Software Verification and Validation LAB (Testing)

Created	@May 16, 2025 7:28 PM
Created by	EMRAN AHMED EMON

We'll focus on the following **two tools:**

1. JUnit (used for unit testing in Java) and

2. Selenium (used for browser automation/testing).

🔷 STEP 1: Understand the Purpose of Each Tool

Tool	Purpose
JUnit	Automated testing framework for Java (tests individual units of code).
Selenium	Automates web browser actions (tests web applications end-to-end).

🔷 STEP 2: Setup Your Environment

✅ Install These (before your exam):

- **Java** (JDK 8+)
- **IDE**: IntelliJ IDEA / Eclipse (IntelliJ recommended for beginners)
- **JUnit**: Comes with most IDEs (JUnit 4 or 5)
- **Selenium**: Download Selenium JAR or use Maven/Gradle dependencies (if allowed)

🔷 STEP 3: Learn JUnit

☑️ What It Does

JUnit lets you **write and run tests** to make sure your Java methods work correctly.

☑️ JUnit 4 Basic Syntax:

```
java
CopyEdit
import org.junit.Test;
import static org.junit.Assert.assertEquals;

public class CalculatorTest {

    @Test
    public void testAddition() {
        int result = 2 + 3;
        assertEquals(5, result);
    }
}
```

✅ Key Concepts:

- `@Test`: Marks a method as a test method
- `assertEquals(expected, actual)`: Compares output
- No `main()` method required

✅ Practice:

- Write a simple class (e.g., Calculator with add, subtract)
- Write test methods to test those functions

🔷 STEP 4: Learn Selenium

☑️ What It Does

Selenium automates browser actions: open page, click button, enter text, etc.

☑️ Basic Example (Selenium with Java):

```
java
CopyEdit
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.By;

public class GoogleSearchTest {
    public static void main(String[] args) {
        System.setProperty("webdriver.chrome.driver", "path_to_chromedrive
r");
        WebDriver driver = new ChromeDriver();

        driver.get("https://www.google.com");
        driver.findElement(By.name("q")).sendKeys("JUnit");
        driver.findElement(By.name("q")).submit();

        System.out.println("Title: " + driver.getTitle());
        driver.quit();
    }
}
```

✅ Key Concepts:

- `WebDriver`: Controls the browser
- `findElement(By...)`: Locates elements on page
- `.sendKeys()`: Types into input field
- `.click()` / `.submit()`: Click or submit

✅ Practice:

- Open a site (e.g., Google)
- Search a keyword
- Close the browser

🔷 STEP 5: Possible Practical Tasks in Lab Exam

Topic	Task Example
JUnit	Write test cases for a class (e.g., Calculator, StringHelper)
Selenium	Automate login, search, or form submission on a given webpage

---

🔷 PART 1: JUNIT – Step-by-Step Setup & Practice

📌 Step 1: Create a JUnit Project in IntelliJ

1. **Open IntelliJ > New Project > Java Project**
2. Right-click on `src > New > Java Class` → Name it `Calculator`
3. Paste this example code:

```
java
CopyEdit
public class Calculator {
    public int add(int a, int b) {
        return a + b;
    }

    public int subtract(int a, int b) {
        return a - b;
    }
}
```

1. Now create a **test class**:
   - Right-click on class → Go to `Generate` (Alt + Insert) → Click on `Test`
   - Choose `JUnit4`
   - IntelliJ will prompt to add JUnit – **click "Fix" or "Download JUnit4"**
2. Add this test case:

```
java
CopyEdit
```

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

public class CalculatorTest {

    @Test
    public void testAdd() {
        Calculator calc = new Calculator();
        assertEquals(5, calc.add(2, 3));
    }

    @Test
    public void testSubtract() {
        Calculator calc = new Calculator();
        assertEquals(1, calc.subtract(3, 2));
    }
}
```

1. **Run the tests**: Right-click on the test class → `Run CalculatorTest`

🔁 Practice More

Try testing:

- Multiplication
- Division (add exception for divide-by-zero)

How to add JUnit Library in Project?

✅ Step 1: Add JUnit Library to Project

1. In IntelliJ, go to:

```
arduino
CopyEdit
File → Project Structure → Modules → Dependencies tab
```

2. Click ➕ (Add) → **Library** → **From Maven**
3. Search for:

```
makefile
CopyEdit
junit:junit:4.13.2
```

4. Click **OK** to add it. It will automatically download and include JUnit in your project.

✅ Step 2: Verify Your Imports

Make sure your test file (`CalculatorTest.java`) has these imports:

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

✅ Step 3: Create Your `Calculator` Class

Make sure this class exists and is correct:

```
java
CopyEdit
public class Calculator {
    public int add(int a, int b) {
        return a + b;
    }

    public int subtract(int a, int b) {
        return a - b;
    }
}
```

✅ Step 4: Fix Your `CalculatorTest` File

Your test file should look like this:

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

public class CalculatorTest {

    @Test
    public void testAdd() {
        Calculator calc = new Calculator();
        assertEquals(5, calc.add(2, 3));
    }

    @Test
    public void testSubtract() {
        Calculator calc = new Calculator();
        assertEquals(1, calc.subtract(3, 2));
    }
}
```

✅ Step 5: Run the Tests

Once the JUnit library is added and everything is saved:

- **Right-click on** `CalculatorTest.java` **→ Run** `CalculatorTest`
- OR click the **green play icon** next to the class or test method.

-)

📘 Simple Answers to Remember

Q: What is JUnit?

> A testing framework for Java used to write and run repeatable test cases.

Q: What does `@Test` do?

> It tells JUnit that this method is a test case.

Q: What is `assertEquals(expected, actual)`?

> It checks if the expected result equals the actual result. If not, the test fails.

✅ More JUnit Examples (Simple and Practical)

🧮 Example 1: Calculator Class (add, subtract, multiply, divide)

Calculator.java

```
java
CopyEdit
public class Calculator {
    public int add(int a, int b)     { return a + b; }
    public int subtract(int a, int b){ return a - b; }
    public int multiply(int a, int b){ return a * b; }
    public int divide(int a, int b)  {
        if (b == 0) throw new ArithmeticException("Cannot divide by zero");
        return a / b;
    }
}
```

CalculatorTest.java

```
java
CopyEdit
import org.junit.Test;
```

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

public class CalculatorTest {
    Calculator calc = new Calculator();

    @Test
    public void testAdd() {
        assertEquals(7, calc.add(4, 3));
    }

    @Test
    public void testSubtract() {
        assertEquals(2, calc.subtract(5, 3));
    }

    @Test
    public void testMultiply() {
        assertEquals(15, calc.multiply(3, 5));
    }

    @Test(expected = ArithmeticException.class)
    public void testDivideByZero() {
        calc.divide(10, 0);  // Should throw exception
    }

    @Test
    public void testDivide() {
        assertEquals(5, calc.divide(10, 2));
    }
}
```

📅 Example 2: Student Class (Simple Object Test)

Student.java

```
java
CopyEdit
public class Student {
```

```
    private String name;
    private int age;

    public Student(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public boolean isAdult() {
        return age >= 18;
    }

    public String getName() {
        return name;
    }
}
```

StudentTest.java

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

public class StudentTest {

    @Test
    public void testIsAdult() {
        Student s1 = new Student("Araf", 20);
        assertTrue(s1.isAdult());

        Student s2 = new Student("Nayeem", 15);
        assertFalse(s2.isAdult());
    }

    @Test
    public void testName() {
```

```
        Student s = new Student("Mehedi", 22);
        assertEquals("Mehedi", s.getName());
    }
}
```

💡 Concepts You Learned in These Examples:

- Asserting equality of values
- Testing boolean conditions (`assertTrue`, `assertFalse`)
- Testing for exceptions (`@Test(expected = ...)`)
- Testing object behavior

🔷 PART 2: SELENIUM – Step-by-Step Setup & Practice

📌 Step 1: Setup Selenium in IntelliJ

1. Go to Selenium Downloads
2. Download the latest **Selenium Java zip file**
3. Extract it → inside, you'll find JAR files (including `libs/`)
4. In IntelliJ:
    - Go to `File > Project Structure > Modules > Dependencies`
    - Click `+ > JARs or directories`
    - Add **all JARs** inside `selenium-java-x.x.x/` and `libs/`

📌 Step 2: Download ChromeDriver

1. Go to ChromeDriver Downloads
2. Download the version that matches your Chrome browser
3. Extract and keep the path (e.g., `C:\Users\YourName\Downloads\chromedriver.exe`)

