INDEX

Developing test cases for specific problem scenarios in formal test case template. Introduction to automated test driven development through JUnit in Netbeans.	
QUES NO.	**Problem Description**
AP1	Construct proper test cases for a general registration form (https://www.editorialmanager.com/bjit/default.aspx). The test scenarios can be as follows: • Verify that the registration page loads successfully (all specified fields appear). • Verify that the required/mandatory fields are marked with * against the field. • Verify the page has both submit and cancel/reset buttons at the end. • Verify that clicking submit button after entering all the required fields, submits the data to the server. • Verify that clicking cancels/reset button after entering all the required fields, cancels the submit request, and reset all the fields. • Verify that not filling the mandatory fields and clicking the submit button will lead to a validation error.
AP2	Demonstrate test-driven development through a simple project called Geometry in Net Beans. Generate test class for a class Square.java that creates a square and calculates its area. Execute the auto-generated class for inputs in which test cases fails as well as inputs where test cases pass. Record both test cases in proper test case templates. Also write the specification of the class Square.java
AP3	Demonstrate test driven development through a class Arithmetic.java that divides two numbers passed to the division method. Generate ArithmeticTest class that creates a Arithmetic object for testing of the Arithmetic class through JUNIT. Execute the ArithmeticTest class for inputs in which test cases fails as well as inputs where test cases pass. Record both test cases in proper test case templates.
AP4	In the class Arithmetic.java in Q3. Above add another sum() method. Also generate the corresponding sumTest() method through Junit. Execute the ArithmeticTest class for inputs in which test cases fails as well as inputs where test cases pass. Record both test cases in proper test case templates.
AP5	Demonstrate test-driven development through a java file, CoffeeMakerTest.java, which properly tests the CoffeeMaker.java class to ensure that a CoffeeMaker electrical device is working properly.
AA1	Demonstrate test driven development through a class MyArray. Write a linear search method to find out the largest element of the array. Design a test class to test this method, using JUNIT framework. Record both test cases in proper test case templates.
WEEK 2 LAB ASSIGNMENT: Creation of Test Suites in JUnit	
BP1	Construct proper test cases for a Bank ATM Machine. The test scenarios can be as follows: • Withdraw money from an ATM. • Deposit money into an ATM. • Transaction failed due to not enough cash
BP2	Demonstrate test driven development through a project Arithmetic. The project should contain classes for Addition, Subtraction, Multiplication, Division of two parameters of different types .Through JUnit create corresponding Test classes for each class. Aggregate these different tests in a test suite and execute the same.

BP3	Demonstrate test driven development through a project ProductRepository. The project should contain a class ProductRepository with appropriate methods for CRUD operations on a ProductRepository instance .Through JUnit create corresponding Test classes for each class. Aggregate these different tests in a test suite and execute the same.
BP4	Imagine that due to current situation caused due to COVID-19, Delhi University has decided to grant undergraduate admissions in science programs based on the following rules: a)Marks in Mathematics>=60 b) Marks in Physics>=50 c) Marks in Chemistry>=50 d) Total in all three subjects>=160 or e) Total in Maths & Physics>=120 If aggregate of an eligible candidate is greater than 225, he will be eligible for honors course. Write appropriate class to represent this situation. Through JUnit create corresponding Test classes for each class. Aggregate these different tests in a test suite and execute the same.
WEEK 3 LAB ASSIGNMENT: Creation of Parameterised Tests in JUnit	
CP1	Demonstrate test driven development through a project MessageUtility. The project should contain a class MessageUtil with appropriate methods for printing a Message .Through JUnit create corresponding Test classes and TestRunner classes to execute and test the above class.
CP2	Demonstrate test driven development through a project PrimeNumberChecker. The project should contain a class PrimeNumberChecker with appropriate methods for validating if an input number is prime or not on a PrimeNumberChecker instance .Through JUnit create corresponding Parameterized Test classes and TestRunner classes to execute the above test on atleast 5 numbers.
CP3	Demonstrate test driven development through a project SquareChecker. The project should contain a class SquareChecker with appropriate methods for returning the square of an integer instance .Through JUnit create corresponding Parameterized Test classes and TestRunner classes to execute the above test on atleast 5 numbers. The test cases should be chosen using boundary value analysis technique for an input range from [100-500].
CP4	Demonstrate test driven development through a project Addition. The project should contain a class Addition with appropriate methods for returning the sum of two numbers. Through JUnit create corresponding Parameterized Test classes and TestRunner classes to execute the above test on a collection of test cases. The test cases should be chosen using boundary value analysis technique for an input range from [1-10] for the first number and [11-20] for the second number .
CA1	Demonstrate test driven development through a project QuadraticEquationChecker. The project should contain a class QuadraticEquationCheck with appropriate methods for returning the type of quadratic equation for a given set of inputs .Through JUnit create corresponding Parameterized Test classes and TestRunner classes to execute the above test on a collection of test cases. The test cases should be chosen using boundary value analysis technique for an input range from [1-100] .
WEEK 4 LAB ASSIGNMENT: Assert and TestCase class of The JUnit API along with revision of parameterized tests. Testing for an exception, timeout and setting up environment in The JUnit API.	
DP1	Create a new project called AssertClass. Refactor the test class and TestRunner class of question CP1 to demonstrate the assertion methods of the assert class to write useful tests.
DP2	In the MessageUtil problem of CP1 add an infinite while loop inside the printMessage() method. In the corresponding test class add timeout of 1000 to the testprintMessage() test case. Also use the TestRunner.java class to execute the test case.
DP3	In the printMessage() method of CP1 add an error condition inside the printMessage() method. Test this exception occurrence through appropriate changes in the test class.

DP4	In Question CP1 above demonstrate the usage of @Ignore annotation with MessageUtilityTest method.
DP5	IN questionCP2 above copy and refactor the class PrimeNumberCheckerTest.java to create other classes PrimeNumberCheckerTest1.java and PrimeNumberChecker11.java. In one assign the object to null and then test the NullPointer Exception and in second convert one test case to false and also test for the timeout parameter. Now create a test suite to execute all these three test cases. Check for the output to check if the mutants were detected or not. Also ignore the modified test cases and rerun the test suite to confirm the outcome.
DP6	Consider University Registration System for Use case testing: a. Identify actors & users. Draw usecase diagram. b. Write short description of each usecase. c. Elaborate any two usecases and write test cases for these two usecases.

AP1. Construct proper test cases for a general registration form (https://www.editorialmanager.com/bjit/default.aspx).

The test scenarios can be as follows:

- **Verify that the registration page loads successfully (all specified fields appear).**
- **Verify that the required/mandatory fields are marked with * against the field.**
- **Verify the page has both, submit and cancel/reset buttons at the end.**
- **Verify that clicking submit button after entering all the required fields, submits the data to the server.**
- **Verify that clicking cancels/reset button after entering all the required fields, cancels the submit request, and reset all the fields.**
- **Verify that not filling the mandatory fields and clicking the submit button will lead to a validation error.**

Solution:

Test Case ID: T001	Module Name: General Registration Form
Section 1: Before Execution	Section 2: After Execution
Purpose: Verify that the registration page loads successfully (all specified fields appear).	**Execution History:** The clickable link was clicked that redirected to the target page.
Pre Conditions: The link provided should be valid.	**Results:** Page loads successfully.
Inputs: Registration Form Link to perform the test.	**If fails, any possible reason (optional):** None
Expected Output: An author registration form to be loaded with submit and cancel/reset buttons.	**Any Other Observation/s:** The registration form contains submit and cancel button, but no reset button found.
Post Conditions: All the buttons and input fields in the form are clickable and editable.	**Any Suggestions:** Reset button should be added.
Written By: Rhea Sidana	**Run By:** Rhea Sidana
Date: 14 September, 2021	**Date:** 14 September, 2021

Test Case ID: T002	Module Name: General Registration Form
Section 1: Before Execution	Section 2: After Execution
Purpose: Verify that the required/mandatory fields are marked with * against the field.	**Execution History:** The form contains required fields.
Pre Conditions: The link provided should be valid.	**Results:** Page loads successfully.
Inputs: Make some field mandatory for the form.	**If fails, any possible reason (optional):** None.
Expected Output: An error should be prompted if user attempts to submit the form without filling the required field in the form.	**Any Other Observation/s:** None.
Post Conditions: All the required fields should be filled.	**Any Suggestions:** None.
Written By: Rhea Sidana	**Run By:** Rhea Sidana
Date: 14 September, 2021	**Date:** 14 September, 2021

Test Case ID: T003	Module Name: General Registration Form
Section 1: Before Execution	**Section 2: After Execution**
Purpose: Verify the page has both, submit and cancel/reset buttons at the end.	**Execution History:** The page is loaded and verified the presence of submit and cancel button, but reset button is not found.
Pre Conditions: The link provided should be valid.	**Results:** Page loads successfully.
Inputs: All fields should be properly filled.	**If fails, any possible reason (optional):** None.
Expected Output: ➢ The form should be submitted after clicking on the submit button or prompt an error if all required fields are not found. ➢ The home page should be loaded, if the cancel button is clicked.	**Any Other Observation/s:** Reset button missing.
Post Conditions: After clicking the submit button, another page(user's logged in) should be loaded.	**Any Suggestions:** Reset button should be added.
Written By: Rhea Sidana	**Run By:** Rhea Sidana
Date: 14 September, 2021	**Date:** 14 September, 2021

Test Case ID: T004	Module Name: General Registration Form
Section 1: Before Execution	**Section 2: After Execution**
Purpose: Verify that clicking submit button after entering all the required fields, submits the data to the server.	**Execution History:** The page is loaded and input should be provided in all the required input fields to perform submission of the details on the server.
Pre Conditions: The link provided should be valid.	**Results:** Page loads successfully.
Inputs: All fields should be properly filled and submit button is clicked.	**If fails, any possible reason (optional):** None.
Expected Output: The form is submitted after clicking on the submit button and data is saved on the server.	**Any Other Observation/s:** Error is prompted, if the required input fields are not filled.
Post Conditions: User is directed to the another page.	**Any Suggestions:** None.
Written By: Rhea Sidana	**Run By:** Rhea Sidana
Date: 14 September, 2021	**Date:** 14 September, 2021

Test Case ID: T005	Module Name: General Registration Form
Section 1: Before Execution	**Section 2: After Execution**
Purpose: Verify that clicking cancels/reset button after entering all the required fields, cancels the submit request, and reset all the fields.	**Execution History:** The page is loaded and inputs are provided in all the required input fields to perform cancel/reset on the form.
Pre Conditions: The link provided should be valid.	**Results:** Page loads successfully.
Inputs: Some fields are filled and cancel/reset button is clicked.	**If fails, any possible reason (optional):** None.
Expected Output: ➢ The form should be directed to home page, if cancel button is clicked.	**Any Other Observation/s:** Reset button not found.

➢ All form fields should set to initial state similar to when loaded, if reset button is clicked.	
Post Conditions: User is directed to home page when cancel button is clicked.	**Any Suggestions:** Reset button should be added.
Written By: Rhea Sidana	**Run By:** Rhea Sidana
Date: 14 September, 2021	**Date:** 14 September, 2021

Test Case ID: T006	**Module Name: General Registration Form**
Section 1: Before Execution	**Section 2: After Execution**
Purpose: Verify that not filling the mandatory fields and clicking the submit button will lead to a validation error.	**Execution History:** The page is loaded and inputs are provided in some of the required input fields to perform submit on the form.
Pre Conditions: The link provided should be valid.	**Results:** Page loads successfully.
Inputs: Some fields are filled and submit button is clicked.	**If fails, any possible reason (optional):** None.
Expected Output: Validation error should be prompted, as not all required fields are provided.	**Any Other Observation/s:** None.
Post Conditions: Validation error is prompted.	**Any Suggestions:** None.
Written By: Rhea Sidana	**Run By:** Rhea Sidana
Date: 14 September, 2021	**Date:** 14 September, 2021

AP2. Demonstrate test-driven development through a simple project called Geometry in Net Beans. Generate test class for a class Square.java that creates a square and calculates its area. Execute the auto-generated class for inputs in which test cases fails as well as inputs where test cases pass. Record both test cases in proper test case templates. Also write the specification of the class Square.java

Solution:

Project: Geometry

Class: Square

Square.java

```
package geometry;

public class Square {

   // side of the square

   private int side;


   //constructor

   public Square(int side){

      this.side = side;

   }


   //function to calculate the area of the square

   public int Area(){

      return (side*side);

   }

}
```

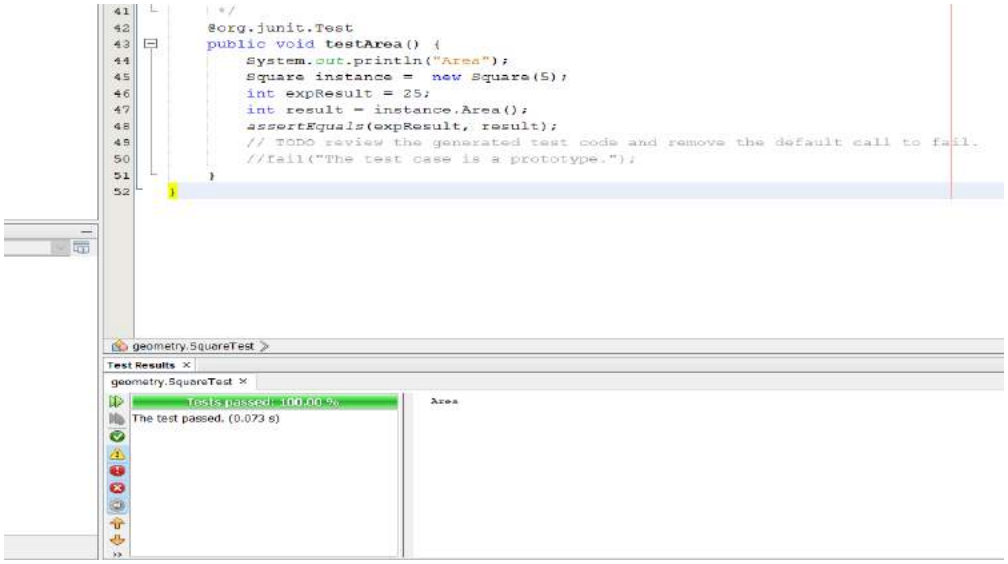
SquareTest.java

```
package geometry;


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 SquareTest {


    public SquareTest() {

    }


    @BeforeClass

    public static void setUpClass() {

    }


    @AfterClass

    public static void tearDownClass() {

    }


    @Before

    public void setUp() {

    }
```

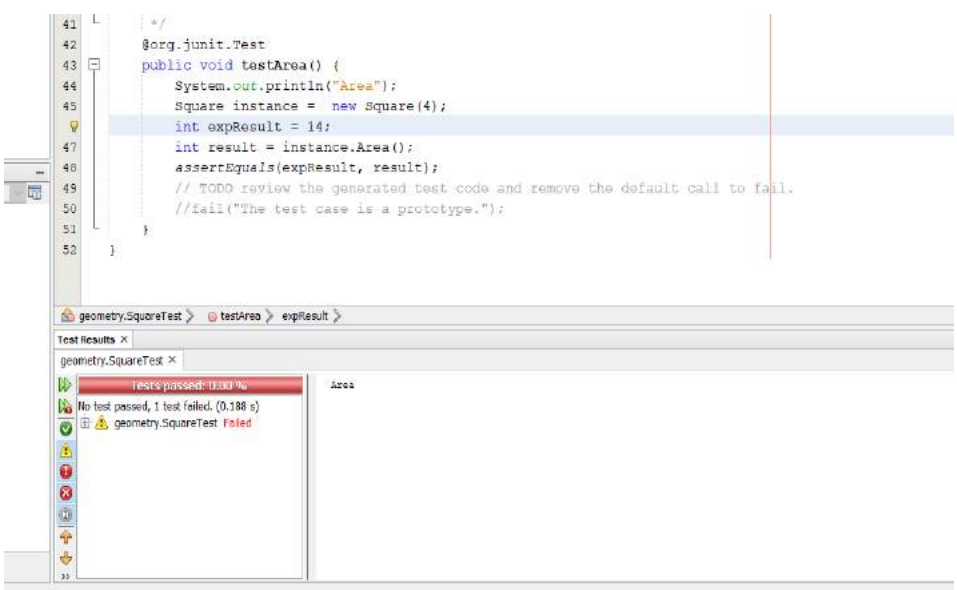
```
    @After
    public void tearDown() {
    }

    /**
     * Test of Area method, of class Square.
     */
    @org.junit.Test
    public void testArea() {
        System.out.println("Area");
        Square instance =  new Square(5);
        int expResult = 25;
        int result = instance.Area();
        assertEquals(expResult, result);
        // TODO review the generated test code and remove the default call to fail.
        //fail("The test case is a prototype.");
    }
}
```

Output:

TestID	Scenario	Precondition	Input	Success/Failure	Output	Expected Output	Post Condition	Result	Written By
1	To test the area of a square	Side must be greater than 0	5	Successful	25	25	The value of variable should be set to 0	Success	Rhea Sidana



TestID	Scenario	Precondition	Input	Success/Failure	Output	Expected Output	Post Condition	Result	Written By
2	To test the area of a square	Side must be greater than 0	4	Fail	16	14	The value of variable should be set to 0	Fail	Rhea Sidana



AP3. Demonstrate test driven development through a class Arithmetic.java that divides two numbers passed to the division method. Generate ArithmeticTest class that creates a Arithmetic object for testing of the Arithmetic class through JUNIT. Execute the ArithmeticTest class for inputs in which test cases fails as well as inputs where test cases pass. Record both test cases in proper test case templates.

