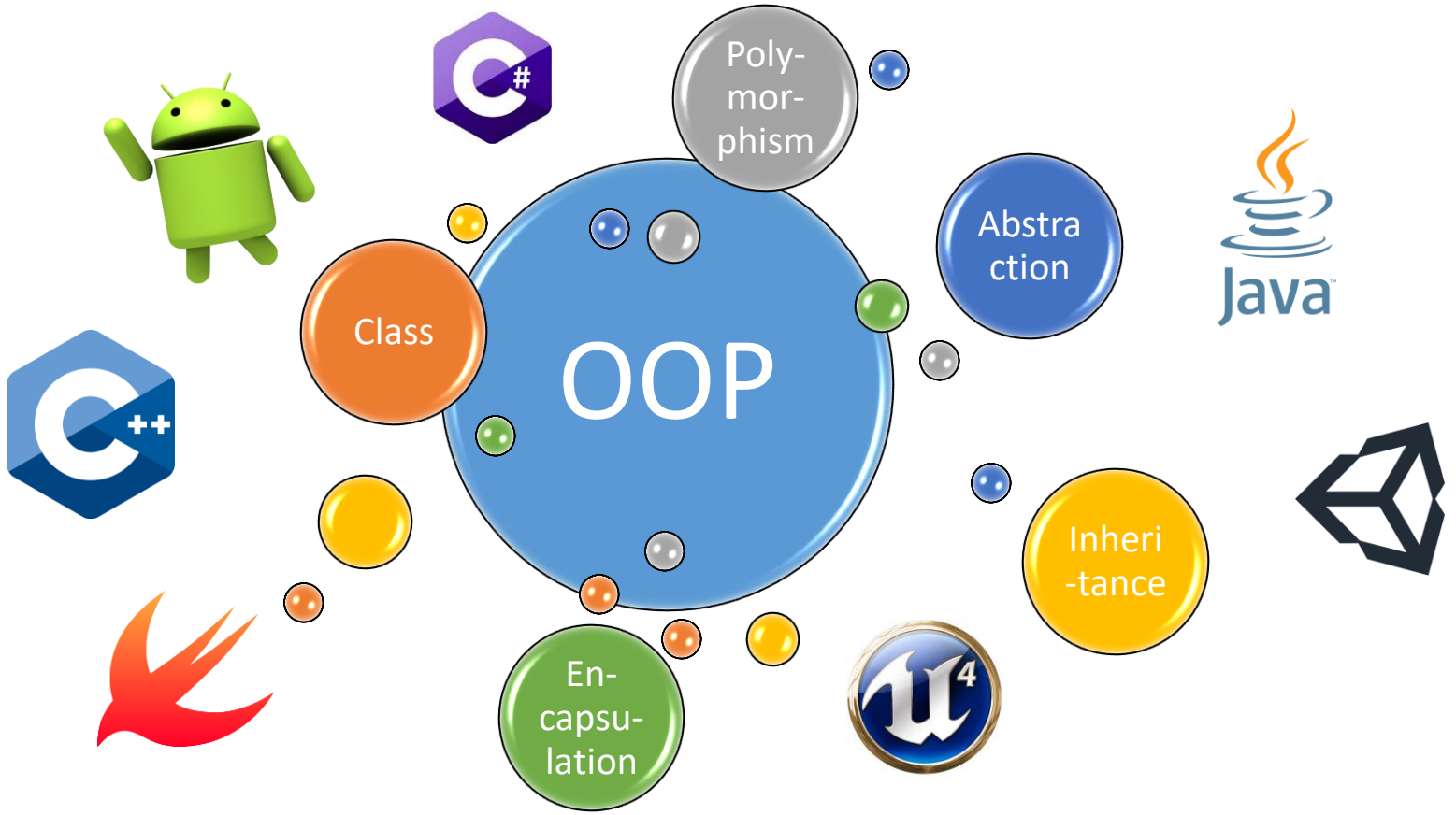


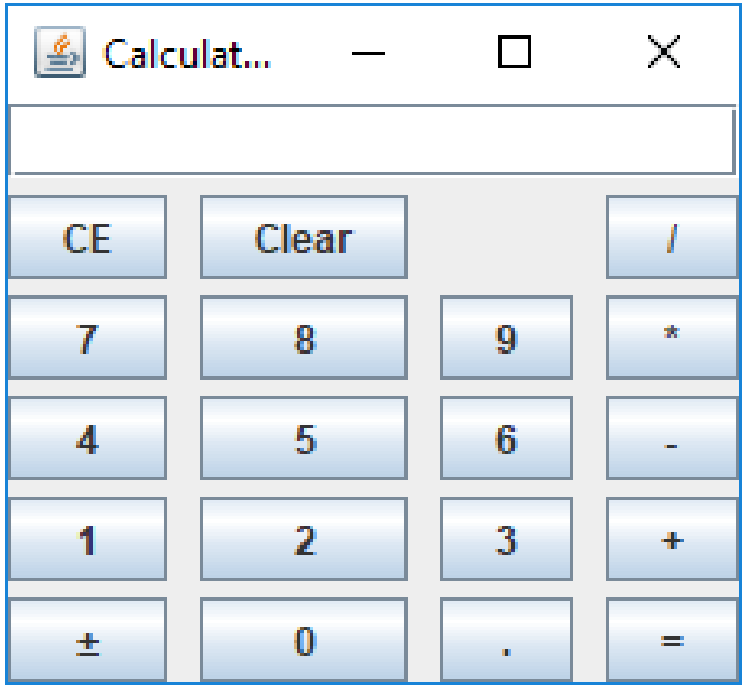
Homework 2 - Basic Calculator

Kuan-Ting Lai

2019/3/20

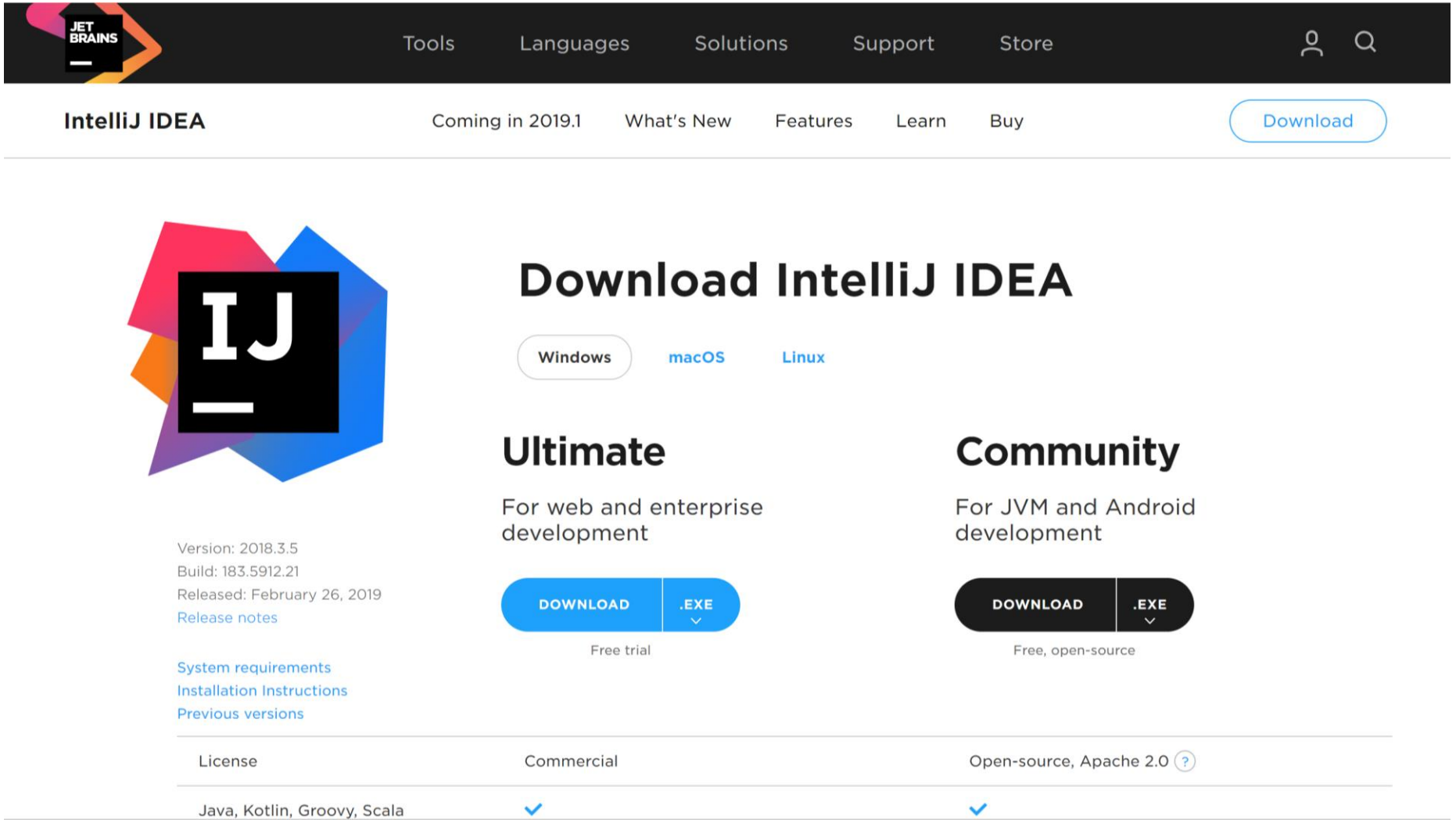
Building a Basic Calculator

- Create a basic calculator using Java Swing
- Implement basic functions:
  - Addition (+)
  - Subtraction (–)
  - Multiplication (*)
