Faculty of Engineering, Architecture and Science
Department of Electrical and Computer Engineering
Laboratory Report Cover Page

Course Number	COE891
Course Title	Software Testing & QA
Semester/Year	Winter 2023
Instructor	Reza Semavi
TA Name	Hamed Karimi

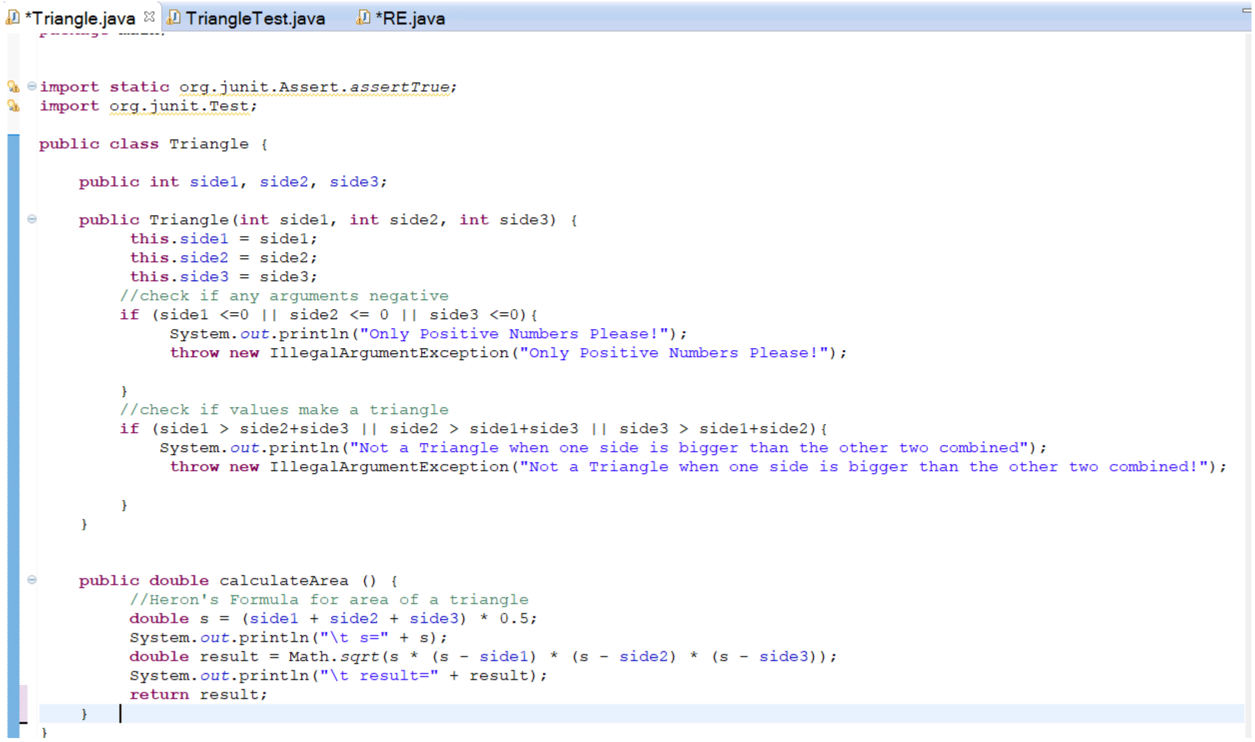
Lab/Tutorial Report No.	2

Section No.	012
Group No.	N/A
Submission Date	Feb 5th, 2023
Due Date	Feb 5th, 2023

Student Name	Student ID	Signature
Hamza Iqbal	500973673	H.I

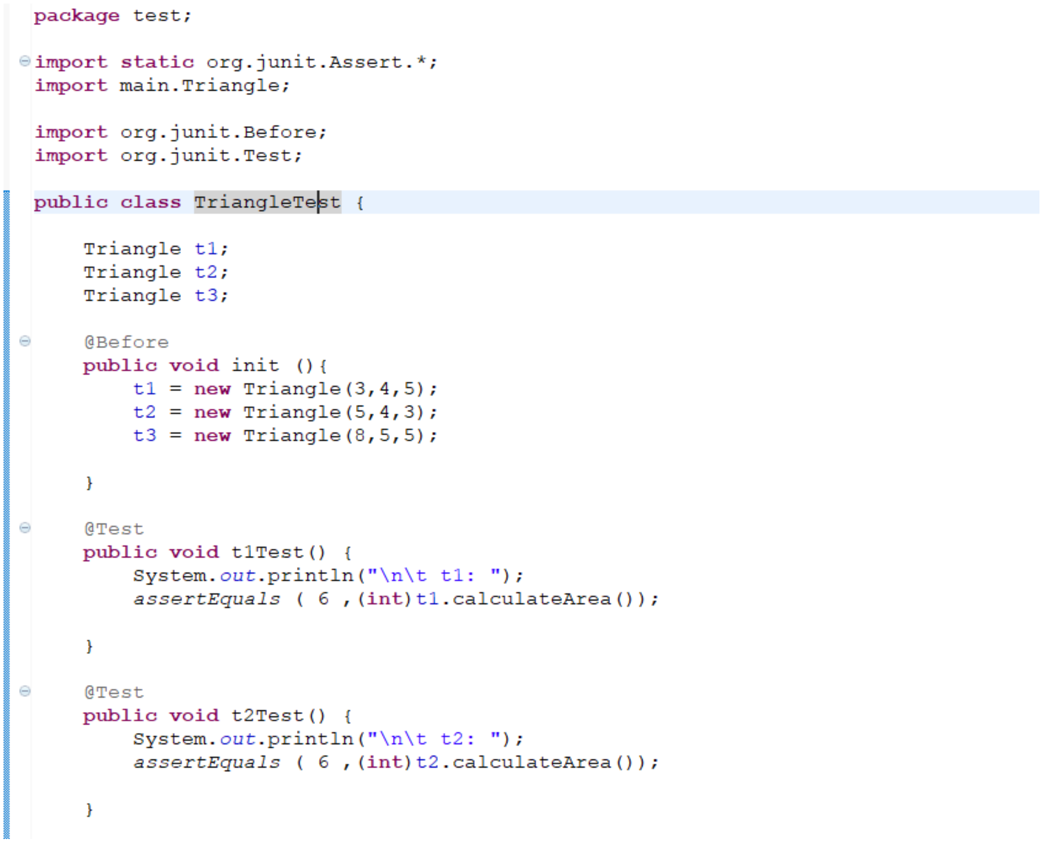
Q1.

Triangle.java Class:

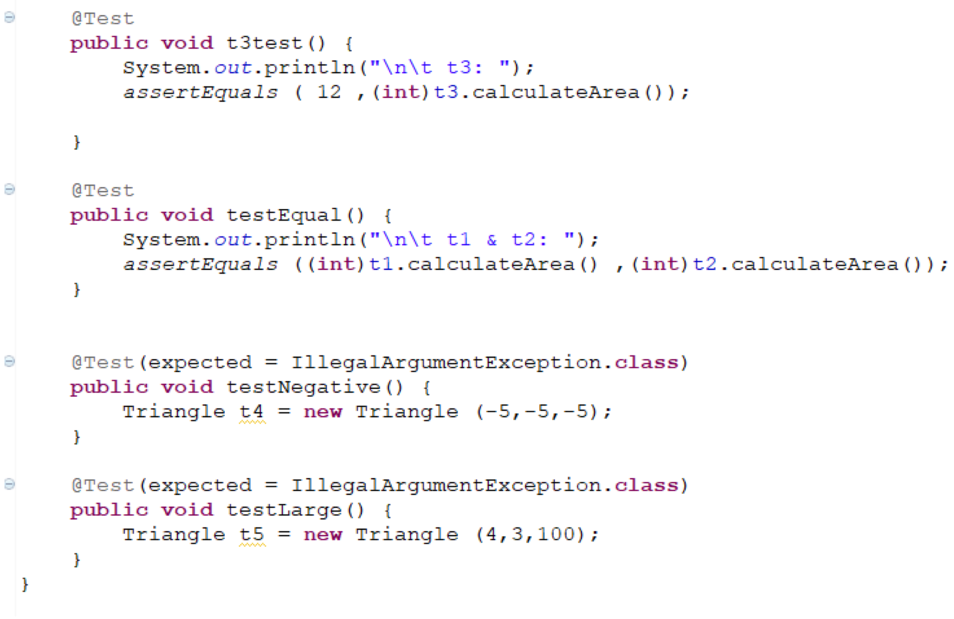


```java
package main;


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

public class Triangle {

    public int side1, side2, side3;

    public Triangle(int side1, int side2, int side3) {
         this.side1 = side1;
         this.side2 = side2;
         this.side3 = side3;
        //check if any arguments negative
        if (side1 <=0 || side2 <= 0 || side3 <=0){
             System.out.println("Only Positive Numbers Please!");
             throw new IllegalArgumentException("Only Positive Numbers Please!");

        }
        //check if values make a triangle
        if (side1 > side2+side3 || side2 > side1+side3 || side3 > side1+side2){
            System.out.println("Not a Triangle when one side is bigger than the other two combined");
             throw new IllegalArgumentException("Not a Triangle when one side is bigger than the other two combined!");

        }
    }


    public double calculateArea () {
         //Heron's Formula for area of a triangle
         double s = (side1 + side2 + side3) * 0.5;
         System.out.println("\t s=" + s);
         double result = Math.sqrt(s * (s - side1) * (s - side2) * (s - side3));
         System.out.println("\t result=" + result);
         return result;
    }
}
```

TriangleTest class:



