Anthony Borza

CMSC 330 6380

Due Date: 2/5/17

Project 1 Writeup

a. A Brief Description of your Approach to the Design:

The purpose of this project was to write a program that parses, using recursive descent, a GUI definition language that is defined in an input file and generate the GUI that it defines, using a supplied lexical analyzer. Each segment of grammar shown below represents a method in the GUIParser.java class. The GUIParser.java class uses a variety of switch statements, and if, else if, and else statements that are defined in each method. The goal of this program is to understand the meaning of the symbols (Terminal, Non-Terminal, and **BNF Metasymbol's**) in the above production. This will help the programmer better understand the necessary actions that are performed when each of the productions (segments of grammar) are parsed.

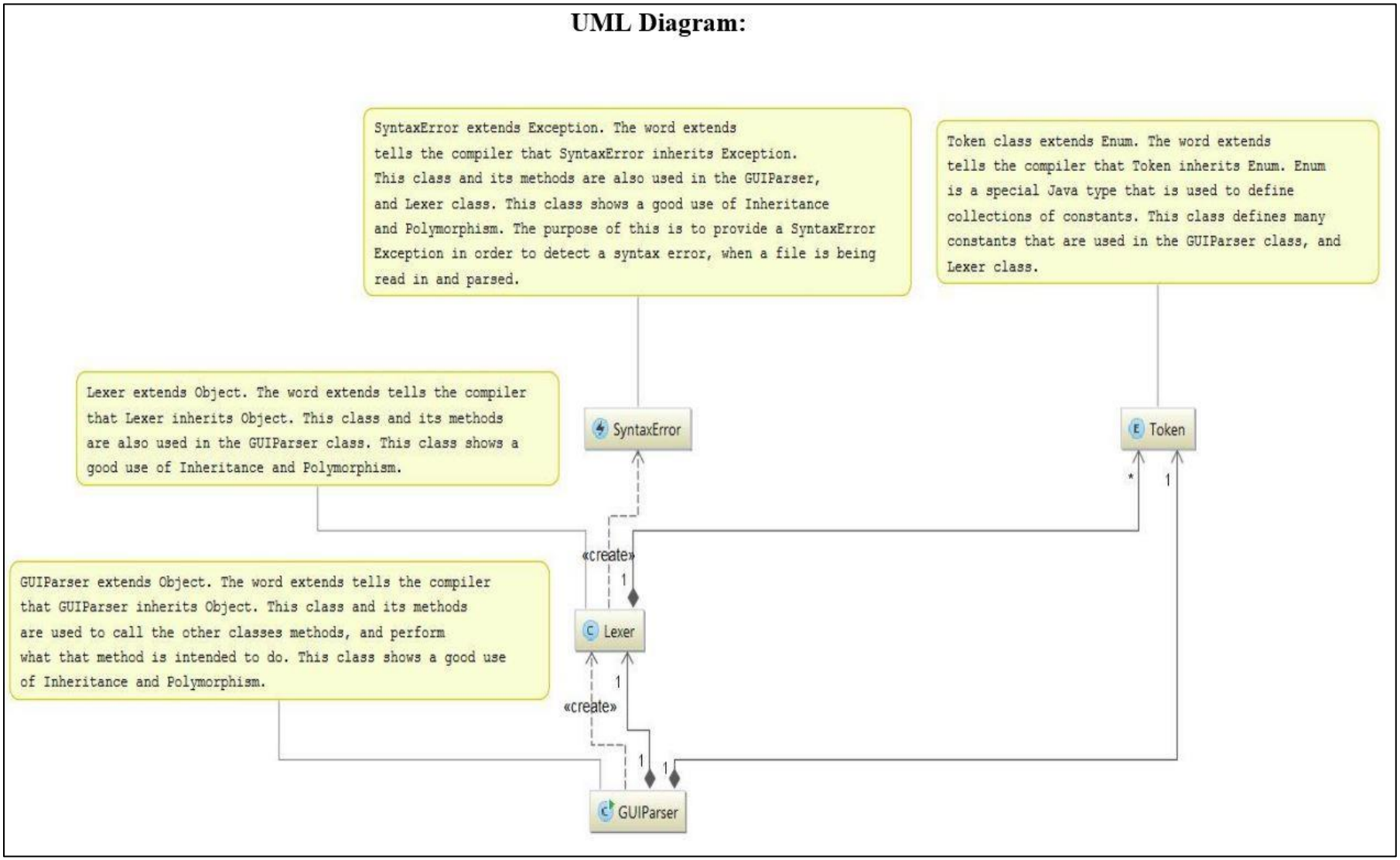
In addition, there are three other classes that are provided and needed to complete this project. The Lexer.java class which is a class that provides a Lexical analyzer. The purpose of a Lexical Analyzer involves the process of converting a sequence of characters (in our case, characters from a file) and then putting them into a sequence of tokens (in our case, strings that are assigned and identified with a meaning. The Token.java class is an enumerated type class that defines a list of tokens (in our case, Constants) that are used in both the Lexer.java class, and GUIParser.java class. The SyntaxError.java class that extends Exception, is a class that defines a syntax error, and creates a syntax error object, and provides the given line number and error if one occurs during execution of the program.