📌 Step 3: Sample Selenium Test Code

```
java
CopyEdit
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

public class GoogleSearchTest {
    public static void main(String[] args) {
        // Set path to chromedriver.exe
        System.setProperty("webdriver.chrome.driver", "C:\\path\\to\\chromed
river.exe");

        // Launch browser
        WebDriver driver = new ChromeDriver();
        driver.get("https://www.google.com");

        // Interact with elements
        WebElement searchBox = driver.findElement(By.name("q"));
        searchBox.sendKeys("JUnit");
        searchBox.submit();

        // Wait and print title
        try {
            Thread.sleep(3000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("Page title is: " + driver.getTitle());
        driver.quit();
    }
}
```

✅ Step-by-Step Run Selenium in IntelliJ

✅ Step 1: Set Up Project with Selenium

1. **Create a new Java Project** in IntelliJ (or use your existing `Software_Testing_Lab-main`).
2. Right-click on `Basics` → `New → Java Class` → Name it `SeleniumTest`.
3. **Add Selenium dependency** to your `pom.xml` if you're using Maven:

```
xml
CopyEdit
<dependencies>
  <dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>4.19.1</version>
  </dependency>
</dependencies>
```

> 💡 If you're NOT using Maven, download the Selenium Java JARs and add them to your project:

- File → Project Structure → Modules → Dependencies → `+` JARs or Directories → select the `.jar` files.

✅ Step 2: Add ChromeDriver Path

1. Extracted `chromedriver.exe` → copy the path.

```
makefile
CopyEdit
Example: C:\Users\MOHAMMAD EMRAN\Downloads\chrome-driver\chromedriver-win64\chromedriver.exe
```

2. In your Java code, provide this path:

```
java
CopyEdit
System.setProperty("webdriver.chrome.driver", "C:\\Users\\MOHAMMAD E
MRAN\\Downloads\\chrome-driver\\chromedriver-win64\\chromedriver.ex
e");
```

✅ Step 3: Write Your First Selenium Test

```
java
CopyEdit
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class SeleniumTest {
    public static void main(String[] args) {
        // 1. Set the path to your chromedriver.exe
        System.setProperty("webdriver.chrome.driver", "C:\\Users\\MOHAMM
AD EMRAN\\Downloads\\chrome-driver\\chromedriver-win64\\chromedrive
r.exe");

        // 2. Create WebDriver instance
        WebDriver driver = new ChromeDriver();

        // 3. Open Google
        driver.get("https://www.google.com");

        // 4. Print the title
        System.out.println("Page title is: " + driver.getTitle());

        // 5. Close browser
        driver.quit();
    }
}
```

✅ Step 4: Run the Test

1. Right-click `SeleniumTest.java` → **Run**.
2. You should see Chrome open, load Google, and close.
3. Console will print something like:

```
csharp
CopyEdit
Page title is: Google
```

🔁 Next Selenium Labs:

- Search something on Google
- Automate login to a sample website
- Check if a web element exists
- Automate Login and Shopping Tests

✅ 1. Search Something on Google

Code

```
java
CopyEdit
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

public class GoogleSearchTest {
    public static void main(String[] args) {
        System.setProperty("webdriver.chrome.driver", "C:\\Users\\YourName
\\Downloads\\chrome-driver\\chromedriver.exe");
        WebDriver driver = new ChromeDriver();
```

```
        driver.get("https://www.google.com");

        WebElement searchBox = driver.findElement(By.name("q"));
        searchBox.sendKeys("OpenAI ChatGPT");
        searchBox.submit();

        try { Thread.sleep(3000); } catch (Exception e) {}

        driver.quit();
    }
}
```

✅ 2. Automate Login to a Sample Website

We'll use: https://www.saucedemo.com

Code

```
java
CopyEdit
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

public class LoginTest {
    public static void main(String[] args) {
        System.setProperty("webdriver.chrome.driver", "C:\\Users\\YourName
\\Downloads\\chrome-driver\\chromedriver.exe");
        WebDriver driver = new ChromeDriver();

        driver.get("https://www.saucedemo.com");

        driver.findElement(By.id("user-name")).sendKeys("standard_user");
        driver.findElement(By.id("password")).sendKeys("secret_sauce");
        driver.findElement(By.id("login-button")).click();
```

```
        try { Thread.sleep(3000); } catch (Exception e) {}

        driver.quit();
    }
}
```

✅ 3. Check if a Web Element Exists

Code

```
java
CopyEdit
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class CheckElementTest {
    public static void main(String[] args) {
        System.setProperty("webdriver.chrome.driver", "C:\\Users\\YourName
\\Downloads\\chrome-driver\\chromedriver.exe");
        WebDriver driver = new ChromeDriver();

        driver.get("https://www.saucedemo.com");

        boolean isPresent = driver.findElements(By.id("login-button")).size() >
0;
        System.out.println("Login button present? " + isPresent);

        driver.quit();
    }
}
```

✅ 4. Automate Login and Shopping Tests (SauceDemo)

Use the following credentials:

- ✅ `standard_user`
- ❌ `locked_out_user`
- Password: `secret_sauce`

You can put all these tests inside one class or separate them.