Solution:

Project: Division

Class: Arithmetic

Arithmetic.java

```
package division;

public class Arithmetic {
   private int number1;
   private int number2;

   //constructor of the class
   public Arithmetic(int number1,int number2){
      this.number1 = number1;
      this.number2 = number2;
   }

   //division method
   public int division(){
      return (number1/number2);
   }
```

```
}
```

ArithmeticTest.java

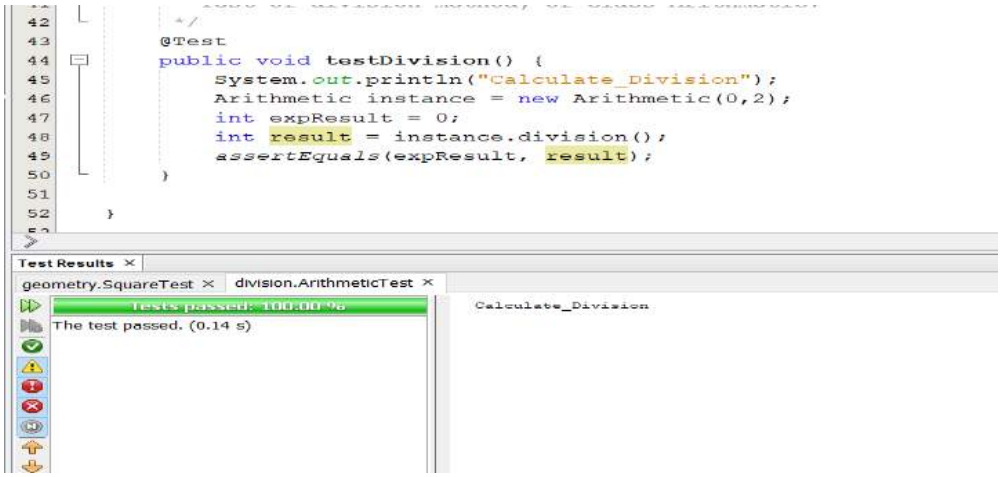
```
package division;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Test;
import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.BeforeClass;
public class ArithmeticTest {

    public ArithmeticTest() {
    }

    @BeforeClass
    public static void setUpClass() throws Exception {
    }

    @AfterClass
    public static void tearDownClass() throws Exception {
    }

    @Before
```

```
    public void setUp() throws Exception {
    }

    @After
    public void tearDown() throws Exception {
    }

    /**
     * Test of division method, of class Arithmetic.
     */
    @Test
    public void testDivision() {
        System.out.println("Calculate_Division");
        Arithmetic instance = new Arithmetic(0,2);
        int expResult = 0;
        int result = instance.division();
        assertEquals(expResult, result);
    }

}
```

Output:

TestID	Scenario	Precondition	Input	Success/Failure	Output	Expected Output	Post Condition	Result	Written By
1	To test the division	Denominator must be greater than	0,2	Success	0	0	The value of variable	Success	Rhea Sidana

	method	0 (zero)					should be set to 0		



TestID	Scenario	Precondition	Input	Success/Failure	Output	Expected Output	Post Condition	Result	Written By
2	To test the division method	Denominator must be greater than 0 (zero)	3,0	Fail	Error	Error	The value of variable should be set to 0	Fail	Rhea Sidana

TestID	Scenario	Precondition	Input	Success/Failure	Output	Expected Output	Post Condition	Result	Written By
3	To test the division method	Denominator must be greater than 0 (zero)	6,3	Fail	2	3	The value of variable should be set to 0	Fail	Rhea Sidana

```
@Test
public void testDivision() {
    System.out.println("Calculate_Division");
    Arithmetic instance = new Arithmetic(6,3);
    int expResult = 3;
    int result = instance.division();
    assertEquals(expResult, result);
}

}
```

division.ArithmeticTest > testDivision > expResult

Test Results

geometry.SquareTest | division.ArithmeticTest

Tests passed: 0.00 %

No test passed, 1 test failed. (0.125 s)

division.ArithmeticTest Failed

Calculate_Division

TestID	Scenario	Precondition	Input	Success/Failure	Output	Expected Output	Post Condition	Result	Written By
4	To test the division method	Denominator must be greater than 0 (zero)	6,3	Success	2	2	The value of variable should be set to 0	Success	Rhea Sidana

```
@Test
public void testDivision() {
    System.out.println("Calculate_Division");
    Arithmetic instance = new Arithmetic(6,3);
    int expResult = 2;
    int result = instance.division();
    assertEquals(expResult, result);
}

}
```

division.ArithmeticTest > testDivision > expResult

Test Results

geometry.SquareTest | division.ArithmeticTest

Tests passed: 100.00 %

The test passed. (0.143 s)

Calculate_Division

AP4. In the class Arithmetic.java in Q3. Above add another sum() method. Also generate the corresponding sumTest() method through Junit. Execute the ArithmeticTest class for inputs in which test cases fails as well as inputs where test cases pass. Record both test cases in proper test case templates.

Solution:

Project: Division

Class: Arithmetic

Arithmetic.java

```
package division;

public class Arithmetic {
   private int number1;
   private int number2;

   //constructor of the class
   public Arithmetic(int number1,int number2){
      this.number1 = number1;
      this.number2 = number2;
   }

   //division method
   public int division(){
      return (number1/number2);
   }
```

```
    //add method
    public int sum(){
        return (number1+number2);
    }
}
```

ArithmeticTest.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package division;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Test;
import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.BeforeClass;
public class ArithmeticTest {

    public ArithmeticTest() {
    }
```

```
@BeforeClass
public static void setUpClass() throws Exception {
}

@AfterClass
public static void tearDownClass() throws Exception {
}

@Before
public void setUp() throws Exception {
}

@After
public void tearDown() throws Exception {
}

/**
 * Test of division method, of class Arithmetic.
 */
@Test
public void testDivision() {
    //System.out.println("division");
    Arithmetic instance = new Arithmetic(6,3);
    int expResult = 2;
    int result = instance.division();
```

```
        assertEquals(expResult, result);
    }

    /**
     * Test of sum method, of class Arithmetic.
     */
    @Test
    public void testSum() {
        System.out.println("sum");
        Arithmetic instance = new Arithmetic(6,3);
        int expResult = 9;
        int result = instance.sum();
        assertEquals(expResult, result);
    }

}
```

Output:

TestID	Scenario	Precondition	Input	Success/Failure	Output	Expected Output	Post Condition	Result	Written By
1	To test the sum method	Numbers must be greater than 0 (zero)	6,3	Success	9	9	The value of variable should be set to 0	Success	Rhea Sidana

```
51          */
52          @Test
53          public void testSum() {
54              System.out.println("sum");
55              Arithmetic instance = new Arithmetic(6,3);
56              int expResult = 9;
57              int result = instance.sum();
58              assertEquals(expResult, result);
59          }
60
61      }
```

Test Results ×

division.ArithmeticTest ×

Tests passed: 100.00 %

Both tests passed. (0.09 s)

sum

TestID	Scenario	Precondition	Input	Success/Failure	Output	Expected Output	Post Condition	Result	Written By
2	To test the sum method	Numbers must be greater than 0 (zero)	6,3	Fail	9	12	The value of variable should be set to 0	Fail	Rhea Sidana

```
52          @Test
53          public void testSum() {
54              System.out.println("sum");
55              Arithmetic instance = new Arithmetic(6,3);
56              int expResult = 12;
57              int result = instance.sum();
58              assertEquals(expResult, result);
59          }
60
61      }
```

division.ArithmeticTest > testSum > expResult >

Test Results ×

division.ArithmeticTest ×

Tests passed: 50.00 %

1 test passed, 1 test failed. (0.086 s)

division.ArithmeticTest Failed

sum

AP5. Demonstrate test-driven development through a java file, CoffeeMakerTest.java, which properly tests the CoffeeMaker.java class to ensure that a CoffeeMaker electrical device is working properly.

Solution:

Project: CoffeeMakerMachine

Class: CoffeeMaker

CoffeeMaker.java

```
package coffeemakermachine;


public class CoffeeMaker {

    private String name, type;

    private String[] ingredients;

    private int power, price, time;


    //constructor

    public CoffeeMaker(String name,String type,String[] ingredients,int power,int price,int time){

        this.name = name;

        this.type = type;

        this.ingredients = ingredients;

        this.power = power;

        this.price = price;

        this.time = time;

    }


    //get power
```

```
public int getPower(){
    if(this.power<0){
        return 0;
    }
    return this.power;
}

//get price
public int getPrice(){
    if(this.type.equals("Mocha") && this.price==40){
        return price;
    }
    else if(this.type.equals("Latte") && this.price==50){
        return price;
    }
    else if(this.type.equals("Irish") && this.price==60){
        return price;
    }
    return 0;
}

//get time
public int getTime(){
    if(this.time<0){
        return 0;
```

```
        }
        return this.time;
    }

    //get number of ingredients
    public int getIngredientsCount(){
        return this.ingredients.length;
    }
}
```

CoffeeMakerTest.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package coffeemakermachine;

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.*;
```

```
/**
 *
 * @author MAX
 */
public class CoffeeMakerTest {

    public CoffeeMakerTest() {
    }

    @BeforeClass
    public static void setUpClass() {
    }

    @AfterClass
    public static void tearDownClass() {
    }

    @Before
    public void setUp() {
    }

    @After
    public void tearDown() {
    }
```

```
/**
 * Test of getPower method, of class CoffeeMaker.
 */
@Test
public void testGetPower() {
    System.out.println("getPower");
    //CoffeeMaker(String name,String type,String[] ingredients,int power,int price,int time)
    String[] ingredients = {"Coffee","Milk"};
    CoffeeMaker instance = new CoffeeMaker("NestCafe","Mocha",ingredients,220,40,10);
    int expResult = 220;
    int result = instance.getPower();
    //System.out.println("Result"+result);
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default call to fail.
    //fail("The test case is a prototype.");
}


/**
 * Test of getPrice method, of class CoffeeMaker.
 */
@Test
public void testGetPrice() {
    System.out.println("getPrice");
    //CoffeeMaker(String name,String type,String[] ingredients,int power,int price,int time)
    String[] ingredients = {"Coffee","Milk"};
```

```
    CoffeeMaker instance = new CoffeeMaker("NestCafe","Mocha",ingredients,220,40,10);
    int expResult = 40;
    int result = instance.getPrice();
    //System.out.println("Result"+result);
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default call to fail.
    //fail("The test case is a prototype.");
}

/**
 * Test of getTime method, of class CoffeeMaker.
 */
@Test
public void testGetTime() {
    System.out.println("getTime");
    //CoffeeMaker(String name,String type,String[] ingredients,int power,int price,int time)
    String[] ingredients = {"Coffee","Milk"};
    CoffeeMaker instance = new CoffeeMaker("NestCafe","Mocha",ingredients,220,40,10);
    int expResult = 10;
    int result = instance.getTime();
    //System.out.println("Result"+result);
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default call to fail.
    //fail("The test case is a prototype.");
}
```

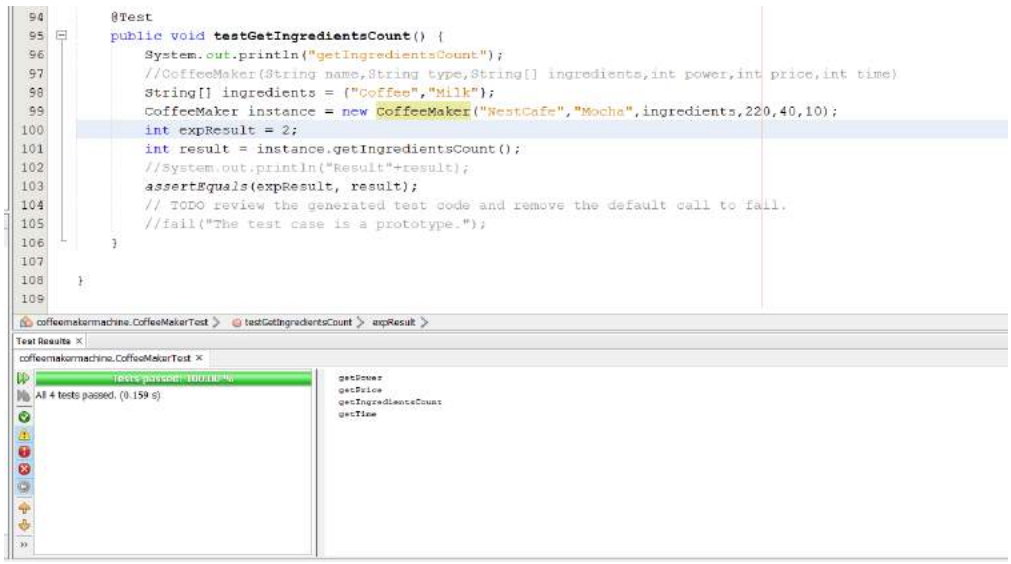
```
    /**
     * Test of getIngredientsCount method, of class CoffeeMaker.
     */
    @Test
    public void testGetIngredientsCount() {
        System.out.println("getIngredientsCount");
        //CoffeeMaker(String name,String type,String[] ingredients,int power,int price,int time)
        String[] ingredients = {"Coffee","Milk"};
        CoffeeMaker instance = new CoffeeMaker("NestCafe","Mocha",ingredients,220,40,10);
        int expResult = 2;
        int result = instance.getIngredientsCount();
        //System.out.println("Result"+result);
        assertEquals(expResult, result);
        // TODO review the generated test code and remove the default call to fail.
        //fail("The test case is a prototype.");
    }

}
```

Output:

(input: CoffeeMaker(String name,String type,String[] ingredients,int power,int price,int time))

(ouput: getPower(), getPrice(), getTime(), getIngredientCount())

TestID	Scenario	Precondition	Input	Success/ Fail	Output	Expected Output	Post Condition	Result	Written By
1	To test the CoffeeMaker class	Code should be executed successfully	"NestCafe", "Mocha", ingredients, 220,40,10	Success	220, 40, 10, 2	220, 40, 10, 2	Code should terminated successfully	Success	Rhea Sidana



TestID	Scenario	Precondition	Input	Success/ Fail	Output	Expected Output	Post Condition	Result	Written By
2	To test the CoffeeMaker class	Code should be executed successfully	"NestCafe", "Mocha", ingredients, 220,40,10	Fail	220, 40, 10, 2	230, 40, 10, 2	Code should terminated successfully	Fail	Rhea Sidana

```
42      */
43      @Test
44      public void testGetPower() {
45          System.out.println("getPower");
46          //CoffeeMaker(String name,String type,String[] ingredients,int power,int price,int time)
47          String[] ingredients = {"Coffee","Milk"};
48          CoffeeMaker instance = new CoffeeMaker("NestCafe","Mocha",ingredients,220,40,10);
49          int expResult = 230;
50          int result = instance.getPower();
51          //System.out.println("Result"+result);
52          assertEquals(expResult, result);
53          // TODO review the generated test code and remove the default call to fail.
54          //fail("The test case is a prototype.");
55      }
56
```

```
coffeemakermachine.CoffeeMakerTest > testGetPower >
Test Results
coffeemakermachine.CoffeeMakerTest
Tests passed: 75.00 %
3 tests passed, 1 test failed. (0.14 s)
coffeemakermachine.CoffeeMakerTest Failed
getPower
getPrice
getIngredientsCount
getTime
```

TestID	Scenario	Precondition	Input	Success/ Fail	Output	Expected Output	Post Condition	Result	Written By
2	To test the CoffeeMaker class	Code should be executed successfully	"NestCafe", "Mocha", ingredients, 220,60,10	Fail	220, 0, 10, 2	230, 40, 10, 2	Code should terminated success fully	Fail	Rhea Sidana

```
60      @Test
61      public void testGetPrice() {
62          System.out.println("getPrice");
63          //CoffeeMaker(String name,String type,String[] ingredients,int power,int price,int time)
64          String[] ingredients = {"Coffee","Milk"};
65          CoffeeMaker instance = new CoffeeMaker("NestCafe","Mocha",ingredients,220,60,10);
66          int expResult = 40;
67          int result = instance.getPrice();
68          //System.out.println("Result"+result);
69          assertEquals(expResult, result);
70          // TODO review the generated test code and remove the default call to fail.
71          //fail("The test case is a prototype.");
72      }
73
74      /**
```

```
coffeemakermachine.CoffeeMakerTest > testGetPower > expResult >
Test Results
coffeemakermachine.CoffeeMakerTest
Tests passed: 50.00 %
2 tests passed, 2 tests failed. (0.14 s)
coffeemakermachine.CoffeeMakerTest Failed
testGetPower Failed: expected:<230> but was:<220>
-expected:<230> but was:<220>
-junit.framework.AssertionFailedError
-at coffeemakermachine.CoffeeMakerTest.testGetPower(CoffeeMakerTest.java:52)
testGetPrice Failed: expected:<40> but was:<0>
-expected:<40> but was:<0>
-junit.framework.AssertionFailedError
-at coffeemakermachine.CoffeeMakerTest.testGetPrice(CoffeeMakerTest.java:69)
getPower
getPrice
getIngredientsCount
getTime
```

AA1. Demonstrate test driven development through a class MyArray. Write a linear search method to find out the largest element of the array. Design a test class to test this method, using JUNIT framework. Record both test cases in proper test case templates.

