# **WEEK 2 J Unit Testing Exercise HANDS ON**

**SUBMITTED BY :-**

NAME – AYUSH SAHOO

SUPERSET ID – 6372243

COURSE – DIGITAL NURTURE 4.0 JAVA FSE

EMAIL – [ayush.sahoo003@gmail.com](mailto:ayush.sahoo003@gmail.com)

COLLEGE – C.V. RAMAN GLOBAL UNIVERSITY

**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>**

**3. Create a new test class in your project.**

**SOLUTION:-**

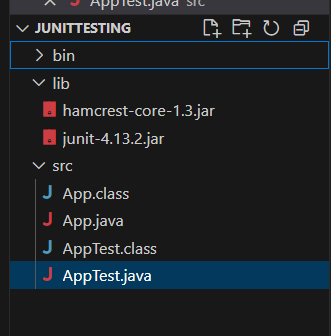
Step 1: Here I have used VS Code for representation (after downloading all the extensions file)

Add .jar files like

hamcrest-core-1.3.jar

junit-4.13.3.jar

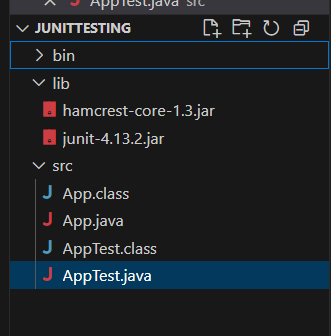
in lib folder

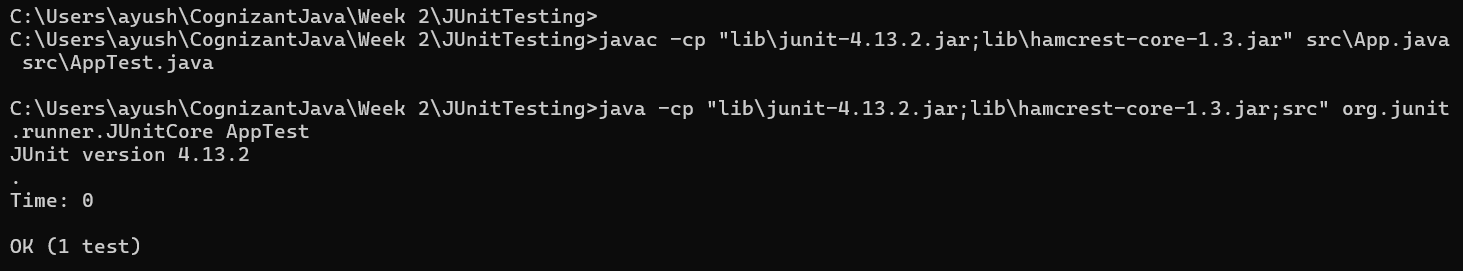


(All the folders have been created accordingly)

Step 2: Since I am not using maven I have done this dependency feature using cmd prompt

Jar file has already bin placed into “lib” folder in “JUnitTesting” folder

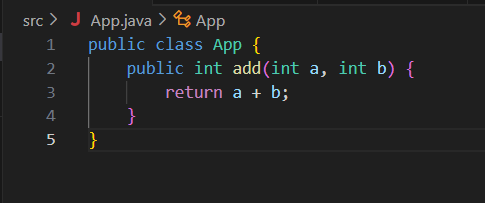


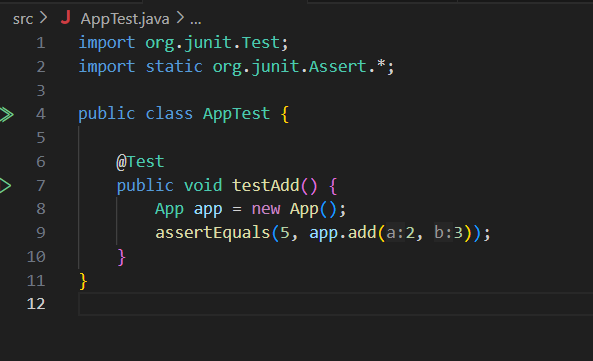


(Compiled using cmd prompt)

