NATIONAL INSTITUTE OF TECHNOLOGY, ROURKELA

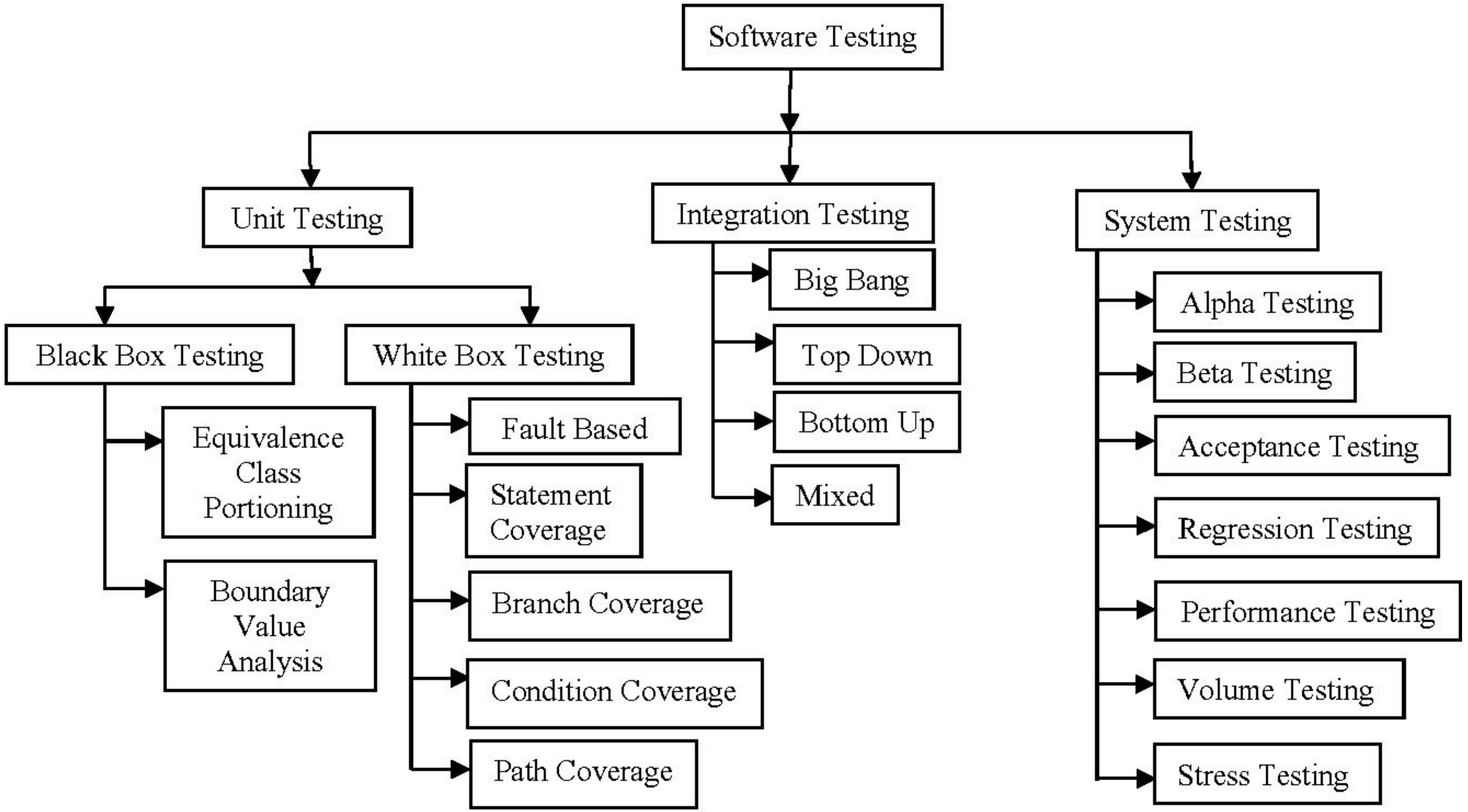
SOFTWARE TESTING LABORATORY (JUNIT)

BISHWAJIT PRASAD GOND

PROF. DURGA PRASAD MOHAPATRA
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

TABLE OF CONTENTS

OVERVIEW OF SOFTWARE TESTING:

