**Digital Nurture 4.0 Java FSE**

**Week 2 TDD using JUnit5 and Mockito**

**(Mandatory)**

**JUnit\_Basic Testing Exercises**

**Exercise 01: Setting Up JUnit**

**Scenario:**

You need to set up JUnit in your Java project to start writing unit tests.

**Steps:**

1. Create a new Java project in your IDE (e.g., IntelliJ IDEA, Eclipse).

2. Add JUnit dependency to your project. If you are using Maven, add the following to your

pom.xml:

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

3. Create a new test class in your project

**Answer - Code:**

**Pom.xml –**

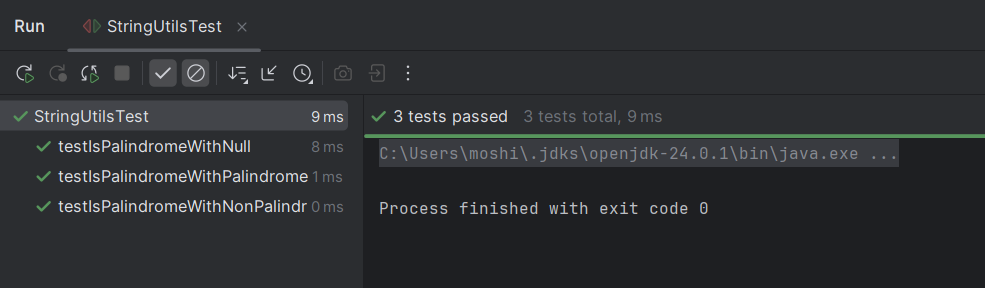
<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>org.example</groupId>  
 <artifactId>Setup</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <properties>  
 <maven.compiler.source>24</maven.compiler.source>  
 <maven.compiler.target>24</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.13.2</version>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
</project>

**Creation of Java Class –**

public class StringUtils {  
 public boolean isPalindrome(String str) {  
 if (str == null) return false;  
 String reversed = new StringBuilder(str).reverse().toString();  
 return str.equals(reversed);  
 }  
}

**Creation of Test class –**

import static org.junit.Assert.\*;  
public class StringUtilsTest {  
 @Test  
 public void testIsPalindromeWithPalindrome() {  
 StringUtils utils = new StringUtils();  
 *assertTrue*(utils.isPalindrome("madam"));  
 }  
 @Test  
 public void testIsPalindromeWithNonPalindrome() {  
 StringUtils utils = new StringUtils();  
 *assertFalse*(utils.isPalindrome("hello"));  
 }  
 @Test  
 public void testIsPalindromeWithNull() {  
 StringUtils utils = new StringUtils();  
 *assertFalse*(utils.isPalindrome(null));  
 }  
}

**Output:**

**Exercise 03: Assertions in Junit**

**Scenario:**

You need to use different assertions in JUnit to validate your test results.

**Steps:**

1. Write tests using various JUnit assertions.

Solution Code:

public class AssertionsTest {

@Test

public void testAssertions() {

// Assert equals

assertEquals(5, 2 + 3);

// Assert true

assertTrue(5 > 3);

// Assert false

assertFalse(5 < 3);

// Assert null

assertNull(null);

// Assert not null

assertNotNull(new Object());

}

}

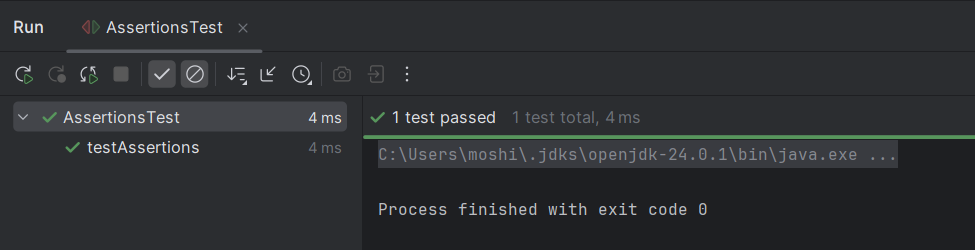
**Answer – Code:**

**Pom.xml –**

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0  
 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>org.example</groupId>  
 <artifactId>assertion</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <properties>  
 <maven.compiler.source>24</maven.compiler.source>  
 <maven.compiler.target>24</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.13.2</version>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
</project>

**Assertion Test Code –**

import org.junit.Test;  
import static org.junit.Assert.\*;  
public class AssertionsTest {  
 @Test  
 public void testAssertions() {  
 // assertEquals  
 *assertEquals*(5, 2 + 3);  
 // assertTrue  
 *assertTrue*(5 > 3);  
 // assertFalse  
 *assertFalse*(5 < 3);  
 // assertNull  
 *assertNull*(null);  
 *assertNotNull*(new Object());  
 }  
}

**Output:**

**Exercise 04: Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and Teardown Methods in Junit**

**Scenario:**

You need to organize your tests using the Arrange-Act-Assert (AAA) pattern and use setup

and teardown methods.