`testSuccessfulLogin()`

```
java
CopyEdit
driver.findElement(By.id("user-name")).sendKeys("standard_user");
driver.findElement(By.id("password")).sendKeys("secret_sauce");
driver.findElement(By.id("login-button")).click();
```

`testInvalidLogin()`

```
java
CopyEdit
driver.findElement(By.id("user-name")).sendKeys("wrong_user");
driver.findElement(By.id("password")).sendKeys("wrong_pass");
driver.findElement(By.id("login-button")).click();
```

`testLockedOutUser()`

```
java
CopyEdit
driver.findElement(By.id("user-name")).sendKeys("locked_out_user");
driver.findElement(By.id("password")).sendKeys("secret_sauce");
driver.findElement(By.id("login-button")).click();
```

testAddToCart()

```
java
CopyEdit
driver.findElement(By.id("add-to-cart-sauce-labs-backpack")).click();
driver.findElement(By.className("shopping_cart_link")).click();
```

testLogout()

```
java
CopyEdit
driver.findElement(By.id("react-burger-menu-btn")).click();
Thread.sleep(1000);
driver.findElement(By.id("logout_sidebar_link")).click();
```

testCheckoutProcess()

```
java
CopyEdit
driver.findElement(By.id("add-to-cart-sauce-labs-backpack")).click();
driver.findElement(By.className("shopping_cart_link")).click();
driver.findElement(By.id("checkout")).click();
driver.findElement(By.id("first-name")).sendKeys("John");
driver.findElement(By.id("last-name")).sendKeys("Doe");
driver.findElement(By.id("postal-code")).sendKeys("12345");
driver.findElement(By.id("continue")).click();
driver.findElement(By.id("finish")).click();
```

testIncompleteCheckoutInformation()

```
java
CopyEdit
driver.findElement(By.id("checkout")).click();
driver.findElement(By.id("continue")).click();
// Error message should appear
```

Short Trick to remember:

JUnit:

```
java
CopyEdit
@Test
public void testMethodName() {
    assertEquals(expected, actual);
    assertTrue(condition);
    assertFalse(condition);
}
```

Selenium:

```
java
CopyEdit
WebDriver driver = new ChromeDriver();
driver.get("URL");
driver.findElement(By.id("id")).sendKeys("text");
driver.findElement(By.name("name")).click();
driver.quit();
```

Explanation-01

SauceTest.java

Explanation:

📦 Imports (Top Section)

```
java
CopyEdit
import io.github.bonigarcia.wdm.WebDriverManager;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
```

What these do:

- `WebDriverManager`: Automatically sets up the correct ChromeDriver binary.
- `JUnit` annotations like `@Before`, `@After`, `@Test`: Help define setup, teardown, and test methods.
- `WebDriver`, `By`, `WebElement`, `ChromeDriver`: Selenium classes for browser control and element interaction.

🧱 Class Definition

```
java
CopyEdit
public class SauceTest {
```

- This is the main class that contains all your test methods for SauceDemo.

🌐 WebDriver Declaration

```
java
CopyEdit
private WebDriver driver;
```

- A **global driver variable** for the entire class.

- All test methods will use this `driver` to control the browser.

⚙️ Setup Method

```
java
CopyEdit
@Before
public void setUp() {
    WebDriverManager.chromedriver().setup();
    driver = new ChromeDriver();
    driver.get("https://www.saucedemo.com/");
}
```

- `@Before`: This method runs **before every test case**
- `WebDriverManager.chromedriver().setup()`: Downloads and configures the correct ChromeDriver.
- `driver = new ChromeDriver()`: Opens a new Chrome browser.
- `driver.get(...)`: Navigates to the SauceDemo login page.

✅ This ensures every test starts with a fresh browser session.