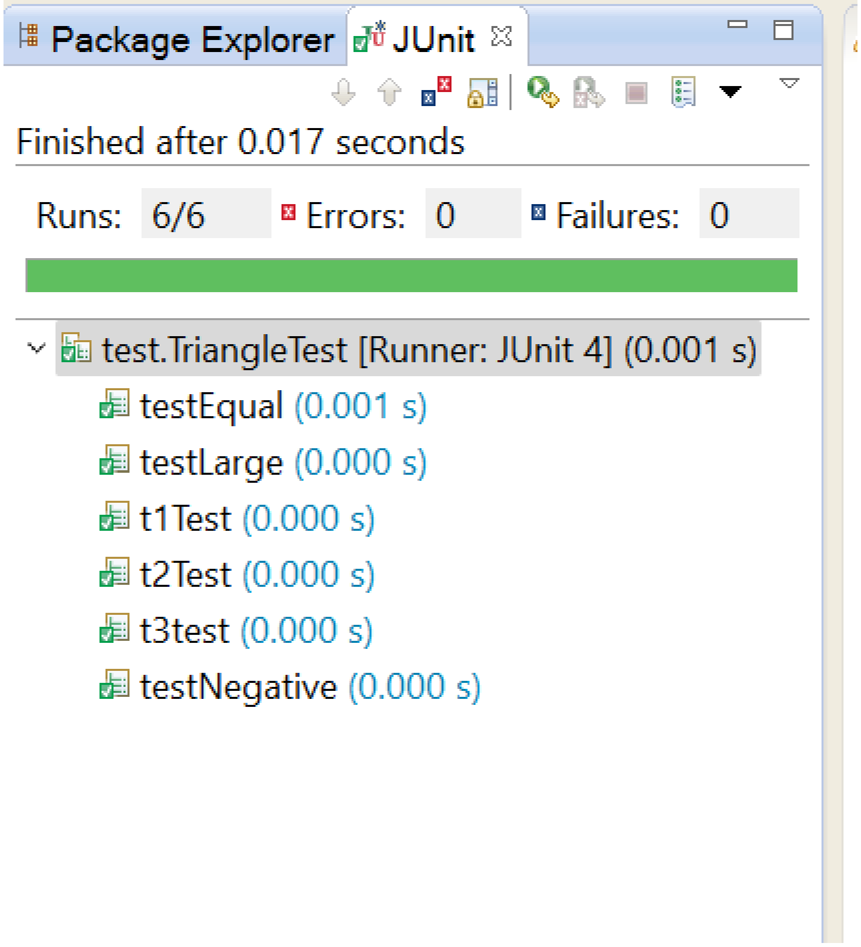
```
package test;

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

import org.junit.Before;
import org.junit.Test;

public class TriangleTest {

    Triangle t1;
    Triangle t2;
    Triangle t3;

    @Before
    public void init (){
        t1 = new Triangle(3,4,5);
        t2 = new Triangle(5,4,3);
        t3 = new Triangle(8,5,5);

    }

    @Test
    public void t1Test() {
        System.out.println("\n\t t1: ");
        assertEquals ( 6 ,(int)t1.calculateArea());

    }

    @Test
    public void t2Test() {
        System.out.println("\n\t t2: ");
        assertEquals ( 6 ,(int)t2.calculateArea());

    }
```



```
    @Test
    public void t3test() {
        System.out.println("\n\t t3: ");
        assertEquals ( 12 ,(int)t3.calculateArea());

    }

    @Test
    public void testEqual() {
        System.out.println("\n\t t1 & t2: ");
        assertEquals ((int)t1.calculateArea() ,(int)t2.calculateArea());
    }


    @Test(expected = IllegalArgumentException.class)
    public void testNegative() {
        Triangle t4 = new Triangle (-5,-5,-5);
    }

    @Test(expected = IllegalArgumentException.class)
    public void testLarge() {
        Triangle t5 = new Triangle (4,3,100);
    }
}
```

If a user tries to create a triangle with the value of the sides being 3,4, & 100, this would not be a triangle in the first place since one side cannot be bigger than the other two combined. Hence, the code should account for input values of such a case, as is seen and modified within the triangle.java's constructor initialization.

JUnit test results of Q1:



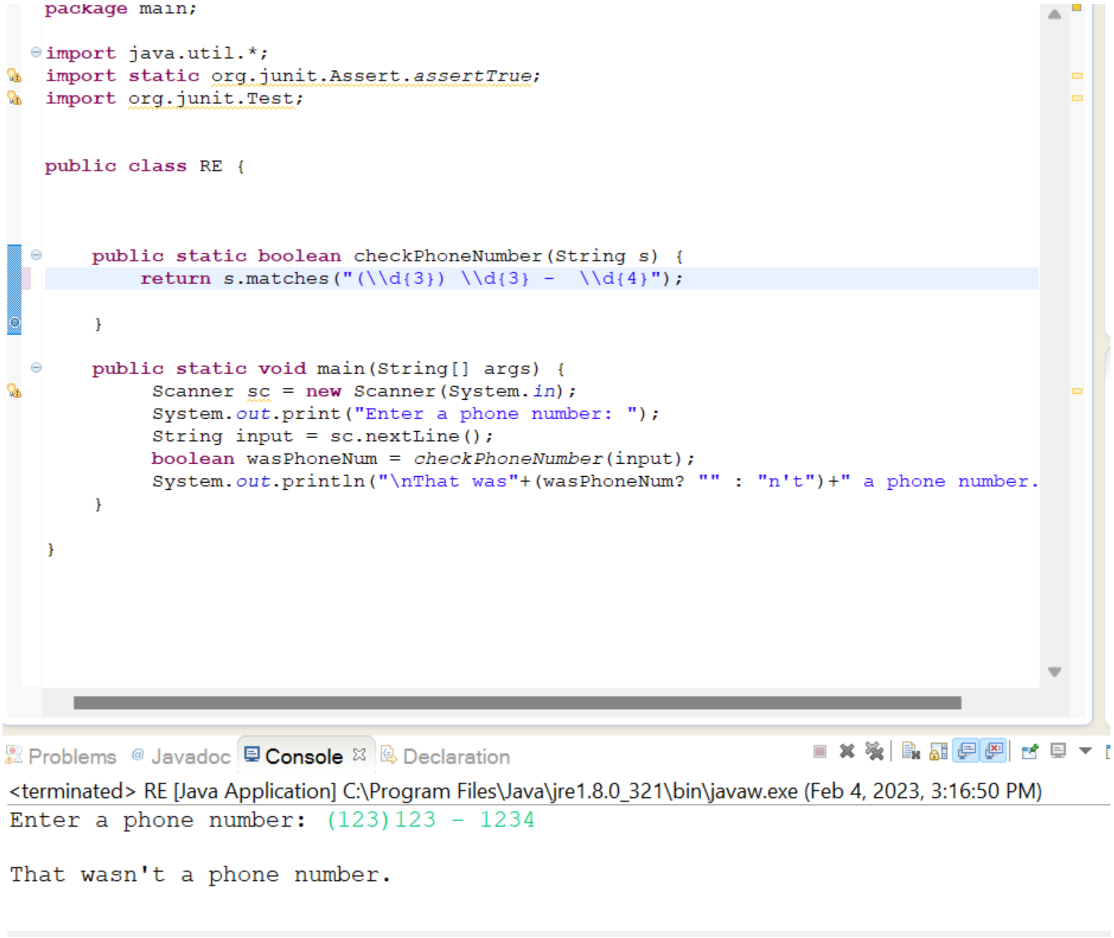
```
Finished after 0.017 seconds

Runs: 6/6    Errors: 0    Failures: 0

test.TriangleTest [Runner: JUnit 4] (0.001 s)
    testEqual (0.001 s)
    testLarge (0.000 s)
    t1Test (0.000 s)
    t2Test (0.000 s)
    t3test (0.000 s)
    testNegative (0.000 s)
```

Q2.

2.
\d is a form of a regular expression, known as 'regex', which is a pattern of characters that describes a set of strings. Regular expressions are used for matching purposes allowing you to test whether a string fits into a specific syntactic form, such as in this case, a phone number. The "\d" was used to match integer digits (0-9) to ensure that the inputs were of the correct form.

3.

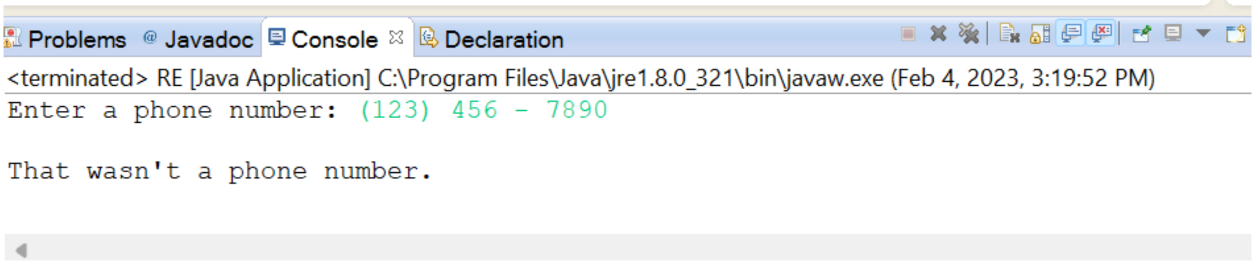
RE.java class + input1 test:



```
package main;

import java.util.*;
import static org.junit.Assert.assertTrue;
import org.junit.Test;


public class RE {



    public static boolean checkPhoneNumber(String s) {
        return s.matches("(\\d{3}) \\d{3} -  \\d{4}");

    }

    public static void main(String[] args) {
         Scanner sc = new Scanner(System.in);
         System.out.print("Enter a phone number: ");
         String input = sc.nextLine();
         boolean wasPhoneNum = checkPhoneNumber(input);
         System.out.println("\nThat was"+(wasPhoneNum? "" : "n't")+" a phone number.
    }

}
```

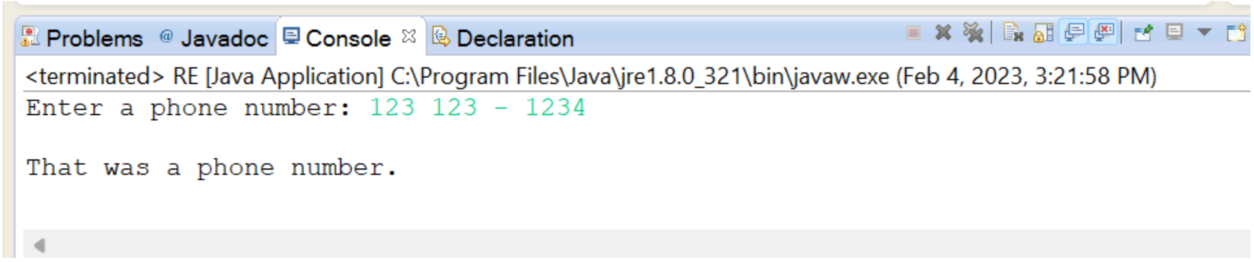
```
<terminated> RE [Java Application] C:\Program Files\Java\jre1.8.0_321\bin\javaw.exe (Feb 4, 2023, 3:16:50 PM)
Enter a phone number: (123)123 - 1234

That wasn't a phone number.
```

input2 test:



```
<terminated> RE [Java Application] C:\Program Files\Java\jre1.8.0_321\bin\javaw.exe (Feb 4, 2023, 3:19:52 PM)
Enter a phone number: (123) 456 - 7890

That wasn't a phone number.
```