  - Division (/)
  - Minus-plus (±)
  - Clear (C)
  - Cancel Entry (CE)



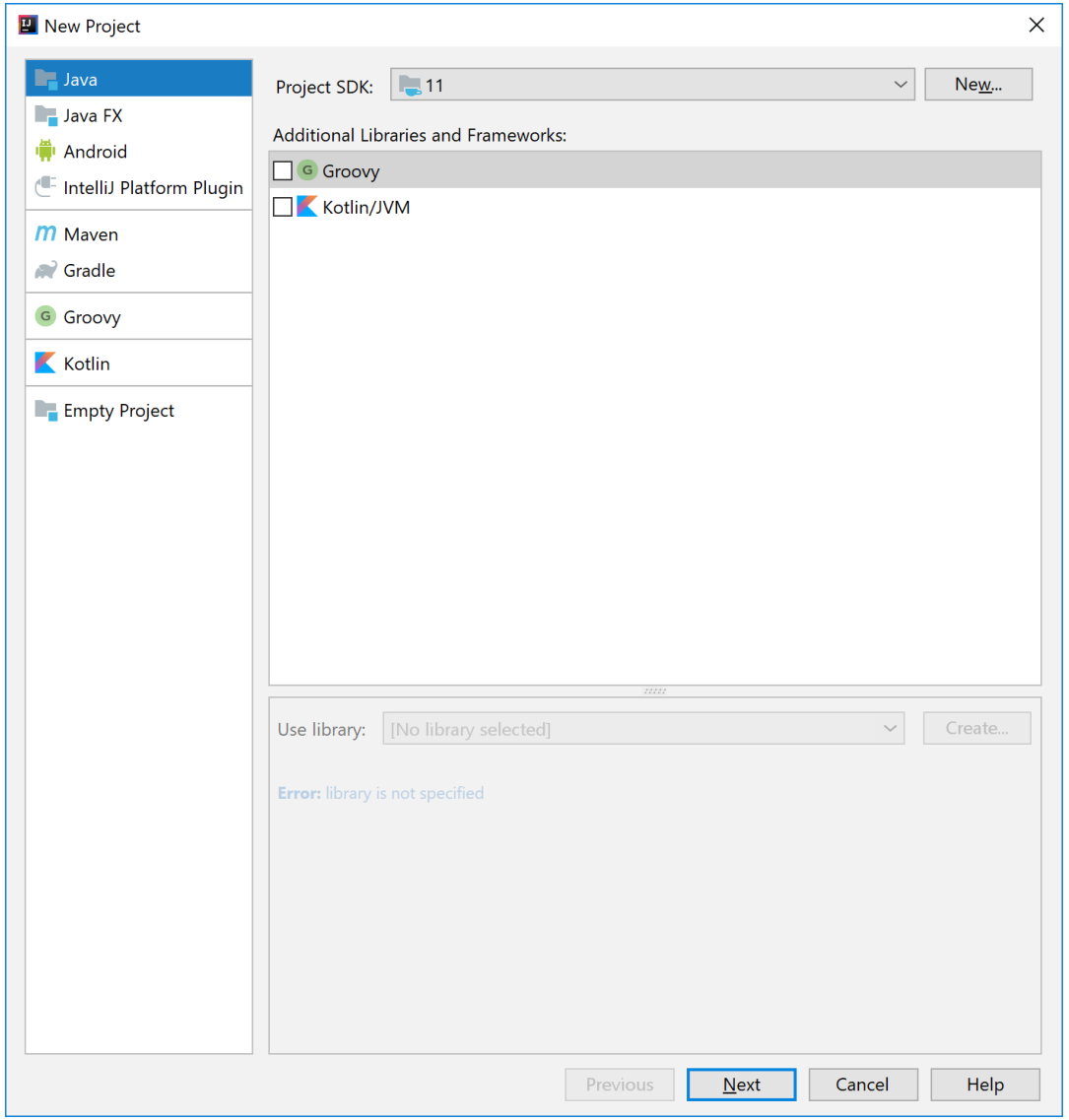
Building Calculator using IntelliJ

- Install IntelliJ Community



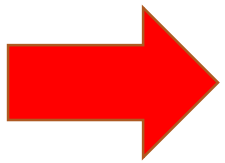
Create a Java Project

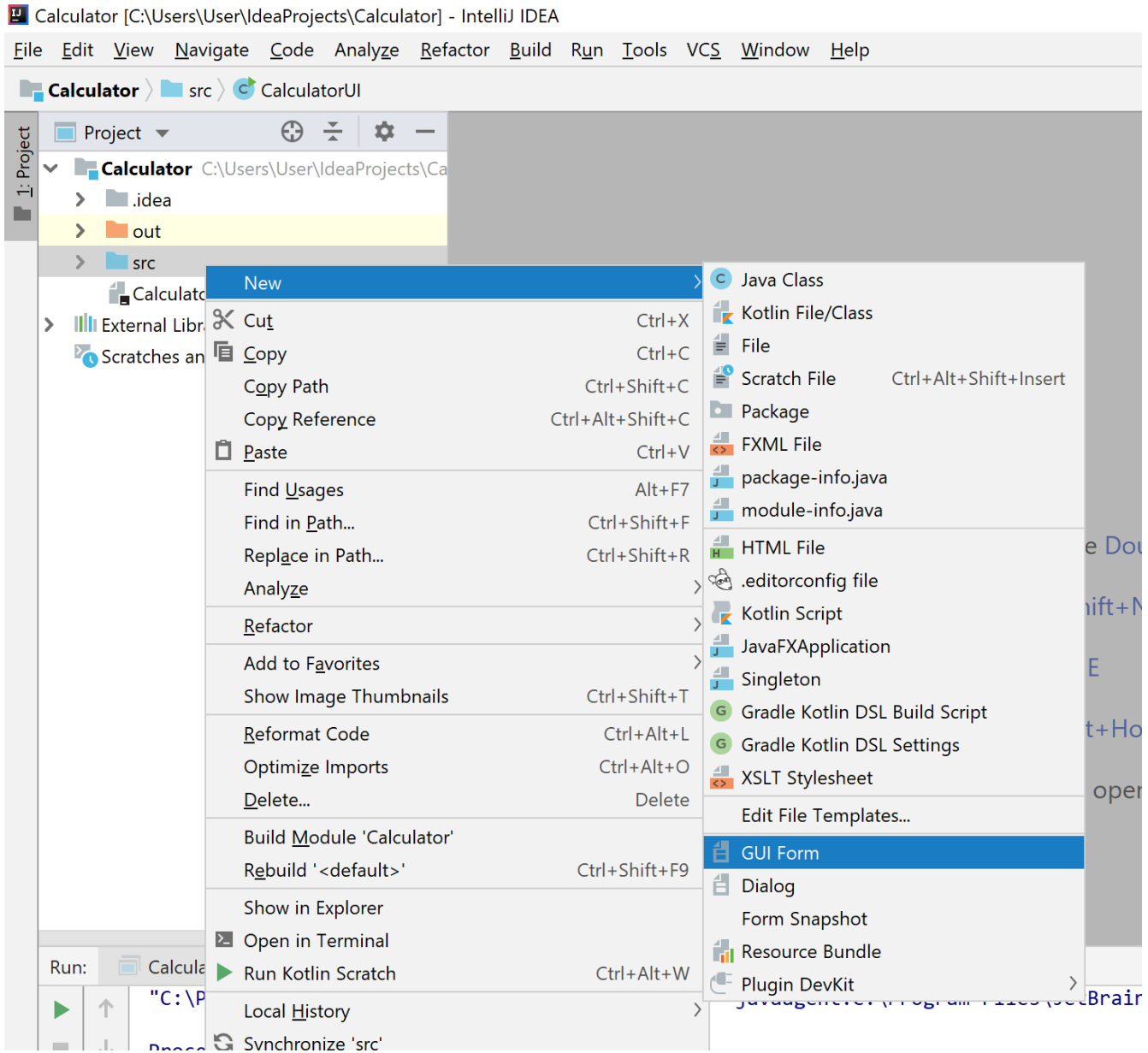
- Press Next
- Don’t select any template



