UNIVERSITI MALAYSIA SABAH

Fakulti Komputeran dan Informatik

SEMESTER II

SESSION 2019/2020

Individual Project

KP14603 OBJECT ORIENTED PROGRAMMING CONCEPT

LECTURER :SITI HASNAH BINTI TANALOL

NAME : VEYNNISHAA A/P KUMARAPPAN

MATRIC NO : BI19160308

INTRODUCTION

GUI stands for Graphical User Interface, a term used not only in Java but in all programming languages that support the development of GUIs. A program's graphical user interface presents an easy-to-use visual display to the user. It is made up of graphical components and varies based on the type of functions such as buttons, labels, windows) through which the user can interact with the page or application.

A major part of creating a graphical user interface in Java is figuring out how to position and lay out the components of the user interface to match the appearance you desire.

Calculator is a device that performs arithmetic operation on numbers. In this project I have done a simple calculator that can do perform calculation involving addition, subtraction, multiplication, and division.

JAVA CODE

```
//NAME : VEYNNISHAA A/P KUMARAPPAN
//NO MATRIC : BI19160308
//PROJECT TITLE : VENNY SIMPLE CALCULATOR

package simplecalculator;
import java.awt.event.*;
import javax.swing.*;
import java.awt.*;

class calculator extends JFrame implements ActionListener {
   // create a frame
   static JFrame f;

   // create a textfield
   static JTextField l;

   // store oprerator and operands
   String s0, s1, s2;

   // default constrcutor
   calculator()
   {
      s0 = s1 = s2 = "";
   }

   // main function
   public static void main(String args[])
   {
      // create a frame
```

```
        f = new JFrame("calculator");

        try {
            // set look and feel
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        }
        catch   (ClassNotFoundException   |   IllegalAccessException   |   InstantiationException   |
UnsupportedLookAndFeelException e) {
            System.err.println(e.getMessage());
        }

        // create a object of class
        calculator c = new calculator();

        // create a textfield
        l = new JTextField(16);

        // set the textfield to non editable
        l.setEditable(false);

        // create number buttons and some operators
        JButton b0, b1, b2, b3, b4, b5, b6, b7, b8, b9, ba, bs, bd, bm, be, beq, beq1;

        // create number buttons
        b0 = new JButton("0");
        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");
```

