Project #3 Solution Description

Nicholas Mills

2/21/18

UMUC

CMSC 350

Ioan Salomie

**1. Assumptions, main design decisions and error handling**

Assumptions

* Spaces separate tokens.
* Input solely deals with integers and fractions. The program does not support floating point numbers.
* There should be no extra elements in the input other than the digits and slashes for fractions. Any anomalous elements will result in an exception being thrown.
* The input will be no more than 30 characters, and the output will be no more than 20. The program should still calculate properly, but the field will be too small to show the full values.

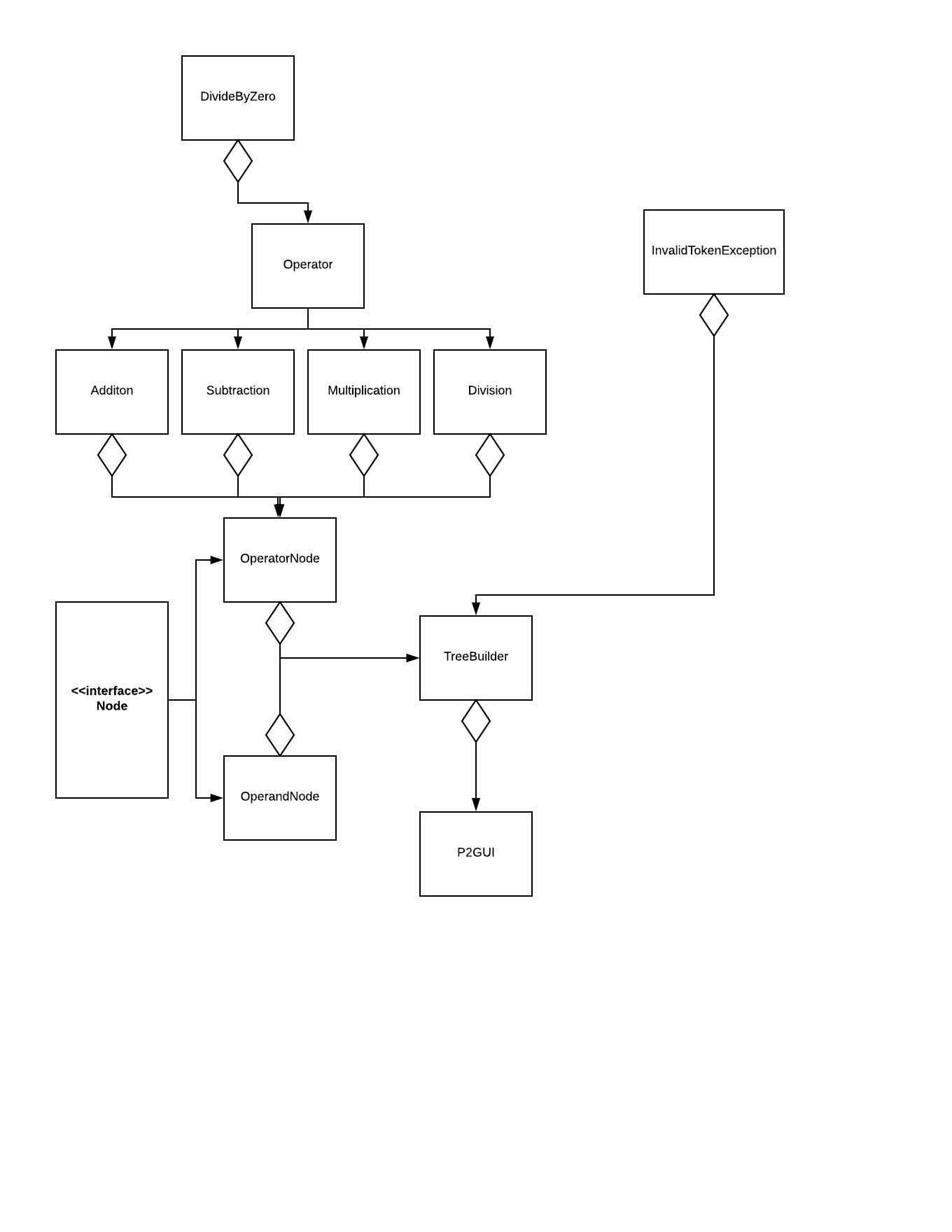
Main design decisions

* For tokenizing the string, I used a Scanner, since it uses space delimiters by default.
* For the GUI, I used a BoxLayout combined with 6 different panels to organize the different elements. In order to get the radio buttons to appear as they did in the prompt, I used a combination of a GridLayout and TitledBorders created in a BorderFactory.
* The most complex and time-consuming part of the project was attempting to parse the tokens without having to repeat large amounts of code. To accomplish this, I retrieved the String-based constructors for Integers and the Fraction class using the Constructor class and passed them to the TreeBuilder. Since these constructors function similarly, creating a numeric object based on a passed string and throwing an exception if said string does not meet the proper format, I was able to cut down on the amount of code used in token parsing considerably. This required me to have a couple of unchecked casts in my code, but since I am guaranteed that the input will be of the proper type through exception handling and minimal options, I decided that it was worth the warnings.

Error handling

* I created a DivideByZero exception that is thrown when calculate is instructed to divide any number by 0. If this happens, then a dialog box pops up informing the user of the issue. The current code should never actually encounter this, but since the calculate method exists in the Division class, I figured this should be included.
* I created an InvalidTokenException that is thrown when anomalous symbols are in the input, such as letters or unsupported operators, such as %. It will also be thrown in the event of a missing space between operators. This is determined whenever a token that does not belong is encountered by the TreeBuilder constructor. This shows the user a dialog box informing them of the issue.
* If the stack operators and operands don’t match, an EmptyStackException ends up being thrown. P2GUI catches this and displays a dialog box informing the user.
* The BufferedWriter that writes the file throws an IOException if something goes wrong with writing the file. This is caught by P2GUI and informs the user.

