Atelier N° 1 : Test Unitaire avec JUnit (Java)

Objectif : Dans cet atelier, nous allons voir ce qu'est un « test unitaire ». Il s'agit du test le plus couramment utilisé et le plus important. Nous verrons comment créer des tests unitaires pour votre code et comment les utiliser. Nous verrons enfin les limitations de ces tests.
L'objectif d'un test unitaire est de permettre au développeur de s'assurer qu'une unité de code ne comporte pas d'erreur de programmation. C'est un test, donc les vérifications sont faites en exécutant une petite partie (une « unité ») de code. En programmation orientée objet, l'unité est la classe. Un test est donc un programme qui exécute le code d'une classe pour s'assurer que celle-ci est correcte, c'est-à-dire que ses résultats correspondent à ce qui est attendu dans des assertions prédéfinies (ex : Addition(1,2) = 1 + 2 doit être vrai).

Concrètement, un test, c'est du code. À chaque classe d'une application, on associe une autre classe qui la teste.
Pour écrire des tests unitaires, vous avez à votre disposition des frameworks qui vont vous faciliter l'écriture des tests. Vous n'aurez plus qu'à écrire les classes de tests et c'est le framework qui se chargera de les trouver, de les lancer et de vous donner les résultats ou les erreurs qui ont été détectées. Ces frameworks portent le nom xUnit, en Java le framework le plus utilisé est Junit.

Etape 1 : Initiation à JUnit

1. Créer un projet java utilitaire qui contient les deux classes suivantes :
   a. Calculator.java possède trois fonctions :
      i. int sigma(int n) =1+2+…n,
      ii. long prod(int p,int q) =qx(q+1)x…p ,
      iii. long facto(int q)=q!
   b. Arithmetic.java qui possède les fonctions suivantes :
      i. boolean pair(int m)
      ii. boolean premier(int n)
      iii. int plusGrandDiviseur(int n)

2. Saisir ce premier exemple de test suivant :

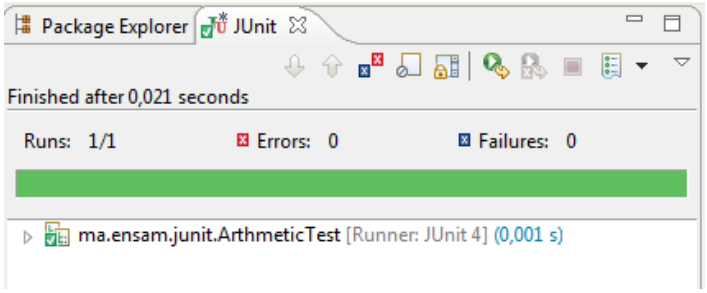
```
import static org.junit.Assert.assertEquals;
import org.junit.Test;

public class ArthmeticTest {

	int pgd=Arthmetic.plusGrandDiviseur(21);
	int testPgd=7;

@Test
	public void testPlusGrandDiviseur()
	{System.out.println("@Test fonction premier() : " + pgd + " = " + testPgd);
	assertEquals(pgd,testPgd);
	}
```

3. Exécuter la classe Test, en clic droit sur la classe ArthmeticTest et sélectionner Run As -> Junit Test, le programme affichera : ***@Test fonction plusGrandDiviseur() : 7 = 7***
   Eclipse fournit une fenêtre pour JUnit qui montre les résultats du test :



4. Changer la valeur testPdg=8 et exécuter de nouveau et visualiser le résultat.
5. Dans le code précédant, l'annotation **@Test** devant la méthode publique testPlusGrandDiviseur indique que cette méthode peut s'exécuter comme test case. La méthode ***assertEquals([String message] , object prévu, objet obtenu)*** prend deux objets comme paramètres d'entrée et vérifie qu'ils sont égaux.
6. Tableau des annotations de base prise en charge par Junit

Annotation	**Description**
@Test public void method()	L'annotation **TEST** indique que la méthode public auquel il est attaché peut être exécuté comme un cas de test.
@Before public void method()	L'annotation **BEFORE** indique que cette méthode doit être exécutée avant chaque test dans la classe, de façon à exécuter certaines conditions préalables nécessaires pour le test.
@BeforeClass public static void method()	L'annotation **BeforeClass** indique que la méthode statique à laquelle est attachée doit être exécutée une fois avant tous les tests de la classe. (par exemple se connecter à la base de données) ..
@After public void method()	L'annotation **After** indique que cette méthode est exécuté après l'exécution de chaque test (par exemple réinitialiser certaines variables après l'exécution de chaque test, supprimer des variables temporaires, etc.)
@AfterClass public static void method()	L'annotation **AfterClass** peut être utilisée lorsqu'une méthode doit être exécutée après l'exécution de tous les tests dans une classe.
@Ignore public static void method()	L'annotation **Ignorer** peut être utilisé lorsque vous souhaitez désactiver temporairement l'exécution d'un test spécifique. Chaque méthode qui est annotée avec @ignore ne sera pas exécuté.