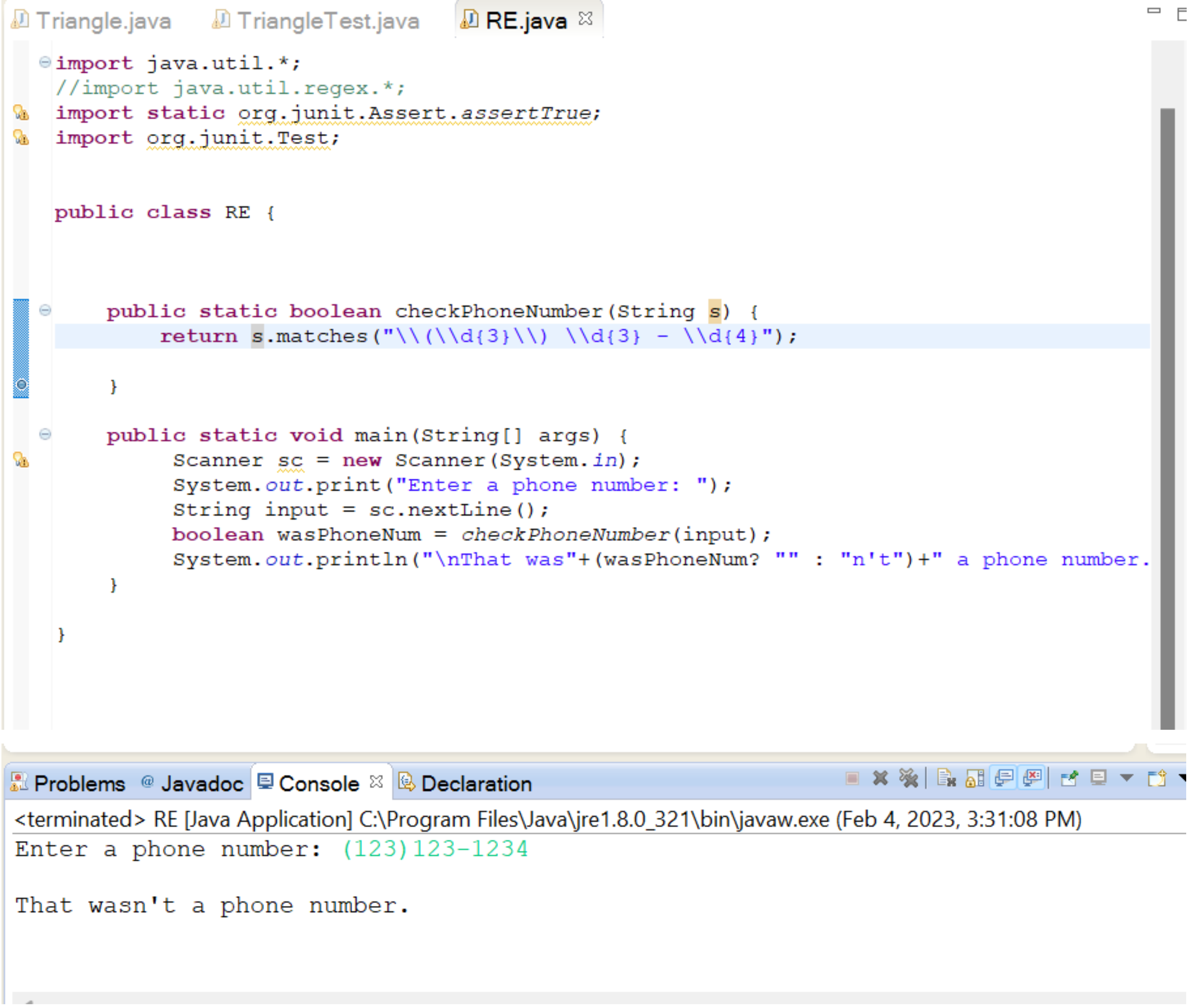
input3 test:



```
<terminated> RE [Java Application] C:\Program Files\Java\jre1.8.0_321\bin\javaw.exe (Feb 4, 2023, 3:21:58 PM)
Enter a phone number: 123 123 - 1234

That was a phone number.
```

4.

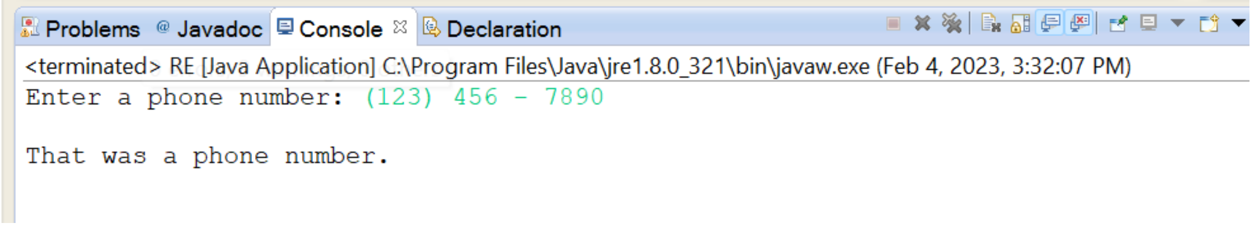
RE.java class + input1 test:



```java
import java.util.*;
//import java.util.regex.*;
import static org.junit.Assert.assertTrue;
import org.junit.Test;


public class RE {



    public static boolean checkPhoneNumber(String s) {
        return s.matches("\\(\\d{3}\\) \\d{3} - \\d{4}");

    }

    public static void main(String[] args) {
         Scanner sc = new Scanner(System.in);
         System.out.print("Enter a phone number: ");
         String input = sc.nextLine();
         boolean wasPhoneNum = checkPhoneNumber(input);
         System.out.println("\nThat was"+(wasPhoneNum? "" : "n't")+" a phone number.
    }

}
```

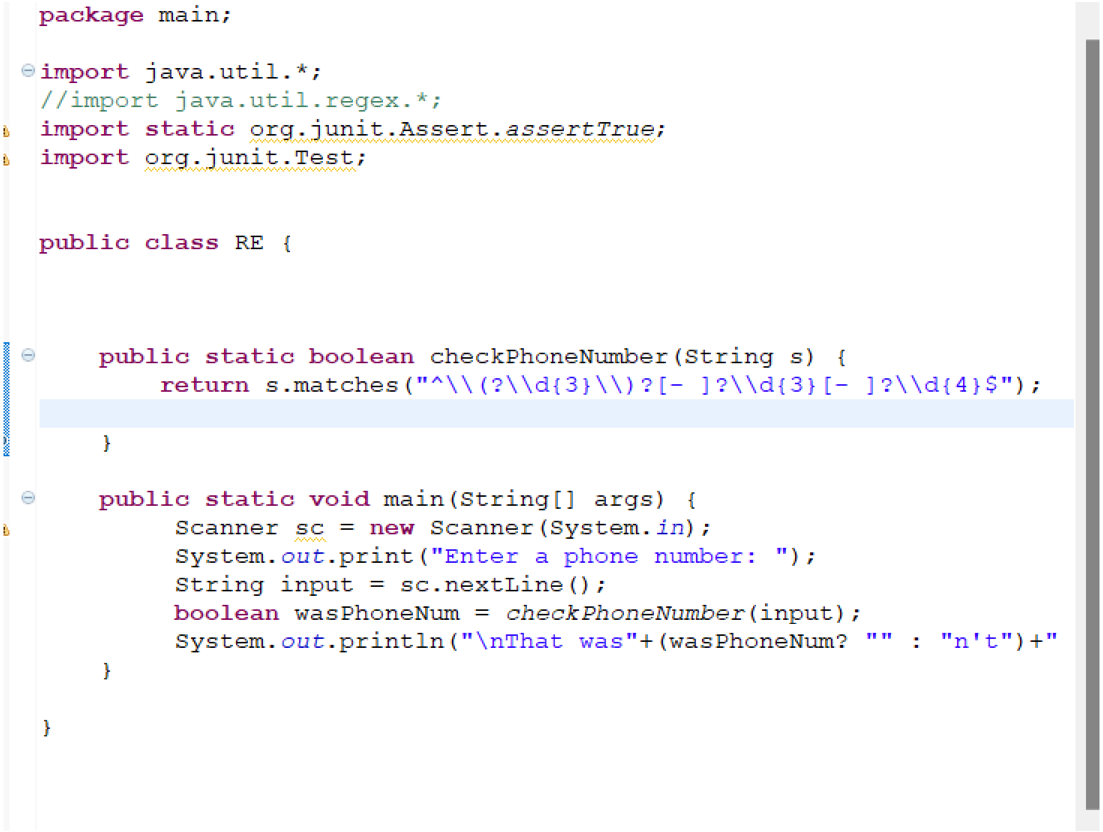
```
<terminated> RE [Java Application] C:\Program Files\Java\jre1.8.0_321\bin\javaw.exe (Feb 4, 2023, 3:31:08 PM)
Enter a phone number: (123)123-1234

That wasn't a phone number.
```

Input 2 test:

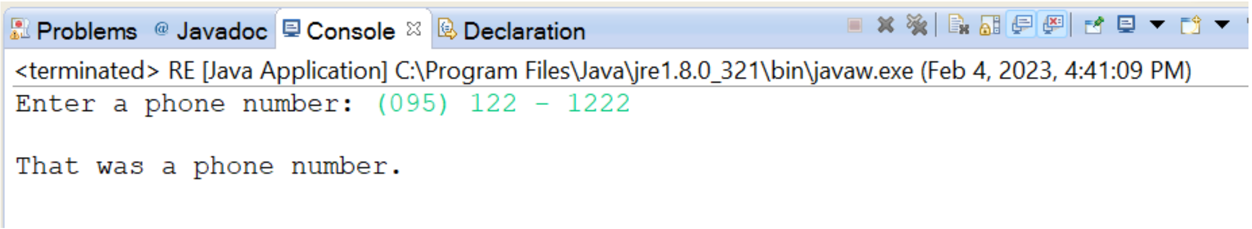


```
<terminated> RE [Java Application] C:\Program Files\Java\jre1.8.0_321\bin\javaw.exe (Feb 4, 2023, 3:32:07 PM)
Enter a phone number: (123) 456 - 7890

That was a phone number.
```

5.

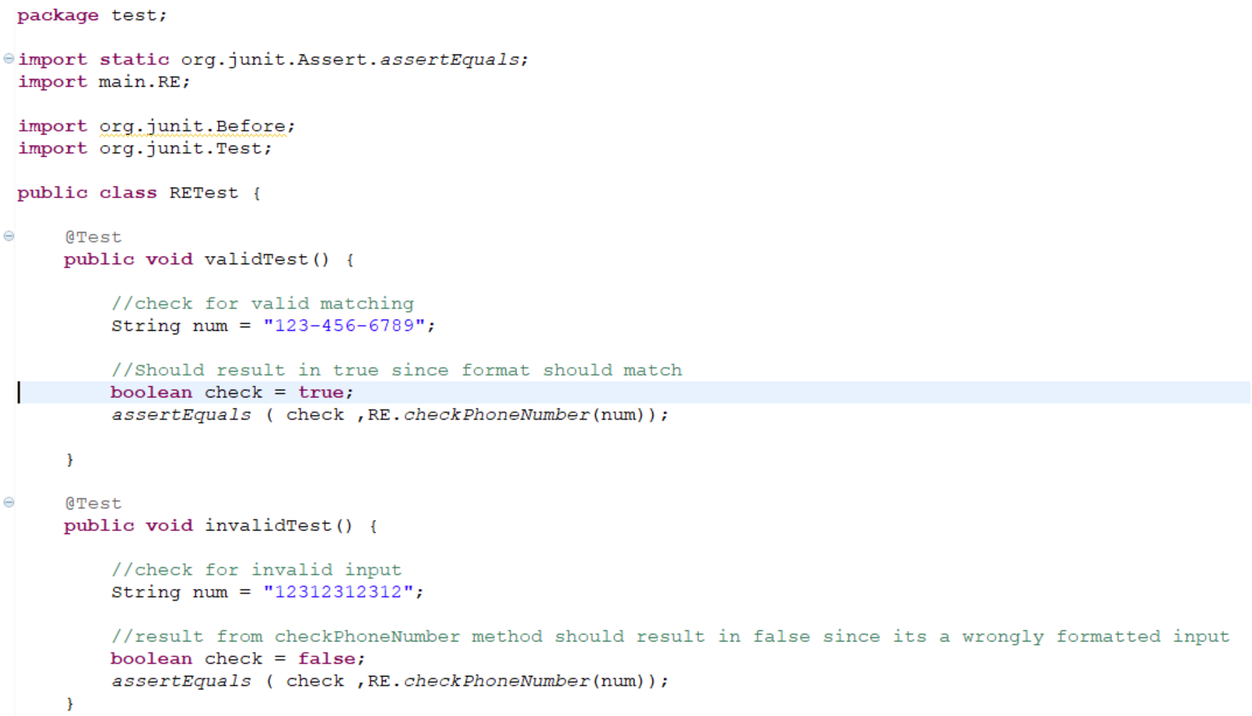


```java
package main;

import java.util.*;
//import java.util.regex.*;
import static org.junit.Assert.assertTrue;
import org.junit.Test;


public class RE {



    public static boolean checkPhoneNumber(String s) {
        return s.matches("^\\(?\\d{3}\\)?[- ]?\\d{3}[- ]?\\d{4}$");

    }

    public static void main(String[] args) {
         Scanner sc = new Scanner(System.in);
         System.out.print("Enter a phone number: ");
         String input = sc.nextLine();
         boolean wasPhoneNum = checkPhoneNumber(input);
         System.out.println("\nThat was"+(wasPhoneNum? "" : "n't")+"
    }

}
```

