digitalent
FORMATION

Tests Unitaires

Initiation à JUnit

xUnit

digitalent FORMATION

- Ensemble de systèmes de tests unitaires dérivés de JUnit
- Ensemble de principes s'appliquant pour différents langages :
  - JsUnit pour Javascript
  - PyUnit pour Python
  - CppUnit pour C++
  - PhpUnit pour Php
  - JUnit pour Java
  - …

Vocabulaire

digitalent FORMATION

- Unité : bloc de code à tester
- Assertion : vérification d'un résultat attendu. Si la vérification échoue, une exception est lancée et le test courant s'arrête
- Fixture : initialisation / terminaison commune à tous les tests unitaires
- Suite : un ensemble de tests unitaires exécutables

Unité

digitalent FORMATION

La classe

```
package fr.imie;

public class MaClasse {

    public String maPremiereMethode() {
        // une implémentation
        return "";
    }

    public void maSecondeMethode() {
        // une implémentation
    }

}
```

Unité

La méthode

✔ Recommandé

```
package fr.imie;

public class MaClasse {

    public String maPremiereMethode() {
        // une implémentation
        return "";
    }

    public void maSecondeMethode() {
        // une implémentation
    }

}
```

Classe de test unitaire

digitalent FORMATION

Classe à tester

```java
package fr.imie;

public class MaClasse {

    public String maPremiereMethode() {
        // une implémentation
        return "";
    }

    public void maSecondeMethode() {
        // une implémentation
    }

}
```

Classe de test unitaire

Convention de nommage de la classe (suffixe « Test »)

Import annotation @Test

```java
package fr.imie;

import org.junit.Test;

public class MaClasseTest {

    @Test
    public void testMaPremiereMethode() {
        // une implémentation
    }

    @Test
    public void testMaSecondeMethode() {
        // une implémentation
    }

}
```

Annotation « @Test » qui indique que la méthode est un test à exécuter

Assertions

digitalent FORMATION

Création d'une instance de la classe à tester

Exécution de la méthode à tester

Assertions qui vérifient les résultats d'exécution du code à tester

```java
package fr.imie;

import org.junit.Test;
import org.junit.Assert;

public class MaClasseTest {

    @Test
    public void testMaPremiereMethode() {

        MaClasse maClasse = new MaClasse();

        String valeurObtenue = maClasse.maPremiereMethode();

        Assert.assertEquals(10, valeurObtenue.length());
        Assert.assertEquals("valeur attendue", valeurObtenue);


    }

    @Test
    public void testMaSecondeMethode() {
        // une implémentation
    }


}
```

Assertions (test d'égalité)

digitalent FORMATION

- void assertEquals (Object expected, Object actual)
- void assertEquals (String message, Object expected, Object actual)
- void assertEquals (long expected, long actual)
- void assertEquals (String message, long expected, long actual)
- void assertEquals (String expected, String actual)
- void assertEquals (String message, String expected, String actual)
- void assertEquals (double expected, double actual)
- ...

Assertions (test de condition)

- void assertTrue (boolean condition)
- void assertTrue (String message, boolean condition)
- void assertFalse (boolean condition)
- void assertFalse (String message, boolean condition)

Assertions (test de nullité)

- void assertNull (Object object)
- void assertNull (String message, Object object)
- void assertNotNull (Object object)
- void assertNotNull (String message, Object object)

Assertions (test objets)

digitalent FORMATION

- void assertSame (Object expected, Object actual)
- void assertSame (String message, Object expected, Object actual)
- void assertNotSame (Object expected, Object actual)
- void assertNotSame (String message, Object expected, Object actual)

Assertions (autres)

digitalent FORMATION

- void assertArrayEquals (T[] expecteds, T[] actuals)
- void assertArrayEquals (String message, T[] expecteds, T[] actuals)
- void assertThat (T actual, Matcher<? super T> matcher)
- void assertThat(String reason, T actual, Matcher<? super T> matcher)
- void fail ()
- Void fail (String message)

Fixtures

digitalent FORMATION

@BeforeClass
Exécution de code avant l'instanciation de la classe de test

@Before
Exécution de code avant l'exécution de chaque test

@After
Exécution de code après l'exécution de chaque test

@AfterClass
Exécution après l'exécution de tous les tests

```
public class MaClasseTest {

    @BeforeClass
    public static void setUpBeforeClass() {
        // implémentation
    }

    @Before
    public void setUp() {
        // implémentation
    }

    @Test
    public void testMaPremiereMethode() {
        // une implémentation
    }

    @Test
    public void testMaSecondeMethode() {
        // une implémentation
    }


    @After
    public void tearDown() {
        // implémentation
    }

    @AfterClass
    public static void tearDownAfterClass() {
        // implémentation
    }
}
```

Suite

@RunWith
Déclaration d'une suite de test

@SuiteClasses
Déclaration de la liste de classes de tests à exécuter par la suite

```java
package fr.imie;

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

@RunWith(Suite.class)
@SuiteClasses(value = {MaClasseTest.class, MaClasseAutreTest.class})
public class MaSuiteTest {

}
```