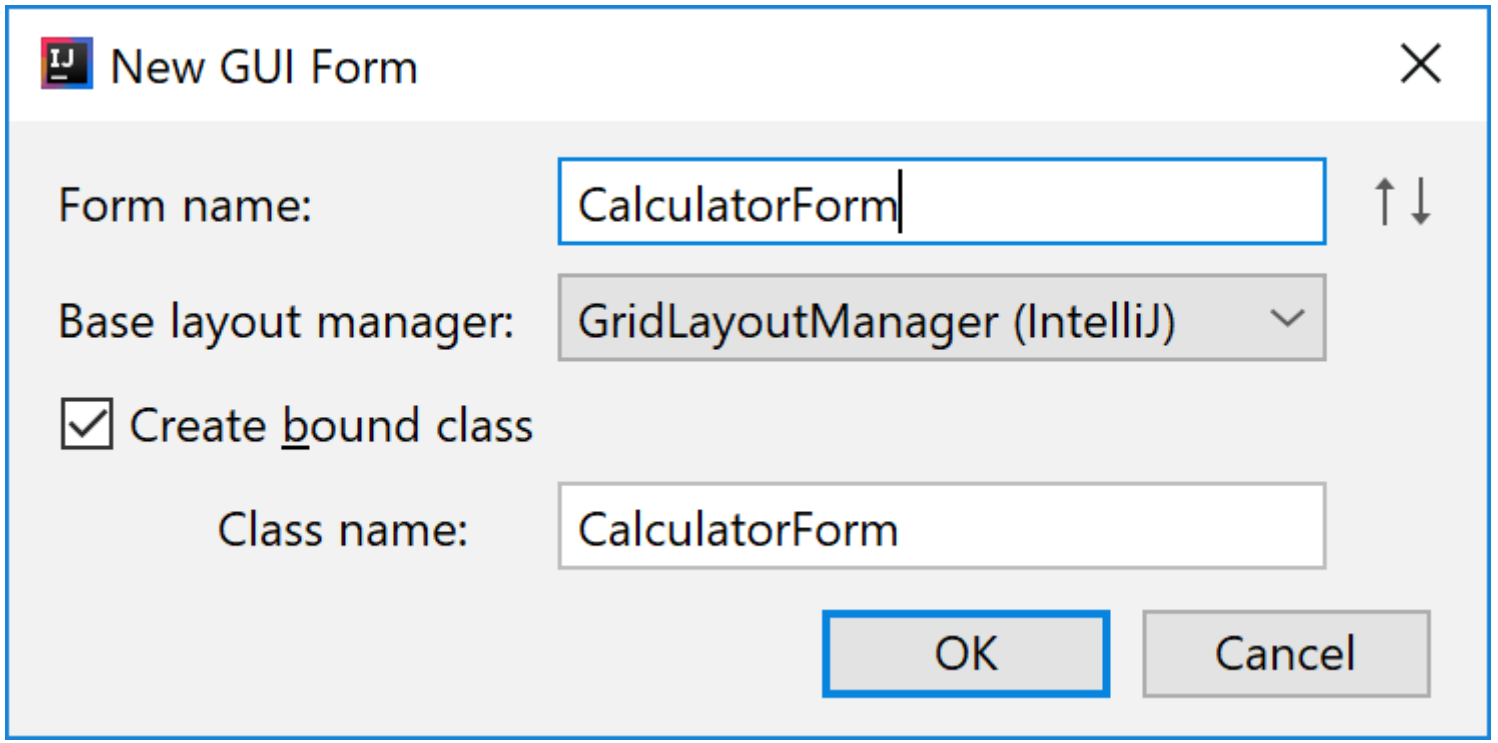
Add GUI Form

Right Click



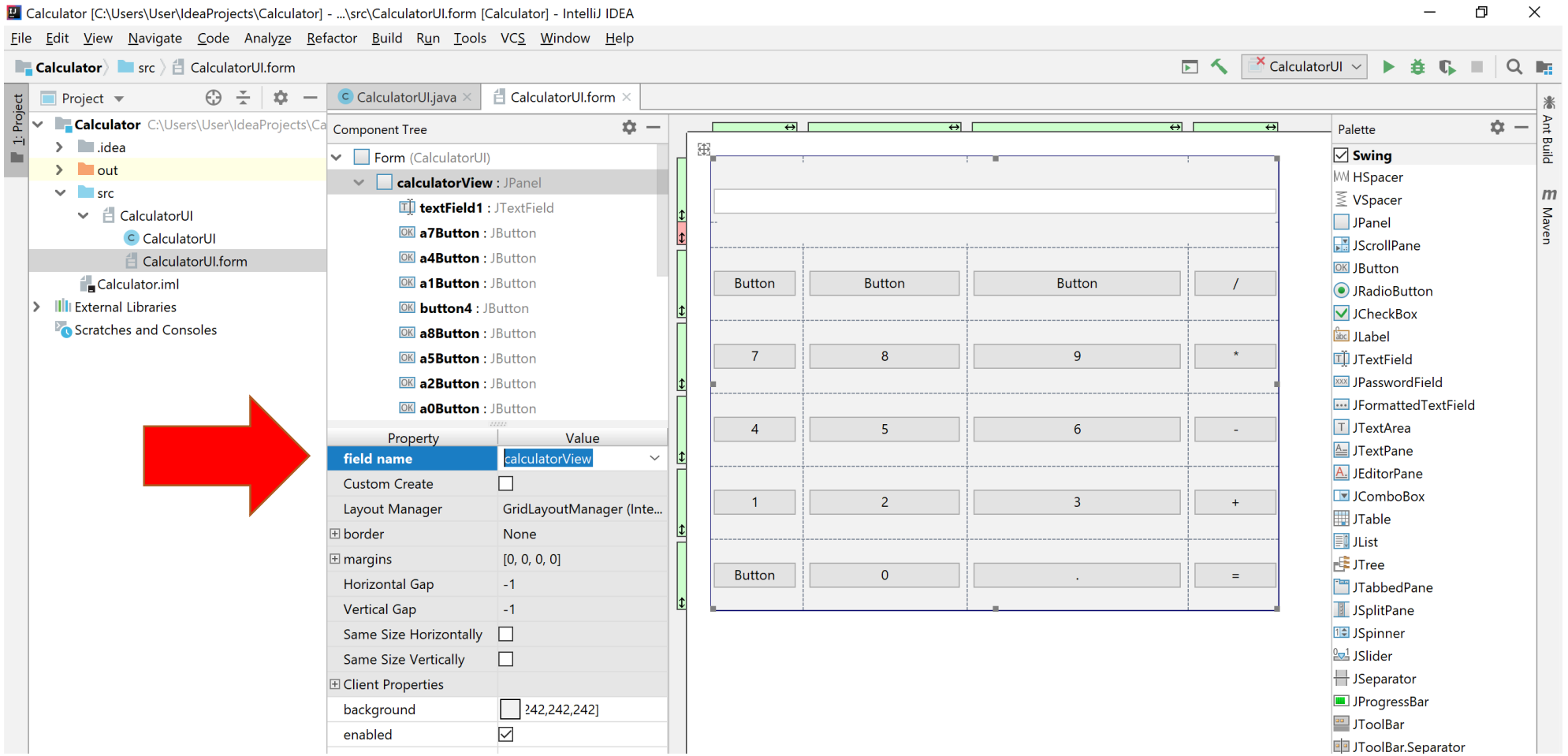


Name Your Form “CalculatorForm”



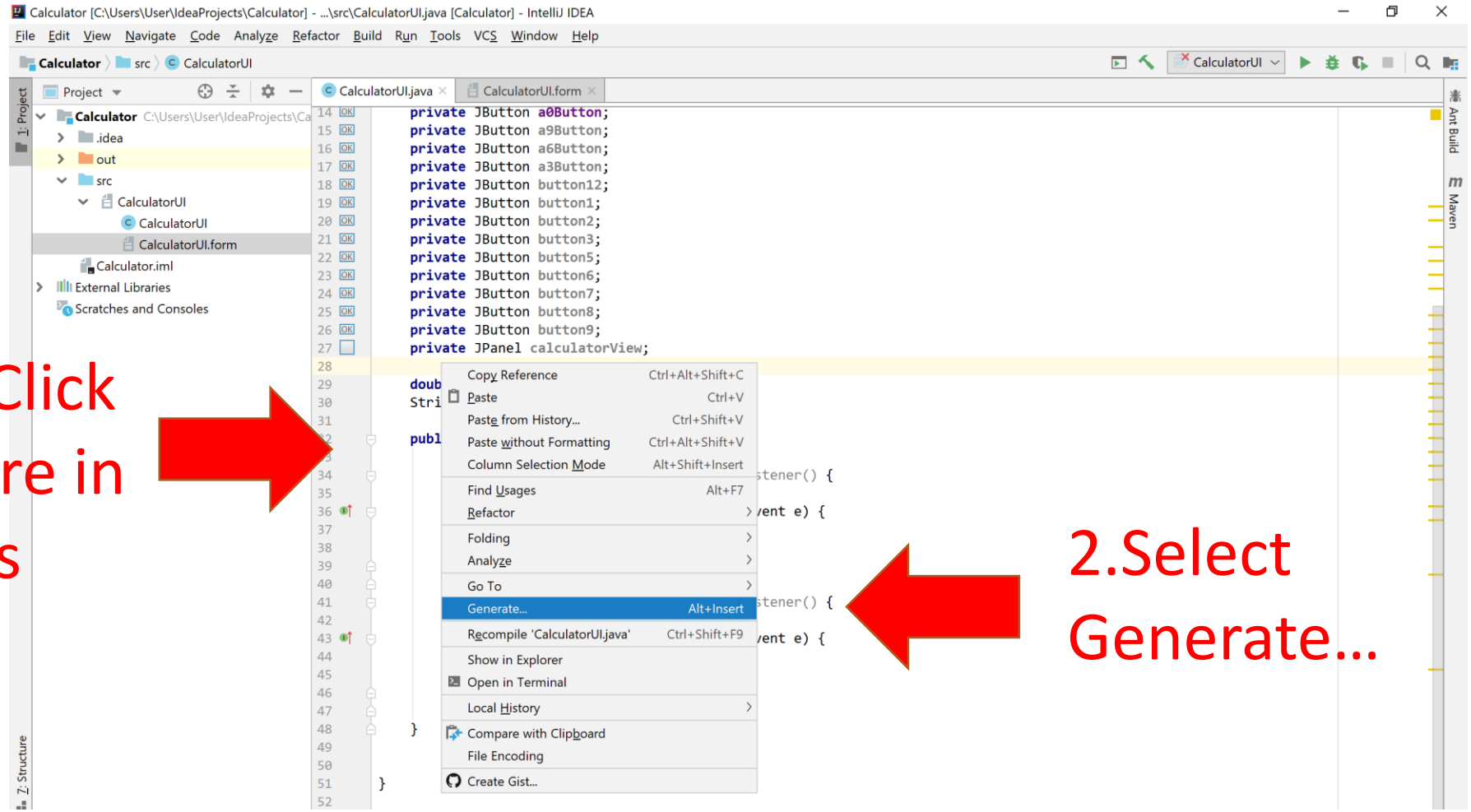
Set JPanel's Name

- Select the JPanel in the Component tree of the form view and update the field name property to calculatorView.

