Batuyong, Genesis L.
Villanueva, Reymond Y.

Activity 3: Build Automation

Utility Classes used:

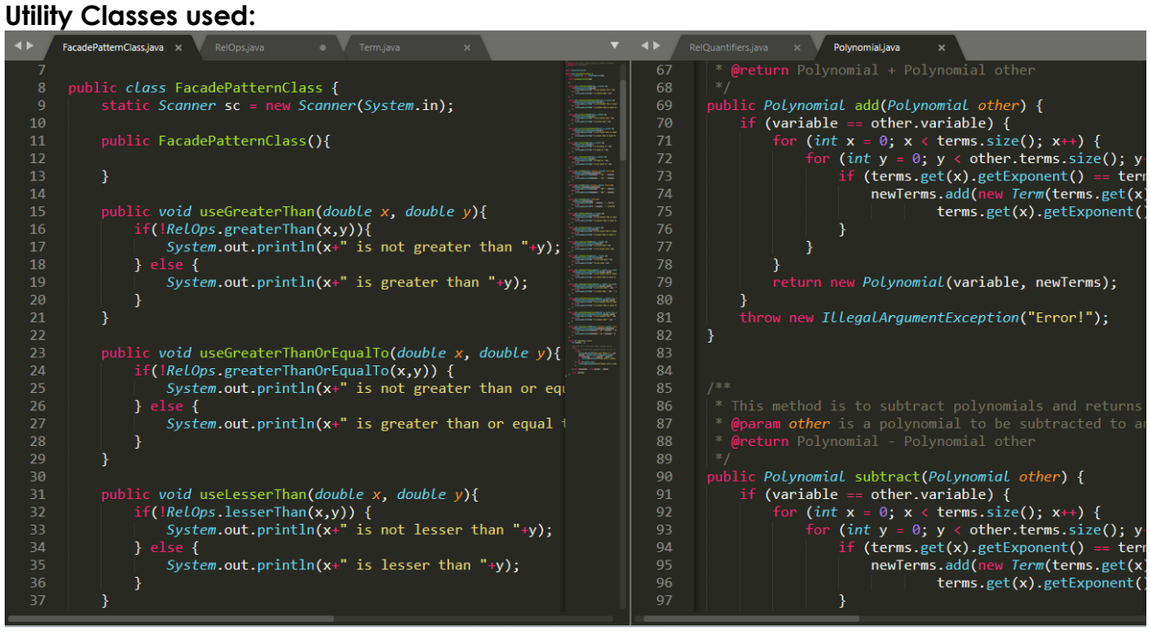

***Figure 1**: Collection of Utility classes.*

Most of the class collection is composed of past projects such as FacadePatternClass.java, RelOps.java, and Term.java. RelQuantifiers.java and Polynomial.java. We choose these classes because they are mostly composed of iterative and conditional statements, as shown in the image above, contains a code fragment of the said utility classes. The Polynomial class primarily has methods with conditional statements and iterative structures such as the subtract(Polynomial other) with parameter Polynomial - other that serves as the polynomial that will be subtracted from the other polynomial. Another is the add(Polynomial other) method with the same parameter, like the subtract method that serves as the polynomial that will be added to the other polynomial.

Classes with the Main Method:

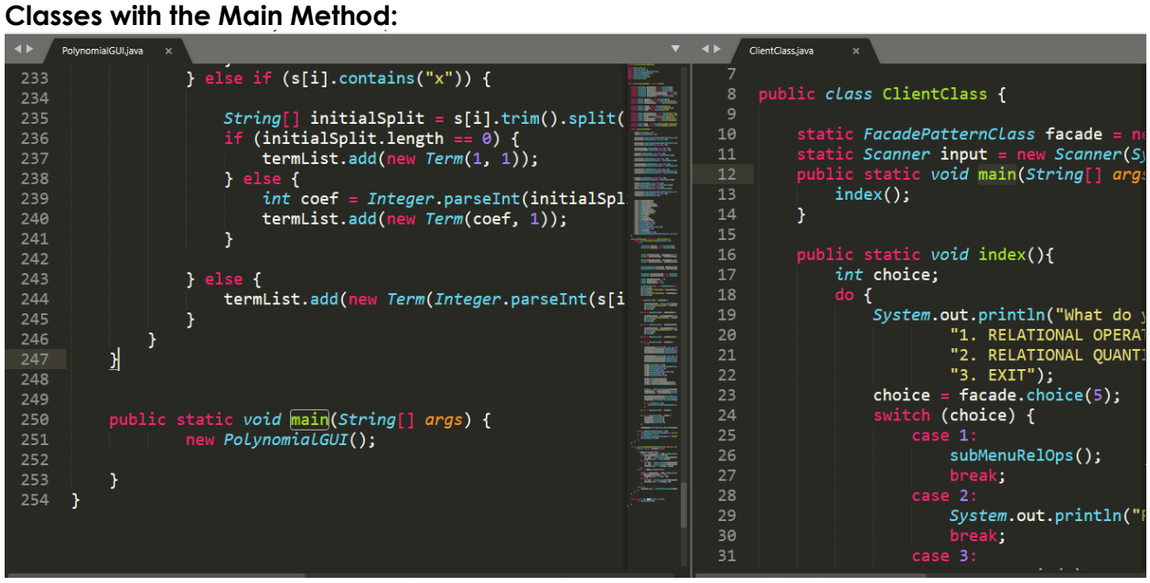

***Figure 2:** Classes with the main method.*

The classes with the main method that we used are also from the previous projects that we had, namely PolynomialGUI.java with a utility class polynomial.java referencing to it. Another class with the main method is ClientClass.java, with a utility class RelOps.java and RelQuantifiers.java.