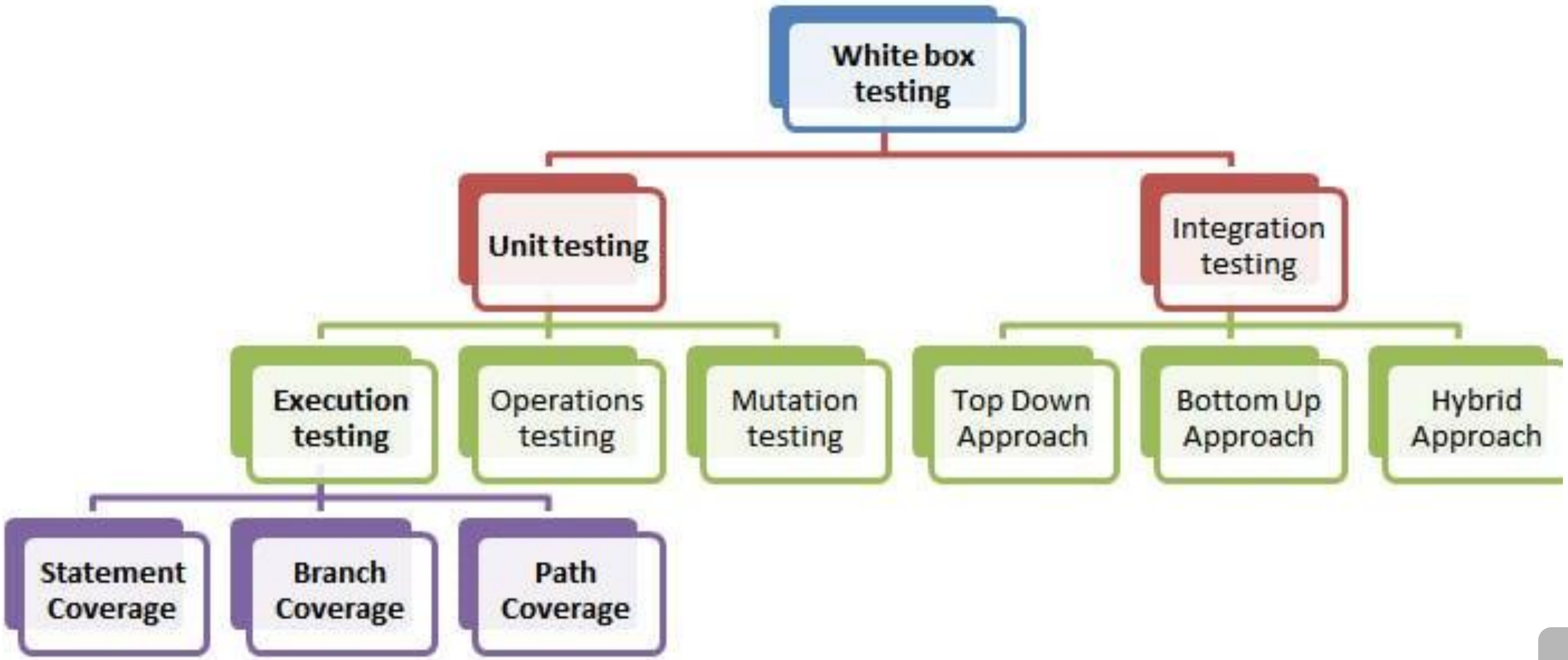


DIFFERENCE BETWEEN WHITE AND BLACK BOX TESTING:

Black-box testing is a testing technique in which the internal structure /design / implementation of the item being tested is not known to the tester. The tester focuses on the functionality of the item being tested and how it responds to inputs.

White-box testing is a testing technique in which the internal structure /design / implementation of the item being tested is known to the tester. The tester uses this knowledge to design tests that will exercise all possible paths through the code.

Types of White Box Testing



UNIT TESTING:

Unit testing is a software testing technique where individual units or components of a software application are tested in isolation from the rest of the system.

- The goal of unit testing is to verify that each unit of the software performs as designed.
- A "unit" in this context refers to the smallest testable part of a software application, often an individual function, method, or procedure.
- Unit testing often involves white-box testing, where the tester has knowledge of the internal workings of the code being tested. This allows for targeted testing of specific paths and conditions within the code.

POPULAR UNIT TESTING TOOLS

Popular unit testing frameworks for Java include JUnit, TestNG, and for other languages, there are frameworks like

- NUnit (for .NET),
- Pytest, PyUnit (for Python),
- and Mocha (for JavaScript).

About JUnit

Developers:

- Kent Beck
- Erich Gamma
- David Saff
- Kris Vasudevan

Stable Release:

- Version: 5.10.0
- Release Date: July 23, 2023 (6 months ago as of January 25, 2024)

Repository:

- https://github.com/junitteam/junit5.git

Written in:

- Java

Operating System:

- Crossplatform

Type:

- Unit testing tool

License:

- Eclipse Public License 2.0 (relicensed previously)

Website:

