TD2 - AssertJ

📌 **Objectif :** Découvrir et utiliser AssertJ pour écrire des tests unitaires expressifs et lisibles en Java.

1. Installation et configuration

Ajoutez AssertJ à votre projet Maven :

```
<dependency>
  <groupId>org.assertj</groupId>
  <artifactId>assertj-core</artifactId>
  <version>3.24.2</version>
  <scope>test</scope>
</dependency>
```

✅ **Vérifiez l'installation en exécutant ce test simple :**

```
import static org.assertj.core.api.Assertions.*;

import org.junit.jupiter.api.Test;

class AssertJTest {

  @Test
  void testInstallation() {
    String message = "AssertJ fonctionne!";
    assertThat(message).isNotNull()
            .isNotEmpty()
            .isEqualTo("AssertJ fonctionne!");
  }
}
```

2. Exercices pratiques

Exercice 1 : Tester des chaînes de caractères

Créez la méthode suivante :

```
public class StringUtils {
   public static String toUpperCase(String input) {
      return (input == null) ? null : input.toUpperCase();
   }
}
```

🔹 **Test à écrire avec AssertJ :**

1. Vérifiez que toUpperCase("java") retourne "JAVA".
2. Vérifiez que toUpperCase(null) retourne null.
3. Vérifiez que la chaîne retournée commence par "J" et a une longueur de 4.

Exercice 2 : Tester des collections

Ajoutez cette méthode :

```
public class ListUtils {
   public static List<String> filterNames(List<String> names) {
      return names.stream().filter(name -> name.startsWith("A")).toList();
   }
}
```

✅ **Testez avec AssertJ :**

1. Vérifiez que filterNames(List.of("Alice", "Bob", "Anna")) contient "Alice" et "Anna".
2. Vérifiez que "Bob" n'est **pas** dans la liste retournée.
3. Vérifiez que la liste a exactement **2 éléments**.

Exercice 3 : Tester des objets

Créez la classe Person :

```
public class Person {
   private final String name;
   private final int age;

   public Person(String name, int age) {
      this.name = name;
      this.age = age;
   }

   public String getName() { return name; }
   public int getAge() { return age; }
}
```

✅ **Testez avec AssertJ :**

1. Vérifiez que new Person("Alice", 30) a name = "Alice" et age = 30.
2. Utilisez usingRecursiveComparison() pour comparer deux objets Person identiques.

Exercice 4 : Tester des exceptions

Ajoutez cette méthode :

```
public class MathUtils {
   public static int divide(int a, int b) {
      if (b == 0) throw new IllegalArgumentException("Division par zéro !");
      return a / b;
   }
}
```

✅ **Testez avec AssertJ :**

1. Vérifiez que divide(4, 2) retourne 2.
2. Vérifiez que divide(4, 0) lève une IllegalArgumentException.

Exercice 5 : Tester des dates

Créez une méthode qui retourne la date du jour :

```
public class DateUtils {
   public static LocalDate getToday() {
      return LocalDate.now();
   }
}
```

✅ **Testez avec AssertJ :**

1. Vérifiez que getToday() est bien la date actuelle.
2. Vérifiez qu'elle est **après** 2000-01-01 et **avant** 2100-01-01.

A rendre le jeudi 20/02/2025 avant 00h

Envoyer lien github : bassiroungom26@gmail.com