Exercise 1. Consider the *j*-- program `$j/j--/tests/SumOfPowers.java`. Assuming you are within the `$j/j--` folder:

a. What is the command to compile the *j*-- compiler?

b. What is the command for just tokenizing the program?

c. What is the command for just parsing the program?

d. What is the command for just pre-analyzing the program?

e. What is the command for just analyzing the program?

f. What is the command for compiling the program into a folder called `out` under `$j/j--`?

g. What is the command for compiling the program into the current folder?

h. What is the command for running `SumOfPowers.class` with command-line arguments 10 and 2?

i. What is the command for decompiling `SumOfPowers.class`, ie, displaying its JVM bytecode instructions?

Exercise 2. Write a *j*-- program called `Countdown.java` that accepts n (int) as command-line argument and writes to standard output a countdown from n to 0, both inclusive.

× ~/workspace/j--

```
$ ./bin/j-- Countdown.java
$ java Countdown 5
5
4
3
2
1
0
$ _
```

Exercise 3. Write a *j*-- program called `IsPrime.java` that accepts n (int) as command-line argument and writes to standard output wheter n is prime or not.

× ~/workspace/j--

```
$ ./bin/j-- IsPrime.java
$ java IsPrime 31
true
$ java IsPrime 42
false
$ _
```

Exercise 4. Express the following class names in internal form:

a. `Countdown`

b. `jminusminus.Parser`

c. `java.io.File`

d. `java.util.ArrayList`

Exercise 5. Give the field descriptors for the following variable declarations:

a. `boolean w;`

b. `long x;`

c. `java.io.File[] y;`

d. `double[][] z;`

Exercise 6. Give the method descriptors for the following constructor/method declarations:

a. `public Employee(String name) { ... }`

b. `public Coordinate(double latitude, double longitude) { ... }`

c. `public Object get(String key) { ... }`

d. `public void put(String key, Object value) { ... }`

e. `public static void sort(int[] a, Boolean ascending) { ... }`

f. `public static Double[][] transpose(double[][] a) { ... }`

Exercise 7. Consider the following JVM bytecode:

```
1  private static int mystery(int);
2     0: iconst_1
3     1: istore_1
4     2: iconst_0
5     3: istore_2
6     4: iload_1
7     5: iload_0
8     6: if_icmpgt      21
9     9: iload_2
10   10: iload_1
11   11: iload_1
12   12: imul
13   13: iadd
14   14: istore_2
15   15: iinc           1, 1
16   18: goto           4
17   21: iload_2
18   22: ireturn
```

a. What does the method call `mystery(10)` return?

b. What does the method call `mystery(n)` return in general?

Exercise 8. Write a program called `GenCountdown.java` that uses the `CLEmitter` interface to generate a program called `Countdown.class` that behaves exactly like the *j*-- program described in Exercise 2.

```
× ~/workspace/j--
$ ./bin/clemitter GenCountdown.java
$ java Countdown 5
5
4
3
2
1
0
$ _
```

Exercise 9. Write a program called `GenIsPrime.java` that uses the `CLEmitter` interface to generate a program called `IsPrime.class` that behaves exactly like the *j*-- program described in Exercise 3.

```
× ~/workspace/j--
$ ./bin/clemitter GenIsPrime.java
$ java IsPrime 31
true
$ java IsPrime 42
false
$ _
```

Exercise 10. Implement the division operator `/` in *j*--.

Solutions

Solution 1.

a. `ant`

b. `./bin/j-- -t tests/SumOfPowers.java`

c. `./bin/j-- -p tests/SumOfPowers.java`

d. `./bin/j-- -pa tests/SumOfPowers.java`

e. `./bin/j-- -a tests/SumOfPowers.java`

f. `./bin/j-- -d out tests/SumOfPowers.java`

g. `./bin/j-- tests/SumOfPowers.java`

h. `java SumOfPowers 10 2`

i. `javap -p -v SumOfPowers`

Solution 2.

× Countdown.java

```
1  import java.lang.Integer;
2  import java.lang.System;
3
4  public class Countdown {
5      public static void main(String[] args) {
6          int n = Integer.parseInt(args[0]);
7          while (n > -1) {
8              System.out.println(n--);
9          }
10     }
11 }
```

Solution 3.