🔒 Test: Successful Login

```
java
CopyEdit
@Test
public void testSuccessfulLogin() {
    driver.findElement(By.id("user-name")).sendKeys("standard_user");
    driver.findElement(By.id("password")).sendKeys("secret_sauce");
    driver.findElement(By.id("login-button")).click();

    WebElement productTitle = driver.findElement(By.className("title"));
    assert productTitle.getText().equals("Products");
}
```

Steps explained:

1. Enters username and password.
2. Clicks login button.
3. Checks if the user lands on the product page by verifying the title is `"Products"`.

❌ Test: Invalid Login

```
java
CopyEdit
@Test
public void testInvalidLogin() {
    driver.findElement(By.id("user-name")).sendKeys("invalid_user");
    driver.findElement(By.id("password")).sendKeys("wrong_password");
    driver.findElement(By.id("login-button")).click();

    WebElement errorMsg = driver.findElement(By.cssSelector("h3[data-test
='error']"));
    assert errorMsg.isDisplayed();
}
```

Steps:

1. Enters wrong credentials.
2. Expects to see an error message.
3. Checks if the error message element is visible.

🔐 Test: Locked Out User

```
java
CopyEdit
@Test
public void testLockedOutUser() {
    driver.findElement(By.id("user-name")).sendKeys("locked_out_user");
    driver.findElement(By.id("password")).sendKeys("secret_sauce");
```

```
    driver.findElement(By.id("login-button")).click();

    WebElement errorMsg = driver.findElement(By.cssSelector("h3[data-test
='error']"));
    assert errorMsg.isDisplayed();
}
```

Steps:

1. Enters credentials for a known locked-out user.
2. Verifies the error message appears after login attempt.

🛒 Test: Add to Cart

```
java
CopyEdit
@Test
public void testAddToCart() {
    loginAsStandardUser();

    driver.findElement(By.id("add-to-cart-sauce-labs-backpack")).click();
    WebElement cartBadge = driver.findElement(By.className("shopping_c
art_badge"));
    assert cartBadge.getText().equals("1");
}
```

Steps:

1. Logs in using helper method `loginAsStandardUser()`.
2. Clicks “Add to Cart” for a backpack item.
3. Verifies that the cart shows `1` item.

🚪 Test: Logout

```
java
CopyEdit
```

```
@Test
public void testLogout() {
    loginAsStandardUser();

    driver.findElement(By.id("react-burger-menu-btn")).click();
    try {
        Thread.sleep(1000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    driver.findElement(By.id("logout_sidebar_link")).click();
    WebElement loginButton = driver.findElement(By.id("login-button"));
    assert loginButton.isDisplayed();
}
```

Steps:

1. Logs in.
2. Clicks the sidebar (hamburger) menu.
3. Waits 1 second (for animation to complete).
4. Clicks logout link.
5. Checks that login button is visible again.

🧾 Test: Checkout Process

```
java
CopyEdit
@Test
public void testCheckoutProcess() {
    loginAsStandardUser();

    driver.findElement(By.id("add-to-cart-sauce-labs-backpack")).click();
    driver.findElement(By.className("shopping_cart_link")).click();
    driver.findElement(By.id("checkout")).click();
    driver.findElement(By.id("first-name")).sendKeys("John");
    driver.findElement(By.id("last-name")).sendKeys("Doe");
```

```
    driver.findElement(By.id("postal-code")).sendKeys("12345");
    driver.findElement(By.id("continue")).click();
    driver.findElement(By.id("finish")).click();

    WebElement confirmation = driver.findElement(By.className("complete
-header"));
    assert confirmation.getText().equals("Thank you for your order!");
}
```

Steps:

1. Logs in and adds an item to cart.
2. Goes to checkout page.
3. Fills customer info.
4. Completes the checkout.
5. Verifies the success message.

⚠️ Test: Incomplete Checkout Info

```
java
CopyEdit
@Test
public void testIncompleteCheckoutInformation() {
    loginAsStandardUser();

    driver.findElement(By.id("add-to-cart-sauce-labs-backpack")).click();
    driver.findElement(By.className("shopping_cart_link")).click();
    driver.findElement(By.id("checkout")).click();
    driver.findElement(By.id("first-name")).sendKeys("John");
    driver.findElement(By.id("last-name")).sendKeys("");
    driver.findElement(By.id("postal-code")).sendKeys("12345");
    driver.findElement(By.id("continue")).click();

    WebElement errorMsg = driver.findElement(By.cssSelector("h3[data-test
='error']"));
    assert errorMsg.isDisplayed();
```

```
}
```

Steps:

1. Leaves the `last-name` field empty during checkout.
2. Expects an error message.
3. Verifies it's shown.

🔁 Helper Method

```
java
CopyEdit
private void loginAsStandardUser() {
    driver.get("https://www.saucedemo.com/");
    driver.findElement(By.id("user-name")).sendKeys("standard_user");
    driver.findElement(By.id("password")).sendKeys("secret_sauce");
    driver.findElement(By.id("login-button")).click();
}
```

- DRY principle: **Don’t Repeat Yourself**
- This method logs in the standard user, used in multiple test cases.

