Assignment 0

- Academic integrity is taken seriously in this course, which has a zero tolerance policy for cheating. All discovered incidents of cheating will be reported and result in an E in the course.
- You ARE allowed to:
  - discuss high level solutions with other students and
  - ask questions in the class slack channel and during office hours
  - use code presented in the lectures.
- You are NOT allowed to
  - Share your code with anyone else (except your official project partner).
  - Receive code from others (except your official project partner), including students in previous semesters.
- All work submitted should be the work of the submitting student(s), created this semester for this assignment.
- Submissions by groups must have approximately equal contribution from both members.

Specification

Your task is to implement a Lexer for the programming language with the given lexical structure. Your code should implement a lexer as described in class or in the textbook. (Do NOT use the java.util.Scanner class)

Provided Code

- Interfaces:
  - ILexer.java interface for lexer. Your solution should implement this interface
  - IToken.java interface for tokens. Your solution should implement this interface (or use the provided one)
- Factory: ComponentFactory.java
  - Factory class to provide instantiations of your Lexer. It will later be extended to handle additional components of the compiler.
- Exceptions:
  - PLCException.java superclass of all Exceptions thrown during compilation due to illegal source.
  - LexicalException.java exception to throw if Lexer detects illegal input.
- Sample or starter implementations:
  - Token.java sample implementation. You should be able to use this unchanged. This was provided because of the hurricane.
  - Lexer.java minimal implementation to run tests.

- Tests:
    - LexerTest.java Junit test case with some useful methods and examples illustrating how to write test with and without expected errors. This class is NOT complete. Your submission should pass all these tests, but passing these tests does not guarantee that your implementation is correct and complete.

Github

Using git while developing your project is required—you are required to submit your git history. The contents will not be graded, but failure to turn it in may result in a zero on the assignment. These files have proved to be helpful in cases of academic dishonesty or problems with partners. It is highly recommended that you also use a cloud repository such as github for backup purposes and coordinating with your partner. Unfortunate events do occur: laptops are stolen, hard disks crash, etc. and you are responsible for maintaining backups of your work. Make sure that your repository is private.

To turn in

A jar file containing the source code (i.e .java files) of all classes necessary to test your scanner with the appropriate directory structure. Do NOT include class file. Be aware that your IDE may provide a way to generate jar files that defaults to including .class files instead of .java files. Double check the contents of your jar file before submission.
A zip file containing your git repository WITH history. To obtain this file, go to the parent of your local git repository and create a zip file. Your zip file should contain a directory called .git (dot git). Do NOT use the "download zip file" option on github--it does not include the history.

Additional requirements

- For the purposes of this project, we will assume that the only combinations that terminate lines are '\r''\n' and '\n'. This means that '\r' can be treated as any other white space and only '\n' need be considered when terminating a comment or incrementing the line number.
- In order to match typical editors, line and column numbers should start counting at 1.
- Repeated calls to next() after the end of the input has been reached should return an EOF token.
- If errors are found, your Lexer should throw a LexicalException.
- NUM_LIT tokens must have an integer value less than 2147483647 (i.e. will fit in the number of bits available to a java int variable) and your Lexer should check this. The

easiest way to check this is to use Integer.parseInt. If the literal is out of range, this will throw a NumberFormatException that you can catch and rethrow as a LexicalException.

Hints

- Decide, with your partner, on an effective git workflow and use it conscientiously.
- Study the lexical structure and determine its DFA. Systematically implement the DFA as discussed in class.
- Work incrementally, testing as you go.
- Avoid static variables as they do not always behave as expected during unit testing.
- You may use
- Before turning in:
    - Double check that you have complied with all the instructions above.
    - Remove debug printing
    - Check for and remove extraneous import statements. Unless otherwise specifically indicated, only classes from the standard java distribution should be used. Nonstandard import statements may cause your code to fail to compile on our system, which would result in a zero on the project.
    - Check the contents of your jar file. It should contain source files only with the appropriate directory structure. The easiest way to do this is from the command line. Here is the provided jar file. Yours should have a similar structure.

```
$ jar -xvf *.jar
 inflated: edu/ufl/cise/cop4020fa23/SourceLocation.java
 inflated: edu/ufl/cise/cop4020fa23/Kind.java
 inflated: edu/ufl/cise/cop4020fa23/ILexer.java
 inflated: edu/ufl/cise/cop4020fa23/Token.java
 inflated: edu/ufl/cise/cop4020fa23/exceptions/LexicalException.java
 inflated: edu/ufl/cise/cop4020fa23/exceptions/PLCCompilerException.java
 inflated: edu/ufl/cise/cop4020fa23/LexerTest.java
 inflated: edu/ufl/cise/cop4020fa23/Lexer.java
 inflated: edu/ufl/cise/cop4020fa23/ComponentFactory.java
 inflated: edu/ufl/cise/cop4020fa23/IToken.java
```