**1.Unit Testing**

**2.Peer Testing.**

**3.Junit**

**3.1. Junit Architecture.**

**3.2. Three-Main component**

**3. 3.Junit5-Jupiter Library**

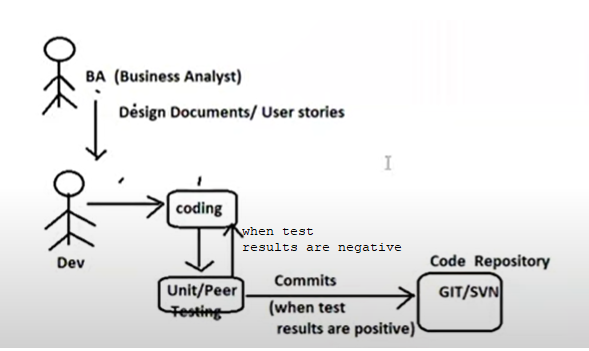
**3.4. Junit with Eclipse IDE**

**3. 5.Junit with Maveen Tool In Eclipse IDE**

1.Unit Testing: - The test done by programmer on own piece of code is called unit testing.

To Developer , one task is given . The task is said to be “Unit”.

2. peer Testing :- The unit test done on 1 programmer’s code /task by his coleague programmer is called peer testing.



The programmer does only following types of testings:

1. Unit Testing
2. Peer Testing
3. Integration Testing:- Usually, The TL does integration testing. In rare cases, programmer does integration testing under supervision of TL(Team Leader).

The Unit Testing can be done in two ways:

1. Manual Unit Testing.
2. Automated Unit Testing.

Limitation of Maual Unit Testing:

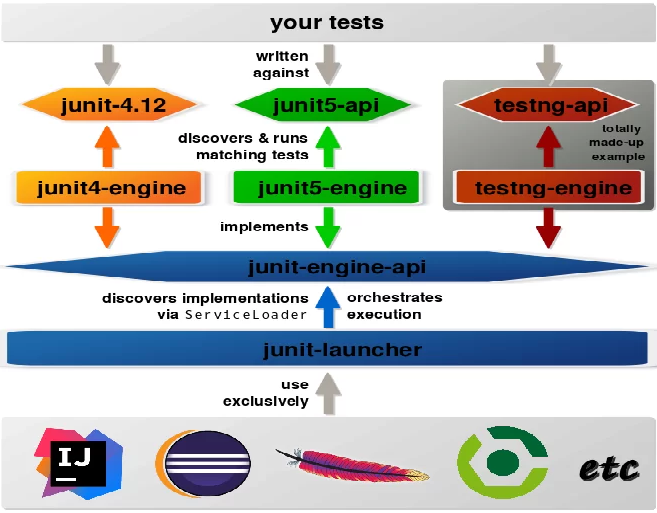
1. Test Regression(Repeat Tests) is very complex.
2. Presenting test plans/test cases to TL or superior is complex process.
3. Writing test report manually is complex process(Excel sheet report)

To overcome these problems, take support unit testing automation tools like JUNIT, HttpUnit(for web applications),mockito, TestNG …etc.

Note:- For HttpUnit,Mockito,TestNG , jUnit is the base tool. All tools are java based unit testing tools.

**JUNIT**

**1.Junit Architecture:**



* Junit contains 3 runtime libraries:

a.JunitJupitor:: Junit5 library.

b. junitVintage::junit3 and junit4 version libraries.

c. Junit Integration :: To allow Junit integration with estNg,Mockito and etc..

1. Junit-Launcher:- It uses the service loader to discover test engine implementations and to start their execution. It provides an API to IDEs ,tools(Mockito,TestNG…etc) so they can get junit engine . Ex: by launching individual tests and showing their results.
2. Your tests:- These test-case classes written by developer.
3. Junit-platform-engine: This engine does following activities

i.loading the test-case class

ii. Creating object object for test-case class

iii. Calling the test-methods

iv. generating test-reports

…etc.

Suppose programmer writes the test-case class using junit-4 library when junit-platform-engine uses junit-vintage-enginee to run the test-case class.

Programmer writes the test-case class using junit-5 libaryary when junit-platform-engine uses junit-jupiter-engine to run the test-case class.

Programmer write the test-case class using TestNg library,Mockito tool library when junit-platform engine uses test-engine to run the test-case class.

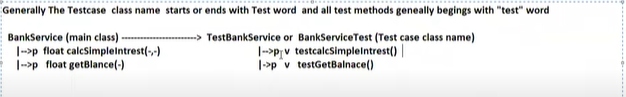
2. We shoul know the 3 main components while working with junit.

**a.Main class/service class** :: This class to be tested. Single unit contains 1 or more than 1 service classes.

**b.Test case class**:: This class contains 1 or more than one test methods.

**GuideLine**-1/Naming convention: Test case class name starts or ends with word “Test”.

**GuideLine**-2/Naming Convention: Test case class method starts with word “test”. Their return type should be “void”.



**c.Test suite class::** Each test case class generates one test report. But if want to combine all test classes reports in singe one then we take support of test suite class.

**3.Junit5-Jupiter Library:-** This api gives

1. Annotations.
2. Assertions API:- This api gives assertXXX(-) methods to match expected results with actual results and to generate the test reports.

All these methods are static methods.

We use them in development of test-case classes and test-case class methods.



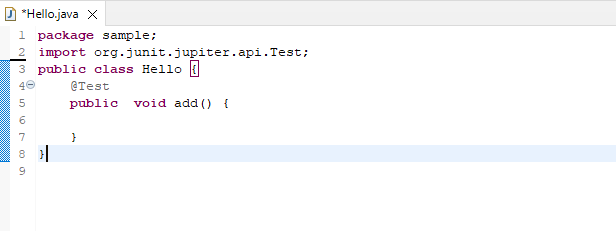
**4.JUNIT with ECLIPSE IDE:**

* Eclipse IDE gives built-in library for JUNIT.

Step1:- create a java project.

Step2:- create class with any name and with one method.