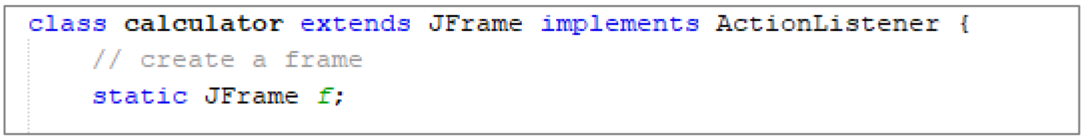
```
b8 = new JButton("8");
b9 = new JButton("9");

// equals button
beq1 = new JButton("=");

// create operator buttons
ba = new JButton("+");
bs = new JButton("-");
bd = new JButton("/");
bm = new JButton("*");
beq = new JButton("C");

// create . button
be = new JButton(".");

// create a panel
JPanel p = new JPanel();

// add action listeners
bm.addActionListener(c);
bd.addActionListener(c);
bs.addActionListener(c);
ba.addActionListener(c);
b9.addActionListener(c);
b8.addActionListener(c);
b7.addActionListener(c);
b6.addActionListener(c);
b5.addActionListener(c);
b4.addActionListener(c);
b3.addActionListener(c);
b2.addActionListener(c);
```

```
b1.addActionListener(c);
b0.addActionListener(c);
be.addActionListener(c);
beq.addActionListener(c);
beq1.addActionListener(c);

// add elements to panel
p.add(l);
p.add(ba);
p.add(b1);
p.add(b2);
p.add(b3);
p.add(bs);
p.add(b4);
p.add(b5);
p.add(b6);
p.add(bm);
p.add(b7);
p.add(b8);
p.add(b9);
p.add(bd);
p.add(be);
p.add(b0);
p.add(beq);
p.add(beq1);

// set Background of panel
p.setBackground(Color.pink);

// add panel to frame
f.add(p);
```

```
    f.setSize(200, 220);
    f.show();
}
public void actionPerformed(ActionEvent e)
{
    String s = e.getActionCommand();

    // if the value is a number
    if ((s.charAt(0) >= '0' && s.charAt(0) <= '9') || s.charAt(0) == '.') {
        // if operand is present then add to second no
        if (!s1.equals(""))
            s2 = s2 + s;
        else
            s0 = s0 + s;

        // set the value of text
        l.setText(s0 + s1 + s2);
    }
    else if (s.charAt(0) == 'C') {
        // clear the one letter
        s0 = s1 = s2 = "";

        // set the value of text
        l.setText(s0 + s1 + s2);
    }
    else if (s.charAt(0) == '=') {

        double te;

        // store the value in 1st
        switch (s1) {
            case "+":
                te = (Double.parseDouble(s0) + Double.parseDouble(s2));
                break;
            case "-":
                te = (Double.parseDouble(s0) - Double.parseDouble(s2));
                break;
            case "/":
                te = (Double.parseDouble(s0) / Double.parseDouble(s2));
                break;
            default:
                te = (Double.parseDouble(s0) * Double.parseDouble(s2));
                break;
        }

        // set the value of text
        l.setText(s0 + s1 + s2 + "=" + te);

        // convert it to string
        s0 = Double.toString(te);

        s1 = s2 = "";
    }
    else {
        // if there was no operand
        if (s1.equals("") || s2.equals(""))
            s1 = s;
        // else evaluate
        else {
            double te;

            // store the value in 1st
            switch (s1) {
                case "+":
                        te = (Double.parseDouble(s0) + Double.parseDouble(s2));
                        break;
                    case "-":
                        te = (Double.parseDouble(s0) - Double.parseDouble(s2));
                        break;
                    case "/":
                        te = (Double.parseDouble(s0) / Double.parseDouble(s2));
                        break;
                    default:
                        te = (Double.parseDouble(s0) * Double.parseDouble(s2));
                        break;
                }

                // convert it to string
                s0 = Double.toString(te);

                // place the operator
                s1 = s;

                // make the operand blank
                s2 = "";
            }

            // set the value of text
            l.setText(s0 + s1 + s2);
        }
    }
}
```

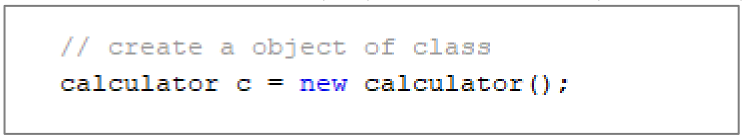
Object Oriented Concept Implementation

1. Class
CLASS are a blueprint or a set of instructions to build a specific type of object. It is a basic concept of Object-Oriented Programming which revolve around the real-life entities. Class in Java determines how an object will behave and what the object will contain



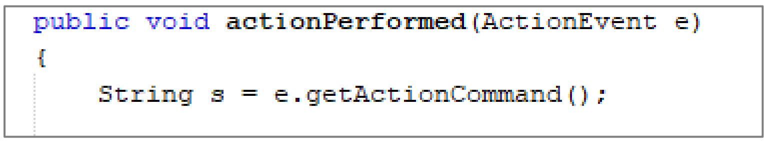
```java
class calculator extends JFrame implements ActionListener {
    // create a frame
    static JFrame f;
```

2. Object
OBJECT is an instance of a class. An object is nothing but a self-contained component which consists of methods and properties to make a particular type of data useful.