Solution:

Project: Searching

Class: MyArray

MyArray.java

```
package searching;

public class MyArray {
    private int[] arr;

    //constructor
    public MyArray(int[] arr){
        this.arr = arr;
    }

    //search the largest element in the array
    public int linearSearch(){
        if(arr.length==0)
            return -1;
        int max=0;
        for(int i=0;i<arr.length;i++){
            if(max<arr[i]){
```

```
            max=arr[i];
        }
    }
    return max;
  }
}
```

MyArrayTest.java

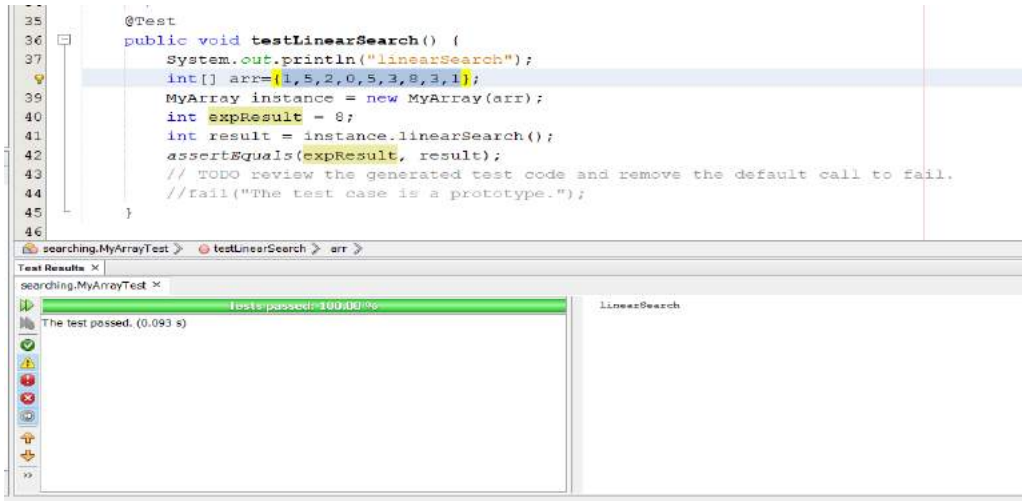
```
package searching;

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 MyArrayTest {

  public MyArrayTest() {
  }

  @BeforeClass
```

```
public static void setUpClass() {
}

@AfterClass
public static void tearDownClass() {
}

@Before
public void setUp() {
}

@After
public void tearDown() {
}

/**
 * Test of linearSearch method, of class MyArray.
 */
@Test
public void testLinearSearch() {
    System.out.println("linearSearch");
    int[] arr={1,5,2,0,5,3,8,3,1};
    MyArray instance = new MyArray(arr);
    int expResult = 8;
    int result = instance.linearSearch();
```

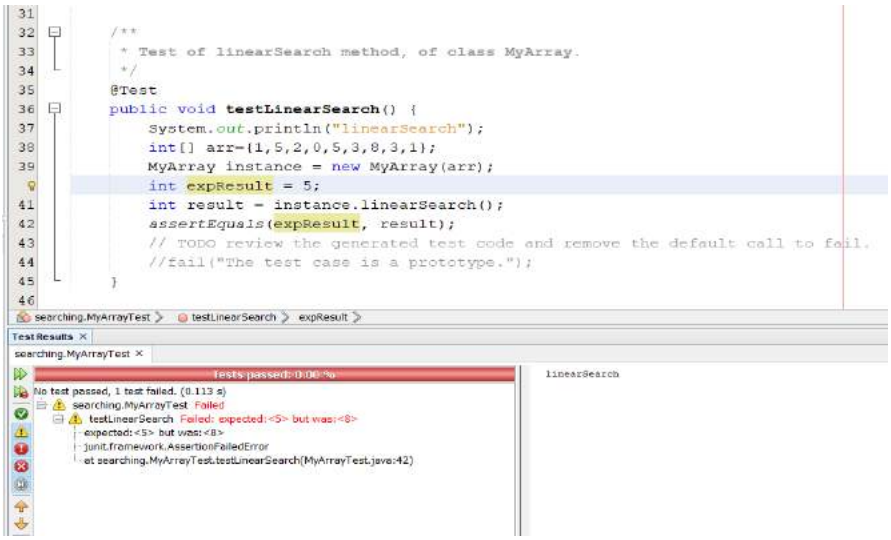
```
        assertEquals(expResult, result);

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

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

    }


}
```

Output:

TestID	Scenario	Precondition	Input	Success/ Fail	Output	Expected Output	Post Condition	Result	Written By
1	To test the linearSearch method	Code should be executed successfully	1,5,2,0,5,3,8,3,1	Success	8	8	Code should terminated successfully	Success	Rhea Sidana



TestID	Scenario	Precondition	Input	Success/ Fail	Output	Expected Output	Post Condition	Result	Written By
2	To test the linearSearch method	Code should be executed successfully	1,5,2,0,5,3,8,3,1	Fail	8	5	Code should terminated success fully	Fail	Rhea Sidana



BP1. Construct proper test cases for a Bank ATM Machine. The test scenarios can be as follows:

- **Withdraw money from an ATM.**
- **Deposit money into an ATM.**
- **Transaction failed due to not enough cash**

Solution:

Test Case ID: T001	Module Name: Bank ATM Machine
Section 1: Before Execution	Section 2: After Execution
Purpose: Verify Withdraw money from ATM	**Execution History:** User's information should be correct.
Pre Conditions: Machine should be in working condition.	**Results:** Machine working properly
Inputs: Pin entered should be correct and the amount should be less than or equal to 30,000.	**If fails, any possible reason (optional):** If pin entered is incorrect, transaction cancels. And if the amount needed is greater than the amount available in the machine.
Expected Output: Money is withdrawn.	**Any Other Observation/s:** None.
Post Conditions: Other options related to the banking are displayed after printing the transaction receipt.	**Any Suggestions:** Fast cash withdrawn option should be available and the amount withdrawn limit should be increased.
Written By: Rhea Sidana	**Run By:** Rhea Sidana
Date: 18 September, 2021	**Date:** 18 September, 2021

Test Case ID: T002	Module Name: Bank ATM Machine
Section 1: Before Execution	Section 2: After Execution
Purpose: Verify deposit money into a ATM.	**Execution History:** User's information should be correct.
Pre Conditions: Machine should be in working condition.	**Results:** Machine working properly
Inputs: Pin entered should be correct and the amount of money to be deposited.	**If fails, any possible reason (optional):** If pin entered is incorrect, transaction cancels.
Expected Output: Money is deposited.	**Any Other Observation/s:** None.
Post Conditions: Other options related to the banking are displayed after printing the transaction receipt.	**Any Suggestions:** None.
Written By: Rhea Sidana	**Run By:** Rhea Sidana
Date: 18 September, 2021	**Date:** 18 September, 2021

Test Case ID: T003	Module Name: Bank ATM Machine
Section 1: Before Execution	Section 2: After Execution
Purpose: Verify transaction failed due to not enough cash present in the ATM machine.	**Execution History:** User's information should be correct.
Pre Conditions: Machine should be in working condition.	**Results:** Machine working properly

Inputs: Pin entered should be correct and the amount of money to be withdrawn should be less than or equal to 30,000.	**If fails, any possible reason (optional):** If pin entered is incorrect, transaction cancels. And if the amount needed is greater than the amount available in the machine.
Expected Output: Money withdrawn fails.	**Any Other Observation/s:** None.
Post Conditions: Error prompt is displayed, due to insufficient amount of money available in the ATM machine.	**Any Suggestions:** If amount of money available in the machine is less than 30,000 then before inputting withdrawal amount, amount available should be displayed.
Written By: Rhea Sidana	**Run By:** Rhea Sidana
Date: 18 September, 2021	**Date:** 18 September, 2021

BP2. Demonstrate test driven development through a project Arithmetic. The project should contain classes for Addition, Subtraction, Multiplication, Division of two parameters of different types .Through JUnit create corresponding Test classes for each class. Aggregate these different tests in a test suite and execute the same.

Solution:

Project: Arithmetic

Addition.java

```
package arithmetic;


public class Addition {

   private int num1,num2;

   private double no1,no2;


   //constructor

   public Addition(int num1,int num2){

      this.num1 = num1;

      this.num2 = num2;

   }

   public Addition(double no1,double no2){

      this.no1 = no1;

      this.no2 = no2;

   }


   //add method

   public int add(){
```

```
        return (num1+num2);
    }
    public double addDouble(){
        return (no1+no2);
    }

}
```

Subtraction.java

```
package arithmetic;

public class Subtraction {
    private int num1,num2;
    private double no1,no2;

    //constructor
    public Subtraction(int num1,int num2){
        this.num1 = num1;
        this.num2 = num2;
    }
    public Subtraction(double no1,double no2){
        this.no1 = no1;
        this.no2 = no2;
    }
```

```
    //subtract method
    public int subtract(){
        return (num1-num2);
    }
    public double subtractDouble(){
        return (no1-no2);
    }
}
```

Multiplication.java

```
package arithmetic;

public class Multiplication {
    private int num1,num2;
    private double no1,no2;

    //constructor
    public Multiplication(int num1,int num2){
        this.num1 = num1;
        this.num2 = num2;
    }
    public Multiplication(double no1,double no2){
        this.no1 = no1;
```

```
        this.no2 = no2;
    }

    //multiply method
    public int multiply(){
        return (num1*num2);
    }
    public double multiplyDouble(){
        return (no1*no2);
    }

}
```

Division.java

```
package arithmetic;

public class Division {
    private int num1,num2;
    private double no1,no2;

    //constructor
    public Division(int num1,int num2){
        this.num1 = num1;
        this.num2 = num2;
```

```
    }
    public Division(double no1,double no2){
        this.no1 = no1;
        this.no2 = no2;
    }

    //divide method
    public int divide(){
        return (num1/num2);
    }
    public double divideDouble(){
        return (no1/no2);
    }
}
```

AdditionTest.java

```
package arithmetic;

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 AdditionTest {

    public AdditionTest() {
    }

    @BeforeClass
    public static void setUpClass() {
    }

    @AfterClass
    public static void tearDownClass() {
    }

    @Before
    public void setUp() {
    }

    @After
    public void tearDown() {
    }

    /**
     * Test of add method, of class Addition.
     */
```

```
@Test
public void testAdd() {
    System.out.println("add");
    Addition instance = new Addition(4,6);
    int expResult = 10;
    int result = instance.add();
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default call to fail.
    //fail("The test case is a prototype.");
}

/**
 * Test of addDouble method, of class Addition.
 */
@Test
public void testAddDouble() {
    System.out.println("addDouble");
    Addition instance = new Addition(6.5,3.5);
    double expResult = 10.0;
    double result = instance.addDouble();
    assertEquals(expResult, result, 0.0);
    // TODO review the generated test code and remove the default call to fail.
    //fail("The test case is a prototype.");
}
```

```
}
```

SubtractionTest.java

```
package arithmetic;

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 SubtractionTest {

    public SubtractionTest() {
    }

    @BeforeClass
    public static void setUpClass() {
    }

    @AfterClass
    public static void tearDownClass() {
    }
```

```
@Before
public void setUp() {
}

@After
public void tearDown() {
}

/**
 * Test of subtract method, of class Subtraction.
 */
@Test
public void testSubtract() {
    System.out.println("subtract");
    Subtraction instance = new Subtraction(9,3);
    int expResult = 6;
    int result = instance.subtract();
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default call to fail.
    //fail("The test case is a prototype.");
}

/**
 * Test of subtractDouble method, of class Subtraction.
```

```
    */
    @Test
    public void testSubtractDouble() {
        System.out.println("subtractDouble");
        Subtraction instance = new Subtraction(9.0,6.0);
        double expResult = 3.0;
        double result = instance.subtractDouble();
        assertEquals(expResult, result, 0.0);
        // TODO review the generated test code and remove the default call to fail.
        //fail("The test case is a prototype.");
    }

}
```

MultiplicationTest.java

```
package arithmetic;

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 MultiplicationTest {

    public MultiplicationTest() {
    }

    @BeforeClass
    public static void setUpClass() {
    }

    @AfterClass
    public static void tearDownClass() {
    }

    @Before
    public void setUp() {
    }

    @After
    public void tearDown() {
    }

    /**
     * Test of multiply method, of class Multiplication.
     */
    @Test
```

```
    public void testMultiply() {
        System.out.println("multiply");
        Multiplication instance = new Multiplication(5,4);
        int expResult = 20;
        int result = instance.multiply();
        assertEquals(expResult, result);
        // TODO review the generated test code and remove the default call to fail.
        //fail("The test case is a prototype.");
    }

    /**
     * Test of multiplyDouble method, of class Multiplication.
     */
    @Test
    public void testMultiplyDouble() {
        System.out.println("multiplyDouble");
        Multiplication instance = new Multiplication(2.5,4.5);
        double expResult = 11.25;
        double result = instance.multiplyDouble();
        assertEquals(expResult, result, 0.0);
        // TODO review the generated test code and remove the default call to fail.
        //fail("The test case is a prototype.");
    }

}
```

DivisionTest.java

```
package arithmetic;

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 DivisionTest {

    public DivisionTest() {
    }

    @BeforeClass
    public static void setUpClass() {
    }

    @AfterClass
    public static void tearDownClass() {
    }
```

```
@Before
public void setUp() {
}

@After
public void tearDown() {
}

/**
 * Test of divide method, of class Division.
 */
@Test
public void testDivide() {
    System.out.println("divide");
    Division instance = new Division(6,2);
    int expResult = 3;
    int result = instance.divide();
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default call to fail.
    //fail("The test case is a prototype.");
}

/**
 * Test of divideDouble method, of class Division.
 */
```

```
    @Test
    public void testDivideDouble() {
        System.out.println("divideDouble");
        Division instance = new Division(6.6,2.0);
        double expResult = 3.3;
        double result = instance.divideDouble();
        assertEquals(expResult, result, 0.0);
        // TODO review the generated test code and remove the default call to fail.
        //fail("The test case is a prototype.");
    }

}
```

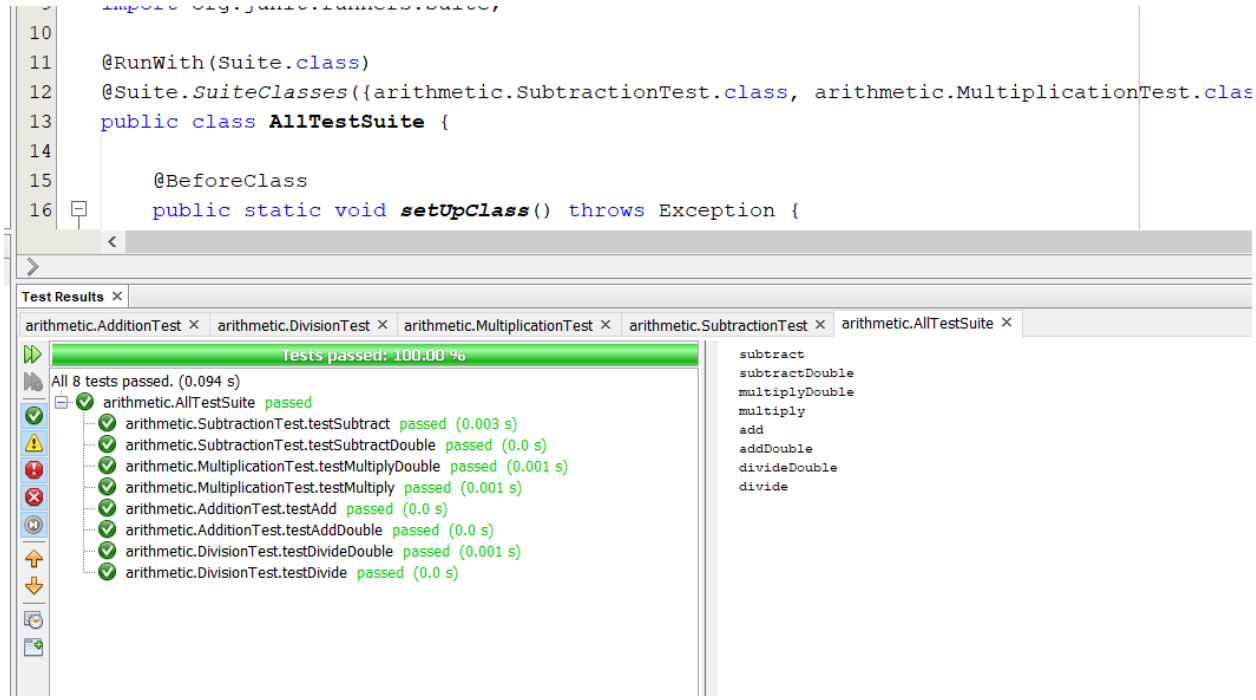
AllTestSuite.java

```
package arithmetic;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.runner.RunWith;
import org.junit.runners.Suite;

@RunWith(Suite.class)
```

```
@Suite.SuiteClasses({arithmetic.SubtractionTest.class, arithmetic.MultiplicationTest.class,
arithmetic.AdditionTest.class, arithmetic.DivisionTest.class})

public class AllTestSuite {


    @BeforeClass

    public static void setUpClass() throws Exception {

    }


    @AfterClass

    public static void tearDownClass() throws Exception {

    }


    @Before

    public void setUp() throws Exception {

    }


    @After

    public void tearDown() throws Exception {

    }


}
```

Output:

TestID	Scenario	Precondition	Input	Success/ Fail	Output	Expected Output	Post Condition	Result	Written By
1	To test the Addition class	Code should be executed successfully	Int: 4,6 Double: 6.5,3.5	Success Success	add(): 10 addDouble(): 10.0	add(): 10 addDouble():10.0	Code should terminated successfully	Success Success	Rhea Sidana
2	To test the Subtraction class	Code should be executed successfully	Int: 9,3 Double: 9.0,6.0	Success Success	subtract():6 subtractDouble():3.0	subtract():6 subtractDouble():3.0	Code should terminated successfully	Success Success	Rhea Sidana
3	To test the Multiplication class	Code should be executed successfully	Int: 5,4 Double: 2.5,4.5	Success Success	multiply():20 multiplyDouble():11.25	multiply():20 multiplyDouble():11.25	Code should terminated successfully	Success Success	Rhea Sidana
4	To test the Division class	Code should be executed successfully	Int: 6,2 Double: 6.6,2.0	Success Success	divide():3 divideDouble():3.3	divide():3 divideDouble():3.3	Code should terminated successfully	Success Success	Rhea Sidana



