**What is Junit**

It is framework for unit testing.

What is unit testing:-Test Driven Development (TDD).it is process where you write the test code first and then you write development code and pass to test.

Advantages

1. Separate your test from your code; no need for main methods.
2. Keep a growing library of tests.
3. It make sure new code does not break the existing code(Regression testing)

Is use to write quality code and maintain quality code

**Junit Architecture**

Junit is a framework for run test cases. Or it is framework for unit testing.

In order to run test cases, there is platform to run the test cases called Junit Engine.

In order to use junit platform, there is couple API to help us to utilise this platform.

There are 3 set of API available with Junit 5.

1. Vintage- it is use for backword compatibility
2. Jupiter-name suggest Ju means java unit
3. 3rd Party API-this help us to code custom api or custom type of test cases.

**How Junit Works?**

Example:

**package** edubridge.com;

**public** **class** RectangleUtilities {

**public** **static** **int** getArea(**int** height, **int** width){

**return** height\* width;

}

**public** **static** **int** getPerimeter(**int** height, **int** width){

**return** 2\*(height+width);

}

}

How do you show to the development team that the above methods is working correctly? Answer, you write junit test and the test passes.

**So we write test cases code**

package edubridge.com;

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

import org.junit.jupiter.api.Test;

class RectangleUtilitesTest {

@Test

public void testGetArea(){

int expected=6;

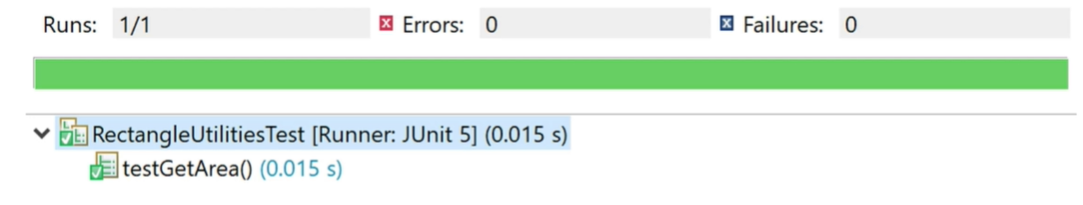
int actual = RectangleUtilities.getArea(2,3);

assertEquals(expected,actual);

}

}

//when you run the test class, if you get green bar(shown bellow)this mean all your test has passed.



Green mean good. Your test(s) have passed.

**What if your Test found an error?**

**Junit show red bar(given bellow) along with a helpful error message.**



In that case, you fix your source code and run the test again until you get the green bar, which let you know all of your test have passed.

**Local environment set up**

1. Eclipse/NetBean/InteliJ and JDK 1.5 or above OR
2. Eclipse with Maven project(here we need to add dependency code in POM.xml) OR
3. Spring boot

Here we will follow first approach that is 1) Eclipse and JDK 1.5 or above version

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

**Basic methods of Junit**

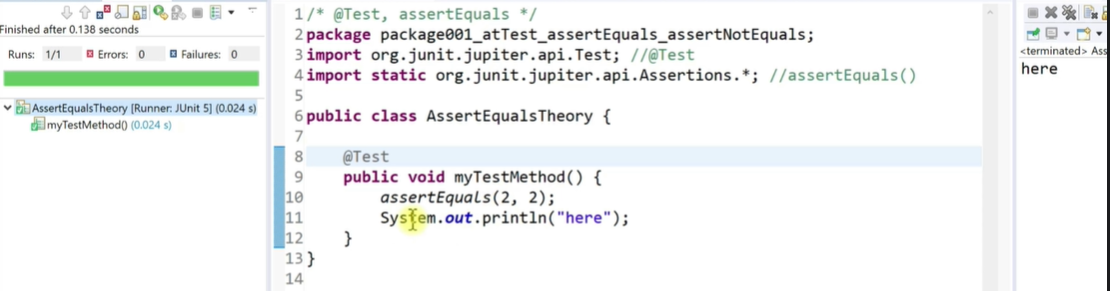
1)assertEquals():-check that 2 primitive/object is equal.

