Tests unitaires avec JUnit (dans Eclipse)

Les tests unitaires sont des développements réalisés pour vérifier le bon fonctionnement de parties spécifiques d'un logiciel. Avec JUnit, les tests sont placés dans des classes de test.

Ajouter une classe de test dans un projet Java (classique) sous Eclipse

1) Ajouter un répertoire de sources

New / Source Folder / -> nom du répertoire de test

2) Créer une classe de test

New / JUnit Test Case

Nom = nom de la classe de test (ex. TestProduit)

Si le code de la librairie JUnit n'est pas disponible, l'IDE demande si l'on souhaite l'ajouter.

Le résultat est une classe dont les méthodes sont annotées (@Test,@Before …)

```
package istia.ei4.magasin;
import static org.junit.Assert.*;
import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;

public class TestProduit {

	@BeforeClass
	public static void setUpBeforeClass() throws Exception{
		System.out.println("Une fois avant tous les tests");
	}

	@AfterClass
	public static void tearDownAfterClass() throws Exception{
		System.out.println("Après tous les tests");
	}

	@Before
	public void setUp() throws Exception {
		System.out.println("Juste avant un test");
	}

	@After
	public void tearDown() throws Exception {
		System.out.println("Juste après un test");
	}

	@Test
	public void test1() {
		System.out.println("Test1");
	}

	@Test
	public void test2() {
		System.out.println("Test2");
	}
}
```

3) Exécuter les tests (sous Eclipse)

Clik droit sur la classe de test / Run As / JUnit Test

L'exécution produit des sorties console qui reflètent l'ordre des opérations.

```
Une fois avant tous les tests
Juste avant un test
Test1
Juste après un test
Juste avant un test
Test2
Juste après un test
Après tous les tests
```

Un compte rendu indique dans l'IDE que les deux tests ont ici été exécutés avec succès.

4) Les assertions vérifiables

Au sein des tests, on va écrire du code pour vérifier que les objets sont réellement dans l'état attendu.

```
import static org.junit.Assert.*;
...
public class TestProduit {

	@Test
	public void test1() {
		IProduit p = new Produit("BN", 2.4);
		assertEquals(2.4,p.getPrix(),0.001); // 2.4 == prix (à 0.001)
		assertEquals("BN",p.getNom());
		// équivaut à org.junit.Assert.assertEquals("BN",p.getNom());
	}

	@Test
	public void test2() {
		IProduit p = new Produit("BN", 2.4);
		p.setPrix(3.1);
		assertEquals(3.1,p.getPrix(),0.001);
	}

	@Test(expected=NullPointerException.class)
	public void test3(){
		IProduit p=null;
		System.out.printf("prix=%f",p.getPrix());
	}
}
```

Les vérifications sont réalisées grâce à des méthodes de classe (fonctions statiques) fournies par la classe **`org.junit.Assert`**. Noter que l'import static permet de simplifier l'écriture des appels. Une assertion non vérifiée produit un échec du test (indiqué dans le rapport d'exécution).

Quelques vérifications possibles :

```
assertTrue(boolean)/assertFalse(boolean)
assertEquals(expected,actual)/assertEquals(expectedDouble,actual,precision)
assertNull(ref)/assertNotNull(ref)
assertSame(expectedRef,actualref)/assertNotSame(expectedRef,actualRef)
fail() : échoue toujours
```

Vérification que du code déclenche une exception : test3 est censé lever une exception de type NullPointerException.

```
@Test(expected=NullPointerException.class)
public void test3(){
	IProduit p=null;
	System.out.printf("prix=%f",p.getPrix());
}
```