Name: SATYAJIT GHANA Registration Number: 17ETCS002159

Laboratory 1

Title of the Laboratory Exercise: GUI design and Exception Handling

1. Introduction and Purpose of Experiment

   Students apply object oriented programming concepts for creating Graphical User Interfaces (GUIs).

2. Aim and Objectives

   Aim

   To apply object oriented programming concepts for creating Graphical User Interfaces

   Objectives

   At the end of this lab, the student will be able to

   - Apply Object Oriented Programming to create Graphical User Interfaces
   - Create stand-alone application user interfaces

3. Experimental Procedure

For the problems listed below, design the data structures, algorithm(s) and write the program(s). Tabulate the output for various inputs and verify against expected values. Compare the programming method in Java with C and Haskell programming languages. Describe your learning along with the limitations of overall approach if any. Suggest how these can be overcome.

a. Create a simple calculator using FXML in Java. Include exception handling logic that reports exceptions such as divide by zero in human readable pop-up messages.

Name: SATYAJIT GHANA Registration Number: 17ETCS002159

Documentation:	
a. Procedure and Algorithm(s):	
b. Conclusions :	

Procedure:

Open Netbeans, Create a new JavaFX FXML Application, this will generate the necessary JavaFX Application from Template Application of NetBeans which gives a default button, and a button listener that reacts to button pressed. We remove these codes and open the fxml file in scene builder. The necessary UI Components are then added to the Scene that is simple drag-and-drop, this approach of adding all the UI first then writing the logic behind each of those components is better and I find it more comfortable. Hence after adding these components of the Calculators, i.e an Anchor Pane and Grid Box, all the controllers and ids are assigned. Then the logic for these are written. After this the Program is run and tested for various inputs and the output is verified for the same.

The UI was designed as follows, the root is an Anchor Pane, which contains the children, Label for the result, another Label for the history and the rest of the Buttons are wrapped inside a Grid Pane. All of these are drag-and-dropped onto the SceneBuilder Canvas.

Algorithms:

```
buttonNumClicked:

1.  Get the source of the event and fetch the text from it, then store it in
    numClicked.
2.  Set the text in resultLabel to the concatenated text of current resultLabel
    and the numClicked
3.  Set the text in historyLabel to the concatenated text of current
    historyLabel and numClicked

buttonOperatorClicked:
1.  Get the source of the event and fetch the text from it, then store it in
    operatorClicked.
2.  Set the text in resultLabel to operatorClicked.
3.  Set the text in historyLabel to the concatenated text of current
    historyLabel, " ", operatorClicked and " ".

buttonEqualsClicked: use the parseEquation Algorithm

Declare operators as a String of "-+*/"

(this is in order of precedence)

parseEquation:
```

Name: SATYAJIT GHANA Registration Number: 17ETCS002159

1. Get the text stored in historyLabel and replace " " (spaces) with "" (empty) and store it in toParse.
2. Tokenize toParse with operators and keep the operator occurences, then store it in tokens. (basically split toParse on operators and keep the operator characters also).
3. Declare Stacks valueStack and operatorStack and initialize them with empty stacks of String.
4. Get the nextToken from tokens and store it in _token.
5. if _token equals to "-" then Push concatenated String of "-" and the nextToken from tokens into valueStack.
   else Push _token into valueStack
6. while there are more token in tokens:
   1. Get the nextToken from tokens and store it in _token
   2. If operators contains the String _token
      1. while the operatorStack is not empty AND the indexOf _token in operators is LESS than OR EQUAL to the indexOf the top element in operatorStack:
         1. Pop an element from valueStack and store it in temp.
         2. Use doCalculation Algorithm with Operand1: Pop an element from valueStack, Operator: Pop and element from operatorStack, Operand2: temp, and the value hence obtained, Push it to valueStack
      2. Push _token into operatorStack
      3. Get the nextToken from tokens and store it in _token
      4. If _token equals "-" then Push the concatenated String "-", nextToken from tokens
         Else push _token into valueStack

   Else Push _token into valueStack

7. Use EvaluateExpression algorithm with operatorStack, and valueStack and the value hence obtained, store it in answer.
8. Set the text in resultLabel to answer
9. Set the text in historyLabel to answer

doCalculation: expects String operand1, String operator, String operand2

1. Initialize variable result as 0
2. parse operand1 as Double and save it in num1
3. parse operand2 as Double and save it in num2
4. switch(the first character in operator)
   if it was '+':
   ADD num1 and num2 and save it in result
   If it was '-'
   SUBTRACT num1 and num2 and save it in result
   If it was '*'
   MULTIPLY num1 and num2 and save it in result
   If it was '/'
   DIVIDE num1 and num2 and save it in result
   If the num2 is 0, throw an ArithmeticException
5. If during any of the steps 2 to 4 there was an Error then Throw the error

```
6.   If there was Exception, Create an Alert with the Error Message
7.   Return the value result

evaluateExpression: expects Stack operatorStack and Stack valueStack

1.   While the operatorStack is NOT empty
     1.  Pop an element from valueStack and store it in temp
     2. Use doCalculation Algorithm with Operand1: Pop an element from
        valueStack, Operator: Pop an element from operatorStack, Operand2: temp
2.   Pop the element from valueStack and Return this value
```

