

Week 4: Project 1

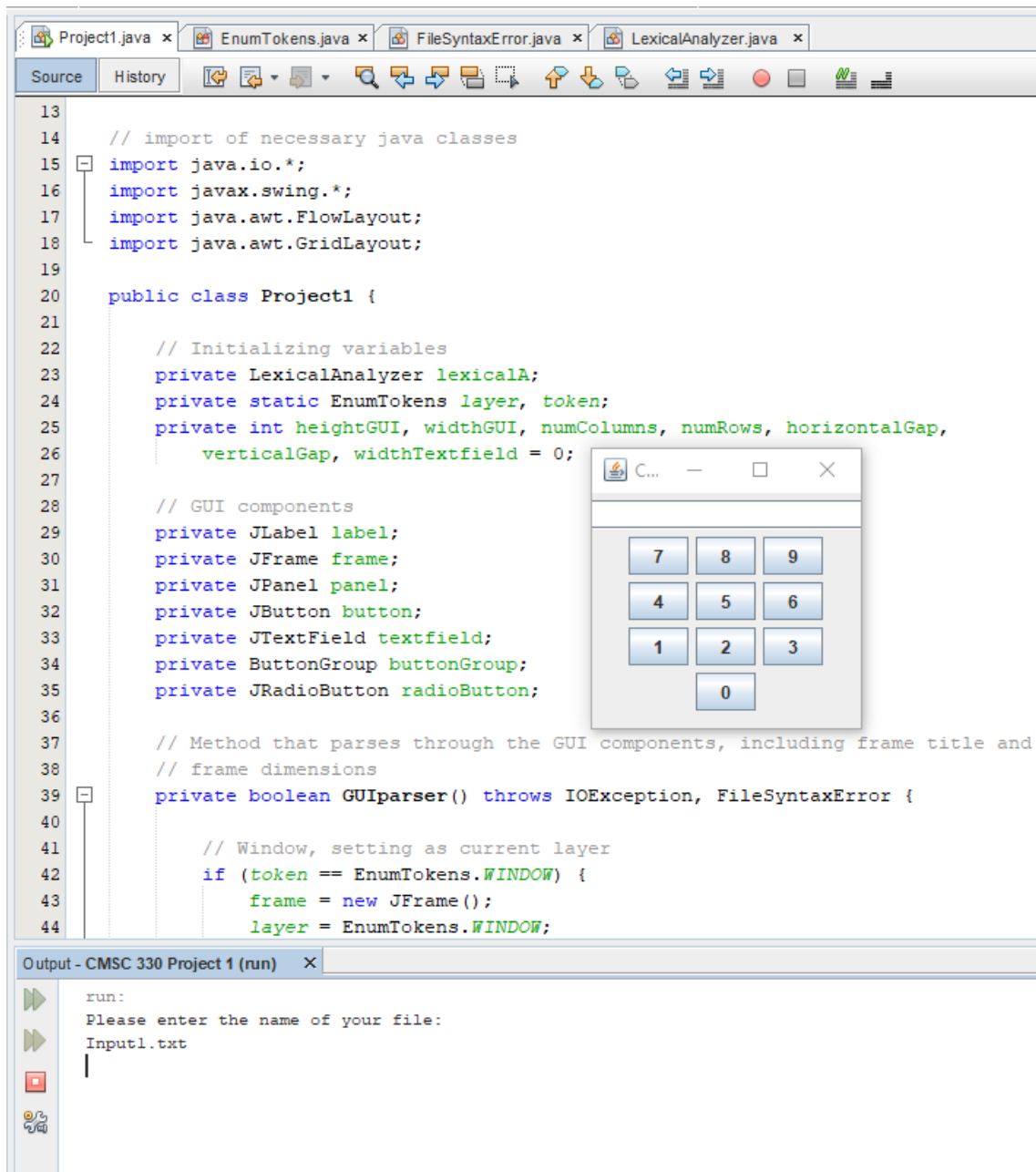
John Kucera

Prof. Tim Boyd

CMSC 330 Advanced Programming Languages

9 April 2020

Week 4: Project 1 Solution Description

Parser and GUI Generator for a GUI Definition Language**Screenshot of successful program compilation, execution, and GUI generation in Netbeans:**

