Project Deliverable 1: Maven & Unit Testing

Faculty of Engineering and Applied Science

SOFE 3650U: Software Design & Architecture | CRN: 73385 | Section: 001

Due: January 29th, 2023

Group 31

Rolf-Jaden Sibal (100845721)

Ontario Tech University

Oshawa, Ontario

rolfjaden.sibal@ontariotechu.net

Dmitri Rios Nadeau (100783206)

Ontario Tech University

Oshawa, Ontario

ericdmitri.riosnadeau@ontariotechu.net

Nathan Perez (100754066)

Ontario Tech University

Oshawa, Ontario

nathan.perez@ontariotechu.net

Noah Toma (100825559)

Ontario Tech University

Oshawa, Ontario

noah.toma@ontariotechu.net

Logan Butler (100828103)

Ontario Tech University

Oshawa, Ontario

logan.butler@ontariotechu.net

Resources

Github Repos:

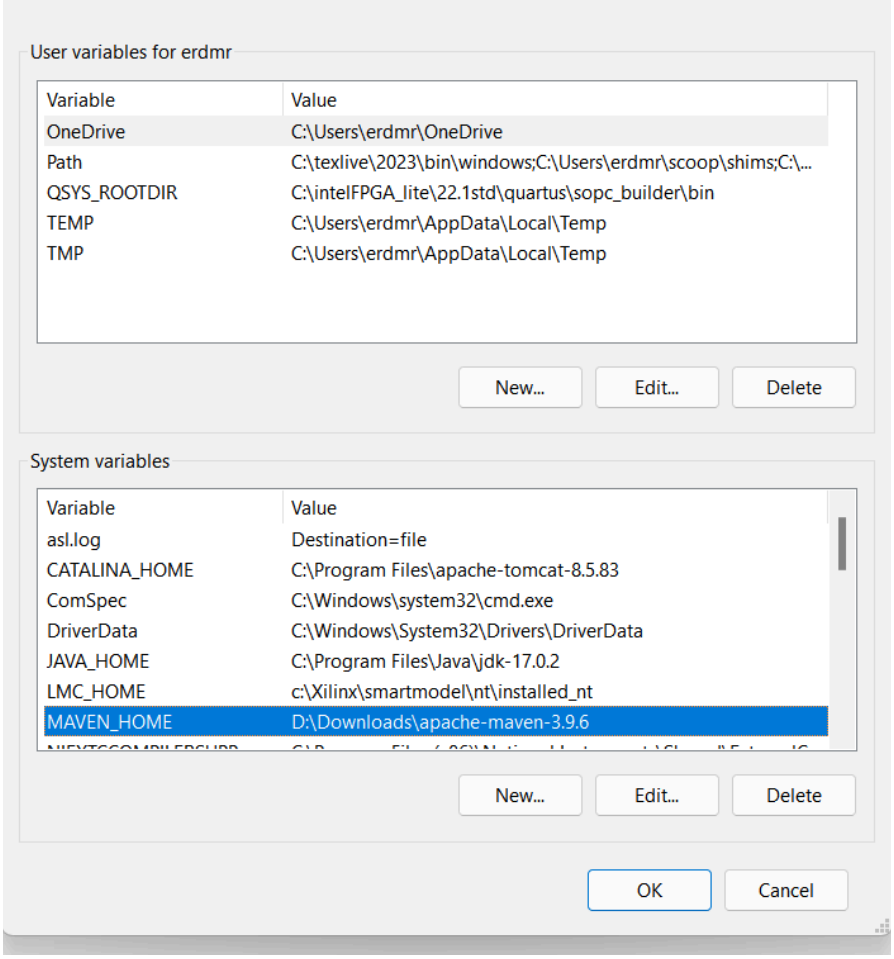
- Dmitri, Logan & Noah (In Respective Folders): https://github.com/D-aces/SQ_System_Design_Project
- Rolf: https://github.com/Neuron-Chan/SOFE3980U-Lab1/tree/master
- Nathan: https://github.com/NathanPerez18/SOFE3980-Project/tree/main

