# Lab #1 - Maven & Unit Testing

SOFE 3980U: Software Quality


Huzaifa Zia (100779087)

Nived Leju Ramachandran Sonia (100782317)

Onosen Aziegbe (100741943)

Ammar Salmawy (100756573)


**CRN: 73385**

**Group D1**

**Instructor: Muhammad El-Darieby**

**Sunday, Jan 27, 2023**

# I. Introduction

Link to recording of submission:
https://www.youtube.com/watch?v=V2h6uZe-7c0&ab_channel=HuzaifaZia

This lab goes over Maven and basic unit testing. The objectives are to install and be familiar with Maven, which is a software project management tool. Understand how to create, configure and build MAven projects. Be able to automatically generate Documentation of the project. Be able to configure the project to automatically add dependencies to the jar file. Finally, learn how to write and run tests in the project.

# II. Discussion

### *Source Code:*

```
public static Binary OR(Binary bin1, Binary bin2) {
    // Parse the binary strings to integers and perform the OR operation
    int num1 = Integer.parseInt(bin1.getValue(), radix: 2);
    int num2 = Integer.parseInt(bin2.getValue(), radix: 2);
    int or = num1 | num2;
    return new Binary(Integer.toBinaryString(or));
}
```

Figure 1: Bitwise OR operation function

**Explanation:**
The bitwise OR operation method takes in two binary numbers as parameters. It parses them into integers and then performs the OR operation on them. The method returns a new Binary number.

```
public static Binary AND(Binary bin1, Binary bin2) {
    // Parse the binary strings to integers and perform the AND operation
    int num1 = Integer.parseInt(bin1.getValue(), radix: 2);
    int num2 = Integer.parseInt(bin2.getValue(), radix: 2);
    int and = num1 & num2;
    return new Binary(Integer.toBinaryString(and));
}
```

Figure 2: Bitwise AND operation function

**Explanation:**
The bitwise AND operation method takes in two binary numbers as parameters and then parses them into integers. It then performs the AND operation on them and returns a new Binary number.

```java
public static Binary multiply(Binary bin1, Binary bin2) {
    // Parse the binary strings to integers and perform the multiplication oper
    Binary result = new Binary(number: "0");
    Binary num1 = new Binary(bin1.getValue());
    Binary num2 = new Binary(bin2.getValue());
    // Loop runs until value of num2 is 0. Loop checks if last digit of num2 is
    // if so then
    // add num1 to result using the add function.
    while (!num2.getValue().equals(anObject: "0")) {
        if (num2.getValue().endsWith(suffix: "1")) {
            result = add(result, num1);
        }
        // A is added to itself
        num1 = add(num1, num1);
        // Will run until num2 is 0
        num2 = new Binary(num2.getValue().substring(beginIndex: 0, num2.getValue
    }
    return result;
}
```

Figure 3: Bitwise MULTIPLY operation function

**Explanation:**

The bitwise multiply operation method takes in two binary numbers as parameters. It sets the result binary value to zero and creates two new binary numbers based on the parameters. Then the while loop runs until the value of num2 does not equal zero. If num2's value ends with one, then the result will be added with num1. After A is added to itself and num2 is shifted right once. The result is then returned.

```java
// DESIGN QUESTIONS

// Construct Binary objects
Binary bin1 = new Binary(number: "1001001");
Binary bin2 = new Binary(number: "0110110");

// Apply operations to binary objects
Binary orRes = Binary.OR(bin1, bin2);
Binary andRes = Binary.AND(bin1, bin2);
Binary mulRes = Binary.multiply(bin1, bin2);

// Print results
System.out.println(x: "/n Design Questions /n");
System.out.println("The following operations will be done on " + bin1.getValue() + " and " + bin2.getValue());
System.out.println("Bitwise OR operation result: " + orRes.getValue());
System.out.println("Bitwise AND operation result:" + andRes.getValue());
System.out.println("Bitwise MULTIPLICATION operation result:" + mulRes.getValue());
```

Figure 4: Add.java code

**Explanation:**
In the Constructor class, two binary objects are created with random binary values. The three bitwise operations are done on those objects and then finally the results are printed.

