|  |  |
| --- | --- |
| H:\LOGO-NXV\Hai__090908__02_1_den.jpg | Faculty of Information Technology  H A N O I U N I V E R S I T Y |

61FIT3NPR – Network Programming

Java SWING

1. Exercise 1: swing from code only

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

**import** java.awt.event.\*;

**import** java.awt.\*;

**import** java.io.\*;

**public** **class** CalculateDemo **extends** JFrame **implements** ActionListener{

**private** JButton btn1, btn2, btn3, btn4;

/\*2 so va ket qua\*/

**private** JTextField tf1, tf2, tf3;

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

/\*dung de nhan tang ContentPane JFrame\*/

**private** Container container;

/\*dung panel de nhom cac thanh phan tren giao dien\*/

**private** JPanel panel1, panel2;

**public** CalculateDemo(String s) {

**super**(s);

/\*lay class ContentPane de dat cac doi tuong hien thi\*/

container = **this**.getContentPane();

/\*tao cac thanh phan tren giao dien\*/

JLabel num1 = **new** JLabel("1st number: ");

tf1 = **new** JTextField();

JLabel num2 = **new** JLabel("1st number: ");

tf2 = **new** JTextField();

JLabel resultl = **new** JLabel("Ket qua: ");

tf3 = **new** JTextField();

tf3.setEditable(**false**);

/\*panel 1\*/

panel1 = **new** JPanel();

/\*layout gom 3 cot\*/

panel1.setLayout(**new** GridLayout(3,2));

/\*panel 1\*/

panel1.add(num1);

panel1.add(tf1);

panel1.add(num2);

panel1.add(tf2);

panel1.add(resultl);

panel1.add(tf3);

/\*tao 4 buttons\*/

btn1 = **new** JButton("+");

btn2= **new** JButton("-");

btn3 = **new** JButton("\*");

btn4 = **new** JButton(":");

/\*Panel2 co 4 nut\*/

panel2 = **new** JPanel();

panel2.add(btn1);

panel2.add(btn2);

panel2.add(btn3);

panel2.add(btn4);

/\*2 panel get in ContentPane\*/

container.add(panel1);

container.add(panel2,"South");

btn1.addActionListener(**this**);

btn2.addActionListener(**this**);

btn3.addActionListener(**this**);

btn4.addActionListener(**this**);

/\*kich thuoc hien thi\*/

**this**.setSize(350, 200);

**this**.setVisible(**true**);

}

**public** **void** Add()

{

result = Double.*parseDouble*(tf1.getText()) + Double.*parseDouble*(tf2.getText());

tf3.setText(String.*valueOf*(result));

}

**public** **void** Minus()

{

result = Double.*parseDouble*(tf1.getText()) - Double.*parseDouble*(tf2.getText());

tf3.setText(String.*valueOf*(result));

}

**public** **void** Multi()

{

result = Double.*parseDouble*(tf1.getText()) \* Double.*parseDouble*(tf2.getText());

tf3.setText(String.*valueOf*(result));

}

**public** **void** Div()

{

result = Double.*parseDouble*(tf1.getText()) / Double.*parseDouble*(tf2.getText());

tf3.setText(String.*valueOf*(result));

}

/\*bat dau tinh toan khi nguoi dung nhap phep tinh\*/

**public** **void** actionPerformed(ActionEvent e)

{

**if** (e.getActionCommand()=="+") {

//khi chua nhap so da bam tinh toan

**if**(tf1.getText().equals("") || tf2.getText().equals("")) {

JOptionPane.*showMessageDialog*(**this**, "Ban chua nhap du");

}**else** {

Add();

}

}

**if** (e.getActionCommand()=="-") {

**if**(tf1.getText().equals("")|| tf2.getText().equals("")) {

JOptionPane.*showMessageDialog*(**this**, "Ban chua nhap du");

}**else** {

Minus();

}

}

**if** (e.getActionCommand()=="\*"){

**if**(tf1.getText().equals("")|| tf2.getText().equals("")) {

JOptionPane.*showMessageDialog*(**this**, "Ban chua nhap du");

}**else** {

Multi();

}

}

**if** (e.getActionCommand()==":") {

**if**(tf1.getText().equals("")|| tf2.getText().equals("")) {

JOptionPane.*showMessageDialog*(**this**, "Ban chua nhap du");

