JUnit Testing Exercises

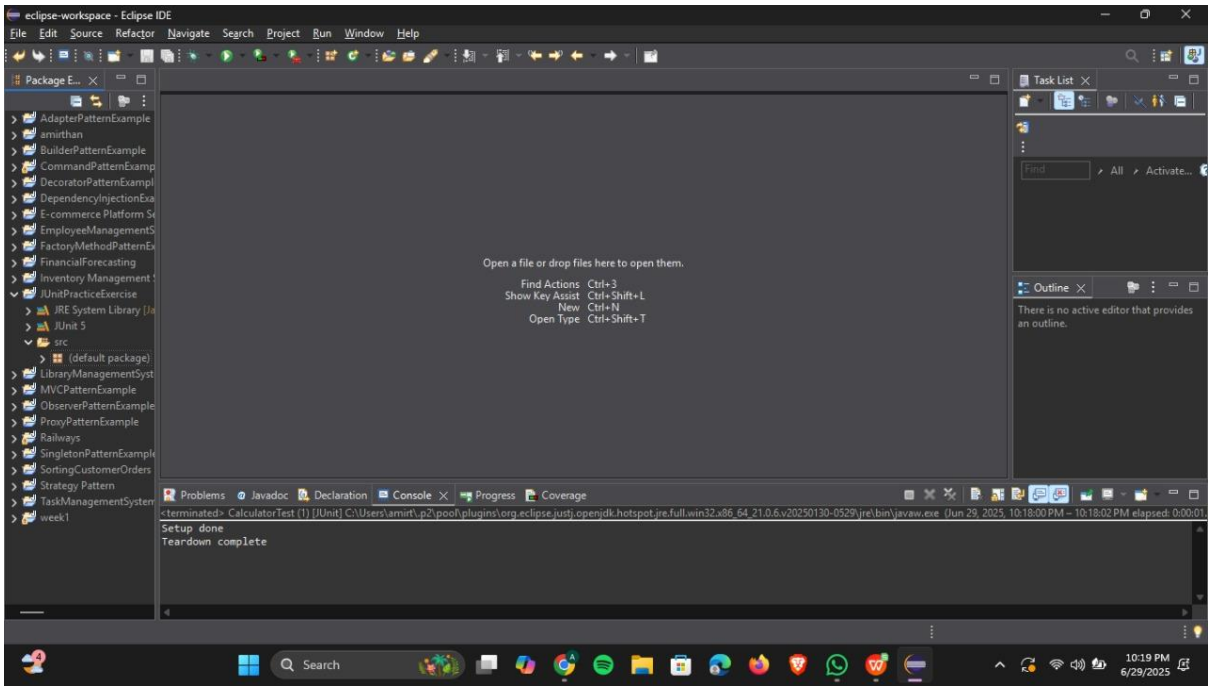
Exercise 1: Setting Up JUnit

Scenario:

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

Steps:

1.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:

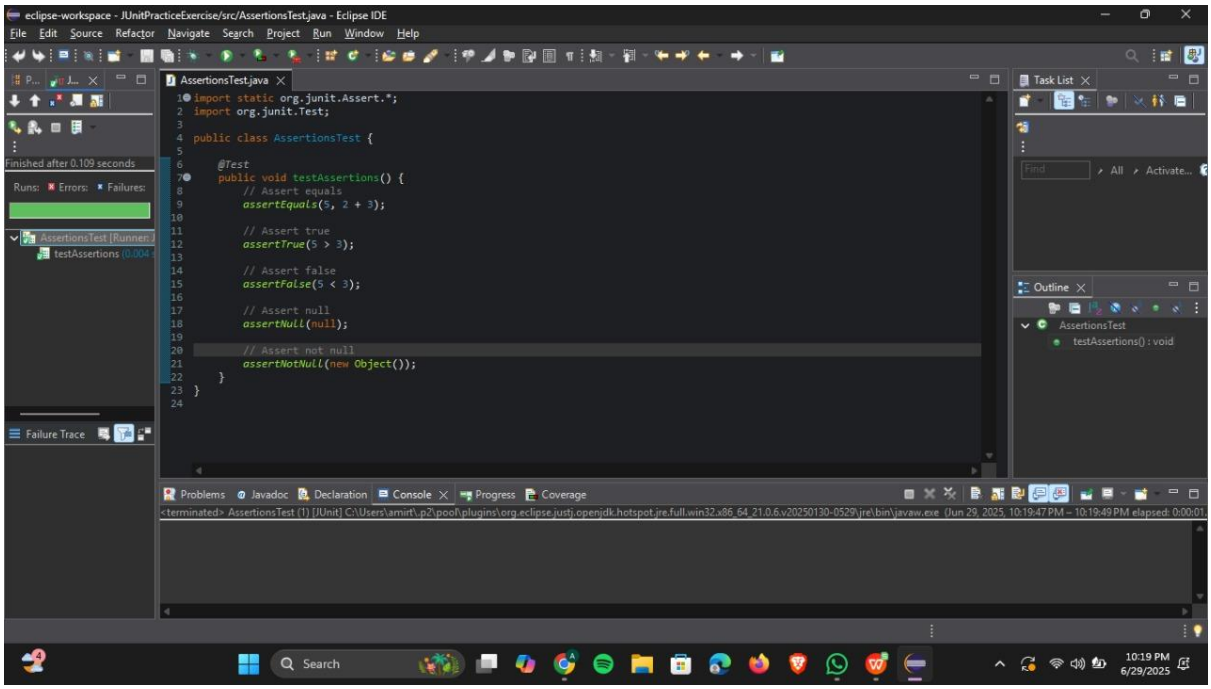
```
<dependency>
 <groupId>junit</groupId>
 <artifactId>junit</artifactId>
 <version>4.13.2</version>
 <scope>test</scope>
</dependency>
```

