Junit 5

Exercises

@RepeatedTest

- Create a method named generateRandomNumber in a class called RandomNumberGenerator.
- The method should return a random integer between 1 and 10.
- Use the @RepeatedTest annotation to run the test 20 times.
- Ensure that the generated number is always between 1 and 10.

@RepeatedTest - Solution

```java
import java.util.Random;

public class RandomNumberGenerator {

    private Random random = new Random();

    public int generateRandomNumber() {
        // Generates a random number between 1 and 10
        return random.nextInt(10) + 1;
    }
}
```

....

```java
public class RandomNumberGeneratorTest {

private RandomNumberGenerator randomNumberGenerator = ne

    @RepeatedTest(20)
    void testGenerateRandomNumber() {

        int randomNumber = randomNumberGenerator.generateRandomNumber();

         assertTrue(randomNumber >= 1 && randomNumber <= 10,
            "The generated number should be between 1 and 10, but was: " + randomNumber);

     }
}
```

@TestMethodOrder

- Create a class named Calculator with methods for basic arithmetic operations: add, subtract, multiply, and divide.
- Create a test class named CalculatorTest.
- Use the @TestMethodOrder annotation to specify the order of test execution.
- Write tests for the Calculator methods and ensure they are executed in a specific order.

@TestMethodOrder - Solution

```java
public class Calculator {

    private double result;

    public Calculator() {
        this.result = 0;
    }

    public double add(double a, double b) {
        result = a + b;
        return result;
    }

    public double subtract(double a, double b) {
        result = a - b;
        return result;
    }

    public double multiply(double a, double b) {
        result = a * b;
        return result;
    }

    public double divide(double a, double b) {
        if (b == 0) {
            throw new IllegalArgumentException("Cannot divide by zero");
        }
        result = a / b;
        return result;
    }

    public double getResult() {
        return result;
    }
}
```

@TestMethodOrder - Solution

```
public class CalculatorTest {

    private Calculator calculator = new Calculator();

    @Test
    @Order(1)
    void testAdd() {
        double result = calculator.add(2, 3);
        assertEquals(5, result, "2 + 3 should equal 5");
    }

    @Test
    @Order(2)
    void testSubtract() {
        double result = calculator.subtract(5, 2);
        assertEquals(3, result, "5 - 2 should equal 3");
    }

    @Test
    @Order(3)
    void testMultiply() {
        double result = calculator.multiply(3, 4);
        assertEquals(12, result, "3 * 4 should equal 12");
    }
```

@TestMethodOrder - Solution

```
    @Test
    @Order(4)
    void testDivide() {
        double result = calculator.divide(10, 2);
        assertEquals(5, result, "10 / 2 should equal 5");
    }

    @Test
    @Order(5)
    void testDivideByZero() {
        assertThrows(IllegalArgumentException.class, () -> calculator.divide(10, 0), "Divide by zero should throw IllegalArgumentException");
    }
}
```

@TestMethodOrder

- Create a class named StringUtils with methods reverse, isPalindrome, and capitalize.
- Create a test class named StringUtilsTest.
- Use the @DisplayName annotation to give each test a meaningful name.

@TestMethodOrder - Solution

```java
public class StringUtils {

    public String reverse(String input) {
        if (input == null) {
            return null;
        }
        return new StringBuilder(input).reverse().toString();
    }

    public boolean isPalindrome(String input) {
        if (input == null) {
            return false;
        }
        String reversed = reverse(input);
        return input.equalsIgnoreCase(reversed);
    }

    public String capitalize(String input) {
        if (input == null || input.isEmpty()) {
            return input;
        }
        return input.substring(0, 1).toUpperCase() + input.substring(1).toLowerCase();
    }
}
```

@TestMethodOrder - Solution

```
public class StringUtilsTest {

    private StringUtils stringUtils = new StringUtils();

    @Test
    @DisplayName("Reversing a non-null string should return the string reversed")
    void testReverse() {
        assertEquals("dcba", stringUtils.reverse("abcd"), "The reversed string of 'abcd' should be 'dcba'");
    }

    @Test
    @DisplayName("Reversing a null string should return null")
    void testReverseNull() {
        assertEquals(null, stringUtils.reverse(null), "Reversing a null string should return null");
    }

    @Test
    @DisplayName("Checking if 'racecar' is a palindrome should return true")
    void testIsPalindromeTrue() {
        assertTrue(stringUtils.isPalindrome("racecar"), "'racecar' should be a palindrome");
    }
}
```

