Bahria University

Information technology Department

Course: Software Testing

Class: BSIT VII A

Submitted By:

Hifsa Basharat 45903
Haris Anwer 45902
M Laraib Kiayani 45916

Submitted To:

Ms. Sumeera Hashmi

Dated: 05-Dec-2019

Department Of Information Technology Fall 2019

REPORT

Testing Tool

JUNIT Test Suite With JaCoCoverage Analysis

Table of content

INTRODUCTION TO SOFTWARE TESTING

Software Testing is necessary because we all make mistakes. Some of those mistakes are unimportant, but some of them are expensive or dangerous. We need to check everything and anything we produce because things can always go wrong – humans make mistakes all the time.

Since we assume that our work may have mistakes, hence we all need to check our own work. However some mistakes come from bad assumptions and blind spots, so we might make the same mistakes when we check our own work as we made when we did it. So we may not notice the flaws in what we have done.

Ideally, we should get someone else to check our work because another person is more likely to spot the flaws.

There are several reasons which clearly tells us as why Software Testing is important and what are the major things that we should consider while testing of any product or application.

Software testing is very important because of the following reasons:

1. Software testing is really required to point out the defects and errors that were made during the development phases.

   Example: Programmers may make a mistake during the implementation of the software. There could be many reasons for this like lack of experience of the programmer, lack of knowledge of the programming language, insufficient experience in the domain, incorrect implementation of the algorithm due to complex logic or simply human error.

2. It's essential since it makes sure that the customer finds the organization reliable and their satisfaction in the application is maintained.

Example: If the customer does not find the testing organization reliable or is not satisfied with the quality of the deliverable, then they may switch to a competitor organization.

3. It is very important to ensure the Quality of the product. Quality product delivered to the customers helps in gaining their confidence. (Know more about Software Quality)

   Example: As explained in the previous point, delivering good quality product on time builds the customers confidence in the team and the organization.

4. Testing is necessary in order to provide the facilities to the customers like the delivery of high quality product or software application which requires lower maintenance cost and hence results into more accurate, consistent and reliable results.

   Example: High quality product typically has fewer defects and requires lesser maintenance effort, which in turn means reduced costs.

5. Testing is required for an effective performance of software application or product.
6. It's important to ensure that the application should not result into any failures because it can be very expensive in the future or in the later stages of the development.

   Example: Proper testing ensures that bugs and issues are detected early in the life cycle of the product or application. If defects related to requirements or design are detected late in the life cycle, it can be very expensive to fix them since this might require redesign, re-implementation and retesting of the application.

7. It's required to stay in the business. Users are not inclined to use software that has bugs. They may not adopt a software if they are not happy with the stability of the application.

JUNIT

JUnit is a unit testing framework for Java programming language. It plays a crucial role test-driven development, and is a family of unit testing frameworks collectively known as JUnit.

JUnit promotes the idea of "first testing then coding", which emphasizes on setting up the test data for a piece of code that can be tested first and then implemented. This approach is like "test a little, code a little, test a little, code a little." It increases the productivity of the programmer and the stability of program code, which in turn reduces the stress on the programmer and the time spent on debugging.

Features of Junit

- JUnit is an open source framework, which is used for writing and running tests.
- Provides annotations to identify test methods.
- Provides assertions for testing expected results.
- Provides test runners for running tests.
- JUnit tests allow you to write codes faster, which increases quality.
- JUnit is elegantly simple. It is less complex and takes less time.
- JUnit tests can be run automatically and they check their own results and provide immediate feedback. There's no need to manually comb through a report of test results.
- JUnit tests can be organized into test suites containing test cases and even other test suites.
- JUnit shows test progress in a bar that is green if the test is running smoothly, and it turns red when a test fails.

What is a Unit Test Case ?

A Unit Test Case is a part of code, which ensures that another part of code (method) works as expected.

To achieve the desired results quickly, a test framework is required. JUnit is a perfect unit test framework for Java programming language.

A formal written unit test case is characterized by a known input and an expected output, which is worked out before the test is executed. The known input should test a precondition and the expected output should test a post-condition.