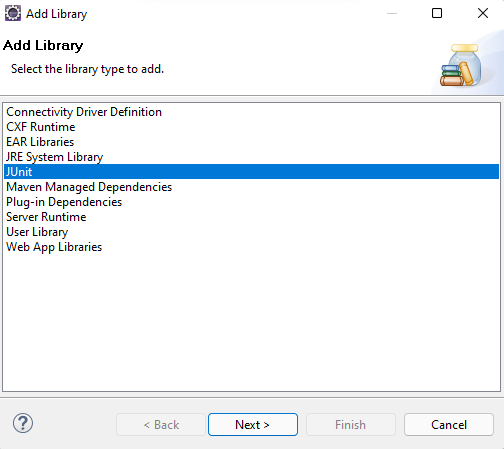
Place @Test annotation on method header

.

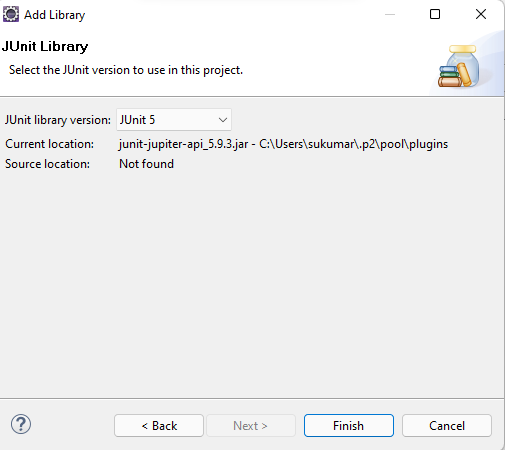
Step3:- Add Junit library to project.

3.1. select project🡪 buildpath 🡪configure buildpath

3.2 . press on “Add Library”. Button.

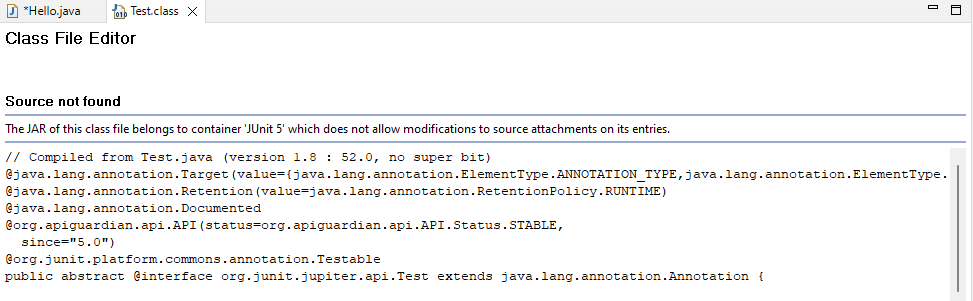


3. click the next



4. click finish button.

Problem: place cursor at @Test annotation and press F3 button , we won’t get any source code of Test.class



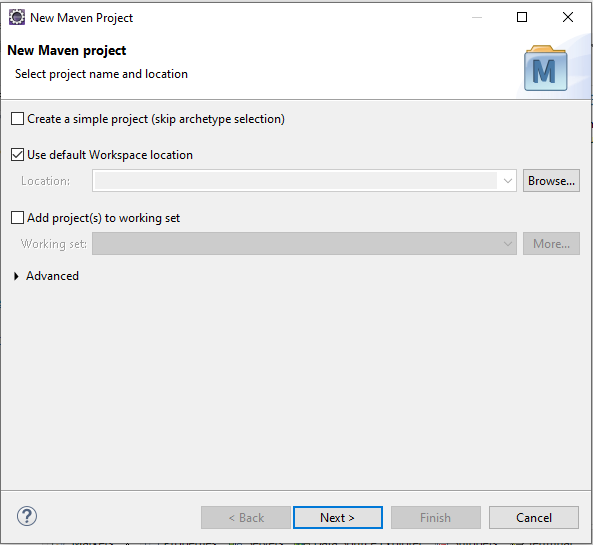
Note:- It is recommened to use junit with Maven/Gradel project becaue it gives the support built-in decompiled to see the source code of Junit library.

**5. Junit with Maveen Tool In Eclipse IDE:-**

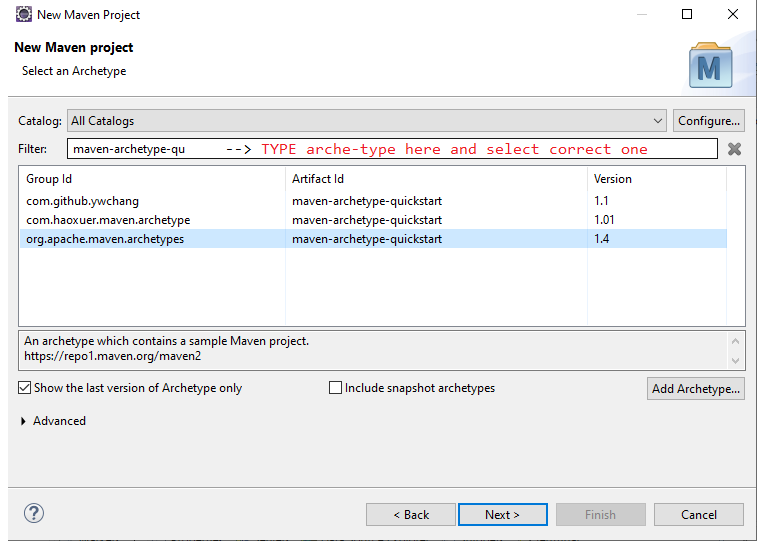
**Step1: Create Maven project in eclipse IDE as standlone project and change the java version to 19.**

Create Maven project.

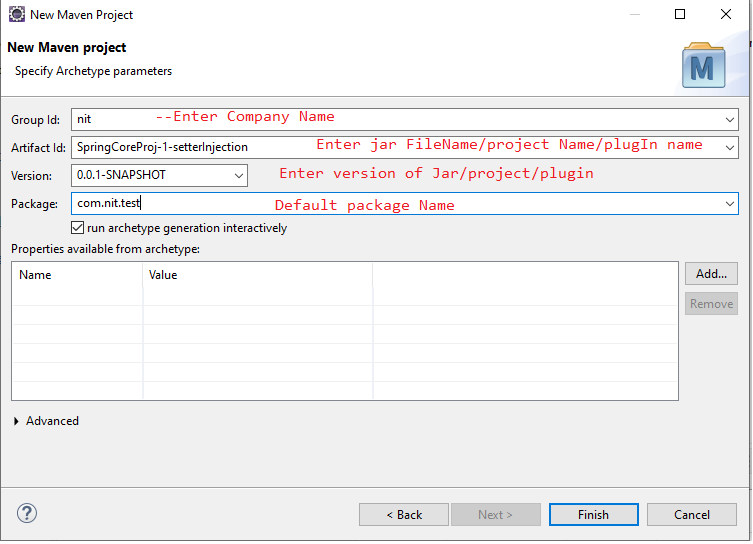
File🡪new 🡪New Maven project



1.1.Click on next button.