@Tag

- Create a class named Calculator with methods for basic arithmetic operations: add, subtract, multiply, and divide.
- Create a test class named CalculatorTest.
- Use the @Tag annotation to categorize tests with single and multiple tags.
- Use multiple @Tag annotation on each test
- Execute the tests, using some of the tags

@Tag - Solution

```
public class CalculatorTest {

    private Calculator calculator = new Calculator();

    @Test
    @Tag("fast")
    @Tag("arithmetic")
    void testAdd() {
        assertEquals(5, calculator.add(2, 3), "2 + 3 should equal 5");
    }

    @Test
    @Tag("fast")
    @Tag("arithmetic")
    void testSubtract() {
        assertEquals(3, calculator.subtract(5, 2), "5 - 2 should equal 3");
    }
}
```

@Tag - Solution

```
public class CalculatorTest {

    @Test
    @Tag("slow")
    @Tag("arithmetic")
    void testMultiply() {
        assertEquals(12, calculator.multiply(3, 4), "3 * 4 should equal 12");
    }

    @Test
    @Tag("slow")
    @Tag("arithmetic")
    void testDivide() {
        assertEquals(5, calculator.divide(10, 2), "10 / 2 should equal 5");
    }

    @Test
    @Tag("fast")
    @Tag("exception")
    void testDivideByZero() {
        assertThrows(IllegalArgumentException.class, () -> calculator.divide(10, 0), "Divide by zero should throw IllegalArgumentException");
    }
}
```

@Timeout

- Create a class named DataProcessor with methods processFastTask and processSlowTask.
- Create a test class named DataProcessorTest.
- Use the @Timeout annotation to specify time limits for the tests.

@Timeout - Solution

```
public class DataProcessor {

    public void processFastTask() {
        // Simulate a fast task (e.g., sleeping for 100 milliseconds)
        try {
            Thread.sleep(100);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }

    public void processSlowTask() {
        // Simulate a slow task (e.g., sleeping for 2000 milliseconds)
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }
}
```

@Timeout - Solution

```java
public class DataProcessorTest {

    private DataProcessor dataProcessor = new DataProcessor();

    @Test
    @Timeout(value = 500, unit = TimeUnit.MILLISECONDS)
    void testProcessFastTask() {
        dataProcessor.processFastTask();
    }

    @Test
    @Timeout(value = 3, unit = TimeUnit.SECONDS)
    void testProcessSlowTask() {
        dataProcessor.processSlowTask();
    }

    @Test
    @Timeout(value = 1, unit = TimeUnit.SECONDS)
    void testProcessFastTaskWithFail() {
        dataProcessor.processSlowTask(); // This should fail due to timeout
    }
}
```

@Disabled

- Create a class named MathUtils with methods add, subtract, multiply, and divide.
- Create a test class named MathUtilsTest.
- Use the @Disabled annotation to disable specific tests.

@Disabled - Solution

```java
public class MathUtils {

    public int add(int a, int b) {
        return a + b;
    }

    public int subtract(int a, int b) {
        return a - b;
    }

    public int multiply(int a, int b) {
        return a * b;
    }

    public int divide(int a, int b) {
        if (b == 0) {
            throw new IllegalArgumentException("Cannot divide by zero");
        }
        return a / b;
    }
}
```

@Disabled - Solution

```
public class MathUtilsTest {

    private MathUtils mathUtils = new MathUtils();

    @Test
    void testAdd() {
        assertEquals(5, mathUtils.add(2, 3), "2 + 3 should equal 5");
    }

    @Test
    @Disabled("Subtraction method is under development")
    void testSubtract() {
        assertEquals(3, mathUtils.subtract(5, 2), "5 - 2 should equal 3");
    }

    @Test
    void testMultiply() {
        assertEquals(12, mathUtils.multiply(3, 4), "3 * 4 should equal 12");
    }
}
```

