TESTS UNITAIRE ET TDD

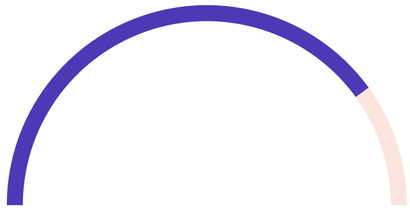
BERNIER Allan - 2023

Introduction aux tests unitaires	Définition des tests unitaires Pourquoi les tests unitaires sont importants? Types de tests unitaires Exemples de frameworks de tests unitaires
JUnit 5	Présentation de JUnit 5 Les annotations JUnit 5 Assertion API Gestion des exceptions
TDD (Test-Driven Development)	Qu'est-ce que TDD? Les avantages et les inconvénients du TDD
Pratique des tests unitaires et de TDD avec JUnit 5	Exemples de tests unitaires avec JUnit 5 Comment écrire des tests avec TDD? Implémentation de code avec TDD

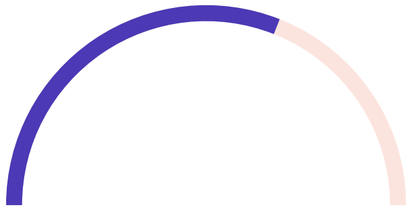
m2i Formation

Introduction aux tests unitaires

Les types de tests



Les tests de validation



Les tests d'erreur

Les tests de performance

Les différents frameworks

JUnit

Java

NUnit / XUnit

.NET

PHPUnit / Pest

Php

PyUnit

Php

JUnit 5

Formation

Présentation de JUnit 5

assertEquals

```
@Test
void trueAssumption() {
    assumeTrue(5 > 1);
    assertEquals(5 + 2, 7);
}

@Test
void falseAssumption() {
    assumeFalse(5 < 1);
    assertEquals(5 + 2, 7);
}
```

Présentation de JUnit 5

assertThrows

```java
@Test
void assertThrowsException() {
    String str = null;
    assertThrows(IllegalArgumentException.class, () -> {
      Integer.valueOf(str);
    });
}
```

Présentation de JUnit 5

assertAll

```
@Test
void groupAssertions() {
    int[] numbers = {0, 1, 2, 3, 4};
    assertAll("numbers",
        () -> assertEquals(numbers[0], 1),
        () -> assertEquals(numbers[3], 3),
        () -> assertEquals(numbers[4], 1)
    );
}
```

Les annotations JUnit 5

@BeforeAll, @BeforeEach, @AfterAll, @AfterEach

```java
@BeforeAll
static void setup() {
    log.info("@BeforeAll - executes once before all test methods in this class");
}

@BeforeEach
void init() {
    log.info("@BeforeEach - executes before each test method in this class");
}
```

```java
@AfterEach
void tearDown() {
    log.info("@AfterEach - executed after each test method.");
}

@AfterAll
static void done() {
    log.info("@AfterAll - executed after all test methods.");
}
```

Les annotations JUnit 5

@DisplayName, @Disabled

```
@DisplayName("Single test successful")
@Test
void testSingleSuccessTest() {
    log.info("Success");
}

@Test
@Disabled("Not implemented yet")
void testShowSomething() {
}
```

Les paramètres de tests avec JUnit 5

@ParameterizedTest, @ValueSource

```
@ParameterizedTest
@ValueSource(ints = {1, 3, 5, -3, 15, Integer.MAX_VALUE}) // six numbers
void isOdd_ShouldReturnTrueForOddNumbers(int number) {
    assertTrue(Numbers.isOdd(number));
}
```

A votre tour

Installations :

