrir les différentes méthodes et/ou états possibles.
3. Définir la méthode *nextDay* dans la classe Date. La mettre au point en la testant avec un nouveau test unitaire DateNextDayTest.

3 Tutoriel (suite)

La notation javadoc suivante permet de vérifier qu'un test ne dépasse pas une durée. Au delà de cette durée, le test passe en erreur. La durée en millisecondes est passée en paramètre à l'annotation.

```
import org.junit.Test;

public class TestDureeLimitee {

@Test(timeout=1000)
public void dureeRespectee() {

}

@Test(timeout=1000)
public void dureeNonRespectee() throws InterruptedException {
Thread.sleep(10000);
}
}
```

La notation javadoc suivante permet de pas passer un test.

```
import org.junit.Assert;
import org.junit.Ignore;
import org.junit.Test;

public class TestIndisponibilite {

@Test
public void nonIgnore1() {

}

@Test
public void nonIgnore2() {
Assert.fail("Echec");
}

@Ignore
@Test
public void ignore() {
Assert.fail("echec ignore");
}
}
```

Les annotations javadoc suivantes permettent d'indiquer une méthode qui sera exécutée avant c**haque test** et un méthode qui sera exécutée après **chaque test**.

```
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class TestAvantApres {

@Before
public void avantTest() {
System.out.println("------------------------");
System.out.println("Avant Test");
}

@After
public void apresTest() {
System.out.println("Apres Test");
System.out.println("------------------------");
}

@Test
public void premierTest() {
System.out.println("Premier Test");
}

@Test
public void deuxiemeTest() {
System.out.println("Deuxieme Test");
}

@Test
public void troisiemeTest() {
System.out.println("Troisieme Test");
}
}
```

Les notations javadoc suivantes permettent d'indiquer une méthode qui sera exécutée avant l'ensemble des tests **d'un cas de tests** et une méthode qui sera exécutée après l'ensemble des tests **d'un cas de tests**.

```
import org.junit.AfterClass;
import org.junit.BeforeClass;
import org.junit.Test;

public class TestAvantApresEnsemble {

@BeforeClass
public static void avantTests() {
System.out.println("------------------------");
System.out.println("Avant Tests");
System.out.println("------------------------");
}

@AfterClass
public static void apresTests() {
System.out.println("------------------------");
System.out.println("Apres Tests");
System.out.println("------------------------");
}

@Test
public void premierTest() {
System.out.println("Premier Test");
}

@Test
public void deuxiemeTest() {
System.out.println("Deuxieme Test");
}

@Test
public void troisiemeTest() {
System.out.println("Troisieme Test");
}
}
```

Exercice 3.

- Ecrire le programme java "Chrono" qui permet de faire un compte à rebours en prenant comme paramètre d'entrée une valeur en second.
- Définir 10 cas de tests différents en limitant à chaque fois la durée du test.
- Faire passer 5 test sur 10.
- Utiliser les before et les after des cas de tests.