}**else** {

Div();

}

}

}

**public** **static** **void** main(String[] args) {

CalculateDemo cal = **new** CalculateDemo("SimpleCalculator");

// cal.setLocationRelativeTo(null);

}

}

1. Exercise 2: with IntelliJ GUI

<https://examples.javacodegeeks.com/desktop-java/ide/intellij-gui-designer-example/>

# IntelliJ GUI Designer Example

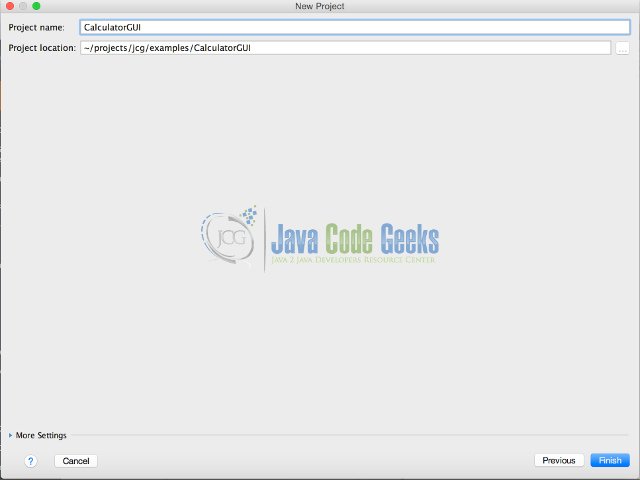
This example demonstrates how you can utilize IntelliJ GUI Designer to create a graphical user interface that is based on the Swing library components. It must be noted that IntelliJ does not support modeling of non-swing components.

When working with the GUI Designer you operate in design view where you are able to drag and drop swing components within the context of a form. All GUI information related to the form are stored in a file with .form extension. When creating a GUI component in IntelliJ you begin by creating a form, this form serves as a container that groups all other components that are required for your application. When a form is created, IntelliJ provides you with an option of also creating a Bound Class. A Bound Class is a Java class that is bound to a form and contains auto-generated code that reflects the components that are added on the form. The bound class can be updated at any point to add specific behavior on the components that have been added in the form.

In this example we shall create a simple calculator application to demonstrate how you work with the GUI Designer.

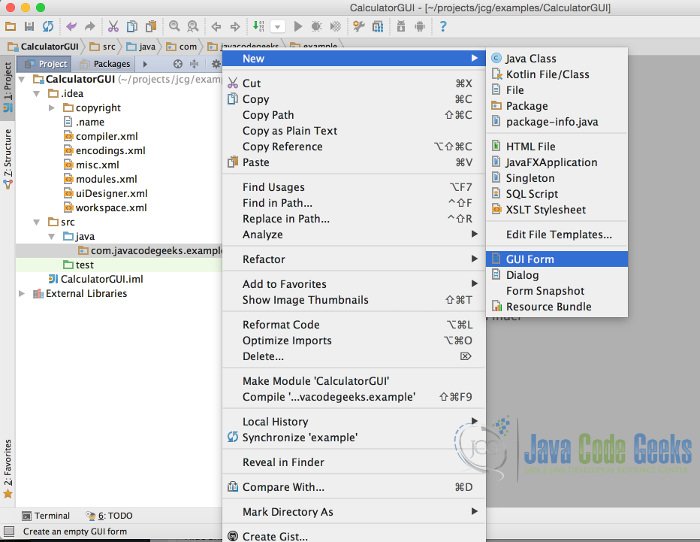
## **1. Creating a new project**

Launch IntelliJ and create a new project called: **CalculatorGUI**.