Java, Eclipse, Spring, Junit5

https://www.oracle.com/java/technologies/javase/jdk17-archive-downloads.html

Java

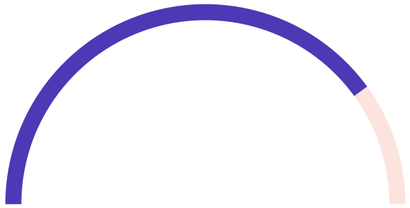
https://www.eclipse.org/downloads/

Eclipse

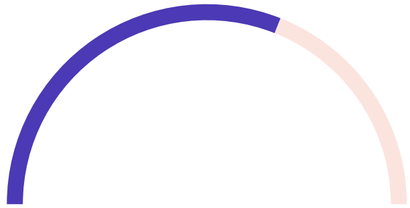
https://start.spring.io/

Spring Initializr

Installations



INSTALLER JAVA



INSTALLER ECLIPSE

CRÉER UN PROJET SPRING

Exercice 1

LA CLASSE CALCULATOR DOIT IMPLÉMENTER LES MÉTHODES SUIVANTES :

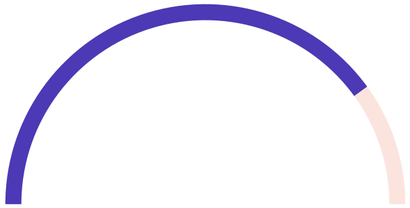
- add(int a, int b): qui retourne la somme de a et b
- subtract(int a, int b): qui retourne la différence entre a et b
- multiply(int a, int b): qui retourne le produit de a et b
- divide(int a, int b): qui retourne le quotient de a et b

Ecrire des tests unitaires pour chaque méthode de la classe Calculator en utilisant JUnit. Testez chaque méthode avec des valeurs d'entrée différentes et assurez-vous que les résultats renvoyés sont corrects.

TDD (Test-Driven Development)

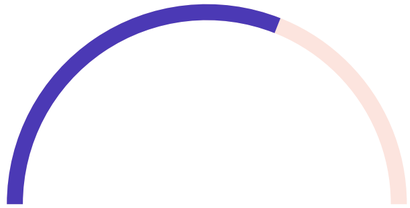
Formation

TDD (Test-Driven Development)



Qu'est-ce que le TDD

Les défis du TDD



Les étapes du TDD

Les meilleures pratiques pour le TDD

Les avantages du TDD

Les outils pour le TDD

Pratique des tests unitaires et de TDD avec JUnit 5

Exercice 2

Objectif

Écrire une fonction qui prend en entrée une liste de nombres entiers et renvoie une nouvelle liste contenant tous les nombres pairs de la liste d'entrée.

Instructions :

1. Écrivez une suite de tests pour vérifier que la fonction renvoie correctement la liste des nombres pairs.
2. Écrivez la fonction pour qu'elle passe tous les tests.
3. Réexécutez les tests pour vérifier que la fonction fonctionne correctement.
4. Optimisez votre code pour qu'il soit plus efficace si possible, tout en maintenant la couverture de test.

Exercice 2

Objectif

Écrire une fonction qui prend en entrée une chaîne de caractères contenant des nombres et des opérations arithmétiques (+, -, *, /) et renvoie le résultat de l'opération. L'entrée est toujours de la forme digit opérateur digit ...

Instructions :

1. Écrivez une suite de tests pour vérifier que la fonction renvoie correctement le résultat de l'opération.
2. Écrivez la fonction pour qu'elle passe tous les tests.
3. Réexécutez les tests pour vérifier que la fonction fonctionne correctement.
4. Optimisez votre code pour qu'il soit plus efficace si possible, tout en maintenant la couverture de test.

Evaluation

Programmer un jeu de pierre feuille ciseaux

routes :
GET /game/play/{action} -> Renvoie "Vous avez jouer {action}, l'ordinateur à jouer {action ordinateur}, vous avez {gagné|perdu| fait égalité}
POST /game/restart -> Remet le score à 0
GET /game/score -> Renvoie le nombre de victoire, égalité et défaites
PUT /game/score/{win}/{lose}/{tie} -> Modifie votre score

Vous devez tester et coder l'entièreté de votre code en TDD et obtenir le meilleur code coverage possible.