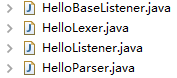
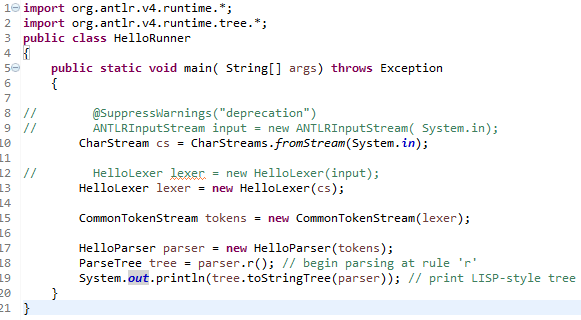
We have the Antlr Grammar which could be identified by Antlr4.7 and we will apply it to Antlr4.7 in Eclipse Neon so that it can automatically generate the parse-tree.

First, Antlr4.7 will identify the .g4 file so that Eclipse will compile it and generate .tokens and .java files.



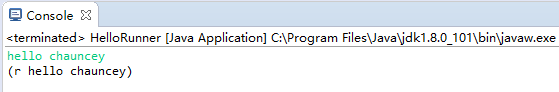
Then, we are going to create a java program to get all those components work.



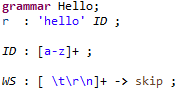
As we all know, if the Grammar is wrong, it will not be compiled. However, in case the grammar is not properly defined but still can be compiled, we test it with some input to make sure that it is generating the expected parse-tree.

So the final step is to test the grammar. Since we will do the Antlr4.7-parsing using Eclipse Neon, we can directly give the input and output parse-tree in the console.

For example:



(r hello chauncey) is the output, the parse-tree. We are going to check if it properly matches the grammar we defined.



“r” means the rule. The rule constrains that the input should start with “hello”, followed by “ID” which means the identifiers that should be a set of lower-case alphabet. So, in this case, the whole process is executed successfully.

In our project, we will translate the whole EBNF grammar into Antlr grammar and test it in the approach mentioned above repeatedly to make sure everything works.