1. **Formulating the Problem**

**1.1 Problem Description**

Design and implement a postfix expression evaluator. The program should use a stack to store and evaluate the postfix expression.

**1.2** **Verbalization**

What is the goal?

Evaluate a postfix expression.

What are the givens?

Operands and operators.

What are the unknowns?

The result of the expression.

**1.3 Information Elicitation**

Goal Postfix Expression Evaluator

Givens Operands and operators

Unknowns Result

**2. Planning the Solution**

**2.1. Solution Strategy**

Get input from the user. If the input is an operand store it in the stack. However, if the input is an operator get the top two operands from the stack and evaluate them. Then store the results in the stack. Repeat this process until all operators are evaluated.

**2.2 Goal Decomposition**

Sub-goal 1

Create the stack.

Sub-goal 2

Get input from user.

Sub-goal 3

Store the input in the stack.

Sub-goal 4

Evaluate the expression.

Sub-goal 5

Display results.

**2.4 Data Organization and Description**

**Inputs**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | **Origin** | **Used in Sub-goal #** |
| operand1 | Stores the first operands | User | 2 |
| operand2 | Stores the second operand | User | 2 |

**Output**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | **Origin** | **Used in Sub-goal #** |
| result | Stores the results of the expression | Screen | 5 |

**3. Designing the Solution**

**3.1 Structure Chart**

First Level Decomposition

Goal Refinement

**Sub-goal 1**

Display the GUI.

**Sub-goal 2**

Create the Stack

**Sub-goal 3**

Get input from the user.

**Sub-goal 3.1**

If input is an operand store it in the stack.

**Sub-goal 3.2**

If input is an operator get the top two operand from the stack and evaluate the expression.

**Sub-goal 4**

Repeat Sub-goal 3 until the expression is completely evaluated.

**Sub-goal 5**

Display the last item(result) in the stack.

Second Level Decomposition

**3.2 Module and Data Specifications**

**Name:** btnClear - Clears the stack and the input fields

**Input:** None.

**Output:** Blank input fields

**Logic:** Clear the stack by using the pop method to remove all of the elements from the stack.

**Name:** btnEqual - display result.

**Input:** Remaining element in the stack.

**Output:** The result of expression.

**Logic:** Pop the remaining element from the stack.

**Name:** btnAdd- perform addition .

**Input:** Top two elements in the stack.

**Output:** None.

**Logic:** Get the top two elements from the stack and add them. Then push the result onto the stack.

**Name:** btnSub- perform subtraction .

**Input:** Top two elements in the stack.

**Output:** None.

**Logic:** Get the top two elements from the stack and subtract them. Then push the result onto the stack.

**Name:** btnMul- perform multiplication.

**Input:** Top two elements in the stack.

**Output:** None.

**Logic:** Get the top two elements from the stack and multiply them. Then push the result onto the stack.

**Name:** btnDiv- perform division .

**Input:** Top two elements in the stack.

**Output:** None.

**Logic:** Get the top two elements from the stack and divide them. Then push the result onto the stack.

**Name:** btnDot- insert a decimal point.

**Input:** A number.

**Output:** Number with a decimal point.

**Logic:** Append the input number with a decimal.

**Name:** btnPostNeg- change a positive number to a negative.

**Input:** A number.

**Output:** A negative number.

**Logic:** Add a negative sign in the input field.

**Name:** btnEnter - display the result.

**Input:** The remaining element in the stack.

**Output:** Result.

**Logic:** Pop the remaining element from the stack and display.

**Name:** main- Launch the program.

**Input:** None.

**Output:** None.

**Logic:** Evaluate postfix expressions.

**Data:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Structure** |
| **operand1** | **Double** | **Variable** |
| **operand1** | **Double** | **Variable** |
| **result** | **Double** | **Variable** |
| **calStack** | **String** | **Stack** |

