**JUnit Testing Exercises**

**Exercise 1: Setting Up Junit**

**Scenario:**

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

**Steps:**

**1. We create a maven project in Eclipse IDE called ‘Junit-demo’ and add the given dependency in the pom.xml file.**

**Junit-demo/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>com.simu</groupId>

<artifactId>junit-demo</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**2. We now create the respective files for testing the dependency – Calculator.java and CalculatorTest.java.**

**src/main/java/com/simu/Calculator.java:**

**package** com.simu;

**public** **class** Calculator {

**public** **int** add (**int** a, **int** b) {

**return** a + b;

}

}

**src/test/java/com/test/CalculatorTest.java:**

**package** com.test;

**import** com.simu.Calculator;

**import** org.junit.Test;

**import** **static** org.junit.Assert.\*;

**public** **class** CalculatorTest {

@Test

**public** **void** testAdd() {

Calculator calc = **new** Calculator();

**int** result = calc.add(2, 3);

*assertEquals*(5, result);

}

}

**Output Screenshot for the above exercise:**

**A screenshot of a computer

AI-generated content may be incorrect.**

The above screenshot is the pom.xml file, where we add the dependencies and the properties for the maven project, respectively. We would have to add the ‘junit’ dependency as mentioned in the exercise that is given.   
  
While adding the dependency, make sure that we add the dependency in the <dependencies> tag. All the dependencies come under this tag, respectively. Also, after adding the dependencies, update the maven project.

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Exercise 3: Assertions in Junit**