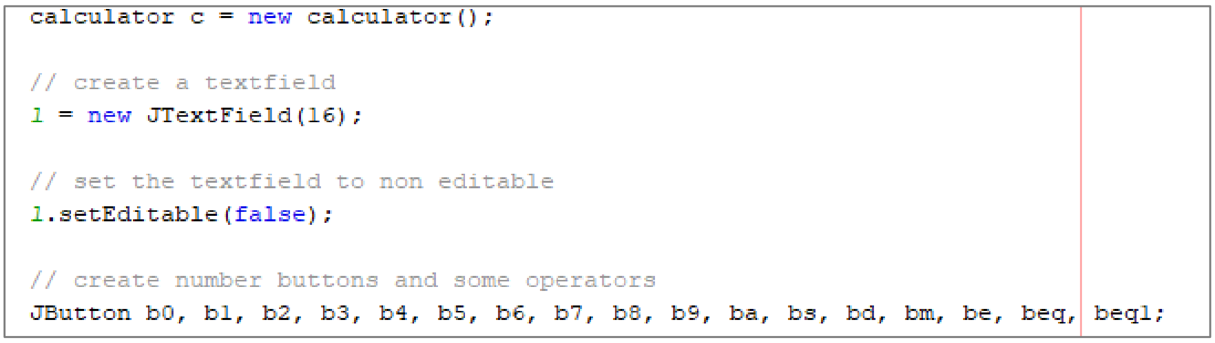
```java
    // create a object of class
    calculator c = new calculator();
```

3. Encapsulation
Encapsulation in Java is a mechanism to wrap up variables(data) and methods(code) together as a single unit. It is the process of hiding information details and protecting data and behavior of the object. It is one of the four important OOP concepts.



```java
public void actionPerformed(ActionEvent e)
{
    String s = e.getActionCommand();
```

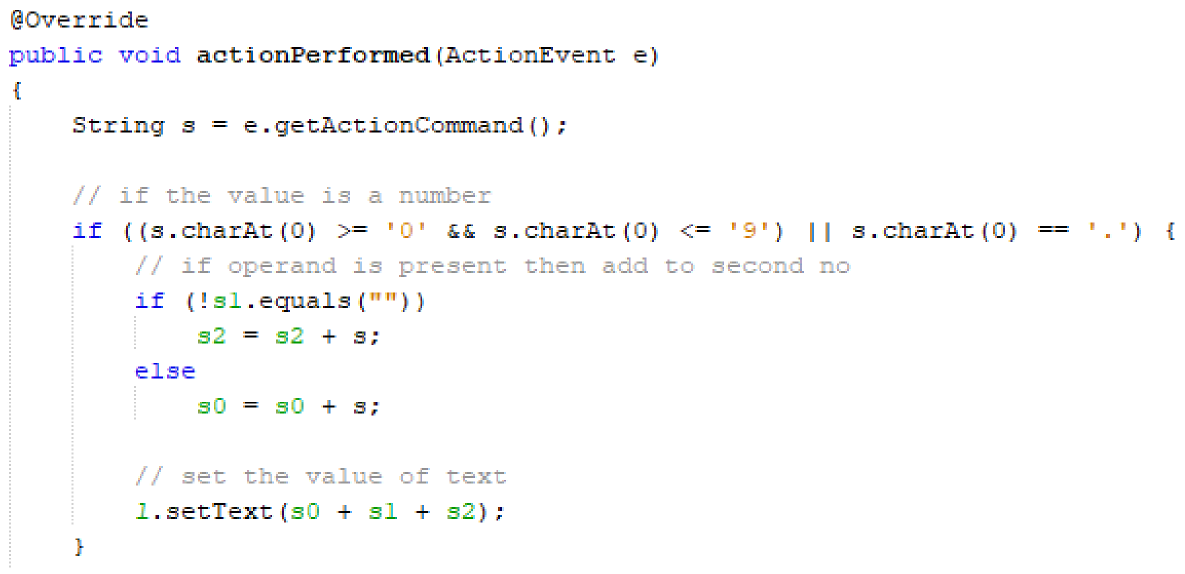
4. Abstraction
Abstraction means using simple things to represent complexity. In Java, abstraction means simple things like objects, classes, and variables represent more complex underlying code and data. This is important because it lets avoid repeating the same work multiple times.



```java
calculator c = new calculator();

// create a textfield
l = new JTextField(16);

// set the textfield to non editable
l.setEditable(false);

// create number buttons and some operators
JButton b0, b1, b2, b3, b4, b5, b6, b7, b8, b9, ba, bs, bd, bm, be, beq, beq1;
```