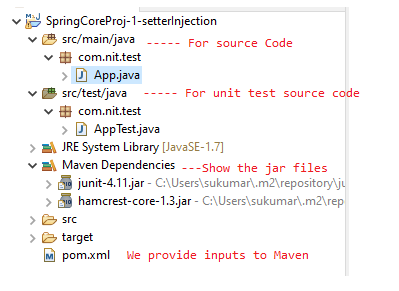


1.2 click the Next button.



1.3. click the Finish Button.

It gives following directory structure for SpringCoreProj-1setterInjection.



Change the Java version. Open pom.xml file and change the java version .



Right click on project 🡪 maven 🡪 update project.

Step2. Develop the main class or service class in com.nit.junit package of src/main/java folder.

Arithmetic.java

===========

**package** com.nit.junit;

**public** **class** Arithmetic {

**public** **float** add(**float** a,**float** b) {

**return** a+b;

}

}

Step3: Develop the Testcase class with Test methods in com.nit.jnuint package of src/test/java folder

TestArithmetic.java

===============

package com.nit.junit;

import org.junit.jupiter.api.Assertions;

import org.junit.jupiter.api.Test;

public class TestArithmetic {

@Test

public void testAddValidInput() {

Arithmetic a1=new Arithmetic();

float actual=a1.add(2.5f,5.5f);

float expected=8.0f;

Assertions.assertEquals(expected,actual);

}

@Test

public void testAddInvalidInput() {

Arithmetic a1=new Arithmetic();

float actual=a1.add(2.5f,5.5f);

float expected=5.0f;

Assertions.assertEquals(expected,actual);

}

@Test

public void testAddInvalidInputMsg() {

Arithmetic a1=new Arithmetic();

float actual=a1.add(2.5f,5.5f);

float expected=5.0f;

Assertions.assertEquals(expected,actual,"TestResult May not be same");

}

@Test

public void testAddInvalidInputWithDelta() {

Arithmetic a1=new Arithmetic();

float actual=a1.add(2.5f,5.5f);

float expected=5.0f;

Assertions.assertEquals(expected,actual,5.0f);

}

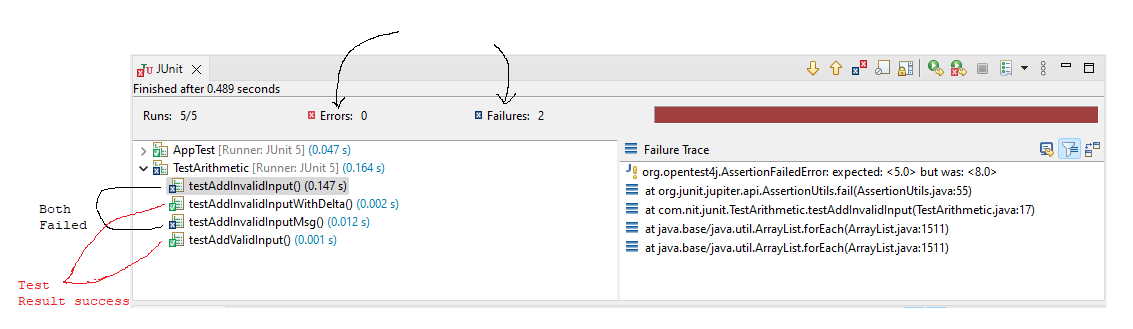
}

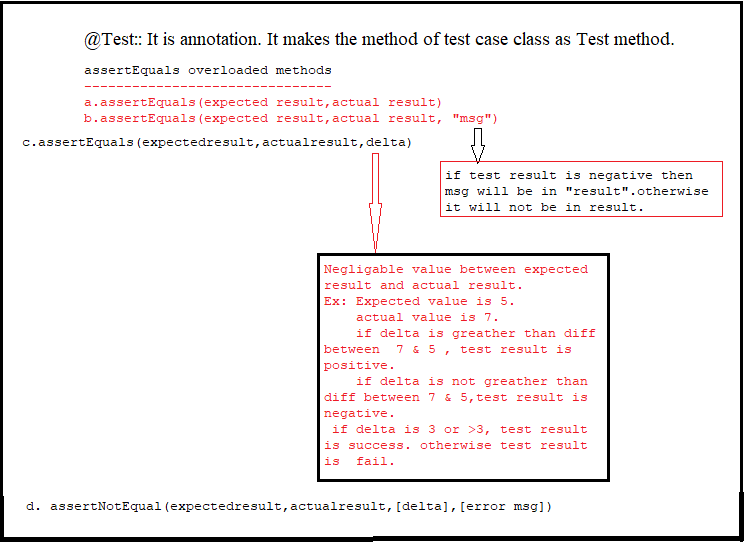
Step4: Run the Application

Select & right click on project🡪 Run As 🡪 Junit Test

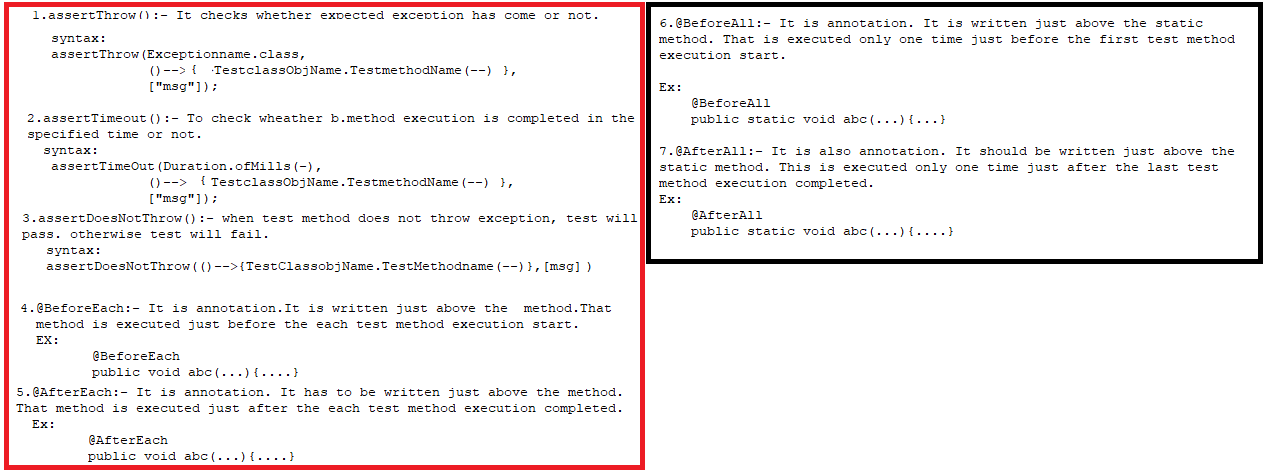
Output:

======





Example:2 The Junit application demonstrates following assertmethods and annotations.