× IsPrime.java

```
1  import java.lang.Integer;
2  import java.lang.System;
3
4  public class IsPrime {
5      // Entry point.
6      public static void main(String[] args) {
7          int n = Integer.parseInt(args[0]);
8          int i = 2;
9          boolean isPrime = true;
10         if (n <= 1) {
11             isPrime = false;
12         }
13         while (isPrime && i * i <= n) {
14             if (mod(n, i) == 0) {
15                 isPrime = false;
```

```
            }
            ++i;
        }
        System.out.println(isPrime);
    }

    // Returns the remainder of a (>= 0) and b (>= 0) computed recursively.
    private static int mod(int a, int b) {
        if (b == 0) {
            return a;
        } else if (a == b) {
            return 0;
        } else if (a > b) {
            return mod(a - b, b);
        }
        return a;
    }
}
```

Solution 4.

a. `Countdown`

b. `jminusminus/Parser`

c. `java/io/File`

d. `java/util/ArrayList`

Solution 5.

a. `Z`

b. `J`

c. `[Ljava/io/File;`

d. `[[D`

Solution 6.

a. `(Ljava/lang/String;)V`

b. `(DD)V`

c. `(Ljava/lang/String;)Ljava/lang/Object;`

d. `(Ljava/lang/String;Ljava/lang/Object;)V`

e. `([ILjava/lang/Boolean;)V`

f. `([[D)[[Ljava/lang/Double;`

Solution 7.

a. 385

b. $1^2 + 2^2 + 3^2 + \cdots + (n-1)^2 + n^2$ (ie, $\Sigma_{i=1}^{n} i^2$)

Solution 8.

× GenCountdown.java

```
import java.util.ArrayList;
import jminusminus.CLEmitter;
import static jminusminus.CLConstants.*;

public class GenCountdown {
    public static void main(String[] args) {
        CLEmitter e = new CLEmitter(true);
        ArrayList<String> modifiers = new ArrayList<String>();

        // public class Countdown {
        modifiers.add("public");
        e.addClass(modifiers, "Countdown", "java/lang/Object", null, true);

        //     public static void main(String[] args) {
        modifiers.add("public");
        modifiers.add("static");
        e.addMethod(modifiers, "main", "([Ljava/lang/String;)V", null, true);

        //         int n = Integer.parseInt(args[0]);
        e.addNoArgInstruction(ALOAD_0);
        e.addNoArgInstruction(ICONST_0);
        e.addNoArgInstruction(AALOAD);
        e.addMemberAccessInstruction(INVOKESTATIC, "java/lang/Integer", "parseInt",
            "(Ljava/lang/String;)I");
        e.addNoArgInstruction(ISTORE_1);

        //         while (n > -1) {
        e.addLabel("Repeat");
        e.addNoArgInstruction(ILOAD_1);
        e.addNoArgInstruction(ICONST_M1);
        e.addBranchInstruction(IF_ICMPLE, "Done");

        //             System.out.println(n--);
        e.addMemberAccessInstruction(GETSTATIC, "java/lang/System", "out",
            "Ljava/io/PrintStream;");
        e.addNoArgInstruction(ILOAD_1);
        e.addMemberAccessInstruction(INVOKEVIRTUAL, "java/io/PrintStream", "println",
            "(I)V");
        e.addIINCInstruction(1, -1);
        e.addBranchInstruction(GOTO, "Repeat");

        //         }
        e.addLabel("Done");

        //     }
        e.addNoArgInstruction(RETURN);

        // }
        e.write();
    }
}
```

Solution 9.