Week 4: Project 1

Description of Process:

There are 4 source code files I created here: Project1.java, EnumTokens.java, LexicalAnalyzer.java, and FileSyntaxError.java. Project1.java contains the main method and parsing methods. EnumTokens.java contains a list of enumerated types which match the types of tokens that should be found in the input file – for example, COMMA, FLOW, GRID, BUTTON, and L_PAREN. LexicalAnalyzer.java contains methods that tokenize the input file's contents and has a getNextToken() method to return an according Enum that matches the token (to be returned to the parsing methods in Project1.java). Finally, FileSyntaxError.java is a custom exception constructor class that handles situations where there is a syntax error in the GUI definition language. Any syntax error detected has its location printed in the output for the user to see.

When the user presses run, the main method (contained in Project1.java) prompts the user to enter the name of the file they want to parse and generate a GUI for. Once the user enters the file name, the main method constructs a LexicalAnalyzer object with the input file as the argument. The constructor method for that is in LexicalAnalyzer.java, where Java's StreamTokenizer class is used to tokenize the file's contents.

Main method then constructs a Project1 object with the LexicalAnalyzer object as the argument, using the constructor method in Project1.java. In this constructor, the GUIparser() method is called, which begins the parsing process. getNextToken() is also used to get the first token in the file and return an according Enum from EnumTokens.java. In GUIparser(), there is a long series of if statements that check whether the retrieved Enum in the file matches the required format. For example, any input file must begin with "Window" – GUIparser() begins with an if statement that compares the first token in the file to WINDOW from EnumTokens.java. If the first token in the file is indeed "Window", GUIparser() creates a JFrame and moves on to the next token using getNextToken() method. If they do not match, a FileSyntaxError is raised, and the location of the syntax error is printed as output. Now, the next token is compared to STRING, then L_PAREN – the process continues to check that the syntax of the file is proper.

As tokens are checked, they are applied to the GUI as components. For example, the first Number that is parsed is typecasted into an integer and used as the width of the GUI's window. The next Number (after a comma token) is also typecasted into an integer and used as the height of the window, then setSize(width, height) is used to complete the window's dimension settings. On a side note, getNumber() method from LexicalAnalyzer.java is called in this case to return the token as a double. Similarly, getLexeme() method is also in LexicalAnalyzer.java which returns any lexemes as Strings to be used in the GUI (as frame titles, labels, or button labels).

Layout, Widgets, and Radio buttons are all individually parsed separately from GUIparser(). When GUIparser() reaches any of those, it calls on their respective parsers to be parsed in other methods. In layoutParser(), there is an if/else-if statement where the enum after "Layout" can match GRID or FLOW. If it is FLOW, then FlowLayout is applied to the current layer. If it is GRID, the parser then checks for following numbers and sets them as the number of rows and

Week 4: Project 1

columns. If there are vertical gaps and horizontal gaps specified, the parser continues on and applies those to the layer too.

In `widgetsParser()` and `radioButtonsParser()`, the same parsing process as before is applied, but they also have helper methods `getWidgets()` and `getRadioButtons()`. These initialize the recursive descent, and are needed for widgets and radio buttons because those are the components that can create more widgets/radio buttons within themselves. As long as those helper methods are used, the recursion can properly start and end.

Finally, I used an `EnumTokens` object called “layer”. Initially, the whole window is assigned to be the layer. However, when a new panel is created from `widgetsParser()`, the new panel becomes assigned as the current layer. This way, other widgets and layout components can be applied to this panel and not the entire window.

As previously mentioned, any syntax error throughout this whole parsing process stops the program and raises `FileSyntaxError`, which prints out the location of the error.