Service Class: Arithmetic.java

=====================

**package** com.nit.junit;

**public** **class** Arithmetic {

**public** **float** add(**float** a,**float** b) **throws** Exception{

**if**(a==0 || b==0) {

Thread.*sleep*(2000);

**throw** **new** ArithmeticException();

}

**else**

{

Thread.*sleep*(2000);

**return** a+b;

}

}

}

TestCase Class:TestArithmetic.java

==========================

**package** com.nit.junit;

**import** java.time.Duration;

**import** org.junit.jupiter.api.AfterAll;

**import** org.junit.jupiter.api.AfterEach;

**import** org.junit.jupiter.api.Assertions;

**import** org.junit.jupiter.api.BeforeAll;

**import** org.junit.jupiter.api.BeforeEach;

**import** org.junit.jupiter.api.Test;

**public** **class** TestArithmetic {

**static** Arithmetic *a1*;

@BeforeEach

**public** **void** first() { *a1*=**new** Arithmetic();

System.***out***.println("First Method Executed"); }

@AfterEach

**public** **void** last() { System.***out***.println("Last Method Executed"); }

@BeforeAll

**public** **static** **void** firstAll() {

*a1*=**new** Arithmetic();

System.***out***.println("This is Before All annotations"); }

@AfterAll

**public** **static** **void** afterAll() {

System.***out***.println("This is After All annotations"); }

@Test

**public** **void** testAddThrow1() {

Assertions.*assertThrows*(ArithmeticException.**class**, ()->{

*a1*.add(20, 30);

});

}

@Test

**public** **void** testAddThrow2() {

Assertions.*assertThrows*(ArithmeticException.**class**, ()->{

*a1*.add(0, 30);

});

}

@Test

**public** **void** testAddTimeOut1() {

Assertions.*assertTimeout*(Duration.*ofMillis*(1000), ()->{

*a1*.add(10, 40);

});

}

@Test

**public** **void** testAddTimeOut2() {

Assertions.*assertTimeout*(Duration.*ofMillis*(6000), ()->{

*a1*.add(10, 40);

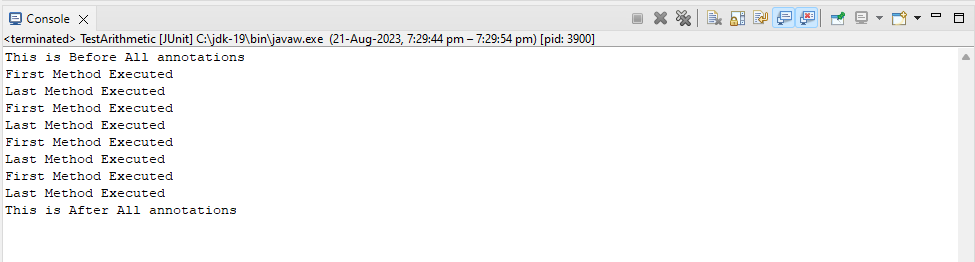
});

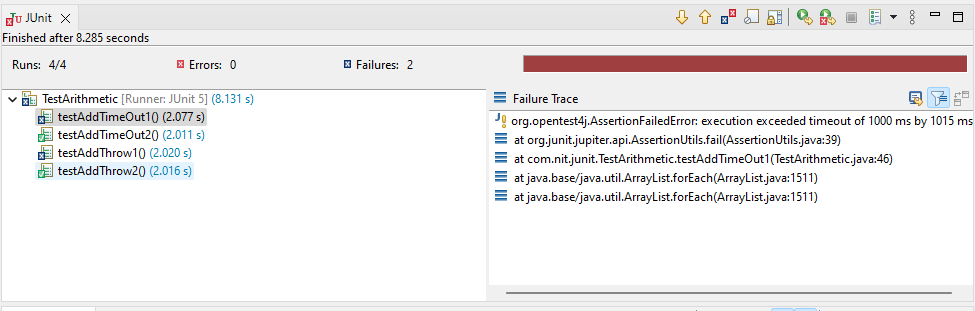
}

}

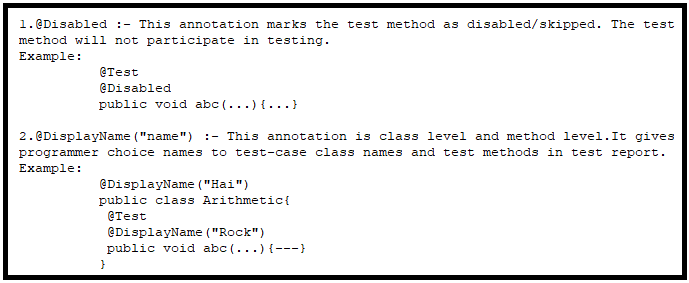
Output:

=====





Example:3 The Junit application demonstrates following annotations.