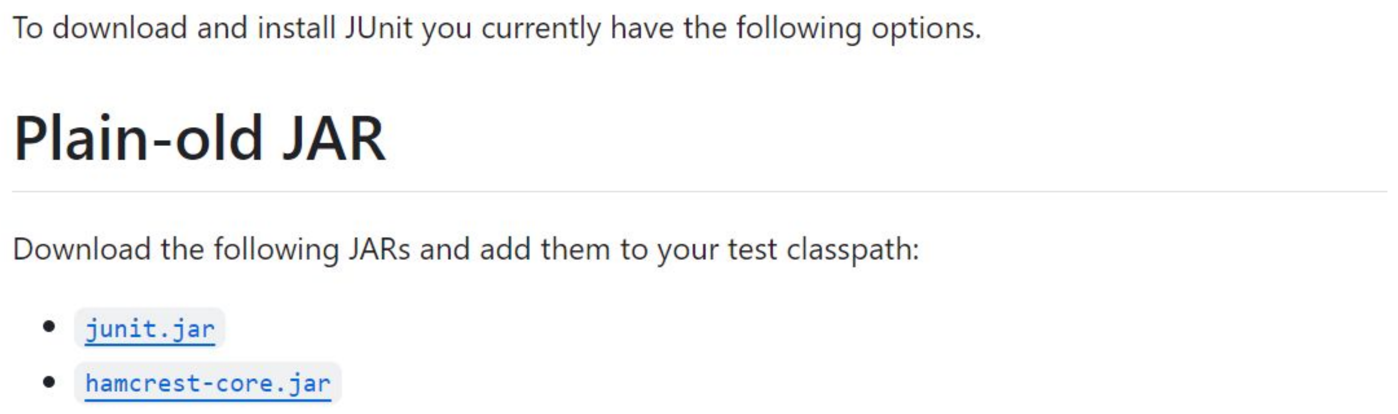
- https://junit.org

JUNIT:

JUNIT IS A UNIT TESTING FRAMEWORK FOR JAVA. IT IS NEITHER BLACK-BOX NOR WHITE-BOX TESTING, BUT RATHER A TOOL THAT CAN BE USED FOR BOTH.

SYSTEM/SOFTWARE REQUIREMENT FOR JUNIT

- OS Windows
- IDE Eclipse with JDK jdk-8u202-windows-i586.exe and JRE jre-8u202-windows-i586.exe
- JAR file ***https://github.com/junit-team/junit4/wiki/Download-and-Install***

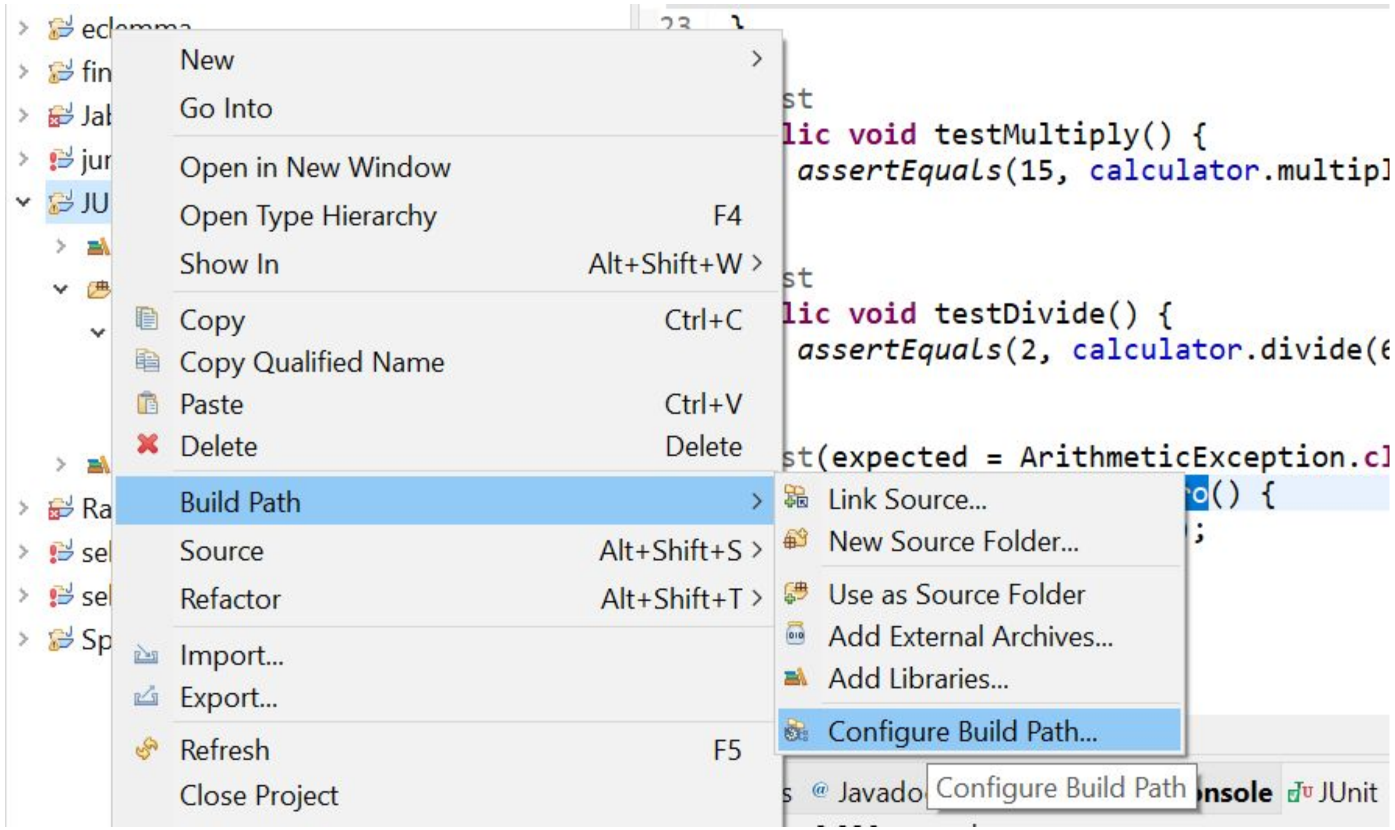

To download and install JUnit you currently have the following options.