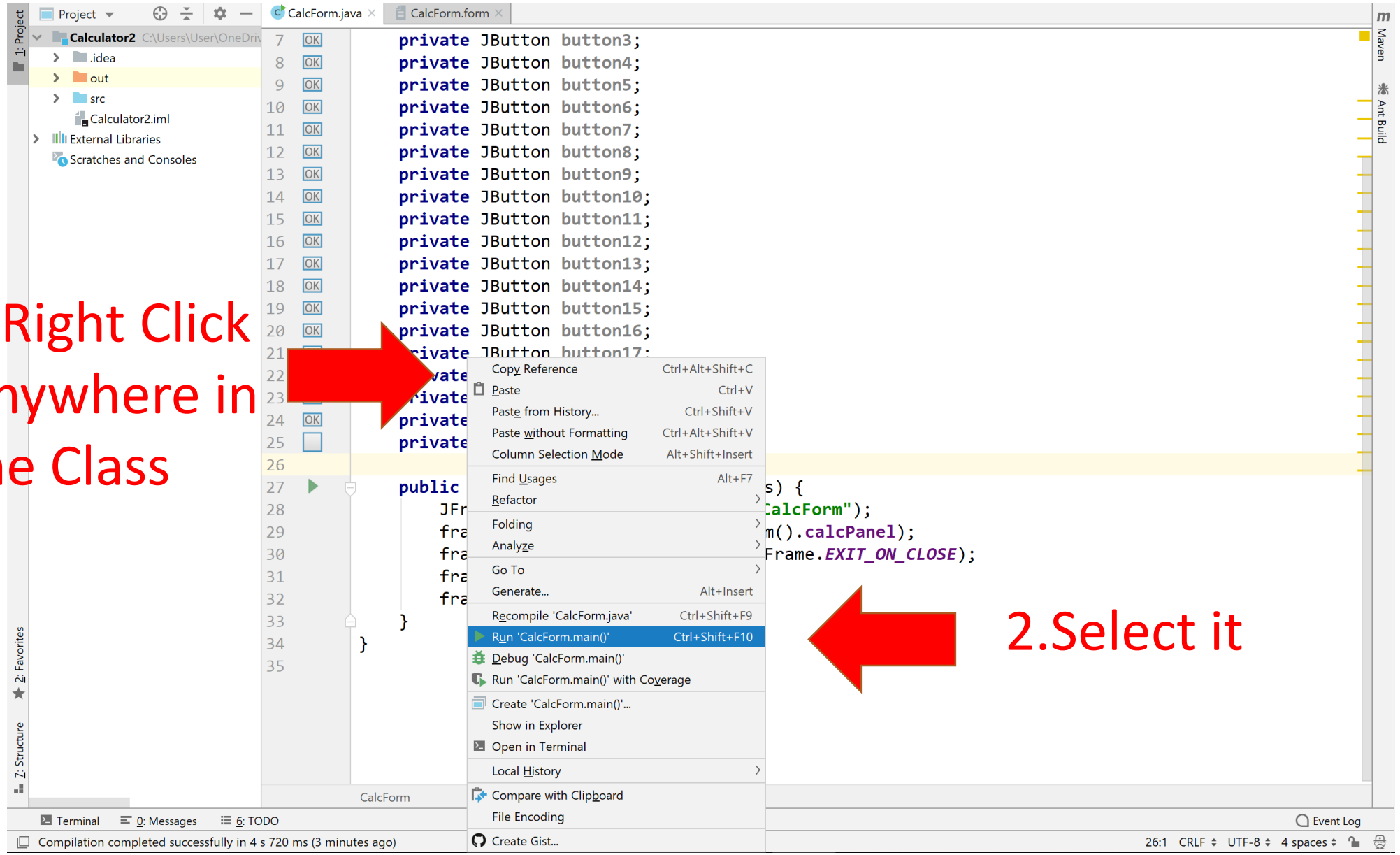


Generate main() Code

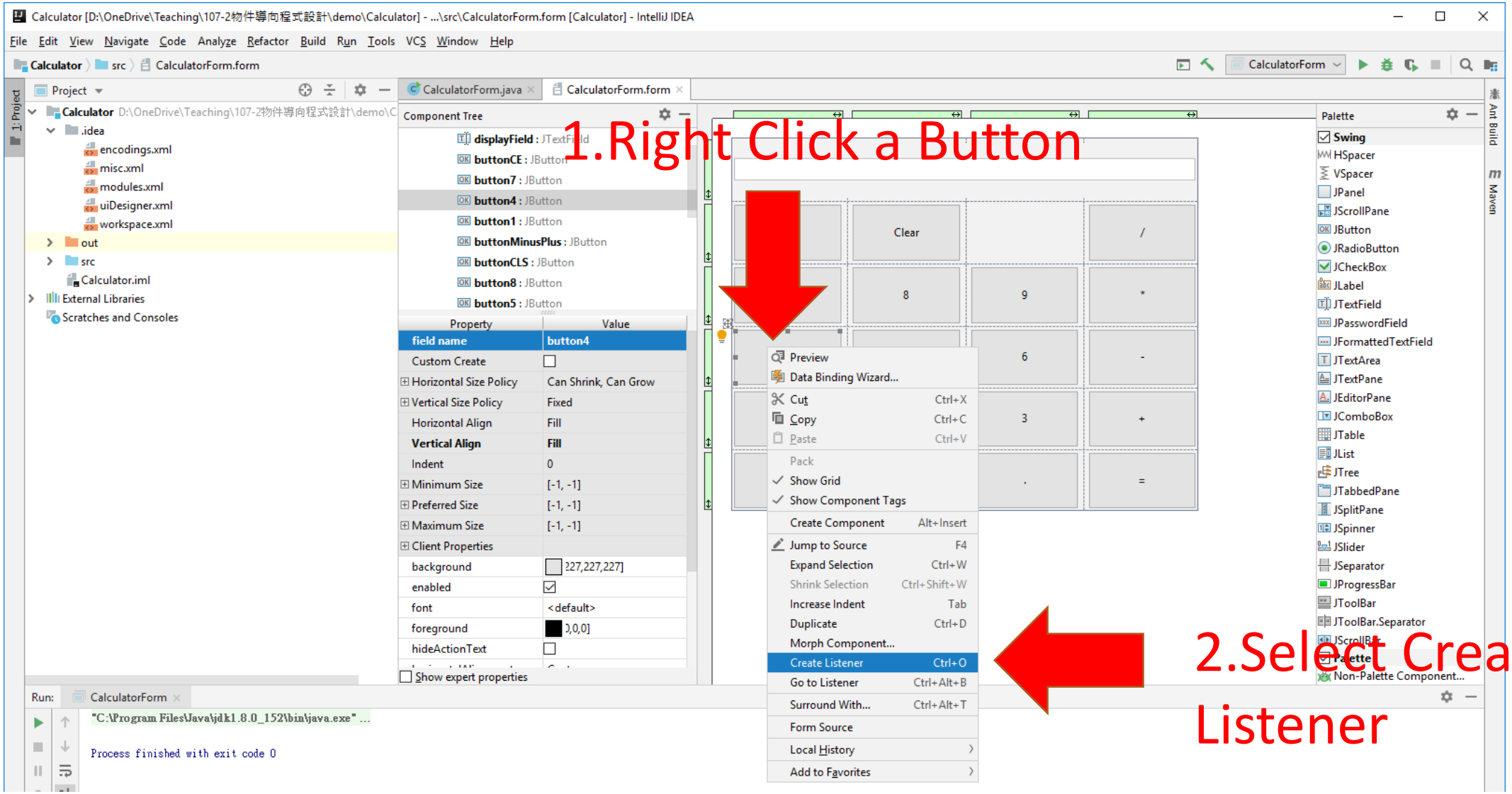
- In the code editor of Calculator.java file select -> Generate… -> Form main()



Run Main()


1.Right Click Anywhere in the Class
2.Select it
Project
Calculator2 C:\Users\User\OneDri
.idea
out
src
Calculator2.iml
External Libraries
Scratches and Consoles
CalcForm.java
CalcForm.form
private JButton button3;
private JButton button4;
private JButton button5;
private JButton button6;
private JButton button7;
private JButton button8;
private JButton button9;
private JButton button10;
private JButton button11;
private JButton button12;
private JButton button13;
private JButton button14;
private JButton button15;
private JButton button16;
Copy Reference Ctrl+Alt+Shift+C
Paste Ctrl+V
Paste from History... Ctrl+Shift+V
Paste without Formatting Ctrl+Alt+Shift+V
Column Selection Mode Alt+Shift+Insert
Find Usages Alt+F7
Refactor
Folding
Analyze
