Objectives.

1. Become familiar with the `CLEmitter`, an abstraction for generating JVM bytecode.
2. Extend the base *j--* language by adding some basic Java operations (on primitive integers) to the language. Supporting these operations requires studying the *j--* compiler in its entirety, if only cursorily, and then making slight modifications to it. Notice that many of the operations have different levels of precedence, just as `*` has a different level of precedence in *j--* than does `+`. These levels of precedence are captured in the Java grammar (see Appendix C of our text); for example, the parser uses one method to parse expressions involving `*` and `/`, and another to parse expressions involving `+` and `-`.

Download and Test the *j--* Compiler.

Download and unzip the base *j--* compiler under some directory (we'll refer to this directory as `$j`). See Appendix A for information on what's in the *j--* distribution.

Run the following command inside the `$j` directory to compile the *j--* compiler.

```
$ ant clean compile jar
```

Run the following command to compile a *j--* program `P.java` using the *j--* compiler, which produces the JVM target program `P.class`.

```
$ sh $j/j--/bin/j-- P.java
```

Run the following command to run `P.class`.

```
$ java P
```


Problem 1. (*Using `CLEmitter`*) Consider the following program `IsPrime.java` that receives an integer n as command-line argument and prints whether or not n is a prime number.

```
// IsPrime.java

public class IsPrime {
    // Returns true if n is prime, and false otherwise.
    private static boolean isPrime(int n) {
        if (n < 2) {
            return false;
        }
        for (int i = 2; i < n / i; i++) {
            if (n % i == 0) {
                return false;
            }
        }
        return true;
    }

    // Entry point.
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        boolean result = isPrime(n);
        if (result) {
            System.out.println(n + " is a prime number");
        }
        else {
            System.out.println(n + " is not a prime number");
        }
    }
}
```

Using programs under `$j/j--/tests/clemitter` as a guide, complete the implementation of the program `GenIsPrime.java` that uses the `CLEmitter` interface to programmatically generate `IsPrime.class`, ie, the JVM bytecode for the program `IsPrime.java` above.

```
$ javac -cp .:$j/j--/lib/j--.jar GenIsPrime.java
$ java -cp .:$j/j--/lib/j--.jar GenIsPrime
$ java IsPrime 42
42 is not a prime number
$ java IsPrime 31
31 is a prime number
```

Problem 2. (*Division Operation*) Follow the process outlined in Section 1.5 of our text to implement the Java division operator `/`.

```
$ $j/j--/bin/j-- tests/Division.java
$ java Division 42 6
7
```

Problem 3. (*Remainder Operation*) Implement the Java remainder operator `%`.

```
$ $j/j--/bin/j-- tests/Remainder.java
$ java Remainder 42 13
3
```

Problem 4. (*Shift Operations*) Implement the Java shift operators: arithmetic left shift `<<`, arithmetic right shift `>>`, logical right shift `>>>`.


```
$ $j/j--/bin/j-- tests/ArithmeticLeftShift.java
$ java ArithmeticLeftShift 1 5
32
```

```
$ $j/j--/bin/j-- tests/ArithmeticRightShift.java
$ java ArithmeticRightShift 32 5
1
$ java ArithmeticRightShift -32 5
-1
```

```
$ $j/j--/bin/j-- tests/LogicalRightShift.java
$ java LogicalRightShift 32 5
1
$ java LogicalRightShift -32 5
134217727
```

Problem 5. (*Bitwise Operations*) Implement the Java bitwise operators: unary complement `~`, inclusive or `|`, exclusive or `^`, and `&`.

```
$ $j/j--/bin/j-- tests/BitwiseNot.java
$ java BitwiseNot 42
-43
```

```
$ $j/j--/bin/j-- tests/BitwiseInclusiveOr.java
$ java BitwiseInclusiveOr 3 5
7
```

```
$ $j/j--/bin/j-- tests/BitwiseExclusiveOr.java
$ java BitwiseExclusiveOr 3 5
6
```

```
$ $j/j--/bin/j-- tests/BitwiseAnd.java
$ java BitwiseAnd 3 5
1
```

Problem 6. (*Unary Plus Operation*) Implement the Java unary plus operaor `+`.

```
$ $j/j--/bin/j-- tests/UnaryPlus.java
$ java UnaryPlus -42
-42
```

Files to Submit

1. `GenIsPrime.java` (`CLEmitter` program that generates `IsPrime.class`)
2. `j--.zip` (*j--* source tree as a single zip file)
3. `report.txt` (project report)

Before you submit:

- Make sure you create the zip file `j--.zip` such that it only includes the source files and not the binaries, which can be done on the terminal as follows:

```
$ cd $j/j--
$ ant clean
$ cd ..
$ tar -cvf j--.tar j--/*
$ gzip j--.tar
```

- Make sure your report isn't too verbose, doesn't contain lines that exceed 80 characters, and doesn't contain spelling/grammatical mistakes