Plain-old JAR

Download the following JARs and add them to your test classpath:

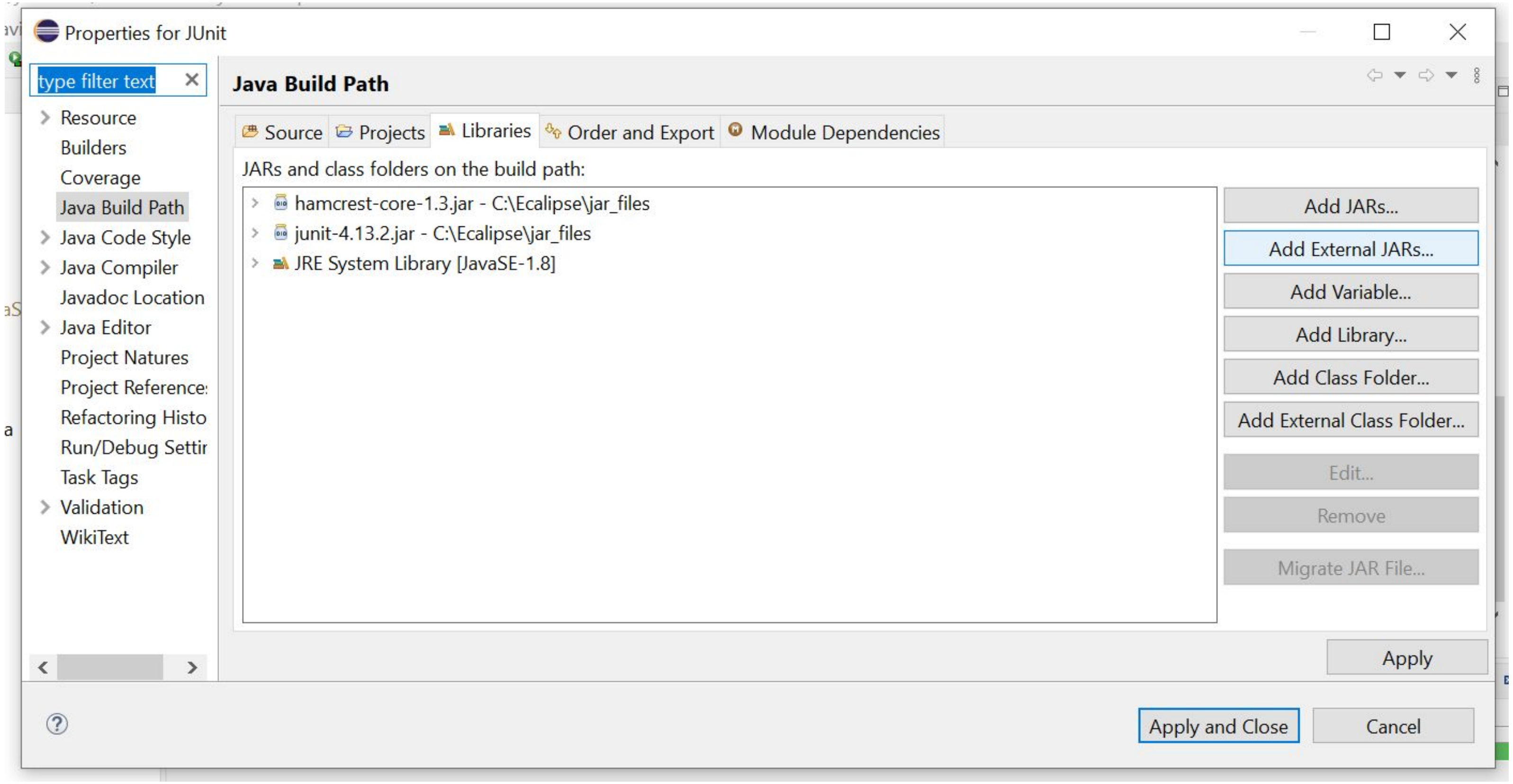
- `junit.jar`
- `hamcrest-core.jar`

JUNIT INSTALLATION



```
}

st
lic void testMultiply() {
 assertEquals(15, calculator.multipl

st
lic void testDivide() {
 assertEquals(2, calculator.divide(6

st(expected = ArithmeticException.cl
o() {
;
```