5. Polymorphism

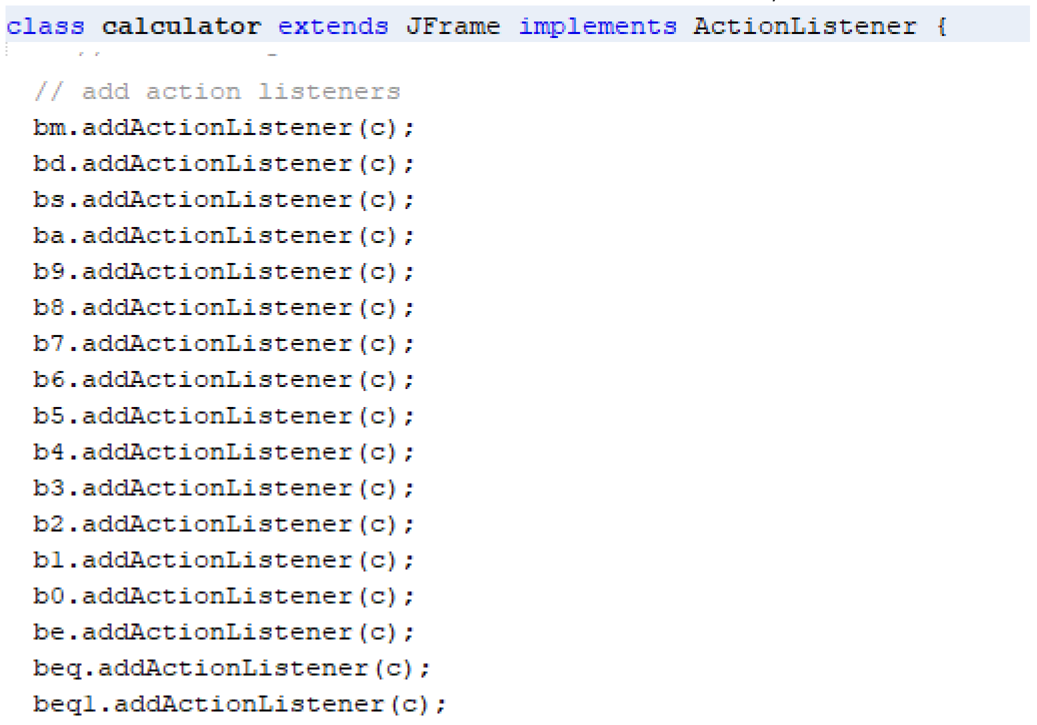
Polymorphism is the concept where an object behaves differently in different situations. Runtime polymorphism is implemented when we have an “IS-A”relationship between objects. This is also called a method overriding because the subclass has to override the superclass method for runtime polymorphism.



```java
@Override
public void actionPerformed(ActionEvent e)
{
    String s = e.getActionCommand();

    // if the value is a number
    if ((s.charAt(0) >= '0' && s.charAt(0) <= '9') || s.charAt(0) == '.') {
        // if operand is present then add to second no
        if (!s1.equals(""))
            s2 = s2 + s;
        else
            s0 = s0 + s;

        // set the value of text
        l.setText(s0 + s1 + s2);
    }
```