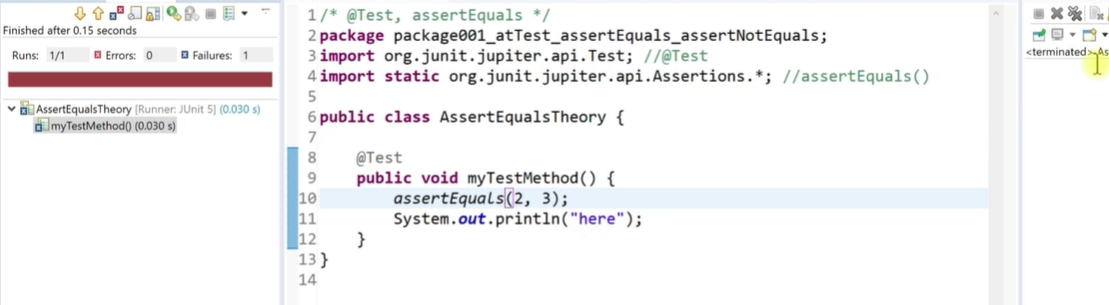
2)assertNotEquals():- check that an object is not equal

3) assertTrue():-

4)assertFalse()

5)assertArrayEquals()





**assertTrue:- assertTrue () eventually ask the question ,is it true that the value passed is true or evaluated to true, if so we will get green bar indicating the test has pass otherwise we will get red bar indicating that test has not been pass.**

**package** edubridge.com;

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

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

**class** AssertTrueTheory {

@Test

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

*assertTrue*(**true**);

}

@Test

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

*assertTrue*(4==4);

}

@Test

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

*assertTrue*("edubridge".length()==3);

}

}

**Assertfalse() :- assertfalse() basically the question, that value inside the parenthesis is false or evaluated to false ,if so then we will get the green bar indicating the test has pass otherwise we will get red bar indicating that test has not been pass.**

package edubridge.co;

import static org.junit.jupiter.api.Assertions.\*

import org.junit.jupiter.api.Test;

class AssertFalseTheory {

@Test

public void myTest(){

assertFalse(false);

assertFalse(2==3);

assertFalse("edubridge".length()==3);

}

}

More example

**package** sam.com;

**import** java.util.HashSet;

**public** **class** StringUtilitiesDemo {

HashSet <Character> lowerCaseVowels = **new** HashSet<>();

HashSet <Character> upperCaseVowels = **new** HashSet<>();

**public** StringUtilitiesDemo() {

lowerCaseVowels.add('a');

lowerCaseVowels.add('e');

lowerCaseVowels.add('i');

lowerCaseVowels.add('o');

lowerCaseVowels.add('u');

upperCaseVowels.add('A');

upperCaseVowels.add('E');

upperCaseVowels.add('I');

upperCaseVowels.add('O');

upperCaseVowels.add('U');

}

**public** **int** vowelCount(String s) {

**int** count=0;

**if**(s==**null**) { **return** 0;}

**for**(**int** i=0; i<s.length();i++) {

**if**(lowerCaseVowels.contains(s.charAt(i))) {

count++;

}**else** **if**(upperCaseVowels.contains(s.charAt(i))) {

count++;

}

}

**return** count;

}

**public** **boolean** isCapitalized(String s) {

**if**(s==**null** || s.length()==0) {**return** **false**;}

**else** {

**return** (upperCaseVowels.contains(s.charAt(0)));

}

}

}

//test method for above program

package sam.com;

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

import org.junit.jupiter.api.Test;

class StringUtilitiseDemoTest {

@Test

void testIsCapatalized() {

String s="Apple";

String t="apple";

String u ="grape";

String v=null;

String w="";

StringUtilitiesDemo stringUtilities =new StringUtilitiesDemo();

assertTrue(stringUtilities.isCapitalized(s));

assertFalse(stringUtilities.isCapitalized(t));

assertFalse(stringUtilities.isCapitalized(u));

assertFalse(stringUtilities.isCapitalized(v));

assertFalse(stringUtilities.isCapitalized(w));