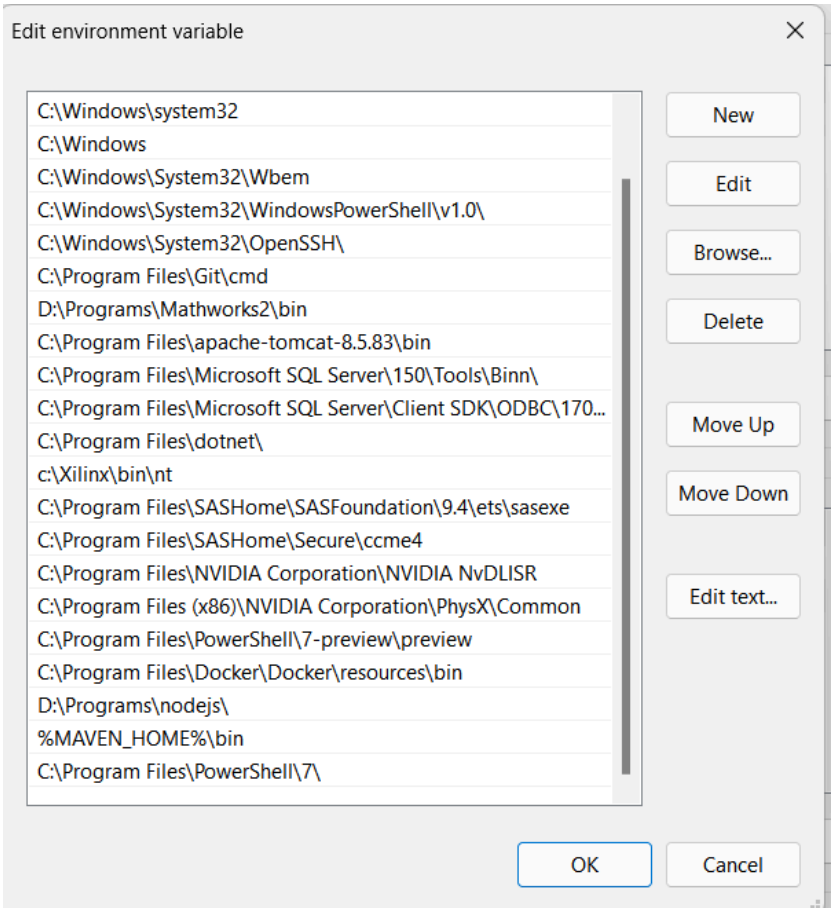
Video Links: Dmitri, Rolf, Nathan, Noah, Logan

Objectives

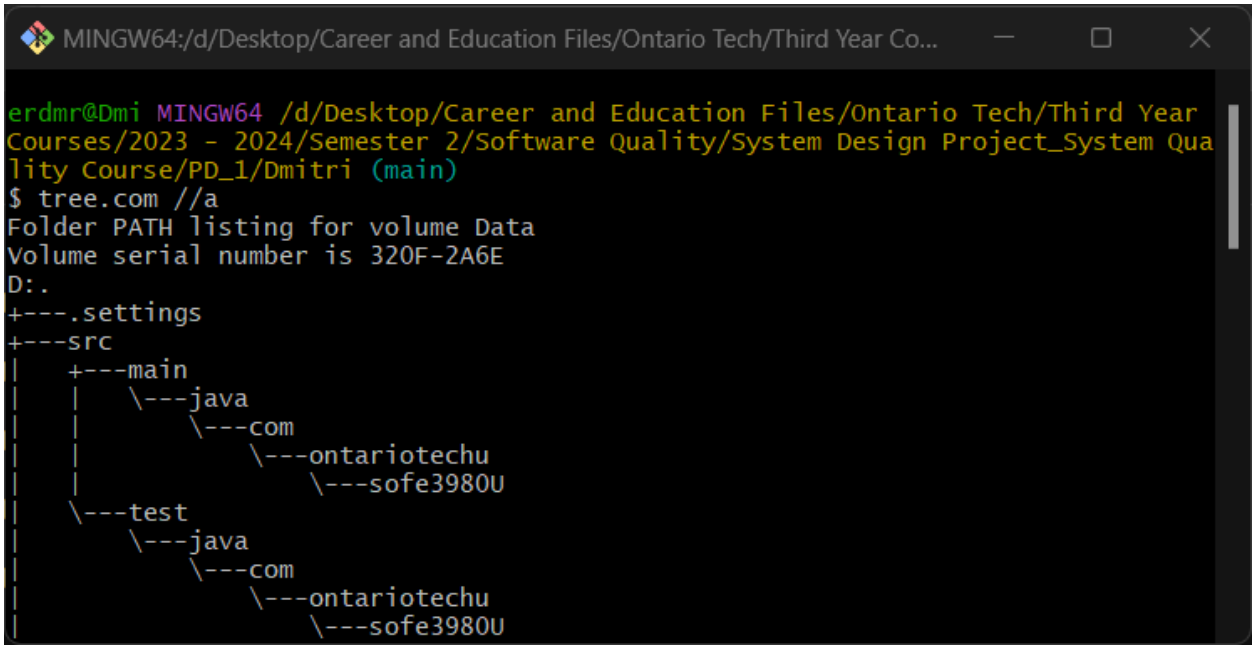
- Learn how to use Maven and add external dependencies to a Java application
- Learn how to use Maven to automate the building and unit testing process
- Learn how to add dependencies and build a jar file using Maven
- Learn how to run a project using Maven