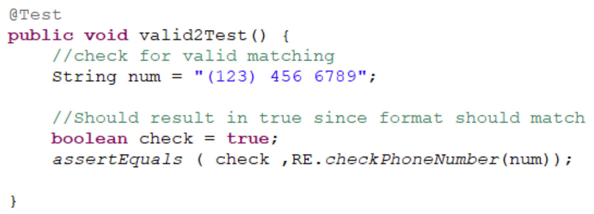


```
<terminated> RE [Java Application] C:\Program Files\Java\jre1.8.0_321\bin\javaw.exe (Feb 4, 2023, 4:41:09 PM)
Enter a phone number: (095) 122 - 1222

That was a phone number.
```

6.

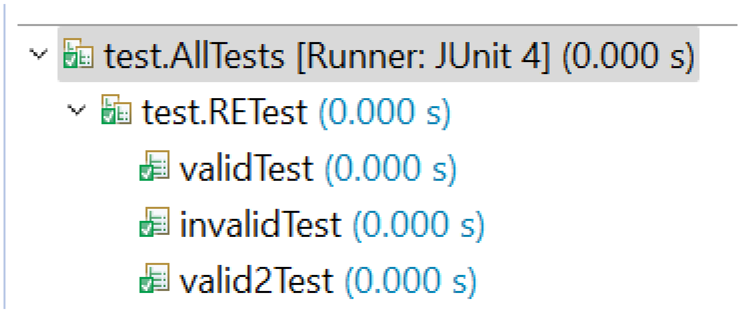


```
package test;

import static org.junit.Assert.assertEquals;
import main.RE;

import org.junit.Before;
import org.junit.Test;

public class RETest {

    @Test
    public void validTest() {

        //check for valid matching
        String num = "123-456-6789";

        //Should result in true since format should match
        boolean check = true;
        assertEquals ( check ,RE.checkPhoneNumber(num));

    }

    @Test
    public void invalidTest() {

        //check for invalid input
        String num = "12312312312";

        //result from checkPhoneNumber method should result in false since its a wrongly formatted input
        boolean check = false;
        assertEquals ( check ,RE.checkPhoneNumber(num));
    }
```



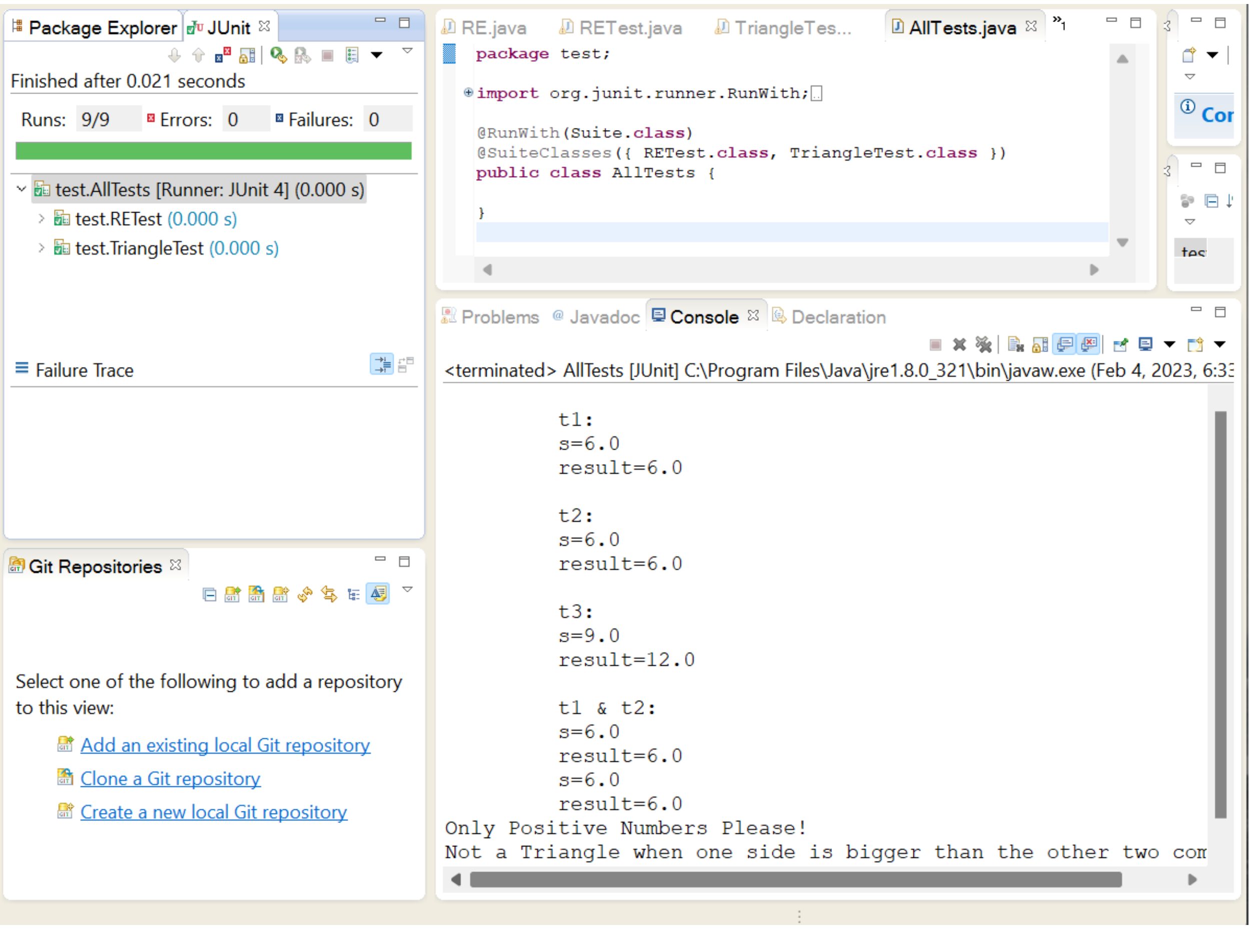
```
@Test
public void valid2Test() {
    //check for valid matching
    String num = "(123) 456 6789";

    //Should result in true since format should match
    boolean check = true;
    assertEquals ( check ,RE.checkPhoneNumber(num));

}
```

Results of Tests:



```
test.AllTests [Runner: JUnit 4] (0.000 s)
    test.RETest (0.000 s)
        validTest (0.000 s)
        invalidTest (0.000 s)
        valid2Test (0.000 s)
```

Q3. AllTests.java (Suite Class) + Results of tests of both test classes:



Finished after 0.021 seconds

Runs: 9/9 Errors: 0 Failures: 0

- test.AllTests [Runner: JUnit 4] (0.000 s)
  - test.RETest (0.000 s)
  - test.TriangleTest (0.000 s)

```java
package test;

import org.junit.runner.RunWith;

@RunWith(Suite.class)
@SuiteClasses({ RETest.class, TriangleTest.class })
public class AllTests {

}
```

Select one of the following to add a repository to this view:

- Add an existing local Git repository
- Clone a Git repository
- Create a new local Git repository

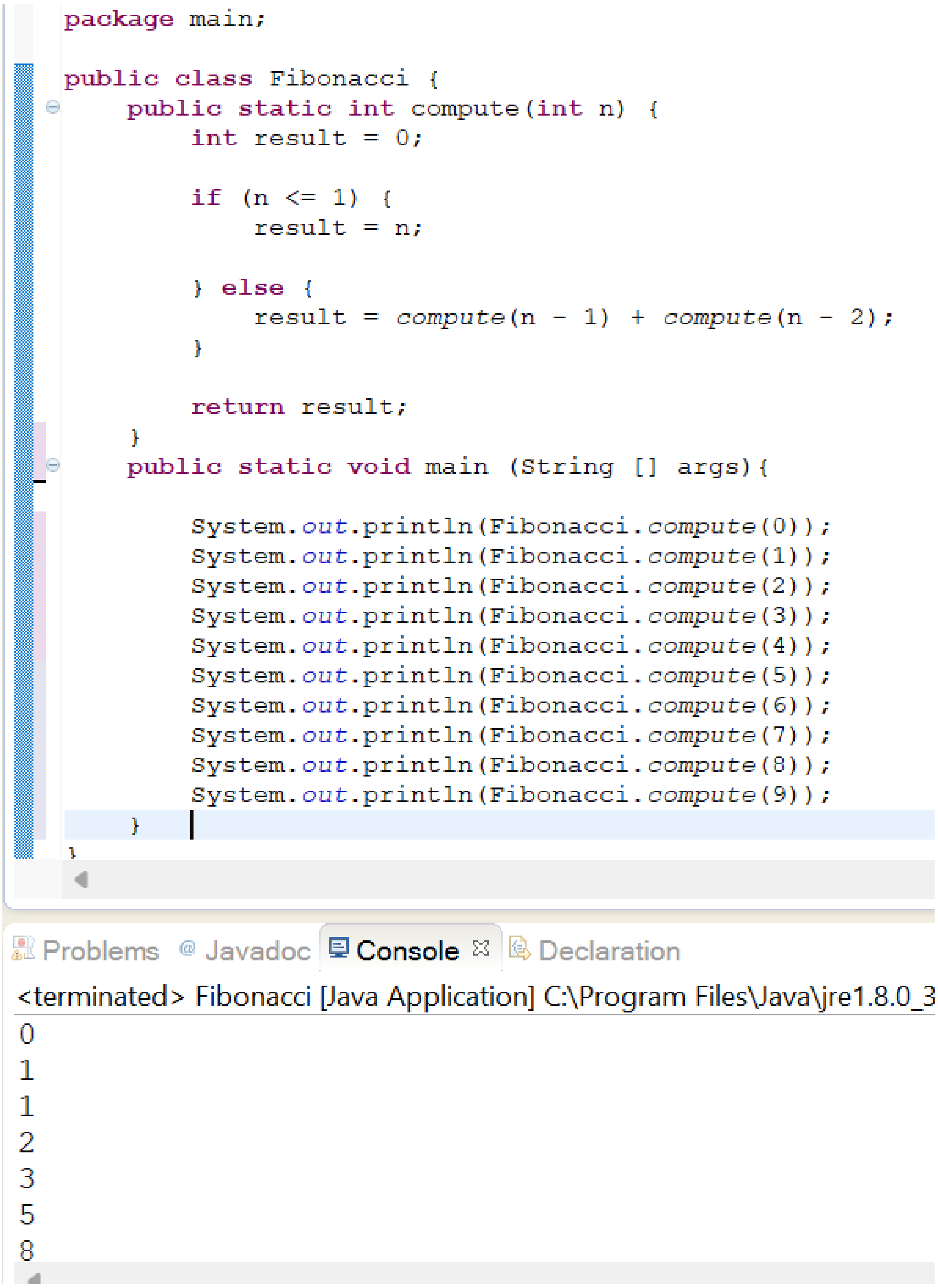
<terminated> AllTests [JUnit] C:\Program Files\Java\jre1.8.0_321\bin\javaw.exe (Feb 4, 2023, 6:33

```
         t1:
         s=6.0
         result=6.0

         t2:
         s=6.0
         result=6.0

         t3:
         s=9.0
         result=12.0

         t1 & t2:
         s=6.0
         result=6.0
         s=6.0
         result=6.0
Only Positive Numbers Please!
Not a Triangle when one side is bigger than the other two com
```

Part 2:

Q1.

