Spring Boot 2, Mockito and JUnit 5

1. Dépendances Maven

La `spring-boot-starter-test` inclut les dépendances JUnit 5 et Mockito. Nous n'avons donc qu'à inclure cette dépendance .

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```

2. Configuration

Dans cet exemple, nous testons principalement deux classes `EmployeeManager` et `EmployeeDao`. Comme son nom l'indique, la classe de gestionnaire représente la couche de service et la classe dao interagit avec la base de données.

`EmployeeManager` La classe a une dépendance sur `EmployeeDao` et délègue les appels de méthode pour obtenir les données qui sont finalement renvoyées aux classes de contrôleur.

Pour tester les méthodes dans `EmployeeManager`, nous pouvons créer une classe de test JUnit TestEmployeeManager ci-dessous de deux manières :

2.1. MockitoJUnitRunner

La classe MockitoJUnitRunner initialise automatiquement tous les objets annotés avec `@Mock` et `@InjectMocks` annotations.

```
@RunWith(MockitoJUnitRunner.class)
public class TestEmployeeManager {

	@InjectMocks
	EmployeeManager manager;

	@Mock
	EmployeeDao dao;

	//tests
}
```

2.2. Méthode MockitoAnnotations.openMocks()

Si nous n'utilisons pas l'approche de classe MockitoJUnitRunner, nous pouvons utiliser la méthode statique `MockitoAnnotations.initMocks()`. Cette méthode, lors de l'initialisation des tests junit, initialise également les objets fictifs.

```
public class TestEmployeeManager {

	@InjectMocks
	EmployeeManager manager;

	@Mock
	EmployeeDao dao;

	@Before
	public void init() {
		MockitoAnnotations.openMocks(this);
	}

	//tests
}
```

2.3. @Mock contre @InjectMocks

- L' annotation @Mock crée une implémentation fictive pour la classe avec laquelle elle est annotée.
- @InjectMocks crée également l'implémentation fictive, injecte en outre les simulacres dépendants qui y sont marqués avec les annotations `@Mock`.

Dans l'exemple ci-dessus, nous avons annoté `EmployeeManager` la classe avec `@InjectMocks`, donc mockito créera l'objet fictif pour la `EmployeeManager`classe et y injectera la dépendance fictive de `EmployeeDao`.

3. Tests JUnit avec Mockito

Voyons quelques exemples d'écriture des tests junit pour *tester unitairement les méthodes de la couche de service* à l' aide d'objets fictifs créés avec mockito.

Quelques exemples de méthodes pourraient être pour *getAllEmployees()* renvoyant une liste d' `EmployeeVO`objets, *`getEmployeeById(int id)`*pour renvoyer un employé avec un identifiant donné ; et `createEmployee()` pour ajouter un objet employé et retourner `void`.