Go To
Generate... Alt+Insert
Recompile 'CalcForm.java' Ctrl+Shift+F9
Run 'CalcForm.main()' Ctrl+Shift+F10
Debug 'CalcForm.main()'
Run 'CalcForm.main()' with Coverage
Create 'CalcForm.main()'...
Show in Explorer
Open in Terminal
Local History
Compare with Clipboard
File Encoding
Create Gist...
CalcForm
Terminal
0: Messages
6: TODO
Event Log
Compilation completed successfully in 4 s 720 ms (3 minutes ago)
26:1 CRLF UTF-8 4 spaces
Maven
Ant Build
1: Project
2: Favorites
7: Structure

Create Action Listeners of Buttons

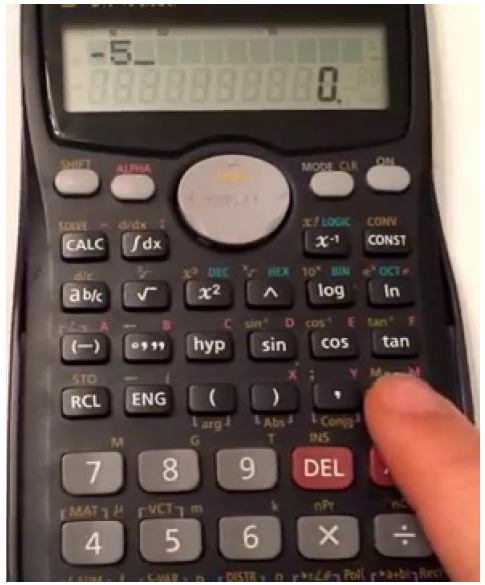


Enter Your Code in ActionListener

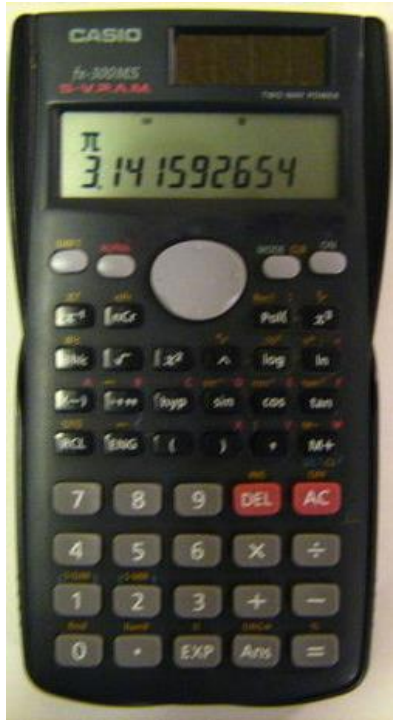
```
public class CalculatorForm {
    private JTextField displayField;
    private JPanel CalcPanel;
    private JButton buttonCE;
    private JButton button0;
    ……
    ……
    public CalculatorForm() {
        button0.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {

            }
        });
     ……
```

