Goal: Become familiar with the Java Virtual Machine (JVM) and the Marvin Machine; and extend the *j--* language by adding support for some arithmetic operators, conditional expression, and do statement.

Zip File: Download and unzip the zip file for the assignment under `$j/j--`.

Problem 1. (*Understanding the JVM*) Consider the following program `IsPrime.java` that accepts n (int) as command-line argument, and writes whether or not n is a prime number.

IsPrime.java

```
public class IsPrime {
    // Entry point.
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        boolean result = isPrime(n);
        System.out.println(result);
    }

    // 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;
    }
}
```

Complete the implementation of the program `$j/j--/simpleops/GenIsPrime.java` such that it uses the `CLEmitter` interface to programmatically generate `IsPrime.class`, ie, the JVM bytecode for the `IsPrime.java` program listed above.

>_ ~/workspace/j--

```
$ ./bin/clemitter simpleops/GenIsPrime.java
$ java IsPrime 42
false
$ java IsPrime 31
true
```

Problem 2. (*Understanding the Marvin Machine*) Implement the `isPrime()` function in the Marvin program `$j/j--/simpleops/IsPrime.marv` such that it returns `1` if the input `n` is prime and `0` otherwise. The function receives the input in register `r0` and is expected to store the return value (ie, the result) in register `r13`.

>_ ~/workspace/j--

```
$ python3 simpleops/marvin.py simpleops/IsPrime.marv
42
0
$ python3 simpleops/marvin.py simpleops/IsPrime.marv
31
1
```

***Java Lite*:** Consult the *Java Lite* Language Specification for the syntactic and semantic rules that you must follow when you make changes to the *j--* language described in the problems below. Pay close attention to any problem-specific exceptions to the language rules, which will be stated in the problem description.

Problem 3. (*Arithmetic Operators*) Implement the division `/`, remainder `%`, and unary plus `+` operators in *j--*.

AST representations to use:

- `JDivideOp` in `JBinaryExpression.java` for `/`.
- `JRemainderOp` in `JBinaryExpression.java` for `%`.
- `JUnaryPlusOp` in `JUnaryExpression.java` for unary `+`.

```
>_ ~/workspace/j--
$ ant
$ ./bin/j-- simpleops/Division.java
$ java Division 60 13
4
$ ./bin/j-- simpleops/Remainder.java
$ java Remainder 60 13
8
$ ./bin/j-- simpleops/UnaryPlus.java
$ java UnaryPlus 60
60
```

Problem 4. (*Conditional Expression*) Add support for conditional expression (`e ? e1 : e2`) in *j--*. Here are the grammar rules related to conditional expression:

```
assignmentExpression ::= conditionalExpression [ ( ASSIGN | PLUS_ASSIGN ) assignmentExpression ]

conditionalExpression ::= conditionalAndExpression [ QUESTION expression COLON conditionalExpression ]
```

Use `JConditionalExpression.java` as the AST representation for a conditional expression.

```
>_ ~/workspace/j--
$ ant
$ ./bin/j-- simpleops/ConditionalExpression.java
$ java ConditionalExpression
Tails
$ java ConditionalExpression
Tails
$ java ConditionalExpression
Heads
```

Problem 5. (*Do Statement*) Add support for a do statement in *j--*. Use `JDoStatement.java` as the AST representation for a do statement.

```
>_ ~/workspace/j--
$ ant
$ ./bin/j-- simpleops/DoStatement.java
$ java DoStatement 100
5050
```

Files to Submit:

1. `GenIsPrime.java`
2. `IsPrime.marv`
3. `JBinaryExpression.java`
4. `JConditionalExpression.java`
5. `JDoStatement.java`
6. `JUnaryExpression.java`
7. `Parser.java`
8. `Scanner.java`
9. `TokenInfo.java`
10. `notes.txt`

Before you submit your files, make sure:

- Your code is adequately commented and follows good programming principles.
- You edit the sections (`#1` mandatory, `#2` if applicable, and `#3` optional) in the given `notes.txt` file as appropriate. Section `#1` must provide a clear high-level description of each problem in no more than 100 words.