**Exercise 1: Setting Up Junit**

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

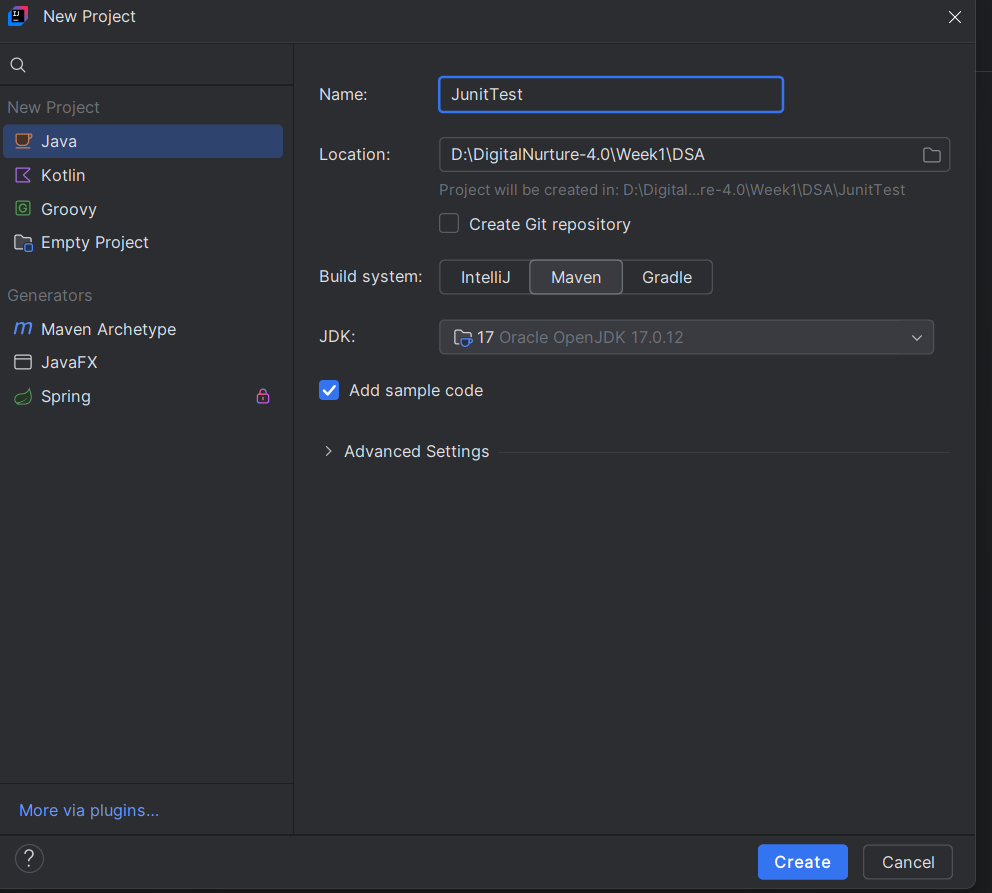
Steps: 1. Create a new Java project in your IDE (e.g., IntelliJ IDEA, Eclipse).

2. Add JUnit dependency to your project. If you are using Maven, add the following to your pom.xml: junit junit 4.13.2 test

<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.

**PROJECT CREATION:**

****

**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>JunitTest</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <properties>  
 <maven.compiler.source>17</maven.compiler.source>  
 <maven.compiler.target>17</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>