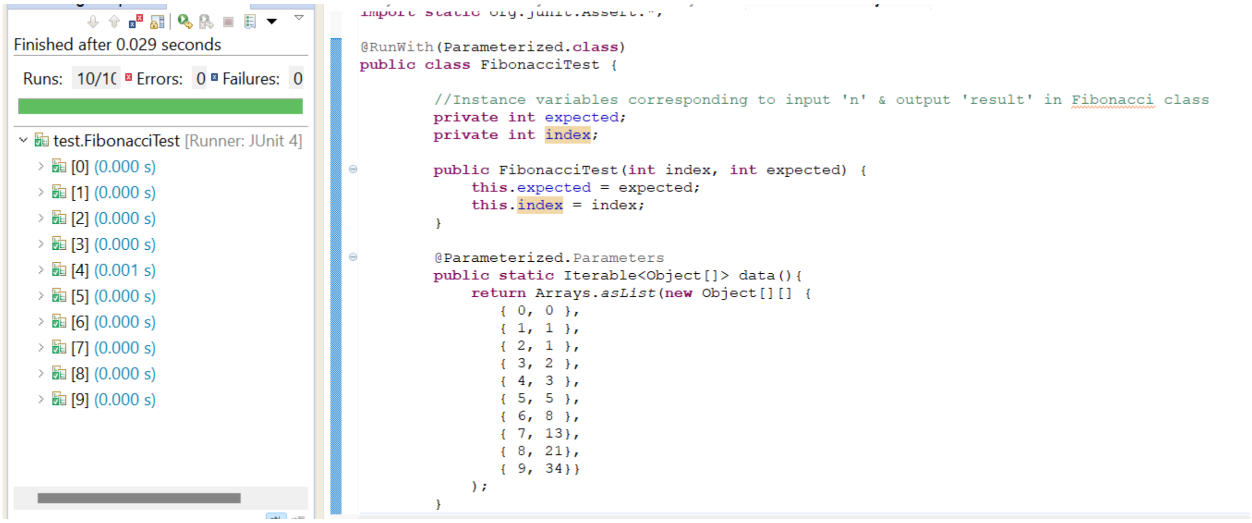
Fibonacci.java:



```
package main;

public class Fibonacci {
    public static int compute(int n) {
        int result = 0;

        if (n <= 1) {
            result = n;

        } else {
            result = compute(n - 1) + compute(n - 2);
        }

        return result;
    }
    public static void main (String [] args){

        System.out.println(Fibonacci.compute(0));
        System.out.println(Fibonacci.compute(1));
        System.out.println(Fibonacci.compute(2));
        System.out.println(Fibonacci.compute(3));
        System.out.println(Fibonacci.compute(4));
        System.out.println(Fibonacci.compute(5));
        System.out.println(Fibonacci.compute(6));
        System.out.println(Fibonacci.compute(7));
        System.out.println(Fibonacci.compute(8));
        System.out.println(Fibonacci.compute(9));
    }
}
```

```
<terminated> Fibonacci [Java Application] C:\Program Files\Java\jre1.8.0_3
0
1
1
2
3
5
8
```

Fibonacci test.java

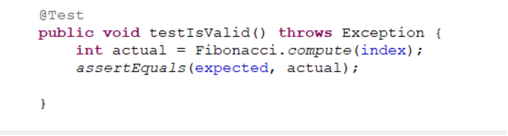


Finished after 0.029 seconds

Runs: 10/1(Errors: 0 Failures: 0

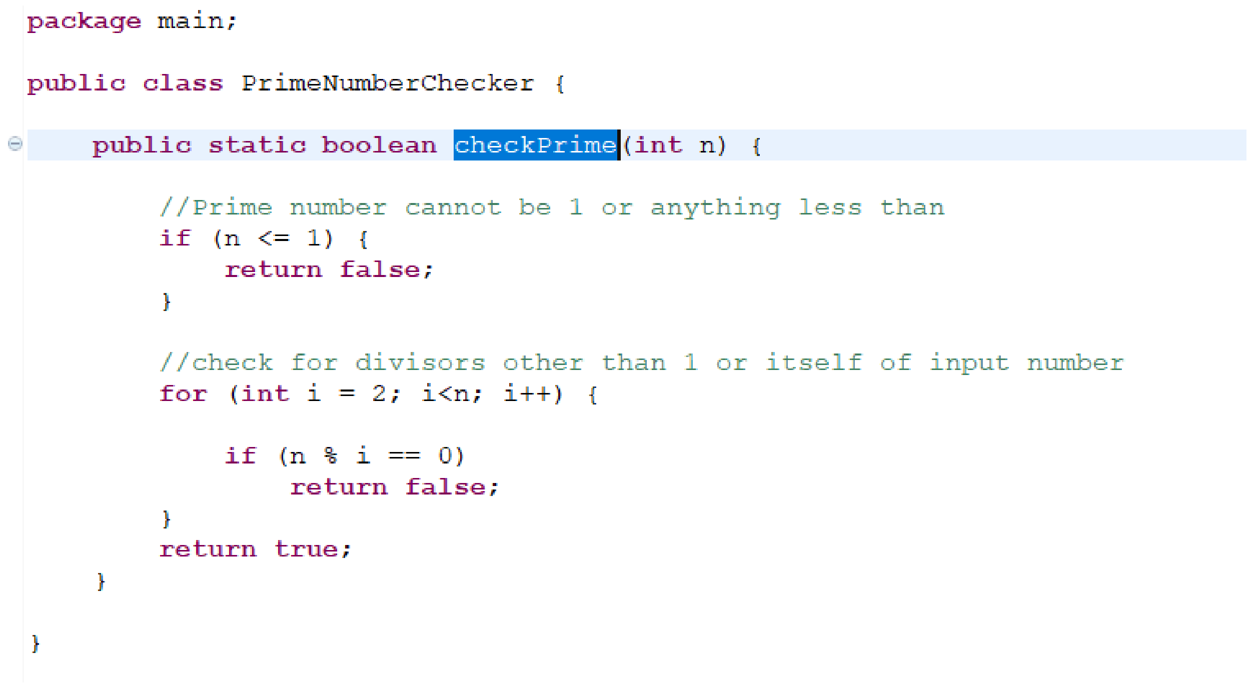
```
test.FibonacciTest [Runner: JUnit 4]
  [0] (0.000 s)
  [1] (0.000 s)
  [2] (0.000 s)
  [3] (0.000 s)
  [4] (0.001 s)
  [5] (0.000 s)
  [6] (0.000 s)
  [7] (0.000 s)
  [8] (0.000 s)
  [9] (0.000 s)
```

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

@RunWith(Parameterized.class)
public class FibonacciTest {

        //Instance variables corresponding to input 'n' & output 'result' in Fibonacci class
        private int expected;
        private int index;

        public FibonacciTest(int index, int expected) {
            this.expected = expected;
            this.index = index;
        }

        @Parameterized.Parameters
        public static Iterable<Object[]> data(){
            return Arrays.asList(new Object[][] {
                { 0, 0 },
                { 1, 1 },
                { 2, 1 },
                { 3, 2 },
                { 4, 3 },
                { 5, 5 },
                { 6, 8 },
                { 7, 13},
                { 8, 21},
                { 9, 34}}
            );
        }
```



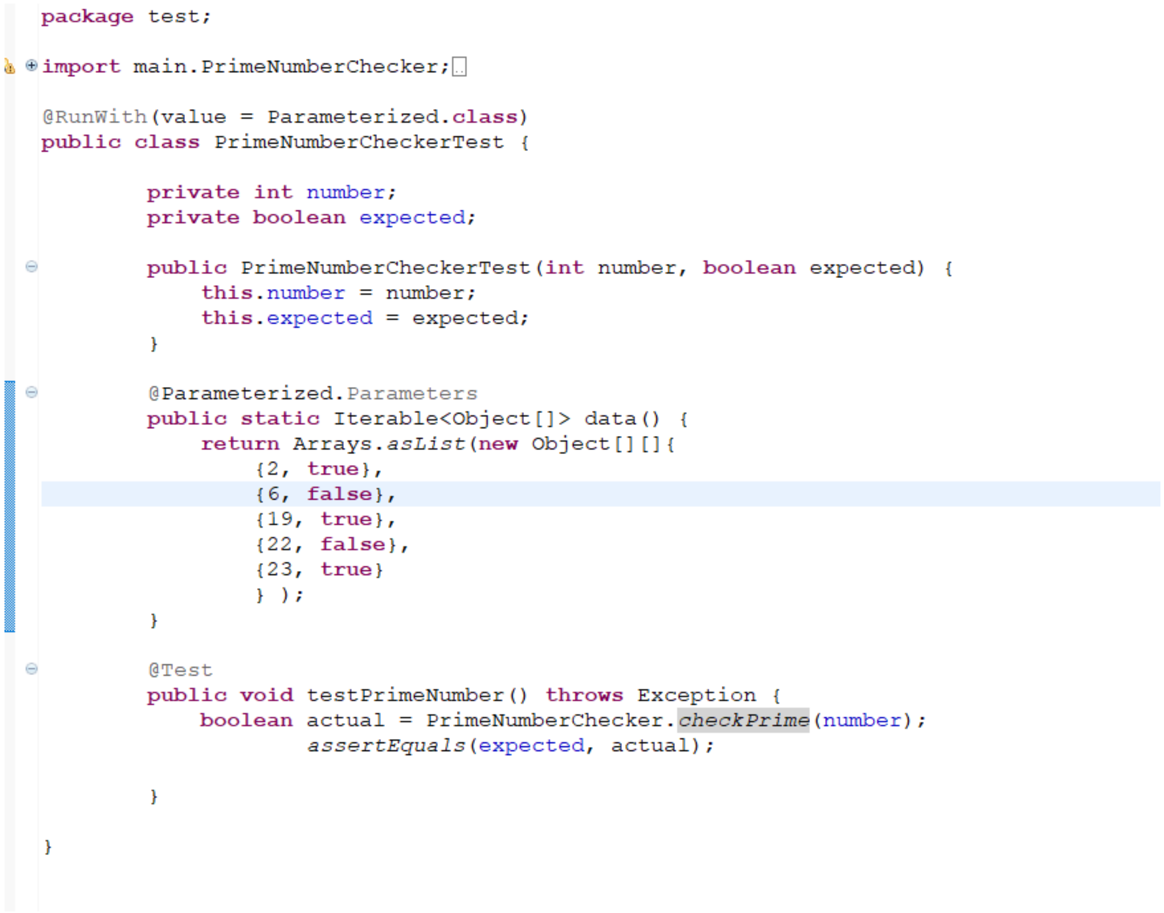
```java
        @Test
        public void testIsValid() throws Exception {
            int actual = Fibonacci.compute(index);
            assertEquals(expected, actual);