JUNIT INSTALLATION



Properties for JUnit

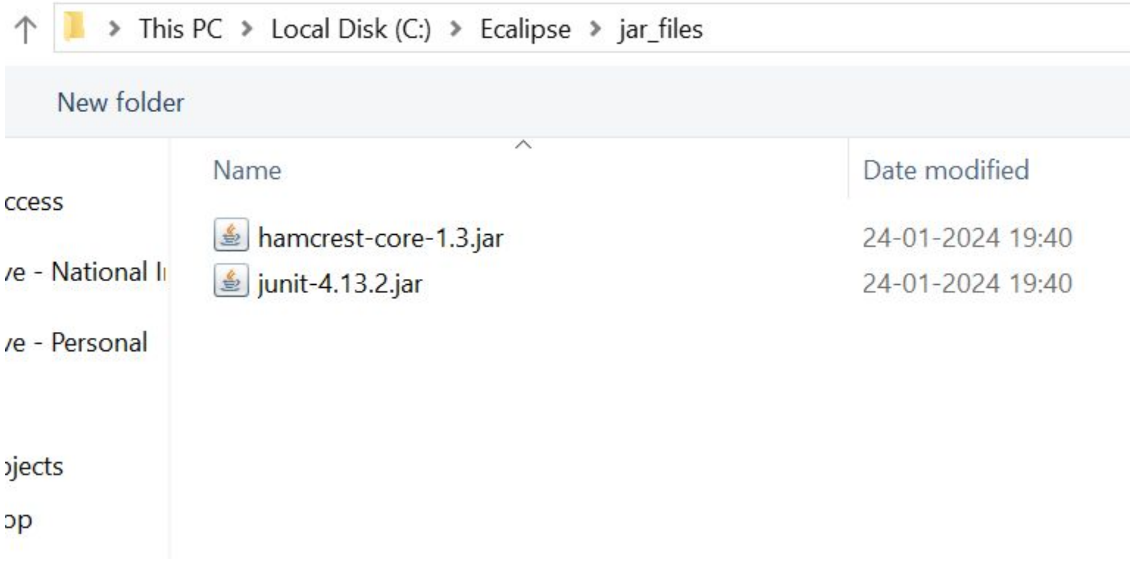
Java Build Path

JARs and class folders on the build path:

- hamcrest-core-1.3.jar - C:\Ecalipse\jar_files
- junit-4.13.2.jar - C:\Ecalipse\jar_files
- JRE System Library [JavaSE-1.8]

JUNIT INSTALLATION

Select the two JAR files from ADD External JARs and click APPLY


This PC > Local Disk (C:) > Ecalipse > jar_files

New folder

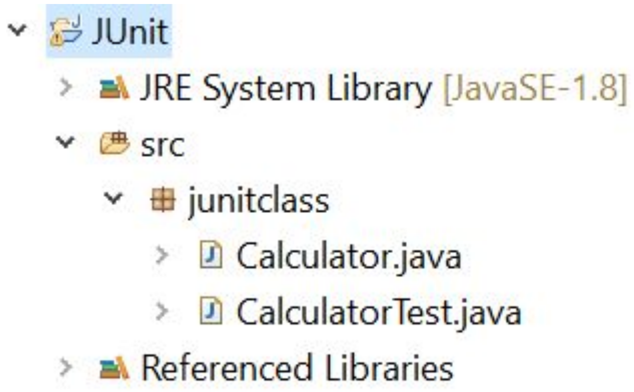
ccess
/e - National I
/e - Personal
ojects
op

Name	Date modified
hamcrest-core-1.3.jar	24-01-2024 19:40
junit-4.13.2.jar	24-01-2024 19:40

