Batuyong, Genesis L.
Villanueva, Reymond Y.

Activity 4: Testing

Test Classes that was used:

```
 9
10  public class PolynomialTest {
11
12      @Test
13      public void testInitialStateZeroPolynomial() {
14          List <Term> terms = new ArrayList <> (3);
15          Term t1 = new Term (0, 0);
16          terms.add(t1);
17          Polynomial p1 = new Polynomial ('x', terms);
18
19          assertEquals("",p1.toString());
20      }
21
22      @Test
23      public void testMonomialAddition() {
24          List <Term> terms = new ArrayList <> (3);
25          Term t1 = new Term (10, 3);
26          terms.add (t1);
27          Term t2 = new Term (5, 2);
28          terms.add (t2);
29          Term t3 = new Term (1, 1);
30          terms.add (t3);
31
32
33          List <Term> terms2 = new ArrayList <> (3);
34          Term tt1 = new Term (4, 3);
35          terms2.add (tt1);
36          Term tt2 = new Term (1, 2);
37          terms2.add (tt2);
```

Figure 1: *PolynomialTest*

```
 3
 4  public class RelOpsTest extends TestCase {
 5
 6      public void testGreaterThan() {
 7          Assert.assertTrue(30.40>10.30);
 8          Assert.assertFalse(13>20);
 9      }
10
11      public void testGreaterThanOrEqualTo() {
12          Assert.assertTrue(30>=30);
13          Assert.assertFalse(12>=78);
14      }
15
16      public void testLesserThan() {
17          Assert.assertTrue(20<30);
18          Assert.assertFalse(30<12);
19      }
20
21      public void testLesserThanOrEqualTo() {
22          Assert.assertTrue(40<=40);
23          Assert.assertFalse(50<30);
24      }
25
26      public void testEqualTo() {
27          Assert.assertTrue(30==30);
28          Assert.assertFalse(40==30);
29      }
30
31      public void testNotEqualTo() {
```

Figure 2: *RelOpsTest*

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

public class RelQuantifiersTest {

    @Test
    public void atLeast() {
        assertTrue(30>=30);
        assertFalse(12>=78);
    }

    @Test
    public void atMost() {
        assertTrue(20<30);
        assertFalse(30<12);
    }

    @Test
    public void notLessThan() {
        assertTrue(30.40>10.30);
        assertFalse(13>20);
    }

    @Test
    public void notMoreThan() {
        assertTrue(20<30);
        assertFalse(30<12);
    }

    @Test
```

Figure 3: *RelQuantifierstest*

Figures 1 to 3 are screenshots of the source code test that was used in the Junit 4 testing (unit testing) and JaCoCo (Test coverage). The polynomialtest.java contains eight methods, and one class tested, while both RelOps.java and RelQuantifiers.java has ten methods and 1 class that was tested. We utilized assert methods in testing every method of every class. We only used these java files because the other java files from our previous activity contain the main method.

```xml
<path id="classpath">
    <path location="lib">
        <fileset dir="lib" includes="*.jar"/>
    </path>
    <path location="${build.dir}/classes"></path>
</path>
```

Figure 4. *Declaration for classpath needed for classpathref in JUnit and Coverage*

```xml
<!-- This task will compile all the needed java source files -->
<target name="compile" description="compile java files">
        <mkdir dir="${build.dir}/classes"/>
        <javac sourcepath="" srcdir="${src.dir}"
        destdir="${build.dir}/classes"
        classpathref="classpath"
        includeantruntime="false"
        fork="true">
    </javac>
</target>
```

Figure 5. *Compile task for all java source files*

```
compile:
    [mkdir] Created dir: C:\Users\Reymond\Desktop\IT 311 (MIDTERMS)\Apache ANT\build\
classes
    [javac] Compiling 6 source files to C:\Users\Reymond\Desktop\IT 311 (MIDTERMS)\Ap
ache ANT\build\classes
```

Figure 6. *Build execution of target "compile"*

Figure 5 shows the source code in compiling java files and creates a directory for every class file compiled. The compiled classes are located in the build folder inside the classes folder. Figure 6 shows the execution of the compile task; in total, there were 6 java files that was compiled from the src directory.