Project Steps





1. Environment variables for Maven and Java are set.



2. Creation of a new project.

```java
public static Binary multiply(Binary num1, Binary num2) {
    int ind2 = num2.number.length() - 1;
    Binary num3 = new Binary("");

    for(int i = 0; i <= ind2; i++){
        if(num2.number.charAt(ind2 - i) == '1'){ // Check if the next char is '1'
            num3 = add(num3, num1); // Add num3 to num1 if true
        }
        num1 = new Binary(num1.number + "0"); // add a zero to the end of num1
    }
    return num3;
}
```

```java
public static Binary and(Binary num1, Binary num2) {
    int ind1 = num1.number.length() - 1;
    int ind2 = num2.number.length() - 1;
    String num3 = "";
    while (ind1 >= 0 || ind2 >= 0) // loop until both of the indices reach 0 (e.g. l
    {
        //Check each bit index to avoid out of bounds issues, if out of bounds by de
        char num1bit = ind1 >= 0 ? num1.number.charAt(ind1) : '0';
        char num2bit = ind2 >= 0 ? num2.number.charAt(ind2) : '0';
        //Check if either bits are zero
        if(num1bit == '0' || num2bit == '0'){
            num3 = '0' + num3; // if either are zero, add a 1 bit to num3
        }
        else{
            num3 = '1' + num3; // if neither are zero, add a 1 bit to num3
        }

        if(ind1>=0){
            ind1--; // decrease appendix for num1 by 1
        }

        if(ind2>=0){
            ind2--; // decrease appendix for num1 by 1
        }
    }
    Binary result = new Binary(num3); // create a new binary object
    return result; // returns the result of the bitwise operator  for the 2 numbers
}
```

