**Tutorial No 4**

**AIM:** To develop the calculator based on the MVC architecture.

**Theory:**

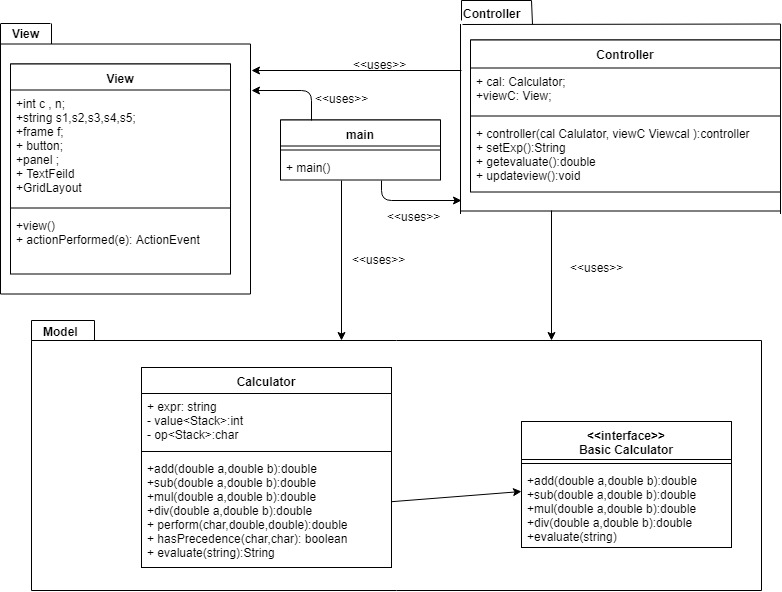
MVC Pattern stands for Model-View-Controller Pattern. This pattern is used to separate application's concerns.

* **Model** - Model represents an object or JAVA POJO carrying data. It can also have logic to update controller if its data changes.
* **View** - View represents the visualization of the data that model contains.
* **Controller** - Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separate.

The **Model View Controller** (MVC) design pattern specifies that an application consist of a data model, presentation information, and control information. The pattern requires that each of these be separated into different objects.

MVC is more of an architectural pattern, but not for complete application. MVC mostly relates to the UI / interaction layer of an application. You’re still going to need business logic layer, maybe some service layer and data access layer.

**Class Diagram:**

****

Code:-

**Controller.java:-**

package controller;

import java.util.Stack;

import java.awt.event.\*;

import javax.swing.\*;

import Model.CalculatorModel;

import view.\*;

public class Controller{

private CalculatorModel m=new CalculatorModel();

private JFrameView v1=new JFrameView();

private CommandLine v2=new CommandLine();

//one more view...

//set all views and user can enter expression in any one of them

public Controller(CalculatorModel m,JFrameView v1,CommandLine v2)

{

//one more arguments of view ...

this.m=m;

this.v1=v1;

this.v2=v2;

//this.v3=v3;

v1.getUserInput();

this.getExpresssion();

// v2.getUserInput();

//v1.getUserInput();

}

//get expression whenever any view changes...

public void getExpresssion() {

while(true) {

if(v1.isExpressionReady)

{

m.setExpression(v1.expression); //Change model

m.setResult(solve(v1.expression));

System.out.println("getting processed");

v1.isExpressionReady=false;

this.updateView();

}

if(v2.isExpressionReady)

{

m.setExpression(v2.expression);//Change model

m.setResult(solve(v2.expression));

System.out.println("getting processed");

v2.isExpressionReady=false;

this.updateView();

}

/\* if(v3.isExpressionReady)

{

m.setExpression(v3.expression);

m.setResult(solve(v3.expression));

this.updateView();

v3.isExpressionReady=false;

}\*/

}

}

private void updateView() {

v1.setResult(m.getResult());

v2.setResult(m.getResult());

//v3.setResult(m.getResult());

v1.getUserInput();

v2.getUserInput();

}

public double solve(String input){

char[] tokens = input.toCharArray();

// Stack for numbers

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

// Stack for Operators

Stack<Character> ops = new Stack<Character>();

for (int i = 0; i < tokens.length; i++)

{

if (tokens[i] == ' ')

continue;

// Current token is a number, push it to stack for numbers

if (tokens[i] >= '0' && tokens[i] <= '9')

{

StringBuffer sbuf = new StringBuffer();

