COP5556 Assignment 1

Implement a scanner for a programming language with the following lexical structure:

InputCharacter ::= any 7-bit ASCII character
LineTerminator ::= LF | CR | CR LF
LF is the ASCII character also known as "newline". The Java character literal is '\n'.
CR is the ASII character also known as "return", The Java character literal is '\r'.
CR immediately followed by LF counts as one line terminator, not two.

Input ::= (WhiteSpace | Comment | Token)*

WhiteSpace ::= SP | HT | FF | LineTerminator
SP is the ASCII character also known as "space". The Java char literal is ' '.
HT is the ASCII character also known as "horizontal tab". The Java char literal is '\t'.
FF is the ASCII character also known as "form feed". The Java char literal is '\f'.

Comment ::= -- (NOT (LineTerminator)) * LineTerminator
Comments start with two adjacent hypens '-'.
~~Token ::= Identifier | Keyword | Literal | Separator | Operator~~
Token ::= Name | Keyword | Literal | OtherToken
Name ::= IdentifierChars but not a Keyword
IdentifierChars ::= IdentiferStart IdentifierPart*
IdentifierStart ::= A..Z | a..z
IdentifierPart ::= IdentifierStart | Digit | _ | $
Literal ::= IntegerLiteral | StringLiteral
StringLiteral ::= " StringCharacter* " | ' StringCharacter* '
StringCharacter ::= InputCharacter but not " or \ or ' | EscapeSequence
EscapeSequence ::= \a (bell)
| \b (backspace)
| \f (form feed)
| \n (newline)
| \r (carriage return)
| \t (horizontal tab)
| \v (vertical tab)
| \\ (backslash)
| \" (quotation mark [double quote])
| \' (apostrophe [single quote]).
IntegerLiteral ::= 0 | NonZeroDigit Digit*

```
NonZeroDigit ::= 1 .. 9
Digit ::= NonZeroDigit | 0
OtherToken ::=    +  |  -  |  *  |  /  | %  |  ^  | #
   | &  | ~  | |  | <<  | >>  | //
   | ==  | ~=  | <=  |  >=  | <  |  >  |  =
   | (  | )  | {  | }  |  [  | ]  |  ::
   | ;  |  :  | ,  |  .  |  ..  |  ...
Keyword ::=    and  |  break  |  do  |  else  |  elseif  |  end
   | false  |  for  |  function  | goto  |  if  |  in
   | local  | nil  |  not  |  or  |  repeat  | return
   | then  |  true  |  until  | while
```

- If an illegal character is encountered, your scanner should throw a LexicalException. The message should contain useful information about the error. The contents of the message will not be graded, but you will appreciate it later if it is descriptive.
- If a numeric literal is provided that is out of the range of the Java equivalent of that type, then your scanner should throw a Lexical exception.
- The contents of the error message will not be graded, but you will appreciate it later if it is descriptive.
- Use the provided Scanner.java and ScannerTest.java as starting points.

Turn in a jar file containing the source code Scanner.java , Token.java, and ScannerTest.java. Turn in a zip file with your git repository containing the history of your development.

You must create the zip file from a local clone of your repository. The file created with the github zipfile option does not include the history.

Note that the default jar file exported from eclipse does not include sources by default. You will need to check a box to ensure they are included.

Your ScannerTest and github repo will not be graded, but may be looked at in case of academic honesty issues. Failure to turn in conforming files may result in a zero on the assignment.

We will subject your scanner to our set of junit tests and your grade will be determined solely by how many tests are passed. Name your jar file in the following format:
firstname_lastname_ufid_hw1.jar

Additional requirements:

- This code must remain in package cop5556sp19 (case sensitive): do not create additional packages.
- Names (of classes, method, variables, etc.) in starter code must not be changed. You may, of course, add additional variables, methods, enums, change or add the bodies of methods, etc.
- Your code should not import any classes other than those from the standard Java distribution.

Comments and suggestions:

- As given, Scanner.java and ScannerTest.java should compile correctly. When executed, only three tests will pass, but all should pass in your completed scanner. You will need to add additional JUnit tests as you go along.
- **Work incrementally: add a single capability along with a junit test to exercise it.**
- Plan your approach. Pay attention to things that are basically the same--for example, if you can handle a semi-colon you can handle a comma, and all other characters that only appear by themselves in Tokens the same way.
- The scanner will be part of all the subsequent assignments. Errors may cause failures in subsequent assignments. A careful job now, including a complete test suite developed now will help you later.