**3.3 Algorithm**

Logic

1.0 Display the GUI.

2.0 Create the stack.

3.0 Get input from the user.

3.1 If the input is a number push it onto the stack.

3.2 If the input is an operator, perform that operator onto the top two elements in the stack.

4.0 Repeat step 3 until the expression is evaluated.

5.0 Display the result by getting the last element remaining in the stack.

Algorithm Description

The program starts by displaying a calculator like GUI. The GUI contains two input fields, number buttons, operator buttons , buttonto clear the input and a buttonto push the input onto the stack. The program gets the input when the user clicks on the Enter button. Each time the user wants to enter a number they must click on the Enter button. Clicking the Enter button pushes the input number onto the stack. When an operator button is clicked the program gets the top two elements in the stack and evaluates them based on which operator button was clicked. The result from this evaluation is pushed back onto the stack. This process repeats every time the user clicks the operator button. The result is display when the user clicks on the Equal(=) button. To display the result, the remaining element is extracted from the stack and displayed onto the screen.

**4 Translation**

**4.1 Source Code**

//================================================================

// Name : Nidhi Patel

// SID : 31379144

// Course : IT114

// Section : 452

// Instructor : Maura Deek

// Assignment # : Programming Assignment 4

// Date : 04/08/2016

// Description : This program evaluates postfix expressions.

//=================================================================

//Import statements

import javafx.application.Application;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.stage.Stage;

import javafx.scene.control.TextField;

import javafx.scene.layout.GridPane;

import javafx.scene.layout.ColumnConstraints;

import javafx.geometry.Insets;

import javafx.event.EventHandler;

import javafx.event.ActionEvent;

import java.util.\*;

//PostFixExpression class

public class PostfixExpression extends Application {

double operand1, //Stores the first operand

operand2, //Stores the second operand

result; //Stored the results

//main method to launch the program

public static void main(String[]args){

launch(args);

}

//Overridden start method which creates the GUI

public void start(Stage primaryStage) throws Exception{

//Set styling properties for the stage

primaryStage.setTitle("Postfix Expression");

primaryStage.setResizable(false);

//Instantiate an object of GridPane

GridPane layout = new GridPane();

//Add styling properties to the stage

layout.setPadding(new Insets(15));

layout.setHgap(5);

layout.setVgap(5);

//Set width for each column in the grid

layout.getColumnConstraints().add(new ColumnConstraints(50));

layout.getColumnConstraints().add(new ColumnConstraints(50));

layout.getColumnConstraints().add(new ColumnConstraints(50));

layout.getColumnConstraints().add(new ColumnConstraints(50));

//Create a TextField to display output

TextField inputTxt1 = new TextField();

inputTxt1.setEditable(false);

//Create a TextField to display the selected operand

TextField inputTxt = new TextField();

inputTxt.setEditable(false);

//Create buttons

Button btn0 = new Button("0");

Button btn1 = new Button("1");

Button btn2 = new Button("2");

Button btn3 = new Button("3");

Button btn4 = new Button("4");

Button btn5 = new Button("5");

Button btn6 = new Button("6");

Button btn7 = new Button("7");

Button btn8 = new Button("8");

Button btn9 = new Button("9");

Button btnAdd = new Button("+");

Button btnSub = new Button("-");

Button btnMul = new Button("\*");

Button btnDiv = new Button("/");

Button btnClear = new Button("Clear");

Button btnEqual = new Button("=");

Button btnPosNeg = new Button("+/-");

Button btnDot = new Button(".");

Button btnEnter = new Button("Enter");

//Set styling properties for the buttons

btn0.setMaxWidth(Double.MAX\_VALUE);

btn1.setMaxWidth(Double.MAX\_VALUE);

btn2.setMaxWidth(Double.MAX\_VALUE);

btn3.setMaxWidth(Double.MAX\_VALUE);

btn4.setMaxWidth(Double.MAX\_VALUE);

btn5.setMaxWidth(Double.MAX\_VALUE);

btn6.setMaxWidth(Double.MAX\_VALUE);

btn7.setMaxWidth(Double.MAX\_VALUE);

btn8.setMaxWidth(Double.MAX\_VALUE);

btn9.setMaxWidth(Double.MAX\_VALUE);

btnClear.setMaxWidth(Double.MAX\_VALUE);

btnEqual.setMaxWidth(Double.MAX\_VALUE);

btnAdd.setMaxWidth(Double.MAX\_VALUE);

btnSub.setMaxWidth(Double.MAX\_VALUE);

btnMul.setMaxWidth(Double.MAX\_VALUE);

btnDiv.setMaxWidth(Double.MAX\_VALUE);

btnDot.setMaxWidth(Double.MAX\_VALUE);

btnPosNeg.setMaxWidth(Double.MAX\_VALUE);

btnEnter.setMaxWidth(Double.MAX\_VALUE);

