Goal

1. Support multiline comment.
2. Support long and double basic types.
3. Support operators.
4. Support conditional expression and switch statement.
5. Support do-while, for, break, and continue statements.
6. Support exception handlers.
7. Support interface type declaration.

Download the Project Tests

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

In this project you will only modify the JavaCC specification file `$j/j--/src/jminusminus/j--.jj` for *j--* to add more Java tokens and programming constructs to the *j--* language. In the first part, you will modify the scanner section of the `j--.jj` file to support the Java tokens that you handled as part of Project 2 (Scanning). In the second part, you will modify the parser section of the file to support the Java programming constructs that you handled as part of Project 3 (Parsing).

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

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

Part I: Additions to JavaCC Scanner

To scan your *j--* programs using the JavaCC scanner, you need to run the `javaccj--` command as follows:

```
>_ ~/workspace/j--
$ bash ./bin/javaccj-- -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/javaccj-- -t project4/tests/MultiLineComment.java
```

See `project4/tests/MultiLineComment.tokens` for output.

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 ./bin/javaccj-- -t project4/tests/ReservedWords.java
```

See `project4/tests/ReservedWords.tokens` for output.

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

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

```
>_ ~/workspace/j--
$ bash ./bin/javaccj-- -t project4/tests/Operators.java
```

See `project4/tests/Operators.tokens` for output.

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

```
>_ ~/workspace/j--
$ bash ./bin/javaccj-- -t project4/tests/Separators.java
```

See `project4/tests/Separators.tokens` for output.

Problem 5. (*Literals*) Add support for (just decimal for now) long and double 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 ./bin/javaccj-- -t project4/tests/Literals.java
```

See `project4/tests/Literals.tokens` for output.

Part II: Additions to JavaCC Parser

To parse your *j--* programs using the JavaCC parser, you need to run the `javaccj--` command as follows:

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

which will only parse `P.java` and print the AST for the program.

Note

1. Consult appendix at the end for the grammar rule for each new construct you will be supporting in *j--*.
2. The AST output provided for each problem is meant to give you an idea as to what the AST ought to look like once the syntactic constructs for that problem are implemented in *j--*. The autograder will not match your AST against ours for correctness, but instead will test if your parser parses our pass tests without errors.

Problem 6. (*Long and Double Basic Types*) Add support for the `long` and `double` basic types.

```
>_ ~/workspace/j--
$ bash ./bin/javaccj-- -p project4/tests/BasicTypes.java
```

See `project4/tests/BasicTypes.ast` for output.

Problem 7. (*Operators*) Add support for the following operators, obeying precedence rules (see appendix at the end).

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

```
>_ ~/workspace/j--
$ bash ./bin/javaccj-- -p project4/tests/Operators.java
```

See `project4/tests/Operators.ast` for output.

Problem 8. (*Conditional Expression*) Add support for conditional expression (`e1 ? e2 : e3`).

```
>_ ~/workspace/j--
$ bash ./bin/javaccj-- -p project4/tests/ConditionalExpression.java
```

See `project4/tests/ConditionalExpression.ast` for output.

Problem 9. (*Switch Statement*) Add support for a switch statement.

```
>_ ~/workspace/j--
$ bash ./bin/javaccj-- -p project4/tests/SwitchStatement.java
```

See `project4/tests/SwitchStatement.ast` for output.

Problem 10. (*Do-while Statement*) Add support for a do-while statement.

```
>_ ~/workspace/j--
$ bash ./bin/javaccj-- -p project4/tests/DoWhileStatement.java
```

See `project4/tests/DoWhileStatement.ast` for output.

Problem 11. (*For Statement*) Add support for a for statement.

```
>_ ~/workspace/j--
$ bash ./bin/javaccj-- -p project4/tests/ForStatement.java
```

See `project4/tests/ForStatemnet.ast` for output.

Problem 12. (*Break Statement*) Add support for a break statement.

```
>_ ~/workspace/j--
$ bash ./bin/javaccj-- -p project4/tests/BreakStatement.java
```

See `project4/tests/BreakStatemnet.ast` for output.

Problem 13. (*Continue Statement*) Add support for a continue statement.

```
>_ ~/workspace/j--
$ bash ./bin/javaccj-- -p project4/tests/ContinueStatement.java
```

See `project4/tests/ContinueStatemnet.ast` for output.

Problem 14. (*Exception Handlers*) Add support for exception handling, which involves supporting the `try`, `catch`, `finally`, `throw`, and `throws` clauses.

```
>_ ~/workspace/j--
$ bash ./bin/javaccj-- -p project4/tests/ExceptionHandlers.java
```

See `project4/tests/ExceptionHandlers.ast` for output.

Problem 15. (*Interface Type Declaration*) Implement support for interface declaration.

```
>_ ~/workspace/j--
$ bash ./bin/javaccj-- -p project4/tests/Interface.java
```

See `project4/tests/Interface.ast` for output.

Files to Submit

1. `$j/j--/project3/report.txt`
2. `$j/j--/src/jminusminus/j--.jj`
3. `$j/j--/src/jminusminus/JAssignment.java`
4. `$j/j--/src/jminusminus/JBinaryExpression.java`
5. `$j/j--/src/jminusminus/JBooleanBinaryExpression.java`
6. `$j/j--/src/jminusminus/JComparison.java`
7. `$j/j--/src/jminusminus/JBreakStatement.java`
8. `$j/j--/src/jminusminus/JClassDeclaration.java`
9. `$j/j--/src/jminusminus/JConditionalExpression.java`
10. `$j/j--/src/jminusminus/JConstructorDeclaration.java`
11. `$j/j--/src/jminusminus/JContinueStatement.java`
12. `$j/j--/src/jminusminus/JDoWhileStatement.java`
13. `$j/j--/src/jminusminus/JForStatement.java`
14. `$j/j--/src/jminusminus/JInterfaceDeclaration.java`
15. `$j/j--/src/jminusminus/JLiteralDouble.java`
16. `$j/j--/src/jminusminus/JLiteralLong.java`
17. `$j/j--/src/jminusminus/JMethodDeclaration.java`
18. `$j/j--/src/jminusminus/JSwitchStatement.java`
19. `$j/j--/src/jminusminus/JThrowStatement.java`
20. `$j/j--/src/jminusminus/JTryCatchFinallyStatement.java`
21. `$j/j--/src/jminusminus/JUnaryExpression.java`
22. `$j/j--/src/jminusminus/Parser.java`
23. `$j/j--/src/jminusminus/Scanner.java`
24. `$j/j--/src/jminusminus/TokenInfo.java`
25. `$j/j--/src/jminusminus/Type.java`

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