There must be at least two unit test cases for each requirement − one positive test and one negative test. If a requirement has sub-requirements, each sub-requirement must have at least two test cases as positive and negative.

FUNCTIONS FOR TESTING

- Algebraic Expressions
  Testing of Three basic algebraic expressions
  (a+b)^2 = a^2+2ab+b^2
  (a-b)^2= a^2+2ab+b^2
  (a^2-b^2)=(a+b)(a-b)

- Armstrong Number
  An Armstrong number is a number such that the sum of its digits raised to the third power is equal to the number, itself

- Concatenation
  Concatenation, in the context of programming, is the operation of joining two strings together. The term "concatenation" literally means to merge two things together. Also known as string concatenation.

- Cyclomatic Complexity
  Cyclomatic complexity is generally used to measure the complexity at the class or the method level
  F(n)= E - N + 2*P

- Greatest Common Factor
  The greatest common divisor of two or more integers, which are not all zero, is the largest positive integer that divides each of the integers.

- Least Common Factor
  the least common multiple, lowest common multiple, or smallest common multiple of two integers a and b, usually denoted by LCM, is the smallest positive integer that is divisible by both a and b.

- Smallest Number
  Finding smallest number among three integer values.

- Square of Number
  Testing the result of square of number, multiplying a number by itself.

- Random Factorial with Timeout
  The factorial of a positive integer n, denoted by n!, is the product of all positive integers less than or equal to n.

- Scaler Multiplication
  Multiplication of a vector by a scalar (where the product is a vector), and must be distinguished from inner product of two vectors (where the product is a scalar).

- Vectors Equality
  To check weather the vectors have equal values.

- Triangle Types
  To check weather the triangle is Isosceles, Scalene or equilateral by measuring dimensions.

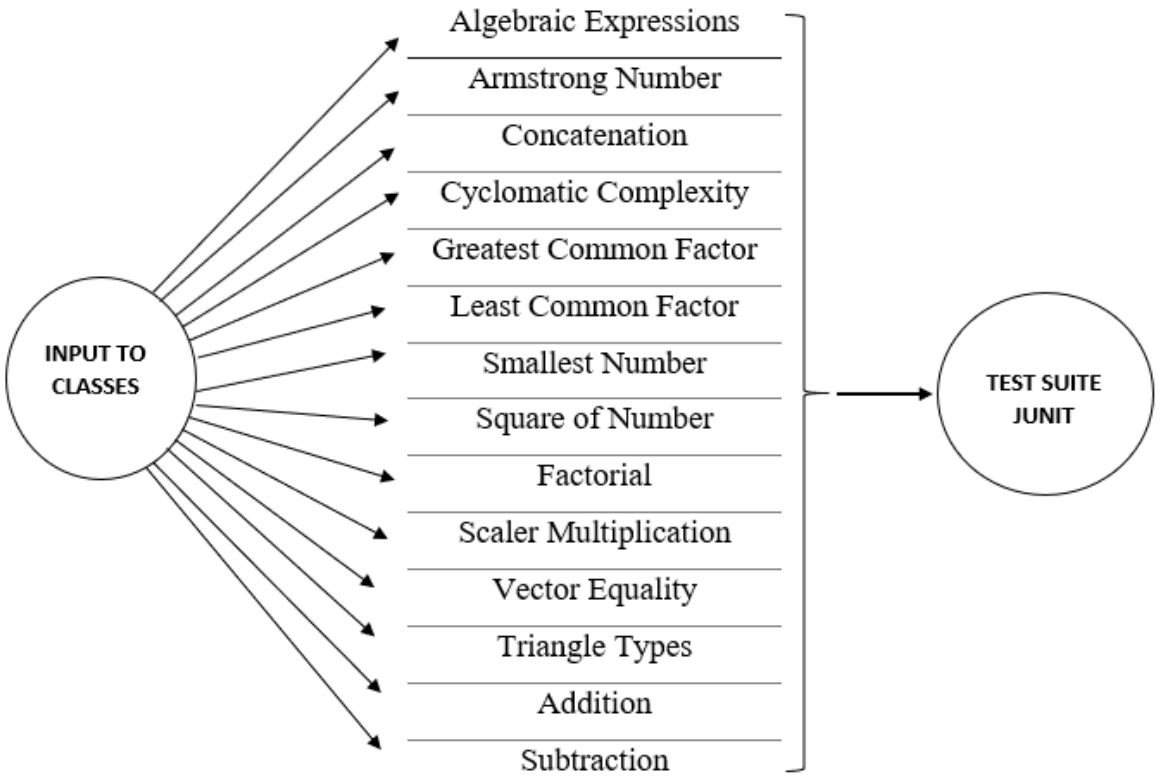
- Simple Calculator Functions
  Simple mathematical functions include Addition, Subtraction & Division to two integers.