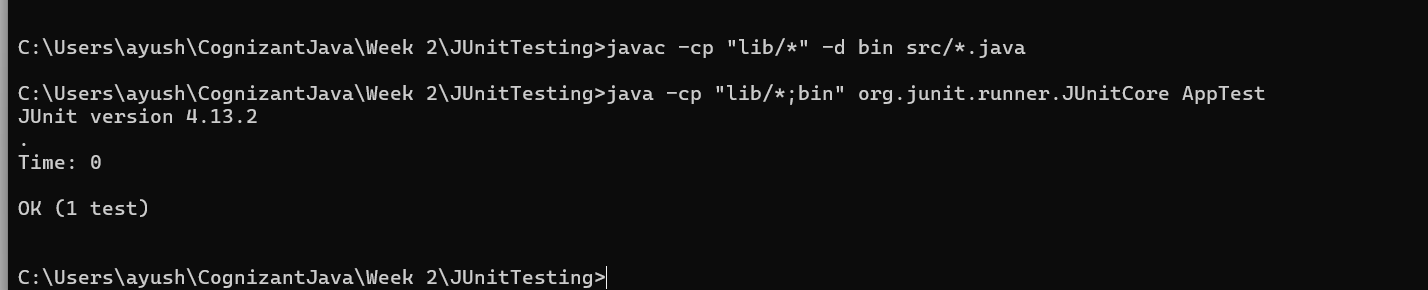
Step 3: Adding new test classes.

Test classes has been added by compiling it in cmd prompt after creating a java file.





(Here “App.java” and “AppTest.java” has been created)



(Here it has been compiled using cmd prompt)

**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);**

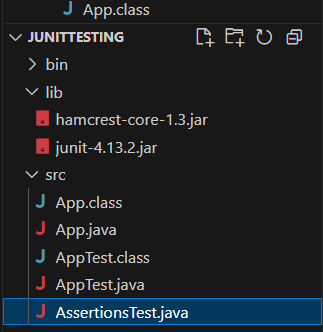
**// Assert not null**

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

**}**

**}**

**SOLUTION:-**

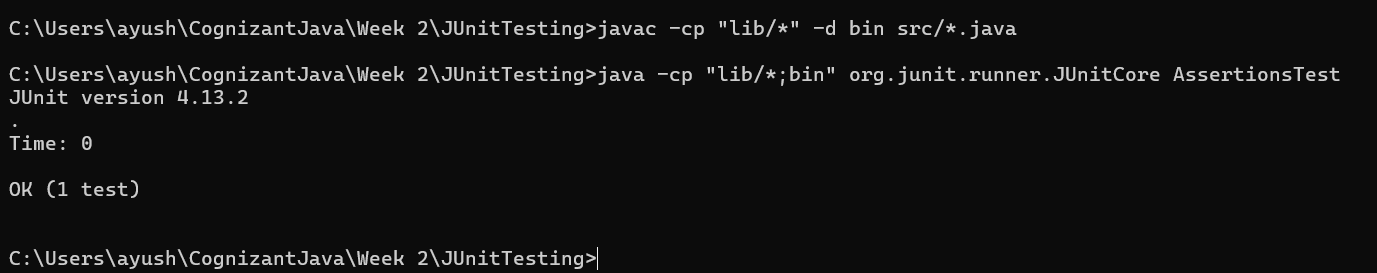
Step 1: Create a file name “Assertion.java” in src folder of “JUnitTesting”. ****

(Last java file is the “Assertion.java” file)

Step 2: Adding the code.

****

Step 3: Compile it using cmd with **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.**

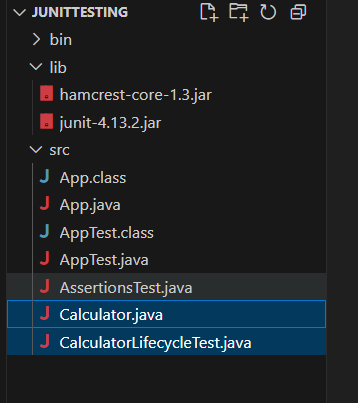
**Steps:**

**1. Write tests using the AAA pattern.**

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

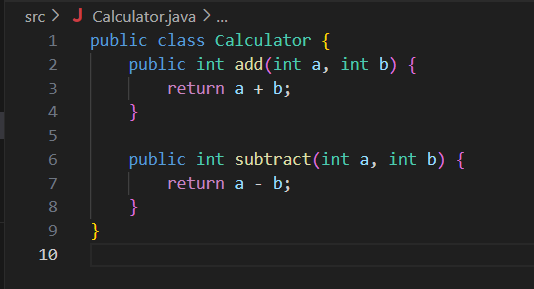
**SOLUTION:**

Step 1: Create a “Calculator.java” file into the src of same folder “JUnitTesting”.