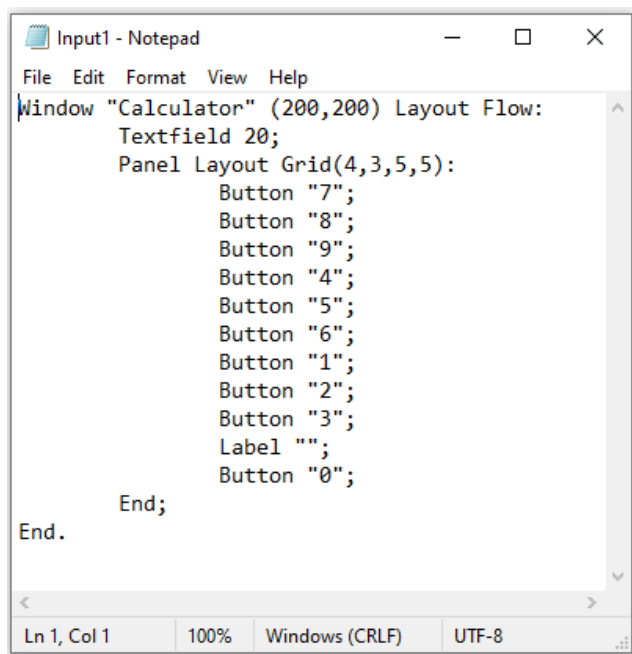
Lessons Learned:

The biggest obstacle for me in this project was how difficult it was to figure out how to develop the program incrementally. I did not expect a parser to be so jam-packed with nested if-statements, which are already a bit of a challenge for me to handle because I always have this thought looming in the back of my mind: “Isn’t there a more efficient way to do this?” But the more I built upon the vast nests of if-statements, the more I understood that this is exactly what creating a parser from scratch is supposed to be like. It wasn’t until I got around to dealing with the widgets and radio buttons that I realized I could just develop incrementally by testing each component of the definition language individually (seems obvious now). After that, I was able to test the GUI part by part, and I became at ease. From now on, I’ll try to write out a testing plan for incremental development. This helps for debugging along the way too.

A bad habit that I really need to get rid of when dealing with recursion is *not using a helper method from the start*. I am fully aware that helper methods are essential in most cases of recursion (so that the recursion can actually initialize and end properly) but it is never the thing I think of from the beginning. This time around, I started off by try to incorporate the recursion in `widgetsParser()`, and it took me too long of a while to figure out why the recursion wasn’t working out – I needed the `getWidgets()` helper method to control the recursion. Luckily, I immediately applied this to the radio buttons as well. I just need to get in the habit of doing this from the start.

Writing this parser was a very valuable and tedious experience to me that I certainly won’t be forgetting. The sheer amount of specificity that needs to be made with a parser – the giant nested if-blocks, the list of enums that covered all tokens, the extensive switch statements to return enums according to the respective tokens – caught me off guard. Even though the content we read in class definitely showed how dense parsers can be, I didn’t realize how many different possible inputs need to be accounted for until writing it myself. On a final note, though, I am quite satisfied after completing and can say I did enjoy writing it.

Week 4: Project 1

Test Case 1: Calculator GUI (provided in the Project 1 rubric)Input file name: *Input1.txt*


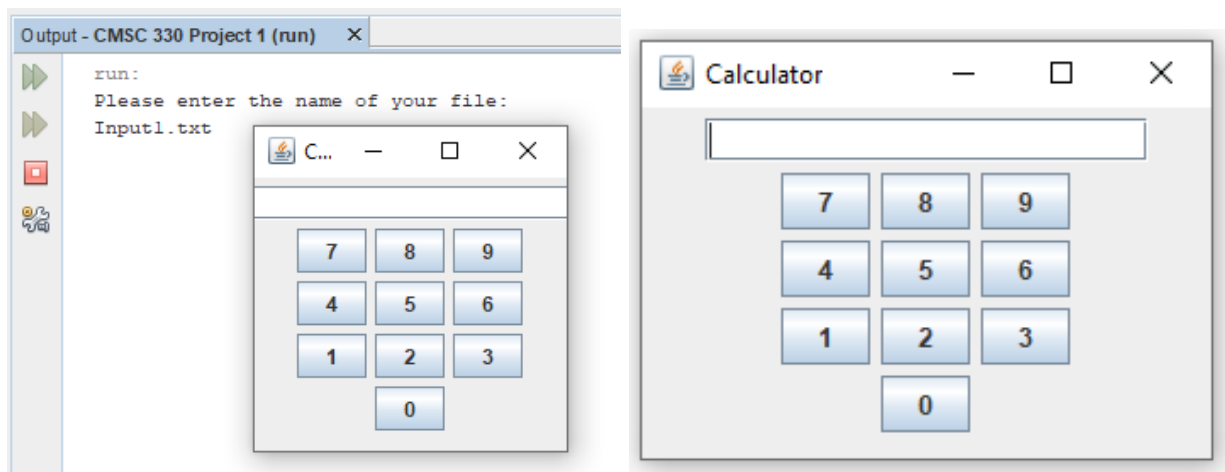
```

Input1 - Notepad
File Edit Format View Help
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.
Ln 1, Col 1 100% Windows (CRLF) UTF-8