TEST SUITE

A test suite, less commonly known as a 'validation suite', is a collection of test cases that are intended to be used to test a software program to show that it has some specified set of behaviors.

Difference between Test Cases & Test Suite:

Test case is a set of test inputs, execution conditions, and expected results developed to test a particular execution path. Usually, the case is a single method. Test suite is a list of related test cases. Suite may contain common initialization and cleanup routines specific to the cases included.



TEST CASES

CODE OF TEST SUITE

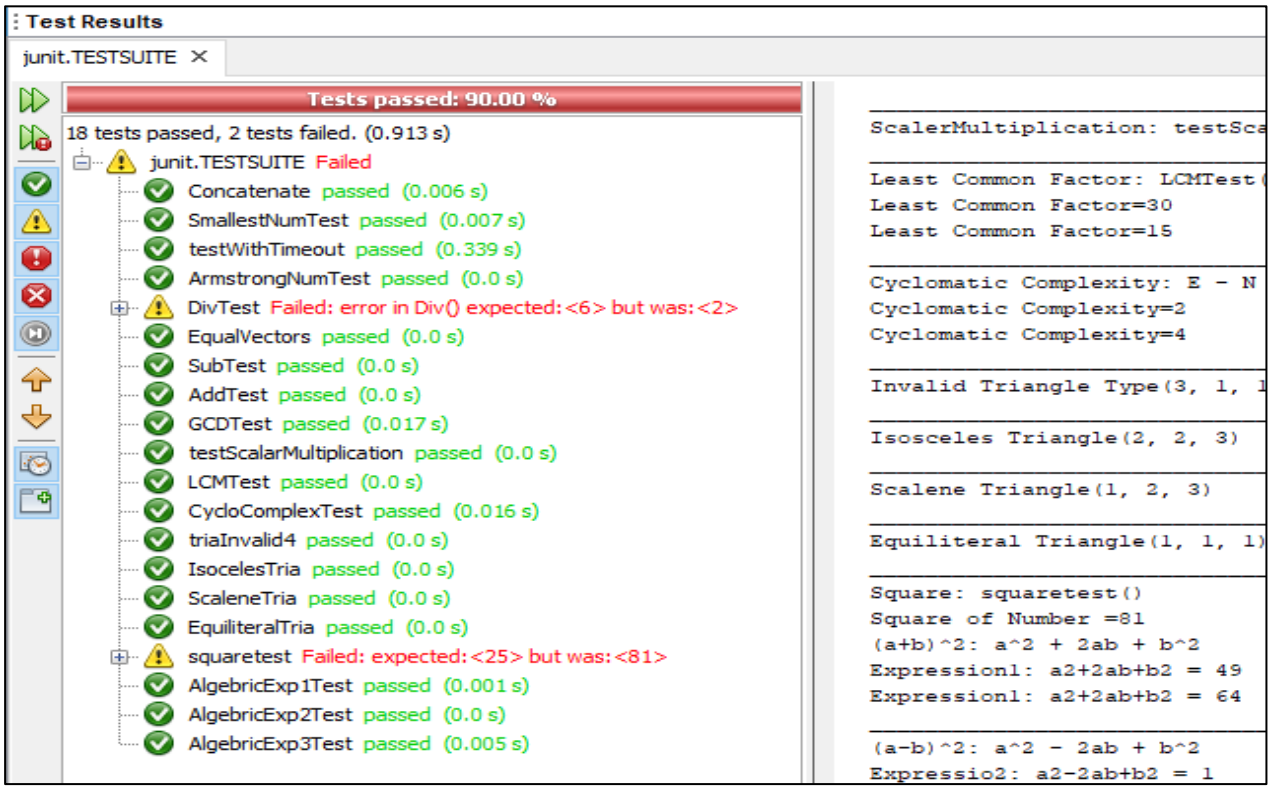
```
package junit;
import static junit.TriangleType.EQUILATERAL;
import static junit.TriangleType.INVALID;
import static junit.TriangleType.ISOSCELES;
import static junit.TriangleType.SCALENE;
import static junit.framework.TestCase.assertFalse;
import static org.junit.Assert.assertEquals;
import static junit.framework.TestCase.assertTrue;
import org.junit.Test;

public class TESTSUITE {
   @Test
   public void AddTest(){
      System.out.println("Addition: AddTest()");
      assertEquals("error in add()", 3, addition.add(1, 2));
      assertEquals("error in add()", -6, addition.add(-4, -2));
      System.out.println("_____");
}
   @Test
   public void CycloComplexTest(){
      System.out.println("Cyclomatic Complexity: E - N + 2*P ");
      assertEquals("error in CycloComplex()", 2, CyclomaticComplexity.cyclocomplex(10, 4, 2));
      assertEquals("error in CycloComplex()", 4, CyclomaticComplexity.cyclocomplex(12,2,3));
      System.out.println("_____");
   }
   @Test
   public void SmallestNumTest(){
      System.out.println("Smallest Number Test");
      assertEquals("error in SmallestNumber()",-2, SmallestNumber.smallestNum(10, 4, -2));
      assertEquals("error in SmallestNumber()", 3, SmallestNumber.smallestNum(19, 6, 3));
      System.out.println("_____");
   }
   @Test
   public void GCDTest(){
      System.out.println("Greatest Common Factor Test");
      assertEquals("error in GCD()",11, GCD_GreatestCommFactor.GCD(55, 121));
      assertEquals("error in GCD()", 1, GCD_GreatestCommFactor.GCD(7, 3));
      System.out.println("_____");
   }
   @Test
   public void AlgebricExp1Test(){
      System.out.println("(a+b)^2: a^2 + 2ab + b^2 ");
      assertEquals("error in Expression1()", 49, AlgebricExpressions.expression1(4, 3));
      assertEquals("error in Expression1()", 64, AlgebricExpressions.expression1(5, 3));
      System.out.println("_____");
   }
   @Test
   public void AlgebricExp2Test(){
      System.out.println("(a-b)^2: a^2 - 2ab + b^2 ");
      assertEquals("error in Expression2()", 1, AlgebricExpressions.expression2(4, 3));
      assertEquals("error in Expression2()", 4, AlgebricExpressions.expression2(5, 3));
      System.out.println("_____");
   }
   @Test
   public void AlgebricExp3Test(){
      System.out.println("(a2-b2): (a+b)(a-b) ");
      assertEquals("error in Expression3()", 7, AlgebricExpressions.expression3(4, 3));
