JUnit 5 – Java Testing Framework

Cài đặt Junit 5 trong dự án Maven

```
<!-- https://mvnrepository.com/artifact/org.junit.platform/junit-platform-launcher -->
<dependency>
    <groupId>org.junit.platform</groupId>
    <artifactId>junit-platform-launcher</artifactId>
    <version>1.10.2</version>
    <scope>test</scope>
</dependency>
<!-- https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-engine -->
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <version>5.10.2</version>
    <scope>test</scope>
</dependency>
<!-- https://mvnrepository.com/artifact/org.junit.vintage/junit-vintage-engine -->
<dependency>
    <groupId>org.junit.vintage</groupId>
    <artifactId>junit-vintage-engine</artifactId>
    <version>5.10.2</version>
    <scope>test</scope>
</dependency>
```

Ref: https://junit.org/junit5/docs/current/user-guide/#overview

JUnit 5 bao gồm các module khác nhau từ 3 dự án con (sub-projects):

JUnit 5 = JUnit Platform + JUnit Jupiter + JUnit Vintage

1. **JUnit Platform** đóng vai trò là nền tảng khởi chạy các framework kiểm thử trên JVM. Nó cũng xác định API TestEngine để phát triển framework kiểm thử chạy trên platform. Ngoài ra, nền tảng này còn cung cấp Console Launcher để khởi chạy platform từ command line và trình chạy dựa trên JUnit 4 để chạy bất kỳ TestEngine nào trên nền tảng trong môi trường dựa trên JUnit 4.

   JUnit Platform cũng tồn tại trong các IDE phổ biến (như IntelliJ IDEA, Eclipse, NetBeans, và VS Code) và các công cụ xây dựng (Gradle, Maven, Ant)

2. **JUnit Jupiter** là sự kết hợp của mô hình lập trình mới, mô hình mở rộng để viết testcase và phần mở rộng trong JUnit 5. Jupiter sub-project cung cấp một TestEngine để chạy các kiểm thử dựa trên nền tảng này.

3. **JUnit Vintage** cung cấp TestEngine để chạy các kiểm thử dựa trên JUnit 3 và JUnit 4.

Annotations

JUnit 5 hỗ trợ các annotations sau để viết unit test.

Annotation	Mô tả
@Test	Đặt ở đầu method để thông báo method được sử dụng để kiểm thử (test method)
@ParameterizedTest	Biểu thị rằng test method có nhiều tham số
@RepeatedTest	Biểu thị rằng test method là một kiểm thử được lặp nhiều lần
@TestFactory	Biểu thị rằng test method là một test factory cho các dynamic test
@TestTemplate	Biểu thị rằng method là một template cho các test case được thiết kế để được gọi nhiều lần
@TestMethodOrder	Cấu hình thứ tự thực thi cho các annotation @Test, tương tự như annotation @FixMethodOrder trong JUnit 4
@TestIntance	Quy định vòng đời của các annotation test class
@DisplayName	Khai báo tên hiển thị tùy chỉnh cho các test class hoặc test method
@DisplayNameGeneration	Khai báo tên hiển thị cho các test class được generate
@BeforeEach	Biểu thị rằng annotated method phải được thực thi trước mỗi method @Test, @RepeatedTest hoặc @TestFactory trong lớp hiện tại, nó tương tự như annotation @Before của JUnit 4
@AfterEach	Biểu thị rằng annotated method phải được thực thi sau mỗi method @Test, @RepeatedTest hoặc @ParameterizedTest trong lớp hiện tại, nó tương tự như annotation @After của JUnit 4
@BeforeAll	Biểu thị rằng annotated method phải được thực thi trước tất cả method @Test, @RepeatedTest, @TestFactory và @ParameterizedTest trong lớp hiện tại, nó tương tự như annotation @BeforeClass của JUnit 4
@AfterAll	Biểu thị rằng annotated method phải được thực thi sau tất cả method @Test, @RepeatedTest, @TestFactory và @ParameterizedTest trong lớp hiện tại, nó tương tự như annotation @AfterClass của JUnit 4
@Nested	Biểu thị rằng annotated class là một test class non static lồng nhau. Các method @BeforeAll và @AfterAll không thể được sử dụng trực tiếp trong test class @Nested trừ khi test instance lifecycle được sử dụng
@Tag	Được sử dụng để khai báo các thẻ filter test, ở cấp độ class hoặc method
@Disabled	Được sử dụng để vô hiệu hóa một test method hay một test class