1. Compile Class/es

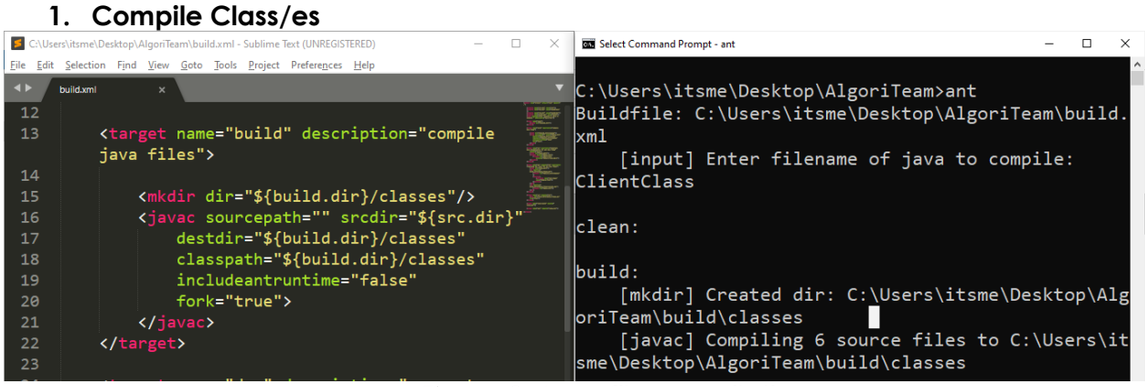

***Figure 3**: Java file compilation.*

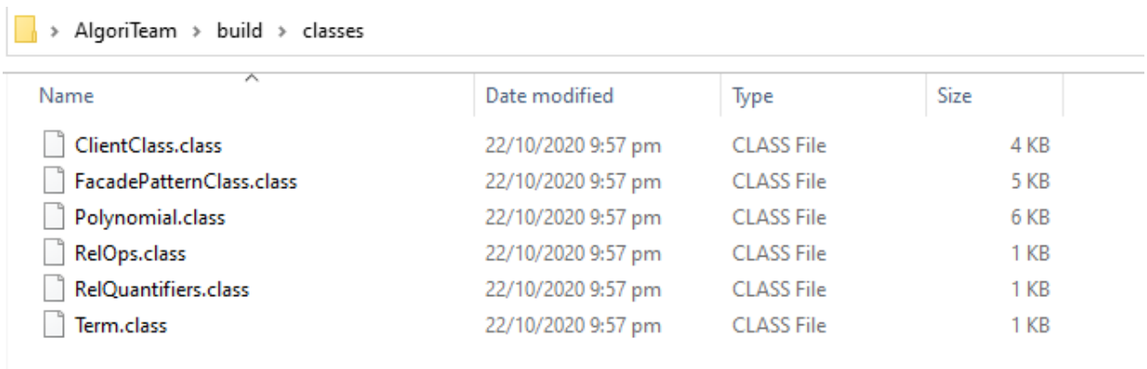

***Figure 4:** Result of target compile.*

As shown in the image above, it successfully compiled every java file in the src folder and then generated a build/classes directory for the class file. The user will be asked for the filename of the java file that will be compiled. The target clean will be executed but wouldn't delete the build directory since there is no existing build directory yet. The target build has a task tag mkdir for creating the build directory, and inside the classes folder wherein it contains the compiled classes. Another task tag is javac used to compile every file from the source path src folder; then, all the compiled java files will be in the build/classes directory. The fork is set to false since we used the running Apache Ant JVM and not an external JVM.