Saisir et exécuter le code de l'exemple suivant

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

public class AnnotationsTest {

    private ArrayList testList;

    @BeforeClass
    public static void onceExecutedBeforeAll() {
        System.out.println("@BeforeClass: Exécuté une seule fois avant tout");
    }

    @Before
 public void executedBeforeEach() {
```

```
        testList = new ArrayList();
        System.out.println("@Before: Exécuté avant chaque test");
    }

    @AfterClass
    public static void onceExecutedAfterAll() {
        System.out.println("@AfterClass: Exécuté une seule fois après tout");
    }

    @After
    public void executedAfterEach() {
        testList.clear();
        System.out.println("@After: Exécuté après chaque test");
    }
    @Test
    public void EmptyCollection() {
        assertTrue(testList.isEmpty());
        System.out.println("@Test: EmptyArrayList");
    }
    @Test
    public void OneItemCollection() {
        testList.add("oneItem");
        assertEquals(1, testList.size());
        System.out.println("@Test: OneItemArrayList");
    }
    @Ignore
    public void executionIgnored() {
        System.out.println("@Ignore: Cette execution est ignorée");
    }
}
```

7. Appliquer les annotations précédentes dans le projet *utilitaire.*
8. Dans le tableau ci-dessous il y a une explication plus détaillée des méthodes d'assertion les plus couramment utilisés.

Assertion	**Description**
void assertEquals([String message], expected value, actual value)	Vérifier l'égalité de deux valeurs de type primitif ou objet (en utilisant la méthode equals()). Il existe de nombreuses surcharges de cette méthode pour chaque type primitif, pour un objet de type Object et pour un objet de type String
void assertTrue([String message], boolean condition)	Vérifier que la valeur fournie en paramètre est vraie
void assertFalse([String message],boolean condition)	Vérifier que la valeur fournie en paramètre est fausse
void assertNotNull([String message], java.lang.Object object)	Vérifier que l'objet fourni en paramètre ne soit pas null
void assertNull([String message], java.lang.Object object)	Vérifier que l'objet fourni en paramètre soit null
void assertSame([String message], java.lang.Object expected, java.lang.Object actual)	Vérifier que les deux objets fournis en paramètre font référence à la même entité Exemples identiques : assertSame("Les deux objets sont

	identiques", obj1, obj2); assertTrue("Les deux objets sont identiques ", obj1 == obj2);
void assertNotSame([String message], java.lang.Object unexpected, java.lang.Object actual)	Vérifier que les deux objets fournis en paramètre ne font pas référence à la même entité
void assertArrayEquals([String message], expectedArray, resultArray)	Vérifier que les deux tableaux sont égaux

Saisir et exécuter le code de l'exemple suivant :

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

public class AssertionsTest {

@Test
public void test() {
	String obj1 = "aiac";
	String obj2 = "aiac";
	String obj3 = "junit";
	String obj4 = "junit";
	String obj5 = null;
	int var1 = 1;
	int var2 = 2;
	int[] arithmetic1 = { 1, 2, 3 };
	int[] arithmetic2 = { 1, 2, 3 };

	assertEquals(obj1, obj2);
	assertSame(obj3, obj4);
	assertNotSame(obj2, obj4);
	assertNotNull(obj1);
	assertNull(obj5);
	assertTrue(var1<var2);
	assertArrayEquals(arithmetic1, arithmetic2);
}
    }
```

9. Rédiger des tests case pour les classes déjà réalise dans le projet utilitaire. Pour cela, clic droit sur la classe à tester → new → JUnit test case.

Etape 2 : Création de suite tests

1. Une suite tests est une collection de tests cases de plusieurs classes qui peuvent s'exécuter toutes ensemble en utilisation les annotations @RunWith et @Suite. Une suite de test est utile lorsqu'on souhaite exécuter un ensemble de test à la fois au lieu de test chaque test case à part. Par exemple :
   import org.junit.runner.RunWith;
   import org.junit.runners.Suite;
   @RunWith(Suite.class)
   @Suite.SuiteClasses({ Calculator.class, Arthmetic.class })
   public class SuitTest {

   }
   Pour réaliser une suite de tests → clic droite sur le package → other →Junit→Suite tests