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 4.13.2 test

3. Create a new test class in your project.

Procedure:

Step 1: Create a Java Maven Project.

Step 2: Add JUnit 4.13.2 Dependency.

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

Step 3: Create Your Main Class.

```
package com.example;
public class Calculator{
public int add(int a, int b) {
return a + b;
}
}
```

Step 4: Create Your JUnit Test Class.

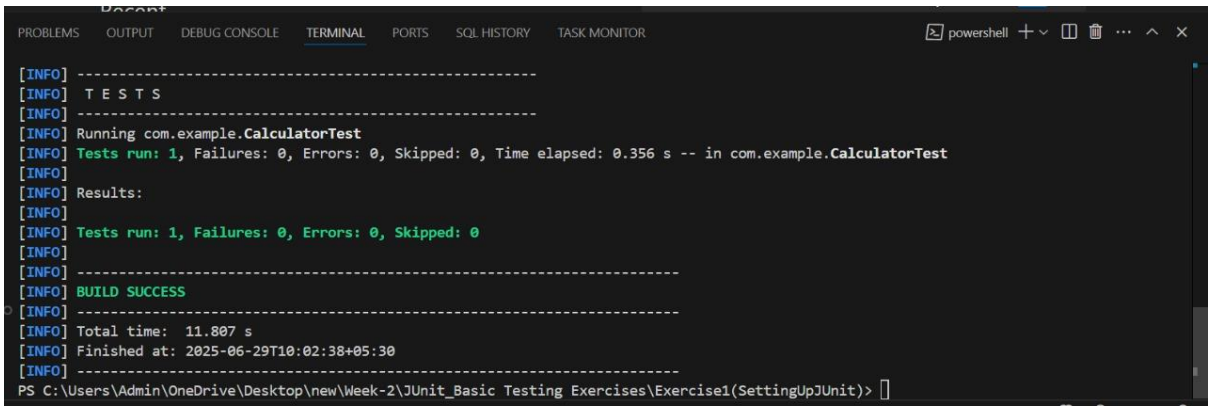
```
package com.example;
import static org.junit.Assert.assertEquals;
import org.junit.Test;
public class CalculatorTest {
@Test
public void testAdd() {
Calculator app = new Calculator();
int result = app.add(3, 7);
```

assertEquals(10, result);

}

}

Step 5: Run the Tests(Out Put)



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());

```
 }
}
```

Procedure:

Step 1: AssertionsTest.java

```
import static org.junit.Assert.*;
import org.junit.Test;
public class AssertionsTest {
@Test
    public void testAssertions() {
        // Assert equals
        assertEquals(5, 2 + 3);
 // Assert true
        assertTrue(5 > 3);
// Assert false
        assertFalse(5 < 3);  // fixed missing semicolon
// Assert null
        assertNull(null);
// Assert not null
        assertNotNull(new Object());
    }
}
```

Place this file under:

src/test/java/com/example/AssertionsTest.java

Step2: Add JUnit 4.13.2 Dependency

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

Step3: Run from VS Code

(Out Put)

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   SQL HISTORY   TASK MONITOR                    powershell
t 97 kB/s)
[INFO]
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running com.example.AssertionsTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.301 s -- in com.example.AssertionsTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  10.677 s
[INFO] Finished at: 2025-06-29T09:35:56+05:30
[INFO] ------------------------------------------------------------------------
PS C:\Users\Admin\OneDrive\Desktop\new\Week-2\JUnit_Basic Testing Exercises\Exercise3(Assertions_in_JUnit)>
```

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.

Procedure:

Step 1: Your Class Under Test (Calculator.java)

```
package com.example;

public class Calculator {

    public int add(int a, int b) {

        return a + b;

    }

}
```

Step 2: Test Class Using AAA Pattern with Setup/Teardown (CalculatorTest.java)

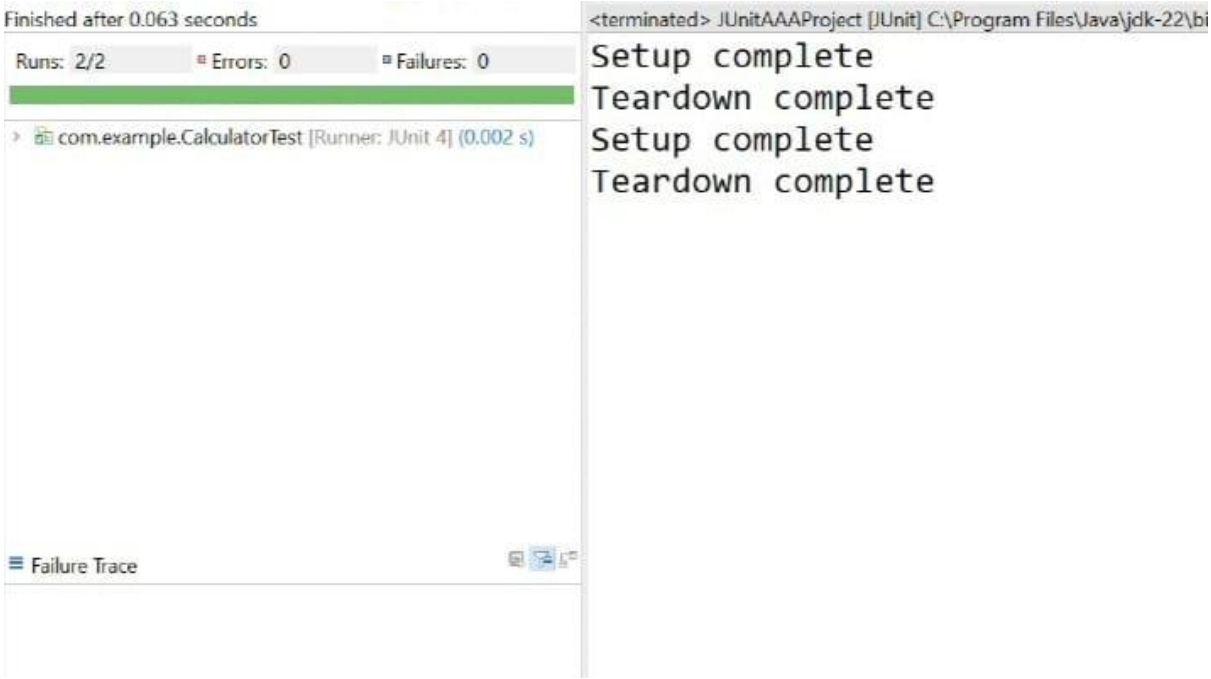
```
package com.example;


import static org.junit.Assert.*;

import org.junit.Before;

import org.junit.After;

import org.junit.Test;
```

```
public class CalculatorTest {
 private Calculator calculator;
@Before
   public void setUp() {
       calculator = new Calculator();
       System.out.println("Setting up...");
   }
 @After
   public void tearDown() {
       calculator = null;
       System.out.println("Tearing down...");
   }
@Test
   public void testAdd_PositiveNumbers() {
       int a = 3;
       int b = 5;
       int result = calculator.add(a, b);
       assertEquals(8, result);
   }
@Test
   public void testAdd_NegativeNumbers() {
       int a = -2;
       int b = -4;
       int result = calculator.add(a, b);
       assertEquals(-6, result);
   }
@Test
   public void testAdd_MixedNumbers() {
       int a = -3;
       int b = 6;
       int result = calculator.add(a, b);
       assertEquals(3, result);
```

```
  }
}
```

Step 3: Run the Tests in VS Code (Maven Project)

Out Put:



s