Maven & Unit Testing

Lab 1:

Student Name	Student #
Issaq Momand	100868239

Source Code Design and Testing for Binary Operations

1. Introduction

This report discusses the design and implementation of the Binary class and its associated operations. It also covers the testing methodology employed to verify the correctness of the implemented operations. The Binary class provides a framework to manipulate binary numbers using operations such as addition, bitwise OR, bitwise AND, and multiplication.

---

2. Source Code Design

Constructor: Validates the input binary string, removes leading zeros, and handles invalid inputs gracefully by defaulting to "0".

public Binary(String number) { ... }

Addition (add): Implements binary addition using a carry mechanism.

public static Binary add(Binary num1, Binary num2) { ... }

Bitwise OR (or): Performs a bitwise logical OR operation on two binary numbers.

public static Binary or(Binary b1, Binary b2) { ... }

Bitwise AND (and): Performs a bitwise logical AND operation on two binary numbers.

public static Binary and(Binary b1, Binary b2) { ... }

Multiplication (multiply): Multiplies two binary numbers using repeated addition.

public static Binary multiply(Binary b1, Binary b2) { ... }

Interaction with App Class

The App class serves as the entry point for user interaction. It collects user input for two binary numbers and performs operations such as addition, OR, AND, and multiplication. Results are displayed in a user-friendly manner.

---

3. Testing Methodology within BinaryTest.java

3.1. Testing Objectives

The objective of the tests is to verify:

- The correctness of the binary operations.
- The handling of edge cases such as invalid inputs, leading zeros, and operations with "0".

3.2. Test Cases

The testing was conducted using the JUnit framework. Here are the key tests implemented:

Addition Tests:

```
@Test
void testAddition() {
   Binary b1 = new Binary("1010");
   Binary b2 = new Binary("110");
   Binary result = Binary.add(b1, b2);
   assertEquals("10000", result.getValue());
}
```

- Verifies proper addition of two binary numbers.
- Ensures carry is handled correctly.

Bitwise OR Tests:

```
@Test
void testOr() {
   Binary b1 = new Binary("1010");
   Binary b2 = new Binary("1100");
   Binary result = Binary.or(b1, b2);
   assertEquals("1110", result.getValue());
}
```

- Confirms that bitwise OR is performed correctly.

Bitwise AND Tests:

```
@Test
void testAnd() {
   Binary b1 = new Binary("1010");
```

```
    Binary b2 = new Binary("1100");
    Binary result = Binary.and(b1, b2);
    assertEquals("1000", result.getValue());
}
```

- Validates that bitwise AND works as expected.

Multiplication Tests:

```
@Test
void testMultiply() {
    Binary b1 = new Binary("101");
    Binary b2 = new Binary("10");
    Binary result = Binary.multiply(b1, b2);
    assertEquals("1010", result.getValue());
}
```

- Verifies multiplication using repeated addition.

3.3. Edge Case Tests

Empty and Invalid Inputs:

```
@Test
void testInvalidInput() {
    Binary b1 = new Binary("");
    Binary b2 = new Binary("abc");
    assertEquals("0", b1.getValue());
    assertEquals("0", b2.getValue());
}
```

- Confirms that invalid or empty inputs default to "0".

Leading Zeros:

```
@Test
void testLeadingZeros() {
    Binary b1 = new Binary("001010");
    Binary b2 = new Binary("000001");
    assertEquals("1010", b1.getValue());
    assertEquals("1", b2.getValue());
}
```

- Ensures leading zeros are removed.

---

4. Results and Observations

- All operations were successfully tested and produced correct results for typical and edge cases.
- The use of helper functions such as padZeros ensured proper alignment of binary strings during operations.
- The multiplication method demonstrated that complex operations could be built using simpler ones like addition.

---

5. Conclusion

The Binary class is a robust implementation for binary number operations, incorporating design principles like encapsulation, reusability, and modularity. The associated test cases effectively validate the functionality and handle edge cases. Future enhancements could include additional binary operations, improved user interaction in the App class, and performance optimization for large binary numbers.