Demo JunitTest

Các file Test sẽ được viết ở folder: src/test/java

```
package json_demo;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNotEquals;

import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;

import entities.Employee;
import jakarta.json.JsonObject;
import services.EmployeeBuilder;
import services.EmployeeWriter;

@DisplayName("Demo JUnit5")
public class JsonDemoTest {

    /**
     * @BeforeAll dùng để chỉ định test method chạy đầu tiên
     * Nó phải được đặt là phương thức tĩnh (static),
     * nếu không chương trình sẽ không biên dịch được
     */
    @BeforeAll
    static void setup() {
        System.out.println("Before ALL");
    }

    /**
     * @BeforeEach chỉ định 1 method sẽ luôn được thực thi trước mỗi test method thực thi
     */
    @BeforeEach
    public void beforeEach() {
        System.out.println("Before Each");
    }
```

```java
    /**
     * @AfterAll Chỉ định method sẽ được thực thi khi tất cả các test method trong class thực thi xong
     * Nó phải được đặt trên static method
     */
    @AfterAll
    public static void afterAll() {
        System.out.println("After ALL");
    }
    @Test
    @DisplayName("whenEmployeeIsConVertedToJsonString_thenAvalidJsonStringIsReturned")
    void whenEmployeeIsConVertedToJsonString_thenAvalidJsonStringIsReturned() {
        Employee emp = new Employee(10001, "John Smith", 10000d);

        JsonObject jsonObject = (new EmployeeWriter(emp)).convertToJsonObject();

        assertEquals(10001L, jsonObject.getJsonNumber("id").longValue(), "Id has expected value");
        assertEquals("John Smith", jsonObject.getString("name"), "Name has expected value");
        assertEquals(10000.0d, jsonObject.getJsonNumber("salary").doubleValue(), "Salary has expected value");

        System.out.println("whenEmployeeIsConVertedToJsonString_thenAvalidJsonStringIsReturned: PASSED");
    }

    @Test
    @DisplayName("whenJsonStringIsConVertedToEmployee_thenAvalidObjectIsReturned")
    void whenJsonStringIsConVertedToEmployee_thenAvalidObjectIsReturned() {
        String json = "{\"id\":1,\"name\":\"John Nguyen\",\"salary\":1000.0}";

        Employee employee = (new EmployeeBuilder(json)).build();

        assertEquals(1L, employee.getId(), "Id has expected value");
        assertEquals("John Nguyen", employee.getName(), "Name has expected value");
        assertEquals(1000.0d, employee.getSalary(), "Salary has expected value");

        System.out.println("whenJsonStringIsConVertedToEmployee_thenAvalidObjectIsReturned: PASSED");
    }


    /**
     * disable() method không được thực thi vì bị chặn bởi @Disabled
     */
    @Test
    @Disabled
    public void disable() {
        System.out.println("Disabled");
    }

    @Test
    @DisplayName("assertNotEqualsExample")
    void assertNotEqualsExample(){
        //Test will failed
        assertNotEquals(2,2);

        //Test will pass
        assertNotEquals(4,5);
    }
}
```

Assertions

Sử dụng class org.junit.jupiter.api.Assertions → cung cấp các static method mà chúng ta có thể sử dụng để viết các assertion

- **assertEquals() và assertNotEquals()**

assertEquals() dùng để xác định giá trị mong đợi và giá trị thực tế bằng nhau.
assertEquals() có nhiều method overloading cho các kiểu dữ liệu khác nhau như: int, float, short,…

```
public static void assertEquals(int expected, int actual)
public static void assertEquals(int expected, int actual, String message)
public static void assertEquals(int expected, int actual, Supplier<String<
messageSupplier)
```

Tương tự, assertNotEquals() dùng để xác định giá trị mong đợi và giá trị thực tế không bằng nhau.
Khác với assertEquals(), assertNotEquals() không có method overloading, mà chỉ chấm nhận kiểu Object

```
public static void assertNotEquals(Object expected, Object actual)
public static void assertNotEquals(Object expected, Object actual, String message)
public static void assertNotEquals(Object expected, Object actual, Supplier<String>
messageSupplier)
```

```
@Test
void assertNotEqualsExample(){
    //Test will failed
    assertNotEquals(2,2);

    //Test will pass
    assertNotEquals(4,5);
}
```

- **assertArrayEquals()**

assertArrayEquals() được áp dụng đối với mảng, nó khẳng định rằng mảng mong đợi và mảng thực tế là bằng nhau. Vì có nhiều kiểu mảng như mảng kiểu int, float, double, … nên assertArrayEquals() cũng có các method overloading cho các kiểu dữ liệu khác nhau.

```
public static void assertArrayEquals(int[] expected, int[] actual)
public static void assertArrayEquals(int[] expected, int[] actual, String message)
public static void assertArrayEquals(int[] expected, int[] actual, Supplier<String>
messageSupplier)
```

Ví dụ:

```
@Test
void assertArrayEqualsExample(){
    //Test will pass
    assertArrayEquals(new int[]{1,2,3}, new int[]{1,2,3});

    //Test will failed because element order is different
    assertArrayEquals(new int[]{1,2,3}, new int[]{1,3,2});

    //Test will failed because number of elements are different
    assertArrayEquals(new int[]{1,2,3}, new int[]{1,2,3,4});
}
```