Enter Your Own Code Here

Two Modes in Calculator



Entering digits



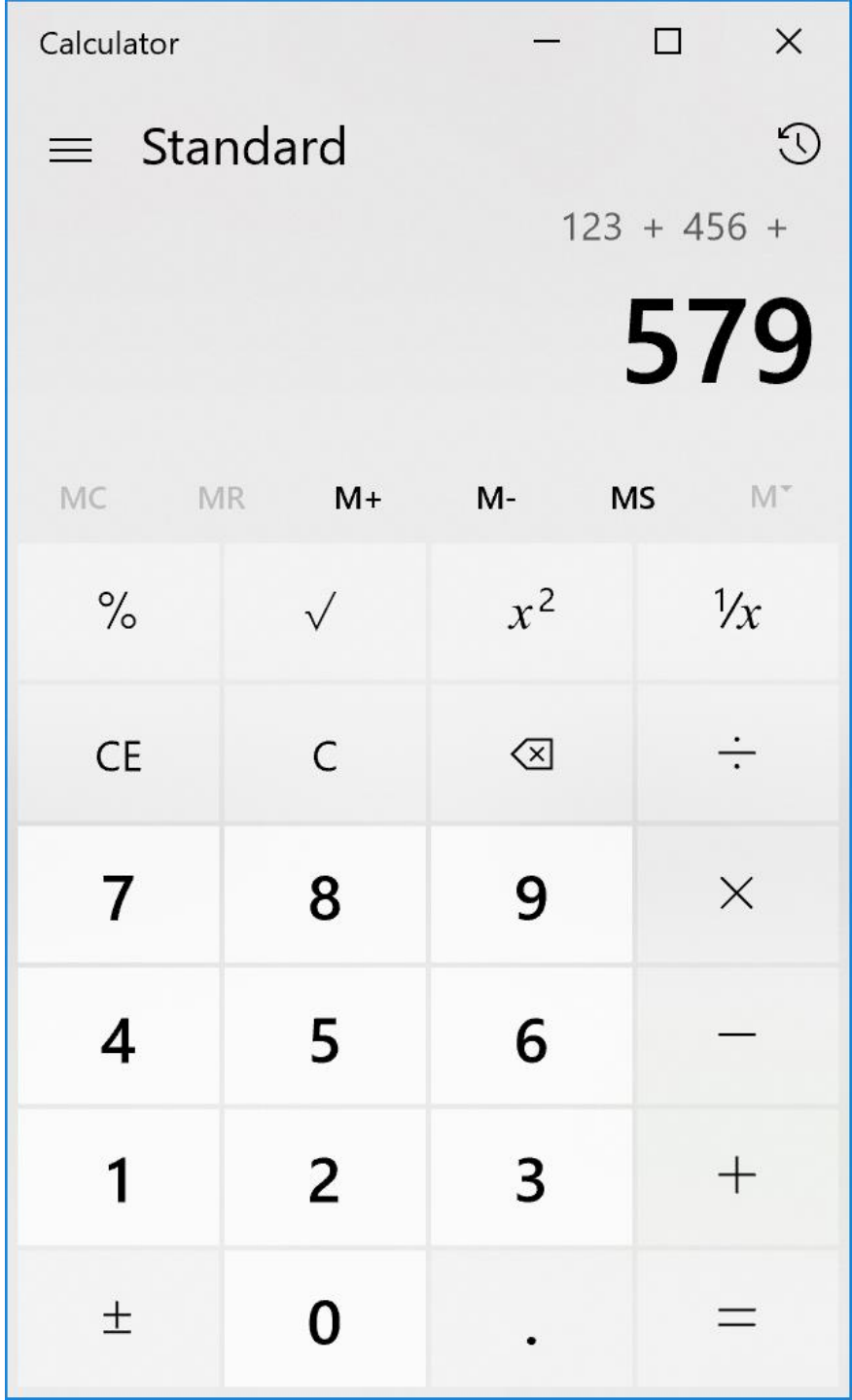
Show results (temporary or final)

Example

123 + 456 –

Show temp result ⇨ 579 –

579 – 789…



Define Variables

- Use **enum** to define operations
- Other variables
  - Mode (**isDigitEnterMode**)
  - Current display (**displayString**)
  - Temporary result (**result**)
  - Last operation (**lastOP**)

```
public class CalculatorForm {
    ......
    ......
    enum CalcOP {NONE, ADD, SUB, MULTIPLY, DIVIDE};

    private boolean isDigitEnterMode = false;
    private String displayString = "";
    private double result = 0;
    private CalcOP lastOP = CalcOP.NONE;
    ......
```

Adding Functions to Listeners of Digit Buttons

```
.........
button0.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        enterDigit("0");
    }
});
button1.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        enterDigit("1");
    }
});
button2.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        enterDigit("2");
    }
});
.........
.........
```

Entering Digits

- Call enterDigit() in each listener of digit buttons

```
private void enterDigit(String digit)
{
    if (!isDigitEnterMode) {
        if (digit == ".")
            displayString = "0.";
        else
            displayString = digit;
        isDigitEnterMode = true;
    }
    else {
        // Only floating-point number
        // can start with 0
        if (displayString == "0" && digit != ".")
            return;
        displayString += digit;
    }
    displayField.setText(displayString);
}
```

Adding Functions to Listeners of OP Buttons

```
.........
buttonMultiply.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        evalLastOP(CalcOP.MULTIPLY);
    }
});
buttonDivide.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        evalLastOP(CalcOP.DIVIDE);
    }
});
buttonEqual.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        evalLastOP(CalcOP.NONE);
    }
});
.........
.........
```

Evaluate Operations

- Evalute operators (+ - * / =)

```
private void evalLastOP(CalcOP currOP)
{
    double value = Double.parseDouble(displayField.getText());
    // Note that we evaluate last Operator, not current
    switch (lastOP) {
            case ADD:
                result += value;
                break;
            case SUB:
                result -= value;
                break;
            case DIVIDE:
                result /= value;
                break;
            case MULTIPLY:
                result *= value;
                break;
            default: // First value
                result = value;
                break;
    }
    displayField.setText(Double.toString(result));
    isDigitEnterMode = false;
    lastOP = currOP;
}
```

Create Test Interfaces

- Need to provide two public test interfaces:

1. `public void testClick(String button) throws Exception`

2.
```
public double getResult() {
    return result;
}
```

3. `public void showWindow()`

```
public void testClick(String button) throws Exception
{
    switch (button)
    {
        case "+": buttonAdd.doClick(); break;
        case "-": buttonSub.doClick(); break;
        case "*": buttonMultiply.doClick(); break;
        case "/": buttonDivide.doClick(); break;
        case ".": buttonDot.doClick(); break;
        case "=": buttonDivide.doClick(); break;
        case "±": buttonMinusPlus.doClick(); break;
        case "CE": buttonCE.doClick(); break;
        case "CLEAR": buttonCLS.doClick(); break;
        case "0": button0.doClick(); break;
        case "1": button1.doClick(); break;
        case "2": button2.doClick(); break;
        case "3": button3.doClick(); break;
        case "4": button4.doClick(); break;
        case "5": button5.doClick(); break;
        case "6": button6.doClick(); break;
        case "7": button7.doClick(); break;
        case "8": button8.doClick(); break;
        case "9": button9.doClick(); break;
        default:
            throw new Exception("Error! No button " + button);
    }
}
```

