Rapport Projet JUnit5 Jupiter

Travail présenté à Olivier Richard

LO54

réalisé par

Adrien Bouyssou ;

Vincent Galinier ;

16 décembre 2020

Table des matières

Première partie

JUnit5 Jupiter

JUnit est le framework java permettant de faire des tests unitaires sur une application back-end. JUnit5 Jupiter est la dernière version en date de ce framework. Dans ce document nous allons voir comment l'utiliser dans un projet JEE Maven.

La documentation écrite ici et un projet d'exemple sont disponibles à l'adresse www.github.com/macdrien/jupiter-example.git

1 Requirement

Pour fonctionner, JUnit5 a besion d'une version Java8+.

2 Dependencies

JUnit est découpé en plusieurs artifacts qui ont chacun leurs usages. Dans tous les cas, le scope des dépendences peut être restreint aux tests. JUnit étant un framework de tests unitaires, il est inutile de le concerver dans les versions de production. Voici donc les trois artifacts proposés par JUnit Jupiter :

2.1 JUnit Jupiter Platform

JUnit Jupiter Platform est le module qui permet de lancer les tests unitaires. Il se charge depuis l'artifact junit-jupiter-engine.
Cet artifact fourni également l'API TestEngine. Cette API permet de créer ses propres frameworks de test. Ces frameworks pourront ensuite être utilisé grâce à l'artifact junit-jupiter-engine.

```
1 <dependency>
2     <groupId>org.junit.jupiter</groupId>
```

```
3       <artifactId>junit-jupiter-engine</artifactId>
4       <version>${junit.jupiter.engine.version}</version>
5       <scope>test</scope>
6   </dependency>
```

2.2 JUnit Jupiter API

JUnit Jupiter API est la dépendence qui apporte les outils de test (méthodes, annotations, ...).

```
1   <dependency>
2       <groupId>org.junit.jupiter</groupId>
3       <artifactId>junit-jupiter-api</artifactId>
4       <version>${junit.jupiter.api.version}</version>
5       <scope>test</scope>
6   </dependency>
```

2.3 JUnit Jupiter Vintage

Cette dépendance n'a pas été utilisée dans le projet d'entraînement. Cependant elle peut s'avérer très utile.

JUnit Jupiter Vintage est une dépendance de JUnit5 qui embarque les fonctionnalités des versions 3 et 4 de JUnit. Cette dépendance n'est donc pas souvent utile. Mais elle peut l'être dans le cadre d'une montée de version de JUnit dans un projet déjà testé.

```
1   <dependency>
2       <groupId>org.junit.vintage</groupId>
3       <artifactId>junit-vintage-engine</artifactId>
4       <version>${junit.vintage.engine.version}</version>
5       <scope>test</scope>
6   </dependency>
```

2.4 Build plugin

Surfire ne prend pas JUnit5 en charge par défaut. Pour que maven puisse lancer les tests JUnits, il faut donc ajouter un plugin au build maven.

Une fois ajouté, la commande 'mvn test' fera tourner les tests JUnit5 Jupiter.

```
<build>
    <plugins>
        <!-- ... -->
        <plugin>
            <artifactId>maven-surefire-plugin</artifactId>
            <version>${maven.surfire.plugin.version}</version>
            <dependencies>
                <dependency>
                    <groupId>org.junit.platform</groupId>
                    <artifactId>junit-platform-surefire-provider</artifactId>
                    <version>${junit.platform.version}</version>
                </dependency>
                <dependency>
                    <groupId>org.junit.jupiter</groupId>
                    <artifactId>junit-jupiter-engine</artifactId>
                    <version>${junit.jupiter.version}</version>
                </dependency>
            </dependencies>
        </plugin>
        <!-- ... -->
    </plugins>
</build>
```

2.5 Méthodes d'assertions

Les méthodes d'assertions sont, avec l'annotation Test, les principaux outils utilisés pour les tests. C'est pour cela que nous verrons ces méthodes en premier. Les méthodes d'assertions sont un ensemble de fonctions qui effectueront un test simple et retourneront un état validé ou échoué.
Si la méthode d'assertion échoue, alors elle va stopper la fonction de test en la marquant comme échouée.
Si elle réussie, la fonction de test va continuer. Si le processus de test arrive à la fin de la fonction de test sans lever d'erreur ou faire échouer une assertion, alors le test sera marqué comme réussi.