**Scenario:**

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

**Code for the above scenario is given as,**

**package** com.test;

**import** org.junit.Test;

**import** **static** org.junit.Assert.\*;

**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**);

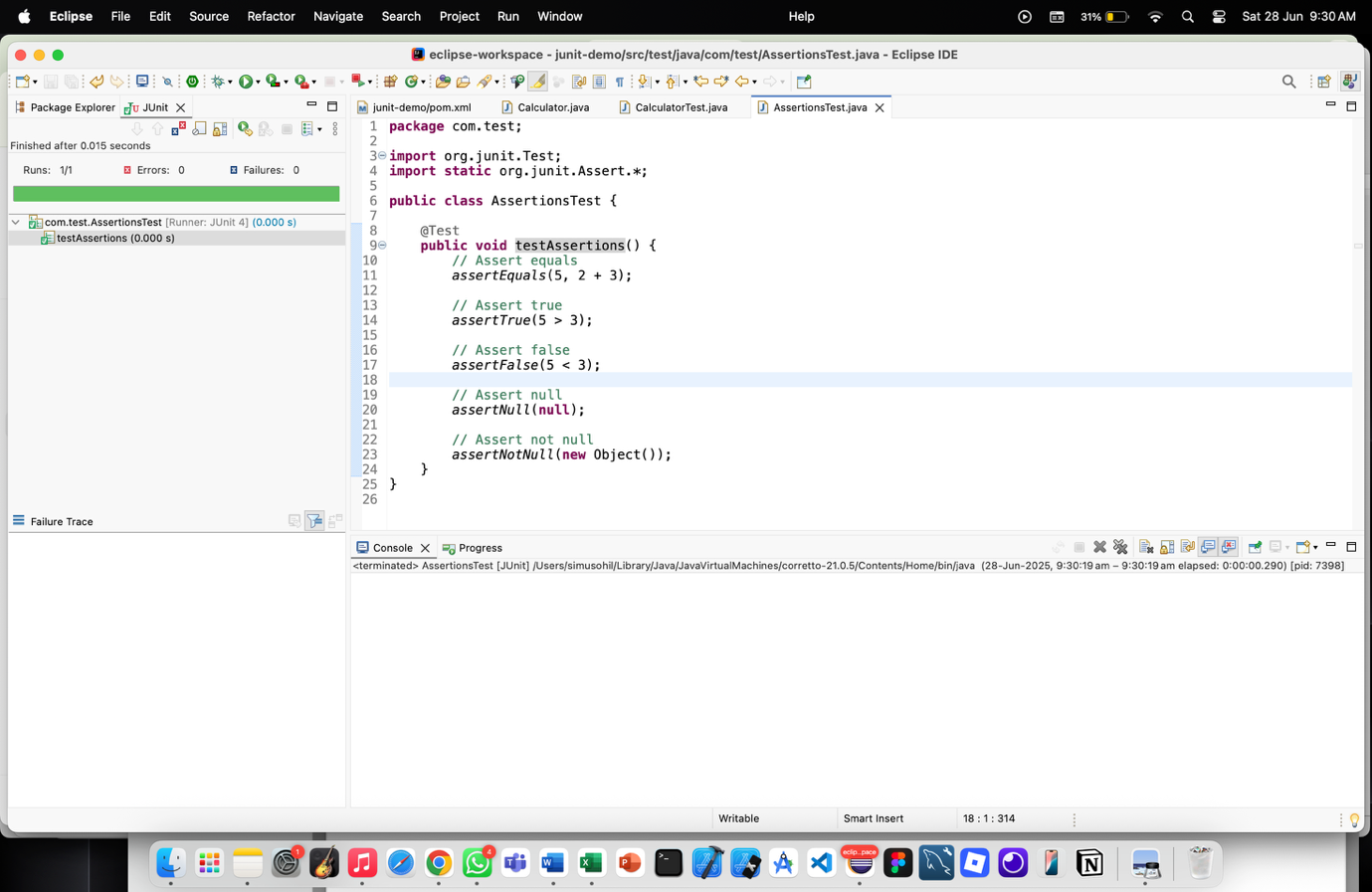
// Assert not null

*assertNotNull*(**new** Object());

}

}

**Output screenshot for the above code is:**

****

**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.

We will be using the same Calculator.java and CalculatorTest.java files but with slight modifications to perform the above exercise.

**Calculator.java:**

**package** com.simu;

**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** com.test;

**import** com.simu.Calculator;

**import** org.junit.Test;

**import** **static** org.junit.Assert.\*;

**import** org.junit.After;

**import** org.junit.Before;

**public** **class** CalculatorTest {

**private** Calculator calculator;

@Before

**public** **void** setUp() {

System.***out***.println("Setting up...");

calculator = **new** Calculator();

}

@After

**public** **void** tearDown() {

System.***out***.println("Cleaning up...");

calculator = **null**;

}

@Test

**public** **void** testAddition() {

**int** result = calculator.add(10, 5);

*assertEquals*(15, result);