6. Interface

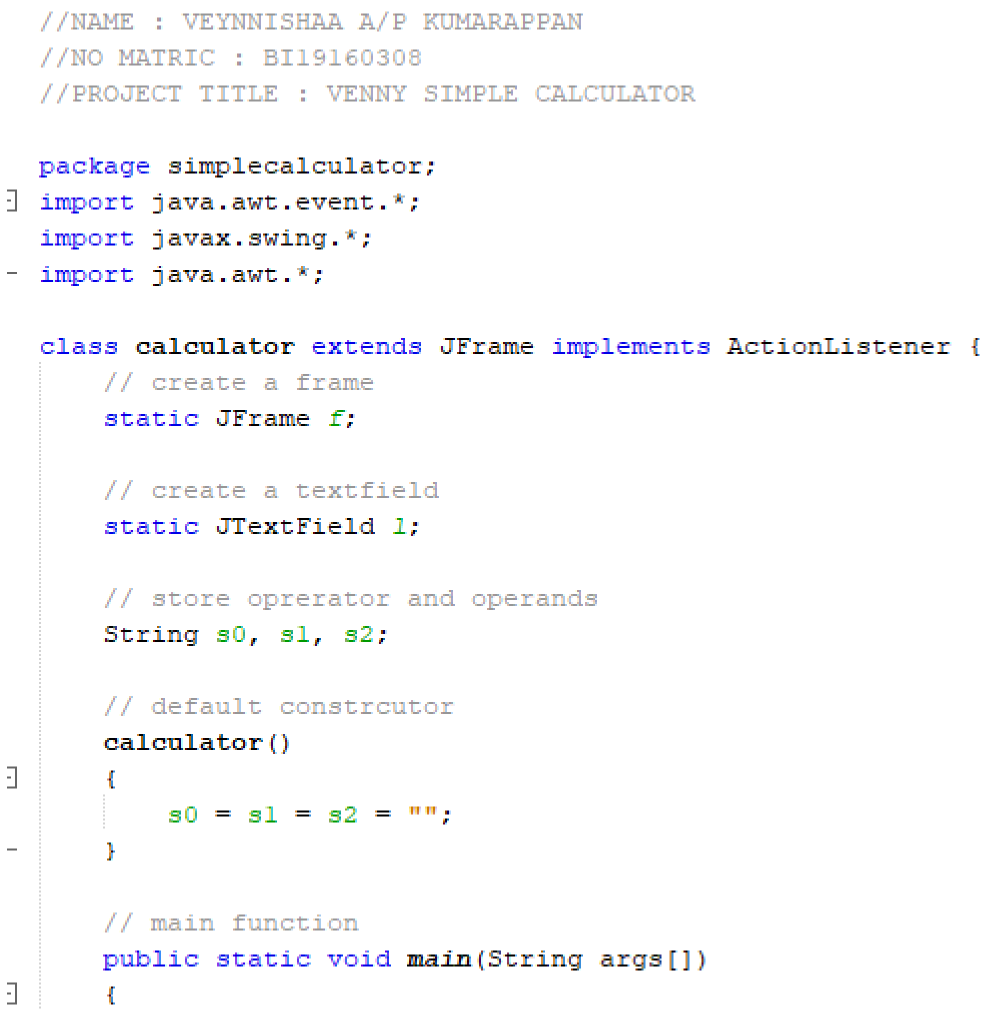
Interface looks like a class but it is not a class. An interface can have methods and variables just like the class but the methods declared in interface are by default abstract.



```java
class calculator extends JFrame implements ActionListener {

  // add action listeners
  bm.addActionListener(c);
  bd.addActionListener(c);
  bs.addActionListener(c);
  ba.addActionListener(c);
  b9.addActionListener(c);
  b8.addActionListener(c);
  b7.addActionListener(c);
  b6.addActionListener(c);
  b5.addActionListener(c);
  b4.addActionListener(c);
  b3.addActionListener(c);
  b2.addActionListener(c);
  b1.addActionListener(c);
  b0.addActionListener(c);
  be.addActionListener(c);
  beq.addActionListener(c);
  beq1.addActionListener(c);
```

READ AND WRITE IMPLEMENTATION

In this project I use JTextField. JTextField is a lightweight component that allows the editing of a single line of text. JTextField is intended to be source-compatible with java.awt.TextField where it is reasonable to do so. The superclass should be consulted for additional capabilities. A text field is a box into which the user can type text strings. A text field is represented by a JTextField object, which can be constructed with a parameter specifying the number of characters that should be able to fit in the text field.
JTextField has a method to establish the string used as the command string for the action event that gets fired. The java.awt.TextField used the text of the field as the command string for the ActionEvent. JTextField will use the command string set with the setActionCommand method if not null, otherwise it will use the text of the field as a compatibility with java.awt.TextField.