TestID	Scenario	Precondition	Input	Success/ Fail	Output	Expected Output	Post Condition	Result	Written By
5	To test the Addition class	Code should be executed successfully	Int: 4,6 Double: 6.5,3.5	Success Success	add(): 10 addDouble(): 10.0	add(): 10 addDouble():10.0	Code should terminated successfully	Success Success	Rhea Sidana
6	To test the Subtraction class	Code should be executed successfully	Int: 9,3 Double: 9.0,6.0	Success Success	subtract():6 subtractDouble():3.0	subtract():6 subtractDouble():3.0	Code should terminated successfully	Success Success	Rhea Sidana
7	To test the Multiplication class	Code should be executed successfully	Int: 5,4 Double: 2.5,4.5	Fail Success	multiply():20 multiplyDouble():11.25	multiply():30 multiplyDouble():11.25	Code should terminated successfully	Fail Success	Rhea Sidana

8	To test the Division class	Code should be executed successfully	Int: 6,2 Double: 6.6,2.0	Success Fail	divide( ):3 divide Double ():3.3	divide():3 divideDouble():5	Code should terminated successfully	Success Fail	Rhea Sidana

```
import org.junit.runner.RunWith;
import org.junit.runners.Suite;

@RunWith(Suite.class)
@Suite.SuiteClasses({arithmetic.SubtractionTest.class, arithmetic.MultiplicationTest.class, arit
public class AllTestSuite {

    @BeforeClass
    public static void setUpClass() throws Exception {
```

Test Results

arithmetic.AdditionTest | arithmetic.DivisionTest | arithmetic.MultiplicationTest | arithmetic.SubtractionTest | arithmetic.AllTestSuite

Tests passed: 75.00 %

6 tests passed, 2 tests failed. (0.107 s)

- arithmetic.AllTestSuite Failed
  - arithmetic.SubtractionTest.testSubtract passed (0.003 s)
  - arithmetic.SubtractionTest.testSubtractDouble passed (0.0 s)
  - arithmetic.MultiplicationTest.testMultiplyDouble passed (0.0 s)
  - arithmetic.MultiplicationTest.testMultiply Failed: expected:<30> but was:<20>
    - expected:<30> but was:<20>
    - junit.framework.AssertionFailedError
    - at arithmetic.MultiplicationTest.testMultiply(MultiplicationTest.java:41)
  - arithmetic.AdditionTest.testAdd passed (0.001 s)
  - arithmetic.AdditionTest.testAddDouble passed (0.001 s)
  - arithmetic.DivisionTest.testDivideDouble Failed: expected:<5.0> but was:<3.3>
    - expected:<5.0> but was:<3.3>
    - junit.framework.AssertionFailedError
    - at arithmetic.DivisionTest.testDivideDouble(DivisionTest.java:55)
  - arithmetic.DivisionTest.testDivide passed (0.001 s)

subtract
subtractDouble
multiplyDouble
multiply
add
addDouble
divideDouble
divide

BP3. Demonstrate test driven development through a project ProductRepository. The project should contain a class ProductRepository with appropriate methods for CRUD operations on a ProductRepository instance .Through JUnit create corresponding Test classes for each class. Aggregate these different tests in a test suite and execute the same.

Solution:

Project: ProductRepository

Classes: product, ProductRepository

Product.java

```
package productrepository;


public class Product {

   private int price;

   private String name;

   private String size;


   //constructor

   public Product(int price, String name, String size){

      this.name=name;

      this.size=size;

      this.price=price;

   }


   //getter and setter for price

   public void setPrice(int price){

      this.price=price;
```

```
    }
    public int getPrice(){
        return this.price;
    }

    //getter and setter for name
    public void setName(String name){
        this.name=name;
    }
    public String getName(){
        return this.name;
    }

    //getter and setter for size
    public void setSize(String size){
        this.size=size;
    }
    public String getSize(){
        return this.size;
    }
}
```

ProductRepository.java

```
package productrepository;


import java.util.*;


public class ProductRepository {
   LinkedList<Product> productList = new LinkedList<Product>();


   //CRUD: addProduct, viewProduct, updateProduct, deleteProduct
   //addProduct
   public int addProduct(Product product){
      productList.add(0,product);
      if(productList.contains(product)){
         return 1;
      }
      return 0;
   }


   //viewProduct
   public int viewProduct(int productId){
      if(productList.size() > productId){
         Product product  = productList.get(productId);
         System.out.println("Product Name : "+product.getName());
         System.out.println("Product Size : "+product.getSize());
         System.out.println("Product Price : "+product.getPrice());
         return 1;
```

```
    }
    return 0;
}

//updateProduct
public int updateProduct(int productId,Product product){
    if(productList.size() > productId){
        this.deleteProduct(productId);
        productList.add(productId,product);
        return 1;
    }
    return 0;
}

//deleteProduct
public int deleteProduct(int productId){
    if(productList.size() > productId){
        productList.remove(productId);
        return 1;
    }
    return 0;
}

/**
 * @param args the command line arguments
```

```
     */
    /*public static void main(String[] args) {
        // TODO code application logic here
    }*/

}
```

ProductRespositoryTest1.java

```
//addProductTest

package productrepository;
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 ProductRespositoryTest1 {

    public ProductRespositoryTest1() {
    }

    @BeforeClass
    public static void setUpClass() {
```

```
}

@AfterClass
public static void tearDownClass() {
}

@Before
public void setUp() {
}

@After
public void tearDown() {
}

// TODO add test methods here.
// The methods must be annotated with annotation @Test. For example:
//
// @Test
// public void hello() {}
@Test
public void addProductTest(){
    System.out.println("ADD Product ");
    //product 1
    Product product  = new Product(350,"T-Shirt","XL");
    ProductRepository productRepository = new ProductRepository();
```

```
        int result  = productRepository.addProduct(product);
        int expected = 1;
        assertEquals(expected, result);
    }
}
```

ProductRepositoryTest2.java

//viewProductTest

```
package productrepository;

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 ProductRepositoryTest2 {

    public ProductRepositoryTest2() {
    }

    @BeforeClass
    public static void setUpClass() {
```

```
}

@AfterClass
public static void tearDownClass() {
}

@Before
public void setUp() {
}

@After
public void tearDown() {
}

// TODO add test methods here.
// The methods must be annotated with annotation @Test. For example:
//
// @Test
// public void hello() {}
@Test
public void viewProductTest(){
    System.out.println("VIEW Product ");
    //product 1
    Product product  = new Product(350,"T-Shirt","XL");
    ProductRepository productRepository = new ProductRepository();
```

```
        int result  = productRepository.addProduct(product);
        result = productRepository.viewProduct(0);
        int expected = 1;
        assertEquals(expected, result);
    }
}
```

ProductRepositoryTest3.java

```
//updateProductTest

package productrepository;

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 ProductRepositoryTest3 {

    public ProductRepositoryTest3() {
    }
```

```
@BeforeClass

public static void setUpClass() {

}


@AfterClass

public static void tearDownClass() {

}


@Before

public void setUp() {

}


@After

public void tearDown() {

}


// TODO add test methods here.

// The methods must be annotated with annotation @Test. For example:

//

// @Test

// public void hello() {}

@Test

public void updateProductTest(){

    System.out.println("UPDATE Product ");

    //product 1
```

```
        Product product  = new Product(350,"T-Shirt","XL");
        ProductRepository productRepository = new ProductRepository();
        int result  = productRepository.addProduct(product);

        //product 2
        product  = new Product(500,"Skirt","L");
        result  = productRepository.addProduct(product);

        product  = new Product(350,"T-Shirt","XXL");
        result = productRepository.updateProduct(1, product);
        int expected = 1;
        assertEquals(expected, result);
    }
}
```

ProductRepositoryTest4.java

```
//deleteProductTest
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package productrepository;

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.*;


/**

 *

 * @author MAX

 */

public class ProductRepositoryTest4 {


    public ProductRepositoryTest4() {

    }


    @BeforeClass

    public static void setUpClass() {

    }


    @AfterClass

    public static void tearDownClass() {

    }


    @Before

    public void setUp() {
```

```
}

@After
public void tearDown() {
}

// TODO add test methods here.
// The methods must be annotated with annotation @Test. For example:
//
// @Test
// public void hello() {}
@Test
public void deleteProductTest(){
   System.out.println("DELETE Product ");
   //product 1
   Product product  = new Product(350,"T-Shirt","XL");
   ProductRepository productRepository = new ProductRepository();
   int result  = productRepository.addProduct(product);

   //product 2
   product  = new Product(500,"Skirt","L");
   result  = productRepository.addProduct(product);

   result = productRepository.deleteProduct(1);
   int expected = 1;
```

```
        assertEquals(expected, result);
    }
}
```

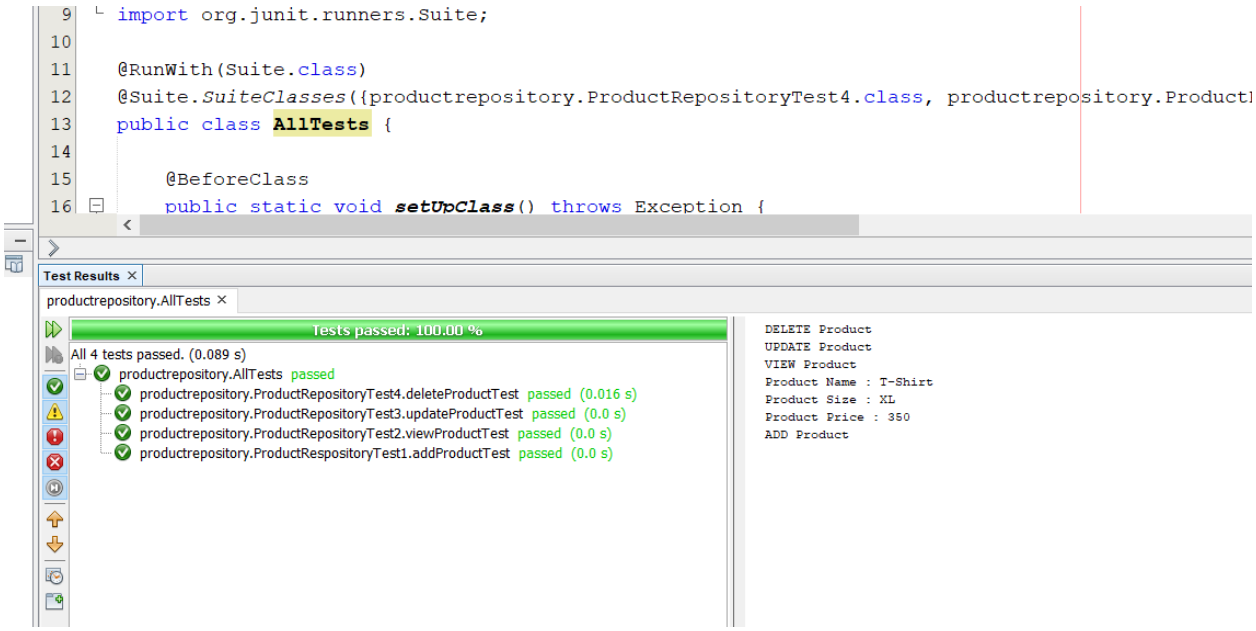
AllTests.java

```
//TestSuite

package productrepository;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.runner.RunWith;
import org.junit.runners.Suite;

@RunWith(Suite.class)
@Suite.SuiteClasses({productrepository.ProductRepositoryTest4.class,
productrepository.ProductRepositoryTest3.class, productrepository.ProductRepositoryTest2.class,
productrepository.ProductRespositoryTest1.class})
public class AllTests {

    @BeforeClass
    public static void setUpClass() throws Exception {
    }
```

```
    @AfterClass
    public static void tearDownClass() throws Exception {
    }

    @Before
    public void setUp() throws Exception {
    }

    @After
    public void tearDown() throws Exception {
    }

}
```

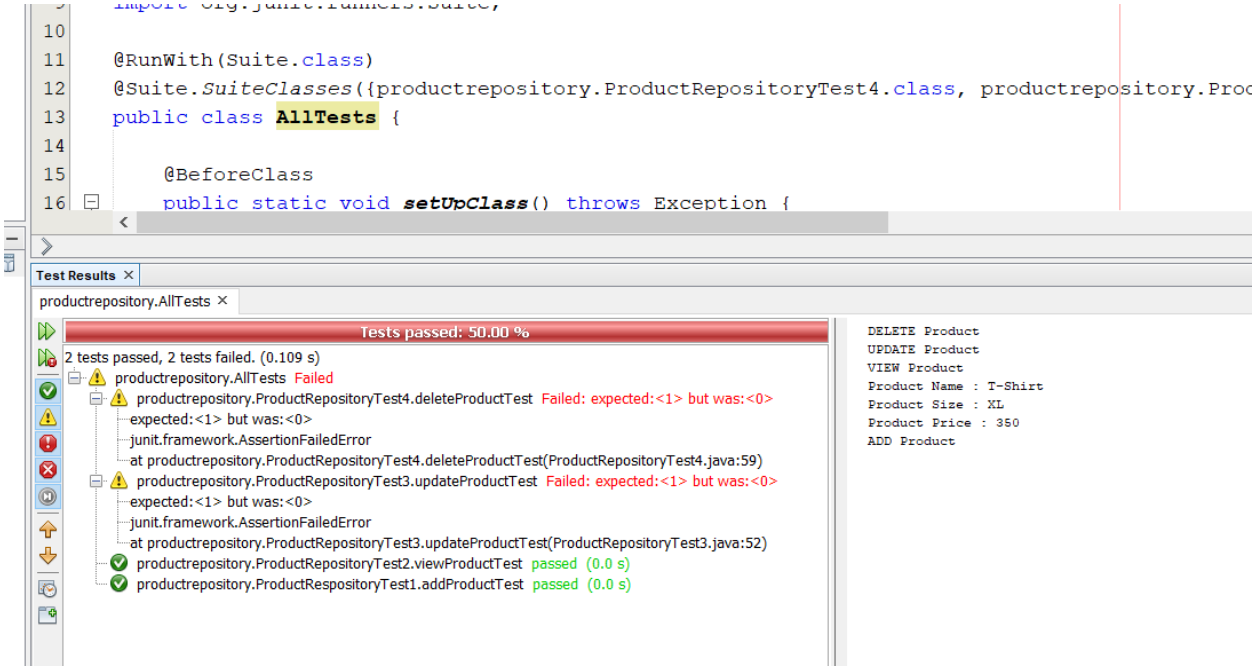
Output:

TestID	Scenario	Precondition	Input	Success/ Fail	Output	Expected Output	Post Condition	Result	Written By
1	To test the addProduct method	Code should be executed successfully	Product(350, "T-Shirt", "XL")	Success	1	1	Code should terminated successfully	Success	Rhea Sidana
2	To test the viewProduct method	Code should be executed successfully	0	Success	1 Product Name : T-Shirt Product Size : XL Product Price : 350	1 Product Name : T-Shirt Product Size : XL Product Price : 350	Code should terminated successfully	Success	Rhea Sidana
3	To test the updateProduct method	Code should be executed successfully	1, Product(350, "T-Shirt", "XXL")	Success	1	1	Code should terminated successfully	Success	Rhea Sidana
4	To test the Division class	Code should be executed successfully	1	Success	1	1	Code should terminated successfully	Success	Rhea Sidana



TestID	Scenario	Precondition	Input	Success/ Fail	Output	Expected Output	Post Condition	Result	Written By
5	To test the addProduct method	Code should be executed successfully	Product(350, "T-Shirt", "XL")	Success	1	1	Code should terminated successfully	Success	Rhea Sidana
6	To test the viewProduct method	Code should be executed successfully	0	Success	1 Product Name : T-Shirt Product Size : XL Product Price : 350	1 Product Name : T-Shirt Product Size : XL Product Price : 350	Code should terminated successfully	Success	Rhea Sidana
7	To test the updateProduct method	Code should be executed successfully	2, Product(350, "T-Shirt", "XXL")	Fail	0	1	Code should terminated successfully	Fail	Rhea Sidana

8	To test the deletePr oduct method	Code should be executed successfully	2	Fail	0	1	Code should terminate d success fully	Fail	Rhea Sidana



BP4. Imagine that due to current situation caused due to COVID-19, Delhi University has decided to grant undergraduate admissions in science programs based on the following rules:

a)Marks in Mathematics>=60

b) Marks in Physics>=50

c) Marks in Chemistry>=50

d) Total in all three subjects>=160 or Total in Maths & Physics>=120

If aggregate of an eligible candidate is greater than 225, he will be eligible for honors course. Write appropriate class to represent this situation. Through JUnit create corresponding Test classes for each class. Aggregate these different tests in a test suite and execute the same.