ECLIPSE IDE

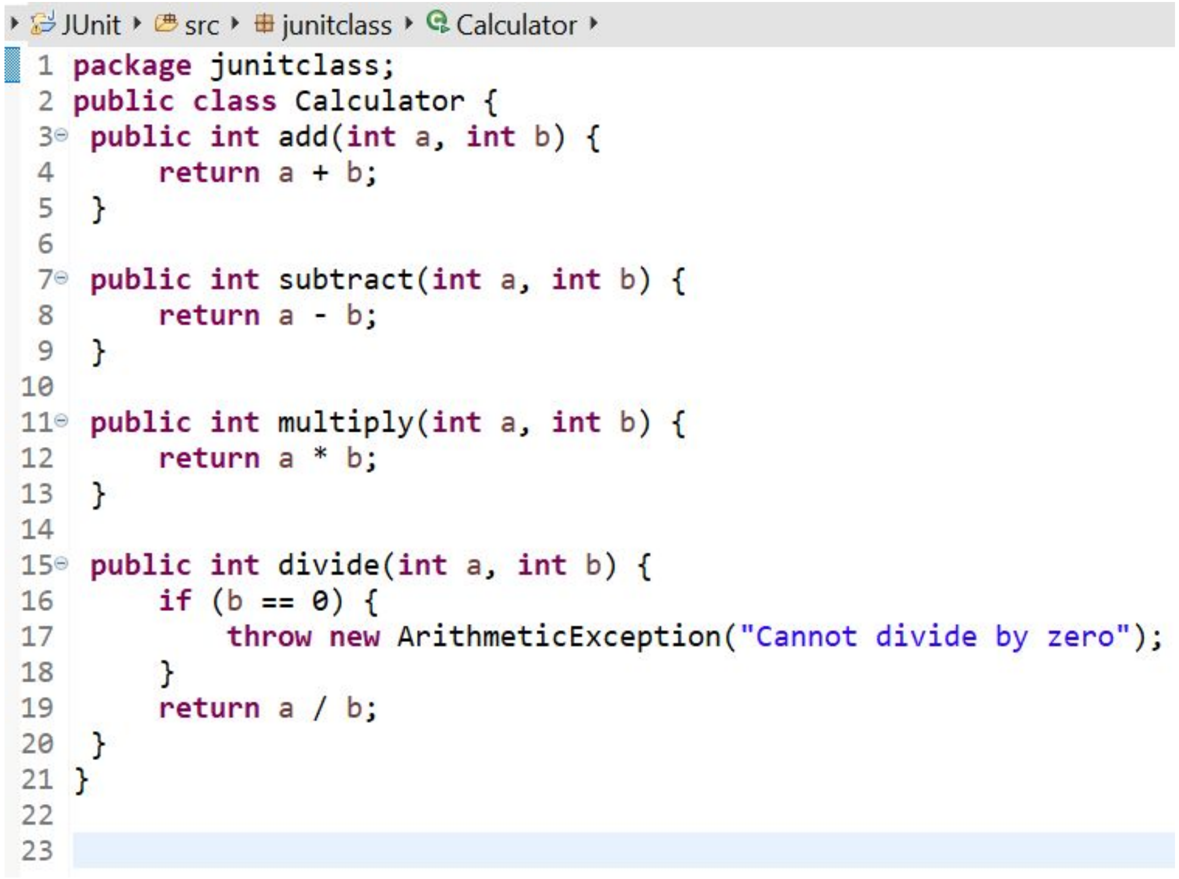
Make New project Name JUnit

- Create new package
- Create class file name ABC.java followed by ABCTest.java for storing test case.



```
JUnit
  JRE System Library [JavaSE-1.8]
  src
    junitclass
      Calculator.java
      CalculatorTest.java
  Referenced Libraries
```

Calculator.java



```
package junitclass;
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;
 }
}
```

Junit Libraries:

```
import static org.junit.Assert.assertEquals;
import org.junit.Before;
import org.junit.Test;
```

1. @Before Annotation:

- The `@Before` annotation is used to designate a method that should be executed before each test method annotated with `@Test`. It is often used for setting up common resources or initializing objects that are needed across multiple test cases.
- In the example, the `setUp()` method is annotated with `@Before`. This method creates an instance of the `Calculator` class and assigns it to the `calculator` field. This ensures that each test method starts with a fresh and consistent state.

2. @Test Annotation:

- The `@Test` annotation is used to indicate that the annotated method is a test method. When JUnit runs, it identifies methods annotated with `@Test` and executes them as test cases.
- In the example, methods like `testAdd()`, `testSubtract()`, `testMultiply()`, and `testDivide()` are marked with `@Test`. Each of these methods represents a specific test case for the corresponding operation in the `Calculator` class.
- The `assertEquals` method inside these test methods is used to verify that the actual result matches the expected result. If the assertion fails, the test case fails.

Other Annotations:

Annotation	Description	Example Usage
@Test	Denotes a method as a test method.	java @Test public void myTestMethod() { /* Test logic goes here */ }
@Before	Executed before each test method. Used for setup activities.	java @Before public void setUp() { /* Setup logic goes here */ }
@After	Executed after each test method. Used for cleanup activities.	java @After public void tearDown() { /* Cleanup logic goes here */ }
@BeforeClass	Executed once before any test method in the class. Used for one-time setup.	java @BeforeClass public static void setUpClass() { /* One-time setup logic goes here */ }

Other Annotations:

Annotation	Description	Example Usage
@AfterClass	Executed once after all test methods in the class. Used for one-time cleanup.	java @AfterClass public static void tearDownClass() { /* One-time cleanup logic goes here */ }
@Ignore	Ignores a test method. Useful when you want to temporarily disable a test.	java @Ignore @Test public void ignoredTest() { /* This test will be ignored */ }

