TESTING

IT 311
9372

Submitted by:

GARCIA, Joshua
MOLINES, Emmanuel

BSIT 3

DOCUMENTATION

The team designed a test automation for J Unit testing and with the use of jacoco code coverage, considering Apache ANT.

```
C:\Users\emman\Desktop\AutoBots[act4]>ant
Buildfile: C:\Users\emman\Desktop\AutoBots[act4]\build.xml

setup:
    [input] Do you want to delete the build folder?[y/n]
y

continue:

clean:
```

Figure 1. Clean/delete old ‘build’ directory

Figure 1 shows the first process of our build.xml file, where it task is to ask the user to clean or delete existing ‘build’ folder.

```
create:
    [mkdir] Created dir: C:\Users\emman\Desktop\AutoBots[act4]\build
    [mkdir] Created dir: C:\Users\emman\Desktop\AutoBots[act4]\build\classes
    [mkdir] Created dir: C:\Users\emman\Desktop\AutoBots[act4]\build\Folder1
    [mkdir] Created dir: C:\Users\emman\Desktop\AutoBots[act4]\build\Folder2
    [mkdir] Created dir: C:\Users\emman\Desktop\AutoBots[act4]\build\Folder3
    [mkdir] Created dir: C:\Users\emman\Desktop\AutoBots[act4]\build\coverageReport\html\jacoco
```

Figure 2. Create directories

Figure 2 shows the second process of our build.xml, where it will now create a different directories which will be used later to store different build files,folders, and code coverage report to the project.

```
compile:
    [javac] Compiling 3 source files to C:\Users\emman\Desktop\AutoBots[act4]\build\classes
```

Figure 3. Compile the project

Figure 3 shows the process where it will now compile the different source files and store it in the created directory called “classes”.

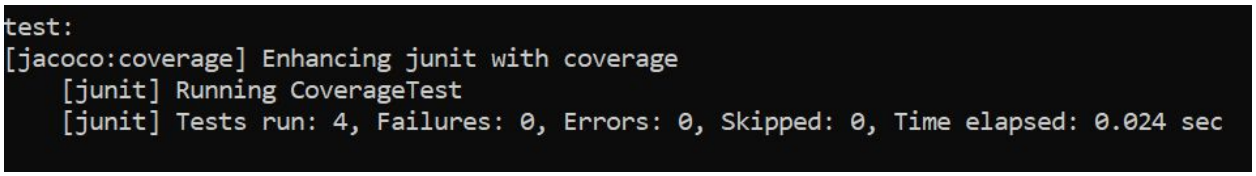

Figure 4. Execution of testing

Figure 4 shows the process where it will now execute the JUnit testing for every class. We can observe how many tests have been executed. Considering the use of junit coverage test where we implement it from every test file, we can observe test failures, errors, and skipped files that have been executed.

Figure 5. Generate jacoco coverage reports

Figure 5 shows the process where it will generate code coverage reports for all executed test files. These reports will be stored and located at the “build\coverageReport” path. These reports will be shown in the section REPORTS. (See figure 9 below)

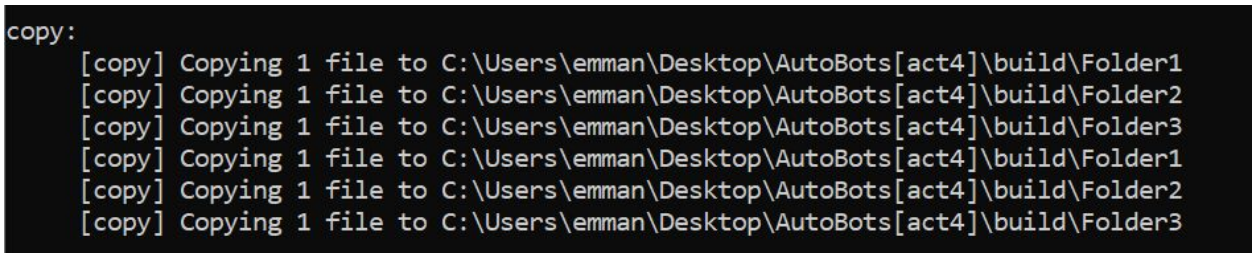

Figure 6.copy source files to different directories

Figure 6 shows that each compiled class in the project will be copied and stored in different directories on which these directories was created earlier in the previous build task.