}

}

assertArrayEquals() method

**package** sam.com;

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

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

**class** ArraryDemoTest {

@Test

**void** test() {

**int**[] array1= {1,2,3};

**int**[] array2= {1,2,3};

**int**[] array3= {1,7,7};

**int**[] array4= {1,2,3,4};

*assertArrayEquals*(array1, array2);//pass same element and same lenght

//assertArrayEquals(array2, array3);//fail one or more different element

//assertArrayEquals(array3, array4); // fail different length

}

}

assertAll() method : help us to execute more than one assert methods

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

Practice example

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

**Annotation in Junit**

1. @Test :-it tell the junit that the method which is attached run as a test case.
2. @BeforeAll:- Method marked with beforeAll run exactly once prior to the running of test methods.
3. @AfterAll:- Method marked with aferAll run exactly once after the running of test methods.

Note:- These methods useful when you need to perform set up and tear down work such as connecting to and disconnecting from a database.

1. @After:- This will perform the method after all tests have finished. This can be used to perform clean-up activities.
2. @Ignore:- The Ignore annotation is used to ignore the test and that test will not be executed.
3. @BeforeClass:- it run once before any of the test methods in the class.
4. @AfterClass:- it run once after any of the test methods in the class.it can be use to perform clean –up activity.
5. @TestInstance(Lifecycle.PER\_CLASS):- No need to write public static method while using @Before and @AfterAll annotation. You can write only void method.
6. @Disabled:- whenever you don’t want to run any method then you write @Disabled above the methods

Program for beforAll() and aferAll()

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

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

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

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

**class** BeforeAndAfterAllDemoTest {

@BeforeAll

**public** **static** **void** setup() {

System.***out***.println("firs");

}

@AfterAll

**public** **static** **void** tearDown() {

System.***out***.println("last");

}

@Test

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

System.***out***.println("now running test 1");

}

@Test

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

System.***out***.println("now running test 2");

}

}

-----

**Bug Fix Process**

1. Create the Junit test that triggers the bug. Run the test and get the Red bar. This shows you found the bug.
2. Fix the bug in source code.
3. Run your Junit test again and get green bar. This shows you fix the bug.
4. Submit your updated source code and unit test to your tech lead for review/merge.

**Feature Request**

A graphics team want a method for the TringleUtilities class that let them know if the triangle is equilateral.

Please implement this method and submit the source code along with a corresponding Junit test.

Thank you,

Software Development Team.

**Feature request process (TTD)**

1. Create an empty method that does nothing (have it thrown an exception, etc.)
2. Write a Junit test(s) that properly exercise the method.
3. Runt the Junit test(s).They will fail initially because the source code method has not yet been implemented.
4. Write and refine the source code method until all of your Junit test pass.
5. Submit your source code and your Junit test(s) to GitHub for review by your tech lead.

**Assertion Documentation and message param**

AssertEquals compare.but it can not compare the array. In order to compare array or iterable there are different types of methods.

There are 3 method are use for the comparison rest all methods are convenient methods

1)assertEquals

2)assertArray

3)assertIterable

**How to test for exceptions**

**package** sam.com;

**public** **class** MathFunctions {

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

**return** a+b;

}

**public** **int** sub(**int** a,**int** b) {

**return** a-b;

}

**public** **int** multiply(**int** a,**int** b) {

**return** a\*b;

}

**public** **int** divide(**int** a,**int** b) {

**return** a/b;

}

}

Test methods

package sam.com;

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

import org.junit.jupiter.api.Test;

class MathFunctionTest {

@Test

void testAdd() {

MathFunctions mathFunction= new MathFunctions();

int expected=40;

int actual= mathFunction.add(10,30);

assertEquals(expected,actual,"Addition operation failed");

}

@Test

void testSub() {

MathFunctions mathFunction= new MathFunctions();

int expected=-20;

int actual= mathFunction.sub(10,30);

assertEquals(expected,actual,"Substraction operation failed");

}

@Test

void testMultiply() {

MathFunctions mathFunction= new MathFunctions();

int expected=300;

int actual= mathFunction.multiply(10,30);

assertEquals(expected,actual,"Multiplication operation failed");