}

@Test

**public** **void** testSubtraction() {

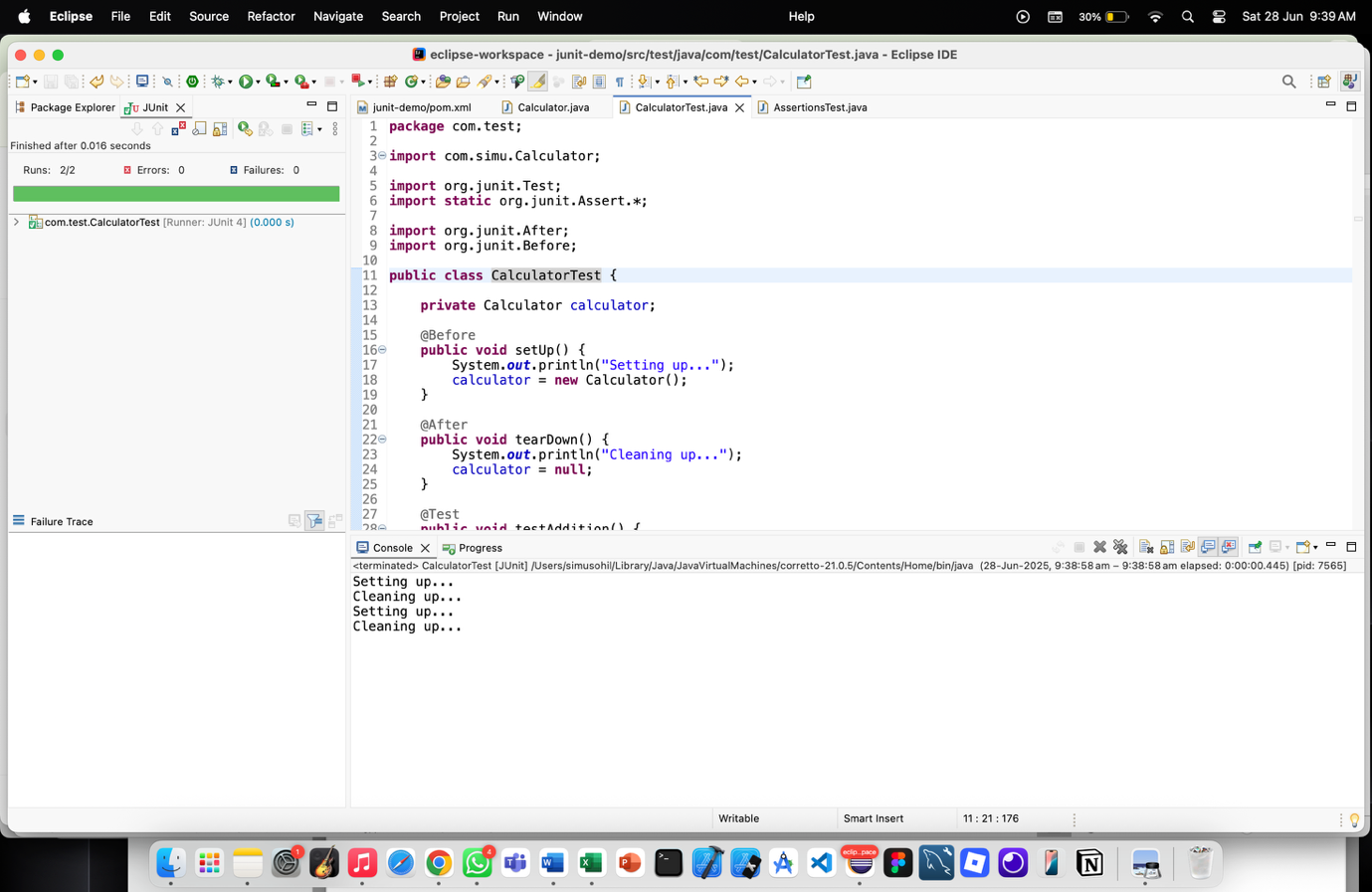
**int** result = calculator.subtract(10, 5);

*assertEquals*(5, result);

}

}

**Output Screenshot for the above exercise:**

****

**Mockito Hands-On 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.

Now, we will have to create a maven in project in Eclipse IDE. After creating the maven project, we should follow the above given steps in order to solve the first exercise, respectively.

**Junit-demo/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>com.simu</groupId>

<artifactId>junit-demo</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>5.12.0</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**src/main/java/com/simu/ExternalApi.java (Interface)**

**package** com.simu;

**public** **interface** ExternalApi {

String getData();

}

**src/test/java/com/test/MyService.java**

**package** com.test;

**import** com.simu.ExternalApi;

**public** **class** MyService {

**private** ExternalApi api;

**public** MyService(ExternalApi api) {

**this**.api = api;

}

**public** String fetchData() {

**return** api.getData();

}

}

**src/test/java/com/test/MyServiceTest.java**

**package** com.test;

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

**import** **static** org.mockito.Mockito.*when*;

**import** org.junit.jupiter.api.Test;

**import** org.mockito.Mockito;

**import** com.simu.ExternalApi;

**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);

}

}

**Output Screenshot for the above exercise:**

**A screen shot of a computer

AI-generated content may be incorrect.**

**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.

**The code for implementing the above exercise:**

**package** com.test;

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

**import** **static** org.mockito.Mockito.*verify*;

**import** **static** org.mockito.Mockito.*when*;

**import** org.junit.jupiter.api.Test;

**import** org.mockito.Mockito;

**import** com.simu.ExternalApi;

**public** **class** MyServiceTest {

@Test

**public** **void** testVerifyInteraction() {

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

MyService service = **new** MyService(mockApi);

service.fetchData();

*verify*(mockApi).getData(); // Verifies that getData() was called

}

}

**Output screenshot for the above exercise:**

**A screenshot of a computer

AI-generated content may be incorrect.**