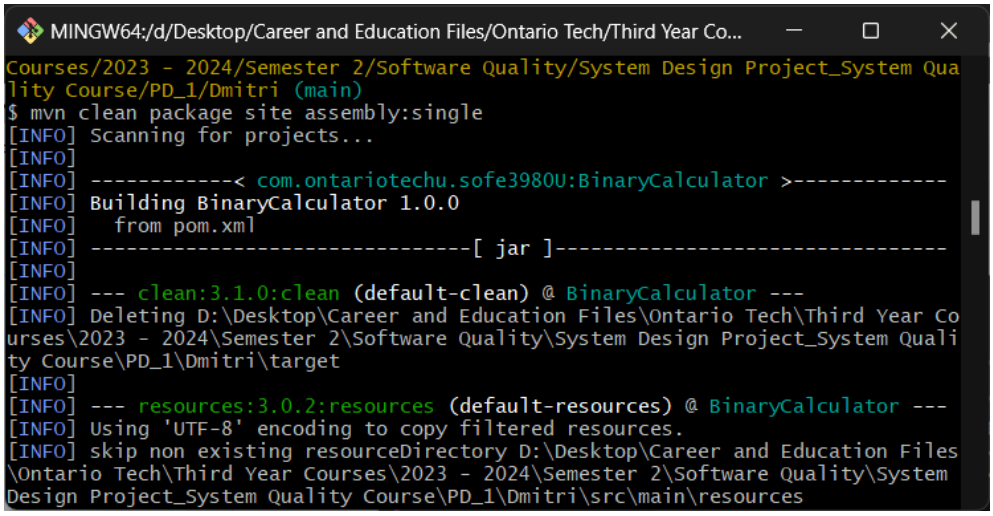
```java
public static Binary or(Binary num1, Binary num2) {
    int ind1 = num1.number.length() - 1;
    int ind2 = num2.number.length() - 1;
    String num3 = "";
    while (ind1 >= 0 || ind2 >= 0) // loop until both of the i
    {
        if (ind1>=0 && num1.number.charAt(ind1) == '1') { // I
            num3 = '1' + num3; // append 1 to the current numb

        }
        else if (ind2>=0 && num2.number.charAt(ind2) == '1') /
        {
            num3 = '1' + num3; // append 1 to the current numb
        }
        else
        {
            num3 = '0' + num3;
        }

        if(ind1>=0){
            ind1--; // decrease appendix for num1 by 1
        }

        if(ind2>=0){
            ind2--; // decrease appendix for num1 by 1
        }
    }
    Binary result = new Binary(num3); // create a new binary o
    return result; // returns the result of the bitwise operat
}
```

```java
@Test
public void or(){
    Binary binary1=new Binary(number:"0");
    Binary binary2=new Binary(number:"1");
    Binary binary3=Binary.or(binary1,binary2);
    assertTrue(binary3.getValue().equals(anObject:"1"));
}
```

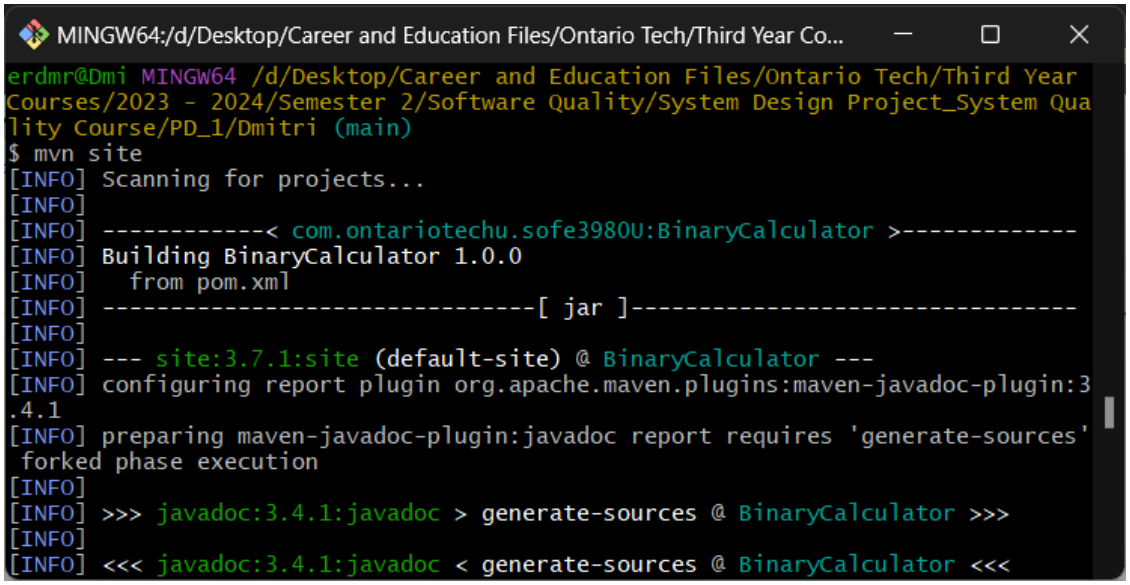
```java
@Test
public void and()
{
    Binary binary1=new Binary(number:"0");
    Binary binary2=new Binary(number:"1");
    Binary binary3=Binary.and(binary1,binary2);
    assertTrue( binary3.getValue().equals(anObject:"0"));
}
```