Service Class: Arithmetic.java

=====================

**package** com.nit.junit;

**public** **class** Arithmetic {

**public** **float** add(**float** a,**float** b) **throws** Exception{

**if**(a==0 || b==0) {

Thread.*sleep*(2000);

**throw** **new** ArithmeticException();

}

**else**

{

Thread.*sleep*(2000);

**return** a+b;

}

}

}

TestCase Class:TestArithmetic.java

==========================

**package** com.nit.junit;

**import** java.time.Duration;

**import** org.junit.jupiter.api.Assertions;

**import** org.junit.jupiter.api.BeforeAll;

**import** org.junit.jupiter.api.Disabled;

**import** org.junit.jupiter.api.DisplayName;

**import** org.junit.jupiter.api.Test;

@DisplayName("Sukumar Test:")

**public** **class** TestArithmetic {

**static** Arithmetic *a1*;

@BeforeAll

**public** **static** **void** abc() {

*a1*=**new** Arithmetic();

}

@Test

**public** **void** testAddThrow1() {

Assertions.*assertThrows*(ArithmeticException.**class**, ()->{

*a1*.add(20, 30);

});

}

@Test

@Disabled

@DisplayName("It will not participate")

**public** **void** testAddThrow2() {

Assertions.*assertThrows*(ArithmeticException.**class**, ()->{

*a1*.add(0, 30);

});

}

@Test

**public** **void** testAddTimeOut1() {

Assertions.*assertTimeout*(Duration.*ofMillis*(1000), ()->{

*a1*.add(10, 40);

});

}

@Test

@DisplayName("AddTimeOut-2")

**public** **void** testAddTimeOut2() {

Assertions.*assertTimeout*(Duration.*ofMillis*(6000), ()->{

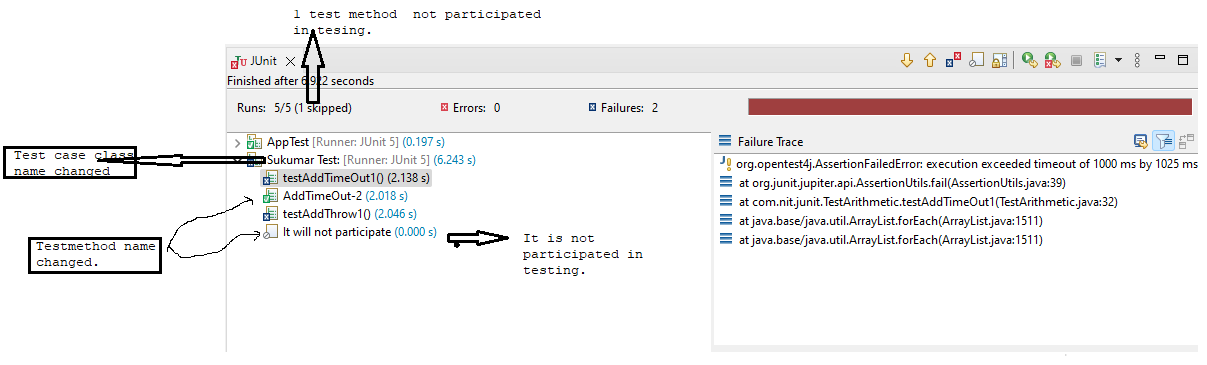
*a1*.add(10, 40);

});

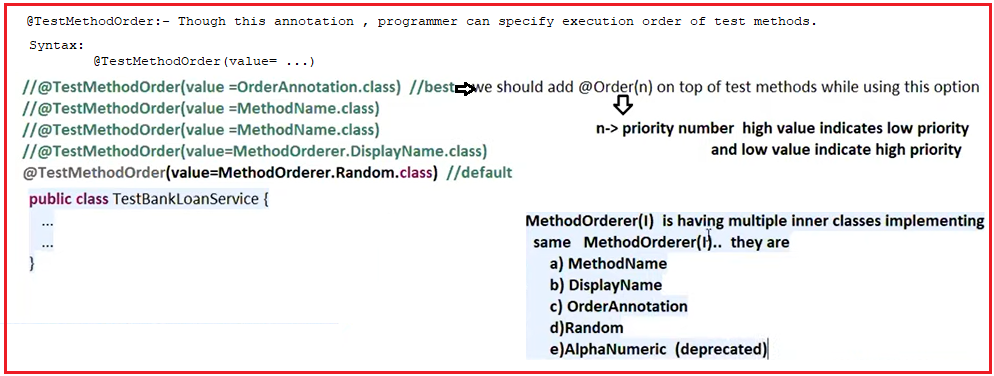
}

}

**Output:-**



Example:4 The Junit application demonstrates the following annotation.



Service Class: Arithmetic.java

=====================

**package** com.nit.junit;

**public** **class** Arithmetic {

**public** **float** add(**float** a,**float** b) **throws** Exception{

**if**(a==0 || b==0) {

Thread.*sleep*(2000);

**throw** **new** ArithmeticException();

}

**else**

{

Thread.*sleep*(2000);

**return** a+b;

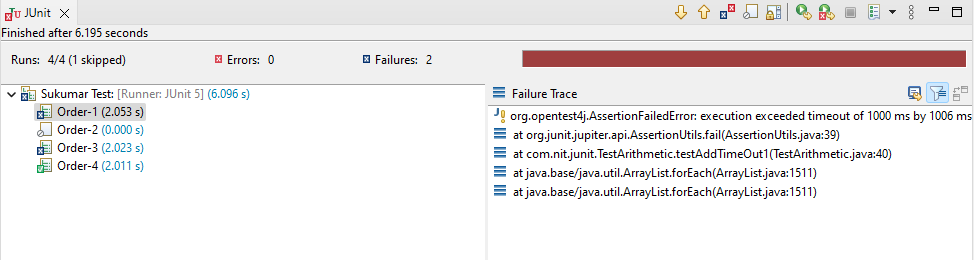
}

}

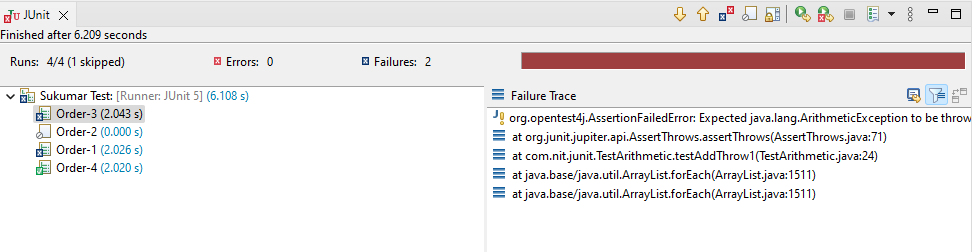
}

Run:1 Test methods are executed based on @Order annotation.