Solution:
Project: Admission
Class: UGCourse

UGCourse.java

```
package admission;


public class UGCourse {

    private float maths,physics,chemistry,total;


    //constructor

    public UGCourse(float maths,float physics, float chemistry){

        this.maths = maths;

        this.physics = physics;

        this.chemistry = chemistry;

        this.total = this.maths + this.chemistry + this.physics;

    }
```

```
//checking maths
public int mathCriteria(){
    if(this.maths>=60)
        return 1;
    return 0;
}

//checking physics
public int physicsCriteria(){
    if(this.physics>=50)
        return 1;
    return 0;
}

//checking chemistry
public int chemistryCriteria(){
    if(this.chemistry>=50)
        return 1;
    return 0;
}

//checking total
```

```
    public int totalCriteria(){
        float sumMathPhy = this.maths + this.physics;
        if(this.total>=160)
            return 1;
        else if (sumMathPhy>=120)
            return 1;
        return 0;
    }


    //checking course type : normal or honors
    public int courseCriteria(){
        if(this.mathCriteria()==1 && this.physicsCriteria()==1 && this.chemistryCriteria()==1
&& this.totalCriteria()==1){
            if(this.total>225){
                System.out.println("Honors");
                return 1;
            }
            else{
                System.out.println("Pass");
                return 2;
            }
        }
        return 0;
    }
```

```
    }
```

UGCourseMathTest.java

```
package admission;


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 UGCourseMathTest {


    public UGCourseMathTest() {
    }


    @BeforeClass
    public static void setUpClass() {
    }


    @AfterClass
    public static void tearDownClass() {
    }
```

```
@Before
public void setUp() {
}

@After
public void tearDown() {
}

// TODO add test methods here.
// The methods must be annotated with annotation @Test. For example:
//
// @Test
// public void hello() {}
@Test
public void mathCriteriaTest() {
    System.out.println("Math Criteria test");

    UGCourse myCourse = new UGCourse(80.0F,80.0F,80.0F);
    int result  = myCourse.mathCriteria();
    int expected = 1;
    assertEquals(expected,result);
}
```

```
}
```

UGCoursePhysicsTest.java

```
package admission;

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 UGCoursePhysicsTest {

    public UGCoursePhysicsTest() {
    }

    @BeforeClass
    public static void setUpClass() {
    }

    @AfterClass
    public static void tearDownClass() {
```

```
}

@Before
public void setUp() {
}

@After
public void tearDown() {
}

// TODO add test methods here.
// The methods must be annotated with annotation @Test. For example:
//
// @Test
// public void hello() {}
@Test
public void physicsCriteriaTest() {
    System.out.println("physics Criteria test");

    UGCourse myCourse = new UGCourse(80.0F,80.0F,80.0F);
    int result  = myCourse.physicsCriteria();
    int expected = 1;
    assertEquals(expected,result);
```

```
    }
}
```

UGCousreChemistryTest.java

```
package admission;

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 UGCousreChemistryTest {

    public UGCousreChemistryTest() {
    }

    @BeforeClass
    public static void setUpClass() {
    }

    @AfterClass
```

```
public static void tearDownClass() {
}

@Before
public void setUp() {
}

@After
public void tearDown() {
}

// TODO add test methods here.
// The methods must be annotated with annotation @Test. For example:
//
// @Test
// public void hello() {}
@Test
public void chemistryCriteriaTest() {
    System.out.println("chemistry Criteria test");

    UGCourse myCourse = new UGCourse(90.0F,90.0F,80.0F);
    int result  = myCourse.chemistryCriteria();
    int expected = 1;
```

```
        assertEquals(expected,result);
    }
}
```

UGCourseTotalTest.java

```
package admission;

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 UGCourseTotalTest {

    public UGCourseTotalTest() {
    }

    @BeforeClass
    public static void setUpClass() {
    }
```

```
@AfterClass
public static void tearDownClass() {
}

@Before
public void setUp() {
}

@After
public void tearDown() {
}

// TODO add test methods here.
// The methods must be annotated with annotation @Test. For example:
//
// @Test
// public void hello() {}
@Test
public void chemistryCriteriaTest() {
    System.out.println("total Criteria test");

    UGCourse myCourse = new UGCourse(90.0F,90.0F,80.0F);
    int result  = myCourse.totalCriteria();
```

```
        int expected = 1;
        assertEquals(expected,result);
    }
}
```

UGCourseCriteriaTest.java

```
package admission;

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 UGCourseCriteriaTest {

    public UGCourseCriteriaTest() {
    }

    @BeforeClass
    public static void setUpClass() {
    }
```

```
@AfterClass
public static void tearDownClass() {
}

@Before
public void setUp() {
}

@After
public void tearDown() {
}

// TODO add test methods here.
// The methods must be annotated with annotation @Test. For example:
//
// @Test
// public void hello() {}
@Test
public void chemistryCriteriaTest() {
    System.out.println("Course Criteria test");

    UGCourse myCourse = new UGCourse(90.0F,90.0F,80.0F);
```

```
        int result  = myCourse.courseCriteria();

        int expected = 1;

        assertEquals(expected,result);

    }

}
```

UGCourseTestSuite.java

```
package admission;


import org.junit.After;

import org.junit.AfterClass;

import org.junit.Before;

import org.junit.BeforeClass;

import org.junit.runner.RunWith;

import org.junit.runners.Suite;


@RunWith(Suite.class)

@Suite.SuiteClasses({admission.UGCourseTotalTest.class,
admission.UGCourseMathTest.class, admission.UGCousreChemistryTest.class,
admission.UGCoursePhysicsTest.class, admission.UGCourseCriteriaTest.class})

public class UGCourseTestSuite {


    @BeforeClass

    public static void setUpClass() throws Exception {
```

```
    }

    @AfterClass
    public static void tearDownClass() throws Exception {
    }

    @Before
    public void setUp() throws Exception {
    }

    @After
    public void tearDown() throws Exception {
    }

}
```

Output:

TestID	Scenario	Precondition	Input	Success/ Fail	Output	Expected Output	Post Condition	Result	Written By
1	To test the mathCriteria method	Code should be executed successfully	UGCourse(80.0F,80.0F,80.0F)	Success	1	1	Code should terminated successfully	Success	Rhea Sidana
2	To test the physicsCriteria method	Code should be executed successfully	UGCourse(80.0F,80.0F,80.0F)	Success	1	1	Code should terminated successfully	Success	Rhea Sidana
3	To test the chemistryCriteria method	Code should be executed successfully	UGCourse(90.0F,90.0F,80.0F)	Success	1	1	Code should terminated successfully	Success	Rhea Sidana
4	To test the totalCriteria method	Code should be executed successfully	UGCourse(90.0F,90.0F,80.0F)	Success	1	1	Code should terminated successfully	Success	Rhea Sidana
5	To test the CourseCriteria method	Code should be executed successfully	UGCourse(90.0F,90.0F,80.0F)	Success	1	1	Code should terminated successfully	Success	Rhea Sidana

```
10
11  @RunWith(Suite.class)
12  @Suite.SuiteClasses({admission.UGCourseTotalTest.class, admission.UGCourseMathTe
13  public class UGCourseTestSuite {
14
```

Test Results

admission.UGCourseMathTest | admission.UGCoursePhysicsTest | admission.UGCousreChemistryTest | admission.UGCourseTotalTest | admission.UGCou

Tests passed: 100.00 %

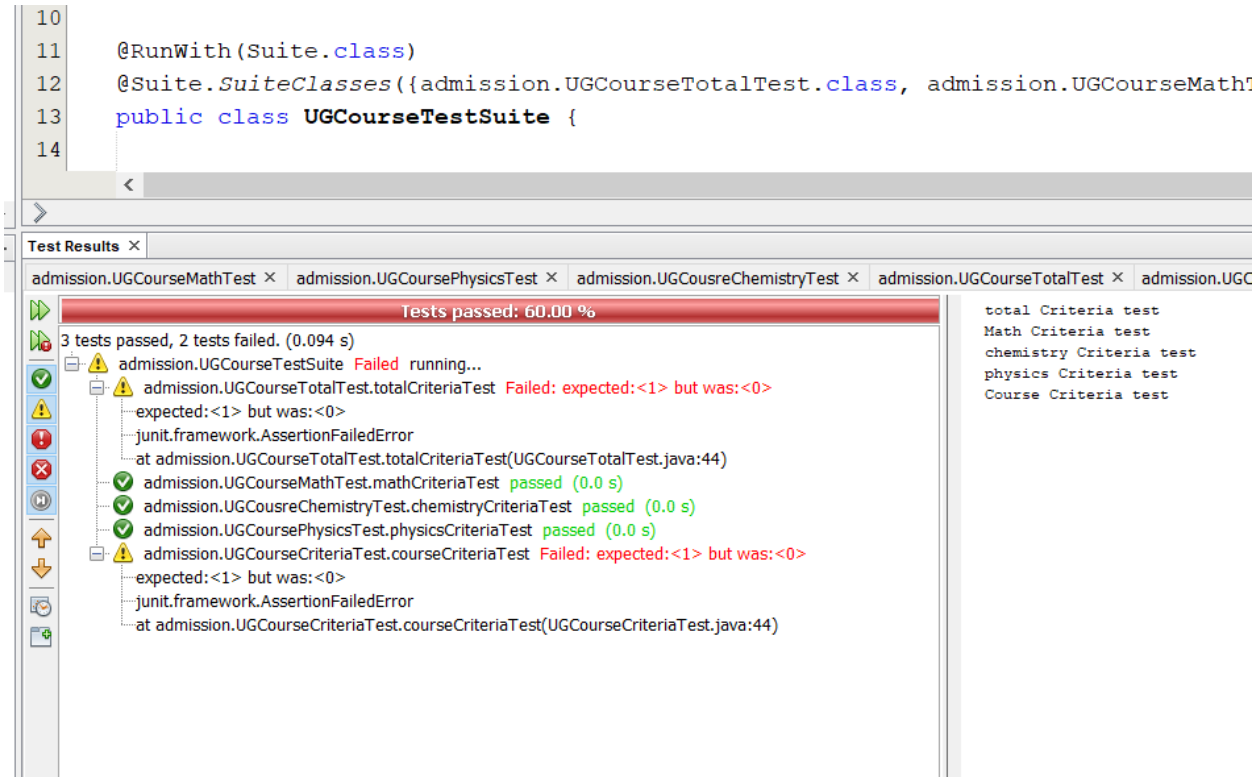
All 5 tests passed. (0.11 s)

- admission.UGCourseTestSuite passed
  - admission.UGCourseTotalTest.chemistryCriteriaTest passed (0.0 s)
  - admission.UGCourseMathTest.mathCriteriaTest passed (0.0 s)
  - admission.UGCousreChemistryTest.chemistryCriteriaTest passed (0.0 s)
  - admission.UGCoursePhysicsTest.physicsCriteriaTest passed (0.0 s)
  - admission.UGCourseCriteriaTest.chemistryCriteriaTest passed (0.0 s)

```
total Criteria test
Math Criteria test
chemistry Criteria test
physics Criteria test
Course Criteria test
Honors
```

TestID	Scenario	Precondition	Input	Success/ Fail	Output	Expected Output	Post Condition	Result	Written By
6	To test the mathCriteria method	Code should be executed successfully	UGCourse(80.0F,80.0F,80.0F)	Success	1	1	Code should terminated successfully	Success	Rhea Sidana
7	To test the physicsCriteria method	Code should be executed successfully	UGCourse(80.0F,80.0F,80.0F)	Success	1	1	Code should terminated successfully	Success	Rhea Sidana
8	To test the chemistryCriteria method	Code should be executed successfully	UGCourse(90.0F,90.0F,80.0F)	Success	1	1	Code should terminated successfully	Success	Rhea Sidana

9	To test the totalCriteria method	Code should be executed successfully	UGCourse(60.0F,40.0F,50.0F)	Success	1	1	Code should terminated successfully	Success	Rhea Sidana
10	To test the CourseCriteria method	Code should be executed successfully	UGCourse(60.0F,40.0F,50.0F)	Success	1	1	Code should terminated successfully	Success	Rhea Sidana



CP1. Demonstrate test driven development through a project MessageUtility. The project should contain a class MessageUtil with appropriate methods for printing a Message .Through JUnit create corresponding Test classes and TestRunner classes to execute and test the above class.

Solution:

MessageUtility.java

```
package messageutility;


public class MessageUtility {

   private String message;


   //Constructor

   public MessageUtility(String message){

      this.message=message;

   }


   public String getMessage(){

      System.out.println("Printing the message : " + this.message);

      return this.message;

   }


   /**

    * @param args the command line arguments

    */

   /*

   public static void main(String[] args) {
```

```
        // TODO code application logic here
    }*/
}
```

MessageUtilityTest.java

```
package messageutility;

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 MessageUtilityTest {
    public MessageUtilityTest() {
    }

    @BeforeClass
    public static void setUpClass() {
    }

    @AfterClass
    public static void tearDownClass() {
    }
```

```
    @Before
    public void setUp() {
    }

    @After
    public void tearDown() {
    }

    /**
     * Test of getMessage method, of class MessageUtility.
     */
    @Test
    public void testGetMessage() {
        System.out.println("getMessage");
        MessageUtility instance = new MessageUtility("Hello! I am Rhea Sidana.");
        String expResult = "Hello! I am Rhea Sidana.";
        String result = instance.getMessage();
        assertEquals(expResult, result);
        // TODO review the generated test code and remove the default call to fail.
        //fail("The test case is a prototype.");
    }

}
```

MessageUtilityTestRunner.java

```
package messageutility;

import org.junit.runner.*;
import org.junit.runner.notification.*;

public class MessageUtilityTestRunner {
   public static void  main(String args[]){
      Result result = JUnitCore.runClasses(MessageUtilityTest.class);

      for(Failure fail : result.getFailures()){
         System.out.println("Failure occrured: "+fail.toString());
      }
      if(result.getFailureCount()==0)
      System.out.println("MessageUtility TestRunner class passed ");
   }
}
```

Output:

TestID	Scenario	Precondition	Input	Success/ Fail	Output	Expected Output	Post Condition	Result	Written By
1	To test the MessageUtility class	Code should be executed successfully	"Hello! I am Rhea Sidana."	Success	"Hello! I am Rhea Sidana ."	"Hello! I am Rhea Sidana."	Code should terminated successfully	Success	Rhea Sidana

2	To test the MessageUtility class	Code should be executed successfully	"Hello. I am Rhea Sidana."	Fails	"Hello. I am Rhea Sidana ."	"Hello! I am Rhea Sidana."	Code should terminated success fully	Fail	Rhea Sidana

```java
public class MessageUtilityTestRunner {
    public static void  main(String args[]){
        Result result = JUnitCore.runClasses(MessageUtilityTest.class);

        for(Failure fail : result.getFailures()){
            System.out.println("Failure occrured: "+fail.toString());
        }
        if(result.getFailureCount()==0)
        System.out.println("MessageUtility TestRunner class passed ");
    }
}
```

messageutility.MessageUtilityTestRunner > main > if (result.getFailureCount() == 0) >

Test Results | Output - MessageUtility (run)

```
run:
getMessage
Printing the message : Hello! I am Rhea Sidana.
MessageUtility TestRunner class passed
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
run:
getMessage
Printing the message : Hello. I am Rhea Sidana.
Failure occrured: testGetMessage(messageutility.MessageUtilityTest): expected:<Hello[!] I am Rhea Sidana.> but was:<Hello[.] I am Rhea Sidana.>
BUILD SUCCESSFUL (total time: 0 seconds)
```

CP2. Demonstrate test driven development through a project PrimeNumberChecker. The project should contain a class PrimeNumberChecker with appropriate methods for validating if an input number is prime or not on a PrimeNumberChecker instance .Through JUnit create corresponding Parameterized Test classes and TestRunner classes to execute the above test on atleast 5 numbers.

Solution:

PrimeNumberChecker.java

```
package primenumerchecker;


public class PrimeNumberChecker {
    private int number;


    //constructor
    public PrimeNumberChecker(int number){
        this.number=number;
    }


    //check if prime
    public boolean isPrime(){
        for(int i=2;i<=number/2;i++){
            if(number%i==0)
                return false;
        }
        return true;
    }
}
```

PrimeNumberCheckerParameterizedTest.java

```
package primenumerchecker;

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.*;
import org.junit.runner.*;
import org.junit.runners.*;
import java.util.*;

@RunWith(Parameterized.class)
public class PrimeNumberCheckerParameterizedTest {
    private int input;
    private boolean expected;
    private PrimeNumberChecker instance;

    public PrimeNumberCheckerParameterizedTest(int input,boolean expected) {
        this.input=input;
        this.expected=expected;
    }
```

```
@BeforeClass
public static void setUpClass() {
}

@AfterClass
public static void tearDownClass() {
}

@Before
public void setUp() {
    this.instance = new PrimeNumberChecker(input);
}

@After
public void tearDown() {
}

//defining parameters
@Parameterized.Parameters
public static Collection primeNumberCollections(){
    return Arrays.asList(new Object[][]{{2,true},{8,false},{3,true},{7,true},{12,false}});
}

/**
 * Test of isPrime method, of class PrimeNumberChecker.
```

```
 */
@Test
public void testIsPrime() {
    System.out.println("isPrime");
    //boolean expResult = false;
    boolean result = this.instance.isPrime();
    assertEquals(expected, result);
    // TODO review the generated test code and remove the default call to fail.
    //fail("The test case is a prototype.");
}

}
```

Output

TestID	Scenario	Precondition	Input	Success/ Fail	Output	Expected Output	Post Condition	Result	Written By
1	To test the PrimeNumberChecker class	Code should be executed successfully	2,8,3,7,12	Success	true,false,true,true,false	true,false,true,true,false	Code should terminated successfully	Success	Rhea Sidana
2	To test the PrimeNumberChecker class	Code should be executed successfully	18,8,3,21,12	Fails	false,false,true,false,false	true,false,true,true,false	Code should terminated successfully	Fail	Rhea Sidana

```
55        boolean result = this.instance.isPrime();
```

primenumerchecker.PrimeNumberCheckerParameterizedTest > primeNumberCollections >

Test Results × | Output - PrimeNumerChecker (test)

messageutility.MessageUtilityTest × | primenumerchecker.PrimeNumberCheckerParameterizedTest ×

Tests passed: 100.00 %

All 5 tests passed. (0.086 s)

- primenumerchecker.PrimeNumberCheckerParameterizedTest passed
  - testIsPrime[0] passed (0.002 s)
  - testIsPrime[1] passed (0.001 s)
  - testIsPrime[2] passed (0.001 s)
  - testIsPrime[3] passed (0.001 s)
  - testIsPrime[4] passed (0.0 s)

```
isPrime
isPrime
isPrime
isPrime
isPrime
```

```
54        //boolean expResult = false;
55        boolean result = this.instance.isPrime();
```

primenumerchecker.PrimeNumberCheckerParameterizedTest > primeNumberCollections >

Test Results × | Output - PrimeNumerChecker (test)

messageutility.MessageUtilityTest × | primenumerchecker.PrimeNumberCheckerParameterizedTest ×

Tests passed: 60.00 %

3 tests passed, 2 tests failed. (0.133 s)

- primenumerchecker.PrimeNumberCheckerParameterizedTest Failed
  - testIsPrime[0] Failed: expected:<true> but was:<false>
    - expected:<true> but was:<false>
    - junit.framework.AssertionFailedError
    - at primenumerchecker.PrimeNumberCheckerParameterizedTest.testIsPrime(PrimeNumberCheckerParamet
  - testIsPrime[1] passed (0.002 s)
  - testIsPrime[2] passed (0.0 s)
  - testIsPrime[3] Failed: expected:<true> but was:<false>
    - expected:<true> but was:<false>
    - junit.framework.AssertionFailedError
    - at primenumerchecker.PrimeNumberCheckerParameterizedTest.testIsPrime(PrimeNumberCheckerParamet
  - testIsPrime[4] passed (0.0 s)

```
isPrime
isPrime
isPrime
isPrime
isPrime
```

CP3. Demonstrate test driven development through a project SquareChecker. The project should contain a class SquareChecker with appropriate methods for returning the square of an integer instance .Through JUnit create corresponding Parameterized Test classes and TestRunner classes to execute the above test on atleast 5 numbers. The test cases should be chosen using boundary value analysis technique for an input range from [100-500].

