Goal

1. Support multiline comments.
2. Support additional tokens (reserved words, operators, and separators).
3. Support `long` and `double` literals.

Download the Project Tests

Download and unzip the tests for this project under `$j/j--`.

In this project, you will only be updating the hand-crafted scanner, which means that the only program files you will be modifying under `$j/j--/src/jminusminus` are `TokenInfo.java` and `Scanner.java`.

Run the following command inside the `$j/j--` directory to compile the *j--* compiler with your changes.

```
>_ ~/workspace/j--
$ ant
```

Run the following command to compile (just scan for now) a *j--* program `P.java` using the *j--* compiler.

```
>_ ~/workspace/j--
$ bash $j/j--/bin/j-- -t P.java
```

which only scans `P.java` and prints the tokens in the program along with the line number where each token appears.

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--
$ bash ./bin/j-- -t project2/tests/MultiLineComment.java
5         : public = public
5         : class = class
5         : <IDENTIFIER> = MultiLineComment
5         : { = {
9         : public = public
9         : static = static
9         : void = void
9         : <IDENTIFIER> = main
9         : ( = (
9         : <IDENTIFIER> = String
9         : [ = [
9         : ] = ]
9         : <IDENTIFIER> = args
9         : ) = )
9         : { = {
13        : } = }
14        : } = }
15        : <EOF> = <EOF>
```

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

```
break         case          catch
continue      default       do
double        final         finally
for           implements    interface
long          switch        throw
throws        try
```

```
>_ ~/workspace/j--
$ bash ./j--/bin/j-- -t project2/tests/ReservedWords.java
1         : break = break
1         : case = case
1         : catch = catch
2         : continue = continue
2         : default = default
2         : do = do
```

```
3         : double = double
3         : final = final
3         : finally = finally
4         : for = for
4         : implements = implements
4         : interface = interface
5         : long = long
5         : switch = switch
5         : throw = throw
6         : throws = throws
6         : try = try
7         : <EOF> = <EOF>
```

Problem 3. (*Operators*) Add support for the following operators. Note that some of the arithmetic, shift, and bitwise operators were added to *j--* in Project 1.

```
?     ~       !=      /       /=
-=    --      *=      %       %=
>>    >>=     >>>     >>>=    >=
<<    <<=     <       ^       ^=
|     |=      ||      &       &=
```

```
>_ ~/workspace/j--
$ bash ./j--/bin/j-- -t project2/tests/Operators.java
1         : ? = ?
1         : ~ = ~
1         : != = !=
1         : / = /
1         : /= = /=
2         : -= = -=
2         : -- = --
2         : *= = *=
2         : % = %
2         : %= = %=
3         : >> = >>
3         : >>= = >>=
3         : >>> = >>>
3         : >>>= = >>>=
3         : >= = >=
4         : << = <<
4         : <<= = <<=
4         : < = <
4         : ^ = ^
4         : ^= = ^=
5         : | = |
5         : |= = |=
5         : || = ||
5         : & = &
5         : &= = &=
6         : <EOF> = <EOF>
```

Problem 4. (*Separators*) Add support for the separator : (colon).

```
>_ ~/workspace/j--
$ bash ./j--/bin/j-- -t project2/tests/Separators.java
1         : ; = ;
2         : : = :
3         : , = ,
4         : . = .
5         : [ = [
5         : { = {
5         : ( = (
5         : ) = )
5         : } = }
5         : ] = ]
6         : <EOF> = <EOF>
```

Problem 5. (*Literals*) Add support for (just decimal for now) long and double literals. Your are *not* allowed to use regular expressions to scan literals.

```
<int_literal> = 0 | (1-9) {0-9} // decimal

<long_literal> = <int_literal> (l | L)

<digits> = (0-9) {0-9}

<exponent> = (e | E) [(+ | -)] <digits>

<suffix> = d | D

<double_literal> = <digits> . [<digits>] [<exponent>] [<suffix>]
                 | . <digits> [<exponent>] [<suffix>]
                 | <digits> <exponent> [<suffix>]
                 | <digits> [<exponent>] <suffix>
```