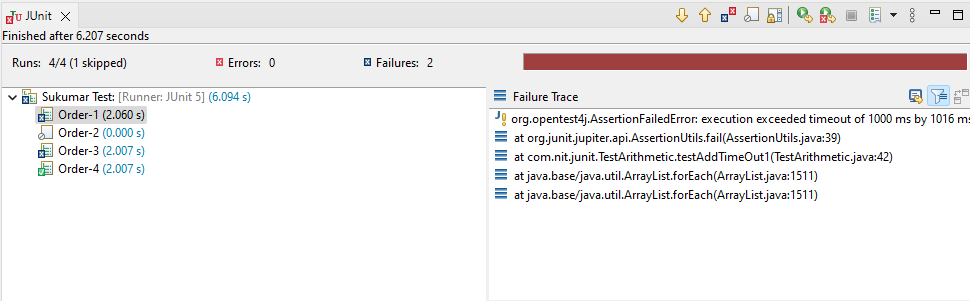
-----



Run:-2 Test methods are executed based method names.

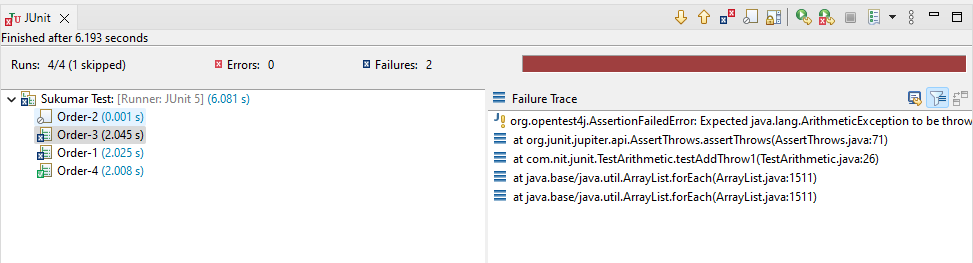


Run:3 Test methods are executed based @DisplayName annotation.

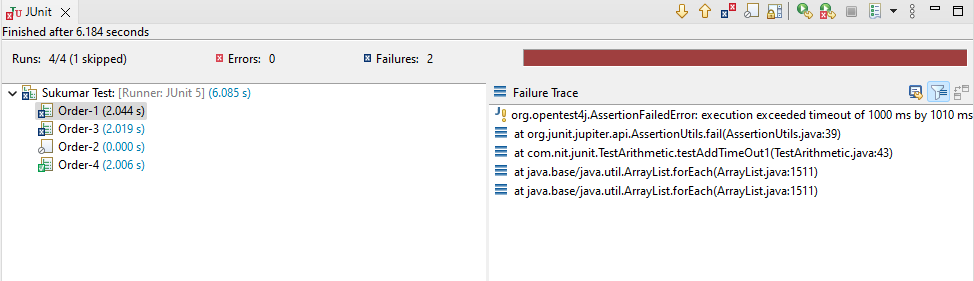


Run4: Test methods are executed Randomly.

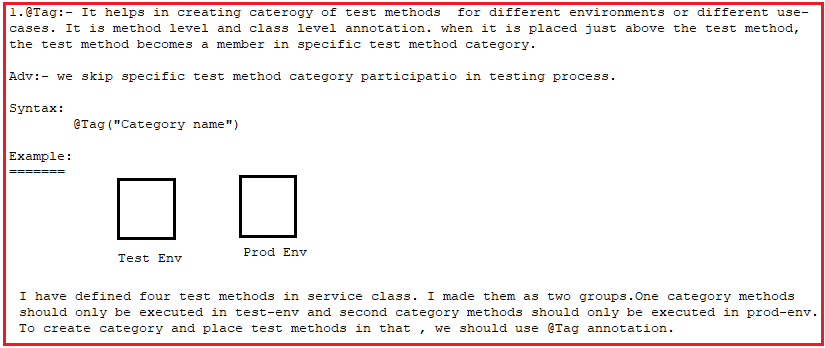
Run 4.1:



Run 4.2:



Example:4 The Junit application demonstrates the following annotations.



Service Class: Arithmetic.java

=====================

**package** com.nit.junit;

**public** **class** Arithmetic {

**public** **float** add(**float** a,**float** b) **throws** Exception{

**if**(a==0 || b==0) {

Thread.*sleep*(2000);

**throw** **new** ArithmeticException();

}

**else**

{

Thread.*sleep*(2000);

**return** a+b;

}

}

}

TestCase Class:TestArithmetic.java

==========================

**package** com.nit.junit;

**import** java.time.Duration;

**import** org.junit.jupiter.api.Assertions;

**import** org.junit.jupiter.api.BeforeAll;

**import** org.junit.jupiter.api.DisplayName;

**import** org.junit.jupiter.api.Tag;

**import** org.junit.jupiter.api.Test;

@DisplayName("Sukumar Test:")

**public** **class** TestArithmetic {

**static** Arithmetic *a1*;

@BeforeAll

**public** **static** **void** abc() {

*a1*=**new** Arithmetic();

}

@Test

@DisplayName("Order-3")

@Tag("test")

**public** **void** testAddThrow1() {

Assertions.*assertThrows*(ArithmeticException.**class**, ()->{

*a1*.add(20, 30);

});

}

@Test

@DisplayName("Order-2")

@Tag("prod")

**public** **void** testAddThrow2() {

Assertions.*assertThrows*(ArithmeticException.**class**, ()->{

*a1*.add(0, 30);

});

}

@Test

@DisplayName("Order-1")

@Tag("prod")

**public** **void** testAddTimeOut1() {

Assertions.*assertTimeout*(Duration.*ofMillis*(1000), ()->{

*a1*.add(10, 40);

});

}

@Test

@DisplayName("Order-4")

@Tag("test")

**public** **void** testAddTimeOut2() {

Assertions.*assertTimeout*(Duration.*ofMillis*(6000), ()->{

*a1*.add(10, 40);

});