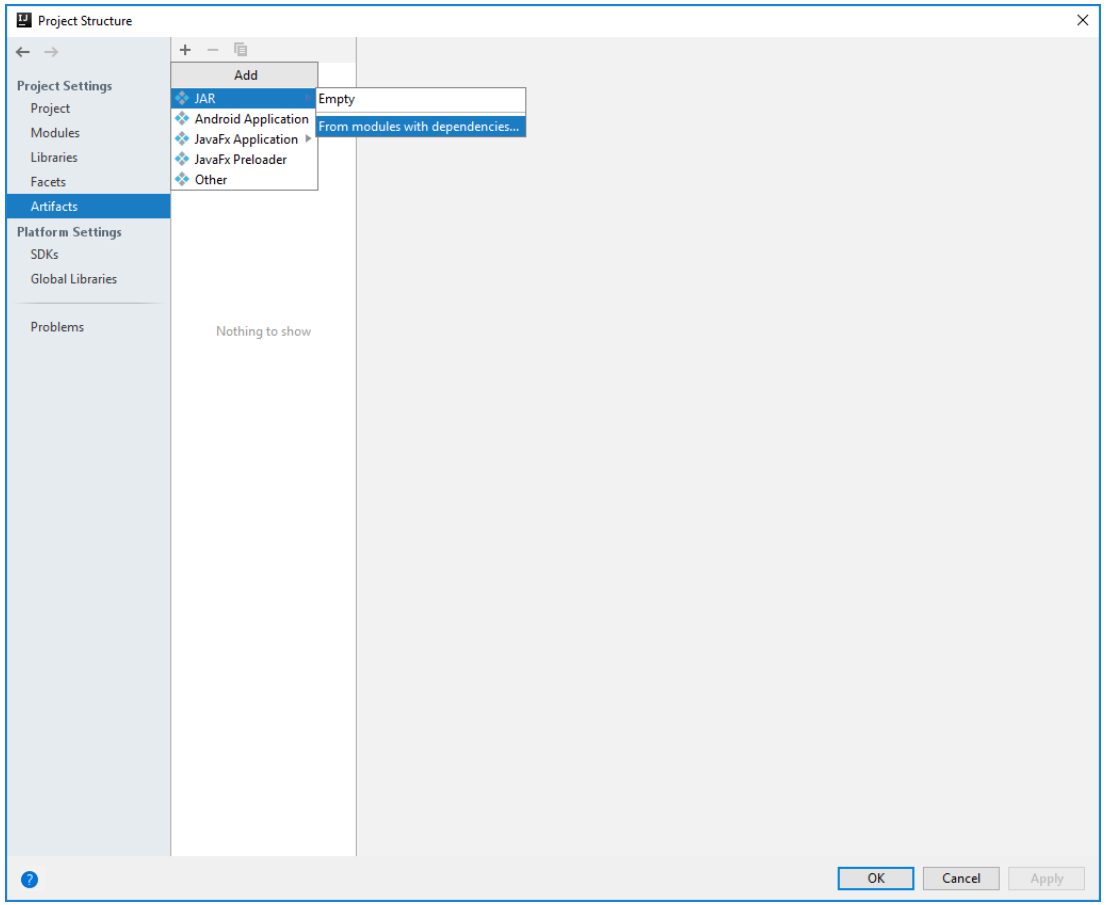
Show the Testing Process

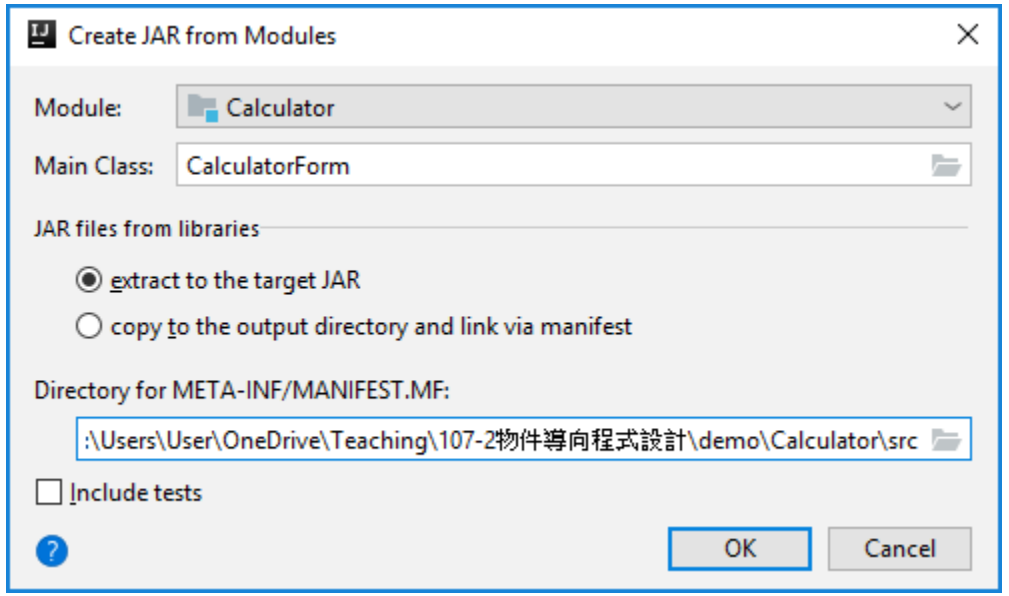
Define a public function `public void showWindow()`

```
public void showWindow() {
    JFrame frame = new JFrame("Calculator");
    frame.setContentPane(this.CalcPanel);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.pack();
    frame.setVisible(true);
}
```

Build a Jar

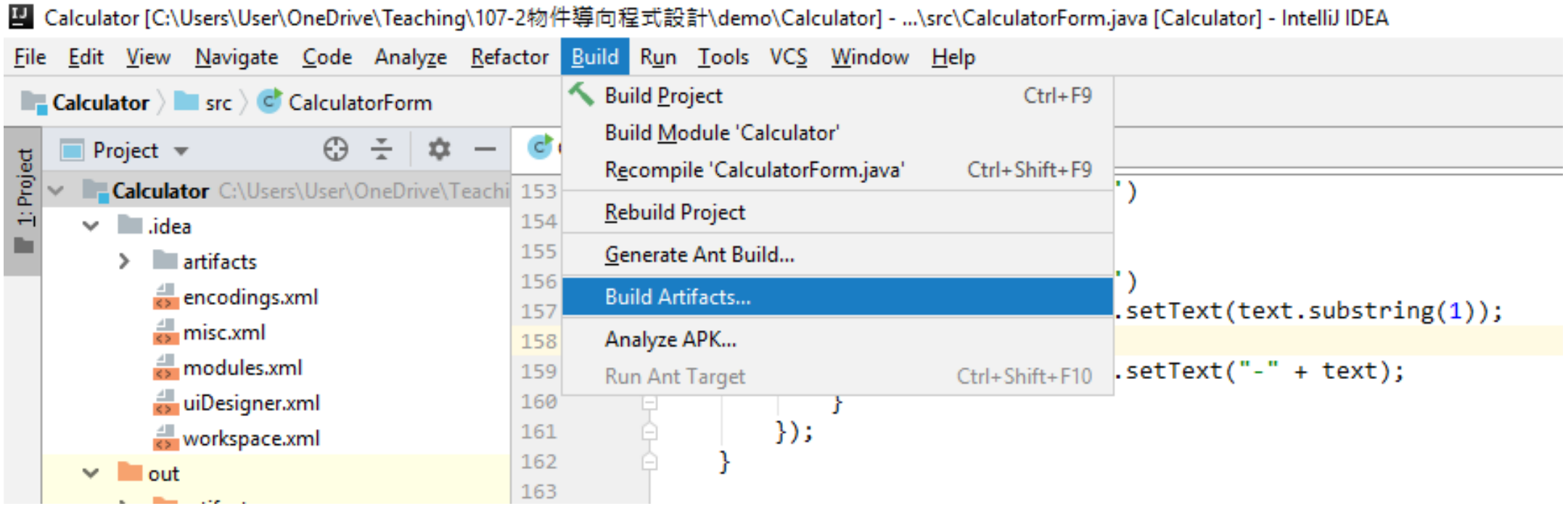
- File -> Project Structure -> Project Settings -> Artifacts -> Click green plus sign -> Jar -> From modules with dependencies...