CODE:

```
import static org.junit.Assert.*;
import org.junit.Test;
```

```
public class AssertionsTest {

    @Test
    public void testAssertions() {

        assertEquals(5, 2 + 3);

        assertTrue(5 > 3)
        assertFalse(5 < 3);
        assertNull(null);


        assertNotNull(new Object());
    }
}
```

OUTPUT:

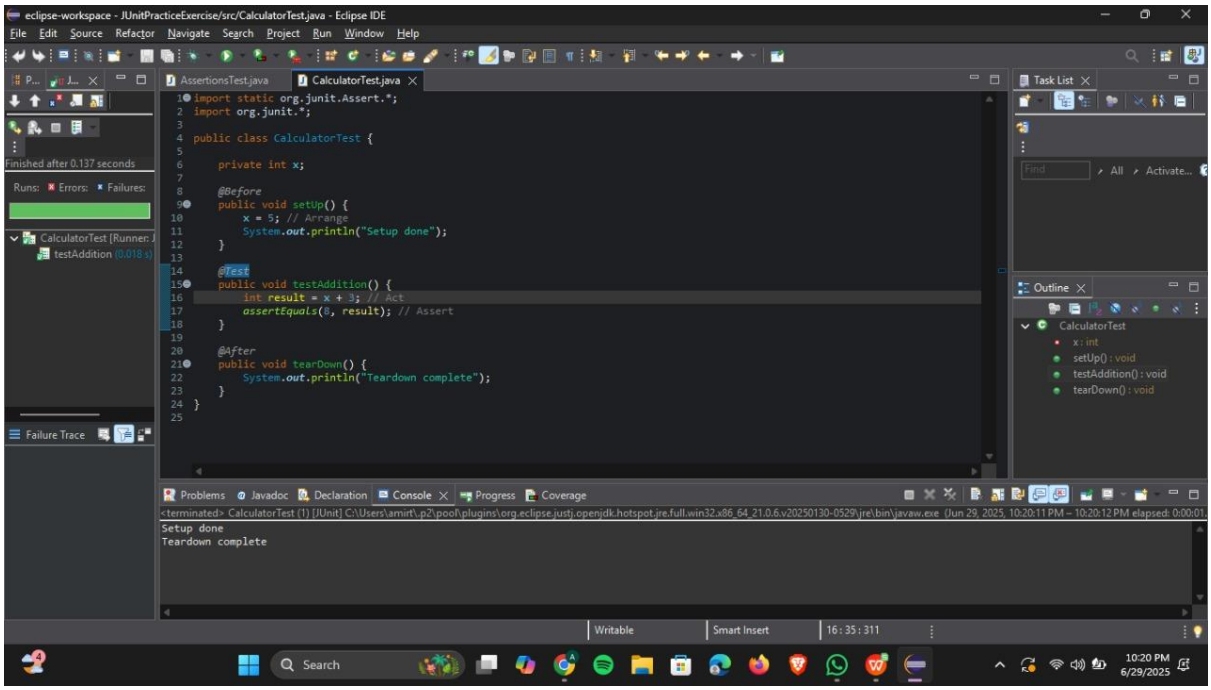


3. Create a new test class in your project.

CODE:

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

public class CalculatorTest {

   private int x;

   @Before
   public void setUp() {
      x = 5;
      System.out.println("Setup done");
   }

   @Test
   public void testAddition() {
      int result = x + 3;
      assertEquals(8, result);
   }

   @After
   public void tearDown() {
      System.out.println("Teardown complete");
   }
}
```

OUTPUT:


eclipse-workspace - JUnitPracticeExercise/src/CalculatorTest.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
AssertionsTest.java
CalculatorTest.java
import static org.junit.Assert.*;
import org.junit.*;
public class CalculatorTest {
private int x;
@Before
public void setUp() {
x = 5; // Arrange
System.out.println("Setup done");
}
@Test
public void testAddition() {
int result = x + 3; // Act
assertEquals(8, result); // Assert
}
@After
public void tearDown() {
System.out.println("Teardown complete");
}
}
Finished after 0.137 seconds
Runs: Errors: Failures:
CalculatorTest [Runner:
testAddition (0.018 s)
Failure Trace
Task List
Outline
CalculatorTest
x : int
setUp() : void
testAddition() : void
tearDown() : void
Problems Javadoc Declaration Console Progress Coverage
<terminated> CalculatorTest (1) [JUnit] C:\Users\amirt\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.6.v20250130-0529\jre\bin\javaw.exe (Jun 29, 2025, 10:20:11 PM – 10:20:12 PM elapsed: 0:00:01
Setup done
Teardown complete
Writable
Smart Insert
16 : 35 : 311
Search
10:20 PM
6/29/2025

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.

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);
 }
}
```