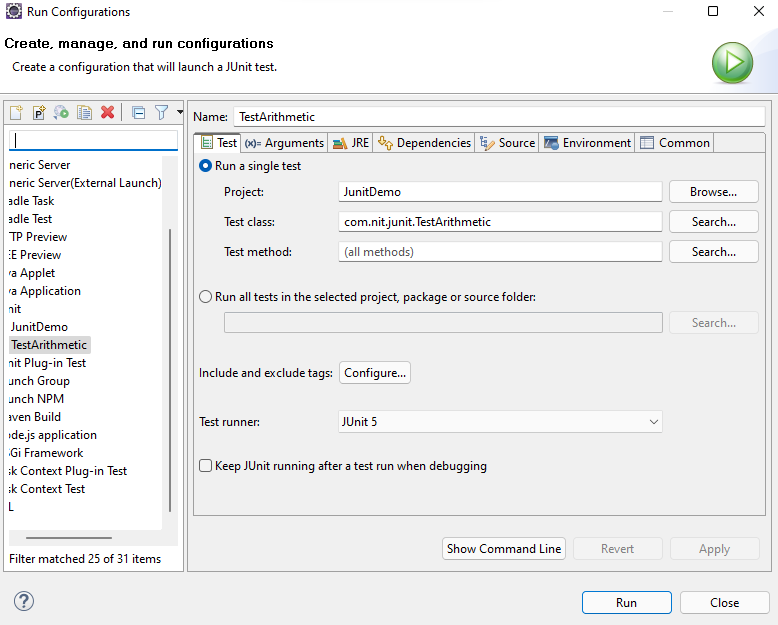
}

}

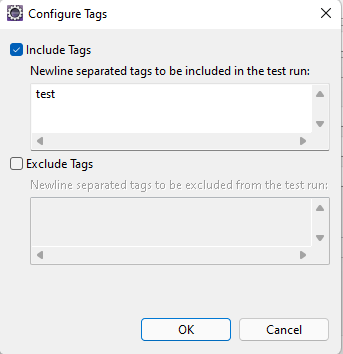
Execution Steps in Eclipse IDE Environment:

=================================

Select project 🡪rightclick on it 🡪run as 🡪 Run Configurations

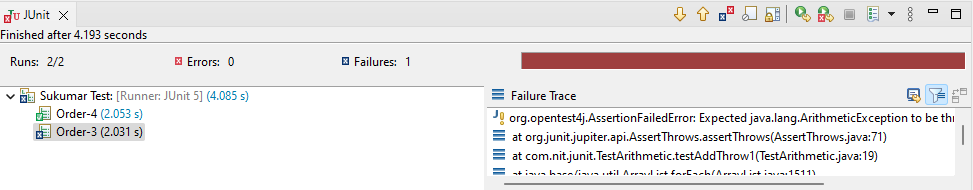


Click on “Test” tab 🡪 Configure



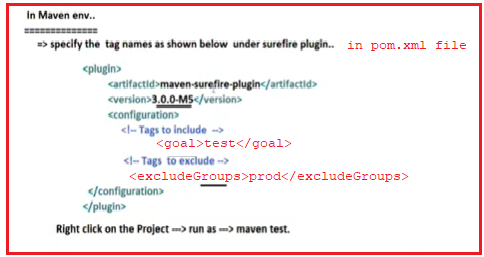
Enter the category names in include tags box and press “OK” button.

Output:-

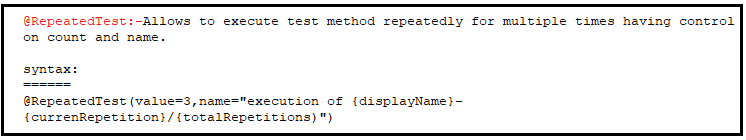


Execution steps in Maven Environments:

=============================



Example:5 The Junit application demonstrates the following annotations.



Example:5.1

----------------

Service Class: Arithmetic.java

=====================

**package** com.nit.junit;

**public** **class** Arithmetic {

**public** **float** add(**float** a,**float** b) {

**return** a+b;

}

}

TestCase Class:TestArithmetic.java

==========================

**package** com.nit.junit;

**import** org.junit.jupiter.api.Assertions;

**import** org.junit.jupiter.api.BeforeAll;

**import** org.junit.jupiter.api.DisplayName;

**import** org.junit.jupiter.api.RepeatedTest;

@DisplayName("Sukumar Test:")

**public** **class** TestArithmetic {

**static** Arithmetic *a1*;

@BeforeAll

**public** **static** **void** abc() {

*a1*=**new** Arithmetic();

}

@RepeatedTest(value=3)

@DisplayName("Order-1")

**public** **void** testAdd() {

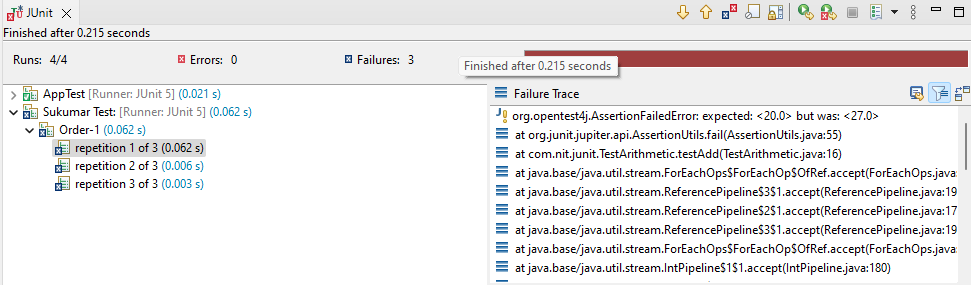
Assertions.*assertEquals*(20.0f, *a1*.add(15.f, 12.0f));

}

}

Output:

=====



Example:5.2

Service Class: Arithmetic.java

=====================

**package** com.nit.junit;

**public** **class** Arithmetic {

**public** **float** add(**float** a,**float** b) {

**return** a+b;

}

}

TestCase Class:TestArithmetic.java

==========================

**package** com.nit.junit;

**import** org.junit.jupiter.api.Assertions;

**import** org.junit.jupiter.api.BeforeAll;

**import** org.junit.jupiter.api.DisplayName;

**import** org.junit.jupiter.api.RepeatedTest;

@DisplayName("Sukumar Test:")

**public** **class** TestArithmetic {

**static** Arithmetic *a1*;

@BeforeAll

**public** **static** **void** abc() {

*a1*=**new** Arithmetic();

}

@RepeatedTest(value=3,name="execution of {displayName}-{currentRepetition}/{totalRepetitions}")

@DisplayName("Method")