2. Create documentation of the class/es used in the project

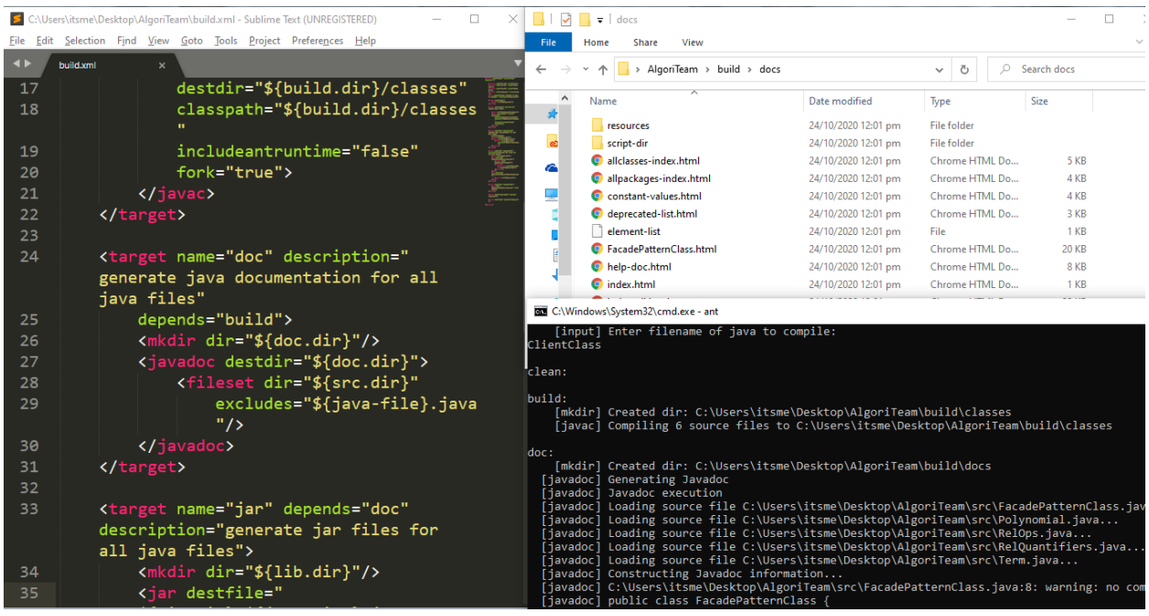

***Figure 5:** Documentation generated.*

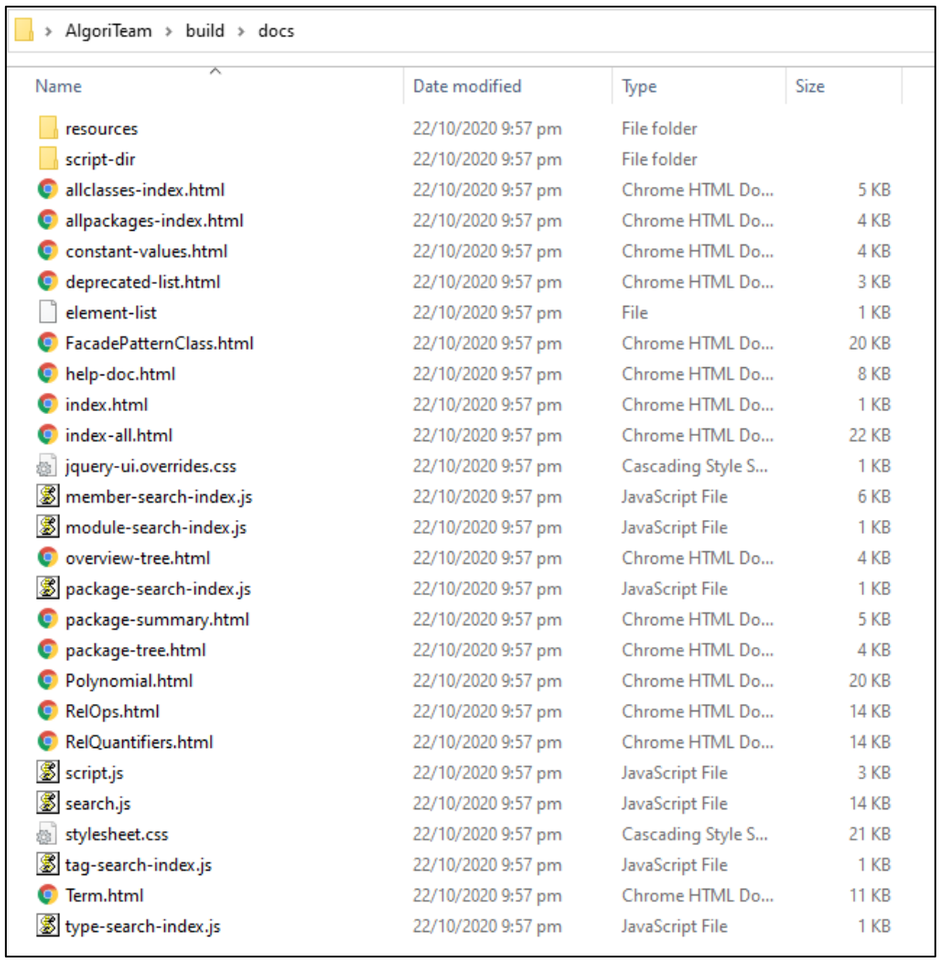

Figure 6: *Result of Target docs.*

The image above shows the code fragment for generating the documentation of java files in the src folder. And the execution of the code from command prompt then the generated documentation such as HTML files and other supporting files and folder located in the docs folder. The target doc has a two task tag, namely mkdir that will create a doc folder inside the build folder. Then javadoc used to generate documentation from the fileset, which is in the src folder, and then put on the destination folder build/docs.