        }
```

Q2.
PrimeNumberChecker.java main class:



```java
package main;

public class PrimeNumberChecker {

    public static boolean checkPrime(int n) {

        //Prime number cannot be 1 or anything less than
        if (n <= 1) {
            return false;
        }

        //check for divisors other than 1 or itself of input number
        for (int i = 2; i<n; i++) {

            if (n % i == 0)
                return false;
        }
        return true;
    }

}
```

PrimeNumberCheckerTest.java test class:

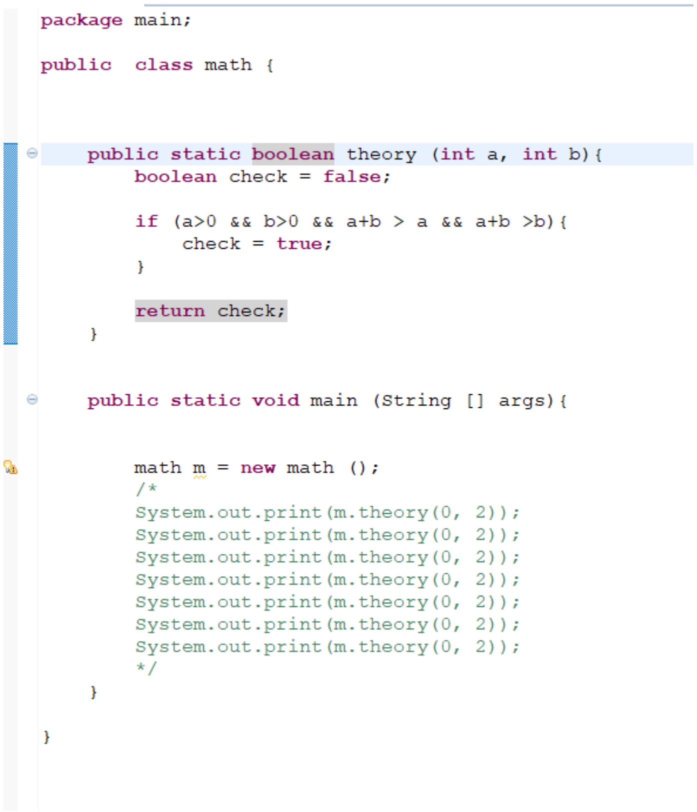


```java
package test;

import main.PrimeNumberChecker;

@RunWith(value = Parameterized.class)
public class PrimeNumberCheckerTest {

        private int number;
        private boolean expected;

        public PrimeNumberCheckerTest(int number, boolean expected) {
            this.number = number;
            this.expected = expected;
        }

        @Parameterized.Parameters
        public static Iterable<Object[]> data() {
            return Arrays.asList(new Object[][]{
                {2, true},
                {6, false},
                {19, true},
                {22, false},
                {23, true}
                } );
        }

        @Test
        public void testPrimeNumber() throws Exception {
            boolean actual = PrimeNumberChecker.checkPrime(number);
                    assertEquals(expected, actual);

        }

}
```

Q3.

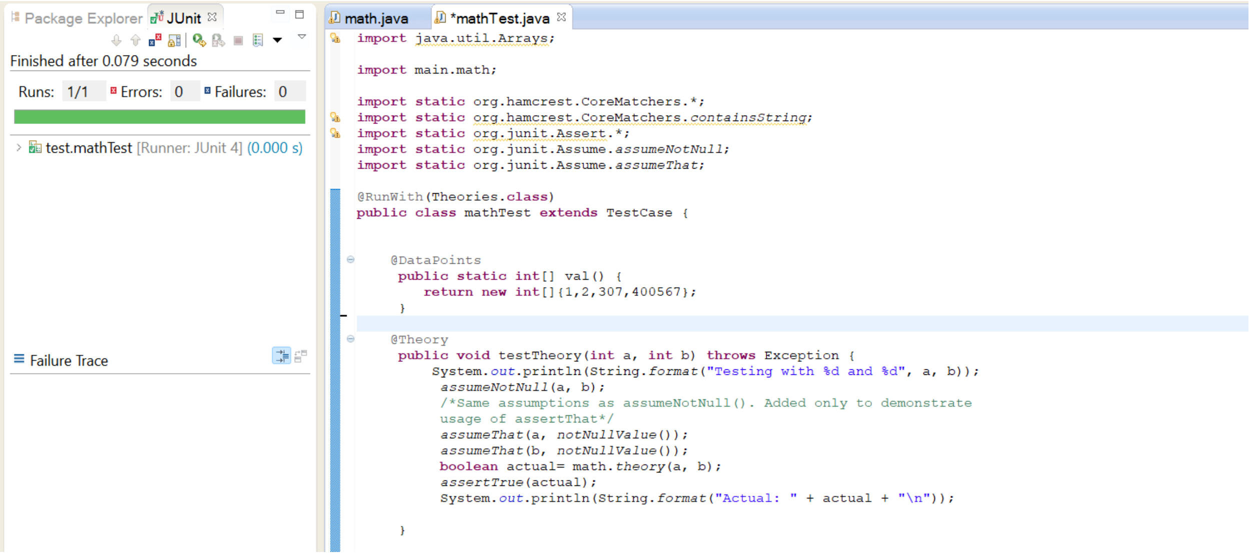
1.

Main (Math class):



```
package main;

public  class math {


    public static boolean theory (int a, int b){
        boolean check = false;

        if (a>0 && b>0 && a+b > a && a+b >b){
            check = true;
        }

        return check;
    }


    public static void main (String [] args){


        math m = new math ();
        /*
        System.out.print(m.theory(0, 2));
        System.out.print(m.theory(0, 2));
        System.out.print(m.theory(0, 2));
        System.out.print(m.theory(0, 2));
        System.out.print(m.theory(0, 2));
        System.out.print(m.theory(0, 2));
        System.out.print(m.theory(0, 2));
        */
    }

}
```

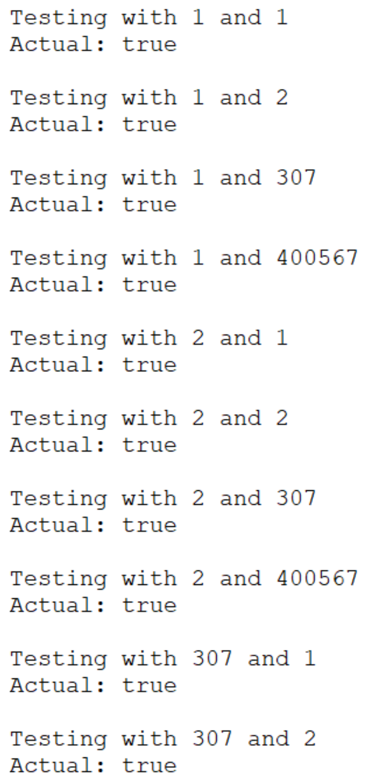
Test Class:



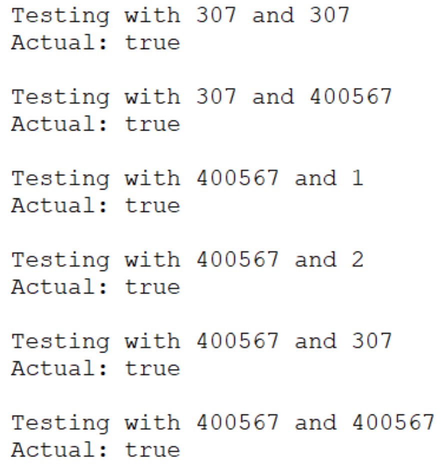
```
Finished after 0.079 seconds

Runs: 1/1    Errors: 0    Failures: 0

test.mathTest [Runner: JUnit 4] (0.000 s)

Failure Trace
```

```
import java.util.Arrays;

import main.math;

import static org.hamcrest.CoreMatchers.*;
import static org.hamcrest.CoreMatchers.containsString;
import static org.junit.Assert.*;
import static org.junit.Assume.assumeNotNull;
import static org.junit.Assume.assumeThat;

@RunWith(Theories.class)
public class mathTest extends TestCase {


    @DataPoints
     public static int[] val() {
        return new int[]{1,2,307,400567};
     }

    @Theory
     public void testTheory(int a, int b) throws Exception {
         System.out.println(String.format("Testing with %d and %d", a, b));
          assumeNotNull(a, b);
          /*Same assumptions as assumeNotNull(). Added only to demonstrate
          usage of assertThat*/
          assumeThat(a, notNullValue());
          assumeThat(b, notNullValue());
          boolean actual= math.theory(a, b);
          assertTrue(actual);
          System.out.println(String.format("Actual: " + actual + "\n"));

     }
```

Results/Output of datapoint values of theory method:



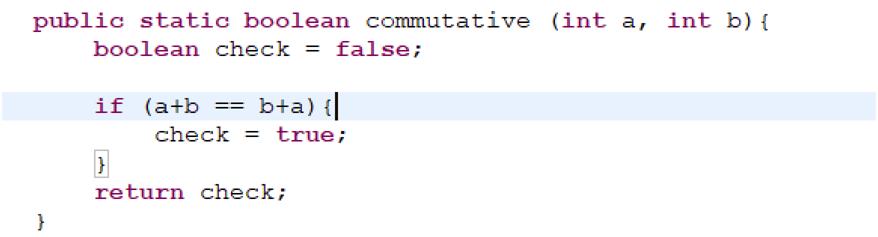
```
Testing with 1 and 1
Actual: true