// There may be more than one digits in number

while (i < tokens.length && tokens[i] >= '0' && tokens[i] <= '9')

sbuf.append(tokens[i++]);

values.push(Double.parseDouble(sbuf.toString()));

}

else if (tokens[i] == '(')

ops.push(tokens[i]);

else if (tokens[i] == ')')

{

while (ops.peek() != '(')

values.push(applyOp(ops.pop(), values.pop(), values.pop()));

ops.pop();

}

else if (tokens[i] == '+' || tokens[i] == '-' ||

tokens[i] == '\*' || tokens[i] == '/')

{

// While top of 'ops' has same or greater precedence to current

// token, which is an operator. Apply operator on top of 'ops'

// to top two elements in values stack

while (!ops.empty() && hasPrecedence(tokens[i], ops.peek()))

values.push(applyOp(ops.pop(), values.pop(), values.pop()));

ops.push(tokens[i]);

}

}

// Entire expression has been parsed at this point, apply remaining ops to remaining values

while (!ops.empty())

values.push(applyOp(ops.pop(), values.pop(), values.pop()));

// Top of 'values' contains result, return it

return values.pop();

}

public static boolean hasPrecedence(char op1, char op2)

{

if (op2 == '(' || op2 == ')')

return false;

if((op1=='^') && (op2=='\*'|| op2=='/' || op2=='+' || op2=='-'))

return false;

if ((op1 == '\*' || op1 == '/') && (op2 == '+' || op2 == '-'))

return false;

else

return true;

}

public static Double applyOp(char op, Double b, Double a)

{

switch (op)

{

case '+':

return a+b;

case '-':

return a-b;

case '\*':

return a\*b;

case '/':

if (b == 0)

throw new

UnsupportedOperationException("Cannot divide by zero");

return a/b;

case '^':

return Math.pow(a, b);

}

return 0.00;

}

}

**CalculatorModel.java:-**

**package** Model;

**public** **class** CalculatorModel {

**private** **double** result;

**private** String expression;

**public** **void** setResult(**double** r) {

**this**.result=r;

}

**public** **void** setExpression(String e) {

**this**.expression=e;

}

**public** **double** getResult() {

**return** **this**.result;

}

**public** String getExpression() {

**return** **this**.expression;

}

}

**MVCdemo.java:-**

**package** Model;

**public** **class** CalculatorModel {

**private** **double** result;

**private** String expression;

**public** **void** setResult(**double** r) {

**this**.result=r;

}

**public** **void** setExpression(String e) {

**this**.expression=e;

}

**public** **double** getResult() {

**return** **this**.result;

}

**public** String getExpression() {

**return** **this**.expression;

}

}

**Commandline.java:-**

/\*\*

\*

\*/

package view;

/\*\*

\* @author PRANAY

\*

\*/

import java.util.Scanner;

public class CommandLine {

public String expression;

public boolean isExpressionReady=false;

public void getUserInput() {

Scanner sc=new Scanner(System.in);

System.out.println("Enter Expression to evaluate(Enter 'exit' to exit):");

expression =sc.nextLine();

if(expression.equals("exit"))

{

System.exit(0);

sc.close();

}

isExpressionReady=true;

System.out.println("expression set");

}

public void setResult(double r) {

System.out.println(r);