Paramètres optionnels de @Test

digitalent FORMATION

Le résultat attendu du test est le lancement d'une exception. Si l'exception n'est pas lancée, le test échoue

Le test échoue si l'exécution prend plus de 1000 ms.

```
public class MaClasseTest {

    @Test (expected = NullPointerException.class)
    public void testMaPremiereMethode() {
        // une implémentation
    }

    @Test (timeout = 1000)
    public void testMaSecondeMethode() {
        // une implémentation
    }

}
```

@Ignore

Ignore un test lors de l'exécution

```
@Ignore
@Test
public void testMaPremiereMethode() {
    // une implémentation
}
```

A éviter
Tous les tests doivent toujours passer

Tests paramétrés (1)

```java
@RunWith(Parameterized.class)
public class MaClasseTest {

    @Parameters
    public static Collection<Object[]> datas() {
        List<Object[]> datas = new ArrayList<Object[]>();
        datas.add(new Object[]{12, 15});
        datas.add(new Object[]{100, 150});
        datas.add(new Object[]{129, 15});
        datas.add(new Object[]{128, 15});
        return datas;
    }

    private int entier1;
    private int entier2;

    public MaClasseTest(int int1, int int2) {
        entier1 = int1;
        entier2 = int2;
    }


    @Test
    public void testMaPremiereMethode() {
        // une implémentation
        System.out.println("testMaPremiereMethode" + entier1);
    }
```

Déclaration du runner « Parameterized » qui permet de paramétrer un test avec un jeu de données

Déclaration du jeu de données à l'aide de l'annotation @Parameters. Toutes les variantes seront testées.

Reprise de la structure des données en paramètres du constructeur. Les valeurs seront injectées par JUnit. N'importe quel type peut être utilisé.

Utilisation de la valeur injectée par JUnit dans le test

Tests paramétrés (2)

Utilisation de l'annotation @Parameter pour injecter directement une donnée.
Le champ value par défaut de l'annotation est 0 (premier paramètre)

Injection de la donnée se trouvant à l'indice 1

```java
@RunWith(Parameterized.class)
public class MaClasseTest {

    @Parameters
    public static Collection<Object[]> datas() {
        List<Object[]> datas = new ArrayList<Object[]>();
        datas.add(new Object[]{12, 15});
        datas.add(new Object[]{100, 150});
        datas.add(new Object[]{129, 15});
        datas.add(new Object[]{128, 15});
        return datas;
    }

    @Parameter public int entier1;
    @Parameter (value =1) public int entier2;


    @Test
    public void testMaPremiereMethode() {
        // une implémentation
        System.out.println("testMaPremiereMethode" + entier1);
    }
```

@Rule

- Déclarer une règle dans JUnit

```
@Rule public TypeRegle nom = valeur
```

@Rule – Exceptions levées

- Contrôler plus finement l'exception levée

```
public class MaClasseTest {

    @Rule public ExpectedException exceptionLevee = ExpectedException.none();

    @Test
    public void testMaPremiereMethode() {
        // une implémentation
        exceptionLevee.expect(ArrayIndexOutOfBoundsException.class);
        exceptionLevee.expectMessage("xxxxxxxx");
    }
```

Contrôle du message renvoyé par l'exception

@Rule – nom de la méthode de test

Configuration de la règle JUnit permettant de récupérer le nom d'une méthode de test

La méthode getMethodName() permet de récupérer le nom du test en cours d'exécution

```
public class MaClasseTest {

    @Rule public TestName nomMethode = new TestName();

    @Test
    public void testMaPremiereMethode() {
        // une implémentation
        System.out.println(nomMethode.getMethodName());
    }

    @Test (timeout = 1000)
    public void testMaSecondeMethode() {
        // une implémentation
        System.out.println(nomMethode.getMethodName());
    }
```

@Rule – Répertoire temporaire

Configuration de la règle JUnit permettant de manipuler un répertoire temporaire pour le test

Créer un fichier ou un répertoire temporaire

```
public class MaClasseTest {

    @Rule public TemporaryFolder repertoireTemporaire = new TemporaryFolder();

    @Test
    public void testMaPremiereMethode() throws IOException {
        // une implémentation
        File nouveauFichier = repertoireTemporaire.newFile();
        File nouveauRepertoire = repertoireTemporaire.newFolder();
        System.out.println(nouveauFichier.getPath());
        System.out.println(nouveauRepertoire.getPath());
    }
```

@Rule – D'autres règles

digitalent FORMATION

- ExternalResource :
  - Gestion d'un système externe.
- ErrorCollector
  - Permet à test de continuer à s'exécuter après une erreur, les erreurs sont alors collectées. Toutes les erreurs sont affichées à la fin du test.
- Il est possible de définir ses propres règles en héritant des classes :
  - Stopwatch : réagir à une violation de la durée d'exécution
  - TestWatcher : réagir après chaque test en fonction de son déroulement
  - Verifier : ajouter une vérification supplémentaire à la fin de chaque test

TDD

digitalent FORMATION