

3 - ECOSYSTÈME DU TEST EN JAVA



R4.02 - Qualité de développement

Institut Universitaire de Technologie de Bayonne et du Pays Basque

Pantxika Dagorret - Damien Urruty

BUT Informatique 2022 / 2023 - Semestre 4

JUnit

NOUS ALLONS UTILISER CE PROJET COMME EXEMPLE

<https://github.com/la-urre/code-smells>

JUNIT

- Le principal framework de test en Java
- Contient: un moteur d'exécution de tests, une API pour contribuer des tests, une API d'assertion, ...
- Dernière version majeure = v5
- Sert à automatiser des tests, peu importe leur granularité (unitaire, intégration, ...)

RAPPEL : STRUCTURE D'UN TEST

- Etant donné (*Given/Arrange*) un état initial
- Lorsque (*When/Act*) on réalise quelque chose
- Alors (*Then/Assert*) on vérifie l'état final

CAS DE TEST JUNIT (*TEST CASE*)

L'élément de base est l'annotation @Test sur une méthode qui ne retourne rien (void)

```
@Test
void un_cas_de_test() {
    // GIVEN
    ...

    // WHEN
    ...

    // THEN
    ...
}
```

MÉTHODE D'ASSERTIONS (JUNIT)

```
import org.junit.jupiter.api.Assertions;  
  
Assertions.assertEquals(expectedValue, actualValue);  
Assertions.assertArrayEquals(expectedArray, actualArray);
```

EXEMPLE DE CAS DE TEST

```
@Test
void condenser_devrait_remplacer_les_milliers_par_un_K() {
    // GIVEN
    int nombre = 35613;

    // WHEN
    String resultat = Utilitaires.condenser(nombre)

    // THEN
    assertEquals("35,6K", resultat);
}
```


CAS DE TEST (*TEST CASE*)

- Privilégier le snake_case
- Espacer les Given / When / Then par une ligne vide
- Un cas de test doit être assez court (max 10-15 lignes)
- Le nom décrit le comportement vérifié
- Utiliser un import statique pour les méthodes assert

SUITE DE TEST (*TEST SUITE*)

- Permet de grouper les tests
- En général une suite de test par classe
- Convention de nommage: si on teste MaClasse.java
=> MaClasseTest.java

MONTAGE / DÉMONTAGE (*SET UP / TEAR DOWN*)

MÉTHODES EXÉCUTÉES AVANT / APRÈS LA SUITE DE TEST

```
@BeforeAll
static void suiteSetUp() { // nom pas imposé
    // initialisations
}

@AfterAll
static void suiteTearDown() { // nom pas imposé
    // nettoyages
}
```

MONTAGE / DÉMONTAGE (*SET UP / TEAR DOWN*)

MÉTHODES EXÉCUTÉES AVANT / APRÈS CHAQUE CAS DE TEST

```
@BeforeEach
void setUp() {
    // initialisations
}

@AfterEach
void tearDown() {
    // nettoyages
}
```

EXEMPLE D'EXÉCUTION

- appel méthode(s) annotée(s) @BeforeAll
- -----
- appel méthode(s) annotée(s) @BeforeEach
- appel 1ère méthode annotée @Test
- appel méthode(s) annotée(s) @AfterEach
- -----
- appel méthode(s) annotée(s) @BeforeEach
- appel 2ème méthode annotée @Test
- appel méthode(s) annotée(s) @AfterEach
- -----
- appel méthode(s) annotée(s) @AfterAll

PROBLÈMES AVEC L'API D'ASSERTIONS DE JUNIT

- Facile d'intervertir "actual" et "expected" (important pour le message d'échec)
- Comparaisons limitées (exemple: objets, collections, exceptions)

ASSERTJ



JUnit
assertions



AssertJ

ASSERTJ

- API "fluent" d'assertions: se lit comme un texte
- Assertions composables
- "Standard" en complément de JUnit

ASSERTJ - EXEMPLES

```
assertThat(actualValue).isEqualTo(expected);  
assertThat(actualList).contains(value);
```



JACOCO














- Outil qui fournit un rapport de la couverture de code d'un projet
- Permet de visualiser le code ouvert et non-couvert
- S'utilise facilement avec Gradle

JACOCO & GRADLE

```
plugins {  
    id 'jacoco'  
}  
  
jacocoTestReport {  
    dependsOn test  
}
```

RAPPORTS JACOCCO

PokeBagarre

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines
 com.montaury.pokebagarre		0%		n/a	2	2	3	3
 com.montaury.pokebagarre.webapi		93%		75%	1	8	2	17
 com.montaury.pokebagarre.ui		100%		100%	0	11	0	61
 com.montaury.pokebagarre.metier		100%		100%	0	23	0	35
 com.montaury.pokebagarre.erreurs		100%		n/a	0	4	0	8
Total	14 of 510	97%	1 of 24	95%	3	48	5	124

Created with [JaCoCo](#) 0.8.8.20

RAPPORTS JACOCO

```
17. public class PokeBuildApi {
18.     private static final String URL_BASE_API_PRODUCTION = "https://pokebuildapi.fr/";
19.     private final String urlBaseApi;
20.
21.     public PokeBuildApi() {
22.         this(URL_BASE_API_PRODUCTION);
23.     }
24.
25.     public PokeBuildApi(String urlBaseApi) {
26.         this.urlBaseApi = urlBaseApi;
27.     }
28.
29.     public CompletableFuture<Pokemon> recupererParNom(String nom) throws ErreurBagarre {
30.         return HttpClient.newHttpClient().sendAsync(HttpRequest.newBuilder()
31.             .uri(getWebApiUri(nom.trim()))
32.             .build(), HttpResponse.BodyHandlers.ofString())
33.             .thenApply(reponseHttp -> essayerDeConvertirReponse(nom, responseHttp));
34.     }
35.
36.     private URI getWebApiUri(String nom) {
37.         try {
38.             return new URI(urlBaseApi + "api/v1/pokemon/" + URLEncoder.encode(nom.toLowerCase(), StandardCharsets.UTF_8));
39.         } catch (URISyntaxException e) {
40.             throw new ErreurRecuperationPokemon(nom);
41.         }
42.     }
43. }
```



MOCKITO

- Librairie qui facilite la création de doublures de test
- La plus connue et utilisée dans le monde Java
- Utilisé dans les tests pour créer de faux collaborateurs

MOCKITO - CRÉATION D'UN MOCK

```
var fauxPrinter = Mockito.mock(USBPrinter.class);
```

MOCKITO - VÉRIFICATION

```
var fauxPrinter = Mockito.mock(USBPrinter.class);  
// utilisation du fauxPrinter par une autre classe  
Mockito.verify(fauxPrinter).print("ligne");
```

MOCKITO - CONFIGURATION

```
var fauxPrinter = Mockito.mock(USBPrinter.class);  
Mockito.when(fauxPrinter.print("ligne")).thenReturn(true);  
  
// utilisation du fauxPrinter par une autre classe
```



AWAITILITY

- Librairie qui aide à tester des traitements asynchrones
- Se combine bien avec AssertJ

AWAITILITY - ATTENTE ACTIVE

```
await().atMost(5, SECONDS).untilAsserted(() ->  
    assertThat(printer.lines()).contains("Hello, world!")  
);
```