\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Experiment no:-7\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

Author:Saurabh Khandagale

Roll No:46

Date :4-April-2021

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

AIM:-

**To perform White-Box Testing to test the functionalities using JUnit testing tool.**

**What is White Box Testing?**

White box testing is a testing technique, that examines the program structure and derives test data from the program logic/code. The other names of glass box testing are clear box testing, open box testing, logic driven testing or path driven testing or structural testing.

**White Box Testing Techniques:**

**Statement Coverage** - This technique is aimed at exercising all programming statements with minimal tests.

**Branch Coverage** - This technique is running a series of tests to ensure that all branches are tested at least once.

**Path Coverage** - This technique corresponds to testing all possible paths which means that each statement and branch is covered.

**Advantages of White Box Testing:**

Forces test developer to reason carefully about implementation.

Reveals errors in "hidden" code.

Spots the Dead Code or other issues with respect to best programming practices.

**Disadvantages of White Box Testing:**

Expensive as one has to spend both time and money to perform white box testing.

Every possibility that few lines of code are missed accidentally.

In-depth knowledge about the programming language is necessary to perform white box testing.

CODE:-

[Q1IT.java]

import java.util.\*;

public class Q1

{

public String qdeq(int a,int b,int c)

{

String str="";

double d= b \* b - 4.0 \* a \* c;

if (d> 0.0)

{

double r1 = (-b + Math.pow(d, 0.5)) / (2.0 \* a);

double r2 = (-b - Math.pow(d, 0.5)) / (2.0 \* a);

str="The roots are " + r1 + " and " + r2;

System.out.println("The roots are " + r1 + " and " + r2);

}

else if (d == 0.0)

{

double r1 = -b / (2.0 \* a);

str="The root is " + r1;

System.out.println("The root is " + r1);

}

else

{

str="Roots are not real.";

System.out.println("Roots are not real.");

}

return str;

}

}

[Q1.java]

import org.junit.After;

import org.junit.AfterClass;

import org.junit.Before;

import org.junit.BeforeClass;

import org.junit.Test;

import static org.junit.Assert.\*;

public class Q1IT {

public Q1IT() {

}

@BeforeClass

public static void setUpClass() {

}

@AfterClass

public static void tearDownClass() {

}

@Before

public void setUp() {

}

@After

public void tearDown() {

}

@Test

public void testQdeq() {

System.out.println("Testcase1");

int a = 1;

int b = -2;

int c = 1;

Q1 instance = new Q1();

String expResult = "Roots are not real.";

String result = instance.qdeq(a, b, c);

assertEquals(expResult, result);

// TODO review the generated test code and remove the default call to fail.

// fail("The test case is a prototype.");

}

@Test

public void testQdeq2() {

System.out.println("Testcase2");

int a = 1;

int b = 5;

int c = 2;

Q1 instance = new Q1();

String expResult = "The roots are -0.4384471871911697 and -4.561552812808831";

String result = instance.qdeq(a, b, c);

assertEquals(expResult, result);

// TODO review the generated test code and remove the default call to fail.

// fail("The test case is a prototype.");

}

@Test

public void testQdeq3() {

System.out.println("Testcase3");

int a = 1;

int b = -2;

int c = 1;

Q1 instance = new Q1();

String expResult = "The root is 1.0";

String result = instance.qdeq(a, b, c);

assertEquals(expResult, result);

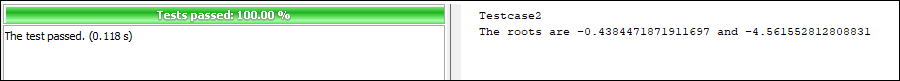
// TODO review the generated test code and remove the default call to fail.

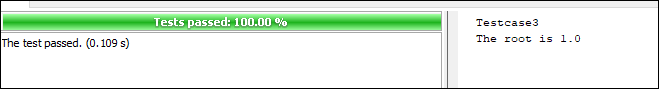
// fail("The test case is a prototype.");

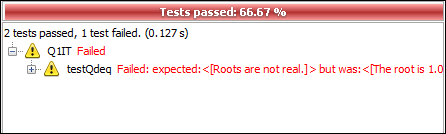
}

}

Output1:-







Input:-a=1,b=-2,c=1

[2]:- Choose any class from your Class Diagram (one which involves substantial computational logic) and implement it. Now create JUnit suit to test this class appropriately.

**delivery\_cost.java**

public class delivery\_cost {

public int cost(int distance)

{

int cost=0;

if(distance<=5){

cost=(int) (distance\*1.5);

}

if( distance<=10)

{

cost=(int) (distance\*1.7);

}

else

{

cost=(int) (distance\*2.5);

}System.out.println(" Testcase-1 = "+cost);

return cost;

} }

**Test.java**

import org.junit.After;

import org.junit.AfterClass;

import org.junit.Before;

import org.junit.BeforeClass;

import org.junit.Test;

import static org.junit.Assert.\*;

public class delivery\_costIT {

public delivery\_costIT() {

}

@BeforeClass

public static void setUpClass() {

}

@AfterClass

public static void tearDownClass() {

}

@Before

public void setUp() {

}

@After

public void tearDown() {

}

@Test

public void testCost() {

System.out.println("TESTCASE-1 [LOW-COST CHECKER]");

int distance = 4;

delivery\_cost instance = new delivery\_cost();

int expResult =6;

int result = instance.cost(distance);

assertEquals(expResult, result);

}

public void testCost() {

System.out.println("TESTCASE-2 [MID-COST CHECKER]");

int distance = 9;

delivery\_cost instance = new delivery\_cost();

int expResult =15 ;

int result = instance.cost(distance);

assertEquals(expResult, result);

}

public void testCost() {

System.out.println("TESTCASE-2 [HIGH-COST CHECKER]");

int distance = 82;

delivery\_cost instance = new delivery\_cost();

int expResult =205 ;

int result = instance.cost(distance);

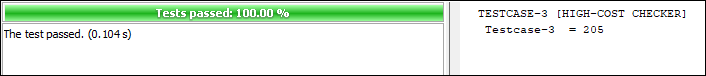
assertEquals(expResult, result);

}

}

Output:-

Distance=82



Distance=84



