Phase 1 – SPLAT Lexer

Target Date: Friday, October 20

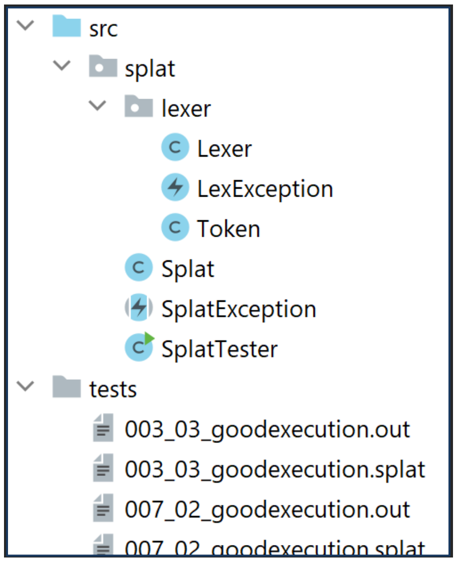
A note about project submissions and deadlines…

Although the final project has been divided up into four separate phases, there will only be one place in Moodle for you to submit your project code. For each phase, we will give you a target date to help keep you on track, and you are strongly encouraged to (re-)submit your project code after you have successfully completed each phase. However, your grade for the project will be determined by the last submission that you make up until the final deadline. Thus, you can make fixes to your code for any project phase at any time up to the final deadline for the entire project.

For each submission, please zip up your **src** folder, and submit it to the proper place in Moodle. For testing purposes, it is critical that you do not change the package structure or the names of the provided Java class files, as such changes may prevent compiling your project code.

PHASE 1 SETUP

For this phase, you need to create the lexer component for SPLAT, which breaks a SPLAT program down into individual tokens, and returns the result as a List of Tokens. But first, we need to set up a new project that we will be using for the remainder of the semester. To begin, create a new Java project in your IDE, and name it something like "SPLAT_Project". Next, unzip the lex_start.zip file, and move the contents in the **src** folder there into the **src** folder in your new project, and copy the whole **tests** folder into the project folder. Things should look something like this in your project folder:



```
src
  splat
    lexer
      Lexer
      LexException
      Token
    Splat
    SplatException
    SplatTester
tests
  003_03_goodexecution.out
  003_03_goodexecution.splat
  007_02_goodexecution.out
  007_02_goodexecution.splat
```

All .splat testing files should be placed in the **tests** directory for them to be accessed by the SplatTester class. Several test files are provided for testing each phase of the project, and you can add some of your own as well (See "TESTING FILES" for more information about how testing works.) *Note: If you find an error in one of the testing files, contact the instructors as soon as you can – the first person who finds an error in any given test file with get extra credit!*

SPLAT programs are written in plain ACSII text, following the syntax and semantic rules outlined in the provided specification documents (grammar and semantics). If you need to review how to understand context-free grammar notation, there is a brief overview here. Exactly one program should be provided in one file. There are no libraries or multi-file programs in SPLAT. SPLAT program file names should end with the extension .splat (not .txt, even though they are written as plain text.)

TASK DESCRIPTION

Here is a brief description of the roles each of the .java files play in our SPLAT project, and what you need to do to complete Phase 1.

- **Splat.java** – This is the main, high-level component that we are building for the final project. Actually, all of the code you will probably need for this class is already provided, though some of it has been commented out since we will not start working on the other phases until later. The constructor takes a file, which should contain a SPLAT program that will be processed and executed when the method processFileAndExecute() is called. We see inside of that method the four primary phases of building and running a SPLAT program, and how their implementing components relate to one another.

- **SplatException.java** – All errors that our SPLAT components detect should result in the throwing of an exception, whose specific type will depend on which of the four components detected it. The four specific exception types will be:

  - **LexException**
  - **ParseException**
  - **SemanticAnalysisException**
  - **ExecutionException**

  all of which are child classes of the parent class **SplatException**. All four of the child exception types inherit attributes to store the **column** and **line** numbers to indicate where the error occurred in the code, along with a **message** which should give a brief description of the error. The code for this class is given, and does not need to be changed.

- **Lexer.java** – Most of your work for this phase will focus on implementing this component—specifically the tokenize() method, which should produce a List of Token objects upon successful completion. You will want to add a field to the class which holds onto the File object for the program file to be tokenized; don't forget to set this field in the constructor. You should probably use an approach similar to the one discussed in the video lesson exercises where we read in the file character by character, since we will need to track both the line and column numbers for each of the Tokens we process. You should also carefully read the discussion on SPLAT token separation from the Lexing.pptx slides for some of the trickier situations. If there is an error during tokenization, a **LexException** should be thrown, which should indicate which character caused the problem, along with the location of the problem (line and column number).