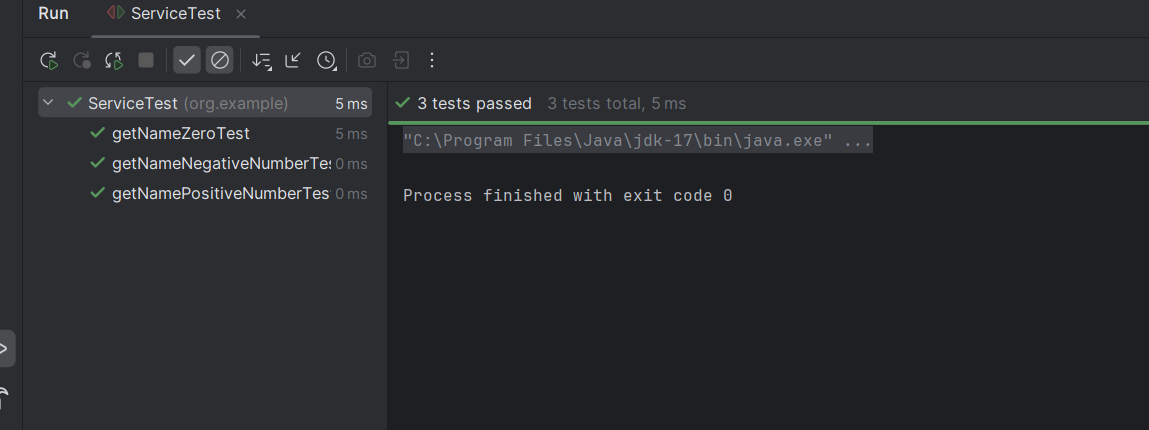
**Service.java**

package org.example;  
public class Service {  
 public String getName(int number)  
 {  
 if(number <0)  
 {  
 return "a";  
 }  
 else{  
 return "b";  
 }  
 }  
}

**ServiceTest.java**

package org.example;  
  
import org.junit.Assert;  
import org.junit.Test;  
  
import static org.junit.Assert.\*;  
  
public class ServiceTest {  
 @Test  
 public void getNameNegativeNumberTest()  
 {  
 Service s=new Service();  
 String res=s.getName(-1);  
 Assert.*assertEquals*("a",res);  
  
 }  
 @Test  
 public void getNamePositiveNumberTest()  
 {  
 Service s=new Service();  
 String res=s.getName(1);  
 Assert.*assertEquals*("b",res);  
  
 }  
  
 @Test  
 public void getNameZeroTest()  
 {  
 Service s=new Service();  
 String res=s.getName(0);  
 Assert.*assertEquals*("b",res);  
  
 }  
}

**OUTPUT:**

****

**Exercise 3: 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());

}

}

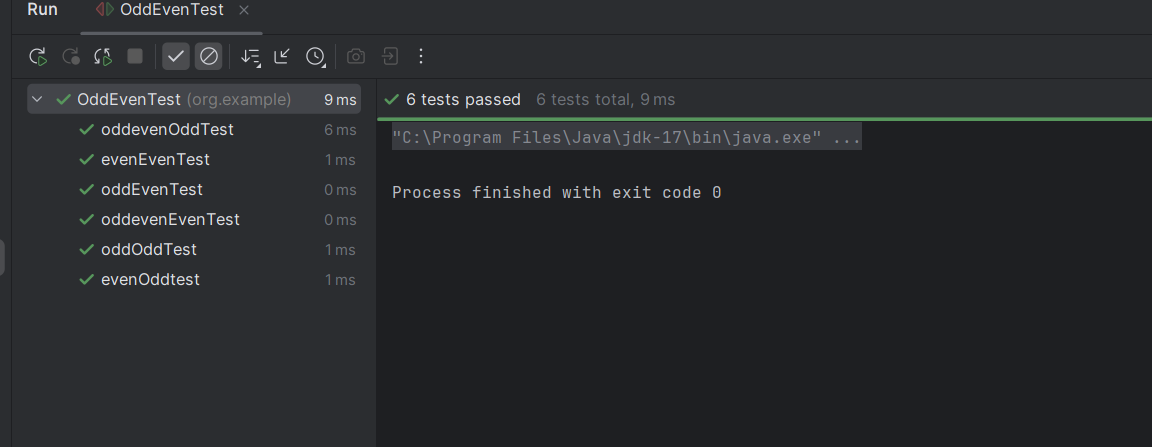
**OddEven.java**

package org.example;  
  
public class OddEven {  
 public String oddeven(int x)  
 {  
 if(x%2==0)  
 {  
 return "even";  
 }  
 else{  
 return "odd";  
 }  
 }  
 public boolean odd(int x)  
 {  
 if(x%2!=0)  
 {  
 return true;  
 }  
 return false;  
 }  
 public boolean even(int x)  
 {  
 if(x%2==0)  
 {  
 return true;  
 }  
 return false;  
 }  
  