}

}

**Jframe.java:-**

**package view;**

**import java.awt.event.ActionEvent;**

**import java.awt.event.ActionListener;**

**import java.util.Scanner;**

**import javax.swing.\*;**

**import controller.\*;**

**public class JFrameView implements ActionListener{**

**public String expression;**

**JFrame f;**

**public JTextField t=new JTextField();**

**public boolean isExpressionReady=false;**

**JButton bclear,bresult,b1,b2,b3,b4,b5,b6,b7,b8,b9,b0,bplus,bminus,bdiv,bmul;**

**//double result;**

**JPanel panel;**

**public JFrameView() {**

**f=new JFrame("Calculator");**

**bclear=new JButton("clear");**

**bresult=new JButton("result");**

**b1=new JButton("1 ");**

**b2=new JButton("2 ");**

**b3=new JButton("3 ");**

**b4=new JButton("4 ");**

**b5=new JButton("5 ");**

**b6=new JButton("6 ");**

**b7=new JButton("7 ");**

**b8=new JButton("8 ");**

**b9=new JButton("9 ");**

**b0=new JButton("0 ");**

**bplus=new JButton("+ ");**

**bminus=new JButton("- ");**

**bdiv=new JButton("/ ");**

**bmul=new JButton("\* ");**

**f.setLayout(null);**

**f.setVisible(true);**

**f.setSize(1000,500);**

**f.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);**

**f.setResizable(true);**

**t.setBounds(40,30,500,40);**

**b1.setBounds(40,100, 50,20);**

**b2.setBounds(140,100, 50, 20);**

**b3.setBounds(240,100, 50, 20);**

**b4.setBounds(40,130, 50,20);**

**b5.setBounds(140,130, 50, 20);**

**b6.setBounds(240,130, 50, 20);**

**b7.setBounds(40,160, 50,20);**

**b8.setBounds(140,160, 50, 20);**

**b9.setBounds(240,160, 50, 20);**

**b0.setBounds(140,190, 50, 20);**

**bplus.setBounds(40,220, 50, 20);**

**bminus.setBounds(100,220, 50, 20);**

**bdiv.setBounds(160,220, 50, 20);**

**bmul.setBounds(220,220, 50, 20);**

**bclear.setBounds(40,250,100,40);**

**bresult.setBounds(170,250,100,40);**

**f.add(b1);**

**f.add(b2);**

**f.add(b3);**

**f.add(b4);**

**f.add(b5);**

**f.add(b6);**

**f.add(b7);**

**f.add(b8);**

**f.add(b9);**

**f.add(b0);**

**f.add(bplus);**

**f.add(bminus);**

**f.add(bdiv);**

**f.add(bmul);**

**f.add(t);**

**f.add(bclear);**

**f.add(bresult);**

**}**

**public void getUserInput() {**

**b1.addActionListener(this);**

**b2.addActionListener(this);**

**b3.addActionListener(this);**

**b4.addActionListener(this);**

**b5.addActionListener(this);**

**b6.addActionListener(this);**

**b7.addActionListener(this);**

**b8.addActionListener(this);**

**b9.addActionListener(this);**

**b0.addActionListener(this);**

**bplus.addActionListener(this);**

**bminus.addActionListener(this);**