```

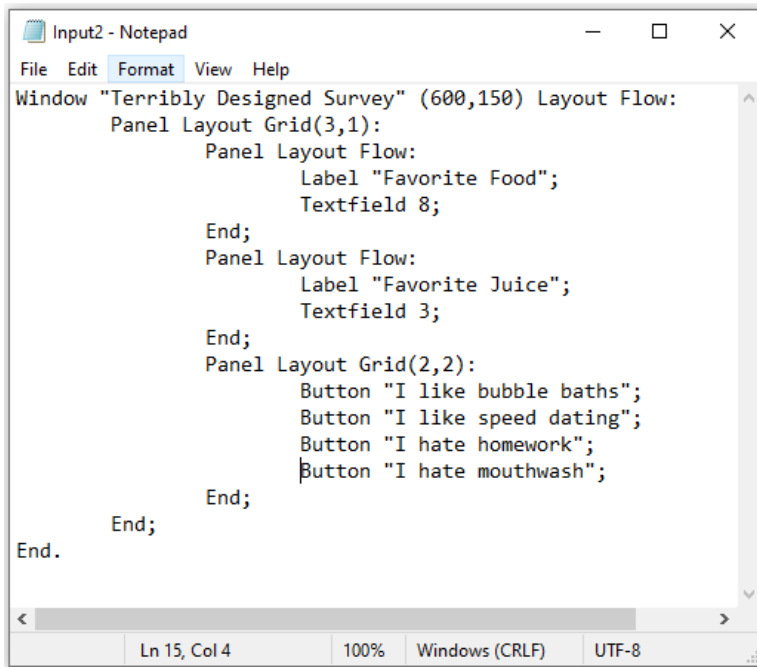
Compilation with generated GUI:

(The 200x200 dimensions were too small to let the frame title show, but I still wanted to prove that this program does indeed output the GUI to be 200x200. Here is a second screenshot of the title "Calculator" after I manually extended the width of the window.)



Results: SUCCESS. The Calculator GUI that was generated contains the correct frame title, textfield width, labels on buttons, Flow layout for the window, and Grid Layout components for the new Panel.