### *Testing Code:*

```
// OR TEST FUNCTIONS
/**
 * Test The OR functions with two binary numbers of the same length
 */
@Test
public void or() {
    Binary binary1 = new Binary(number: "1000");
    Binary binary2 = new Binary(number: "1111");
    Binary binary3 = Binary.OR(binary1, binary2);
    assertTrue(binary3.getValue().equals(anObject: "1111"));
}

/**
 * Test The OR functions with two binary numbers, the length of the first being
 * Longer
 */
@Test
public void or2() {
    Binary binary1 = new Binary(number: "1010");
    Binary binary2 = new Binary(number: "11");
    Binary binary3 = Binary.OR(binary1, binary2);
    assertTrue(binary3.getValue().equals(anObject: "1011"));
}

/**
 * Test The OR functions with two binary numbers, the length of the second being
 * Longer
 */
@Test
public void or3() {
    Binary binary1 = new Binary(number: "11");
    Binary binary2 = new Binary(number: "1010");
    Binary binary3 = Binary.OR(binary1, binary2);
    assertTrue(binary3.getValue().equals(anObject: "1011"));
}
```

Figure 5: Bitwise OR operation test functions

**Explanation:**
These are the test cases for the OR function. Each test case is given different numbers and then the assertTrue method gets those values and checks to see if they are equal to the answer provided. All these test cases passed.

```java
// AND TEST FUNCTIONS
/**
 * Test The AND functions with two binary numbers of the same length
 */
@Test
public void and() {
    Binary binary1 = new Binary(number: "1000");
    Binary binary2 = new Binary(number: "1111");
    Binary binary3 = Binary.AND(binary1, binary2);
    assertTrue(binary3.getValue().equals(anObject: "1000"));
}

/**
 * Test The AND functions with two binary numbers, the length of the first being
 * longer
 */
@Test
public void and2() {
    Binary binary1 = new Binary(number: "1010");
    Binary binary2 = new Binary(number: "11");
    Binary binary3 = Binary.AND(binary1, binary2);
    assertTrue(binary3.getValue().equals(anObject: "10"));
}

/**
 * Test The AND functions with two binary numbers, the length of the second
 * being longer
 */
@Test
public void and3() {
    Binary binary1 = new Binary(number: "11");
    Binary binary2 = new Binary(number: "1010");
    Binary binary3 = Binary.AND(binary1, binary2);
    assertTrue(binary3.getValue().equals(anObject: "10"));
}
```

Figure 6: Bitwise AND operation test functions

**Explanation:**

These are the test cases for the AND function. Each test case is given different numbers and then the assertTrue method gets those values and checks to see if they are equal to the answer provided. All these test cases passed.

```java
// Multiply TEST FUNCTIONS
/**
 * Test The Multiply functions with two binary numbers of the same length
 */
@Test
public void multiply() {
    Binary binary1 = new Binary(number: "1000");
    Binary binary2 = new Binary(number: "1111");
    Binary binary3 = Binary.multiply(binary1, binary2);
    assertTrue(binary3.getValue().equals(anObject: "1111000"));
}

/**
 * Test The Multiply functions with two binary numbers, the length of the first
 * being longer
 */
@Test
public void multiply2() {
    Binary binary1 = new Binary(number: "1010");
    Binary binary2 = new Binary(number: "11");
    Binary binary3 = Binary.multiply(binary1, binary2);
    assertTrue(binary3.getValue().equals(anObject: "11110"));
}

/**
 * Test The Multiply functions with two binary numbers, the length of the secon
 * being longer
 */
@Test
public void multiply3() {
    Binary binary1 = new Binary(number: "11");
    Binary binary2 = new Binary(number: "1010");
    Binary binary3 = Binary.multiply(binary1, binary2);
    assertTrue(binary3.getValue().equals(anObject: "11110"));
}
```

Figure 7: Bitwise MULTIPLY operation test functions

**Explanation:**

These are the test cases for the multiply function. Each test case is given different numbers and then the assertTrue method gets those values and checks to see if they are equal to the answer provided. All these test cases passed.