Create a new package in the src/java directory called **com.javacodegeeks.example**

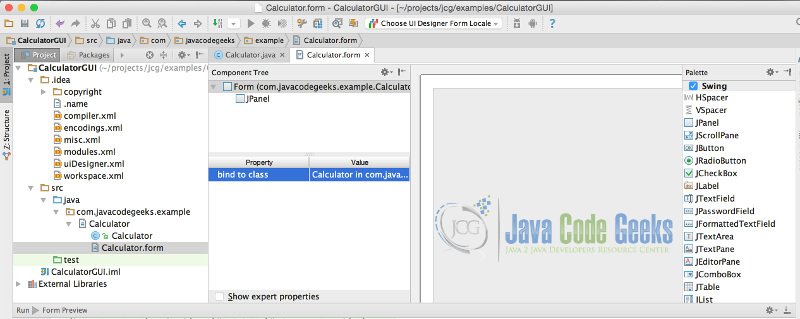
Right-click the new package and select -> **New** -> **GUI Form**



In the New GUI Form Window populate the following:

* **Form Name**: Calculator
* **Base layout manager**: GridLayoutManager(IntelliJ)
* **Create bound class**:selected
* **Class name**: Calculator

Clicking Ok on the GUI Form window once its populated, should display design-time graphical view as seen below:



## **2. Graphical view**

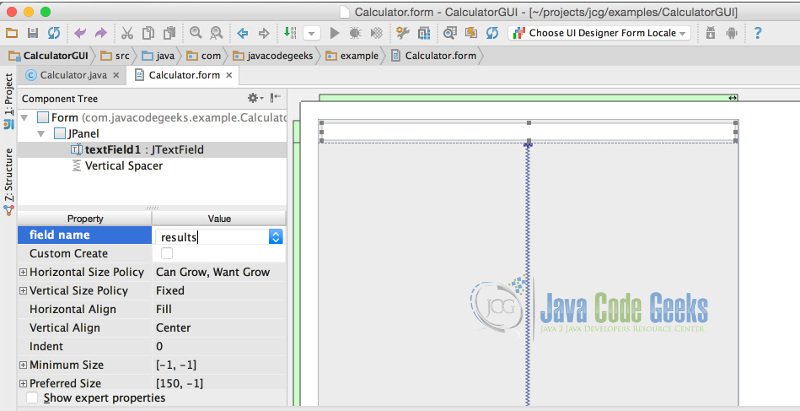
The graphical view allows you to drag and drop swing components to be added on the Form of the application. When any file that has a .form extension has been selected into the view, the following windows are displayed:

1. Component tree – displays the hierarchical view of the components that have been added on the form.
2. Property editor – displays the properties of the components added on the form which can be edited.
3. Design area – the graphical view of the form. Components are dragged into this area to be added on the form.
4. Palette – contains all available swing components that can be selected to be added on the form.

## **3. Creating the GUI**

### **3.1 Add Results display**

Drag the JTextField from the palette and drop it in the design area. Update the field name in the property editor to read: **results**

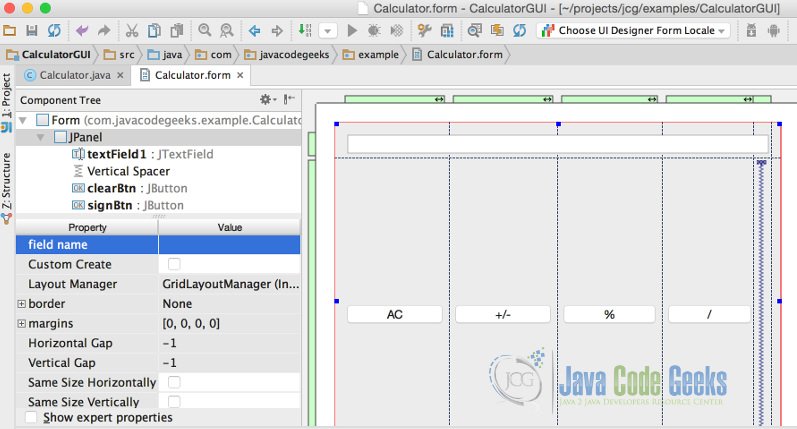


When you add the JTextField the Vertical Spacer also gets automatically added.

### **3.2 Add Buttons**

Drag the JButton from the palette and drop it in the design area. Add the JButton on the left-hand side of the Vertical Spacer. As you release the mouse a tooltip will be displayed showing JPanel(Row 1, Before Col 0), which indicates the position where the component will be placed in the grid. Repeat the process to add 4 buttons in the same row. Update the properties of the 4 buttons to the following:

* **button1**: field name change to **clearBtn**,  Text change to**AC**
* **button2**: field name change to **signBtn**. Text change to **+/-**
* **button3**: field name change to **percentBtn**.Text change to **%**
* **button4**: field name change to **divideBtn**. Text change to **/**



Add the rest of the buttons, in total should have 5 rows and 4 columns populated with buttons. You can now remove the Vertical Spacer.



