Shafro Batyrov **Project 1** **9/16/2018**

The program works by having the user run the program, then enter the name of the text file they are inputting to be parsed which will then display the supposed program.

Examples of successful execution of source code in examples of test cases:

Case 1 uses the input file: test1.txt

This case tests the provided example in the project instructions. I worked with this input file the most and based my design off of it. After it worked successfully it was a just a matter of figuring out how to nest Panels and then the rest of the test cases were working like a charm.

Here is the **input**:

```
Window "Calculator" (200, 200) Layout
 Flow: Textfield 20;
 Panel Layout Grid(4, 3, 5, 5):
 Button "7";
 Button "8";
 Button "9";
 Button "4";
 Button "5";
 Button "6";
 Button "1";
 Button "2";
 Button "3";
 Label "";
 Button "0";
 End;
End.
```

Here is an example of this test case executing successfully.


Calculator
7
8
9
4
5
6
1
2
3
0

Case 2 uses the input file: errortest.txt

This case tests for syntax errors in the input file. My code design included removing all punctuation so it just looked for the needed keywords. If any keyword in the input file did not match one of the keywords provided in the grammar from the project instructions, an error message is thrown and the program exits. This input file has '123' added to one of the Radio keywords which causes that token to take on a String that never matches the required keywords. This is what causes the error message to pop up.

Here is the **input**:

```
Window "ErrorTest" (650, 300) Layout
 Flow: Textfield 20;
 Textfield 40;
 Textfield 60;
 Group
 Radio "Radio1";
 Radio123 "Radio2";
 Radio "Radio3";
 End;
 Panel Layout Grid(4, 3, 5, 5):
 Button "7";
 Button "8";
 Button "9";
 Button "4";
 Button "5";
 Button "6";
 Button "1";
 Button "2";
 Button "3";
 Label "";
 Button "0";
 End;
End.
```

Here is the screenshot of the pop up message when an error exists in the input file:

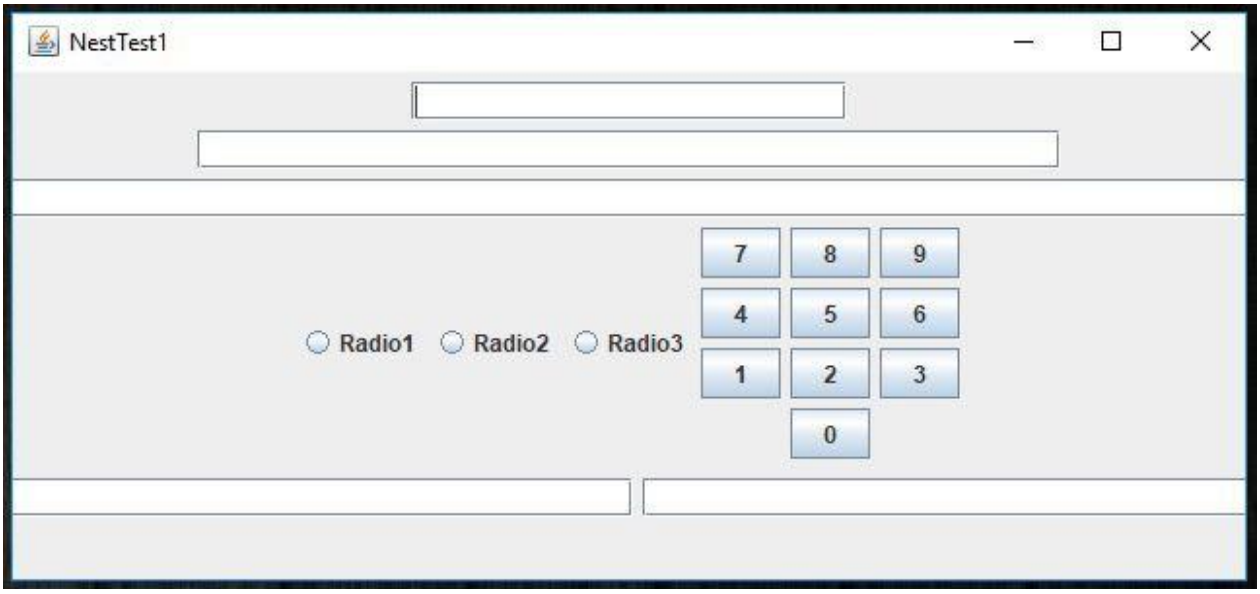
Case 3 uses the input file: nesttest1.txt

This casetests for a few things by editing the original test1.txt. A production is added before a nested panel as well as testing for if one nested panel will work.