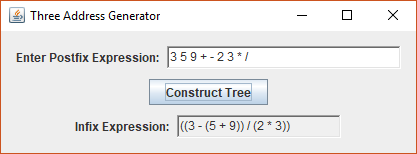
**2. UML class diagrams**

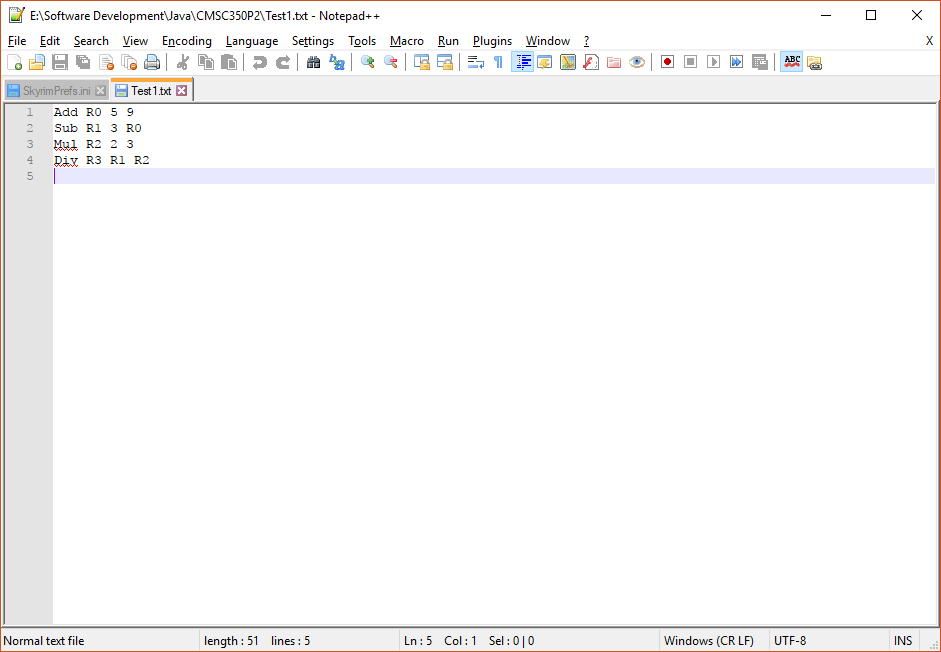
****

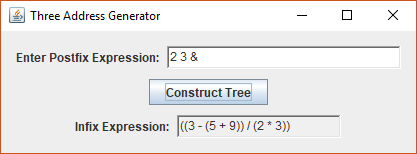
**3. Test cases**

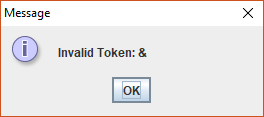
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **What aspect is tested** | **Input** | **Expected Output** | **Actual Output** | **Pass / Fail** |
| Basic Function | 3 5 9 + - 2 3 \* / | ((3 – (5 + 9)) / (2 \* 3))  Test1.txt | ((3 – (5 + 9)) / (2 \* 3))  Test1.txt | **P** |
| InvalidTokenException | 2 3 & | Invalid Token: & | Invalid Token: & | **P** |
| Space Delimiters | 3 5 9 +- 2 3 \* / | Invalid Token: +- | Invalid Token: +- | **P** |
| Invalid Token Beginning with Digit | 3\* | Invalid Token: 3\* | Invalid Token: 3\* | **P** |
| Miswritten Postfix Expression | 3 + 4 | Please enter a valid postfix statement. | Please enter a valid postfix statement | **P** |

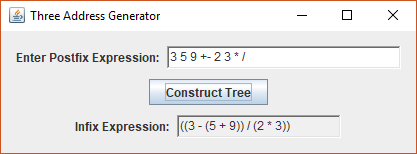
**4. Screenshots**

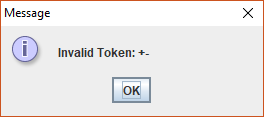


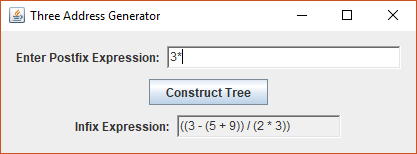


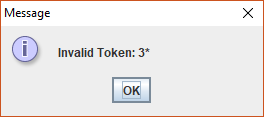


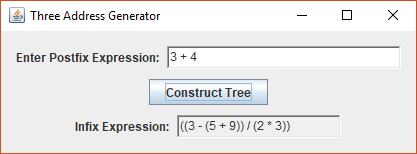














**5. Lessons learned**

Sometimes the prompt does not include all of the tools one can use to create the solution. With the way that I solved this problem, I needed to add more methods to the classes outlined in the course materials. I initially was trying to solve the problems that I had without modifying those classes, since I assumed those were complete. I realized later that I was likely restricting myself needlessly and the solution started flowing much more easily once I opened myself up to more possibilities.

It is more important that something works than that it is elegant. I spent a lot of time trying to think of a solution to this problem that looks better than the one that I ended up with; time that I didn’t have. Function over form sometimes has to be the goal in order to get something done on a time crunch. That being said, give yourself enough time to complete the task at hand. Though I was busy this past week, I could still have managed my time better. Going forward, I need to put more thought into doing that.