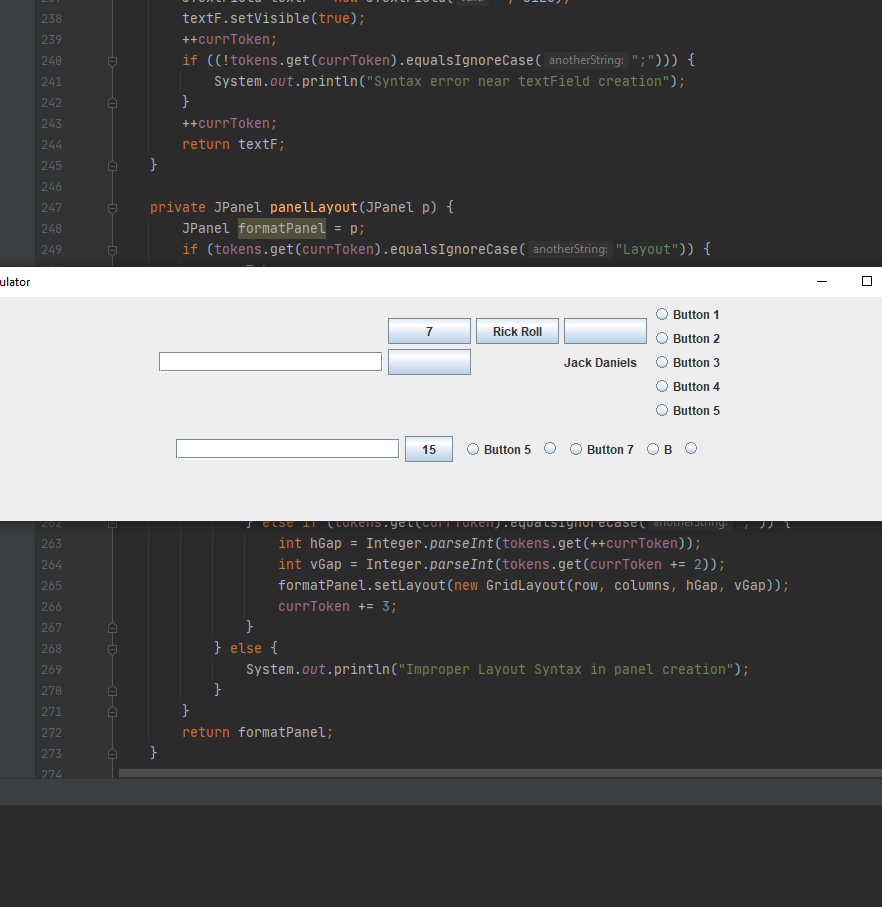
# Process:

I started by examining the provided language, got an understanding of how things were supposed to work together. Then I started with the base of the language, creating a main frame/window, that all else goes into. I used regex to split the input file into what I called tokens, even though after separation not each thing that I added to an array list is a token.

I increment the arraylist with a currToken variable to move down the list and reach each token. Every time one of the widgets names are seen, the widgets method calls the corresponding object to be created. That object doe its own incrementing as to find all the info and set up for the next object to be found and created.

The panel object is different because it recursively calls widgets until End is found. This allows widgets or other panels to be added to the panel. Group was a bit difficult, because groupbuttons are not able to be added as a component. So, I created a special case for when the program runs into group, that calls radiowidget until the end is found.

# Test case 1(no nesting all test):

Input 1: Pass, see note below image

NOTE: Window must be resized to see all components. I tried repainting and revalidating everywhere to no avail. Seems to only effect flow layout panels.

Window "Calculator" (1000, 500) Layout Flow:

Textfield 20;

Panel Layout Grid(3, 8, 5, 5):

Button "7";

Button "Rick Roll";

Button "";

Button " ";

Label "";

Label "Jack Daniels";

Label " ";

End;

Panel Layout Grid(5, 1):

Group

Radio "Button 1";

Radio "Button 2";

Radio "Button 3";

Radio "Button 4";

Radio "Button 5";

End;

End;

Panel Layout Flow:

Textfield 20;

Button "15";

Panel Layout Flow:

Group

Radio "Button 5";

Radio " ";

Radio "Button 7";

Radio "B";

Radio "";