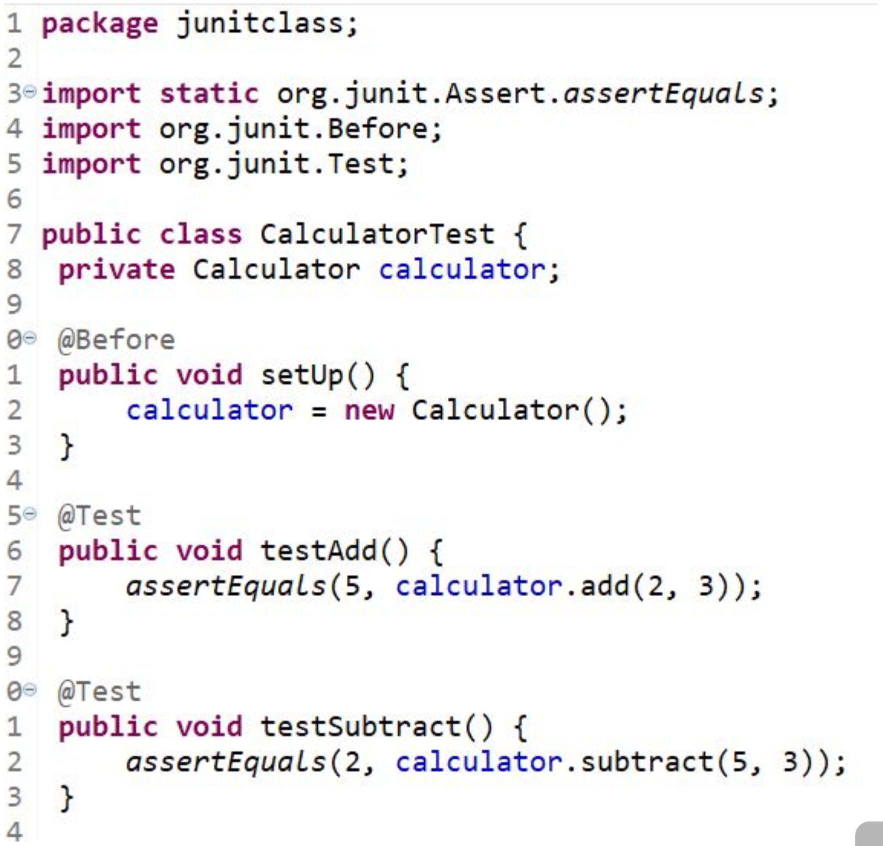
CalculatorTest.java

Test Annotation (`@Test`):

- Marks a method as a test method.
- Syntax: `@Test`
- Example: `@Test public void testAdd() { … }`

Before Annotation (`@Before`):

- Marks a method that should be run before each test method.
- Syntax: `@Before`
- Example: `@Before public void setUp() { … }`



```
package junitclass;

import static org.junit.Assert.assertEquals;
import org.junit.Before;
import org.junit.Test;

public class CalculatorTest {
 private Calculator calculator;

 @Before
 public void setUp() {
     calculator = new Calculator();
 }

 @Test
 public void testAdd() {
     assertEquals(5, calculator.add(2, 3));
 }

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

```

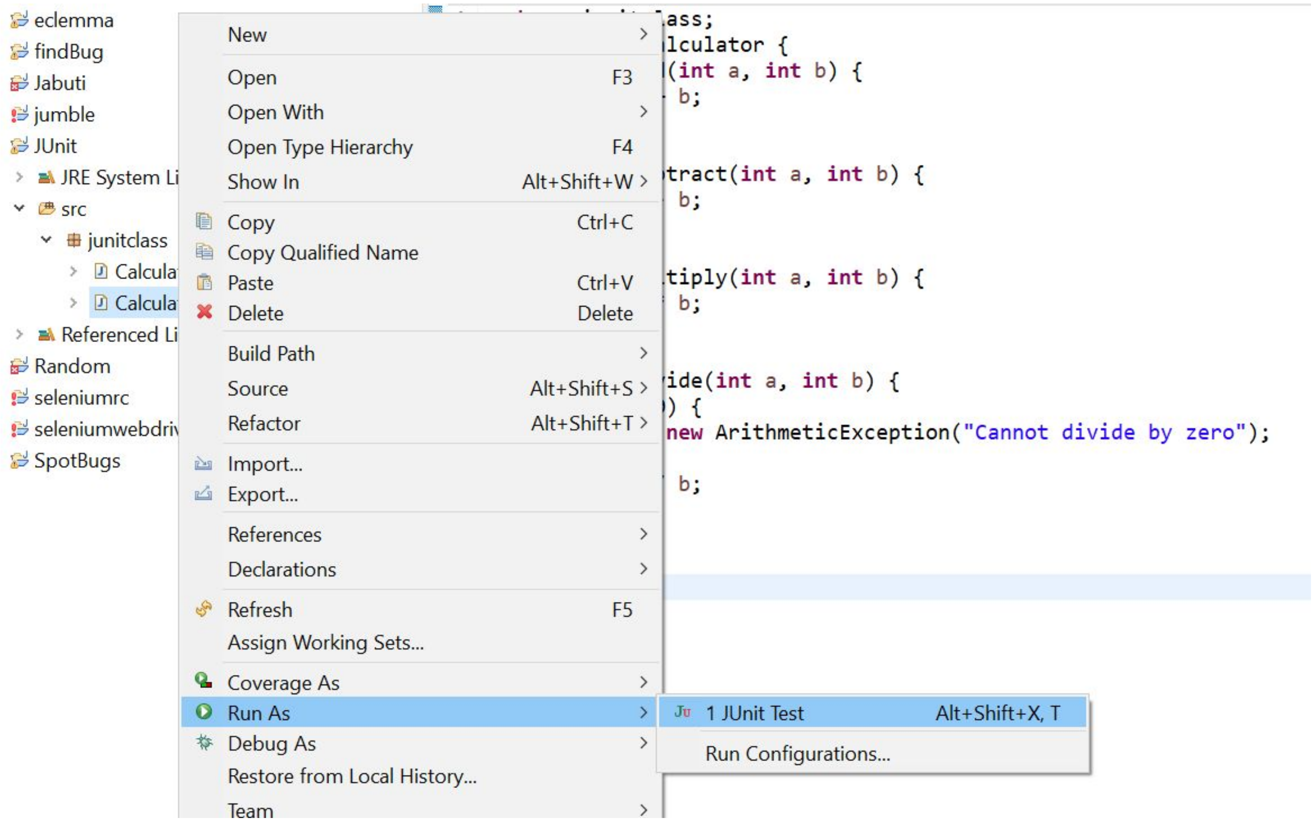
Different Type of Assert in JUnit

Assertion	Description	Example
assertEquals(expected, actual)	Checks if the expected value is equal to the actual value.	assertEquals(5, result);
assertTrue(condition)	Checks if the given condition is true.	assertTrue(x > 0);
assertFalse(condition)	Checks if the given condition is false.	assertFalse(list.isEmpty());
assertNull(object)	Checks if the given object is null.	assertNull(obj);