      assertEquals("error in Expression3()", 16, AlgebricExpressions.expression3(5, 3));
      System.out.println("_____");
   }
   @Test
   public void LCMTest(){
      System.out.println("Least Common Factor: LCMTest() ");
      assertEquals("error in LCM()", 30, LCM_LeastCommonFactor.LCM(6, 10));
```

```
    assertEquals("error in LCM()", 15, LCM_LeastCommonFactor.LCM(5, 3));
    System.out.println("_____");
  }
  @Test
  public void SubTest(){
    System.out.println("Subtraction: SubTest()");
    assertEquals("error in Sub()", 2, subtraction.sub(3, 1));
    assertEquals("error in Sub()", 4, subtraction.sub(6, 2));
    System.out.println("_____");
  }
  @Test
  public void DivTest(){
    System.out.println("Division: DivTest()");
    assertEquals("error in Div()", 2, Division.divide(6, 3));
    assertEquals("error in Div()", 3, Division.divide(12, 4));
    System.out.println("_____");
  }
  @Test
  public void squaretest()
  {
    System.out.println("Square: squaretest()");
    Square test = new Square();
    int output = test.square(5);
    assertEquals(25,output);
    System.out.println("_____");
  }
  @Test
  public void ArmstrongNumTest()
  {
    System.out.println("Armstrong Numbers: Test()");
    assertEquals("error in ArmstrongTest()", 371, ArmstrongNumber.Armstrong(371));
    assertEquals("error in ArmstrongTest()", 153, ArmstrongNumber.Armstrong(153));
    System.out.println("_____");
  }
  @Test
  public void testScalarMultiplication() {
    System.out.println("ScalerMultiplication: testScalarMultiplication()");
    assertEquals(-39, ScalerMultiplication.scalarMultiplication(new int[]{-3, 4}, new int[]{5, -6}));
    assertEquals(100, ScalerMultiplication.scalarMultiplication(new int[]{6, 8}, new int[]{6, 8}));
    System.out.println("_____");
  }
  @Test
  public void EqualVectors() {
    System.out.println("EqualVectors: testEqual()");
    assertFalse(VectorsEquality.equal(new int[]{0, 0, 0}, new int[]{0, 0, 1}));
    assertFalse(VectorsEquality.equal(new int[]{0, 0, 1}, new int[]{0, 0, 3}));
    System.out.println("_____");
  }
  @Test
  public void Concatenate() {
    System.out.println("");
    System.out.println("                (TESTS SUMMARY)                ");
    System.out.println("_____");
    System.out.println("");
    System.out.println("Concatenation: testHelloWorld()");
    assertEquals("Hello, world!", Concatenation.concatWords("Hello", ", ", "world", "!"));
    System.out.println("_____");
  }
  @Test(timeout = 0)
  public void testWithTimeout() {
    System.out.println("Factorial: testWithTimeout()");
    final int factorialOf = 1 + (int) (3000 * Math.random());
    System.out.println("computing " + factorialOf + '!');
    System.out.println(factorialOf + "! = " + Factorial.computeFactorial(factorialOf));
    System.out.println("_____");
  }
 @Test
 public void ScaleneTria() {
   System.out.println("Scalene Triangle(1, 2, 3)");
   final TriangleType type = Triangle.classify(1, 2, 3);
   assertEquals(SCALENE, type);
```