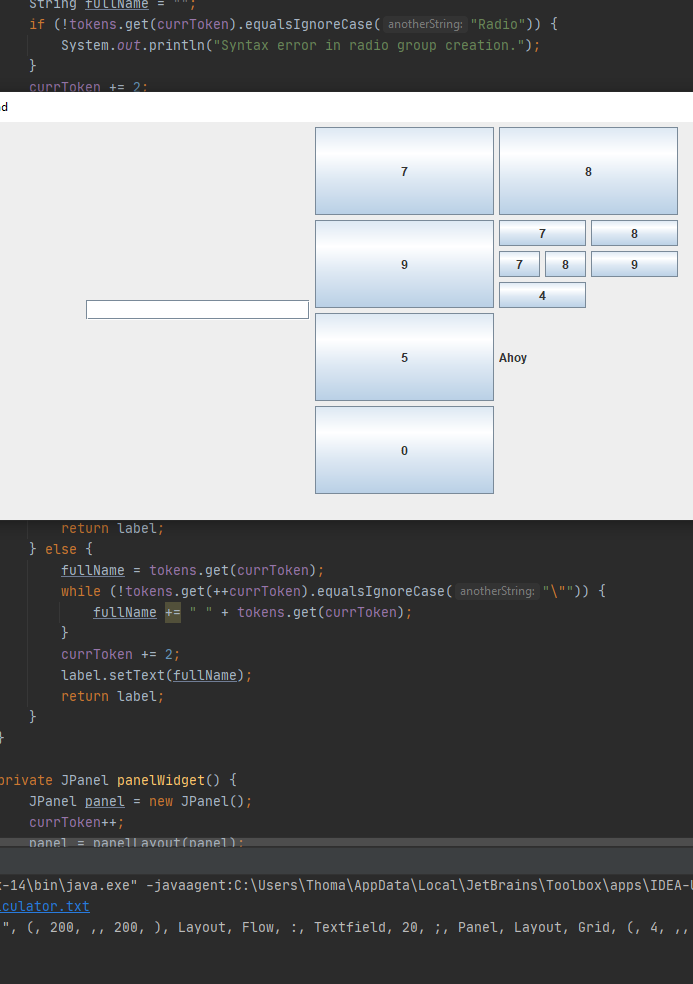
End;

End;

End;

End.

# Test Case 2(nesting panels):

Input 2: Pass all elements show without resize

Window "Nested send" (900, 800) Layout Flow:

Textfield 20;

Panel Layout Grid(4, 4, 5, 5):

Button "7";

Button "8";

Button "9";

Panel Layout Grid(3, 3, 5, 5):

Button "7";

Button "8";

Panel Layout Grid(1, 2, 5, 5):

Button "7";

Button "8";

End;

Button "9";

Button "4";

End;

Button "5";

Label "Ahoy";

Button "0";

End;

End.

# Test Case 3(given example):

Input 3: Pass

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";

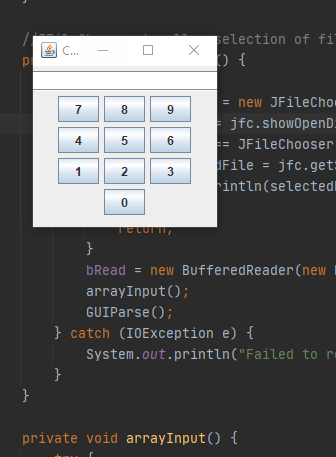
Button "3";

Label "";

Button "0";

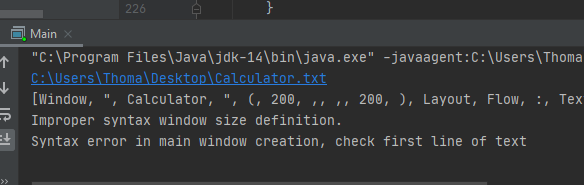
End;

End.



# Test Case 4(error handling):

poor



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.

NOTE: I think that the way I did the project made error checking difficult. At the beginning I could use try catch for an error in window size definition, because I had to parse int here, but as for the rest of the program I could not think of a case to throw an error. The issue is if say I check if the current token is what it is supposed to be in buttonWidget, I have no sure fire check that proves it is not what it is supposed to be. However, I can check something like radioWidget, because when I enter it from group, the token needs to be radio.

# Lessons Learned:

* Error handling – I was considering error handling and checking while completing this project, but I did not recognize the fact that the way I have things set up, there is no true way I can see to check for any error that it could face, at least that I can think of.
* Recursion is neat in this case, if I am not mistaken the panels here are recursive due to panel widget calling a method that can call panel widget. If I am correct that it is, it seemed very natural in this case. It seems to make sense that something that can be nested inside itself until a condition is met, to be recursive.
* When I was looking on how to use regex to separate input, I saw many ways to use regex, but seems that splitting everything on delimiters using lookbehind and lookahead was not the right choice. There is probably a mor intuitive way to go about it. In hindsight it probably would have been much easier to just split on words and keep the colons, semicolons and quotation marks with the words to be dealt with in the methods. That spacing in names would not affect where I would find a semicolon to check syntax.