Goal

1. Support long and double basic types.
2. Support operators.
3. Support conditional expression and switch statement.
4. Support do, for, break, and continue statements.
5. Support exception handlers.
6. 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 be supporting the parsing of the above programming constructs.

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 `XYZ.java` using the *j--* compiler.

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

which will only parse `XYZ.java` and print the AST for the program. The file `project3/XYZ.ast` provides the reference (ie, expected) output.

Problem 1. (*Long and Double Basic Types*) Add support for the `long` and `double` basic types.

AST representation(s):

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

Directions:

- Modify `Parser.java` to parse longs and doubles.

Problem 2. (*Operators*) Add support for the following operators. Note that parsing support for some of the operators was added to *j--* in Project 1.

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

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

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

AST representation(s):

- `JConditionalExpression.java`

Directions:

- Modify `Parser.java` to parse a conditional expression, correctly capturing the precedence rules by parsing the expression in the right place.

Problem 4. (*Do Statement*) Add support for a do statement.

AST representation(s):

- `JDoStatement.java`

Directions:

- Modify `Parser.java` to parse a do statement.

Problem 5. (*For Statement*) Add support for a for statement.

AST representation(s):

- `JForStatement.java`

Directions:

- Modify `Parser.java` to parse a for statement.
- If `forInit()` is not looking at a local variable declaration, 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 6. (*Break Statement*) Add support for a break statement.

AST representation(s):

- `JBreakStatement.java`

Directions:

- Modify `Parser.java` to parse a break statement.

Problem 7. (*Continue Statement*) Add support for a continue statement.

AST representation(s):

- `JContinueStatement.java`

Directions:

- Modify `Parser.java` to parse a continue statement.

Problem 8. (*Switch Statement*) Add support for a switch statement.

AST representation(s):

- `JSwitchStatement.java`

Directions:

- Modify `Parser.java` to parse a switch statement. After parsing `SWITCH parExpression LCURLY`, parse a `switchBlockStatementGroup` until you see an `RCURLY` or `EOF`. Then scan an `RCURLY`.
- In `switchBlockStatementGroup()`, after parsing one or more occurrences of `switchLabel`, parse a `blockStatement` until you see a `CASE`, `DEFLT`, or `RCURLY`

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

AST representation(s):

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

Directions:

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

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

AST representation(s):

- `JInterfaceDeclaration.java`

Directions:

- Modify `Parser.java` 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. `TokenInfo.java`
2. `Scanner.java`
3. `Parser.java`
4. `JBinaryExpression.java`
5. `JUnaryExpression.java`
6. `report.txt`