Solution:

SquareChecker.java

```
package squarechecker;


public class SquareChecker {
   private int number;


   //constuctor
   public SquareChecker(int number){
      this.number = number;
   }


   //return square
   public int retSquare(){
      return (number*number);
   }
}
```

SquareCheckerParameterizedTest.java

```
package squarechecker;
```

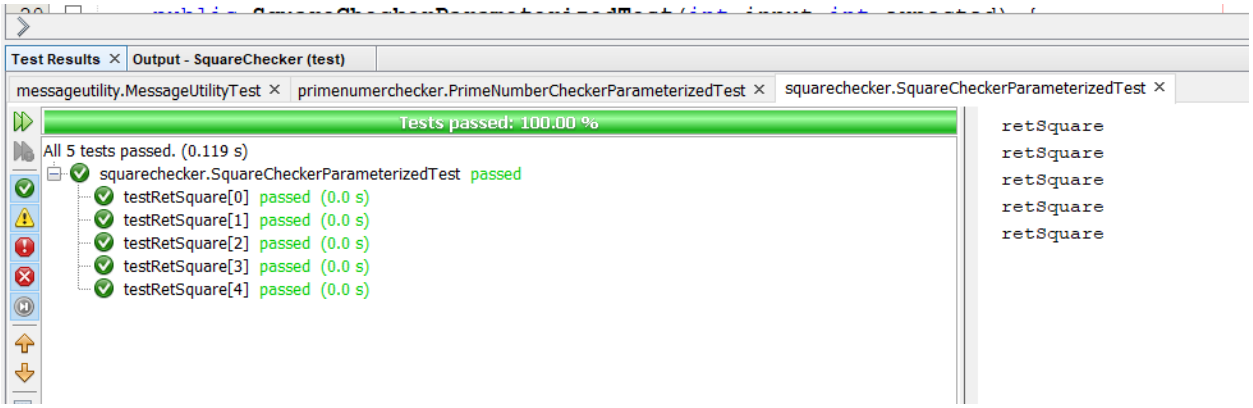
```
import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;
import org.junit.runner.*;
import org.junit.runners.Parameterized;
import java.util.*;

@RunWith(Parameterized.class)
public class SquareCheckerParameterizedTest {
    private int input;
    private int expected;
    private SquareChecker instance;

    public SquareCheckerParameterizedTest(int input,int expected) {
        this.input=input;
        this.expected=expected;
    }

    @BeforeClass
    public static void setUpClass() {
    }
```

```
@AfterClass
public static void tearDownClass() {
}

@Before
public void setUp() {
    instance=new SquareChecker(input);
}

@After
public void tearDown() {
}

@Parameterized.Parameters
public static Collection squareNumbers(){
    return Arrays.asList(new Integer[][]{{2,4},{4,16},{12,144},{16,256},{21,441}});
}

/**
 * Test of retSquare method, of class SquareChecker.
 */
@Test
public void testRetSquare() {
    System.out.println("retSquare");
    int result = instance.retSquare();
```

```
        assertEquals(expected, result);
        // TODO review the generated test code and remove the default call to fail.
        //fail("The test case is a prototype.");
    }

}
```

Output:

TestID	Scenario	Precondition	Input	Success/ Fail	Output	Expected Output	Post Condition	Result	Written By
1	To test the SquareChecker class	Code should be executed successfully	2,4,12,16,21	Success	4,16,144,256,441	4,16,144,256,441	Code should terminated successfully	Success	Rhea Sidana
2	To test the SquareChecker class	Code should be executed successfully	1,4,12,16,21	Fail	1,16,144,256,441	4,12,144,266,441		Fail	Rhea Sidana



```
@Test
```

squarechecker.SquareCheckerParameterizedTest > squareNumbers >

Test Results | Output - SquareChecker (test)

messageutility.MessageUtilityTest | primenumerchecker.PrimeNumberCheckerParameterizedTest | squarechecker.SquareCheckerParameterizedTest

Tests passed: 40.00 %

2 tests passed, 3 tests failed. (0.154 s)

- squarechecker.SquareCheckerParameterizedTest Failed
  - testRetSquare[0] Failed: expected:<4> but was:<1>
    - expected:<4> but was:<1>
    - junit.framework.AssertionFailedError
    - at squarechecker.SquareCheckerParameterizedTest.testRetSquare(SquareCheckerParameterizedTest.java:54)
  - testRetSquare[1] Failed: expected:<12> but was:<16>
    - expected:<12> but was:<16>
    - junit.framework.AssertionFailedError
    - at squarechecker.SquareCheckerParameterizedTest.testRetSquare(SquareCheckerParameterizedTest.java:54)
  - testRetSquare[2] passed (0.0 s)
  - testRetSquare[3] Failed: expected:<266> but was:<256>
    - expected:<266> but was:<256>
    - junit.framework.AssertionFailedError
    - at squarechecker.SquareCheckerParameterizedTest.testRetSquare(SquareCheckerParameterizedTest.java:54)
  - testRetSquare[4] passed (0.0 s)

```
retSquare
retSquare
retSquare
retSquare
retSquare
```

CP4. Demonstrate test driven development through a project Addition. The project should contain a class Addition with appropriate methods for returning the sum of two numbers. Through JUnit create corresponding Parameterized Test classes and TestRunner classes to execute the above test on a collection of test cases. The test cases should be chosen using boundary value analysis technique for an input range from [1-10] for the first number and [11-20] for the second number.

Solution:

Addition.java

```
package addition;


public class Addition {

   private int number1,number2;


   //constructor

   public Addition(int number1,int number2){

      this.number1=number1;

      this.number2=number2;

   }


   //add

   public int add(){

      return (number1+number2);

   }


   /**

    * @param args the command line arguments

    */
```

```
    /*
    public static void main(String[] args) {
        // TODO code application logic here
    }
    */
}
```

AdditionParameterizedBVATest.java

```
package addition;

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.*;
import org.junit.runner.*;
import org.junit.runners.*;
import java.util.*;

@RunWith(Parameterized.class)
public class AdditionParameterizedBVATest {
    private int input1, input2,expected;
    private Addition instance;
```

```
public AdditionParameterizedBVATest(int input1,int input2,int expected) {
    this.input1 = input1;
    this.input2 = input2;
    this.expected = expected;
}

@BeforeClass
public static void setUpClass() {
}

@AfterClass
public static void tearDownClass() {
}

@Before
public void setUp() {
    instance = new Addition(input1,input2);
}

@After
public void tearDown() {
}

//defining parameters
@Parameterized.Parameters
```

```
    public static Collection additionParams(){
        return Arrays.asList(new Integer[][]{{1,12,13},{5,16,21},{0,12,-1},{12,12,-1},{5,9,-1},{3,23,-1}});
    }

    /**
     * Test of add method, of class Addition.
     */
    @Test
    public void testAdd() {
        System.out.print("add");
        //Addition instance = null;
        //int expResult = 0;
        int result = this.instance.add();
        assertEquals(this.expected, result);
        // TODO review the generated test code and remove the default call to fail.
        //fail("Invalid Inputs.");
        if(this.expected==result)
            System.out.println(" passed");
        else System.out.println(" failed");

    }

}
```

AdditionTestRunner.java

```
package addition;

import org.junit.runner.*;

import org.junit.runner.notification.*;


public class AdditionTestRunner {

   public static void  main(String args[]){

      Result result = JUnitCore.runClasses(AdditionParameterizedBVATest.class);


      for(Failure fail : result.getFailures()){

         System.out.println("Failure occrured: "+fail.toString());

      }

      if(result.getFailureCount()==0)

      System.out.println("MessageUtility TestRunner class passed ");

   }


}
```

BVA:

Number1: [1-10],

Number2: [11-20]

Number1 values:

a. Minimum value = 1
b. Just above Minimum Value = 2
c. Middle value = 5

d. Just below Maximum value = 9
e. Maximum Value = 10

Number2 values:

a. Minimum value = 11
b. Just above Minimum Value = 12
c. Middle value = 15
d. Just below Maximum value = 19
e. Maximum Value = 20

Test Case	Number1	Number2	Expected Output
1	1	15	16
2	2	15	17
3	5	15	20
4	9	15	24
5	10	15	25
6	5	11	16
7	5	12	17
8	5	19	24
9	5	20	25

Output:

TestID	Scenario	Precondition	Input	Success/ Fail	Output	Expected Output	Post Condition	Result	Written By
1	To test the Addition class	Code should be executed successfully	{1,15}, {2,15}, {5,15}, {9,15}, {10,15}, {5,11}, {5,12}, {5,19}, {5,20}	Success	{16}, {17}, {20}, {24}, {25}, {16}, {17}, {24}, {25}	{16}, {17}, {20}, {24}, {25}, {16}, {17}, {24}, {25}	Code should terminated successfully	Success	Rhea Sidana

Test Results | Output - Addition (test)

addition.AdditionParameterizedBVATest

Tests passed: 100.00 %

All 9 tests passed. (0.09 s)

- addition.AdditionParameterizedBVATest passed
  - testAdd[0] passed (0.0 s)
  - testAdd[1] passed (0.0 s)
  - testAdd[2] passed (0.0 s)
  - testAdd[3] passed (0.0 s)
  - testAdd[4] passed (0.0 s)
  - testAdd[5] passed (0.0 s)
  - testAdd[6] passed (0.0 s)
  - testAdd[7] passed (0.0 s)
  - testAdd[8] passed (0.0 s)

```
add passed
add passed
add passed
add passed
add passed
add passed
add passed
add passed
add passed
```

addition.AdditionParameterizedBVATest > additionParams

Test Results | Output - Addition (run)

```
run:
add passed
add passed
add passed
add passed
add passed
add passed
add passed
add passed
add passed
MessageUtility TestRunner class passed
BUILD SUCCESSFUL (total time: 0 seconds)
```

CA1. Demonstrate test driven development through a project QuadraticEquationChecker. The project should contain a class QuadraticEquationCheck with appropriate methods for returning the type of quadratic equation for a given set of inputs .Through JUnit create corresponding Parameterized Test classes and TestRunner classes to execute the above test on a collection of test cases. The test cases should be chosen using boundary value analysis technique for an input range from [1-100].

Solution:

QuadraticEquationChecker.java

```
package quadraticequationchecker;


public class QuadraticEquationChecker {

   private int a,b,c;


   // calculate the determinant

   private int determinant(){

      return ((this.b*this.b)-(4*this.a*this.c));

   }


   //constructor

   public QuadraticEquationChecker(int a,int b,int c){

      this.a=a;

      this.b=b;

      this.c=c;

   }


   //calculate root types

   public String getRoots(){
```

```
        int d = this.determinant();
        if(d<0){
            return "Imaginary Roots";
        }
        else if(d>0){
            return "Real Roots";
        }
        return "Real Root";
    }

    /**
     * @param args the command line arguments
     */
    /*
    public static void main(String[] args) {
        // TODO code application logic here
    }
    */
}
```

QuadraticEquationCheckerTest.java

```
package quadraticequationchecker;

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

```
import java.util.*;
import org.junit.runner.*;
import org.junit.runners.*;

@RunWith(Parameterized.class)
public class QuadraticEquationCheckerTest {
    private int input1,input2,input3;
    private String expected;
    private QuadraticEquationChecker instance;

    public QuadraticEquationCheckerTest(int input1,int input2,int input3,String expected) {
        this.input1 = input1;
        this.input2 = input2;
        this.input3 = input3;
        this.expected = expected;
    }

    @BeforeClass
    public static void setUpClass() {
    }

    @AfterClass
    public static void tearDownClass() {
    }
```

```
@Before
public void setUp() {
    instance = new QuadraticEquationChecker(input1,input2,input3);
}

@After
public void tearDown() {
}

/* for a
@Parameterized.Parameters
public static Collection quadraticParams(){
    return Arrays.asList(new Object[][]{{1,50,50,"Real Roots"},{2,50,50,"Real
Roots"},{50,50,50,"Imaginary Roots"},
            {99,50,50,"Imaginary Roots"},{100,50,50,"Real Root"}});
}*/
/* for b
@Parameterized.Parameters
public static Collection quadraticParams(){
    return Arrays.asList(new Object[][]{{50,1,50,"Imaginary Roots"},{50,2,50,"Imaginary Roots"},
        {50,99,50,"Imaginary Roots"},{50,100,50,"Real Root"}});
}*/
/* for c */
@Parameterized.Parameters
public static Collection quadraticParams(){
```

```
        return Arrays.asList(new Object[][]{{50,50,1,"Real Roots"},{50,50,2,"Real
Roots"},{50,50,99,"Imaginary Roots"},
            {50,50,100,"Imaginary Roots"}});
    }

    /**
     * Test of getRoots method, of class QuadraticEquationChecker.
     */
    @Test
    public void testGetRoots() {
        System.out.println("getRoots");
        String result = instance.getRoots();
        assertEquals(expected, result);
        // TODO review the generated test code and remove the default call to fail.
        //fail("The test case is a prototype.");
    }

}
```

TestRunner.java

```
package quadraticequationchecker;

import org.junit.runner.*;
import org.junit.runner.notification.*;

public class TestRunner {
```

```
    public static void main(String[] args){
        Result res = JUnitCore.runClasses(QuadraticEquationCheckerTest.class);

        boolean hasFailureOccured = false;
        for(Failure fail : res.getFailures()){
            System.out.println("Failure occured: \n\t"+fail);
            hasFailureOccured = true;
        }

        if(hasFailureOccured == false){
            System.out.println("Test passed successfully!");
        }
    }
}
```

BVA:

Inputs: a,b,c

Range: [1,100]

Values:

a. Minimum value: 1
b. Just above minimum value: 2
c. Middle value: 50
d. Just below maximum value: 99
e. Maximum value: 100

Number of Test cases: 4n+1: (4*3)+1: 13

Output values: [d=(b*b)-(4*a*c)]

a. Real roots (d>0)

b. Real root (d=0)
c. Imaginary Roots (d<0)

Test Case	A	B	C	Expected Output
1	1	50	50	Real roots
2	2	50	50	Real roots
3	50	50	50	Imaginary Roots
4	99	50	50	Imaginary Roots
5	100	50	50	Imaginary Roots
6	50	1	50	Imaginary Roots
7	50	2	50	Imaginary Roots
8	50	99	50	Imaginary Roots
9	50	100	50	Real root
10	50	50	1	Real roots
11	50	50	2	Real roots
12	50	50	99	Imaginary Roots
13	50	50	100	Imaginary Roots

Output:

TestID	Scenario	Precondition	Input	Success/ Fail	Output	Expected Output	Post Condition	Result	Written By
1	To test the QuadraticEquationChecker class	Code should be executed successfully	{1,50,50}, {2,50,50}, {50,50,50}, {99,50,50}, {100,50,50}	Fail	{"Real Roots"}, {"Real Roots"}, {"Imaginary Roots"}, {"Imaginary Roots"}, {"Imaginary Roots"}	{"Real Roots"}, {"Real Roots"}, {"Imaginary Roots"}, {"Imaginary Roots"}, {"Real Root"}	Code should terminated successfully	Fail	Rhea Sidana

```
Test Results | Output - QuadraticEquationChecker (run)
run:
getRoots
getRoots
getRoots
getRoots
getRoots
Failure occured:
        testGetRoots[4](quadraticequationchecker.QuadraticEquationCheckerTest): expected:<[Real Root]> but was:<[Imaginary Roots]>
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
39
40      @Parameterized.Parameters
41      public static Collection quadraticParams(){
42          return Arrays.asList(new Object[][]{{1,50,50,"Real Roots"},{2,50,50,"Real Roots"},{50,50,50,"Imaginary Roots"},
43                  {99,50,50,"Imaginary Roots"},{100,50,50,"Real Root"}});
44      }
```

