

Préambule : Avez-vous des questions sur les enseignements dispensés précédemment ?


Objectif :

- Mettre en place des doublures de test
- Se familiariser avec Mockito
- Améliorer la couverture de test

Lors de la première séance de TP, nous avons couvert de tests unitaires le coeur de l'application : l'algorithme qui détermine entre 2 Pokémons lequel remporte la bagarre. Nous allons maintenant ajouter des tests unitaires un cran plus haut : au niveau de la classe *Bagarre*. Cette classe est le point d'entrée fonctionnel, c'est elle qui orchestre la bagarre en validant les noms de Pokémons saisis, en récupérant leurs statistiques via l'API web, puis en déterminant le vainqueur.

La classe *Bagarre* a un seul collaborateur : *PokeBuildApi*. Afin de mettre en place des tests unitaires solitaires, nous allons depuis les tests fournir une fausse implémentation de *PokeBuildApi* : une doublure.

Travail à faire :

1. Lire la classe *Bagarre.java*. Combien faudra-t-il de cas de tests pour vérifier son comportement ?
Lesquels ?
2. Dans le fichier *build.gradle*, ajouter une dépendance sur Mockito dans la section *dependencies* :
`testImplementation 'org.mockito:mockito-core:5.2.0'`
3. Rafraichir le projet dans la vue Gradle dans IntelliJ 
4. Créer un fichier de test *BagarreTest.java* (*Ctrl+Maj+T* depuis la classe *Bagarre*)
5. Ecrire les tests imaginés à la question 1 et s'assurer qu'ils passent

Quelques morceaux de code qui pourront vous être utiles:

- Pour attraper une erreur et faire des assertions dessus :
`Throwable thrown = catchThrowable(() -> codeQuiJetteUneException());`

```
assertThat(throwN)
```

```
.assertInstanceOf(TypeException.class)
```

```
.hasMessage("boom");
```

- Pour créer une doublure de la classe *PokeBuildApi* dans les tests :

```
var fausseApi = Mockito.mock(PokeBuildApi.class) ;
```

- Pour préparer cette doublure à retourner un Pokémon :

```
Mockito.when(fausseApi.recupererParNom("pikachu"))
```

```
.thenReturn(CompletableFuture.completedFuture(new Pokemon("pikachu",  
"url1", new Stats(1, 2))));
```

- Pour simuler un échec lors de la récupération du Pokémon:

```
Mockito.when(fausseApi.recupererParNom("pikachu"))
```

```
.thenReturn(CompletableFuture.failedFuture(new  
ErreurRecuperationPokemon("pikachu")));
```

- Quelques assertions sur les futures :

```
assertThat(futurVainqueur)
```

```
.succeedsWithin(Duration.ofSeconds(2))
```

```
.satisfies(pokemon -> {
```

```
    assertThat(pokemon.getNom()).isEqualTo("Pikachu");
```

```
    // autres assertions...
```

```
});
```

```
assertThat(futurVainqueur)
```

```
.failsWithin(Duration.ofSeconds(2))
```

```
.withThrowableOfType(ExecutionException.class)
```

```
.havingCause()
```

```
.assertInstanceOf(ErreurRecuperationPokemon.class)
```

```
.withMessage("Blabla") ;
```

Voir aussi le chapitre 3 du cours « Ecosystème du test en Java » pour d'autres morceaux de code utiles