**bdiv.addActionListener(this);**

**bmul.addActionListener(this);**

**bclear.addActionListener(this);**

**bresult.addActionListener(this);;**

**}**

**public void actionPerformed(ActionEvent e)**

**{**

**System.out.println("Inside actionPerformed");**

**if(e.getSource()==bclear)**

**t.setText("");**

**if(e.getSource()==bresult)**

**{**

**expression=t.getText();**

**this.isExpressionReady=true;**

**System.out.println(expression);**

**}**

**if(e.getSource()==b1)**

**{**

**t.setText(t.getText().concat("1 "));**

**}**

**if(e.getSource()==b2)**

**{**

**t.setText(t.getText().concat("2 "));**

**}**

**if(e.getSource()==b3)**

**{**

**t.setText(t.getText().concat("3 "));**

**}**

**if(e.getSource()==b4)**

**{**

**t.setText(t.getText().concat("4 "));**

**}**

**if(e.getSource()==b5)**

**{**

**t.setText(t.getText().concat("5 "));**

**}**

**if(e.getSource()==b6)**

**{**

**t.setText(t.getText().concat("6 "));**

**}**

**if(e.getSource()==b7)**

**{**

**t.setText(t.getText().concat("7 "));**

**}**

**if(e.getSource()==b8)**

**{**

**t.setText(t.getText().concat("8 "));**

**}**

**if(e.getSource()==b9)**

**{**

**t.setText(t.getText().concat("9 "));**

**}**

**if(e.getSource()==b0)**

**{**

**t.setText(t.getText().concat("0 "));**

**}**

**if(e.getSource()==bplus)**

**{**

**t.setText(t.getText().concat("+ "));**

**}**

**if(e.getSource()==bminus)**

**{**

**t.setText(t.getText().concat("- "));**

**}**

**if(e.getSource()==bdiv)**

**{**

**t.setText(t.getText().concat("/ "));**

**}**

**if(e.getSource()==bmul)**

**{**

**t.setText(t.getText().concat("\* "));**

**}**

**}**

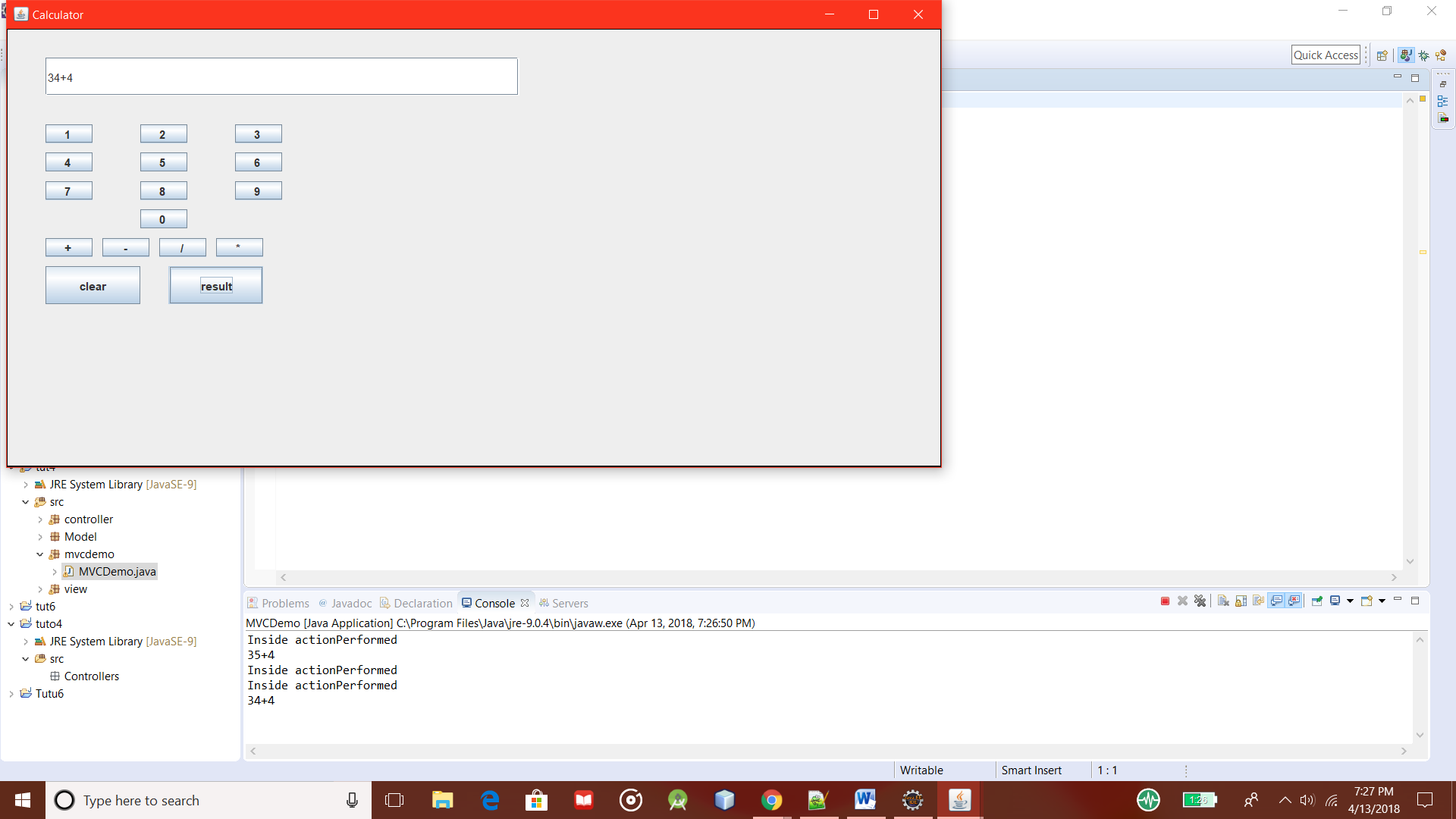
**public void setResult(double r) {**

**t.setText(String.valueOf(r));**

**}**

**}**

**Input & Output:**



**Observation:**

Here the Model-View-Controller Model has been used as an architectural style. The MVC Pattern highlights the working of a highly abstract model structure which is quite complex to implement but very helpful to implement various models, view and controller where multiple developers can work simultaneously on the model, controller and views. Also MVC enables logical grouping of related actions on a controller together. The views for a specific model are also grouped together. Similarly various views can be implemented like web HTML page, Jframe or Console. Thus a model can have many views.

**Conclusion:**

Thus we have implemented the MVC pattern based calculator and learned its use through implementing it in various views and providing the basic functionality in the architecture.