Assignment 3: Unit Testing With JUnit

Overview

This assignment will give you the opportunity to write your own unit tests using JUnit, which is a tool supported in Eclipse. You will be writing unit tests for a number of methods in the TokenStream class based on these methods' specifications. Your goal is to write relevant tests with as much coverage as possible.

Passing in a String Reader

In previous assignments, the TokenStream object in the main function of GenerateTokenStream was created by passing a new FileReader object to TokenStream's constructor. The source code found in the specified file would then be read character by character using the relevant methods in the TokenStream class. Unfortunately, while FileReaders are useful when the JavaScript code to parse is long, they can be a bit of a pain when the code to parse is very short because of the additional burden of having to create a new file each time. This is especially true when writing unit tests.

When writing unit tests, it may be more helpful to pass a StringReader to the TokenStream constructor rather than a FileReader. In this case, instead of specifying a file containing the JavaScript code to parse, you will specify a string representing the JavaScript code itself. For instance, consider the following JavaScript code:

var a = 5;

To parse this JavaScript line using TokenStream, you can create a file containing this line, save the file, and pass the file via a FileReader to TokenStream. Assuming this file's name is SampleCode.js, the call to the constructor would look as follows:

```
TokenStream ts = new TokenStream(new FileReader("SampleCode.js"), "",
1);
```

However, you could have also passed in a StringReader to spare you from the extra burden of having to create a file for such a short line of code:

```
TokenStream ts = new TokenStream(new StringReader("var a = 5;"), "",
1);
```

For this assignment, you are free to use either StringReaders or FileReaders (or a mixture of both) when writing your unit tests. However, if you're using FileReaders, please make sure you hand in the relevant JavaScript files.

Instructions

1. Read the tutorial from Lab 3, which teaches you how to write and how to run unit tests using the JUnit Eclipse plugin.
2. Download the file eece310_assn3.zip from the website. Unzip the file and add it to Eclipse as a project.
3. Create a separate source folder in the eece310_assn3 project and call it “tests”. This source folder will contain your unit test files. (see Lab 3 for instructions).
4. Using the JML specifications, write unit tests for the following methods in TokenStream.java (for each test you create, make sure you specify if it is a black box test or a glass box test):

- TokenStream() (i.e., the constructor)
- stringToKeyword()
- getToken()
- stringToNumber()
- isJSSpace()
- getString()
- getNumber()
- isNumberOctal()
- eof()

5. Once you’ve written your tests, try running them with JUnit (see Lab 3 for instructions on how to do this).

Deliverables

Please submit a **zipped** version of the eece310_assn3 project. Your project must include **all** the .java files that were originally there (these files should be unmodified), the unit tests you’ve written, and all relevant .js files containing JavaScript code (if any). **Important Note**: If you use a FileReader for a unit test but fail to submit the relevant .js file(s), that particular unit test will not be marked and therefore will not be counted as a test. So please make sure the .js files are submitted.

1. zip the folder eece310_assn3 (it will contain all the existing files and your new test files within the test directory)
2. name that zipped folder as your student numbers (Eg: 31067199_3452190.zip)

*Y*ou should give the zipped folder the following name:

<StudentNumber1>_<StudentNumber2>.zip (**eg: 21779990_23001345.zip**)
Please submit the zip file as mentioned in the email guidelines.

Evaluation

You will be marked based on the relevance/correctness of your unit tests and the coverage of your unit tests based on the specifications. The breakdown is as follows:

Constructor unit tests - 18%
stringToKeyword() unit tests - 18%
getToken() unit tests - 18%
stringToNumber() unit tests - 18%
isJSSpace() unit tests - 5%
getString() unit tests - 5%
getNumber() unit tests - 5%
isNumberOctal() unit tests - 5%
eof() unit tests - 5%
Coding style – 3%

Side note: If you find a bug in the code and can prove, using your unit tests, that the bug you found is indeed a bug, you may be awarded a maximum of 3 bonus marks.

As a side note for the coding style, please include comments in your code stating what your test function is doing (also, don't forget to specify your test input and expected output).