Project 1

CMSC 330 6381

Professor Ellis, Brenda

Wanner HernandezR

February 8, 2021

**Assignment Details**

The first programming project involves writing a program that parses, using recursive descent, a GUI definition language defined in an input file and generates the GUI that it defines. The grammar for this language is defined below:

gui ::=  
Window STRING '(' NUMBER ',' NUMBER ')' layout widgets End '.'

layout ::=  
Layout layout\_type ':'

layout\_type ::= Flow |

Grid '(' NUMBER ',' NUMBER [',' NUMBER ',' NUMBER] ')' widgets ::=

widget widgets |

widget widget ::=

Button STRING ';' |  
Group radio\_buttons End ';' | Label STRING ';' |  
Panel layout widgets End ';' | Textfield NUMBER ';'

radio\_buttons ::=  
radio\_button radio\_buttons | radio\_button

radio\_button ::= Radio STRING ';'

In the above grammar, the red symbols are nonterminals, the blue symbols are tokens and the black punctuation symbols are BNF metasymbols. Among the tokens those in title case are keywords. The character literals are punctuation tokens.

Below is an explanation of the meaning of some of the symbols in the above productions that should help you understand the actions that are to be performed when each of the productions is parsed:

•  In the window production the string is the name that is to appear in the top border of the window and the two numbers are the width and height of the window

•  In the production for layout\_type that define the grid layout, the first two numbers represent the number of rows and columns, and the optional next two the horizontal and vertical gaps

•  In the production for widget that defines a button, the string is the name of the button

•  In the production for widget that defines a label, the string is text that is to be placed in the label

•  In the production for widget that defines a text field, the number is the width of the text field

•  In the production for radio button, the string is the label of the button

You parser should properly handle the fact that panels can be nested in other panels. Recursive productions must be implemented using recursion. Syntactically incorrect input files should detect and report the first error.

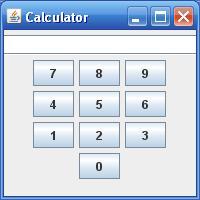
Below is an example of an input file:

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.

The above input file should produce the GUI shown below:



## **Process and Lessons Learned**

Initially I was quite intimidated at the thought of taking theoretical knowledge and applying it practically, however, it quickly became apparent the project would be quite easy. The first task I set for myself was determining how to read the file In particular I was concerned with how I would track and report what line errors occurred on. Because I needed to track what line I was on, my initial thought was to tokenize the newline character itself however after some research I settled on using two scanners rather than one. The first would pull the entire line (and thus would allow me to count what line I was on) and the second would tokenize the line with regex. This was accomplished with the next() and nextLine() methods. My primary purpose with the regex was to not only split by whitespace but to split in-between special characters like quotes, parentheses, semicolons and colons so that these characters would be tokens by themselves.

My second task was creating methods to read through the scanner and creating the recursive descent parser. I accomplished this with nested if statements, which while far from elegant, was effective. Each if statement verified that a series of tokens matched the grammar of a given method. Any failed if statement throws a custom exception that indicates where the syntax error occurred. This, while the easiest part of the programming, took the longest.  
  
Third I implemented the full functionality of the compiler and ensured that correctly parsed statements created the desired effect in the GUI. This meant associating Swing components with each method as well as determining the variables that would need to be passed to each method.

This project helped me learned a great deal about compilers in a practical sense rather than a theoretical one. Such methods could very easily be adapted to simply allow for more modular programs. In the future, I would devote more time to finding a more elegant solution rather than nested if statements which really destroys readability.

**Compiled:Graphical user interface, text, application

Description automatically generated**

**Test case Input/ Images:**

**Test case 1:**

Window "Calculator" (400, 800) Layout Flow:

Textfield 20;

Panel Layout Grid(4, 3, 5, 15):

Button "7";

Button "8";

Button "9";

Button "4";

Button "5";

Button "6";

Button "1";

Button "2";

Button "3";

Label "";

Button "0";

End;

End.

Image:

Graphical user interface, text, application

Description automatically generated

**Test case 2:**

Window "Calculator" (400, 400) Layout Flow:

Textfield 20;

Panel Layout Grid(4, 10, 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.

Image:

Graphical user interface, text, application

Description automatically generated

**Test case 3:**

Window "Calculator" (800, 800) Layout Flow:

Textfield 20;

Panel Layout Grid(4, 10, 5, 5):

Button "7";

Button "cook";

Button "9";

Button "4";

Button "5";

Button "6";

Button "1";

Button "2";

Button "breakfast";

Label "";

Button "delete";

End;

End.

Image:

Graphical user interface, text, application

Description automatically generated

**Test case 4:**

Window "Calculator" (400, 400) Layout Flow:

Textfield 20;

Panel Layout Grid(50, 35, 5, 10):

Button "7";

Button "8";

Button "9";

Button "4";

Button "5";

Button "6";

Button "1";

Button "2";

Button "3";

Label "";

Button "0";

End;

End.

Image:

**Graphical user interface, text, application

Description automatically generated**

**Test Case 5:**

Window "Calculator" (400, 400) Layout Flow:

Textfield 35;

Panel Layout Grid(4, 3, 25, 20):

Button "7";

Button "LKASJFH";

Button "9";

Button "#$%";

Button "5";

Button "6";

Button "1";

Button "2";

Button "case";

Label "";

Button "0";

End;

End.

Image:

Graphical user interface, text, application

Description automatically generated

**Test case 6:**

When the file is not in the program file this error will happen.

File not found

java.io.FileNotFoundException: Input.txt (No such file or directory)

at java.base/java.io.FileInputStream.open0(Native Method)

at java.base/java.io.FileInputStream.open(FileInputStream.java:211)

at java.base/java.io.FileInputStream.<init>(FileInputStream.java:153)

at java.base/java.util.Scanner.<init>(Scanner.java:639)

at com.mycompany.cmsc330project1.GUICreator.main(GUICreator.java:45)

Image:

Graphical user interface, text, application, email

Description automatically generated