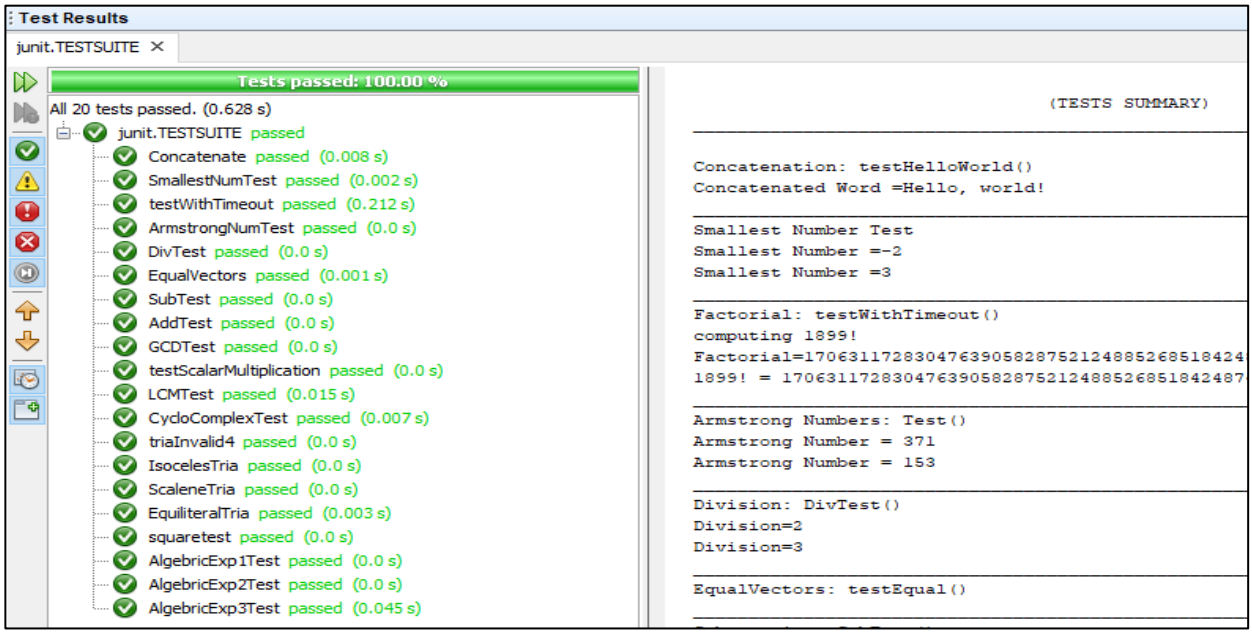
```
    System.out.println("_____");
  }
  @Test
  public void EquiliteralTria() {
    System.out.println("Equiliteral Triangle(1, 1, 1)");
    final TriangleType type = Triangle.classify(1, 1, 1);
    assertEquals(EQUILATERAL, type);
    System.out.println("_____");
  }
  @Test
  public void IsocelesTria() {
    System.out.println("Isosceles Triangle(2, 2, 3)");
    final TriangleType type = Triangle.classify(2, 2, 3);
    assertEquals(ISOSCELES, type);
    System.out.println("_____");
  }
  @Test
  public void triaInvalid4() {
    System.out.println("Invalid Triangle Type(3, 1, 1)");
    final TriangleType type = Triangle.classify(3, 1, 1);
    assertEquals(INVALID, type);
    System.out.println("_____");
  }
}
```