//Add the UI elements to the layout

layout.add(inputTxt,0,1,4,1);

layout.add(btnClear,0,2,2,1);

layout.add(btnPosNeg,2,2);

layout.add(btnEnter,0,6);

layout.add(btnDiv,3,2);

layout.add(btn7,0,3);

layout.add(btn8,1,3);

layout.add(btn9,2,3);

layout.add(btnMul,3,3);

layout.add(btn4,0,4);

layout.add(btn5,1,4);

layout.add(btn6,2,4);

layout.add(btnSub,3,4);

layout.add(btn1,0,5);

layout.add(btn2,1,5);

layout.add(btn3,2,5);

layout.add(btnAdd,3,5);

layout.add(btn0,1,6);

layout.add(btnDot,2,6);

layout.add(btnEqual,3,6);

layout.add(inputTxt1,0,0,4,1);

//Create a new stack

Stack<Double> calStack = new Stack<Double>();

//Handle the events related to btn0

btn0.setOnAction(new EventHandler<ActionEvent>(){

public void handle(ActionEvent event){

inputTxt.appendText("0");

}

});

//Handle the events related to btn1

btn1.setOnAction(new EventHandler<ActionEvent>(){

public void handle(ActionEvent event){

inputTxt.appendText("1");

}

});

//Handle the events related to the btn2

btn2.setOnAction(new EventHandler<ActionEvent>(){

public void handle(ActionEvent event){

inputTxt.appendText("2");

}

});

//Handle the events related to the btn3

btn3.setOnAction(new EventHandler<ActionEvent>(){

public void handle(ActionEvent event){

inputTxt.appendText("3");

}

});

//Handle the events related to the btn4

btn4.setOnAction(new EventHandler<ActionEvent>(){

public void handle(ActionEvent event){

inputTxt.appendText("4");

}

});

//Handle the events related to the btn5

btn5.setOnAction(new EventHandler<ActionEvent>(){

public void handle(ActionEvent event){

inputTxt.appendText("5");

}

});

//Handle the events related to the btn6

btn6.setOnAction(new EventHandler<ActionEvent>(){

public void handle(ActionEvent event){

inputTxt.appendText("6");

}

});

//Handle the events related to the btn7

btn7.setOnAction(new EventHandler<ActionEvent>(){

public void handle(ActionEvent event){

inputTxt.appendText("7");

}

});

//Handle the events related to the btn8

btn8.setOnAction(new EventHandler<ActionEvent>(){

public void handle(ActionEvent event){

inputTxt.appendText("8");

}

});

//Handle the events related to the btn9

btn9.setOnAction(new EventHandler<ActionEvent>(){

public void handle(ActionEvent event){

inputTxt.appendText("9");

}

});

//Handle the events related to the btnDiv

btnDiv.setOnAction(new EventHandler<ActionEvent>(){

public void handle(ActionEvent event){

inputTxt1.appendText("/");

operand1 = calStack.pop();

operand2 = calStack.pop();

result = operand2 / operand1;

calStack.push(result);

System.out.println(calStack);

}

});

//Handle the events related to the btnMul

btnMul.setOnAction(new EventHandler<ActionEvent>(){

public void handle(ActionEvent event){

inputTxt1.appendText("\*");

operand1 = calStack.pop();

operand2 = calStack.pop();

result = operand1 \* operand2;

calStack.push(result);

System.out.println(calStack);

}

});

//Handle the events related to the btnAdd

btnAdd.setOnAction(new EventHandler<ActionEvent>(){

public void handle(ActionEvent event){

inputTxt1.appendText("+");

operand1 = calStack.pop();

operand2 = calStack.pop();

result = operand1 + operand2;

calStack.push(result);