Select the JPanel in the Component tree of the form view and update the field name property to calculatorView. Clicking on the Calculator.java should now have the following fields inserted:

*Calculator.java*

package com.javacodegeeks.example;

import javax.swing.\*;

public class Calculator {

    private JTextField resultsTxt;

    private JButton clearBtn;

    private JButton signBtn;

    private JButton percentBtn;

    private JButton divideBtn;

    private JButton sevenBtn;

    private JButton eightBtn;

    private JButton nineBtn;

    private JButton multiplyBtn;

    private JButton fourBtn;

    private JButton fiveBtn;

    private JButton sixBtn;

    private JButton minusBtn;

    private JButton oneBtn;

    private JButton twoBtn;

    private JButton threeBtn;

    private JButton addBtn;

    private JButton zeroBtn;

    private JButton equalBtn;

    private JButton digitBtn;

    private JPanel calculatorView;

}

## **4. Making the form functional**

In order for the form to be functional it requires a runtime frame to be created. We will create the main() method that will will be responsible for creating and disposing the runtime frame.

In the code editor of Calculator.java file select -> Generate… -> Form main()

The following code gets generated:

Calculator.java main method

public static void main(String[] args) {

        JFrame frame = new JFrame("Calculator");

        frame.setContentPane(new Calculator().calculatorView);

        frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

        frame.pack();

        frame.setVisible(true);

    }

Now we create an enum for handling the different calculator operations. The constructor of the enum takes in a function in this case DoubleBinaryOperator (provided in Java 8) which represents an operation on two double-valued operands that produces a double-valued result.

Operation.java

package com.javacodegeeks.example;  
  
import java.util.function.DoubleBinaryOperator;  
  
public enum Operation {  
 *ADDITION*((x, y) -> x+y),  
 *SUBTRACTION*((x, y) -> x-y),  
 *DIVISION*((x, y) -> x/y),  
 *MULTIPLICATION*((x, y) -> x\*y),  
 *PERCENTAGE*((x, y) -> x%y);  
  
  
 private DoubleBinaryOperator operator;  
  
 Operation(DoubleBinaryOperator operator) {  
 this.operator = operator;  
 }  
  
 public DoubleBinaryOperator getOperator() {  
 return operator;  
 }  
}

## **5. Putting everything together**

Now we add action listeners that will be triggered when the buttons of the calculator get clicked. We then bind the buttons to those action listeners.

Calculator.java

package com.javacodegeeks.example;  
  
import javax.swing.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
  
public class Calculator {  
 private boolean control = false;  
 private JTextField resultsTxt;  
 private JButton clearBtn;  
 private JButton signBtn;  
 private JButton percentBtn;  
 private JButton divideBtn;  
 private JButton sevenBtn;  
 private JButton eightBtn;  
 private JButton nineBtn;  
 private JButton multiplyBtn;  
 private JButton fourBtn;  
 private JButton fiveBtn;  
 private JButton sixBtn;  
 private JButton minusBtn;  
 private JButton oneBtn;  
 private JButton twoBtn;  
 private JButton threeBtn;  
 private JButton addBtn;  
 private JButton zeroBtn;  
 private JButton equalBtn;  
 private JButton digitBtn;  
 private JPanel calculatorView;  
 private Double leftOperand;  
 private Double rightOperand;  
 private Operation calcOperation;  
  
 public Calculator() {  
  
 sevenBtn.addActionListener(new NumberBtnClicked(sevenBtn.getText()));  
 eightBtn.addActionListener(new NumberBtnClicked(eightBtn.getText()));  
 nineBtn.addActionListener(new NumberBtnClicked(nineBtn.getText()));  
 fourBtn.addActionListener(new NumberBtnClicked(fourBtn.getText()));  
 fiveBtn.addActionListener(new NumberBtnClicked(fiveBtn.getText()));  
 sixBtn.addActionListener(new NumberBtnClicked(sixBtn.getText()));  
 oneBtn.addActionListener(new NumberBtnClicked(oneBtn.getText()));  
 twoBtn.addActionListener(new NumberBtnClicked(twoBtn.getText()));  
 threeBtn.addActionListener(new NumberBtnClicked(threeBtn.getText()));  
 zeroBtn.addActionListener(new NumberBtnClicked(zeroBtn.getText()));  
  