× GenIsPrime.java

```
import java.util.ArrayList;
import jminusminus.CLEmitter;
import static jminusminus.CLConstants.*;

public class GenIsPrime {
    public static void main(String[] args) {
        CLEmitter e = new CLEmitter(true);
        ArrayList<String> modifiers = new ArrayList<String>();

        // public class IsPrime {
        modifiers.add("public");
        e.addClass(modifiers, "IsPrime", "java/lang/Object", null, true);

        //     public static void main(String[] args) {
        modifiers.add("public");
        modifiers.add("static");
        e.addMethod(modifiers, "main", "([Ljava/lang/String;)V", null, true);

        //         int n = Integer.parseInt(args[0]);
        e.addNoArgInstruction(ALOAD_0);
        e.addNoArgInstruction(ICONST_0);
        e.addNoArgInstruction(AALOAD);
        e.addMemberAccessInstruction(INVOKESTATIC, "java/lang/Integer", "parseInt",
            "(Ljava/lang/String;)I");
        e.addNoArgInstruction(ISTORE_1);

        //         System.out.println(isPrime(n));
        e.addMemberAccessInstruction(GETSTATIC, "java/lang/System", "out",
            "Ljava/io/PrintStream;");
        e.addNoArgInstruction(ILOAD_1);
        e.addMemberAccessInstruction(INVOKESTATIC, "IsPrime", "isPrime", "(I)Z");
        e.addMemberAccessInstruction(INVOKEVIRTUAL, "java/io/PrintStream", "println",
            "(Z)V");

        //     }
        e.addNoArgInstruction(RETURN);

        //     private static boolean isPrime(int n) {
        modifiers.clear();
        modifiers.add("private");
        modifiers.add("static");
        e.addMethod(modifiers, "isPrime", "(I)Z", null, true);

        //         if (n < 2) { return false; }
        e.addNoArgInstruction(ILOAD_0);
        e.addNoArgInstruction(ICONST_2);
        e.addBranchInstruction(IF_ICMPLT, "NotPrime");

        //         int i = 2;
        e.addNoArgInstruction(ICONST_2);
```

```
        e.addNoArgInstruction(ISTORE_1);

        //          while (i * i <= n) {
        e.addLabel("Repeat");
        e.addNoArgInstruction(ILOAD_1);
        e.addNoArgInstruction(ILOAD_1);
        e.addNoArgInstruction(IMUL);
        e.addNoArgInstruction(ILOAD_0);
        e.addBranchInstruction(IF_ICMPGT, "Prime");

        //              if (n % i == 0) { return false; }
        e.addNoArgInstruction(ILOAD_0);
        e.addNoArgInstruction(ILOAD_1);
        e.addNoArgInstruction(IREM);
        e.addBranchInstruction(IFEQ, "NotPrime");

        //              i++;
        e.addIINCInstruction(1, 1);

        //          }
        e.addBranchInstruction(GOTO, "Repeat");

        // Return true.
        e.addLabel("Prime");
        e.addNoArgInstruction(ICONST_1);
        e.addNoArgInstruction(IRETURN);

        // Return false.
        e.addLabel("NotPrime");
        e.addNoArgInstruction(ICONST_0);
        e.addNoArgInstruction(IRETURN);

        // }
        e.write();
    }
}
```

Solution 10.
Changes to `TokenInfo.java`:

```
enum TokenKind {
    ...
    DIV ("/"),
    ...
}
```

Changes to `Scanner.java`:

```
   ...
            if (ch == '/') {
                nextCh();
                if (ch == '/') {
                    // CharReader maps all new lines to '\n'.
```

```
                    while (ch != '\n' && ch != EOFCH) {
                        nextCh();
                    }
                } else {
                    return new TokenInfo(DIV, line);
                }
            }
    ...
```

Changes to `Parser.java`:

```
    /**
     * Parses a multiplicative expression and returns an AST for it.
     *
     * <pre>
     *   multiplicativeExpression ::= unaryExpression { ( DIV | STAR )
                                                        unaryExpression }
     * </pre>
     *
     * @return an AST for a multiplicative expression.
     */
    private JExpression multiplicativeExpression() {
        int line = scanner.token().line();
        boolean more = true;
        JExpression lhs = unaryExpression();
        while (more) {
            if (have(STAR)) {
                lhs = new JMultiplyOp(line, lhs, unaryExpression());
            }
            else if (have(DIV)) {
                lhs = new JDivideOp(line, lhs, unaryExpression());
            }
            else {
                more = false;
            }
        }
        return lhs;
    }
```

Change to `analyze()` and `codegen()` in `JDivideOp` defined in `JBinaryExpression.java`:

```
class JDivideOp extends JBinaryExpression {
    public JExpression analyze(Context context) {
        lhs = (JExpression) lhs.analyze(context);
        rhs = (JExpression) rhs.analyze(context);
        lhs.type().mustMatchExpected(line(), Type.INT);
        rhs.type().mustMatchExpected(line(), Type.INT);
        type = Type.INT;
        return this;
    }

    public void codegen(CLEmitter output) {
        lhs.codegen(output);
        rhs.codegen(output);
```

```
14          output.addNoArgInstruction(IDIV);
15      }
16  }
```