**Steps:**

1. Write tests using the AAA pattern.

2. Use @Before and @After annotations for setup and teardown methods

**Answer – Code:**

**Pom.xml –**

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0  
 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>org.example</groupId>  
 <artifactId>assertion</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <properties>  
 <maven.compiler.source>24</maven.compiler.source>  
 <maven.compiler.target>24</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.13.2</version>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
  
</project>

**Creation of Java class –**

public class TemperatureConverter {

public double toFahrenheit(double celsius) {

return (celsius \* 9.0 / 5) + 32;

}

public double toCelsius(double fahrenheit) {

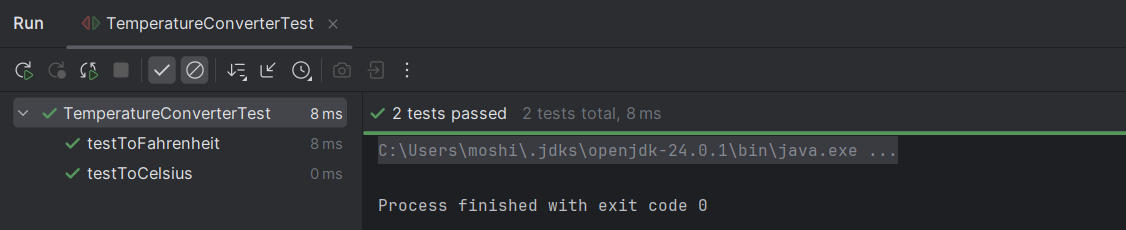
return (fahrenheit - 32) \* 5.0 / 9

}

}

**Creation of Test Class –**

import org.junit.Before;  
import org.junit.After;  
import org.junit.Test;  
import static org.junit.Assert.\*;  
public class TemperatureConverterTest {  
 private TemperatureConverter converter;  
 @Before  
 public void setUp() {  
 converter = new TemperatureConverter();  
 }  
 @After  
 public void tearDown() {  
 converter = null;  
 }  
 @Test  
 public void testToFahrenheit()   
 double result = converter.toFahrenheit(0);  
 *assertEquals*(32.0, result, 0.001);  
 }  
  
 @Test  
 public void testToCelsius() {  
 double result = converter.toCelsius(212);  
 *assertEquals*(100.0, result, 0.001);  
 }  
}

**Output:**

**Mockito Exercises**

**Exercise 01: Mocking and Stubbing**

**Scenario:**

You need to test a service that depends on an external API. Use Mockito to mock the external API and stub its methods.

**Steps:**

1. Create a mock object for the external API.

2. Stub the methods to return predefined values.

3. Write a test case that uses the mock object.

**Solution Code:**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

@Test

public void testExternalApi() {

ExternalApi mockApi = Mockito.mock(ExternalApi.class);

when(mockApi.getData()).thenReturn("Mock Data");

MyService service = new MyService(mockApi);

String result = service.fetchData();

assertEquals("Mock Data", result);

}

}

**Answer – Code:**

**EXTERNALAPI.java**

public interface ExternalApi {

String getData();

String getData(String type);

void saveData(String data);

}

**MYSERVICES.java**

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

public String fetchTypedData(String type) {

return api.getData(type);

}

public void save(String data) {

api.saveData(data);

}

}

**MYSERVICETEST.java**

import org.junit.jupiter.api.Test;

import org.mockito.InOrder;

import org.mockito.Mockito;

import static org.mockito.Mockito.\*;

import static org.junit.jupiter.api.Assertions.\*;

public class MyServiceTest {

@Test

public void testMockingAndStubbing() {

ExternalApi mockApi = Mockito.mock(ExternalApi.class);

when(mockApi.getData()).thenReturn("Mock Data");

MyService service = new MyService(mockApi);

assertEquals("Mock Data", service.fetchData());

}

**Exercise 02: Verifying Interactions**

**Scenario:**

You need to ensure that a method is called with specific arguments.

**Steps:**

1. Create a mock object.

2. Call the method with specific arguments.

3. Verify the interaction.

Solution Code:

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

ExternalApi mockApi = Mockito.mock(ExternalApi.class);

MyService service = new MyService(mockApi);

service.fetchData();

verify(mockApi).getData();

}

}

**Answer – Code:**

@Test

public void testVerifyInteraction() {

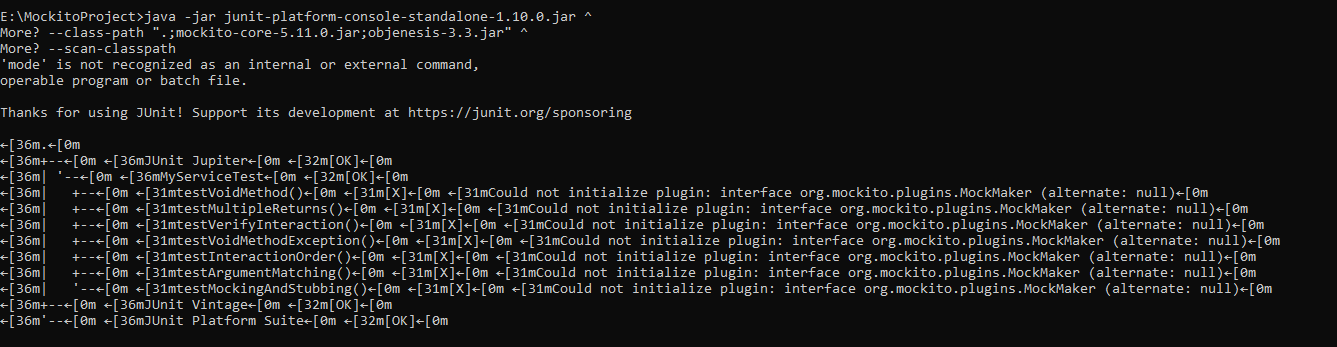
ExternalApi mockApi = Mockito.mock(ExternalApi.class);

MyService service = new MyService(mockApi);

service.fetchData();

verify(mockApi).getData();

}

 **Output:(Exercise 01 & Exercise 02)**