Conclusions:

A very simple and limited operations calculator was made, the operations that cat be performed includes **+ , − , / , *** , in which **−** can act as both a unary operator or a binary operator. This will give the ability to work with negative numbers also, since any, however basic should definitely include negative numbers, no-one would use a calculator that cannot deal with negative numbers. Event-Driven Programming was used for making these, the Application presents the user with a good looking UI with Buttons and Labels for output. This is also an example of Object-Action type of interaction with the user, where the action is presented after the object.

There are a few limitations of course, and there's a reason it was named the naïve calculator, it uses naïve methods to do the operations, a simple Value Stack and an Operator Stack is used to evaluate the infix operation. Even though we have used Floats as the variable to store the numbers and do the calculations, the calculator still cannot do calculations involving very high precision since the logic used to parse the String expression does not work with the Scientific Notations, i.e. in simple words this cannot parse a Scientific Notation and throws an Exception when such a situation occurs. Another limitation is that it does not take input with decimals, though there is a workaround i.e. first divide the number with 10^<something>, the calculator uses BODMAS and can parse the equation with correct operator precedence, another feature that it lacks is that it does not have parenthesis support, which is somewhat vital for a calculator. But adding all these features may clutter the display UI of the calculator and then it would need a UI redesign with intuitive design such that any ordinary user can get used to it easily.

Now that we discussed these Limitations, the problem is even if you add the code to fix this then the Controller Class will become monstrous and very difficult to maintain, hence that is the main limitation of this program, it's not modularized and neither the Calculator is separated from the Controller.

One recommendation would be to separate the Calculator Logic in its own Class, this would give more Object Oriented Practice into our program, whenever we want to use a calculator then we would create an instance of this calculator and do the operations, this also separates our code and makes the Controller less complex and more readable. The Algorithm used to parse the infix operation is not as optimized and can be further optimized.

Name: SATYAJIT GHANA Registration Number: 17ETCS002159

Results and Discussions:
Screenshot:
Discussion:

Source Code of the Core Methods:

```java
    private String operators = "-+*/";

    private void parseEquation() {
        /* To Remove thw White Spaces */
        String toParse = historyLabel.getText().replace(" ", "");
        /* Tokenize the String */
        StringTokenizer tokens = new StringTokenizer(toParse, this.operators, true);
        Stack<String> valueStack = new Stack();
        Stack<String> operatorStack = new Stack();

        String _token = tokens.nextToken();
        /* If there is a negative at the beginning */
        if (_token.equals("-")) {
            valueStack.push("-" + tokens.nextToken());
        } else {
            valueStack.push(_token);
        }
        while (tokens.hasMoreTokens()) {
            _token = tokens.nextToken();
            if (operators.contains(_token)) {
                while (!operatorStack.isEmpty() && (operators.indexOf(_token) <= operators.indexOf(operatorStack.peek()))) {
                    String temp = valueStack.pop();
                    valueStack.push(doCalculation(valueStack.pop(), operatorStack.pop(), temp));
                }
                operatorStack.push(_token);
                _token = tokens.nextToken();
                /* If there is a negative number */
                if (_token.equals("-")) {
                    valueStack.push("-"+tokens.nextToken());
                } else {
                    valueStack.push(_token);
                }
            } else {
                valueStack.push(_token);
            }
        }
        String answer = evaluateExpression(operatorStack, valueStack);
        resultLabel.setText(answer);
        historyLabel.setText(answer);
    }

    private String evaluateExpression(Stack<String> operatorStack, Stack<String> valueStack) {
        while (!operatorStack.isEmpty()) {
            String temp = valueStack.pop();
            valueStack.push(doCalculation(valueStack.pop(), operatorStack.pop(), temp));
        }
        return valueStack.pop();
    }
```