- **SplatTester.java** – The code for this testing class is provided for you—you shouldn't have to make any changes to it. The main method in this class essentially goes through the .splat test files in the **tests** folder, and constructs and runs a Splat object on each of these SPLAT programs. A test case is counted as successful if:
    - There is a particular type of error in the code of the .splat file, and the Splat object detects it and throws the proper type of SplatException, or
    - The .splat file contains an error-free SPLAT program, and the output produced by the Splat object matches the expected output.

  While it is good practice to go through the code here and try to understand what it is doing, it isn't critical at this point that you understand everything here (especially if you are relatively new to Java.)

- **LexException.java** – The specific type of exception that should be thrown by the Lexer when something goes wrong there. Nothing needs to be added or changed with this class.
- **Token.java** – This class will be used to store the lexeme **value** (a String), and the **line** and **column** numbers for the first character of a token read in from the program file. You will need to add most of the code here, but it should be quite easy. Just create the necessary three fields, a constructor that sets all three, and "getter" methods for each of the three. You should probably also create a toString() method here that outputs all of the field information—this will be useful for testing purposes.

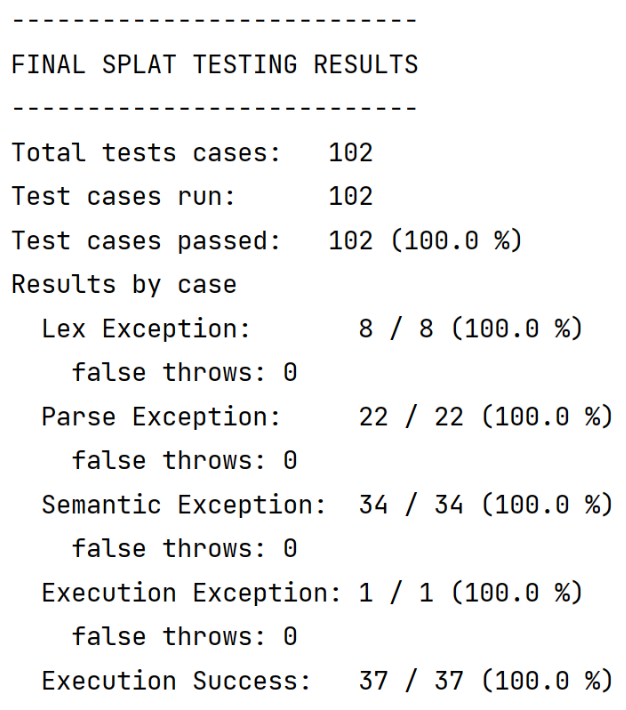
MORE ON TESTING

We have provided you with 102 .splat files for testing your project throughout all four phases of development. Note that the SplatTester class uses the filenames of the .splat files to determine the expected outcome of attempting to run your Splat implementation on it. Specially, if the filename ends with:

- `_badlex.splat`, it contains a lex error,
- `_badparse.splat`, it should lex successfully, but contains a parse error,
- `_badsemantics.splat`, it should parse successfully, but contains a semantic/typechecking error,
- `_badexecution.splat`, it should pass the semantic checks, but contains an execution error,
- `_goodexecution.splat`, the program should run successfully.

Furthermore, plaintext output files corresponding to the proper output produced by each of the successful execution test cases are provided. These output files use the exact same base file name, but end with the suffix `.out`. For example, if you have a test case `bt1_goodexecution.splat`, there is a corresponding output file `bt1_goodexecution.out` that is used by the tester.

After you initially set up your project for Phase 1, you can actually run the SplatTester then, as everything should compile (though most of your Lexer code is empty, and the other three SPLAT components are still missing). The test results should look something like this:



```
---------------------------
FINAL SPLAT TESTING RESULTS
---------------------------
Total tests cases:   102
Test cases run:      102
Test cases passed:   102 (100.0 %)
Results by case
  Lex Exception:       8 / 8 (100.0 %)
    false throws: 0
  Parse Exception:     22 / 22 (100.0 %)
    false throws: 0
  Semantic Exception:  34 / 34 (100.0 %)
    false throws: 0
  Execution Exception: 1 / 1 (100.0 %)
    false throws: 0
  Execution Success:   37 / 37 (100.0 %)
```

Since the processFileAndExecute method in Splat.java only runs the (mostly empty) Lexer component, it essentially does nothing. Note that this still results in two passed tests… these are SPLAT programs that run, but don’t produce any output!