Week 4: Project 1

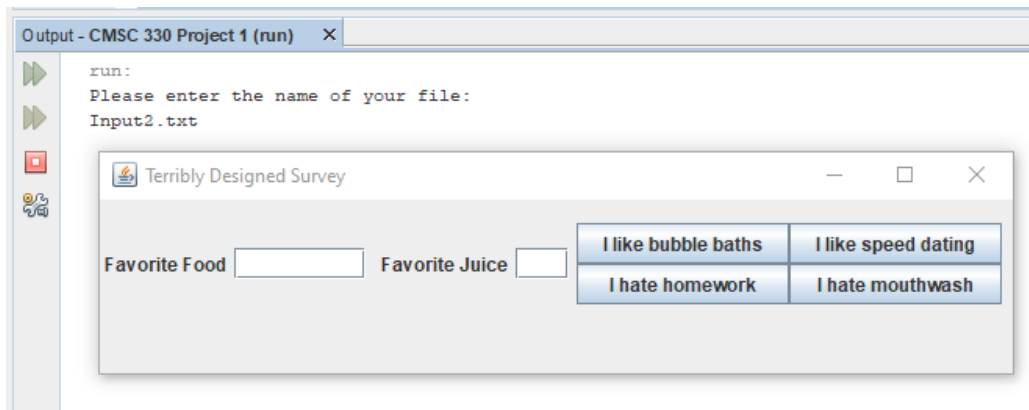
Test Case 2: Survey GUIInput file name: *Input2.txt*


```

Window "Terribly Designed Survey" (600,150) Layout Flow:
    Panel Layout Grid(3,1):
        Panel Layout Flow:
            Label "Favorite Food";
            Textfield 8;
        End;
        Panel Layout Flow:
            Label "Favorite Juice";
            Textfield 3;
        End;
        Panel Layout Grid(2,2):
            Button "I like bubble baths";
            Button "I like speed dating";
            Button "I hate homework";
            Button "I hate mouthwash";
        End;
    End;
End.

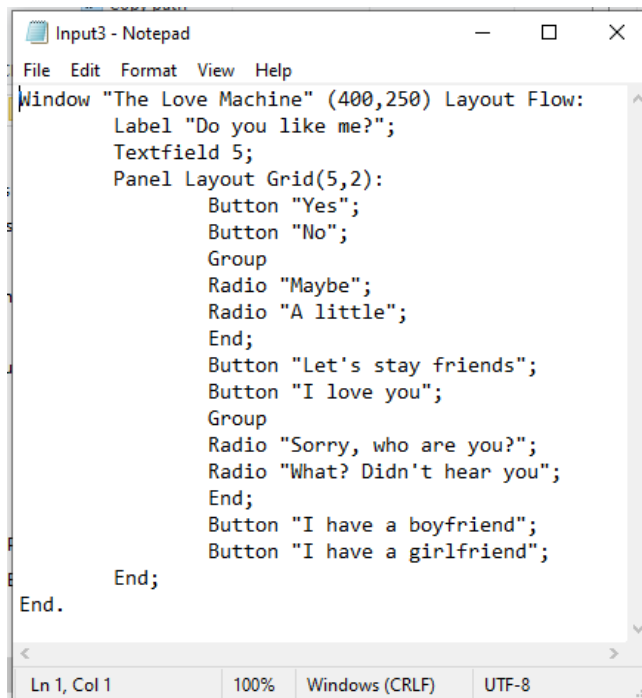
```

Compilation with generated GUI:



Results: SUCCESS. The "Terribly Designed Survey" GUI that was generated contains the correct frame title, Label names, textfield widths, labels on buttons, a panel that has three different panels nested inside it (one panel containing a Grid Layout), Grid layout for those inner three panels, button labels, and window dimensions.

Week 4: Project 1

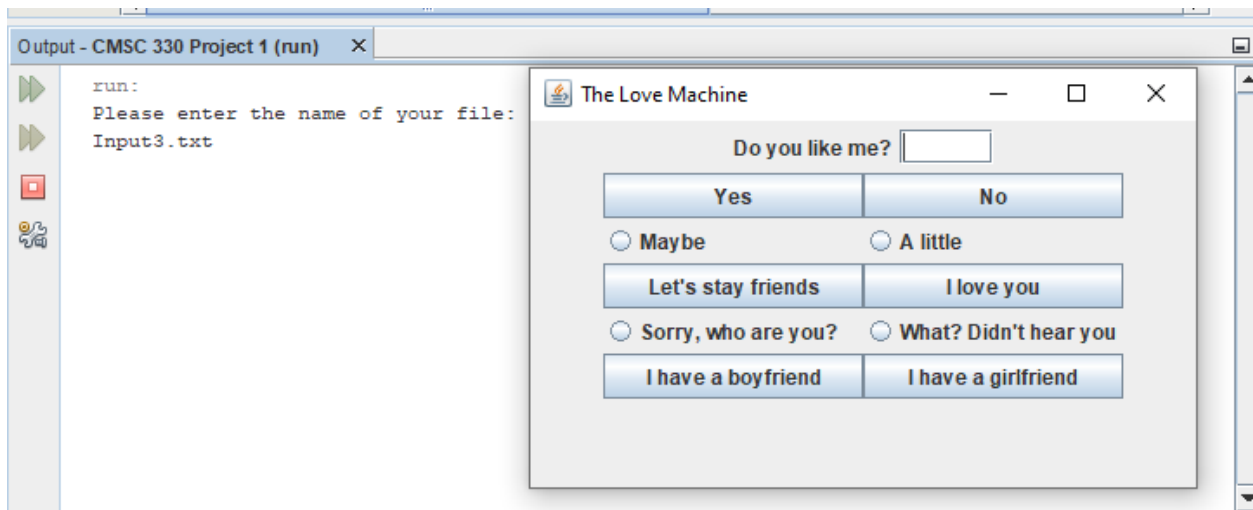
Test Case 3: Love Machine GUIInput file name: *Input3.txt*


```

File Edit Format View Help
Window "The Love Machine" (400,250) Layout Flow:
  Label "Do you like me?";
  Textfield 5;
  Panel Layout Grid(5,2):
    Button "Yes";
    Button "No";
    Group
    Radio "Maybe";
    Radio "A little";
    End;
    Button "Let's stay friends";
    Button "I love you";
    Group
    Radio "Sorry, who are you?";
    Radio "What? Didn't hear you";
    End;
    Button "I have a boyfriend";
    Button "I have a girlfriend";
  End;
End.
Ln 1, Col 1 100% Windows (CRLF) UTF-8

