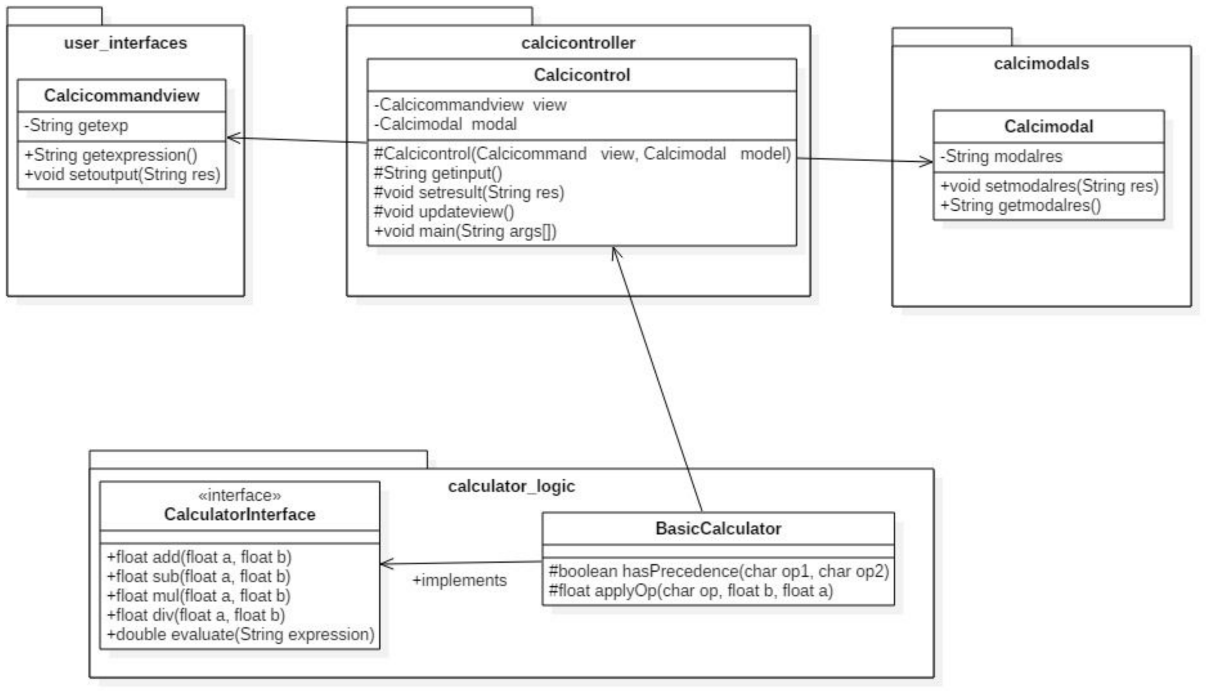
Tutorial No.4

A. Apply MVC(Modal View Controller) design pattern on Basic Calculations operations where command prompt as view , calcicontrol as controller and calcimodal as Modal .

Class diagram:


user_interfaces
Calcicommandview
-String getexp
+String getexpression()
+void setoutput(String res)
calcicontroller
Calcicontrol
-Calcicommandview view
-Calcimodal modal
#Calcicontrol(Calcicommand view, Calcimodal model)
#String getinput()
#void setresult(String res)
#void updateview()
+void main(String args[])
calcimodals
Calcimodal
-String modalres
+void setmodalres(String res)
+String getmodalres()
calculator_logic
«interface»
CalculatorInterface
+float add(float a, float b)
+float sub(float a, float b)
+float mul(float a, float b)
+float div(float a, float b)
+double evaluate(String expression)
+implements
BasicCalculator
#boolean hasPrecedence(char op1, char op2)
#float applyOp(char op, float b, float a)

View Class:

```
package user_interfaces;
import java.util.*;
public class Calcicommandview {
Scanner scan=new Scanner(System.in);
private String getexp;
public String getexpression()
{
        System.out.print("Enter Expression:");
        getexp=scan.nextLine();
```

```
          return getexp;
}
public void setoutput(String res)
{
          System.out.println(res);
}
}
```

Controller class:

```
    package calcicontroller;
import user_interfaces.*;
import calcimodals.*;
import  calculator_logic.*;
public class Calcicontrol {
 private Calcicommandview  view;
 private Calcimodal  modal;
 CalculatorInterface cal =new BasicCalculator();
protected  Calcicontrol(Calcicommandview view, Calcimodal modal)
 {
           this.view=view;
           this.modal=modal;
 }
   protected String getinput()
   {
            return view.getexpression();
   }
   protected void setresult(String exp)
   {
            modal.setmodalres(Double.toString(cal.evaluate(exp))) ;
   }
 protected void updateview()
 {
           view.setoutput(modal.getmodalres());
 }

          public static void main(String args[])
          {
            Calcicommandview view=new Calcicommandview();
            Calcimodal modal=new Calcimodal();
            Calcicontrol controller=new Calcicontrol(view,modal);
            controller.setresult( controller.getinput());
            controller.updateview();



          }

}
```

Modal Class:

```
package calcimodals;

public class Calcimodal {
private String modalres;
public void setmodalres(String res)
{
          modalres=res;
}
public  String getmodalres() {
          return modalres;
}
}
```

//Calculator Interface

```
package calculator_logic;
public interface CalculatorInterface {
          public float add(float a,float b);
          public float sub(float a,float b);
          public float mul(float a,float b);
          public float div(float a,float b);
          public double evaluate(String expression);

}
```

//Basic Calculator class

```
package calculator_logic;
import java.util.Stack;
 public class BasicCalculator implements CalculatorInterface {
	@Override
	public float add(float a, float b) {
		// TODO Auto-generated method stub
		return a+b;
	}

	@Override
	public float sub(float a, float b) {
		// TODO Auto-generated method stub
		return a-b;
	}

	@Override
	public float mul(float a, float b) {
		// TODO Auto-generated method stub
		return a*b;
	}

	@Override
	public float div(float a, float b) {
		// TODO Auto-generated method stub
		return b/a;
	}

	public double evaluate(String expression)
	{
		Stack<Float> values;
		Stack<Character> ops;
		char[] tokens;
		tokens = expression.toCharArray();
		values = new Stack<>();
		ops = new Stack<Character>();
		for (int i = 0; i < tokens.length; i++)
		{
			if (tokens[i] >= '0' && tokens[i] <= '9')
```

```
            {
                StringBuffer sbuf = new StringBuffer();
                while (i < tokens.length && tokens[i] >= '0' && tokens[i] <=
'9')
                    sbuf.append(tokens[i++]);
                values.push(Float.parseFloat(sbuf.toString()));
                i--;
            }
            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 (!ops.empty() && hasPrecedence(tokens[i],
ops.peek()))
                values.push(applyOp(ops.pop(), values.pop(),
values.pop()));
                ops.push(tokens[i]);
            }
        }
        while (!ops.empty())
            values.push(applyOp(ops.pop(), values.pop(), values.pop()));
        return values.pop();
    }
    protected static boolean hasPrecedence(char op1, char op2)
    {
        if (op2 == '(' || op2 == ')')
            return false;
        if ((op1 == '*' || op1 == '/') && (op2 == '+' || op2 == '-'))
            return false;
        else
            return true;
        }
        protected float applyOp(char op, float b, float a)
        {
                switch (op)
                {
                case '+':
                        return add(a,b);
                case '-':
                        return sub(a,b);
                case '*':
                        return mul(a,b);
                case '/':
                        if (b != 0)
                                return div(a,b);
                }
                return 0;
        }

}
```