Testing with 1 and 2
Actual: true

Testing with 1 and 307
Actual: true

Testing with 1 and 400567
Actual: true

Testing with 2 and 1
Actual: true

Testing with 2 and 2
Actual: true

Testing with 2 and 307
Actual: true

Testing with 2 and 400567
Actual: true

Testing with 307 and 1
Actual: true

Testing with 307 and 2
Actual: true
```



```
Testing with 307 and 307
Actual: true

Testing with 307 and 400567
Actual: true

Testing with 400567 and 1
Actual: true

Testing with 400567 and 2
Actual: true

Testing with 400567 and 307
Actual: true

Testing with 400567 and 400567
Actual: true
```

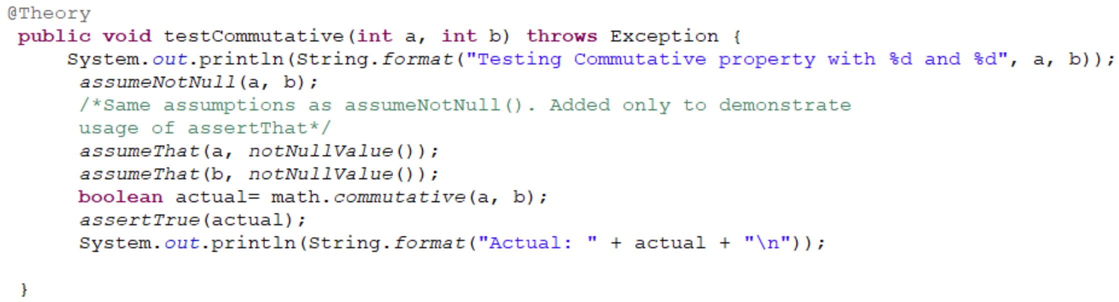
2.

Commutative property in math.java class:



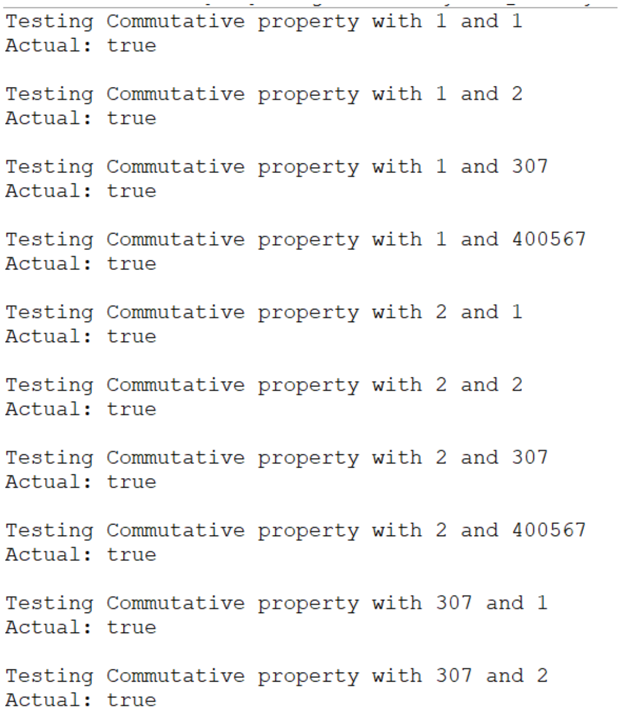
```
public static boolean commutative (int a, int b){
    boolean check = false;

    if (a+b == b+a){
        check = true;
    }
    return check;
}
```

Testing the commutative property in test class:



```
@Theory
 public void testCommutative(int a, int b) throws Exception {
     System.out.println(String.format("Testing Commutative property with %d and %d", a, b));
      assumeNotNull(a, b);
      /*Same assumptions as assumeNotNull(). Added only to demonstrate
      usage of assertThat*/
      assumeThat(a, notNullValue());
      assumeThat(b, notNullValue());
      boolean actual= math.commutative(a, b);
      assertTrue(actual);
      System.out.println(String.format("Actual: " + actual + "\n"));

 }
```

<u>Results/Output of datapoint values of theory method:</u>

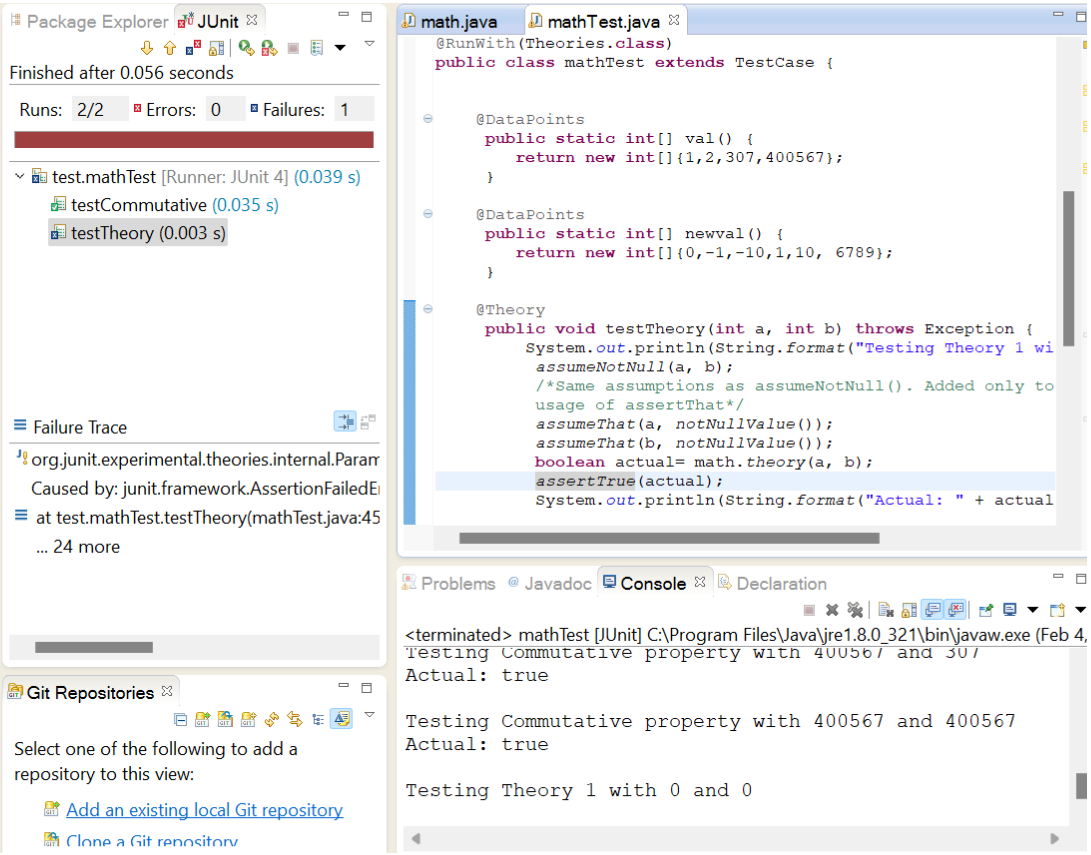


```
Testing Commutative property with 1 and 1
Actual: true

Testing Commutative property with 1 and 2
Actual: true

Testing Commutative property with 1 and 307
Actual: true

Testing Commutative property with 1 and 400567
Actual: true

Testing Commutative property with 2 and 1
Actual: true

Testing Commutative property with 2 and 2
Actual: true

Testing Commutative property with 2 and 307
Actual: true

Testing Commutative property with 2 and 400567
Actual: true

Testing Commutative property with 307 and 1
Actual: true

Testing Commutative property with 307 and 2
Actual: true
```

```
Testing Commutative property with 307 and 2
Actual: true

Testing Commutative property with 307 and 307
Actual: true

Testing Commutative property with 307 and 400567
Actual: true

Testing Commutative property with 400567 and 1
Actual: true

Testing Commutative property with 400567 and 2
Actual: true

Testing Commutative property with 400567 and 307
Actual: true

Testing Commutative property with 400567 and 400567
Actual: true
```

3. For the first mathematical statement/theory, the results should be true for all positive integers. For equal to 0 or negative, they will return false however, since the theory only works for positive numbers greater than 0.

   For the commutative property, all combinations will return true regardless.



Finished after 0.056 seconds

Runs: 2/2 Errors: 0 Failures: 1

```
test.mathTest [Runner: JUnit 4] (0.039 s)
    testCommutative (0.035 s)
    testTheory (0.003 s)
```

Failure Trace

```
org.junit.experimental.theories.internal.Param
Caused by: junit.framework.AssertionFailedE
at test.mathTest.testTheory(mathTest.java:45
... 24 more
```

Select one of the following to add a repository to this view:

- Add an existing local Git repository
- Clone a Git repository

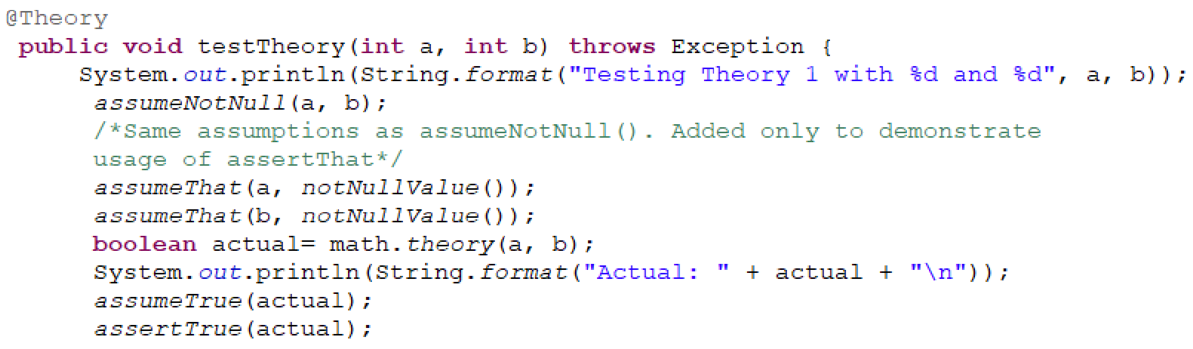
```java
@RunWith(Theories.class)
public class mathTest extends TestCase {


    @DataPoints
     public static int[] val() {
        return new int[]{1,2,307,400567};
     }

    @DataPoints
     public static int[] newval() {
        return new int[]{0,-1,-10,1,10, 6789};
     }

    @Theory
     public void testTheory(int a, int b) throws Exception {
         System.out.println(String.format("Testing Theory 1 wi
          assumeNotNull(a, b);
          /*Same assumptions as assumeNotNull(). Added only to
          usage of assertThat*/
          assumeThat(a, notNullValue());
          assumeThat(b, notNullValue());
          boolean actual= math.theory(a, b);
          assertTrue(actual);
          System.out.println(String.format("Actual: " + actual
```

<terminated> mathTest [JUnit] C:\Program Files\Java\jre1.8.0_321\bin\javaw.exe (Feb 4,