Here is the input:

```
Window "NestTest1" (650, 300) Layout
 Flow: Textfield 20;
 Textfield 40;
 Textfield 60;
 Group
 Radio "Radio1";
 Radio "Radio2";
 Radio "Radio3";
 End;
 Panel Layout Grid(4, 3, 5, 5):
 Button "7";
 Button "8";
 Button "9";
 Button "4";
 Button "5";
 Button "6";
 Button "1";
 Button "2";
 Button "3";
 Label "";
 Button "0";
  Panel Layout Flow:
  Textfield 20;
  Textfield 40;
  Textfield 60;
  End;
End;
End.
```

Below is a screenshot of the program running successfully:



The group that was added is the group of three radio buttons. The only issue I couldn't figure out was why the nested Panel did not take on the required Flow layout. This should have caused the three TextField's that were added to look the same as the first three that were added to the initial JFrame. While debugging, I did see that this Panel had a Flow layout assigned to it, but still it wouldn't work. It was the one thing I couldn't quite get right, although it kind of worked in the next test case.

Case 4 uses the input file: nesttest2.txt

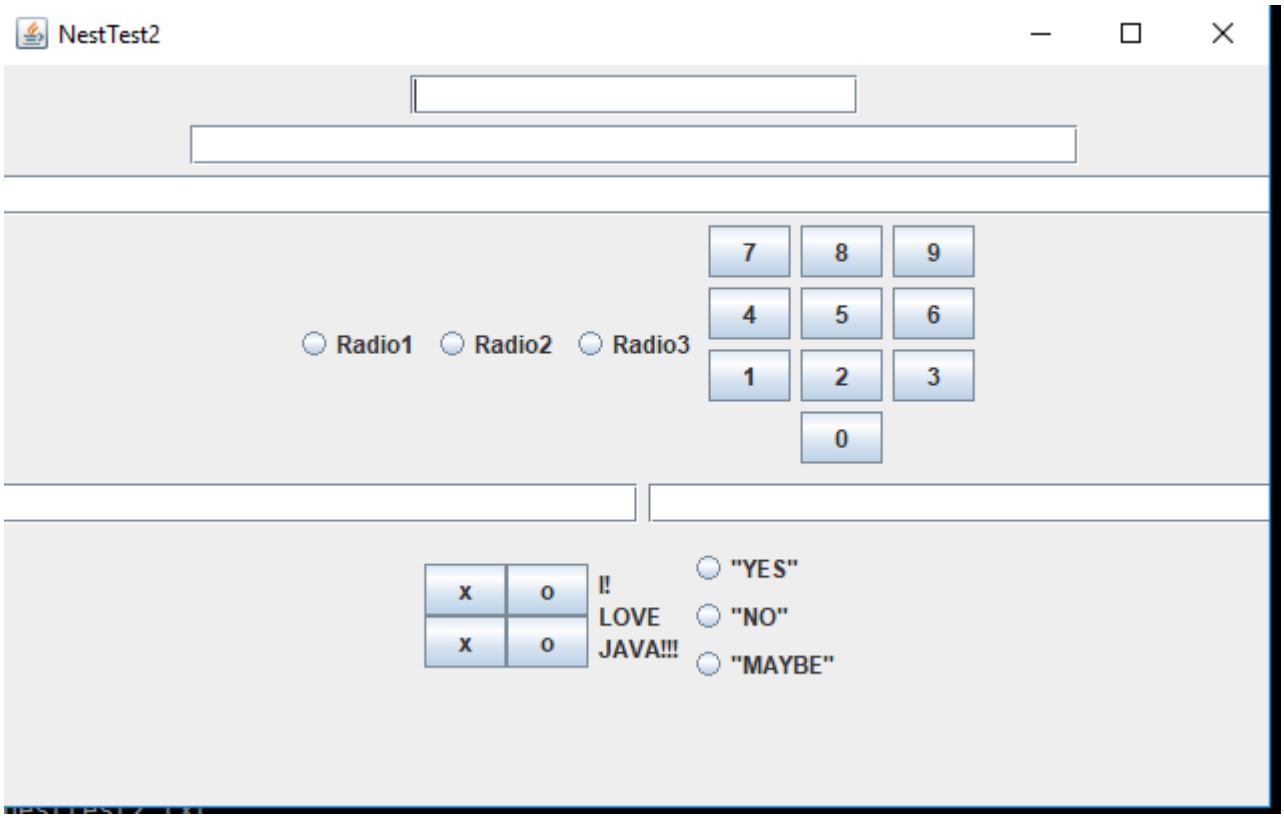
This case tests another file similar to case 3, but wants to see if further nested panels will add on with no errors, which they do. The interesting thing about all Panel's that are nested again are each one does take on the assigned Layout as they are supposed to, but the initial Panel still does not take on the Flow layout it was supposed to. In hindsight maybe I should have added another loop that added an empty Panel as the first nested Panel, since all the ones following acted like they should.