 percentBtn.addActionListener(new OperationBtnClicked(Operation.*PERCENTAGE*));  
 multiplyBtn.addActionListener(new OperationBtnClicked(Operation.*MULTIPLICATION*));  
 divideBtn.addActionListener(new OperationBtnClicked(Operation.*DIVISION*));  
 minusBtn.addActionListener(new OperationBtnClicked(Operation.*SUBTRACTION*));  
 addBtn.addActionListener(new OperationBtnClicked(Operation.*ADDITION*));  
 equalBtn.addActionListener(new EqualBtnClicked());  
 clearBtn.addActionListener(new ClearBtnClicked());  
 signBtn.addActionListener(new SignBtnClicked());  
 digitBtn.addActionListener(new DigitBtnClicked());  
 }  
  
 private class NumberBtnClicked implements ActionListener {  
  
 private String value;  
 private String btnvalue;  
 public NumberBtnClicked(String btnvalue) {  
 this.btnvalue = btnvalue;  
 }  
  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 if(control == true){  
 resultsTxt.setText("");  
 control = false;  
 }  
 if(leftOperand == null || leftOperand == 0.0) {  
 value = resultsTxt.getText() + btnvalue;  
 //leftOperand = Double.valueOf(resultsTxt.getText());  
 }else{  
 value = resultsTxt.getText() + btnvalue;  
 rightOperand = Double.*valueOf*(value);  
 }  
 resultsTxt.setText(value);  
  
 }  
 }  
  
 private class OperationBtnClicked implements ActionListener {  
  
 private Operation operation;  
  
 public OperationBtnClicked(Operation operation) {  
 this.operation = operation;  
 }  
  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 calcOperation = operation;  
 leftOperand = Double.*valueOf*(resultsTxt.getText());  
 control = true;  
 }  
 }  
  
 private class ClearBtnClicked implements ActionListener {  
  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 resultsTxt.setText("");  
 leftOperand = 0.0;  
 rightOperand = 0.0;  
 control = false;  
 }  
 }  
  
 private class DigitBtnClicked implements ActionListener {  
  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 resultsTxt.setText(resultsTxt.getText() + ".");  
  
 }  
 }  
  
 private class EqualBtnClicked implements ActionListener {  
  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 Double output = calcOperation.getOperator().applyAsDouble(leftOperand, rightOperand);  
 resultsTxt.setText(output%1==0?String.*valueOf*(output.intValue()):String.*valueOf*(output));  
 leftOperand = 0.0;  
 rightOperand = 0.0;  
 control = true;  
 }  
 }  
  
 private class SignBtnClicked implements ActionListener {  
 private String value;  
  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 if (control == false) {  
 switch (resultsTxt.getText()) {  
 case "":  
 resultsTxt.setText("-");  
 break;  
 case "-":  
 resultsTxt.setText("");  
 break;  
 default: {  
 Double out = Double.*valueOf*(resultsTxt.getText());  
 if (out > 0.0) {  
 resultsTxt.setText("-" + resultsTxt.getText());  
  
 }  
 if (out < 0.0) {  
 out = -out;  
 resultsTxt.setText(out % 1 == 0 ? String.*valueOf*(out.intValue()) : String.*valueOf*(out));  
 }  
 if (!(leftOperand == null || leftOperand == 0.0)) {  
 value = resultsTxt.getText();  
 rightOperand = Double.*valueOf*(value);  
 }  
  
 }  
 }  
 }  
 //when control is true  
 else {  
 resultsTxt.setText("-");  
 control = false;  
 }  
  
  
 }  
 }  
  
  
  
 public static void main(String[] args) {  
 JFrame frame = new JFrame("Calculator");  
 frame.setContentPane(new Calculator().calculatorView);  
 frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 frame.pack();  
 frame.setVisible(true);  
 }  
}

## **6. Running your application**

Right-click Calculator.java and select Run Calculator.main()