```
Testing Commutative property with 400567 and 307
Actual: true

Testing Commutative property with 400567 and 400567
Actual: true

Testing Theory 1 with 0 and 0
```

As seen above, the tests stop running for the testTheory () method, as the value 0 doesn't satisfy the requirements of the first mathematical statement/theory, and detects a false evaluation while expecting true, causing it to fail. This would occur for all the data point values within newval () that are less than or equal to 0.

4.



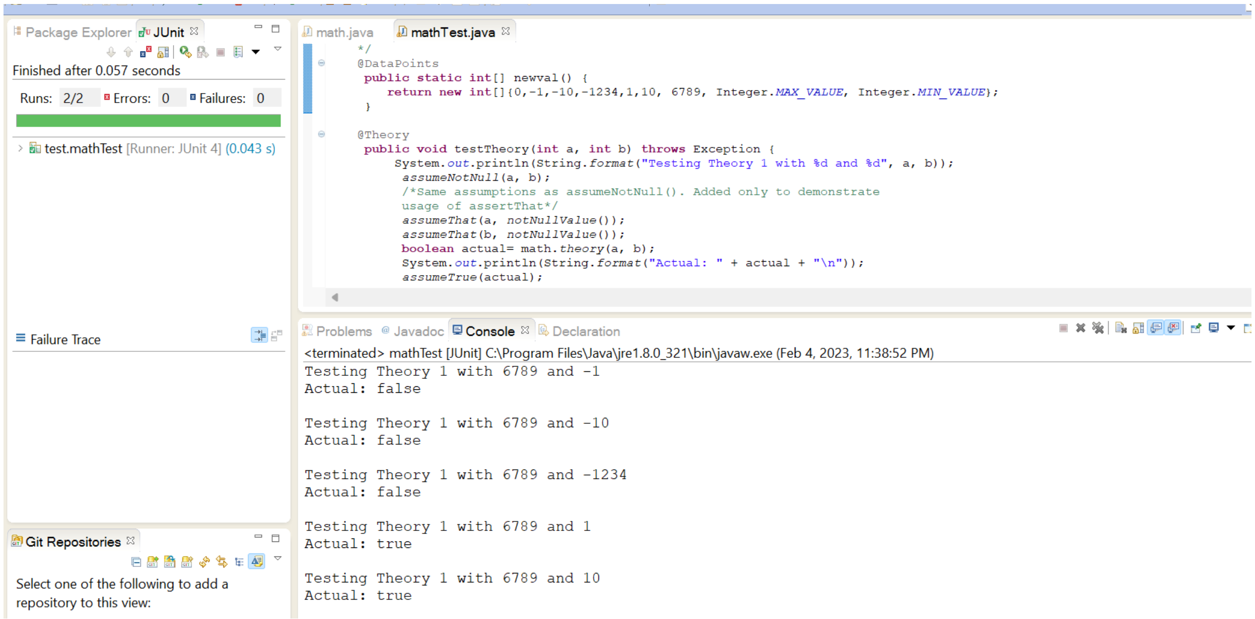
```
@Theory
 public void testTheory(int a, int b) throws Exception {
     System.out.println(String.format("Testing Theory 1 with %d and %d", a, b));
      assumeNotNull(a, b);
      /*Same assumptions as assumeNotNull(). Added only to demonstrate
      usage of assertThat*/
      assumeThat(a, notNullValue());
      assumeThat(b, notNullValue());
      boolean actual= math.theory(a, b);
      System.out.println(String.format("Actual: " + actual + "\n"));
      assumeTrue(actual);
      assertTrue(actual);
```

The assumption “AssumeTrue ()” was added to check for false cases within the testTheory method (note: this method tests the first mathematical statement specified as part of Q3). This causes the false cases (negative or less than 0 values) to simply be ignored without failing the entire test case/program.

5. The testing results will be similar as before, where for the first mathematical statement/theory, the results should be true for all positive integers. However for values equal to 0 or negative, they will return false,since the theory only works for positive numbers greater than 0.

For the commutative property, all combinations will return true regardless.

<u>Test Results/Output of datapoint values of updated theory method:</u>

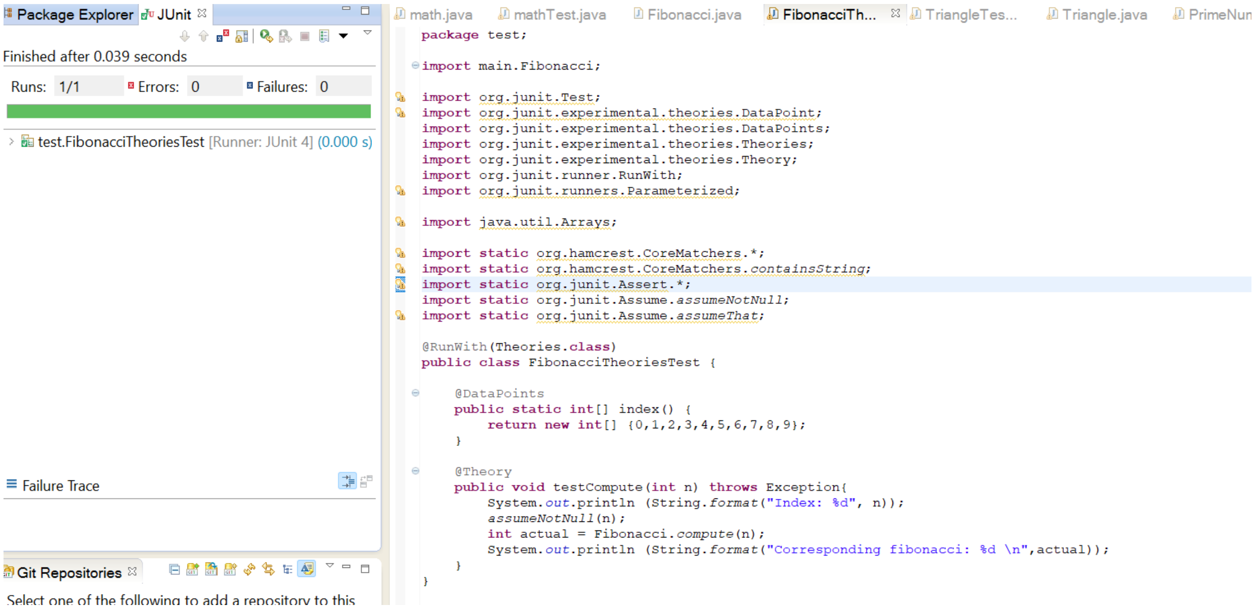


```java
    */
    @DataPoints
     public static int[] newval() {
        return new int[]{0,-1,-10,-1234,1,10, 6789, Integer.MAX_VALUE, Integer.MIN_VALUE};
     }

    @Theory
     public void testTheory(int a, int b) throws Exception {
         System.out.println(String.format("Testing Theory 1 with %d and %d", a, b));
          assumeNotNull(a, b);
          /*Same assumptions as assumeNotNull(). Added only to demonstrate
          usage of assertThat*/
          assumeThat(a, notNullValue());
          assumeThat(b, notNullValue());
          boolean actual= math.theory(a, b);
          System.out.println(String.format("Actual: " + actual + "\n"));
          assumeTrue(actual);
```

```
<terminated> mathTest [JUnit] C:\Program Files\Java\jre1.8.0_321\bin\javaw.exe (Feb 4, 2023, 11:38:52 PM)
Testing Theory 1 with 6789 and -1
Actual: false

Testing Theory 1 with 6789 and -10
Actual: false

Testing Theory 1 with 6789 and -1234
Actual: false

Testing Theory 1 with 6789 and 1
Actual: true

Testing Theory 1 with 6789 and 10
Actual: true
```

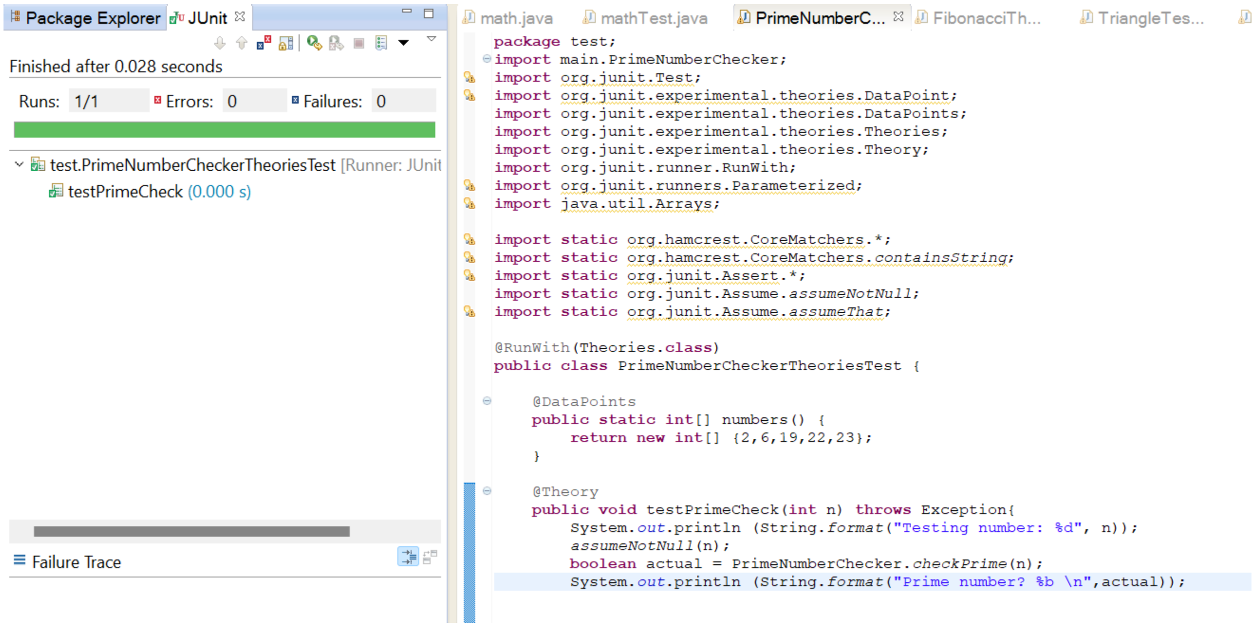
Q4.

Updated FibonacciTest class using JUnit Theories:



```java
package test;

import main.Fibonacci;

import org.junit.Test;
import org.junit.experimental.theories.DataPoint;
import org.junit.experimental.theories.DataPoints;
import org.junit.experimental.theories.Theories;
import org.junit.experimental.theories.Theory;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;

import java.util.Arrays;

import static org.hamcrest.CoreMatchers.*;
import static org.hamcrest.CoreMatchers.containsString;
import static org.junit.Assert.*;
import static org.junit.Assume.assumeNotNull;
import static org.junit.Assume.assumeThat;

@RunWith(Theories.class)
public class FibonacciTheoriesTest {

    @DataPoints
    public static int[] index() {
        return new int[] {0,1,2,3,4,5,6,7,8,9};
    }

    @Theory
    public void testCompute(int n) throws Exception{
        System.out.println (String.format("Index: %d", n));
        assumeNotNull(n);
        int actual = Fibonacci.compute(n);
        System.out.println (String.format("Corresponding fibonacci: %d \n",actual));
    }
}
```

Updated PrimeNumberCheckerTest class using JUnit Theories:



Finished after 0.028 seconds

Runs: 1/1 Errors: 0 Failures: 0

test.PrimeNumberCheckerTheoriesTest [Runner: JUnit

testPrimeCheck (0.000 s)

Failure Trace

```java
package test;
import main.PrimeNumberChecker;
import org.junit.Test;
import org.junit.experimental.theories.DataPoint;
import org.junit.experimental.theories.DataPoints;
import org.junit.experimental.theories.Theories;
import org.junit.experimental.theories.Theory;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;
import java.util.Arrays;

import static org.hamcrest.CoreMatchers.*;
import static org.hamcrest.CoreMatchers.containsString;
import static org.junit.Assert.*;
import static org.junit.Assume.assumeNotNull;
import static org.junit.Assume.assumeThat;

@RunWith(Theories.class)
public class PrimeNumberCheckerTheoriesTest {

    @DataPoints
    public static int[] numbers() {
        return new int[] {2,6,19,22,23};
    }

    @Theory
    public void testPrimeCheck(int n) throws Exception{
        System.out.println (String.format("Testing number: %d", n));
        assumeNotNull(n);
        boolean actual = PrimeNumberChecker.checkPrime(n);
        System.out.println (String.format("Prime number? %b \n",actual));
```