```
>_ ~/workspace/j--
$ bash ./j--/bin/j-- -t project2/tests/Literals.java
1         : <INT_LITERAL> = 0
1         : <INT_LITERAL> = 2
1         : <INT_LITERAL> = 372
2         : <LONG_LITERAL> = 19961
2         : <LONG_LITERAL> = 777L
2         : <DOUBLE_LITERAL> = .3D
3         : <DOUBLE_LITERAL> = 3.14
3         : <DOUBLE_LITERAL> = 6.022137e+23
3         : <DOUBLE_LITERAL> = 1e-9d
4         : <EOF> = <EOF>
```

Files to Submit

1. `$j/j--/src/jminusminus/TokenInfo.java`
2. `$j/j--/src/jminusminus/Scanner.java`
3. `$j/j--/src/jminusminus/Parser.java`
4. `$j/j--/src/jminusminus/JBinaryExpression.java`
5. `$j/j--/src/jminusminus/JUnaryExpression.java`
6. `$j/j--/project2/report.txt`

Before You Submit

- Make sure you name the classes and files you create exactly as suggested in this writeup. Remember, names are case-sensitive.
- Make sure your report uses the given template, isn't too verbose, doesn't contain lines that exceed 80 characters, and doesn't contain spelling mistakes.

Appendix: Java Syntax

compilationUnit ::= [`package` qualifiedIdentifier ;]
{ `import` qualifiedIdentifier ; }
{ typeDeclaration }
`EOF`

qualifiedIdentifier ::= `<identifier>` { . `<identifier>` }

typeDeclaration ::= typeDeclarationModifiers (classDeclaration | interfaceDeclaration)
| ;

typeDeclarationModifiers ::= { `public` | `protected` | `private` | `static` | `abstract` | `final` }

classDeclaration ::= `class` `<identifier>` [`extends` qualifiedIdentifier]
[`implements` qualifiedIdentifier { , qualifiedIdentifier }]
classBody

interfaceDeclaration ::= `interface` `<identifier>` // can't be `final`
[`extends` qualifiedIdentifier { , qualifiedIdentifier }]
interfaceBody

modifiers ::= { `public` | `protected` | `private` | `static` | `abstract` | `final` }

classBody ::= `{` { ;
| `static` block
| block
| modifiers memberDecl
}
`}`

interfaceBody ::= `{` { ;
| modifiers interfaceMemberDecl
}
`}`

memberDecl ::= `<identifier>` // constructor
formalParameters
[`throws` qualifiedIdentifier { , qualifiedIdentifier }] block
| (`void` | type) `<identifier>` // method
formalParameters
[`throws` qualifiedIdentifier { , qualifiedIdentifier }] (block | ;)
| type variableDeclarators ; // fields

interfaceMemberDecl ::= (`void` | type) `<identifier>` // method
formalParameters
[`throws` qualifiedIdentifier { , qualifiedIdentifier }] ;
| type variableDeclarators ; // fields; must have inits

block ::= `{` { blockStatemnt } `}`

blockStatement ::= localVariableDeclarationStatement
| statement

statement ::= block
| `if` parExpression statement [`else` statement]
| `for` `(` [forInit] ; [expression] ; [forUpdate] `)` statement
| `while` parExpression statement
| `do` statement `while` parExpression ;
| `try` block
{ `catch` `(` formalParameter `)` block }
[`finally` block] // must be present if no catches
| `switch` parExpression `{` { switchBlockStatementGroup } `}`
| `return` [expression] ;
| `throw` expression ;
| `break` [`<identifier>`] ;
| `continue` [`<identifier>`] ;
| ;
| `<identifier>` : statement
| statementExpression ;

formalParameters ::= `(` [formalParameter { , formalParameter }] `)`

formalParameter ::= [`final`] type `<identifier>`

parExpression ::= `(` expression `)`

forInit ::= statementExpression { , statementExpression }
| [`final`] type variableDeclarators

forUpdate ::= statementExpression { , statementExpression }

switchBlockStatementGroup ::= switchLabel { switchLabel } { blockStatement }

switchLabel ::= `case` expression `:` // must be constant
| `default :`

localVariableDeclarationStatement ::= [`final`] type variableDeclarators `;`

variableDeclarators ::= variableDeclarator { , variableDeclarator }

variableDeclarator ::= `<identifier>` [`=` variableInitializer]

variableInitializer ::= arrayInitializer | expression

arrayInitializer ::= `{` [variableInitializer { , variableInitializer }] `}`

arguments ::= `(` [expression { , expression }] `)`

type ::= basicType | referenceType

basicType ::= `boolean` | `byte` | `char` | `short` | `int` | `float` | `long` | `double`

referenceType ::= basicType `[ ]` { `[ ]` }
| qualifiedIdentifier { `[ ]` }

statementExpression ::= expression // but must have side-effect, eg, `i++`

expression ::= assignmentExpression

assignmentExpression ::= conditionalExpression // must be a valid lhs
[
(=
| +=
| -=
| *=
| /=
| %=
| >>=
| >>>=
| <<=
| &=
| |=
| ^=
) assignmentExpression]

conditionalExpression ::= conditionalOrExpression [? assignmentExpression : conditionalExpression]

conditionalOrExpression ::= conditionalAndExpression { || conditionalAndExpression }

conditionalAndExpression ::= inclusiveOrExpression { && inclusiveOrExpression }

inclusiveOrExpression ::= exclusiveOrExpression { | exclusiveOrExpression }

exclusiveOrExpression ::= andExpression { ^ andExpression }

andExpression ::= equalityExpression { & equalityExpression }

equalityExpression ::= relationalExpression { (== | !=) relationalExpression }

relationalExpression ::= shiftExpression ({ (< | > | <= | >=) shiftExpression } | instanceof referenceType)

shiftExpression ::= additiveExpression { (<< | >> | >>>) additiveExpression }

additiveExpression ::= multiplicativeExpression { (+ | -) multiplicativeExpression }

multiplicativeExpression ::= unaryExpression { (* | / | %) unaryExpression }

unaryExpression ::= ++ unaryExpression
| -- unaryExpression
| (+ | -) unaryExpression
| simpleUnaryExpression

simpleUnaryExpression ::= ~ unaryExpression
| ! unaryExpression
| (basicType) unaryExpression // basic cast
| (referenceType) simpleUnaryExpression // reference cast
| postfixExpression

postfixExpression ::= primary { selector } { ++ | -- }

selector ::= . qualifiedIdentifier [arguments]
| `[` expression `]`

primary ::= parExpression
| `this` [arguments]
| `supper` (arguments | . `<identifier>` [arguments])
| literal
| `new` creator
| qualifiedIdentifer [arguments]

creator ::= (basicType | qualifiedIdentifier)
(arguments
| `[ ]` { `[ ]` } [arrayInitializer]
| newArrayDeclarator
)

newArrayDeclarator ::= `[` [expression] `]` { `[` [expression] `]` }

literal ::= `<int_literal>` | `<char_literal>` | `<string_literal>` | `<float_literal>`
| `<long_literal>` | `<double_literal>` | `true` | `false` | `null`