System.out.println(calStack);

}

});

//Handle the events related to the btnSub

btnSub.setOnAction(new EventHandler<ActionEvent>(){

public void handle(ActionEvent event){

inputTxt1.appendText("-");

operand1 = calStack.pop();

operand2 = calStack.pop();

result = operand2 - operand1;

calStack.push(result);

System.out.println(calStack);

}

});

//Handle the events related to the btnPosNeg

btnPosNeg.setOnAction(new EventHandler<ActionEvent>(){

public void handle(ActionEvent event){

inputTxt.setText("-");

}

});

//Handle the events related to the btnDot

btnDot.setOnAction(new EventHandler<ActionEvent>(){

public void handle(ActionEvent event){

inputTxt.appendText( ".");

}

});

//Handle the events related to the btnClear

btnClear.setOnAction(new EventHandler<ActionEvent>(){

public void handle(ActionEvent event){

for(int i=calStack.size(); i > 0;i--)

{

calStack.pop();

}

inputTxt.setText("");

inputTxt1.setText("");

}

});

//Handle the events related to the btnEqual

btnEqual.setOnAction(new EventHandler<ActionEvent>(){

public void handle(ActionEvent event){

inputTxt.setText(Double.toString(calStack.pop()));

System.out.println(calStack);

}

});

//Handle the events related to the btnEnter

btnEnter.setOnAction(new EventHandler<ActionEvent>(){

public void handle(ActionEvent event){

inputTxt1.appendText(inputTxt.getText() + " ");

calStack.push(Double.parseDouble(inputTxt.getText()));

inputTxt.setText("");

}

});

//Create a scene

Scene scene = new Scene(layout);

//Set the scene onto the stage

primaryStage.setScene(scene);

//Display the stage

primaryStage.show();

}

}

**4.2 Program and Module Description**

**btnClear** - Clears the stack and the input fields. Clears the stack by using the pop method and a loop.

**btnEqual:** - Display the result by using the pop method to extract the last element remaining in the stack.

**btnAdd**- Add the top two operands in the stack and push the result back onto the stack.

**btnSub**- Subtract the top two operands in the stack and push the result back onto the stack.

**btnMul**- Multiply the top two operands in the stack and push the result back onto the stack.

**btnDiv**- Divide the top two operands in the stack and push the result back onto the stack.

**btnDot**- Allows the user to enter a decimal.

**btnPostNeg**- Allows the user to enter a negative number.

**btnEnter** - Push the input number onto the stack.

**main**- Launch the program.

**5. Solution Testing**

**Test Case 1**

|  |  |
| --- | --- |
| **Operand** | **Button to click** |
| **2** | **Enter** |
| **10** | **Enter** |
|  | **+** |
| **9** | **Enter** |
| **6** | **Enter** |
|  | **-** |
|  | **/** |
|  | **Equal** |
| **Answer = 4 .0** | |

**Test Case 2**

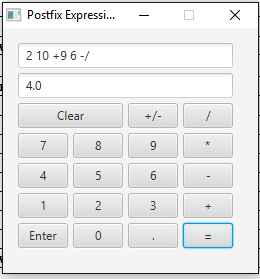
|  |  |
| --- | --- |
| **Operand** | **Button to click** |
|  | **Clear** |
| **1** | **Enter** |
| **2** | **Enter** |
|  | **+** |
| **3** | **Enter** |
|  | **\*** |
|  | **Equal** |
| **Answer = 9.0** | |

**Test Case 3**

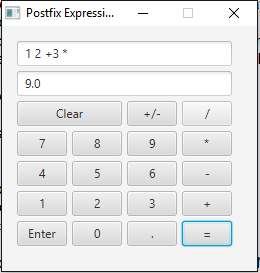
|  |  |
| --- | --- |
| **Operand** | **Button to click** |
|  | **Clear** |
| **10** | **Enter** |
|  | **+/-** |
| **5** | **Enter** |
|  | **+** |
|  | **Equal** |
| **Answer = 5.0** | |

**6 Testing Output**

**Test Case 1**



**Test Case 2**



**Test Case 3**