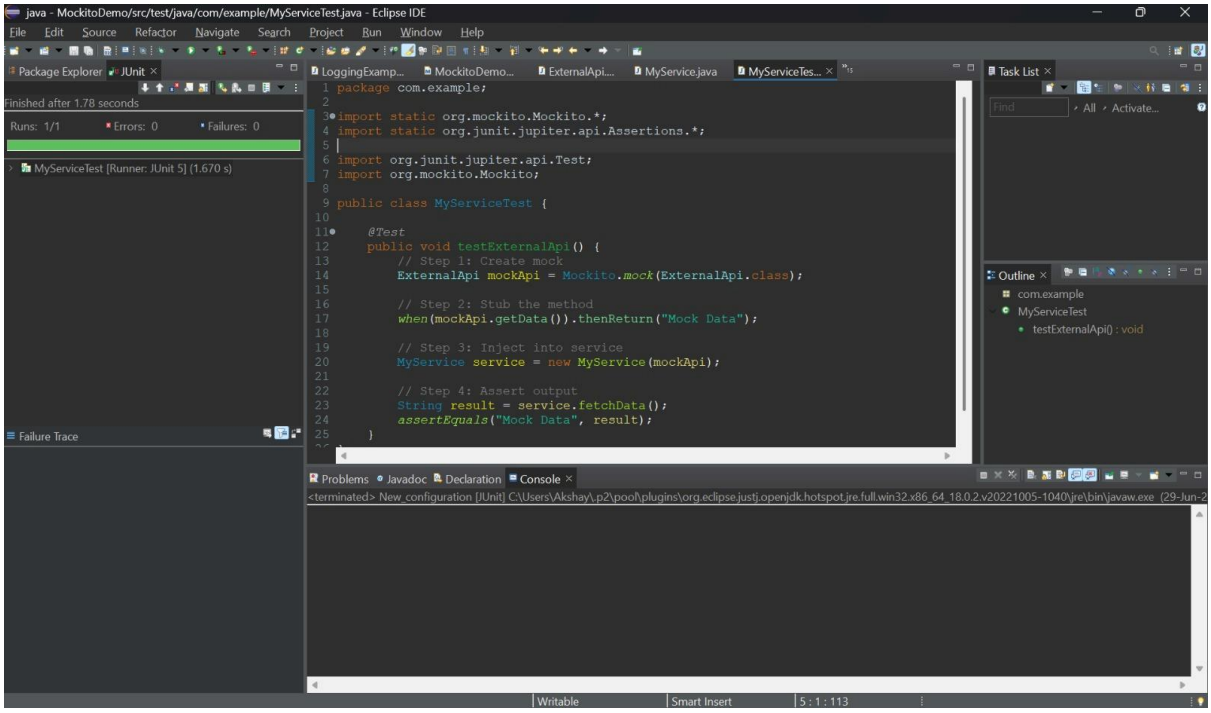
CODE:

```
package com.example;

public interface ExternalApi {
        String getData();
}
```

```
package com.example;

public class MyService {
    private ExternalApi api;

    public MyService(ExternalApi api) {
        this.api = api;
    }

    public String fetchData() {
        return api.getData();
    }
}

package com.example;

import static org.mockito.Mockito.*;
import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.Test;
import org.mockito.Mockito;

public class MyServiceTest {

    @Test
    public void testExternalApi() {

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

        // Step 2: Stub the method
```

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

        // Step 3: Inject into service
        MyService service = new MyService(mockApi);

        // Step 4: Assert output
        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();
 }
}
```

CODE :

```
package com.example;

import static org.mockito.Mockito.*;
import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.Test;
import org.mockito.Mockito;
```

```
public class MyServiceTest {

    @Test
    public void testExternalApi() {
        // Step 1: Create mock
        ExternalApi mockApi = Mockito.mock(ExternalApi.class);

        // Step 2: Stub the method
        when(mockApi.getData()).thenReturn("Mock Data");

        // Step 3: Inject into service
        MyService service = new MyService(mockApi);

        // Step 4: Assert output
        String result = service.fetchData();
        assertEquals("Mock Data", result);
    }

    @Test
    public void testVerifyInteraction() {
        // Step 1: Create mock
        ExternalApi mockApi = Mockito.mock(ExternalApi.class);

        // Step 2: Inject mock into service
        MyService service = new MyService(mockApi);

        // Step 3: Call the method
        service.fetchData();

        // Step 4: Verify interaction
```

```
    verify(mockApi).getData();
} }
```

OUTPUT: