Project 4 (Scanning and Parsing with JavaCC)

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.

Grammars

The lexical and syntactic grammars for *j--* and Java can be found at `https://www.cs.umb.edu/j--/grammar.pdf`.

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 project4/XYZ.java
```

which only scans `XYZ.java` and prints the tokens in the program along with the line number where each token appears. The file `project4/XYZ.tokens` provides the reference (ie, expected) output.

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

Directions:

- Using the rules for single line comment as a model, write down rules for scanning a multiline comment.

Problem 2. (*Operators*) Add support for the following operators.

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

Directions:

- List the operators in `j--.jj`.

Problem 3. (*Reserved Words*) Add support for the following reserved words.

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

Directions:

- List the reserved words in `j--.jj`.

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

Directions:

- Using the regular expressions for the currently supported literals as a model, write down regular expressions for scanning long and double literals.

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 project4/XYZ.java
```

which will only parse `XYZ.java` and print the AST for the program. The file `project4/XYZ.ast` provides the reference (ie, expected) output.

Problem 5. (*Long and Double Basic Types*) Add support for the `long` and `double` basic types.

AST representation(s):

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

Directions:

- Modify `j--.jj` to parse longs and doubles.

Problem 6. (*Operators*) Add support for the following operators.

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

AST representation(s):

- `-=`: `JMinusAssignOp` in `JAssignment.java`

- `*=`: `JStarAssignOp` in `JAssignment.java`
- `/=`: `JDivAssignOp` in `JAssignment.java`
- `%=`: `JRemAssignOp` in `JAssignment.java`
- `|=`: `JOrAssignOp` in `JAssignment.java`
- `&=`: `JAndAssignOp` in `JAssignment.java`
- `^=`: `JXorAssignOp` in `JAssignment.java`
- `<<=`: `JALeftShiftAssignOp` in `JAssignment.java`
- `>>=`: `JARightShiftAssignOp` in `JAssignment.java`
- `>>>=`: `JLRightShiftAssignOp` in `JAssignment.java`
- `/`: `JDivideOp` in `JBinaryExpression.java`
- `%`: `JRemainderOp` in `JBinaryExpression.java`
- `|`: `JOrOp` in `JBinaryExpression.java`
- `^`: `JXorOp` in `JBinaryExpression.java`
- `&`: `JAndOp` in `JBinaryExpression.java`
- `<<`: `JALeftShiftOp` in `JBinaryExpression.java`
- `>>`: `JARightShiftOp` in `JBinaryExpression.java`
- `>>>`: `JLRightShiftOp` in `JBinaryExpression.java`
- `||`: `JLogicalOrOp` in `JBooleanBinaryExpression.java`
- `!=`: `JNotEqualOp` in `JBooleanBinaryExpression.java`
- `>=`: `JGreaterEqualOp` in `JComparison.java`
- `<`: `JLessThanOp` in `JComparison.java`
- `~`: `JComplementOp` in `JUnaryExpression.java`
- `++`: `JPostIncrementOp` in `JUnaryExpression.java`
- `--`: `JPreDecrementOp` in `JUnaryExpression.java`
- `+`: `JUnaryPlusOp` in `JUnaryExpression.java`

Directions:

- Modify `j--.jj` to parse the operators, correctly capturing the precedence rules by parsing the operators in the right places.
- Update `statementExpression()` in `j--.jj` to include post-increment and pre-decrement expressions.

Problem 7. (*Conditional Expression*) Add support for conditional expression (`e ? e1 : e2`).

AST representation(s):

- `JConditionalExpression.java`

Directions:

- Modify `j--.jj` to parse a conditional expression.

Problem 8. (*Do Statement*) Add support for a do statement.

AST representation(s):

- `JDoStatement.java`

Directions:

- Modify `j--.jj` to parse a do statement.

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

AST representation(s):

- `JForStatement.java`

Directions:

- Modify `j--.jj` to parse a for statement.
- If `forInit()` is looking at a statement expression, then it must return a list of statement expressions. Otherwise, it must return a list containing a single `JVariableDeclaration` object encapsulating the variable declarators.

Problem 10. (*Break Statement*) Add support for a break statement.

AST representation(s):

- `JBreakStatement.java`

Directions:

- Modify `j--.jj` to parse a break statement.

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

AST representation(s):

- `JContinueStatement.java`

Directions:

- Modify `j--.jj` to parse a continue statement.

Problem 12. (*Switch Statement*) Add support for a switch statement.

AST representation(s):

- `JSwitchStatement.java`

Directions:

- Modify `j--.jj` to parse a switch statement. After parsing `SWITCH parExpression LCURLY`, parse zero or more occurrences of a `switchBlockStatementGroup`, and then scan an `RCURLY`.
- In `switchBlockStatementGroup()`, after parsing one or more occurrences of `switchLabel`, parse zero or more occurrences of a `blockStatement`.

Problem 13. (*Exception Handlers*) Add support for exception handling, which involves supporting the `try`, `catch`, `finally`, `throw`, and `throws` clauses. Note that there has to be a `finally` clause if there are not `catch` clauses.

AST representation(s):

- `JTryStatement.java`
- `JThrowStatement.java`

Directions:

- Modify `j--.jj` to parse a try statement, a throw statement, and the throws clause in constructor and method declarations.

Problem 14. (*Interface Type Declaration*) Implement support for interface declaration.

AST representation(s):

- `JInterfaceDeclaration.java`

Directions:

- Modify `j--.jj` to parse an interface declaration and the implements clause in class declaration.

Before you submit your files, make sure:

- Your code is adequately commented and follows good programming principles.
- You use the template file `report.txt` for your report.
- Your report meets the prescribed guidelines.

Files to submit:

1. `j--.jj`
2. `TokenInfo.java`
3. `Scanner.java`
4. `Parser.java`
5. `JBinaryExpression.java`
6. `JUnaryExpression.java`
7. `report.txt`