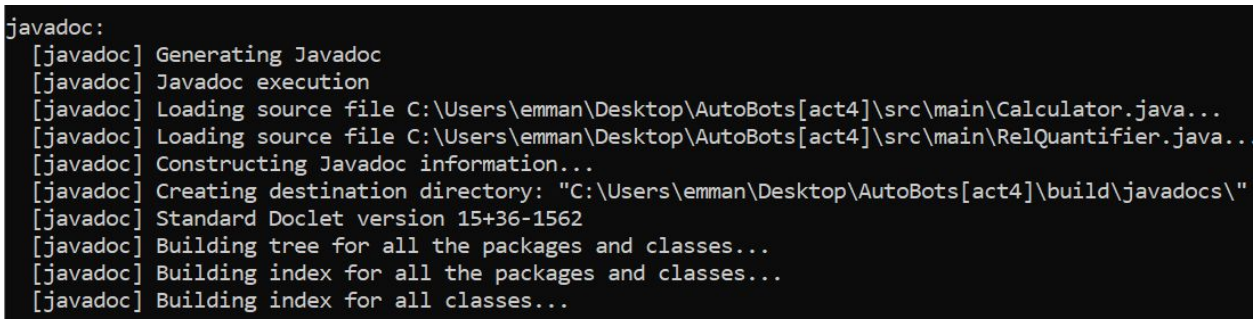

Figure 7. Create documentation of the class

Figure 7 shown above, after creating and copying files and directory in figure 6, a documentation for every java class is being created and being stored in the “build\javadocs” directory.

```
jar-classes:
     [jar] Building jar: C:\Users\Lenovo\Desktop\AutoBots[act4]\build\classes.jar

jar-docu:
     [jar] Building jar: C:\Users\Lenovo\Desktop\AutoBots[act4]\build\documentaries.jar
```

Figure 8. Generate jar files

This will be the last process of the build ant task, its task is to create a jar file which allows to archive folders and files into java archive format. The compiled classes and javadoc are being converted into jar files and being stored in the build directory.

REPORTS

We decided to use jacoco plugin to generate our code coverage report for our java files. In this section you can see different reports of the executed test files. We've also considered breakdowns of these reports to show how they've individually tested.

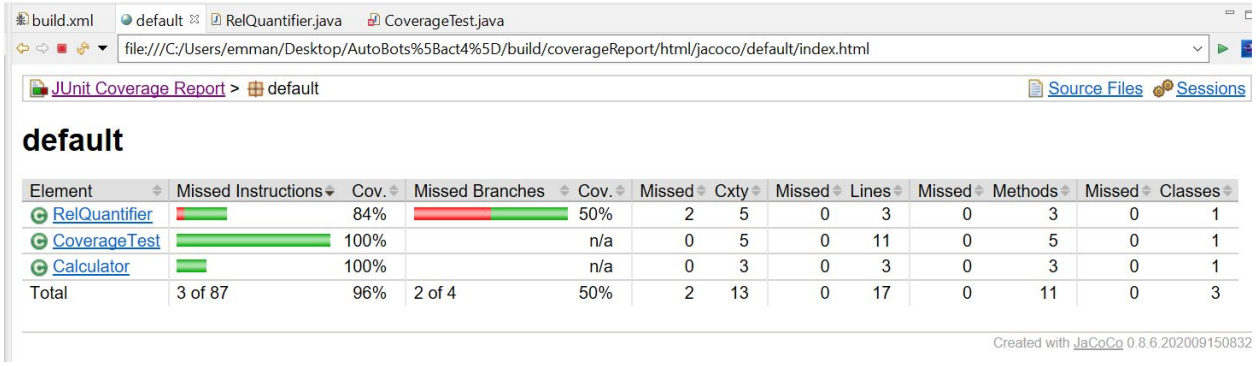

Figure 9. Summary report

Figure 9 shows the summary code coverage report of different source files that has been tested. In the report, a total number of 3 classes has been tested which show how many classes and methods missed the code coverage of jacoco.