```

Compilation with generated GUI:

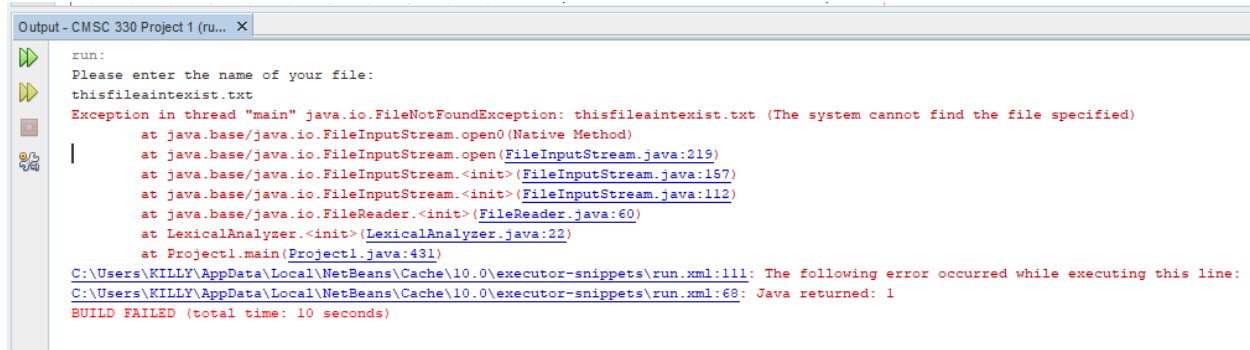


Results: SUCCESS. The Love Machine GUI that was generated contains the correct frame title, label name, textfield width, window dimensions, Grid Layout of 5 rows by 2 columns, button labels, two groups of radio buttons with two radio buttons inside each, and radio button names. The order in which each button is added is also correct.

Week 4: Project 1

Test Case 4: Will FileNotFoundException be caught?Input file name: *thisfileaintexist.txt*

Attempt to generate GUI from this nonexistent file:

The screenshot shows an IDE's output window titled "Output - CMSC 330 Project 1 (run...)". The window contains the following text:

```
run:
Please enter the name of your file:
thisfileaintexist.txt
Exception in thread "main" java.io.FileNotFoundException: thisfileaintexist.txt (The system cannot find the file specified)
    at java.base/java.io.FileInputStream.open0(Native Method)
    at java.base/java.io.FileInputStream.open(FileInputStream.java:219)
    at java.base/java.io.FileInputStream.<init>(FileInputStream.java:157)
    at java.base/java.io.FileInputStream.<init>(FileInputStream.java:112)
    at java.base/java.io.FileReader.<init>(FileReader.java:60)
    at LexicalAnalyzer.<init>(LexicalAnalyzer.java:22)
    at Project1.main(Project1.java:431)
C:\Users\KILLY\AppData\Local\NetBeans\Cache\10.0\executor-snippets\run.xml:111: The following error occurred while executing this line:
C:\Users\KILLY\AppData\Local\NetBeans\Cache\10.0\executor-snippets\run.xml:68: Java returned: 1
BUILD FAILED (total time: 10 seconds)
```