```
quadraticequationchecker.QuadraticEquationCheckerTest > testGetRoots >
Test Results | Output - QuadraticEquationChecker (test)
addition.AdditionParameterizedBVATest | quadraticequationchecker.QuadraticEquationCheckerTest
Tests passed: 80.00 %
4 tests passed, 1 test failed. (0.116 s)
quadraticequationchecker.QuadraticEquationCheckerTest Failed
  testGetRoots[0] passed (0.002 s)
  testGetRoots[1] passed (0.0 s)
  testGetRoots[2] passed (0.001 s)
  testGetRoots[3] passed (0.001 s)
  testGetRoots[4] Failed: expected:<[Real Root]> but was:<[Imaginary Roots]>
    expected:<[Real Root]> but was:<[Imaginary Roots]>
    junit.framework.AssertionFailedError
    at quadraticequationchecker.QuadraticEquationCheckerTest.testGetRoots(QuadraticEquationCheckerTest.java:53)

getRoots
getRoots
getRoots
getRoots
getRoots
```

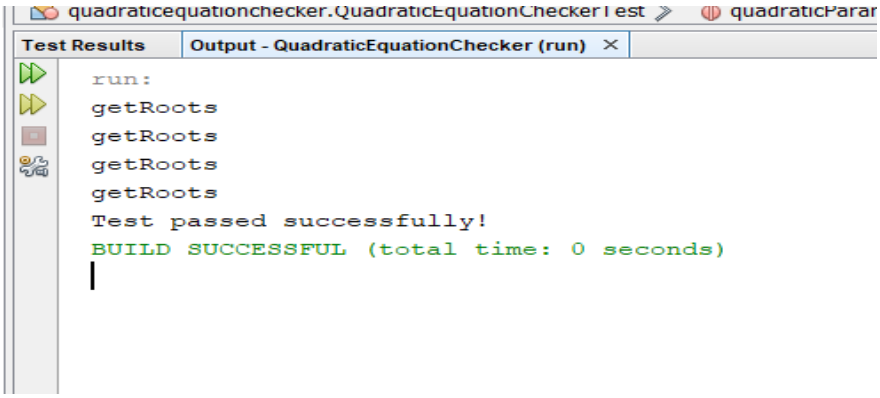
TestID	Scenario	Precondition	Input	Success/ Fail	Output	Expected Output	Post Condition	Result	Written By
2	To test the QuadraticEquationChecker class	Code should be executed successfully	{50,1,50}, {50,2,50}, {50,99,50}, {50,100,50}	Success	{"Imaginary Roots"}, {"Imaginary Roots"}, {"Imaginary Roots"}, {"Imaginary Roots"}, {"Real Root"}	{"Imaginary Roots"}, {"Imaginary Roots"}, {"Imaginary Roots"}, {"Imaginary Roots"}, {"Real Root"}	Code should terminated successfully	Success	Rhea Sidana

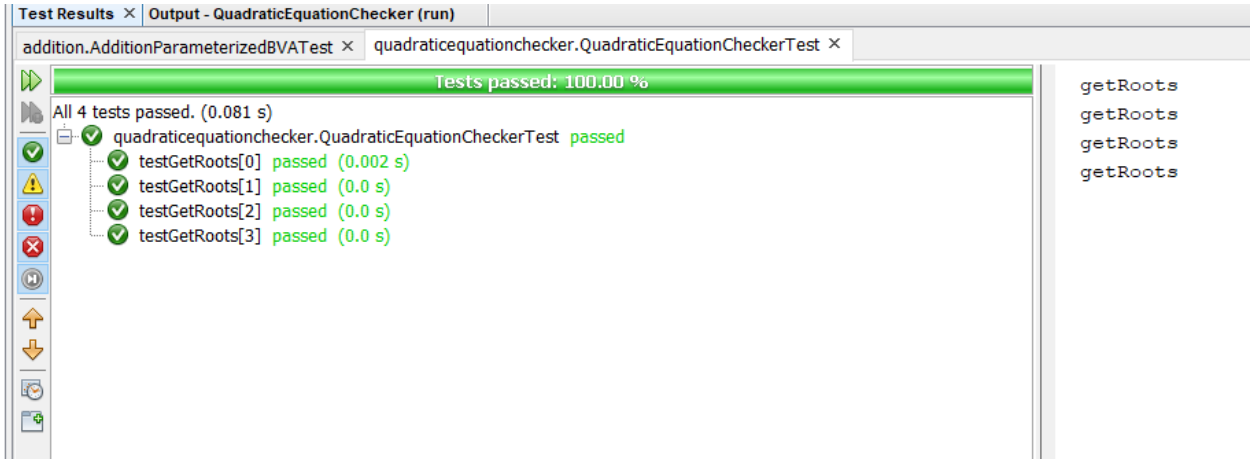
```
Test Results | Output - QuadraticEquationChecker (run)
run:
getRoots
getRoots
getRoots
getRoots
Test passed successfully!
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
46      /* for b*/
47      @Parameterized.Parameters
48      public static Collection quadraticParams(){
49          return Arrays.asList(new Object[][]{{50,1,50,"Imaginary Roots"},{50,2,50,"Imaginary Roots"},
50              {50,99,50,"Imaginary Roots"},{50,100,50,"Real Root"}});
51      }
52
53      /**
```

```
quadraticequationchecker.QuadraticEquationCheckerTest > quadraticParams
Test Results | Output - QuadraticEquationChecker (test)
addition.AdditionParameterizedBVATest | quadraticequationchecker.QuadraticEquationCheckerTest
Tests passed: 100.00 %
All 4 tests passed. (0.087 s)
quadraticequationchecker.QuadraticEquationCheckerTest passed
  testGetRoots[0] passed (0.002 s)
  testGetRoots[1] passed (0.0 s)
  testGetRoots[2] passed (0.0 s)
  testGetRoots[3] passed (0.003 s)

getRoots
getRoots
getRoots
getRoots
```

TestID	Scenario	Precondition	Input	Success/ Fail	Output	Expected Output	Post Condition	Result	Written By
3	To test the QuadraticEquationChecker class	Code should be executed successfully	{50,50,1}, {50,50,2}, {50,50,99}, {50,50,100}	Success	{"Real Roots"}, {"Real Roots"}, {"Imaginary Roots"}, {"Imaginary Roots"}	{"Real Roots"}, {"Real Roots"}, {"Imaginary Roots"}, {"Imaginary Roots"}	Code should terminated success fully	Success	Rhea Sidana


quadraticequationchecker.QuadraticEquationCheckerTest
quadraticParar
Test Results
Output - QuadraticEquationChecker (run)
run:
getRoots
getRoots
getRoots
getRoots
Test passed successfully!
BUILD SUCCESSFUL (total time: 0 seconds)


Test Results
Output - QuadraticEquationChecker (run)
addition.AdditionParameterizedBVATest
quadraticequationchecker.QuadraticEquationCheckerTest
Tests passed: 100.00 %
All 4 tests passed. (0.081 s)
quadraticequationchecker.QuadraticEquationCheckerTest passed
testGetRoots[0] passed (0.002 s)
testGetRoots[1] passed (0.0 s)
testGetRoots[2] passed (0.0 s)
testGetRoots[3] passed (0.0 s)
getRoots
getRoots
getRoots
getRoots

DP1. Create a new project called AssertClass. Refactor the test class and TestRunner class of question CP1 to demonstrate the assertion methods of the assert class to write useful tests.

Solution:

MessageUtility.java

```
package assertclass;


public class MessageUtility {

   private String message;


   //Constructor

   public MessageUtility(String message){

      this.message=message;

   }


   //return the message

   public String getMessage(){

      System.out.println("Printing the message : " + this.message);

      return this.message;

   }

}
```

MessageUtilityTest.java

```
package assertclass;


import org.junit.*;
```

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

public class MessageUtilityTest {

    public MessageUtilityTest() {
    }

    @BeforeClass
    public static void setUpClass() {
    }

    @AfterClass
    public static void tearDownClass() {
    }

    @Before
    public void setUp() {
    }

    @After
    public void tearDown() {
    }

    /**
     * Test of getMessage method, of class MessageUtility.
```

```
 */
@Test
public void testGetMessage() {
    System.out.println("getMessage");
    MessageUtility instance = new MessageUtility("Hello! I am Rhea Sidana.");
    String expResult = "Hello! I am Rhea Sidana.";
    String result = instance.getMessage();
    System.out.println("Checking AssertEquals.");
    assertEquals(expResult, result);

    MessageUtility instance2 = null;
    System.out.println("Checking AssertNull.");
    assertNull(instance2);

    System.out.println("Checking AssertNotNull.");
    assertNotNull(instance);

    System.out.println("Checking AssertTrue.");
    assertTrue(expResult.equals(instance.getMessage()));

    System.out.println("Checking AssertFalse.");
    assertFalse(!(expResult.equals(instance.getMessage())));

    // TODO review the generated test code and remove the default call to fail.
    //fail("Checking Failure !");
```

```
    }

}
```

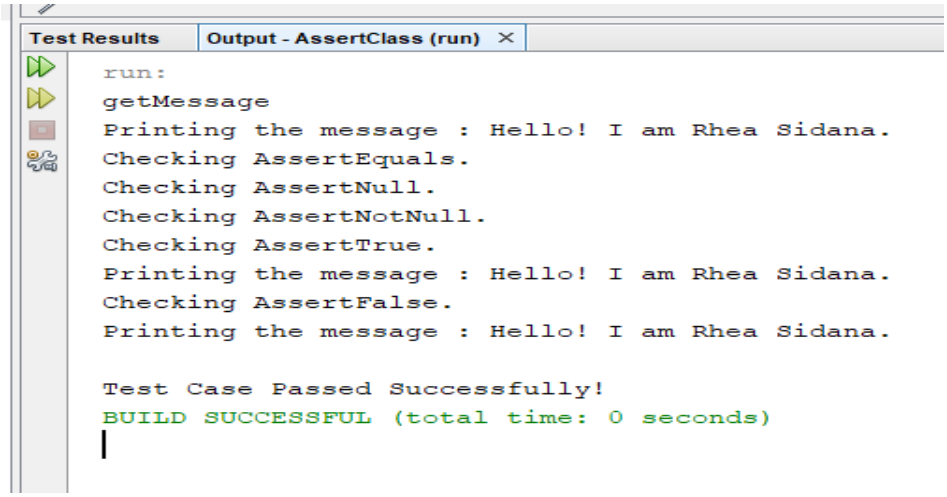
TestRunner.java

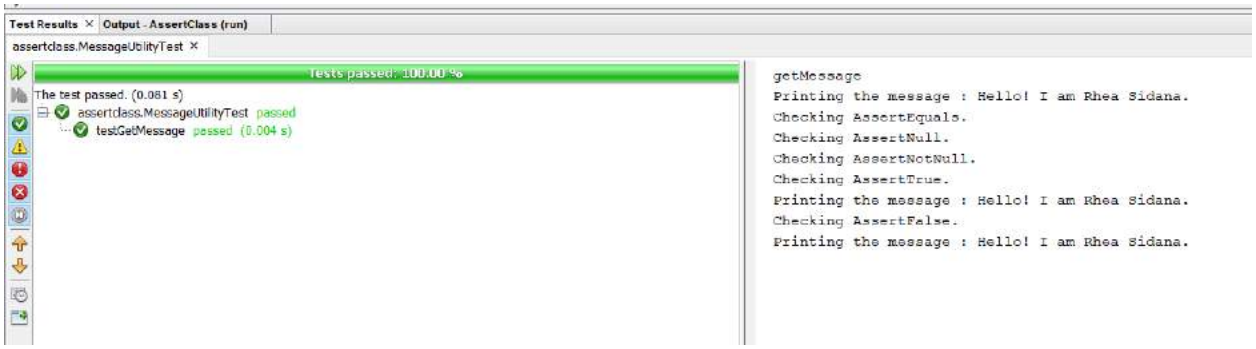
```
package assertclass;

import org.junit.runner.*;
import org.junit.runner.notification.*;

public class TestRunner {
    public static void main(String[] args){
        Result result = JUnitCore.runClasses(MessageUtilityTest.class);

        boolean noFailure = true;
        for(Failure fail : result.getFailures()){
            System.out.println("Failure occured: "+fail);
            noFailure=false;
        }

        if(noFailure){
            System.out.println("\nTest Case Passed Successfully!");
        }
    }
```

}

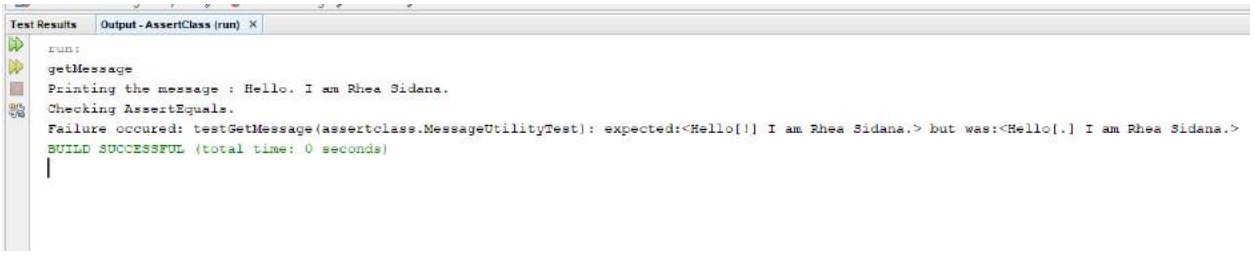
Output:

TestID	Scenario	Precondition	Input	Success/ Fail	Output	Expected Output	Post Condition	Result	Written By
1	To test the MessageUtility class	Code should be executed successfully	"Hello! I am Rhea Sidana."	Success	"Hello! I am Rhea Sidana."	"Hello! I am Rhea Sidana."	Code should terminated success fully	Success	Rhea Sidana





TestID	Scenario	Precondition	Input	Success/ Fail	Output	Expected Output	Post Condition	Result	Written By
2	To test the MessageUtility class	Code should be executed successfully	"Hello. I am Rhea Sidana."	Success	"Hello. I am Rhea Sidana."	"Hello! I am Rhea Sidana."	Code should terminated success fully	Success	Rhea Sidana



DP2. In the MessageUtil problem of CP1 add an infinite while loop inside the printMessage() method. In the corresponding test class add timeout of 1000 to the testprintMessage() test case. Also use the TestRunner.java class to execute the test case.

Solution:

MessageUtil.java

```
package messageutil;


public class MessageUtil {

   private String message;


   //Constructor

   public MessageUtil(String message){

      this.message=message;

   }


   //return the message

   public String getMessage(){

      //infinite loop

      int i=0;

      while(i==0){}


      System.out.println("Printing the message : " + this.message);

      return this.message;

   }

}
```

MessageUtilTest.java

```
package messageutil;


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



public class MessageUtilTest {


    public MessageUtilTest() {
    }


    @BeforeClass
    public static void setUpClass() {
    }


    @AfterClass
    public static void tearDownClass() {
    }


    @Before
    public void setUp() {
    }


    @After
```

```
    public void tearDown() {
    }

    /**
     * Test of getMessage method, of class MessageUtil.
     */
    @Test(timeout= 1000)
    public void testGetMessage() {
        System.out.println("getMessage");
        MessageUtil instance = new MessageUtil("Hello. I am Rhea Sidana.");
        String expResult = "Hello! I am Rhea Sidana.";
        String result = instance.getMessage();
        assertEquals(expResult, result);
        // TODO review the generated test code and remove the default call to fail.
        fail("The test case is a prototype.");
    }

}
```

MessageUtilTestRunner.java

```
package messageutil;

import org.junit.runner.*;
import org.junit.runner.notification.*;
```

```
public class MessageUtilTestRunner {
    public static void main(String[] args){
        Result result = JUnitCore.runClasses(MessageUtilTest.class);


        boolean noFailure = true;
        for(Failure fail : result.getFailures()){
            System.out.println("Failure occured: "+fail);
            noFailure=false;
        }


        if(noFailure){
            System.out.println("\nTest Case Passed Successfully!");
        }
    }
}
```

Output:

TestID	Scenario	Precondition	Input	Success/ Fail	Output	Expected Output	Post Condition	Result	Written By
1	To test the MessageUtil class	Code should be executed successfully	"Hello. I am Rhea Sidana."	Fail	"Hello! I am Rhea Sidana."	"Hello! I am Rhea Sidana."	Code should terminated success fully	Fail	Rhea Sidana

Test Results | Output - MessageUtil (run)

```
run:
getMessage
Failure occured: testGetMessage(messageutil.MessageUtilTest): test timed out after 1000 milliseconds
BUILD SUCCESSFUL (total time: 1 second)
```

Test Results | Output - MessageUtil (run)

assertclass.MessageUtilityTest | assertclass.MessageUtilTest | messageutil.MessageUtilTest

Tests passed: 0.00 %

No test passed, 1 test caused an error. (1.116 s)

- messageutil.MessageUtilTest Failed
  - testGetMessage caused an ERROR: test timed out after 1000 milliseconds
    - test timed out after 1000 milliseconds
    - org.junit.runners.model.TestTimedOutException
    - at messageutil.MessageUtil.getMessage(MessageUtil.java:16)
    - at messageutil.MessageUtilTest.testGetMessage(MessageUtilTest.java:37)
    - at java.util.concurrent.FutureTask.run(FutureTask.java:266)
    - at java.lang.Thread.run(Thread.java:748)

getMessage

DP3. In the printMessage() method of CP1 add an error condition inside the printMessage() method. Test this exception occurrence through appropriate changes in the test class.

