Goal: Modify the handcrafted parser to support `long` and `double` basic types; additional operators; and for, break, continue, and switch statements in *j*--.

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

***Java Lite*:** Consult the *Java Lite* Language Specification for the syntactic rules that you must follow when you make changes to the *j*-- language described in the problems below.

Problem 1. (*Operators*) Add support for the following operators:

```
-=   *=   /=   %=   !=   >=   <   ||   ++   --
```

AST representations to use:

- `JMinusAssignOp`, `JStarAssignOp`, `JDivAssignOp`, and `JRemAssignOp` in `JAssignment.java` for `-=`, `*=`, `/=`, and `%=`.
- `JNotEqualOp` in `JBooleanBinaryExpression.java` for `!=`.
- `JGreaterEqualOp` and `JLessThanOp` in `JComparisonExpression.java` for `>=` and `<`.
- `JLogicalOrOp` in `JBooleanBinaryExpression.java` for `||`.
- `JPreDecrementOp` and `JPostIncrementOp` in `JUnaryExpression.java` for pre `--` and post `++`.

```
× ~/workspace/j--
$ ant
$ ./bin/j-- -p parsing/Operators.java
```

Compare your output with the reference output in `parsing/Operators.ast`.

Problem 2. (*Long and Double Basic Types*) Add support for the `long` and `double` basic types. Use `JLiteralLong` and `JLiteralDouble` as the AST representation for a `long` and `double` literal, respectively.

```
× ~/workspace/j--
$ ant
$ ./bin/j-- -p parsing/Factorial.java
$ ./bin/j-- -p parsing/Quadratic.java
```

Compare your output with the reference output in `parsing/Factorial.ast` and `parsing/Quadratic.ast`.

Problem 3. (*For Statement*) Add support for a for statement. Use `JForStatement.java` as the AST representation for a for statement.

```
× ~/workspace/j--
$ ant
$ ./bin/j-- -p parsing/ForStatement.java
```

Compare your output with the reference output in `parsing/ForStatement.ast`.

Problem 4. (*Break Statement*) Add support for a break statement. Use `JBreakStatement.java` as the AST representation for a break statement.

```
× ~/workspace/j--
$ ant
$ ./bin/j-- -p parsing/BreakStatement.java
```

Compare your output with the reference output in `parsing/BreakStatement.ast`.

Problem 5. (*Continue Statement*) Add support for a continue statement. Use `JContinueStatement.java` as the AST representation for a continue statement.

```
× ~/workspace/j--
$ ant
$ ./bin/j-- -p parsing/ContinueStatement.java
```

Compare your output with the reference output in `parsing/ContinueStatement.ast`.

Problem 6. (*Switch Statement*) Add support for a switch statement. Use `JSwitchStatement.java` as the AST representation for a switch statement.

```
× ~/workspace/j--
$ ant
$ ./bin/j-- -p parsing/SwitchStatement.java
```

Compare your output with the reference output in `parsing/SwitchStatement.ast`.

Files to Submit:

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

Before you submit your files, make sure:

- Your code is clean, well-organized, uses meaningful variable names, includes useful comments, and is efficient.
- You edit the sections (`#1` mandatory, `#2` if applicable, and `#3` optional) in the given `notes.txt` file as appropriate. In section `#1`, for each problem, state its goal in your own words and describe your approach to solve the problem along with any issues you encountered and if/how you managed to solve those issues.