


Goal

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

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 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 test program, say `project2/XYZ.java` using the `j--` compiler.

```
>_ ~/workspace/j--  
$ bash ./bin/j-- -t project2/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 `project2/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:

- Draw a state transition diagram for recognizing multiline comments
- Use the diagram to implement code in `Scanner.java` for scanning multiline comments

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

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

Directions:

- Define tokens for the operators in `TokenInfo.java`
- Modify `Scanner.java` to recognize the operators

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

Project 2 (Scanning)

<code>break</code>	<code>case</code>	<code>catch</code>	<code>continue</code>	<code>default</code>	<code>do</code>
<code>double</code>	<code>finally</code>	<code>for</code>	<code>implements</code>	<code>interface</code>	<code>long</code>
<code>switch</code>	<code>throw</code>	<code>throws</code>	<code>try</code>		

Directions:

- Define tokens for the reserved words in `TokenInfo.java`
- Modify `Scanner.java` to recognize the reserved words

Problem 4. (*Literals*) Add support for long and double literals (just decimal). You are *not* allowed to use regular expressions to scan literals.

Directions:

- Define tokens for long and double literals in `TokenInfo.java`
- Extend the state transition diagram for recognizing integer literals to also recognize long and double literals
- Use the diagram to implement code in `Scanner.java` for scanning long and double literals

Files to submit:

1. `TokenInfo.java`
2. `Scanner.java`
3. `Parser.java`
4. `JBinaryExpression.java`
5. `JUnaryExpression.java`
6. `notes.txt`

Before you submit your files, make sure:

- Your code is adequately commented and follows good programming principles.
- You update the `notes.txt` file.