Name: SATYAJIT GHANA Registration Number: 17ETCS002159

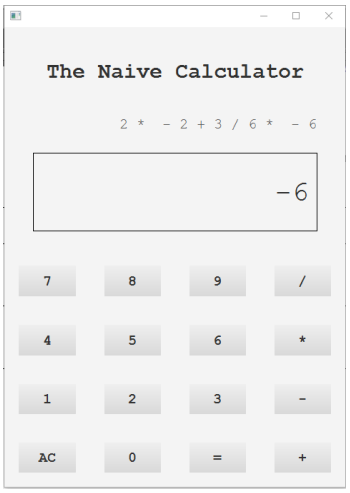
```java
@FXML
    private void clearClicked(ActionEvent event) {
        resultLabel.setText("");
        historyLabel.setText("");
        result = 0d;
    }

    @FXML
    private void buttonNumClicked(ActionEvent event) {
        String numClicked = ((Button) event.getSource()).getText();
        resultLabel.setText(resultLabel.getText() + numClicked);
        historyLabel.setText(historyLabel.getText() + numClicked);
    }

    @FXML
    private void buttonEqualsClicked(ActionEvent event) {
        parseEquation();
    }

    @FXML
    private void buttonOperatorClicked(ActionEvent event) {
        String operatorClicked = ((Button) event.getSource()).getText();
        resultLabel.setText(operatorClicked);
        historyLabel.setText(historyLabel.getText() + " " + operatorClicked + " ");
    }

```

```java
    private String doCalculation(String op1, String opr, String op2) {
        Double result = 0d;
        try {
            Double num1 = Double.parseDouble(op1);
            Double num2 = Double.parseDouble(op2);

            switch (opr.charAt(0)) {
                case '+':
                    result = num1 + num2;
                    break;
                case '-':
                    result = num1 - num2;
                    break;
                case '*':
                    result = num1 * num2;
                    break;
                case '/':
                    result = num1 / num2;
                    if (num2 == 0) throw new ArithmeticException();
                    break;
            }
        } catch (NumberFormatException e) {
            Alert alert = new Alert(Alert.AlertType.ERROR);
                alert.setTitle("Scientific Notation Detected");
                alert.setHeaderText("Error: cannot parse a Scientific Notation");
            alert.setContentText(e.toString());
            alert.showAndWait();
        } catch (ArithmeticException e) {
            Alert alert = new Alert(Alert.AlertType.ERROR);
                alert.setTitle("Cannot Divide By Zero");
                alert.setHeaderText("Error: This is a Naive Calculator not a Scientific Calculator, though i've still tried my best.");
            alert.setContentText(e.toString());
            alert.showAndWait();
        } finally {
                    return result.toString();
        }

    }
```

Ramaiah University of Applied Sciences
Private University Established in Karnataka State by Act No. 15 of 2013

Name: SATYAJIT GHANA

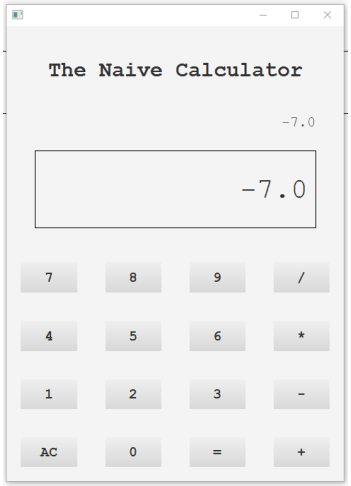
Registration Number: 17ETCS002159

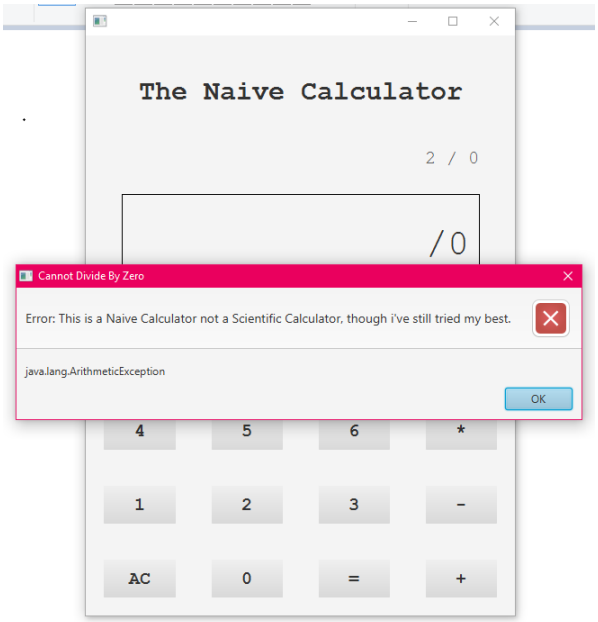


The Following Buttons were pressed in sequence to reach this screen,

2 * - 2 + 3 / 6 * - 6

Then the button = was clicked on, the corresponding function that was a observer to this was called, which produced the output and displayed it on the screen.





Lets Now see what happens when we try to divide a number by 0. A alert is raised with a suitable text. The result is though set to Infinity since logically a positive constant other than 0 divided by 0 is Infinity, if 0/0 is tried it will result in NaN.