OUTPUT OF TEST SUITE

90% Test passed



100% Test passed



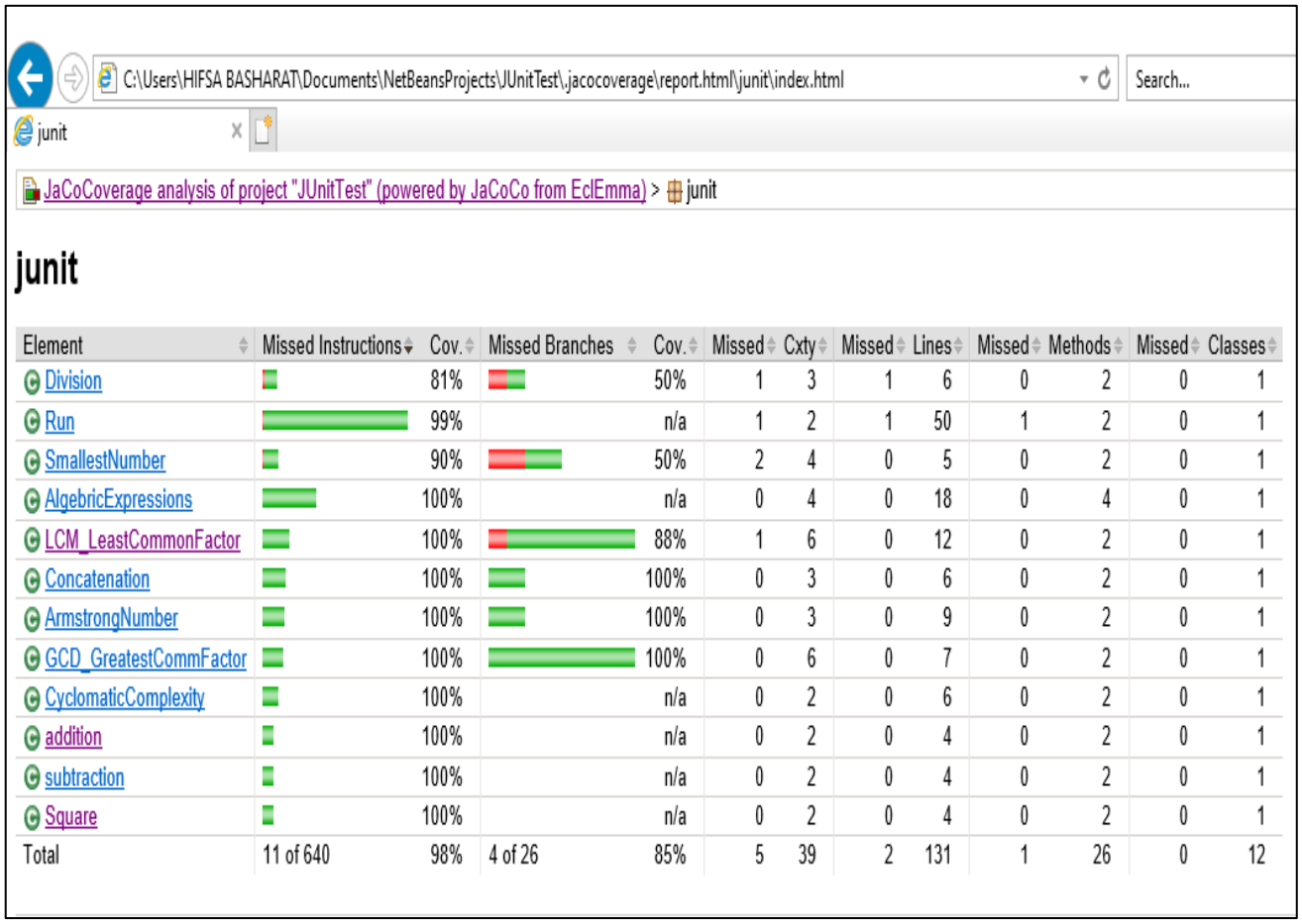
TIKI ONE JACOCOVERAGE PLUGIN

The JaCoCoverage Plugin is a NetBeans plugin that enhances the existing NetBeans functionality with new code coverage features. The plugin works as a transparent additional service that colors all java files according to the unit tests coverage information. With code coverage enabled user continues to work with his/her project in the usual way but can easily view the test coverage of the project classes.

The code coverage plugin will update the code coverage data and refresh editors markup every time a unit test (or any selected Ant target) is executed for the project. Currently the Java Application, Java Library, Java Web and Java EE (not tested with EAR and EJB projects but may work), Java Project with Existing Sources are supported. Maven support with JaCoCo is already integrated in NetBeans base installation, please check online how-to for details.

Coverage collections are based on JaCoCo in order to support Java 5, 6, 7 and Java 8 bytecode. Take it as a modern alternative to the EMMA and Cobertura based plugins. JaCoCo is a free code coverage library for Java, which has been created by the EclEmma team.