Solution:

Message.java

```
package message;


public class Message {

   private String message;


   //Constructor

   public Message(String message){

      this.message=message;

   }


   //return the message

   public String getMessage(){

      //System.out.println("Printing the message : " + this.message);

      this.message = null;

      return this.message;

   }

}
```

MessageTest.java

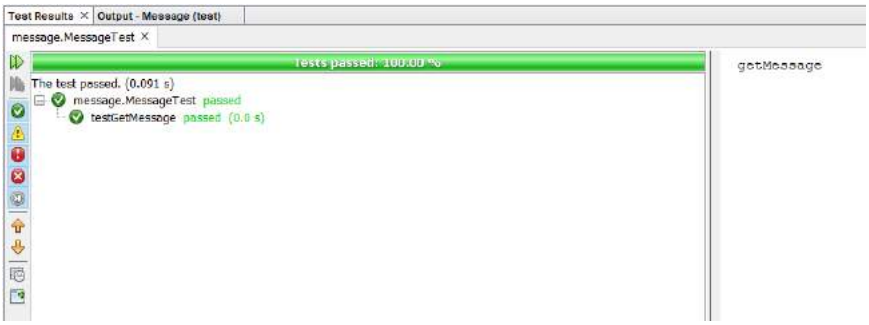
```
package message;


import org.junit.*;
```

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

public class MessageTest {

    public MessageTest() {
    }

    @BeforeClass
    public static void setUpClass() {
    }

    @AfterClass
    public static void tearDownClass() {
    }

    @Before
    public void setUp() {
    }

    @After
    public void tearDown() {
    }

    /**
```

```
     * Test of getMessage method, of class Message.
     */
    @Test(expected = NullPointerException.class)
    public void testGetMessage() {
        System.out.println("getMessage");
        Message instance = new Message("Hello. I am Rhea Sidana.");
        //String expResult = "Hello! I am Rhea Sidana.";
        String result = instance.getMessage();
        if(result.equals(null)){
            throw new NullPointerException();
        }
        //assertEquals(expResult, result);
        // TODO review the generated test code and remove the default call to fail.
        //fail("The test case is a prototype.");
    }

}
```

Output:

TestID	Scenario	Precondition	Input	Success/ Fail	Output	Expected Output	Post Condition	Result	Written By
1	To test the Message class	Code should be executed successfully	"Hello! I am Rhea Sidana."	Success	NullPointerException	NullPointerException	Code should terminated success fully	Success	Rhea Sidana


Test Results
Output - Message (test)
message.MessageTest
Tests passed: 100.00 %
The test passed. (0.091 s)
message.MessageTest passed
testGetMessage passed (0.0 s)
getMessage

DP4. In Question CP1 above demonstrate the usage of @Ignore annotation with MessageUtilityTest method.

Solution:

MessageUtils.java

```
package messageutils;


public class MessageUtils {

   private String message;


   //Constructor

   public MessageUtils(String message){

      this.message=message;

   }


   public String getMessage(){

      System.out.println("Printing the message : " + this.message);

      return this.message;

   }

}
```

MessageUtilsTest.java

```
package messageutils;


import org.junit.*;

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

```
public class MessageUtilsTest {

    public MessageUtilsTest() {
    }

    @BeforeClass
    public static void setUpClass() {
    }

    @AfterClass
    public static void tearDownClass() {
    }

    @Before
    public void setUp() {
    }

    @After
    public void tearDown() {
    }

    /**
     * Test of getMessage method, of class MessageUtils.
     */
    @Test
```

```
    public void testGetMessage() {
        System.out.println("getMessage");
        MessageUtils instance = new MessageUtils("Hello! I am Rhea Sidana.");
        String expResult = "Hello! I am Rhea Sidana.";
        String result = instance.getMessage();
        assertEquals(expResult, result);
        // TODO review the generated test code and remove the default call to fail.
        //fail("The test case is a prototype.");
    }

    @Ignore
    @Test(expected = NullPointerException.class)
    public void testGetMessageScond() {
        System.out.println("getMessage");
        MessageUtils instance = new MessageUtils("Hello. I am Rhea Sidana.");
        //String expResult = "Hello! I am Rhea Sidana.";
        String result = instance.getMessage();
        if(result.equals(null)){
            throw new NullPointerException();
        }
        //assertEquals(expResult, result);
        // TODO review the generated test code and remove the default call to fail.
        //fail("The test case is a prototype.");
    }
}
```

Output:

TestID	Scenario	Precondition	Input	Success/ Fail	Output	Expected Output	Post Condition	Result	Written By
1	To test the Message class	Code should be executed successfully	"Hello! I am Rhea Sidana."	Success	"Hello! I am Rhea Sidana."	"Hello! I am Rhea Sidana."	Code should terminated success fully	Success	Rhea Sidana

DP5. In questionCP2 above copy and re-factor the class PrimeNumberChecker.java to create other classes PrimeNumberCheckerTest1.java and PrimeNumberChecker11.java. In one assign the object to null and then test the NullPointer Exception and in second convert one test case to false and also test for the timeout parameter. Now create a test suite to execute all these three test cases. Check for the output to check if the mutants were detected or not. Also ignore the modified test cases and re-run the test suite to confirm the outcome.

Solution:

PrimeNumberCheck.java

```
package primenumbercheck;


public class PrimeNumberCheck {

   private int number;


   //constructor

   public PrimeNumberCheck(int number){

      this.number=number;

   }


   //check if prime

   public boolean isPrime(){

      for(int i=2;i<=number/2;i++){

         if(number%i==0)

            return false;

      }

      return true;

   }
```

```
}
```

PrimeNumberCheckTest.java

```
package primenumbercheck;

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.*;
import org.junit.runner.*;
import org.junit.runners.*;
import java.util.*;

@RunWith(Parameterized.class)
public class PrimeNumberCheckTest {
    private int input;
    private boolean expected;
    private PrimeNumberCheck instance;

    public PrimeNumberCheckTest(int input,boolean expected) {
        this.input=input;
        this.expected=expected;
```

```
}

@BeforeClass
public static void setUpClass() {
}

@AfterClass
public static void tearDownClass() {
}

@Before
public void setUp() {
    this.instance = new PrimeNumberCheck(input);
}

@After
public void tearDown() {
}

//defining parameters
@Parameterized.Parameters
public static Collection primeNumberCollections(){
    return Arrays.asList(new Object[][]{{2,true},{8,false},{3,true},{7,true},{12,false}});
}
```

```
    /**
     * Test of isPrime method, of class PrimeNumberChecker.
     */
    @Test
    public void testIsPrime() {
        System.out.println("isPrime");
        //boolean expResult = false;
        boolean result = this.instance.isPrime();
        assertEquals(expected, result);
        // TODO review the generated test code and remove the default call to fail.
        //fail("The test case is a prototype.");
    }

}
```

PrimeNumberCheckTest1.java

```
package primenumbercheck;

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

public class PrimeNumberCheckTest1 {

    public PrimeNumberCheckTest1() {
```

```
}

@BeforeClass
public static void setUpClass() {
}

@AfterClass
public static void tearDownClass() {
}

@Before
public void setUp() {
}

@After
public void tearDown() {
}

// TODO add test methods here.
// The methods must be annotated with annotation @Test. For example:
//
// @Test
// public void hello() {}
// false test case
@Test
```

```
    public void testIsPrime() {
        System.out.println("isPrime");
        PrimeNumberCheck instance = new PrimeNumberCheck(17);
        boolean expected = false;
        boolean result = instance.isPrime();
        assertEquals(expected, result);
        // TODO review the generated test code and remove the default call to fail.
        //fail("The test case is a prototype.");
    }

    @Test(timeout = 50)
    public void testIsPrime1() {
        System.out.println("isPrime");
        PrimeNumberCheck instance = new PrimeNumberCheck(17);
        boolean expected = true;
        while(expected){}
        boolean result = instance.isPrime();
        assertEquals(expected, result);
        // TODO review the generated test code and remove the default call to fail.
        //fail("The test case is a prototype.");
    }
}
```

PrimeNumberCheckTest2.java

```
package primenumbercheck;


import org.junit.*;

import static org.junit.Assert.*;

import java.lang.*;


public class PrimeNumberCheckTest2 {


    public PrimeNumberCheckTest2() {

    }


    @BeforeClass

    public static void setUpClass() {

    }


    @AfterClass

    public static void tearDownClass() {

    }


    @Before

    public void setUp() {

    }


    @After
```

```
    public void tearDown() {
    }

    // TODO add test methods here.
    // The methods must be annotated with annotation @Test. For example:
    //
    // @Test
    // public void hello() {}
    // null object
    @Test(expected = NullPointerException.class)
    public void testIsPrime() {
        System.out.println("isPrime");
        PrimeNumberCheck instance = null;
        boolean expected = false;
        boolean result = instance.isPrime();
        assertEquals(expected, result);
        // TODO review the generated test code and remove the default call to fail.
        //fail("The test case is a prototype.");
    }
}
```

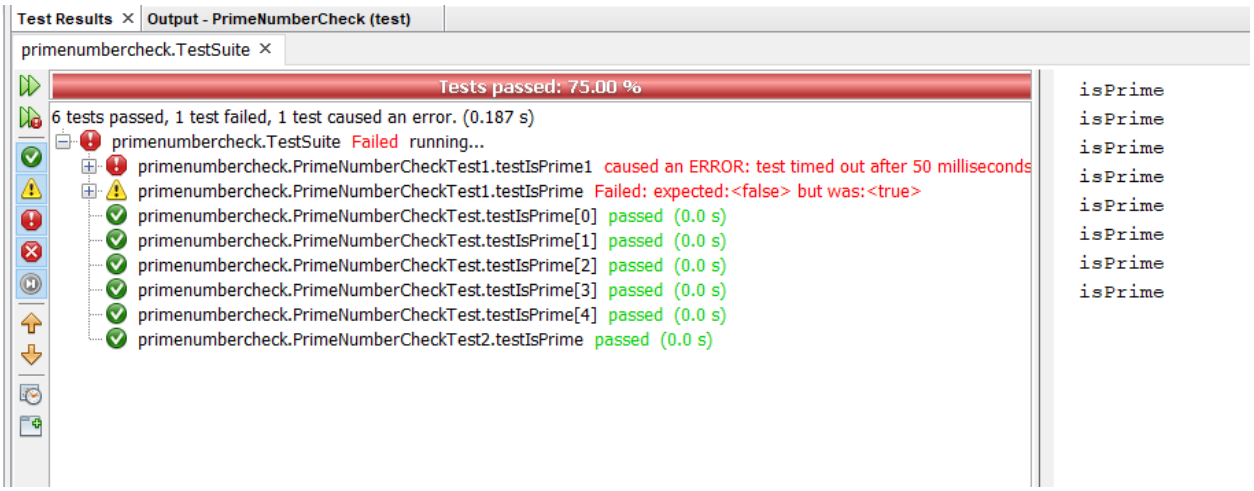
TestSuite.java

```
package primenumbercheck;


import org.junit.After;

import org.junit.AfterClass;

import org.junit.Before;

import org.junit.BeforeClass;

import org.junit.runner.RunWith;

import org.junit.runners.Suite;


@RunWith(Suite.class)

@Suite.SuiteClasses({primenumbercheck.PrimeNumberCheckTest1.class,
primenumbercheck.PrimeNumberCheckTest.class, primenumbercheck.PrimeNumberCheckTest2.class})

public class TestSuite {


    @BeforeClass

    public static void setUpClass() throws Exception {

    }


    @AfterClass

    public static void tearDownClass() throws Exception {

    }


    @Before

    public void setUp() throws Exception {

    }
```

```
    @After
    public void tearDown() throws Exception {
    }

}
```

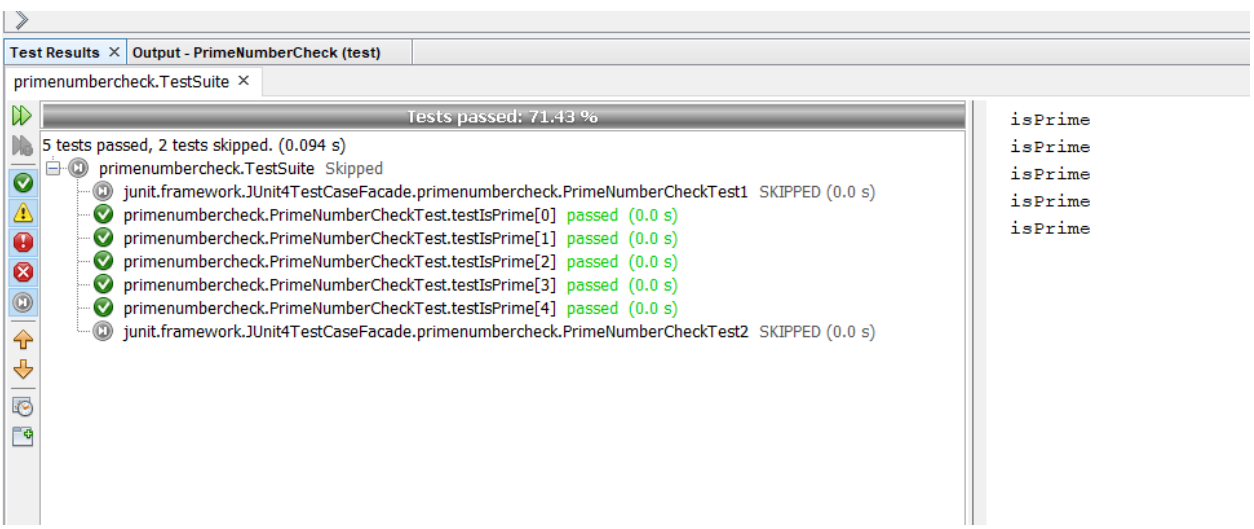
Output:

TestID	Scenario	Precondition	Input	Success/ Fail	Output	Expected Output	Post Condition	Result	Written By
1	To test the PrimeNumberChecker class	Code should be executed successfully	2,8,3,7,12	Success	true,false,true,true,false	true,false,true,true,false	Code should terminated successfully	Success	Rhea Sidana
2	To test the PrimeNumberChecker class	Code should be executed successfully	17	Fail	True	false	Code should terminated successfully	Fail	Rhea Sidana
3	To test the PrimeNumberChecker class	Code should be executed successfully	17	Success	NullPointerException	NullPointerException	Code should terminated successfully	Success	Rhea Sidana
3	To test the PrimeNumberChecker class	Code should be executed successfully	17	Fail	Timeout	true	Code should terminated successfully	Fail	Rhea Sidana



AFTER @Ignore

TestID	Scenario	Precondition	Input	Success/ Fail	Output	Expected Output	Post Condition	Result	Written By
4	To test the PrimeNumberChecker class	Code should be executed successfully	2,8,3,7,12	Success	true,false,true,true, false	true,false,true,true,false	Code should terminated successfully	Success	Rhea Sidana

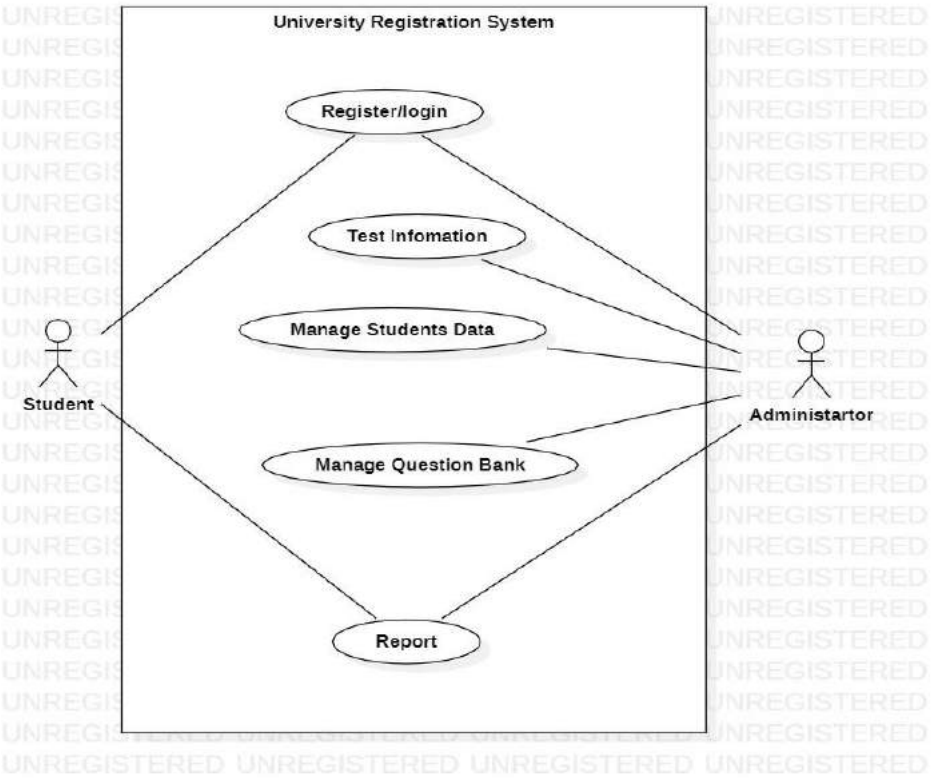


DP6. Consider University Registration System for Use case testing:

a) **Identify actors & users. Draw usecase diagram.**
b) **Write short description of each usecase.**
c) **Elaborate any two usecases and write test cases for these two usecases.**

Solution:

a. UseCase Diagram



b. UseCase Description

Use Case	1
Use Case Name	Register/Login
Objective	Student and administrator registers into the college
Actors	Student, Administration
Pre-Conditions	Student and administrator should have a unique id to register.
Post-Conditions	Register/login either successful or failed
Flow of Events	1. Student Enters his/her username, password with course details. Admin can also register/login with username and password 2. And click on the register button 3. Information goes to the administrator to verify

Use Case	2
Use Case Name	Test Information
Objective	University/College manages all the exams.
Actors	Administration
Pre-Conditions	College faculty/administrator must have logged in.

Post-Conditions	Marks has been uploaded
Flow of Events	1. Faculty/administrator enters details about exam. 2. They enter marks of their subject for each student.

Use Case	3
Use Case Name	Manage Students Information
Objective	Students personal and academic details should be managed by the administrator
Actors	Administration
Pre-Conditions	College faculty/administrator must have login
Post-Conditions	Details are managed
Flow of Events	1. Faculty verifies the details submitted by the student 2. And save it into the database. 3. Update any changes if required

Use Case	4
Use Case Name	Manage Question Bank
Objective	Important and previous year question of each subject is maintained
Actors	Administration
Pre-Conditions	College faculty/administrator must have login
Post-Conditions	Question has been uploaded
Flow of Events	1. Faculty goes to their subject section. 2. Upload pdf of questions with solutions

Use Case	5
Use Case Name	Report
Objective	Student's report is generated
Actors	Student, Administration
Pre-Conditions	College administrator/student must be logged in
Post-Conditions	Report is generated.
Flow of Events	1. Administrator must generate the report on semester wise, year wise and entire course wise. 2. Student can view the report to track the progress.

Output:

TestID	Scenario	Precondition	Input	Success/ Fail	Output	Expected Output	Post Condition	Result	Written By
1	To test the Login	Code should be executed successfully	"abc", "123a"	Success	Logged in	Logged in	Code should terminated success fully	Success	Rhea Sidana

2	To test the Login	Code should be executed successfully	“xyz”, “9ad”	Fail	Logged in	Logged in	Code should terminated successfully	Fail	Rhea Sidana
3	To test the Report	Code should be executed successfully	Abc,12	Success	Report Generation	Report Generation	Code should terminated successfully	Success	Rhea Sidana
4	To test the Login	Code should be executed successfully	Xyz,15	Fail	Wrong report generated	Correct Report Generated	Code should terminated successfully	Fail	Rhea Sidana