ICSI 311 Assignment 2 – Enhance the Lexer

This assignment is extremely important – (nearly) every assignment after this one uses this one!

If you have bugs or missing features in this, you will need to fix them before you can continue on to new assignments. This is very typical in software development outside of school.

You must submit .java files. Any other file type will be ignored. <u>Especially</u> ".class" files.

You must not zip or otherwise compress your assignment. Brightspace will allow you to submit multiple files.

You must submit every file for every assignment.

<u>You must submit buildable .java files for credit.</u>

Introduction

In this assignment, we will complete our lexer (yes, already!).

As you know, in programming languages (like Java), there are known words (like "for" and "while") and words that we can't forsee (like "myVariableName" and "lineNumber").

One concept that we will use extensively in this assignment is HashMap. Remember that these are O(1) lookups that have we can use as a lookup table. We will use this to look for keywords and convert them to Token Types. A short reference is here: https://www.geeksforgeeks.org/java-util-hashmap-in-java-with-examples/

We will use hash maps to keep track of the known "words" and symbols and hold the token type for them.

For example – if we were lexing Java, we might have a token enum "if". We could build something like this:

```
if (word == "if")
      tokens.add(new Token(TokenType.IF, lineNumber, characterPosition);
if (word == "else")
      tokens.add(new Token(TokenType.ELSE, lineNumber, characterPosition);
      /* … */
else
      tokens.add(new Token(TokenType.WORD, word, lineNumber,
characterPosition);
```

But this would take a LOT of code. Instead, we will build a hashmap:
HashMap<String, TokenType> knownWords;

knownWords.put("if", TokenType.IF); // add all of the rest…

Then you can simply say:

```
if (knownWords.containsKey(word))
      tokens.add(new Token(knownWords.get(word), lineNumber,
characterPosition);
else
      tokens.add(new Token(TokenType.WORD, word, lineNumber,
characterPosition);
```

Details

Make a HashMap of <String, TokenType> in your Lex class. Below is a list of the keywords that you need. Make token types and populate the hash map in your constructor (I would make a helper method that the constructor calls). Make sure that you don't create this hashmap more than once.

Print, Read, Input, Data,   Gosub, For,To,Step,Next, Return, If,Then, Function, While, End

Modify "ProcessWord" so that it checks the hash map for known words and makes a token specific to the word with no value if the word is in the hash map, but WORD otherwise.

For example,

Input: for while hello do

Output: FOR WHILE WORD(hello) DO

You are familiar with string literals in Java (String foo = "hello world";) BASIC has them as well. Make a token type for string literals. In Lex, when you encounter a ", call a new method (I called it HandleStringLiteral()) that reads up through the matching " and creates a string literal token (STRINGLITERAL(hello world)). Be careful of two things: make sure that an empty string literal ("") works and make sure to deal with escaped " (String quote = "She said, \"hello there\" and then she left.";)

The last thing that we need to deal with in our lexer is symbols. Most of these will be familiar from Java, but a few I will detail a bit more. We will be using two different hash maps – one for two-character symbols (like ==, &&, ++) and one for one character symbols (like +, -, $). Why? Well, some two-character symbols start with characters that are also symbols (for example, + and +=). We need to prioritize the += and only match + if it is not a +=.

Two-character symbols:

<=.   >=.   <> (NOTEQUALS)

Next create the token types and the hash maps for the single character symbols (I used String, not char):
= (token type **must** be EQUALS)   <.   >   (LPAREN   ) RPAREN

+ - * /

If the lexer encounters a word followed by :, it should NOT be an identifier, but a “LABEL”.

Rubric	Poor	OK	Good	Great
Code Style	Few comments, bad names (0)	Some good naming, some necessary comments (3)	Mostly good naming, most necessary comments (6)	Good naming, non-trivial methods well commented, static only when necessary, private members (10)
Unit Tests	Don’t exist (0)	At least one (6)	Missing tests (12)	All functionality tested (20)
Keywords	Unmodified (0)	Processed ad hoc (3)	Hashmap created and populated (6)	Keywords properly recognized and proper tokens created (15)
Hashmaps Created once	Hashmaps created per multiple times (0)			Hashmaps are created once (5)
StringLiteral	Not processed (0)	Token type exists (3)	Token type exists and Method exists and is called (6)	String literals properly processed into their own tokens; line count and column updated (10)
Accepts labels	Does not allow labels (0)	Doesn’t properly tokenize labels (5)		Properly tokenizes labels (10)
Symbol – One Character	No hashmap (0)	Hash map exists (3)	Most symbols exist (6)	All symbols exist with reasonable token types (10)
Symbol – Two Character	No hashmap (0)	Hash map exists (3)	Most symbols exist (6)	All symbols exist with reasonable token types (10)
ProcessSymbol	None (0)	Exists, looks in one map (3)	Exists, looks in both maps (6)	Exists, looks in twoChar, then oneChar, updates CodeManager and position correctly (10)