@Disabled - Solution

```java
public class MathUtilsTest {


    @Test
    void testMultiply() {
        assertEquals(12, mathUtils.multiply(3, 4), "3 * 4 should equal 12");
    }

    @Test
    @Disabled("Division by zero handling is being revised")
    void testDivideByZero() {
        assertThrows(IllegalArgumentException.class, () -> mathUtils.divide(10, 0), "Divide by zero should throw IllegalArgumentException");
    }

    @Test
    void testDivide() {
        assertEquals(5, mathUtils.divide(10, 2), "10 / 2 should equal 5");
    }
}
```

Assertions

- Create a class named Person with methods getFullName, getAge, and updateAddress.
- Create a test class named PersonTest.
- Use different types of assertions to verify the behavior of the Person class methods.

@Assertions - Solution

```java
public class Person {

    private String firstName;
    private String lastName;
    private LocalDate birthDate;
    private String address;

    public Person(String firstName, String lastName, LocalDate birthDate, String address) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.birthDate = birthDate;
        this.address = address;
    }

    public String getFullName() {
        return firstName + " " + lastName;
    }

    public int getAge() {
        return Period.between(birthDate, LocalDate.now()).getYears();
    }
}
```

@Assertions - Solution

```
public class Person {

    public String getAddress() {
        return address;
    }

    public void updateAddress(String newAddress) {
        this.address = newAddress;
    }
}
```

@Assertions - Solution

```java
public class PersonTest {

    @Test
    void testGetFullName() {
        Person person = new Person("John", "Doe", LocalDate.of(1990, 1, 1), "123 Main St");
        assertEquals("John Doe", person.getFullName(), "Full name should be 'John Doe'");
    }

    @Test
    void testGetAge() {
        Person person = new Person("Jane", "Doe", LocalDate.of(2000, 1, 1), "456 Elm St");
        assertTrue(person.getAge() > 20, "Age should be greater than 20");
    }

    @Test
    void testAddressNotNull() {
        Person person = new Person("Emily", "Smith", LocalDate.of(1985, 5, 15), "789 Oak St");
        assertNotNull(person.getAddress(), "Address should not be null");
    }
}
```

@Assertions - Solution

```
public class PersonTest {

    @Test
    void testUpdateAddress() {
        Person person = new Person("Robert", "Brown", LocalDate.of(1975, 7, 20), "111 Pine St");
        person.updateAddress("222 Maple St");
        assertEquals("222 Maple St", person.getAddress(), "Updated address should be '222 Maple St'");
    }

    @Test
    void testMultipleAssertions() {
        Person person = new Person("Alice", "Johnson", LocalDate.of(1995, 12, 10), "333 Birch St");

        assertAll("person",
            () -> assertEquals("Alice Johnson", person.getFullName(), "Full name should be 'Alice Johnson'"),
            () -> assertTrue(person.getAge() > 25, "Age should be greater than 25"),
            () -> assertEquals("333 Birch St", person.getAddress(), "Address should be '333 Birch St'")
        );
    }
}
```

@Assertions - Solution

```java
public class PersonTest {

    @Test
    void testExceptionThrown() {
        Person person = new Person("Eve", "White", LocalDate.of(2010, 3, 25), "444 Cedar St");

        IllegalArgumentException thrown = assertThrows(IllegalArgumentException.class, () -> {
            if (person.getAge() < 18) {
                throw new IllegalArgumentException("Person is underage");
            }
        });
        assertEquals("Person is underage", thrown.getMessage());
    }
    @Test
    void testArrayEquals() {
        Person person1 = new Person("Tom", "Green", LocalDate.of(1980, 6, 30), "555 Willow St");
        Person person2 = new Person("Tom", "Green", LocalDate.of(1980, 6, 30), "555 Willow St");

        String[] expectedFullName = { "Tom", "Green" };
        String[] actualFullName = person1.getFullName().split(" ");

        assertArrayEquals(expectedFullName, actualFullName, "Full name parts should be equal");
    }
}
```

Verisoft

Good Luck