CODE COVERAGE ANALYSIS REPORT



C:\Users\HIFSA BASHARAT\Documents\NetBeansProjects\JUnitTest\.jacocoverage\report.html\junit\index.html

Search...

junit

JaCoCoCoverage analysis of project "JUnitTest" (powered by JaCoCo from EclEmma) > junit

junit

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Division		81%		50%	1	3	1	6	0	2	0	1
Run		99%		n/a	1	2	1	50	1	2	0	1
SmallestNumber		90%		50%	2	4	0	5	0	2	0	1
AlgebricExpressions		100%		n/a	0	4	0	18	0	4	0	1
LCM_LeastCommonFactor		100%		88%	1	6	0	12	0	2	0	1
Concatenation		100%		100%	0	3	0	6	0	2	0	1
ArmstrongNumber		100%		100%	0	3	0	9	0	2	0	1
GCD_GreatestCommFactor		100%		100%	0	6	0	7	0	2	0	1
CyclomaticComplexity		100%		n/a	0	2	0	6	0	2	0	1
addition		100%		n/a	0	2	0	4	0	2	0	1
subtraction		100%		n/a	0	2	0	4	0	2	0	1
Square		100%		n/a	0	2	0	4	0	2	0	1
Total	11 of 640	98%	4 of 26	85%	5	39	2	131	1	26	0	12

REFERENCES

- https://www.google.com/search?ei=Tp3mXZfqGpKHhbIPzriN4Ak&q=junit+JAVA&oq=junit+JAVA&gs_l=psy-ab.3..0l10.2575.3651..3828...0.2..0.454.1174.0j2j0j1j1.....0....1..gws
- https://www.vogella.com/tutorials/JUnit/article.html
- https://www.eclemma.org/jacoco/
- https://www.google.com/search?q=software+testing&oq=SOFTWARE+TESTING&aqs=chrome.0.0j69i60j69i61j69i65j69i60j0.6261j0j9&sourceid=chrome&ie=UTF-8