Different Type of Assert in JUnit

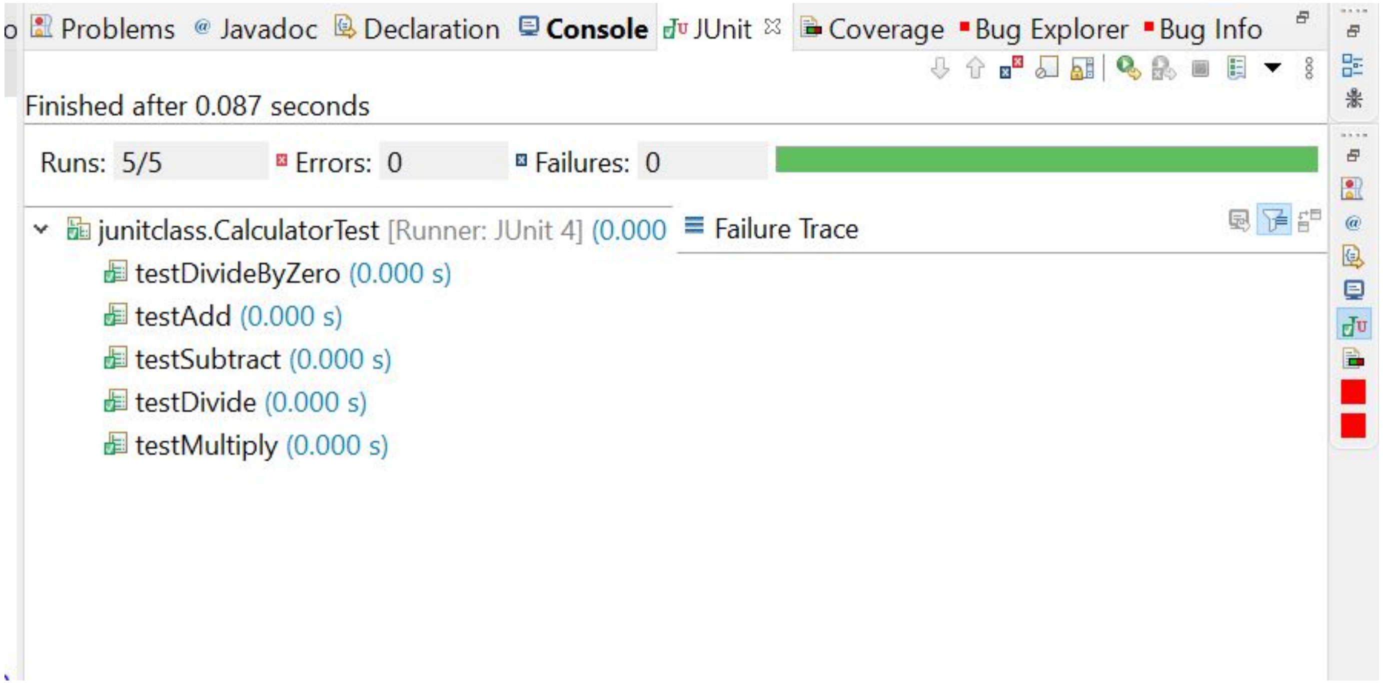
Assertion	Description	Example
assertNotNull(object)	Checks if the given object is not null.	assertNotNull(str);
assertSame(expected, actual)	Checks if the expected and actual objects refer to the same object in memory.	assertSame(obj1, obj2);
assertNotSame(expected, actual)	Checks if the expected and actual objects do not refer to the same object in memory.	assertNotSame(obj1, obj2);
assertArrayEquals(expectedArray, actualArray)	Checks if the expected and actual arrays are equal.	assertArrayEquals(expected, actual);

How To RUN



```
ass;
lculator {
(int a, int b) {
 b;

tract(int a, int b) {
 b;

tiply(int a, int b) {
 b;

ide(int a, int b) {
) {
new ArithmeticException("Cannot divide by zero");

 b;
```

After Running



Finished after 0.087 seconds

Runs: 5/5 Errors: 0 Failures: 0

```
junitclass.CalculatorTest [Runner: JUnit 4] (0.000
    testDivideByZero (0.000 s)
    testAdd (0.000 s)
    testSubtract (0.000 s)
    testDivide (0.000 s)
    testMultiply (0.000 s)
```

ANY QUESTIONS??