The grammar for this project is the following:

```
// This would represent a method in the GUIParser.java class

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

// This would represent a method in the GUIParser.java class

layout ::=
    Layout layout_type ':'

// This would represent a method in the GUIParser.java class

layout_type ::=
    Flow |
    Grid '(' NUMBER ',' NUMBER [',' NUMBER ',' NUMBER] ')'

// This would represent a method in the GUIParser.java class

widgets ::=
    widget widgets |
    widget

// This would represent a method in the GUIParser.java class

widget ::=
    Button STRING ';' |
    Group radio_buttons End ';' |
    Label STRING ';' |
    Panel layout widgets End ';' |
    Textfield NUMBER ';'

// This would represent a method in the GUIParser.java class

radio_buttons ::=
    radio_button radio_buttons |
    radio_button

// This would represent a method in the GUIParser.java class

radio_button ::=
    Radio STRING ';'
```

b. A UML Class diagram that Includes all Classes Including Any That Were Supplied. Do Not Include Predefined Classes. You Need Only Include the Class Name for Each Individual Class, not the Variables or Methods

UML Diagram:



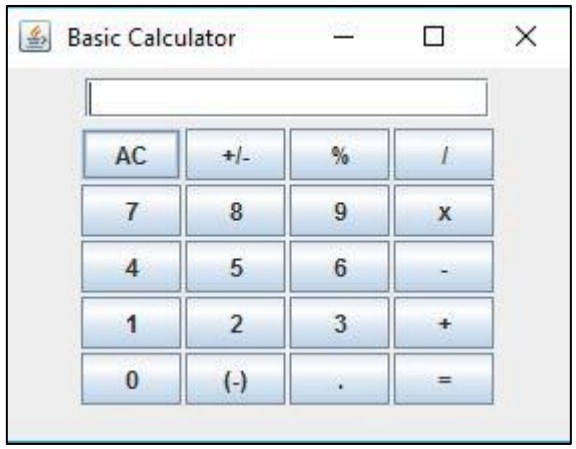
c. <u>A Test Plan that Includes Test Cases that you Have Created Indicating What aspects of the Program Each One is Testing.</u>

- I will provide three test cases:

<u>Test Plan Example 1:</u>

```
Window " Basic Calculator" (300, 225) Layout Flow:
  Textfield 18;
  Panel Layout Grid (5,4,2,2):
    Button "AC";
    Button "+/-";
    Button "%";
    Button "/";
    Button "7";
    Button "8";
    Button "9";
    Button "x";
    Button "4";
    Button "5";
    Button "6";
    Button "-";
    Button "1";
    Button "2";
    Button "3";
    Button "+";
    Button "0";
    Button "(-)";
    Button ".";
    Button "=";
  End;
End.
```

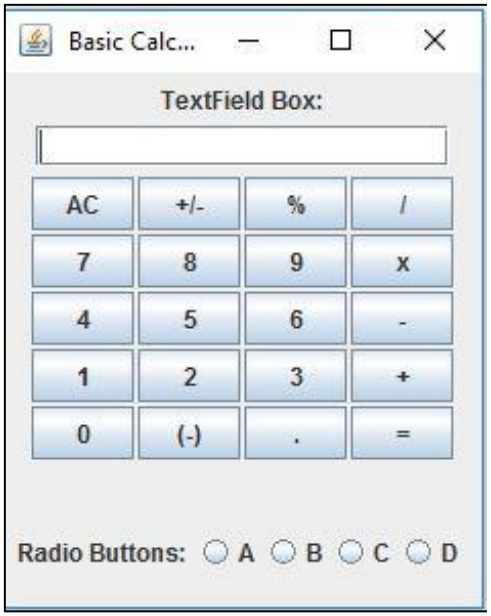
Output Results:



Test Plan Example 2:

```
Window " Basic Calculator" (250, 300) Layout Flow:
  Label "TextField Box:";
  Textfield 18;
  Panel Layout Grid (6,3,2,2):
    Button "AC";
    Button "+/-";
    Button "%";
    Button "/";
    Button "7";
    Button "8";
    Button "9";
    Button "x";
    Button "4";
    Button "5";
    Button "6";
    Button "-";
    Button "1";
    Button "2";
    Button "3";
    Button "+";
    Button "0";
    Button "(-)";
    Button ".";
    Button "=";
  End;
   Label "Radio Buttons:";
   Panel Layout Grid (1,4):
       Group
              Radio "A";
              Radio "B";
              Radio "C";
              Radio "D";
       End;
  End;
End.
```

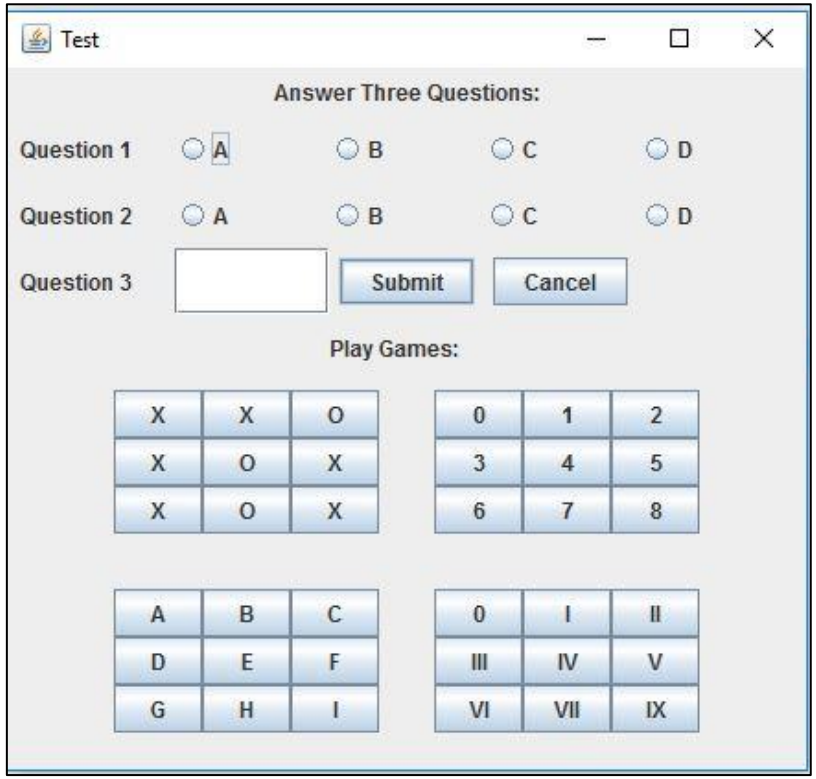
Output Results:



Test Plan Example 3:

```
Window "Test" (450, 420) Layout Flow:
    Label "Answer Three Questions:";
     Panel Layout Grid (0, 5):
         Label "Question 1";
         Group
             Radio "A";
             Radio "B";
             Radio "C";
             Radio "D";
         End;
         Label "Question 2";
         Group
             Radio "A";
             Radio "B";
             Radio "C";
             Radio "D";
         End;
         Label "Question 3";
         Textfield 1;
         Panel Layout Flow:
             Button "Submit";
         End;
         Panel Layout Flow:
             Button "Cancel";
         End;
          Label "";
          Label "";
          Label "";
          Label "Play Games:";
          Label "";
     End;
     Panel Layout Grid (2, 2, 30, 30):
         Panel Layout Grid (3, 3):
             Button "X";
             Button "X";
             Button "O";
             Button "X";
```