Il y a une grand nombre de méthodes d'assertions, aussi, nous ne verront que les plus communes. Toutes ces méthodes sont trouvable dans la clases org.junit.jupiter.api.Assertions. Pour des raisons pratiques, elles seront importées de manière statique via l'import suivant :

```
1 import static org.junit.jupiter.api.Assertions.*;
```

La plus part des méthodes qui vont être présentées peuvent prendre un dernier argument. Ce dernier contiendra un message à afficher en cas d'erreur du test. En l'absence de cet argument, un message par défaut sera généré.

Les méthodes que nous utilisons le plus souvent sont les suivantes :

2.5.1 assertEquals

Cette méthode demande deux arguments. Le premier est le résultat attendu, le second est la variable a tester.

```
1 assertEquals(expect, origin.add(second));
```

Cette méthode va appeler la méthode equals du premier argument en y donnant le second argument. Si l'objet donné n'implémente pas la méthode equals, alors la méthode utilisera l'opérateur == pour faire son test. Le test sera alors fait sur les références des deux objets.

2.5.2 assertTrue et assertFalse

Ces deux méthodes n'ont besoin que d'un seul argument. Cet argument est une condition ou une valeur booléenne à tester. L'assertion réussira si la condition booléenne est vraie pour la première méthode, ou fausse pour la seconde méthode.

```
1 assertTrue(5 + 3 == 8);   // Success
2 assertFalse(5 + 5 == 8); // Success
3 assertTrue(5 + 5 == 8);   // Fail
4 assertFalse(5 + 3 == 8); // Fail
```

2.5.3 assertNull et assertNotNull

AssertNull et assertNotNull demandent également un seul argument qui est la variable/-méthode à tester. La première réussira si l'argument est null, la seconde si le l'argument est non-null.

```
1 assertNull(null);      // Success
2 assertNotNull(5);      // Success
3 assertNull(5);         // Fail
4 assertNotNull(null); // Fail
```

2.5.4 assertThrows

AssertThrows, comme assertEquals, a besion de deux arguments. Le premier est l'exception qui est attendue. Le second argument est une lambda expression qui doit être testée.

```
1 assertThrows(IllegalArgumentException.class,
2                 () -> origin.add(null));
```

Cette assertion réussira si la lambda expression lève l'exception passée. Elle échoura si aucune exception n'est levée ou si l'exception levée n'est pas celle attendue.

2.6 Annotations

2.6.1 Test

Cette annotation est la plus commune. Placée sur une fonction, elle marque cette dernière comme étant une fonction de test. Comme les tests sont lancés, JUnit5 recherche toutes les méthodes marquées de @Test pour les lancer.

Note : Une méthode de test ne retourne rien. Les méthodes d'assertions se chargeront de relever les succès/erreurs au besoin.

```
1 @Test
2 void testAddWithNullArgument() {
3     assertThrows(IllegalArgumentException.class,
4                  () -> origin.add(null));
5 }
```

2.6.2 BeforeAll

BeforeAll doit être passée sur une méthode. Elle la marque comme étant à lancer une unique fois avant toutes les autres fonctions de la même classe pendant la phase de test. Cette annotation est notamment utile pour écrire une méthode préparant un contexte global à la classe courante.

Note : Ces méthodes doivent être statique et ne retournent rien.

```
1 @BeforeAll
2 static void init() {
3     origin = new Number();
4     second = new Number();
5     expect = new Number();
6 }
```

2.6.3 BeforeEach

Comme BeforeAll, cette annotation doit être passée sur une méthode. Cette annotation va marquer la méthode comme étant à lancer avant chaque nouveau test de la classe courrante. BeforeEach peut être utile pour préparer un contexte spécifique au test.

Note : Ces méthodes doivent être statique et ne retournent rien.

```
1 @BeforeAll
2 static void stup() {
3     origin.setNumber(6);
4     second.setNumber(2);
5 }
```

2.6.4 AfterAll

L'annotation afterAll marque une méthode comme étant à lancer après tous les tests de la classe courrante. Elle est très utile pour nettoyer l'environnement ou la base de test.

Note : Ces méthodes doivent être statique et ne retournent rien.

```
1 @AfterAll
2 static void shutdown() {
3     if (file != null)
4         file.close();
5 }
```

2.6.5 AfterEach

L'annotation afterEach est l'opposée de beforeEach. La méthode marquée par cette annotation sera alors lancée après chaque test de la classe courante.

```
1 @AfterEach
2 static void clean() {
3     created.delete();
4 }
```