}

@Test

void testDivide() {

MathFunctions mathFunction= new MathFunctions();

int expected=0;

int actual= mathFunction.divide(10,30);

//int actual= mathFunction.divide(10,0);

assertEquals(expected,actual,"Division operation failed");

//bellow line will throw exeception

//assertThrows(Exception.class,()->mathFunction.divide(10,0));

assertThrows(ArithmeticException.class,()->mathFunction.divide(10,0),"ArithmeticException expected");

}

}

**Junit hooks**

Before anything (any method) call in class

After every method call in class

Method

Method

Class

Method

Method

Before every method call

After every method call

There are four hook in junit

1. @BeforeAll:- it will get executed before each and every methods in the class
2. @AfterAll :- it will get executed after each and every methods in the class.
3. @BeforeEach:-it will executed before every method call in the class
4. @AfterEach :- it will executed after every method call in the class.

Eg.

**package** sam.com;

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

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

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

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

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

//in previous ex ie AssertionMethod and Array project, we take the project.in this project there is code redenducancy.

// redendancy is MathFunctions mathFunction= new MathFunctions(); available in each method.

//how do we remove redendancy.

**class** MathFunctionTest {

MathFunctions mathFunction;

@BeforeEach

**void** BeforeEach() {

mathFunction =**new** MathFunctions();

}

@AfterEach

**void** afterEach() {

System.***out***.println("Execute after each");

}

@Test

**void** testAdd() {

**int** expected=40;

**int** actual= mathFunction.add(10,30);

*assertEquals*(expected,actual,"Addition operation failed");

}

@Test

**void** testSub() {

**int** expected=-20;

**int** actual= mathFunction.sub(10,30);

*assertEquals*(expected,actual,"Substraction operation failed");

}

@Test

**void** testMultiply() {

**int** expected=300;

**int** actual= mathFunction.multiply(10,30);

*assertEquals*(expected,actual,"Multiplication operation failed");

}

@Test

**void** testDivide() {

**int** expected=0;

**int** actual= mathFunction.divide(10,30);

//int actual= mathFunction.divide(10,0);

*assertEquals*(expected,actual,"Division operation failed");

//bellow line will throw exeception

//assertThrows(Exception.class,()->mathFunction.divide(10,0));

*assertThrows*(ArithmeticException.**class**,()->mathFunction.divide(10,0),"ArithmeticException expected");

}

}

**Assume method**

Surefire plugins:- it is plugin which enable us to run Junit test cases directly from the maven

**@Tag- it will run only those test cases which has @Tag**

@Tag(“runIt”)

To run @Tag –Right click on project- run as – run configuration – click on configure- click check box on include and write runIT. And try to run the test cases.