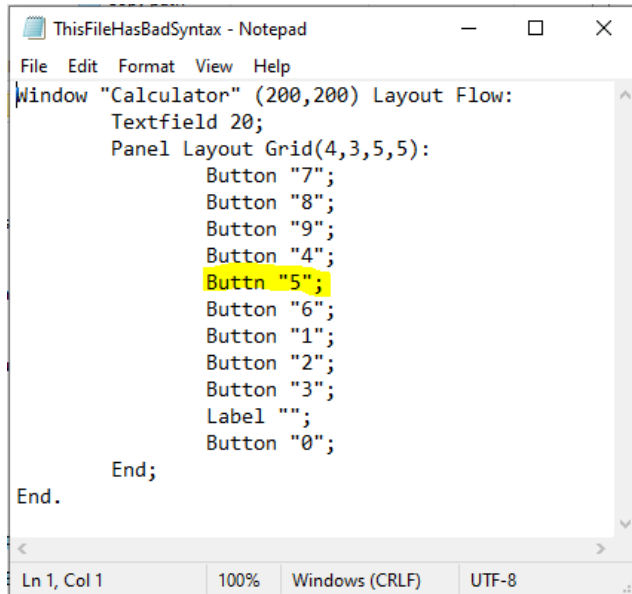
Results: SUCCESS. *thisfileaintexist.txt*, does in fact, NOT EXIST. This successfully raises `FileNotFoundException` that was thrown in the program and ends the run.

Week 4: Project 1

Test Case 5: Will FileSyntaxError be caught?

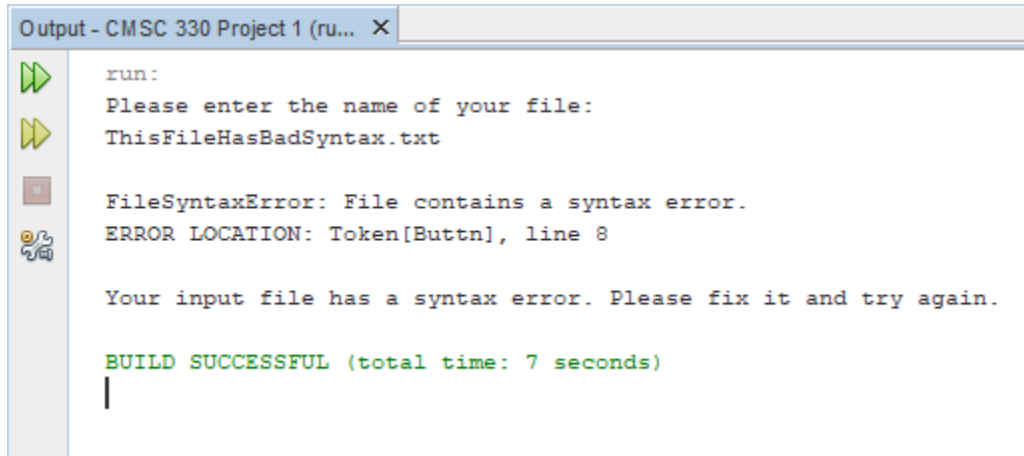
Input file name: *ThisFileHasBadSyntax.txt*

(I highlighted the line that has an error in the syntax)



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

Attempt to generate GUI from this file with improper syntax:



```
run:
Please enter the name of your file:
ThisFileHasBadSyntax.txt

FileSyntaxError: File contains a syntax error.
ERROR LOCATION: Token[Buttn], line 8

Your input file has a syntax error. Please fix it and try again.

BUILD SUCCESSFUL (total time: 7 seconds)
```

Results: SUCCESS. In Line 8, Button is missing the character “o”. The program successfully detects it, raises the exception, and prints out the location of the error WITH the error token itself. The run is then ended, as intended. Note that this is almost exactly the Calculator GUI from Input1.txt – a GUI can be generated for this file otherwise, but now that Line 8 has the syntax error, the GUI cannot be generated and FileSyntaxError is raised.