Compilation coursework

Overview

Your job is to implement a predictive recursive descent parser for the formal language L_0 generated by the following context-free grammar G_0 (expressed in Backus-Naur form):

```
<start>  ::= <start> "+" <term>   | <term>   | <start> "-" <term>
<term>   ::= <term>  "*" <factor> | <factor>
<factor> ::= "(" <start> ")"      | <float>
```

where the `<float>` non-terminal should match any floating-point number.
You are expected to use the techniques shown in the lectures, which apply to LL(1) grammars. Therefore, you should start by checking whether G_0 is LL(1). You will find it is not, and hence need to find (**task0**) an LL(1) grammar equivalent to G_0: that, is, also generating L_0.
The main job (**task1**) of your parser is to act as a recogniser: that is, abort exactly on the input strings which are not in the language.
Additionally (**task2**), it should provide useful error messages about why an input string is not in the language.
Finally (**task3**), if the input string is in L_0, it should evaluate it, meaning that it should evaluate the corresponding arithmetic expression, and print the result on the screen.
For example, if the input string is "`( 4 - ( 3 * 2 ) )`", your program should not abort (since the string is in L_0), and should print "`-2.0`".
Note that task3 is always possible because all the strings in L_0 are well-formed arithmetic expressions.

Coding details

You are provided with four .java files. You should only modify `Parser.java`.
You are provided with a tokenizer in `Lexer.java`. It is rudimental, so please make sure your input string is **on one line** and has **all tokens separated** by at least one space.
The tokenizer adds an end-of-file (EOF) token, `$`, at the end of the input, which will make your life easier when implementing.
It reads the string to be parsed from standard input. Therefore, the command to compile and run your code will be something along the lines of:

```
javac *.java && echo "( 4 - ( 3 * 2 ) )" | java Driver
```

or, if your input is in a txt file:

```
javac *.java && cat input.txt | java Driver
```

where on windows you might need to replace `cat` with `type`. Also, you can omit the `&&` and what comes before if you do not wish to recompile your java code.
You should test your code on as many inputs, both valid and invalid, as you can.

Submission

Your code implementing tasks 1 to 3 should be contained in `Parser.java`.
For task0, write the LL(1) grammar you chose to implement **in the comments** of `Parser.java`.
Submit `Parser.java`, along with the other .java unedited files, as one .zip file without any directory in it.
Submit only after making sure your code appropriately works on the lab Linux machines or on the "SCC Lab" virtual machine on mylab.

Marking criteria

You will get marks for each of the tasks. Therefore, it is suggested that you start from task0, then focus on recognizing the input string (task1) without elaborated error messages or evaluating it. Only after you have this baseline, you should start adding error diagnostic messages for task2. They should contain non-terminal causing the error, its position in the token stream and, if the error occurs when recognising a terminal, which terminal was expected and which one was given by the lexer instead.
For task3, you will need to think how to modify the code for tasks1 and task2.