Here is the **input**:

```
Window
"NestTest2" (650,
300) Layout Flow:
 Textfield 20;
 Textfield 40;
 Textfield 60;
 Group
 Radio "Radio1";
 Radio "Radio2";
 Radio "Radio3";
 End;
 Panel Layout
Grid(4, 3, 5, 5):
 Button "7";
 Button "8";
 Button "9";
 Button "4";
 Button "5";
 Button "6";
 Button "1";
 Button "2";
 Button "3";
 Label "";
 Button "0";
  Panel Layout
Flow:
  Textfield 20;
  Textfield 40;
  Textfield 60;
    Panel Layout
Grid(2, 2):
       Button "x";
```

```
        Button "o";
        Button "x";
        Button "o";
          Panel Layout
  Grid(3, 1):
        Label "HOW
  OLD"
        Label "ARE
  YOU"
        Label
  "NOW?"
          Panel
  Layout Grid(3, 1);
                Group
                Radio
  "I AM"
                Radio
  "50 YEARS"
                Radio
  "OLD NOW"
        End;
       End;
    End;
   End;
  End.
```

Here is an example of the final test case:



All the panels reacted as they should and took on the correct Grid layouts that were provided. If the window is adjusted in size, they stack on each other like they should because of the Flow layout that was initially added to this Panel.

Lessons Learned:

As I worked my way through this assignment, many lessons were learned starting with a better understanding of just how useful Backus Naur form is. I stared at the provided grammar in the instructions for a very long time and finally looked up a few videos of BNF being written out based on a simple grammar provided. I then applied this to the grammar provided us which made an easier to read road map for how the many if/else and while statements would look. Using this, I first revisited how to get input from a text file, then started playing with that input. I first used the String[] split method, but then later used the Tokenizer class as I have used it before and find it to be very useful when dealing with input text from a text file like this. I thought about loading them into a stack or heap, but felt the Tokenizer would work just as well which it did. I never called a recursive method because I did not make each production into its own method, as I could have done, but instead just looped through and caught keywords in such a way that it naturally nested Panels like it was supposed to. After writing the program like I did, I noticed many different ways I could have done it, but it would have included starting kind of from scratch, which I really didn't want to do, due to time constraints. In the end the program was compiling successfully as well as giving the correct output with correctly formatted input, as well as showing a pop up message when the input file had a syntax error. I understand much better now the importance, as well as the sheer magnitude of work it must taking in writing the language as well as the parser when creating a new programming language. It is pretty cool to have some understanding now as to what is going on under the hood of the compiler.