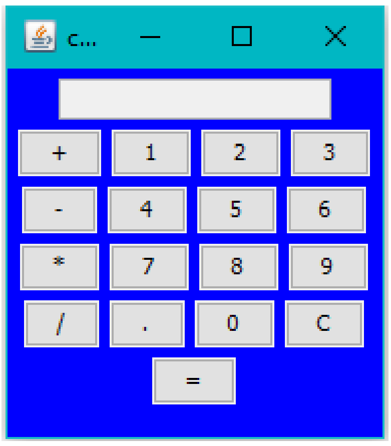


```
//NAME : VEYNNISHAA A/P KUMARAPPAN
//NO MATRIC : BI19160308
//PROJECT TITLE : VENNY SIMPLE CALCULATOR

package simplecalculator;
import java.awt.event.*;
import javax.swing.*;
import java.awt.*;

class calculator extends JFrame implements ActionListener {
    // create a frame
    static JFrame f;

    // create a textfield
    static JTextField l;

    // store oprerator and operands
    String s0, s1, s2;

    // default constrcutor
    calculator()
    {
        s0 = s1 = s2 = "";
    }

    // main function
    public static void main(String args[])
    {
```

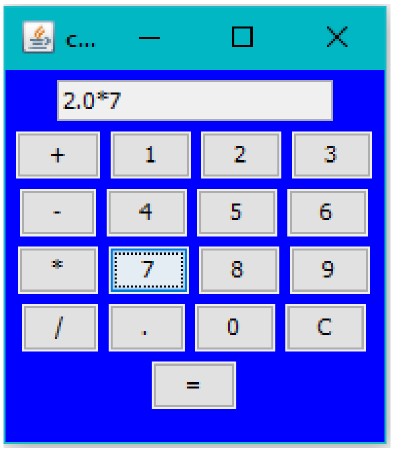
USER MANUAL

Basic calculator project is calculator with a basic operation. For example, addition, subtraction, divided and multiplication.

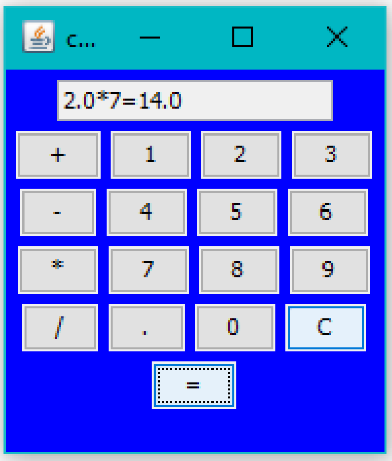
1. Run the project until you see this GUI Output appear



2. Click any number, operation sign, and followed by equals on the screen to retrieve calculations



3. Once achieved desired answer click Clear to clear off previous calculation



CONCLUSION

GUIs are potentially very complex entities because they involve a large number of interacting objects and classes. Each onscreen component and window are represented by an object, so a programmer starting out with GUIs must learn many new class, method, and package names. In addition, if the GUI is to perform sophisticated tasks, the objects must interact with each other and call each other's methods, which raises tricky communication and scoping issues.
Another factor that makes writing GUIs challenging is that the path of code execution becomes non-deterministic. When a GUI program is running, the user can click any of the buttons and interact with any of the other onscreen components in any order. Because the program's execution is driven by the series of events that occur, we say that programs with GUIs are event-driven. Last but not least I hope that my simple calculator will provide as good medium to new learners and in the near future I try my very best to come up a newer and with much more sophisticated interface and functions as well.