****

(Highlighted part are the newly created one)

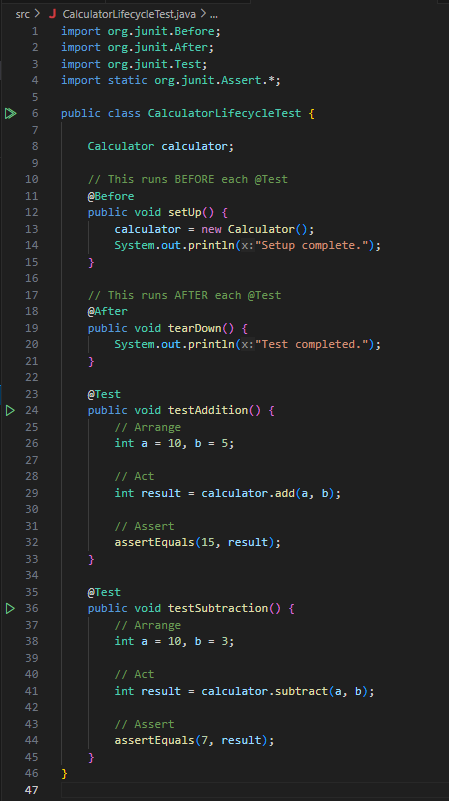
Step 2: Arrange the test using AAA pattern.



HERE:-

Each part matches AAA:

* **Arrange:** int a = 10, b = 5;
* **Act:** int result = calculator.add(a, b);
* **Assert:** assertEquals(15, result);

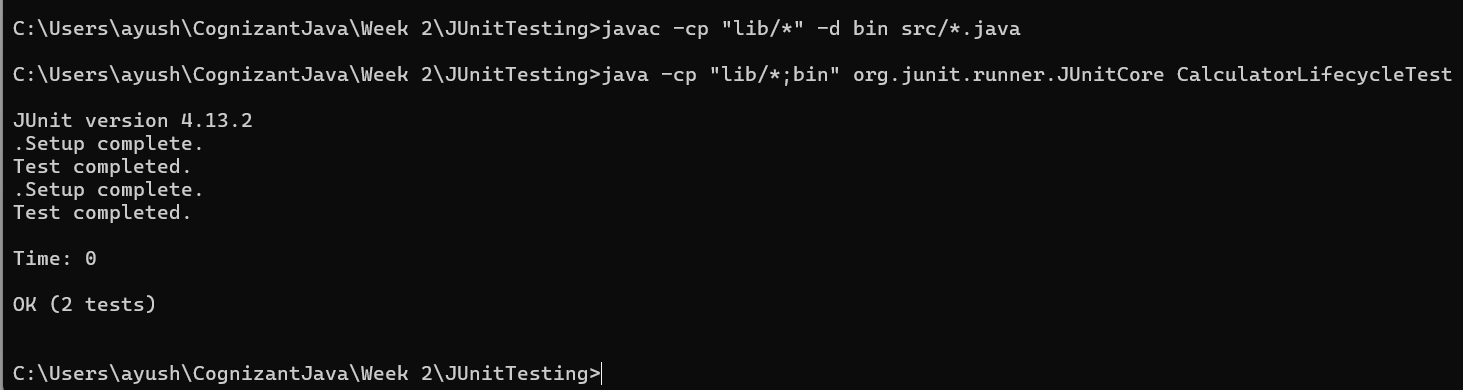


HERE:-

We have used @Before and @After for setup and teardown methods

| **Annotation** | **Purpose** |
| --- | --- |
| @Before | Runs **before each** test method |
| @After | Runs **after each** test method |
| @BeforeClass | Runs **once before all** test methods |
| @AfterClass | Runs **once after all** test methods |

Step 3: Compile in cmd and **OUTPUT.**



Visual Flow of @setup and @teardown

1. **setUp()** → Before test

2. **testAddition() or testSubtraction()** → Actual test

3. **tearDown()** → After test

**x----x----x**