Submission Checklist

- **Make sure that sources are included in the jar file**. Many IDEs (including Eclipse) do not do this by default.
    - A quick reference for how to export a jar file from Eclipse
    - If you are not using Eclipse, you can use the command:
        - *jar -cMf packagename.jar <list_of_target_folders_and_files>*
        - example: *jar - cMf fname_lname_12345678_hw1.jar cop5556fa19 testInputFiles*
    - For more details check Creating a JAR file
    - **DO NOT package any other external jar package(s) in your submission jar file. e.g. JUnit, ASM should be excluded from your submission jar package.**
    - **DO NOT PUT an extra directory level above your assignment java package, i.e. *cop5556fa19 .git testFiles* should be at the top level inside the jar package**
- Versions of Softwares we are using for Grading:

- **JDK/JRE 11.x.x,** technically you can use any language features in **Java 11 and prior**. But if you are using any new features in **Java 12 or above**, your code will fail to compile.
- **JUnit 5.5.x**: We are using JUnit 5.5.x for grading. Because your own test cases will not be graded, so it does not matter which JUnit version you are using to write your own code. But using the same or later version of JUnit to test your own code (will be detailed later) will guarantee your submission works good with our grading code.

- To ensure that we will be able to **compile and run** your submission:
  - If you are using the same version of the aforementioned packages, and following the instructions in this doc, you can do the verification on your **local machine**.
  - If you want to test on the same environment we are using for grading, you can upload your jar file(homework) to one of the UF CISE servers (e.g. storm.cise.ufl.edu), you should also, decompress it, and run it from the command line. (See https://www.cise.ufl.edu/help/access/remote for information about remote access to CISE machines) Instructions:
    - Copy/upload your file to a CISE server. Suppose your CISE ID is *username*, the following instruction will upload the *HW1.jar* to your home folder on cise server:
      ```
      scp /my/path/to/HW1.jar username@storm.cise.ufl.edu:~/
      ```
    - **S**ome CISE servers have only Java 8 installed, if your code uses language features from a later version, it may fail to compile. You can switch to one having the version you want, or test on your local machine.
  - Decompress file:
    ```
    jar xf fname_lname_12345678_hw1.jar
    ```
    - If you packaged everything correctly, your decompressed project directory structure will look like the following:
      ```
      ├── cop5556fa19
      │   ├── package-info.java
      │   ├── Scanner.java
      │   ├── ScannerTest.java
      │   ├── Token.java
      │   └── All the other files(if necessary)
      └── testInputFiles
          └── test2.input
      ```
  - Prerequisite for configuring the same setting as the grading environment:
    - You need 3 jar packages:
      - `junit-jupiter-api-5.5.x.jar`
      - `apiguardian-api-1.x.x.jar`
      - `junit-platform-console-standalone-1.5.x.jar`
    - You can download the related version of jar packages from Maven/Gradle or Google search. For your convenience, we created an **all-in-one** zip package including **all three jar files**. You can get it through this link.
    - Download/extract and put all three jar files to an appropriate directory on your local disk

    - Create a folder for output class files:
      `mkdir out`
    - Compile (for windows users, please replace **forward slash** to **backslash**, and replace colon **":"** semicolon **";",** enclose **ALL** paths with a pair of double quotes):
      `javac -d out -cp ".:<path-to-jar>/junit-jupiter-api-5.5.1.jar:<path-to-jar>/apiguardian-api-1.1.0.jar" cop5556fa19/ScannerTest.java`
    - Run junit test from command line:
      `java -jar "<path-to-jar>/junit-platform-console-standalone-1.5.1.jar" -cp ./out --scan-classpath --details=tree`
- **Make sure that your jar file has the same directory structure as the original one that you downloaded from Canvas. If it doesn't, the grading script will fail resulting in a grade of 0.**
- **Turn in a zip file with your git repository containing the history of your development. To be more specific, if you start your code with a git repository and commit gradually, there will be an *.git* folder under your root directory.**
- Note that you can try this upload before you are finished with the assignment, giving you time to figure out what is wrong if you have problems.
- **No matter how your program runs on your own machine, if it fails to compile/run on the CISE server (storm or thunder) with the aforementioned instructions, your submission will get a zero grade, and there will be no regrading. So double check before your submission.**
- If you are non-CISE student who does not have a CISE account, see https://www.cise.ufl.edu/help/account for instructions to get one.
- If you use Eclipse, we suggest creating a project and importing the jar files (eg. HW1.jar) provided by each assignment into the project. After completing your work on the source files (keep all source files within the package cop5556fa19), you can export the package cop5556fa19 as a jar file for submission (remember to select the option of including source files in the jar package), so that it will have the same directory structure as the original jar file.
- You can submit multiple times and we will only grade your latest submission. The system will append a suffix to the file name for resubmissions—this is OK. And only the latest one will be graded.
- **Important!!! It is YOUR RESPONSIBILITY to submit the assignment on time, where time is determined by Canvas. If you wait until the last minute, it is possible that your submission will fail due to server overload. Start early and submit early.**
- Double check your Canvas submissions has uploaded properly by navigating away from the submission page, then returning and downloading your submission. In order to receive consideration for any Canvas problems, you must submit a report to the UF help desk and forward your trouble ticket to the TA.