**How can we repeat the test certain number of times?**

**@Repeated Test annotation use when you want to run any method multiple time .**

**Ans :- @RepeatedTest(5)**

**@Test**

**Public void test1(){**

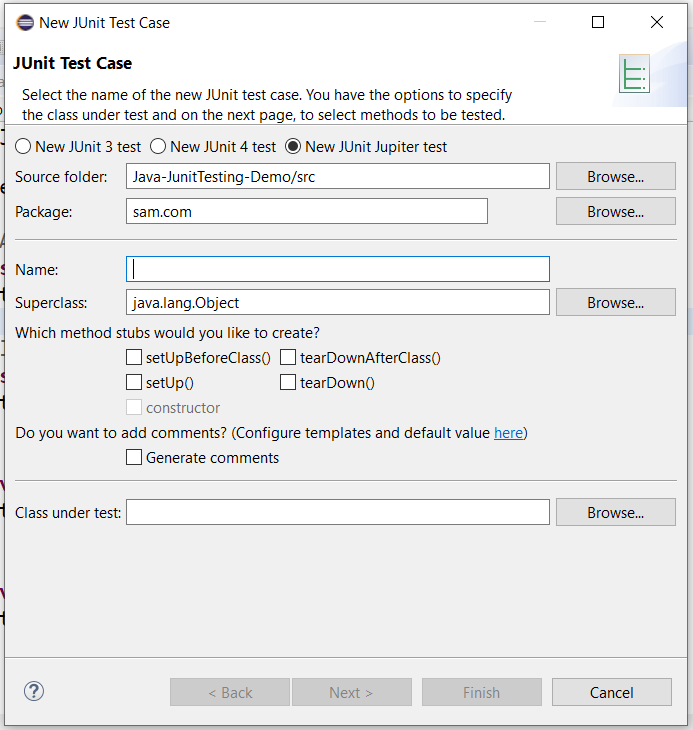
**Sop(“run again”)**

**}**

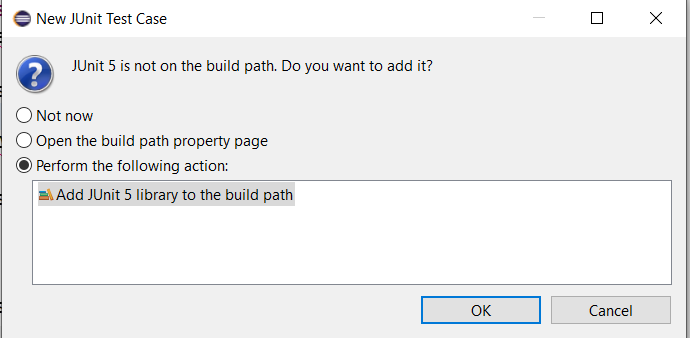
1. **Junit Project in Eclipse**

Step 1) open eclipse –file –create new java project

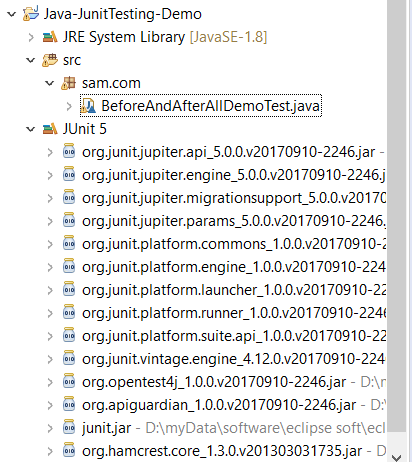
Step 2) Right click created project- new-junit test cases-write class name Like TestDemo



Step3)click next- it will show bellow diagram- click add junit 5 library to the build path- ok



Step 4) it will automatically add required jar file. See bellow diagram.



Step 5) write a test cases

1. **Eclipse with Maven project**

Maven can be used to install/manage JUnit, but it is not necessary to use Maven.

We will create simple Maven project with support of junit

1)new-maven project-create simple project (click check box)-next-give group id “sam.com”

Arrtifact id is name of project name- finished

2) Junit 5 Basic Maven configuration sample file (add bellow code in pom.xml)

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>org.srudyeasy</groupId>

<artifactId>JunitDemos</artifactId>

<version>0.0.1-SNAPSHOT</version>

<properties>

<maven.compiler.target>11</maven.compiler.target>

<maven.compiler.source>11</maven.compiler.source>

<juint.version>5.4.0</juint.version>

</properties>

<dependencies>

<!-- https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-engine -->

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-engine</artifactId>

<version>${juint.version}</version>

<scope>test</scope>

</dependency>

<!-- https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-api -->

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-api</artifactId>

<version>${juint.version}</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

3) **Write junit test method**

Public int add(int a,int b){

Return a+b;

}

1. **Now create a test case**

//old way

Class MathFuctionTest{

@Test

Void testAdd(){

MathFunctionTest mft=new MathFunctionTest();

Int expected=40;

Int actual =mft.add(10,30);

If(expected==actual){

Sop(“pass”)

}else{

Fail(“failed”);

}

//new way

Class MathFuctionTest{

@Test

Void testAdd(){

MathFunctionTest mft=new MathFunctionTest();

Int expected=40;

Int actual =mft.add(10,30);

assertEquals(expected,actual);

**The development process summery**

1. Write test that strategically test your code.
2. Keep writing/fixing/refining your code until all of your tests pass.
3. Ensure you did not break existing code

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