Output Results:



```
    Button "O";
    Button "X";
    Button "X";
    Button "O";
    Button "X";
End;
Panel Layout Grid (3,3):
    Button "0";
    Button "1";
    Button "2";
    Button "3";
    Button "4";
    Button "5";
    Button "6";
    Button "7";
    Button "8";
End;
Panel Layout Grid (3,3):
    Button "A";
    Button "B";
    Button "C";
    Button "D";
    Button "E";
    Button "F";
    Button "G";
    Button "H";
    Button "I";
End;
Panel Layout Grid (3, 3):
    Button "0";
    Button "I";
    Button "II";
    Button "III";
    Button "IV";
    Button "V";
    Button "VI";
    Button "VII";
    Button "IX";
End;
```

```
    End;
End.
```

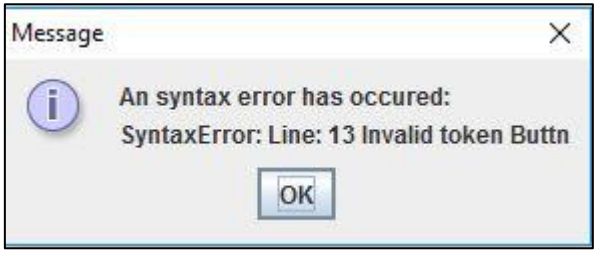
Exception Checks:

Example 1: Invalid Layout Type

```
Window " Basic Calculator" (250, 300) Layout Flow:
  Label "TextField Box:";
  Textfield 18;
  Panel Layout rid (6,3,2,2):
    Button "AC";
    Button "+/-";
    Button "%";
    Button "/";
    Button "7";
    Button "8";
    Button "9";
    Button "x";
    Button "4";
    Button "5";
    Button "6";
    Button "-";
    Button "1";
    Button "2";
    Button "3";
    Button "+";
    Button "0";
    Button "(-)";
    Button ".";
    Button "=";
  End;
End.
```

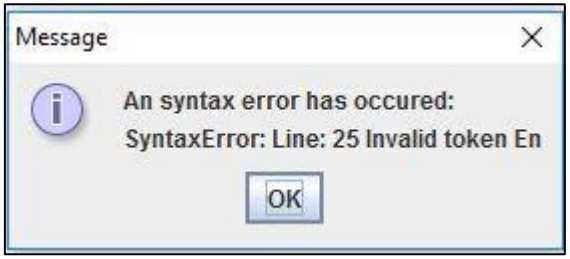
Example 2: Invalid Widget:

```
Window " Basic Calculator" (250, 300) Layout Flow:
  Label "TextField Box:";
  Textfield 18;
  Panel Layout rid (6,3,2,2):
    Button "AC";
    Button "+/-";
    Button "%";
    Button "/";
    Button "7";
    Button "8";
    Button "9";
    Button "x";
    Buttn "4";
    Button "5";
    Button "6";
    Button "-";
    Button "1";
    Button "2";
    Button "3";
    Button "+";
    Button "0";
    Button "(-)";
    Button ".";
    Button "=";
  End;
End.
```



Example 3: Invalid Token:

```
Window " Basic Calculator" (250, 300) Layout Flow:
  Label "TextField Box:";
  Textfield 18;
  Panel Layout Grid (6,3,2,2):
    Button "AC";
    Button "+/-";
    Button "%";
    Button "/";
    Button "7";
    Button "8";
    Button "9";
    Button "x";
    Button "4";
    Button "5";
    Button "6";
    Button "-";
    Button "1";
    Button "2";
    Button "3";
    Button "+";
    Button "0";
    Button "(-)";
    Button ".";
    Button "=";
  En;
End.
```



d. A Short Paragraph on Lessons Learned from the Project:

After completing project 1, there are many things that I learned throughout the process of creating a Java Parser that would parse using the recursive descent, a GUI definition language defined in an input file and generates the GUI that is defined, using a supplied lexical analyzer. The first thing, like with all programs you write, is understanding what is being asked, and what is the best, and most efficient way for completing it. I really found week 2 Recursive and Descent Parsing and Semantics PowerPoint to be very helpful when developing the methods in my parser class. The most valuable part of the PowerPoint were the slides on Production with one RHS, and Production with multiple RHS. This was the foundation that I went off when developing the methods in my parser class. The hardest part for me was figuring out how to parse nested panels from the input file. With guidance and suggestions from the Professor, and other students, it took me a good two days to realize that you had to pass in a container to all the parse methods created. Once I figured this out, my program was then able to handled nested panels when reading in a file to parse. Overall, I am very satisfied with the results of my program, and feel very confident with the concept of recursive descent parsing.