- **assertNull() và assertNotnull()**

assertNull() khẳng định rằng một object là null. Ngược lại assertNotNull() khẳng định rằng object là not null. Cả hai đều có 3 method overloading:

```
public static void assertNotNull(Object actual)
public static void assertNotNull(Object actual, String message)
public static void assertNotNull(Object actual, Supplier<String> messageSupplier)

public static void assertEquals(Object actual)
public static void assertEquals(Object actual, String message)
public static void assertEquals(Object actual, Supplier<String> messageSupplier)
```

Ví dụ:

```
@Test
void assertNull_assertNotNull(){
    String nullString = null;
    String notNullString = "Demo Junit 5";

    //Test will pass
    assertNull(nullString);
    assertNotNull(notNullString);

    //Test will failed
    assertNull(notNullString);
    assertNotNull(nullString);
}
```

- **assertTrue() và assetFalse()**

assertTrue dùng để xác minh điều kiện phải trả về *true.*

```
public static void assertTrue(boolean condition)
public static void assertTrue(boolean condition, String message)
public static void assertTrue(boolean condition, Supplier<String> messageSupplier)
public static void assertTrue(BooleanSupplier booleanSupplier)
public static void assertTrue(BooleanSupplier booleanSupplier, String message)
public static void assertTrue(BooleanSupplier booleanSupplier, Supplier<String>
messageSupplier)
```

assertFalse() dùng để xác minh điều kiện trả về là *false*

```
public static void assertFalse(boolean condition)
public static void assertFalse(boolean condition, String message)
public static void assertFalse(boolean condition, Supplier<String> messageSupplier)
public static void assertFalse(BooleanSupplier booleanSupplier)
public static void assertFalse(BooleanSupplier booleanSupplier, String message)
public static void assertFalse(BooleanSupplier booleanSupplier, Supplier<String>
messageSupplier)
```

Ví dụ:

```
@Test
void assertTrue_assertFalse(){
    //Test will pass
    assertTrue(true);
    assertFalse(false);
    assertTrue(5 > 4, "5 is greater the 4");
    assertTrue(null == null, "null is equal to null");
    assertFalse(4 > 5, "5 is greater the 4" );

    //Test will failed
    assertTrue(false);
    assertTrue(5 < 4, "4 is greater the 5");
    assertFalse(4 < 5, "4 is greater the 5" );
}
```

- **assertSame() và assertNotSame()**

assertSame() khẳng định rằng 2 object có cùng tham chiếu tới chính xác cùng một object. assertNotSame() khẳng định rằng 2 object không tham chiếu đến cùng một đối tượng.

```
public static void assertNotSame(Object actual)
public static void assertNotSame(Object actual, String message)
public static void assertNotSame(Object actual, Supplier<> messageSupplier)

public static void assertSame(Object actual)
public static void assertSame(Object actual, String message)
public static void assertSame(Object actual, Supplier<String> messageSupplier)
```

Ví dụ:

```
@Test
void assertSame_assertNotSame(){
    String originalObject = "Demo Junit 5";
    String cloneObject = originalObject;
    String otherObject = "JUnit 5";

    //Test will pass
    assertSame(originalObject, cloneObject);
    assertNotSame(originalObject, otherObject);

    //Test will failed
    assertSame(originalObject,originalObject);
    assertNotSame(originalObject,cloneObject);
}
```

- **assertThrows()**

assertThrows() xác định rằng việc thực thi Excutable được cung cấp sẽ ném ra một exception của exceptionTYpe và trả về exception

```
public static <T extends Throwable> T assertThrows(Class<T> expectedType,
Executable executable)
```

Ví dụ:

```
@Test
void assertThrowsExample(){
    Throwable exception = assertThrows(IllegalArgumentException.class, () -> {
        throw new IllegalArgumentException("error message");
    });
}
```

- **fail()**: dùng để đánh dấu một test case chưa hoàn thiện. thường được sử dụng để đánh dấu các test cho những phần task đang trong quá trình phát triển

```
public static void fail(String message)
public static void fail(Throwable cause)
public static void fail(String message, Throwable cause)
public static void fail(Supplier<String> messageSupplier)
```

Ví dụ:

```
@Test
void failExample() {
     fail("FAIL - try to development");
}
```

- **assertAll():** cho phép tạo một nhóm các assertions, trong đó tất cả các assertions đều được thực thi

Ví dụ:

```
@Test
void assertAllExamples(){
    assertAll(
            "heading",
            () -> {
                assertTrue(5 > 4, "5 is greater the 4");
                assertFalse(4 > 5, "5 is greater the 4" );
            },
            () -> {
                String nullString = null;
                String notNullString = "Techmaster";

                assertNull(nullString);
                assertNotNull(notNullString);
            },
            () ->{
                assertArrayEquals(new int[]{1,2,3}, new int[]{1,2,3});
            }
    );
}
```