3. Create Java archive files for the project

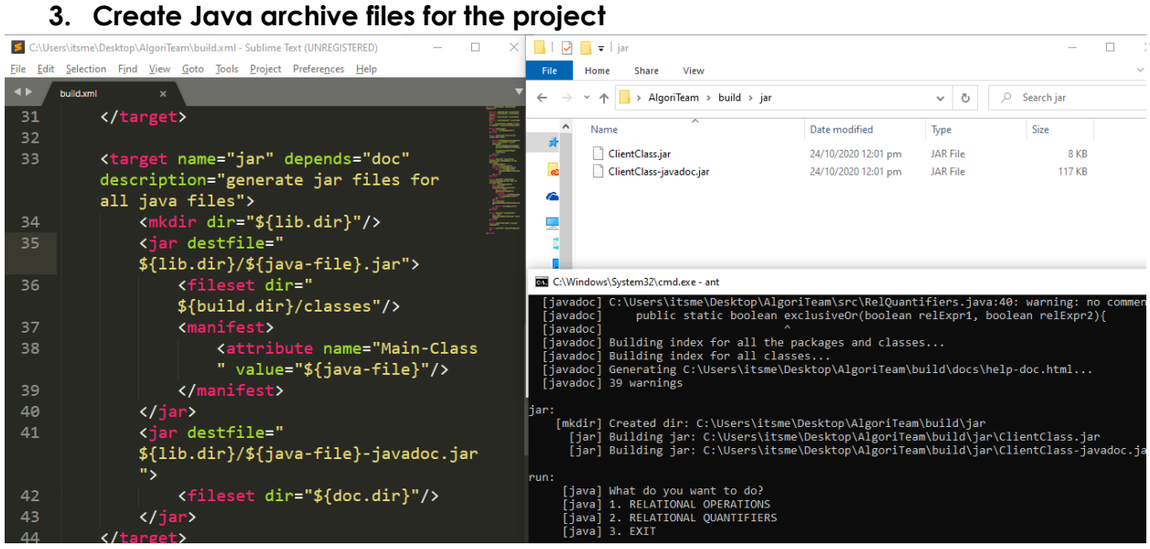

Figure 7: *Jar files generation*

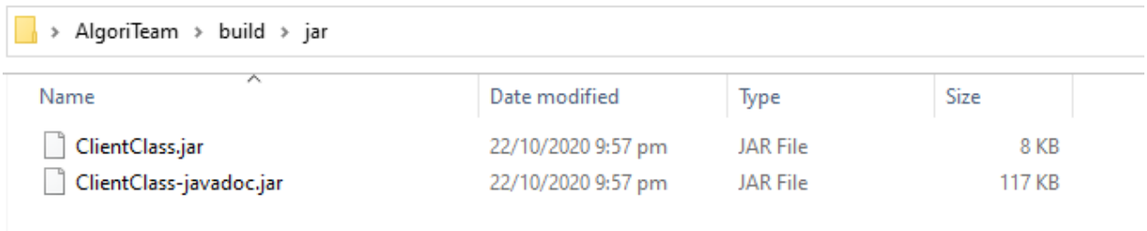

Figure 8: *Result of Target jar.*

The image above shows the execution of the code fragment that will generate a Java ARchive file. The asked user input java file will be the one that will be used to generate a Java archive file, which in this example is the ClientClass.java from the src folder. The mkdir task tag will create a folder lib inside the build folder where it is intended to contain the generated Java ARchive for the compiled ClientClass.class and the said class's documentation.

4. Run a java program

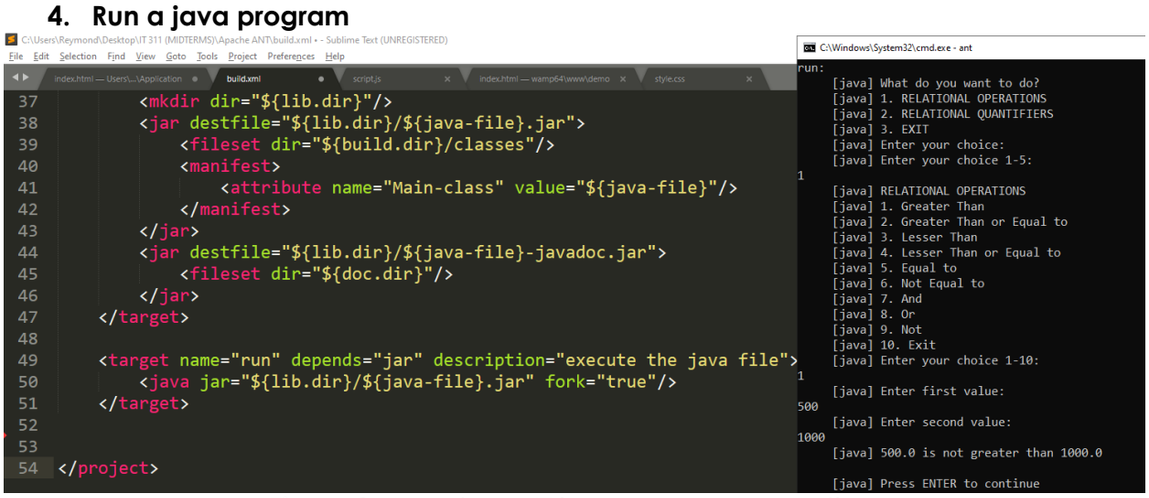

Figure 9: *Apache ANT java program execution.*

The figure above shows the execution of the java program in the Apache ANT. The ANT will ask the user at the start of the script for the name of the file to be compiled and it will create a jar file that will be utilized to run the java program inside the ANT. The script will execute the task "Java" and will utilize the attribute "jar" and indicate the directory of the created jar file, next attribute is the "fork" for it to enable the class execution into another JVM.