**public** **void** testAdd() {

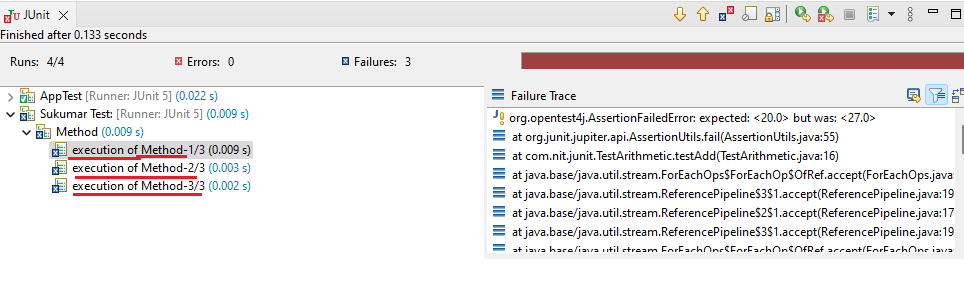
Assertions.*assertEquals*(20.0f, *a1*.add(15.f, 12.0f));

}

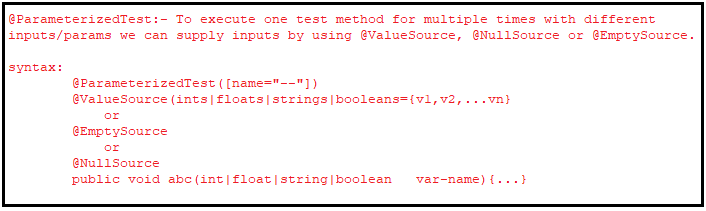
}

Output;

=====



Example:6 This JUnit application demonstrates following annotation.



Service Class: Arithmetic.java

=====================

**package** com.nit.junit;

**public** **class** Arithmetic {

**public** **float** add(**float** a,**float** b) {

**return** a+b;

}

**public** **int** slen(String a) {

**return** a.length();

}

}

TestCase Class:TestArithmetic.java

==========================

**package** com.nit.junit;

**import** org.junit.jupiter.api.Assertions;

**import** org.junit.jupiter.api.BeforeAll;

**import** org.junit.jupiter.api.DisplayName;

**import** org.junit.jupiter.params.ParameterizedTest;

**import** org.junit.jupiter.params.provider.EmptySource;

**import** org.junit.jupiter.params.provider.ValueSource;

@DisplayName("Sukumar Test:")

**public** **class** TestArithmetic {

**static** Arithmetic *a1*;

@BeforeAll

**public** **static** **void** abc() {

*a1*=**new** Arithmetic();

}

@ParameterizedTest(name="Float Type Arguments-")

@ValueSource(floats={10,20})

@DisplayName("Test ADD")

**public** **void** testAdd(**float** a) {

Assertions.*assertEquals*(30.0f, *a1*.add(a,10.0f));

}

@ParameterizedTest(name="String Type Arguments-")

@ValueSource(strings={"suku"," ","veena"})

@DisplayName("Float Arguments:")

**public** **void** testSlen(String x) {

Assertions.*assertEquals*(4, *a1*.slen(x));

}

@ParameterizedTest(name="Empty Arguments-")

@EmptySource

@DisplayName("Empty Arguments:")

**public** **void** testSlen1(String x) {

Assertions.*assertEquals*(1, *a1*.slen(x));

}

@ParameterizedTest(name="Null Argument-")

@NullSource

@DisplayName("null Argument:")

**public** **void** testSlen2(String x) {

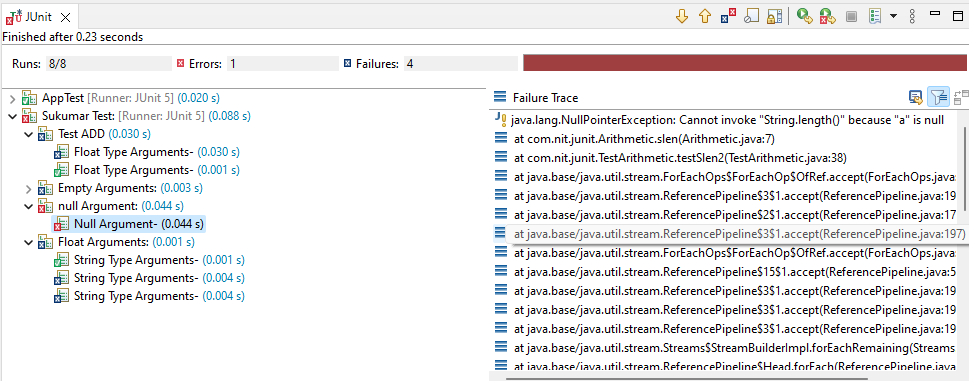
Assertions.*assertEquals*(1, *a1*.slen(x));

}

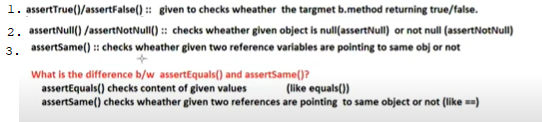
}

Output:

======



Example:7 This Application demonstrates the following Assertion methods.





Service Class: Arithmetic.java

=====================

**package** com.nit.junit;

**public** **class** Arithmetic {

**private** Arithmetic() {}

**private** **static** Arithmetic *inst*=**new** Arithmetic();

**public** **static** Arithmetic getInstance() {

**return** *inst*;

}

**public** **static** Arithmetic *x*=**null**;

}

TestCase Class:TestArithmetic.java

==========================

**package** com.nit.junit;

**import** org.junit.jupiter.api.Assertions;

**import** org.junit.jupiter.api.DisplayName;

**import** org.junit.jupiter.api.Test;

@DisplayName("Sukumar Test:")

**public** **class** TestArithmetic {

@Test

@DisplayName("Assert Not Null Methods")

**public** **void** test1() {

Assertions.*assertNotNull*(Arithmetic.*x*);

}

@Test

@DisplayName("Assert Null Methods")

**public** **void** test2() {

Assertions.*assertNull*(Arithmetic.*x*);

}

@Test

@DisplayName("Assert Same Methods")

**public** **void** test3() {

Arithmetic a1=Arithmetic.*getInstance*();

Arithmetic a2=Arithmetic.*getInstance*();

Assertions.*assertSame*(a1,a2);

}

@Test

@DisplayName("Assert Same Methods")

**public** **void** test4() {

Arithmetic a1=**null**;

Arithmetic a2=Arithmetic.*getInstance*();

Assertions.*assertSame*(a1,a2);

}

@Test

@DisplayName("Assertion .fail(-) method")

**public** **void** test5() {

Arithmetic a1=**null**;

**if**(a1==**null**) {

Assertions.*fail*("a1 does not have null:");

}

}

}