}

**OddEvenTest.java**

package org.example;  
  
import org.junit.Assert;  
import org.junit.Test;  
  
import static org.junit.Assert.\*;  
import static org.junit.Test.\*;  
public class OddEvenTest {  
 @Test  
 public void oddevenEvenTest()  
 {  
 OddEven o=new OddEven();  
 int inp=2;  
 String exp="even";  
 *assertEquals*(exp,o.oddeven(inp));  
 }  
 @Test  
 public void oddevenOddTest()  
 {  
 OddEven o=new OddEven();  
 int inp=7;  
 String exp="odd";  
 *assertEquals*(exp,o.oddeven(inp));  
 }  
 @Test  
 public void oddOddTest()  
 {  
 OddEven o=new OddEven();  
 *assertTrue*(o.odd(7));  
 }  
 @Test  
 public void oddEvenTest()  
 {  
 OddEven o=new OddEven();  
 *assertFalse*(o.odd(8));  
 }  
 @Test  
 public void evenEvenTest()  
 {  
 OddEven o=new OddEven();  
 *assertTrue*(o.even(8));  
 }  
 @Test  
 public void evenOddtest()  
 {  
 OddEven o=new OddEven();  
 *assertFalse*(o.even(7));  
 }  
  
}

**OUTPUT:**

****

**Exercise 4: 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.

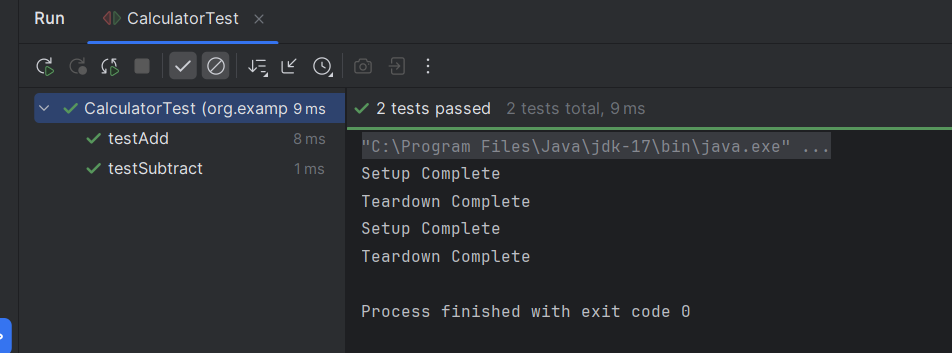
**Calculator.java**

package org.example;  
  
public class Calculator {  
 public int add(int a, int b) {  
 return a + b;  
 }  
  
 public int subtract(int a, int b) {  
 return a - b;  
 }  
}

**CalculatorTest.java**

package org.example;  
import org.junit.After;  
import org.junit.Before;  
import org.junit.Test;  
import static org.junit.Assert.\*;  
public class CalculatorTest {  
 private Calculator calculator;  
 @Before  
 public void setUp() {  
 calculator = new Calculator();  
 System.*out*.println("Setup Complete");  
 }  
 @After  
 public void tearDown() {  
 System.*out*.println("Teardown Complete");  
 }  
 @Test  
 public void testAdd() {  
 int result = calculator.add(10, 5);  
 *assertEquals*(15, result);  
 }  
 @Test  
 public void testSubtract() {  
 int result = calculator.subtract(10, 3);  
 *assertEquals*(7, result);  
 }  
}

**OUTPUT:**

****

**MOCKITO EXERCISES:**

**Exercise 1: 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);

}

}

**ExternalApi.java**

package org.example;  
  
public interface ExternalApi {  
 String getData();  
}

**MyService.java**

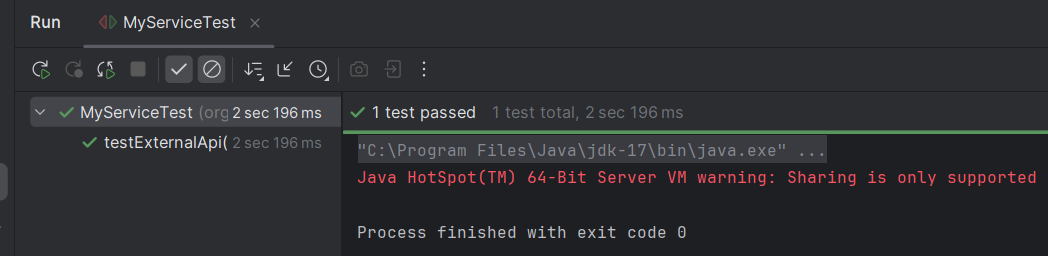
package org.example;  
  