CODE COVERAGE REPORTS

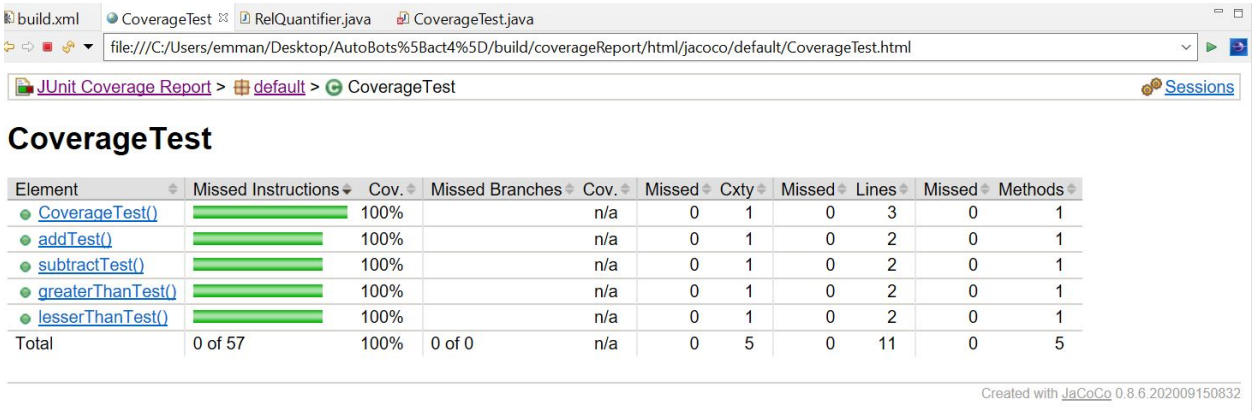

Figure 9.1. CoverageTest Class report

For the code coverage called “CoverageTest” it shows that all methods used to test different methods for the Calculator and Relquantifiers java class have all been successfully used or executed. Successfully executed means that there are no missed methods, lines,branches, or instructions.

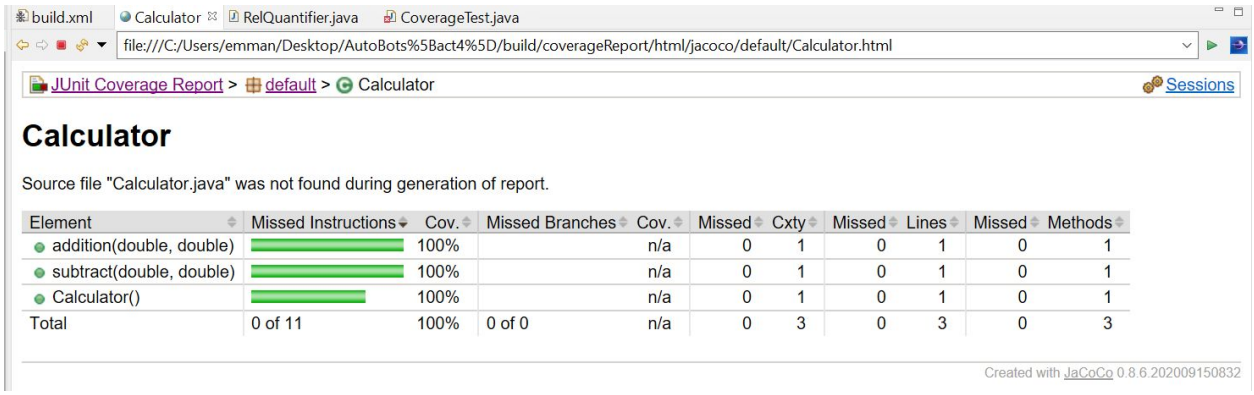

Figure 9.2. Test calculator report

For the specific class code coverage called “Calculator” it also shows that all methods that were used by this class have all been successfully used or executed. Successfully executed means that there are no missed methods, lines,branches, or instructions.

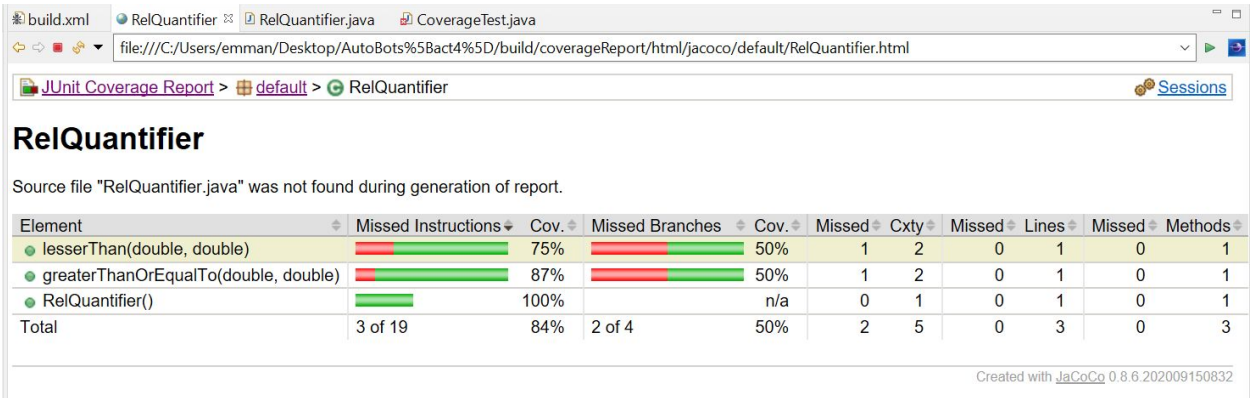

Figure 9.3. Test RelQuantifier report

For the specific class code coverage called “Relquantifier”, it shows that there are some instructions and branches on which the code coverage missed. This is an example of the use of code coverage in code programs, it shows that even if the program is successfully running there are still some codes in the program which are unused. With the help of the code coverage the developers will be able to know that there are unused instructions and branches that need to be removed in the Relquantifiers java class.