Goal: Modify the `j--.jj` file used for generating a scanner and parser (using JavaCC) for *j--* to support multiline comments; `long` and `double` basic types; additional tokens (reserved words and operators); conditional expression; do, for, break, continue, and switch statements; exception handlers; and interface type declaration.

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 and semantic rules that you must follow when you make changes to the *j--* language described in the problems below.

Part I: Additions to the Scanner

Problem 1. (*Multiline Comment*) Add support for multiline comment, where all the text from the ASCII characters `/*` to the ASCII characters `*/` is ignored.

```
>_ ~/workspace/j--
$ ant
$ ./bin/javaccj-- -t javacc_frontend/MultilineComment.java
3        : "import" = import
3        : <IDENTIFIER> = java
...
19       : "}" = }
20       : <EOF> =
```

Compare your output with the reference output in `javacc_frontend/MultilineComment.tokens`.

Problem 2. (*Reserved Words*) Add support for the following reserved words:

```
break   case   continue   default   do   double   for   long   switch
```

```
>_ ~/workspace/j--
$ ant
$ ./bin/javaccj-- -t javacc_frontend/Keywords.java
1        : "abstract" = abstract
2        : "boolean" = boolean
...
33       : "while" = while
33       : <EOF> =
```

Compare your output with the reference output in `javacc_frontend/Keywords.tokens`.

Problem 3. (*Operators*) Add support for the following operators:

```
/    %    ?    :    -=   *=
/=   %=   !=   >=   <    ||
```

```
>_ ~/workspace/j--
$ ant
$ ./bin/javaccj-- -t javacc_frontend/Operators1.java
1        : "=" = =
2        : ":" = :
...
24       : "*=" = *=
24       : <EOF> =
```

Compare your output with the reference output in `javacc_frontend/Operators1.tokens`.

Problem 4. (*Literals*) Add support for long and double literals.

```
>_ ~/workspace/j--
$ ant
$ ./bin/javaccj-- -t javacc_frontend/IntLiterals.java
1         : <INT_LITERAL> = 0
2         : <INT_LITERAL> = 9
...
5         : <INT_LITERAL> = 1234567890
5         : <EOF> =
$ ./bin/javaccj-- -t javacc_frontend/LongLiterals.java
1         : <LONG_LITERAL> = 1l
2         : <LONG_LITERAL> = 9L
...
6         : <LONG_LITERAL> = 1234567890L
6         : <EOF> =
$ ./bin/javaccj-- -t javacc_frontend/DoubleLiterals1.java
1         : <DOUBLE_LITERAL> = 0.
2         : <DOUBLE_LITERAL> = 1.
...
74        : <DOUBLE_LITERAL> = 123456789.e-135D
74        : <EOF> =
$ ./bin/javaccj-- -t javacc_frontend/DoubleLiterals2.java
1         : <DOUBLE_LITERAL> = .0
2         : <DOUBLE_LITERAL> = .1
...
32        : <DOUBLE_LITERAL> = .098765e-135
32        : <EOF> =
$ ./bin/javaccj-- -t javacc_frontend/DoubleLiterals3.java
1         : <DOUBLE_LITERAL> = 0e2
2         : <DOUBLE_LITERAL> = 9e9
...
21        : <DOUBLE_LITERAL> = 246e-13D
21        : <EOF> =
$ ./bin/javaccj-- -t javacc_frontend/DoubleLiterals4.java
1         : <DOUBLE_LITERAL> = 0d
2         : <DOUBLE_LITERAL> = 0D
...
6         : <DOUBLE_LITERAL> = 0987654321D
6         : <EOF> =
```

Compare your output with the reference output in `javacc_frontend/IntLiterals.tokens`, `javacc_frontend/LongLiterals.tokens`, and `javacc_frontend/DoubleLiterals*.tokens`.

Part II: Additions to the Parser

Problem 5. (*Operators*) Add support for the following operators:

`/` `%` `-=` `*=` `/=` `%=` `!=` `>=` `<` `||` `+` (unary) `++` `--`

AST representations to use:

- `JDivideOp` and `JRemainderOp` in `JBinaryExpression.java` for `/` and `%`.
- `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 `||`.
- `JUnaryPlusOp`, `JPreDecrementOp` and `JPostIncrementOp` in `JUnaryExpression.java` for unary `+`, pre `--` and post `++`.

```
>_ ~/workspace/j--
$ ant
$ ./bin/javaccj-- -p javacc_frontend/Operators2.java
```

Compare your output with the reference output in `javacc_frontend/Operators2.ast`.

Problem 6. (*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/javaccj-- -p javacc_frontend/Factorial.java
$ ./bin/javaccj-- -p javacc_frontend/Quadratic.java
```

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

Problem 7. (*Conditional Expression*) Add support for conditional expression (`e ? e1 : e2`. Use `\lstinline{JConditionalExpression.java` as the AST representation for a conditional expression.

```
>_ ~/workspace/j--
$ ant
$ ./bin/javaccj-- -p javacc_frontend/ConditionalExpression.java
```

Compare your output with the reference output in `javacc_frontend/ConditionalExpression.ast`.

Problem 8. (*Do Statement*) Add support for a do statement. Use `JDoStatement.java` as the AST representation for a do statement.

```
>_ ~/workspace/j--
$ ant
$ ./bin/javaccj-- -p javacc_frontend/DoStatement.java
```

Compare your output with the reference output in `javacc_frontend/DoStatement.ast`.

Problem 9. (*For Statement*) Add support for a for statement. Use `JForStatement.java` as the AST representation for a for statement.

```
>_ ~/workspace/j--
$ ant
$ ./bin/javaccj-- -p javacc_frontend/ForStatement.java
```

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

Problem 10. (*Break Statement*) Add support for a break statement. Use `JBreakStatement.java` as the AST representation for a break statement.

```
>_ ~/workspace/j--
$ ant
$ ./bin/javaccj-- -p javacc_frontend/BreakStatement.java
```

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

Problem 11. (*Continue Statement*) Add support for a continue statement. Use `JContinueStatement.java` as the AST representation for a continue statement.

```
>_ ~/workspace/j--
$ ant
$ ./bin/javaccj-- -p javacc_frontend/ContinueStatement.java
```

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

Problem 12. (*Switch Statement*) Add support for a switch statement. Use `JSwitchStatement.java` as the AST representation for a switch statement.

```
>_ ~/workspace/j--
$ ant
$ ./bin/javaccj-- -p javacc_frontend/SwitchStatement.java
```

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

Files to Submit:

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

Before you submit your files, make sure:

- Your code follows good programming principles (ie, it is clean and well-organized; uses meaningful variable names; and includes useful comments).
- 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, you must include in nore more than 100 words: a short, high-level description of the problem; your approach to solve it; and any issues you encountered and if/how you managed to solve them.