Goal

1. Support long and double basic types.
2. Support operators.
3. Support conditional expression and switch statement.
4. Support do-while, for, break, and continue statements.
5. Support exception handlers.
6. 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 be supporting the parsing of the above programming constructs and their representations in the abstract syntax tree (AST).

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 parse for now) a *j--* program `P.java` using the *j--* compiler.

```
>_ ~/workspace/j--
$ bash ./bin/j-- -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--*. You are expected to implement the `writeToStdOut()` method in the `J*` files for the constructs such that your AST output is something similar. 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 1. (*Long and Double Basic Types*) Add support for the `long` and `double` basic types.
AST representation(s):

- `JLiteralLong.java`
- `JLiteralDouble.java`

```
>_ ~/workspace/j--
$ bash ./bin/j-- -p project3/tests/BasicTypes.java
```

See `project3/tests/BasicTypes.ast` for output.

Problem 2. (*Operators*) Add support for the following operators, obeying precedence rules (see appendix at the end).

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

AST representation(s):

- `JNotEqualOp` in `JComparison.java`
- `JDivAssignOp` in `JAssignment.java`
- `JMinusAssignOp` in `JAssignment.java`
- `JPostIncrementOp` in `JUnaryExpression.java`
- `JPreDecrementOp` in `JUnaryExpression.java`
- `JStarAssignOp` in `JAssignment.java`
- `JRemAssignOp` in `JAssignment.java`
- `JARightShiftAssignOp` in `JAssignment.java`
- `JLRightShiftAssignOp` in `JAssignment.java`
- `JGreaterEqualOp` in `JComparison.java`
- `JALeftShiftAssignOp` in `JAssignment.java`
- `JLessThanOp` in `JComparison.java`
- `JXorAssignOp` in `JAssignment.java`
- `JOrAssignOp` in `JAssignment.java`
- `JLogicalOrOp` in `JBooleanBinaryExpression.java`
- `JAndAssignOp` in `JAssignment.java`

```
>_ ~/workspace/j--
$ bash ./bin/j-- -p project3/tests/Operators.java
```

See `project3/tests/Operators.ast` for output.

Problem 3. (*Conditional Expression*) Add support for conditional expression (`e ? e1 : e2`).

AST representation(s):

- `JConditionalExpression.java`

```
>_ ~/workspace/j--
$ bash ./bin/j-- -p project3/tests/ConditionalExpression.java
```

See `project3/tests/ConditionalExpression.ast` for output.

Problem 4. (*Switch Statement*) Add support for a switch statement.

AST representation(s):

- `JSwitchStatement.java`

```
>_ ~/workspace/j--
$ bash ./bin/j-- -p project3/tests/SwitchStatement.java
```

See `project3/tests/SwitchStatement.ast` for output.

Problem 5. (*Do-while Statement*) Add support for a do-while statement.

AST representation(s):

- `JDoWhileStatement.java`

```
>_ ~/workspace/j--
$ bash ./bin/j-- -p project3/tests/DoWhileStatement.java
```

See `project3/tests/DoWhileStatement.ast` for output.

Problem 6. (*For Statement*) Add support for a for statement.

AST representation(s):

- `JForStatement.java`

```
>_ ~/workspace/j--
$ bash ./bin/j-- -p project3/tests/ForStatement.java
```

See `project3/tests/ForStatement.ast` for output.

Problem 7. (*Break Statement*) Add support for a break statement.

AST representation(s):

- `JBreakStatement.java`

```
>_ ~/workspace/j--
$ bash ./bin/j-- -p project3/tests/BreakStatement.java
```

See `project3/tests/JBreakStatement.ast` for output.

Problem 8. (*Continue Statement*) Add support for a continue statement.

AST representation(s):

- `JContinueStatement.java`

```
>_ ~/workspace/j--
$ bash ./bin/j-- -p project3/tests/ContinueStatement.java
```

See `project3/tests/JContinueStatement.ast` for output.

Problem 9. (*Exception Handlers*) Add support for exception handling, which involves supporting the `try`, `catch`, `finally`, `throw`, and `throws` clauses.

AST representation(s):

- `JTryCatchFinallyStatement.java`
- `JThrowStatement.java`

```
>_ ~/workspace/j--
$ bash ./bin/j-- -p project3/tests/ExceptionHandlers.java
```

See `project3/tests/ExceptionHandlers.ast` for output.

Problem 10. (*Interface Type Declaration*) Implement support for interface declaration.

AST representation(s):

- `JInterfaceDeclaration.java`

```
>_ ~/workspace/j--
$ bash ./bin/j-- -p project3/tests/Interface.java
```

See `project3/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`