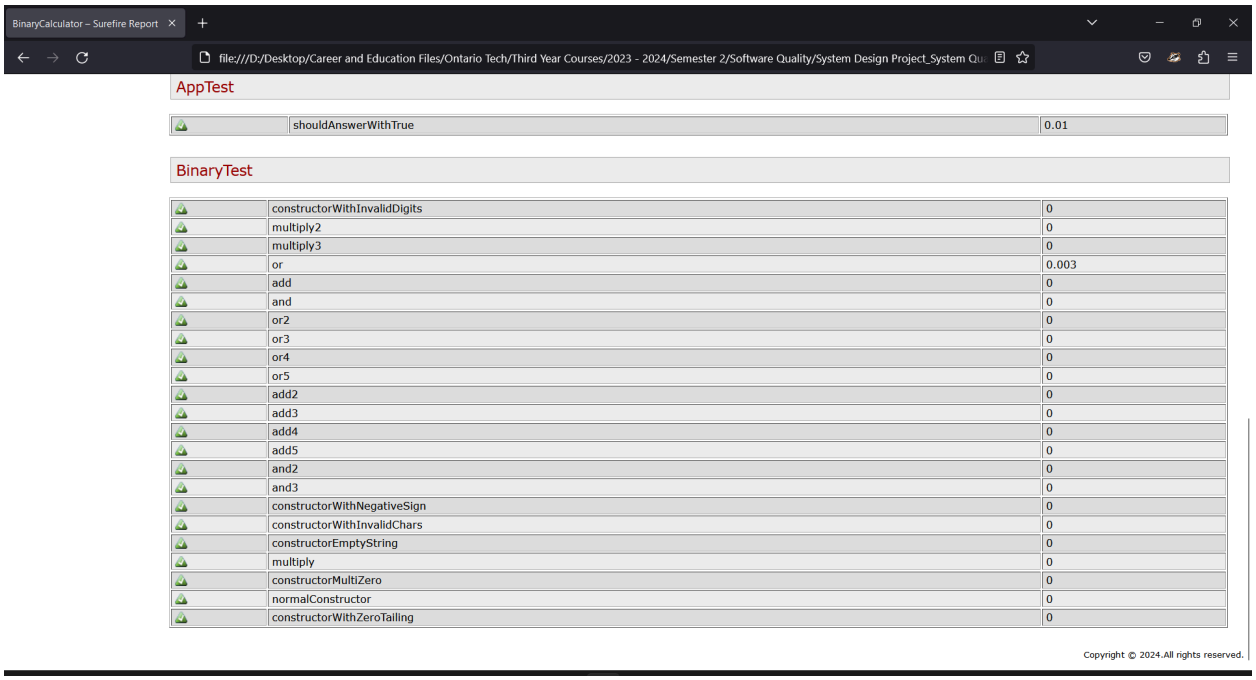
```java
@Test
public void multiply()
{
    Binary binary1=new Binary(number:"101");
    Binary binary2=new Binary(number:"111");
    Binary binary3=Binary.multiply(binary1,binary2);
    assertTrue(binary3.getValue(), binary3.getValue().equals(anObject:"100011"));
}
```

3. After all appropriate dependencies and classes are added, new functions and test methods are implemented.



4. The project is then rebuilt.





5. The web javadocs are updated.

Discussion of Source and Test Code

A simple truth table can be used to test single bit test cases, checking false false, true false, and true true outcomes. Handling for cases in which two variables of different lengths are involved can be observed in the tests written, as well as handling for cases in which leading zeros are a factor. The source code was refactored extensively to accommodate for the tests, ensuring that loops ensue properly regardless of variable length, and that situations affected by leading zeros are accounted for.