```
<!-- This taskdef will import the JaCoCo ANT library or the JaCoCo ANT task -->
<taskdef uri="antlib:org.jacoco.ant" resource="org/jacoco/ant/antlib.xml">
    <classpath path="lib/jacocoant.jar"/>
</taskdef>
```

Figure 7: *Import the JaCoCo ANT task library*

Figure 7 shows the a taskdef tag to import the JaCoCo ant library that will be used in the text coverage located in lib directory namely jacocoant.jar.

```
<!-- This task will will perform the junit with JaCoCo coverage task -->
<target name="testing" depends="compile">
    <mkdir dir="${report.dir}/junit-reports"/>
    <mkdir dir="${result.report.dir}"/>
    <jacoco:coverage destfile="${result.exec.file}" xmlns:jacoco="antlib:org.jacoco.ant"
        exclclassloader="sun.reflect.DelegatingClassLoader:javassist.Loader">
        <!-- Perform the junit inside the coverage test -->
    <junit printsummary="yes" haltonfailure="no" fork="true">
        <classpath refid="classpath"/>
        <formatter type="plain" />
        <batchtest fork="yes" todir="${report.dir}/junit-reports">
            <fileset dir="${test.dir}">
            <include name="**/*Test*.java" />
            </fileset>
        </batchtest>
    </junit>
    </jacoco:coverage>
</target>
```

Figure 8: *Coverage test within the JUnit testing*

Figure 8 shows the source code for the test coverage using JaCoCo and the Junit testing; since Junit should be executed first before the test coverage can give results. Thus, we combined our Unit Testing and Test coverage in a single target tag named testing.

```
testing:
    [mkdir] Created dir: C:\Users\itsme\Desktop\Apache ANT\build\reports\junit-reports
    [mkdir] Created dir: C:\Users\itsme\Desktop\Apache ANT\build\reports\coverage-reports
[jacoco:coverage] Enhancing junit with coverage
    [junit] Running PolynomialTest
    [junit] Tests run: 7, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.126 sec
    [junit] Running RelOpsTest
    [junit] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.032 sec
    [junit] Running RelQuantifiersTest
    [junit] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.052 sec

report:
[jacoco:report] Loading execution data file C:\Users\itsme\Desktop\Apache ANT\build\reports\coverage-reports\jacoco.exec

[jacoco:report] Writing bundle 'JUnit Coverage Report' with 3 classes
[jacoco:report] To enable source code annotation class files for bundle 'JUnit Coverage Report' have to be compiled with
 debug information.
```

Figure 9: *Build execution of target "testing"*

Figure 9 displays the execution of the target test from the command line. Firstly, two directories were made inside the build folder, namely, JUnit-reports for the generated report text files of the unit testing and coverage-reports that contains the generated reports from the test coverage using JaCoCo. Then the unit testing of the classes polynomialTest with seven tests run, RelOpsTest with nine tests run, and RelQuantifiersTest with nine tests runs.

```
<!-- This task will generate coverage report in different formats
html, csv, xml -->
<target name="report" depends="testing">
 <!-- Create coverage report  -->
 <jacoco:report>
    <!--  This task needs the collected execution data ..  -->
    <executiondata>
        <file file="${result.exec.file}"/>
    </executiondata>
    <!--  This task needed the class files and optional source files
    that will be needed to generate report...  -->
    <structure name="JUnit Coverage Report">
        <classfiles>
        <fileset dir="${build.dir}/classes">
            <include name="**/*Test*"/>
        </fileset>
        </classfiles>
        <sourcefiles encoding="UTF-8">
        <fileset dir="${test.dir}"/>
        </sourcefiles>
    </structure>
    <!--  to produce reports in different formats.  -->
    <html destdir="${result.report.dir}"/>
    <csv destfile="${result.report.dir}/report.csv"/>
    <xml destfile="${result.report.dir}/report.xml"/>
 </jacoco:report>
 </target>
```

Figure 10: *Coverage test(JaCoCo) generate reports*

```
report:
[jacoco:report] Loading execution data file C:\Users\Reymond\Desktop\IT 311 (MIDTERMS)\
Apache ANT\build\reports\coverage-reports\jacoco.exec
[jacoco:report] Writing bundle 'JUnit Coverage Report' with 6 classes
[jacoco:report] To enable source code annotation class files for bundle 'JUnit Coverage
 Report' have to be compiled with debug information.

test:

BUILD SUCCESSFUL
Total time: 2 seconds
```

Figure 11: *Build execution of target "report"*

Figure 10 shows the source code for the test coverage using the Jacoco, with generated reports with different formats such as HTML, CSV, and XML files. Figure 11 executes the target tag report; first, it loads the collected execution data and then generates test coverage reports.

TEST-RelOpsTest.txt | build.xml

```
1  Testsuite: RelOpsTest
2  Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.017 sec
3
4  Testcase: testLesserThan took 0.004 sec
5  Testcase: testEqualTo took 0 sec
6  Testcase: testGreaterThanOrEqualTo took 0 sec
7  Testcase: testAnd took 0.002 sec
8  Testcase: testNot took 0 sec
9  Testcase: testOr took 0 sec
10 Testcase: testGreaterThan took 0 sec
11 Testcase: testNotEqualTo took 0 sec
12 Testcase: testLesserThanOrEqualTo took 0 sec
13
```

Figure 12: *Output of RelOps Test in JUnit testing in plain text file*

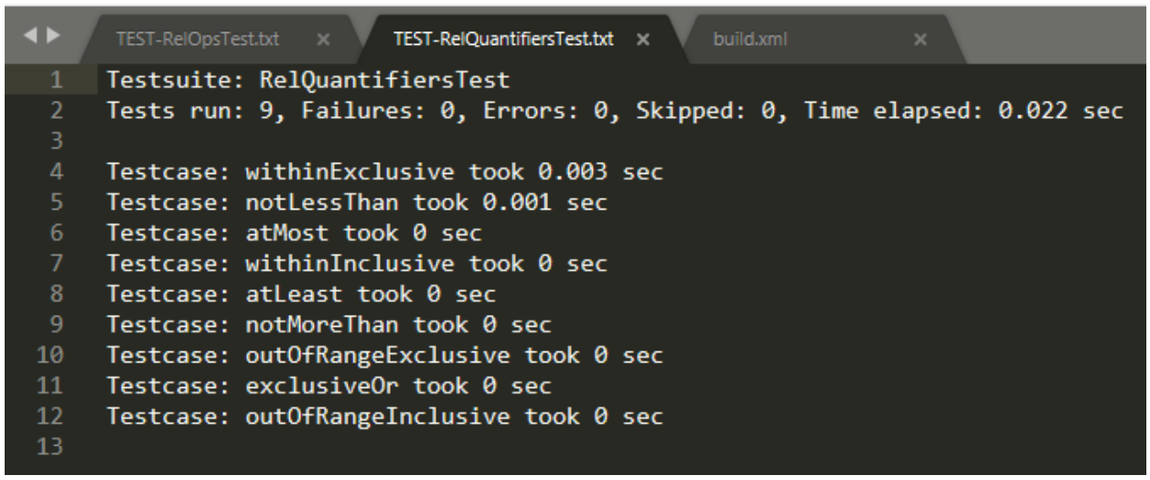

Figure 13: *Output of RelQuantifiers Test in JUnit testing in plain text file*

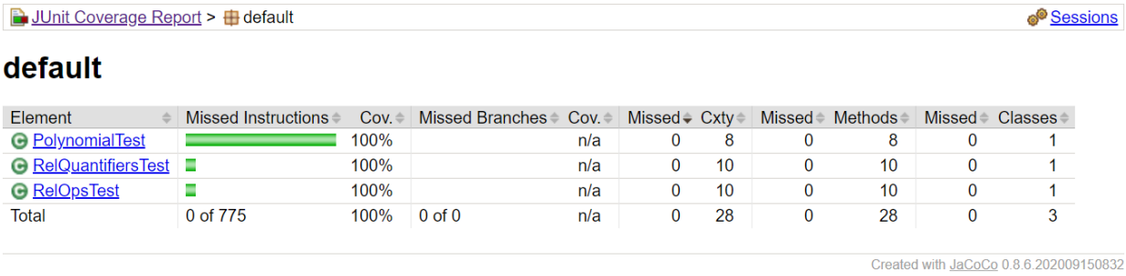

Figure 14: *Output of Java test source files in coverage testing with JaCoCo*

Figure 12 and 13 is the generated text file unit testing report for the RelOpsTest and RelQuantifiers class. They have both nine tests runs in total with 0 failures, errors, and skipped with time elapsed of 0.017 seconds for the RelOpsTest class and O.022 seconds for the RelQuantifierstest class. Figure 14 is the generated test coverage report created with JaCoCo, it shows the data in tabulated form, and the image above is the HTML file report.