public class MyService {  
 private ExternalApi api;  
  
 public MyService(ExternalApi api) {  
 this.api = api;  
 }  
  
 public String fetchData() {  
 return api.getData();  
 }  
}

**MyServiceTest.java**

package org.example;  
import org.junit.jupiter.api.Test;  
import static org.mockito.Mockito.\*;  
import static org.junit.jupiter.api.Assertions.\*;

class MyServiceTest {  
 @Test  
 public void testExternalApi() {  
 ExternalApi mockApi = *mock*(ExternalApi.class);  
 *when*(mockApi.getData()).thenReturn("Mock Data");  
 MyService service = new MyService(mockApi);  
 String result = service.fetchData();  
 *assertEquals*("Mock Data", result);  
 }  
}

**OUTPUT:**

****

**Exercise 2: 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();

}

}

**ExternalApi.java**

package org.example;  
  
public interface ExternalApi {  
 String getData();  
}

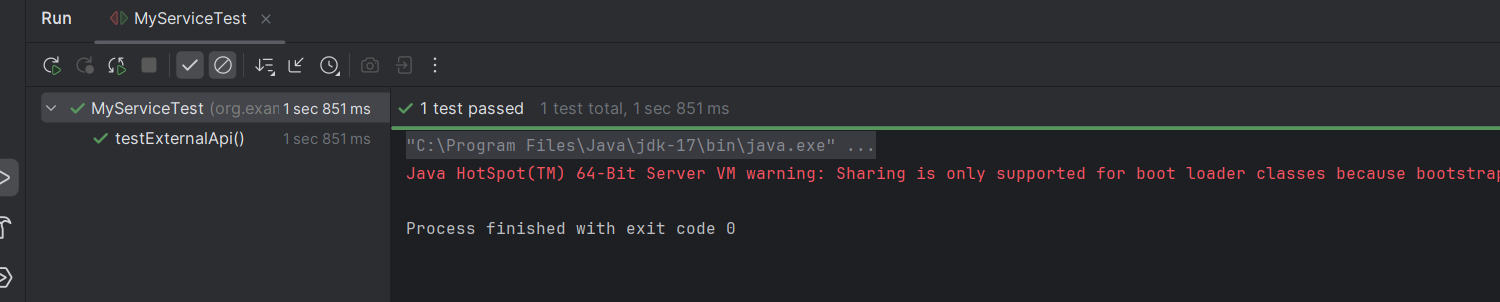
**MyService.java**

package org.example;  
  
public class MyService {  
 private ExternalApi api;  
  
 public MyService(ExternalApi api) {  
 this.api = api;  
 }  
  
 public String fetchData() {  
 return api.getData();  
 }  
}

**MyServiceTest.java**

package org.example;  
import org.junit.jupiter.api.Test;  
import static org.mockito.Mockito.\*;  
import static org.junit.jupiter.api.Assertions.\*;

class MyServiceTest {  
 @Test  
 public void testExternalApi() {  
 ExternalApi mockApi = *mock*(ExternalApi.class);  
 *when*(mockApi.getData()).thenReturn("Mock Data");  
 MyService service = new MyService(mockApi);  
 String result = service.fetchData();  
 *assertEquals*("Mock Data", result);  
 }  
}

**OUTPUT: **

**Logging using SLF4J**

**Exercise 1: Logging Error Messages and Warning Levels Task:**

Write a Java application that demonstrates logging error messages and warning levels using SLF4J.

**Step-by-Step Solution:**

1. Add SLF4J and Logback dependencies to your `pom.xml` file:

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.30</version>

</dependency>

<dependency>

<groupId>ch.qos.logback</groupId>

<artifactId>logback-classic</artifactId>

<version>1.2.3</version>

</dependency>

2. Create a Java class that uses SLF4J for logging:

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class LoggingExample {

private static final Logger logger = LoggerFactory.getLogger(LoggingExample.class);

public static void main(String[] args) {

logger.error("This is an error message");

logger.warn("This is a warning message");

}

} **LoggingExample.java**

package org.slf4j;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
public class LoggingExample {  
 private static final Logger *logger* = LoggerFactory.*getLogger*(LoggingExample.class);  
 public static void main(String[] args) {  
 *logger*.error("This is an error message");  
 *logger*.warn("This is a warning message");  
 }  
}

**OUTPUT:**