❌ Teardown Method

```
java
CopyEdit
@After
public void tearDown() {
    if (driver != null) {
        driver.quit();
    }
}
```

- Runs **after each test**

- Closes the browser properly to release memory.

✅ Summary

Section	Purpose
`@Before`	Sets up browser and opens site
`@Test` methods	Automate real user actions (login, cart, checkout, etc.)
`@After`	Closes browser after test
`WebDriverManager`	Automatically handles browser driver download

Explanation-02

CalculatorTest.java

✅ Full Step-by-Step Explanation of `CalculatorTest.java`

📦 1. Import Statements

```
java
CopyEdit
import org.junit.Test;
import static org.junit.Assert.assertEquals;
```

🔍 What They Do:

- `import org.junit.Test;`

  → Lets you use the `@Test` annotation to mark a method as a test method.

- `import static org.junit.Assert.assertEquals;`

  → Imports the `assertEquals` method statically so you can directly call it to **compare expected and actual results**.

🧮 2. The Calculator Class (Assumed External)

There's no `Calculator` class code inside this file. That means `CalculatorTest.java` is **only the test class**, and it's testing methods from a separate class named `Calculator`.

We assume the `Calculator` class has methods like:

- `add(int a, int b)`
- `subtract(int a, int b)`
- `multiply(int a, int b)`
- `divide(int a, int b)`

🧪 3. Test Class Declaration

```
java
CopyEdit
public class CalculatorTest {
```

This defines the test class. The name `CalculatorTest` follows the convention:

- It's testing the `Calculator` class.
- Ends with `Test` to signal it contains test cases.

✅ 4. Test Case 1: testAdd()

```
java
CopyEdit
@Test
public void testAdd() {
    Calculator calculator = new Calculator();
    int result = calculator.add(10, 5);
    assertEquals(15, result);
}
```

Step-by-step:

1. `@Test` tells JUnit this is a test method.
2. A new object of `Calculator` is created.
3. `calculator.add(10, 5)` is called and returns `15`.
4. `assertEquals(15, result)` checks if the result is actually `15`.
   - ✅ If yes, test passes.
   - ❌ If no, test fails.

➖ 5. Test Case 2: testSubtract()

```
java
CopyEdit
@Test
public void testSubtract() {
    Calculator calculator = new Calculator();
    int result = calculator.subtract(10, 5);
    assertEquals(5, result);
}
```

- Calls `subtract(10, 5)` → should return `5`.
- Asserts it equals `5`.

✖️ 6. Test Case 3: testMultiply()

```
java
CopyEdit
@Test
public void testMultiply() {
    Calculator calculator = new Calculator();
    int result = calculator.multiply(10, 5);
    assertEquals(50, result);
}
```

- Calls `multiply(10, 5)` → should return `50`.
- Asserts it equals `50`.

➗ 7. Test Case 4: testDivide()

```
java
CopyEdit
@Test
public void testDivide() {
    Calculator calculator = new Calculator();
    int result = calculator.divide(10, 5);
    assertEquals(2, result);
}
```

- Calls `divide(10, 5)` → should return `2`.
- Asserts it equals `2`.

🔁 Summary Table

Test Method	Calculator Method Called	Inputs	Expected Output	assertEquals() Value
`testAdd()`	`add(10, 5)`	10, 5	15	`assertEquals(15, result)`
`testSubtract()`	`subtract(10, 5)`	10, 5	5	`assertEquals(5, result)`
`testMultiply()`	`multiply(10, 5)`	10, 5	50	`assertEquals(50, result)`
`testDivide()`	`divide(10, 5)`	10, 5	2	`assertEquals(2, result)`

✅ What You're Learning Here:

- **Unit testing**: Testing one method at a time.
- **Assertions**: Used to compare expected and actual values.
- **JUnit Framework**: Commonly used in Java for test automation.