Build a Jar (Cont'd)

- Build | Build Artifact



Testing Your Calculator with JUnit

- Download junit-4.12.jar & hamcrest-core-1.3.jar
- Download CalculatorFormTest.java

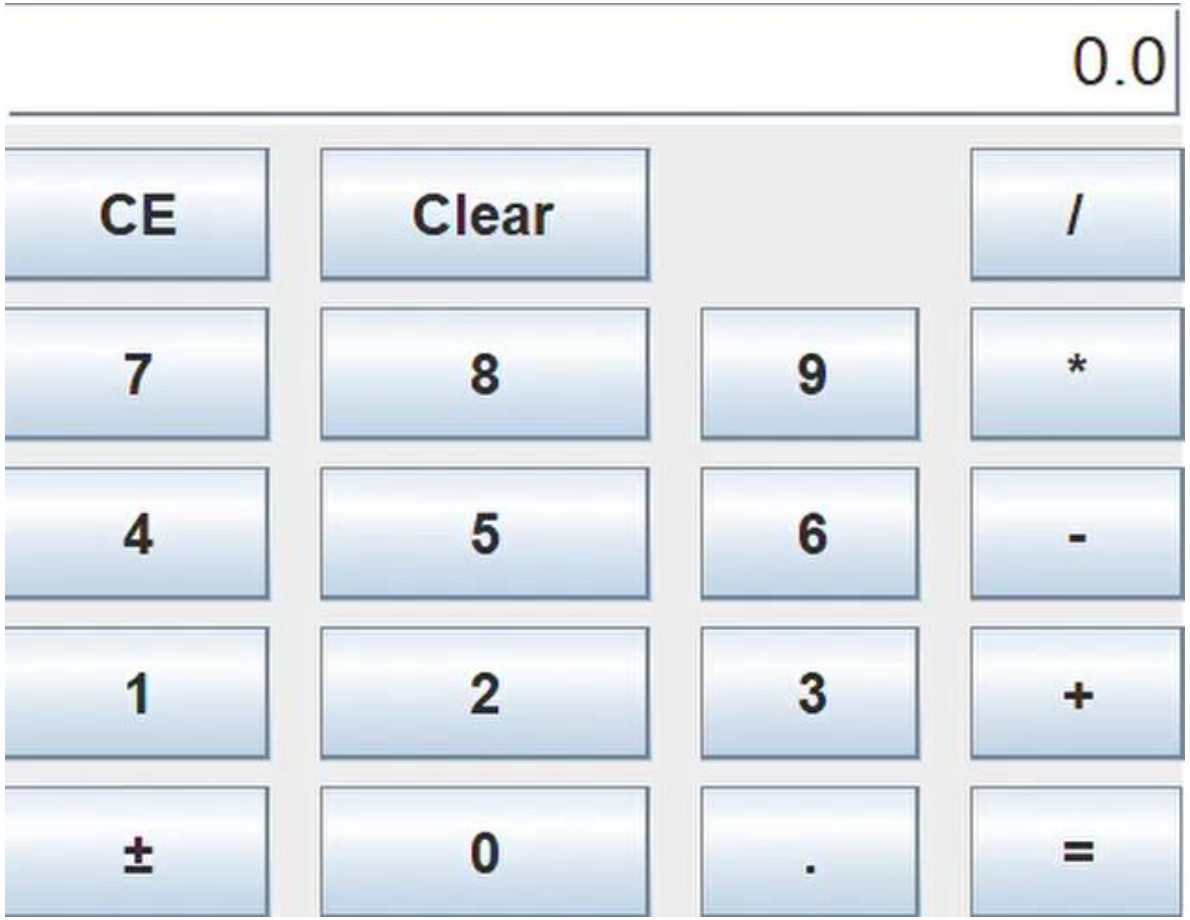
- Compile CalculatorFormTest.java with your jar

```
C:\> javac -cp ".;junit-4.12.jar;Calculator.jar" CalculatorFormTest.java
```

- Run the Test

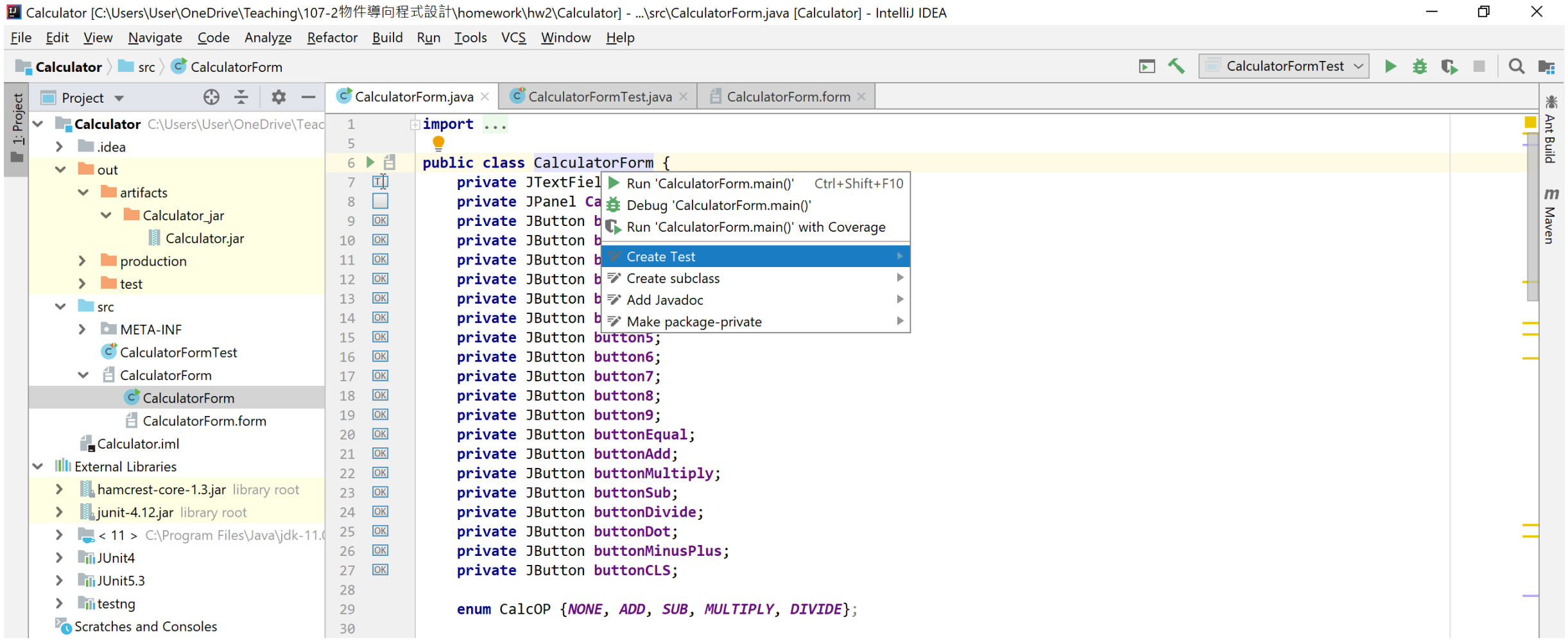
```
C:\> java -cp ".;junit-4.12.jar;Calculator.jar" CalculatorFormTest
```

Testing Process



Create Unit Testing using IntelliJ

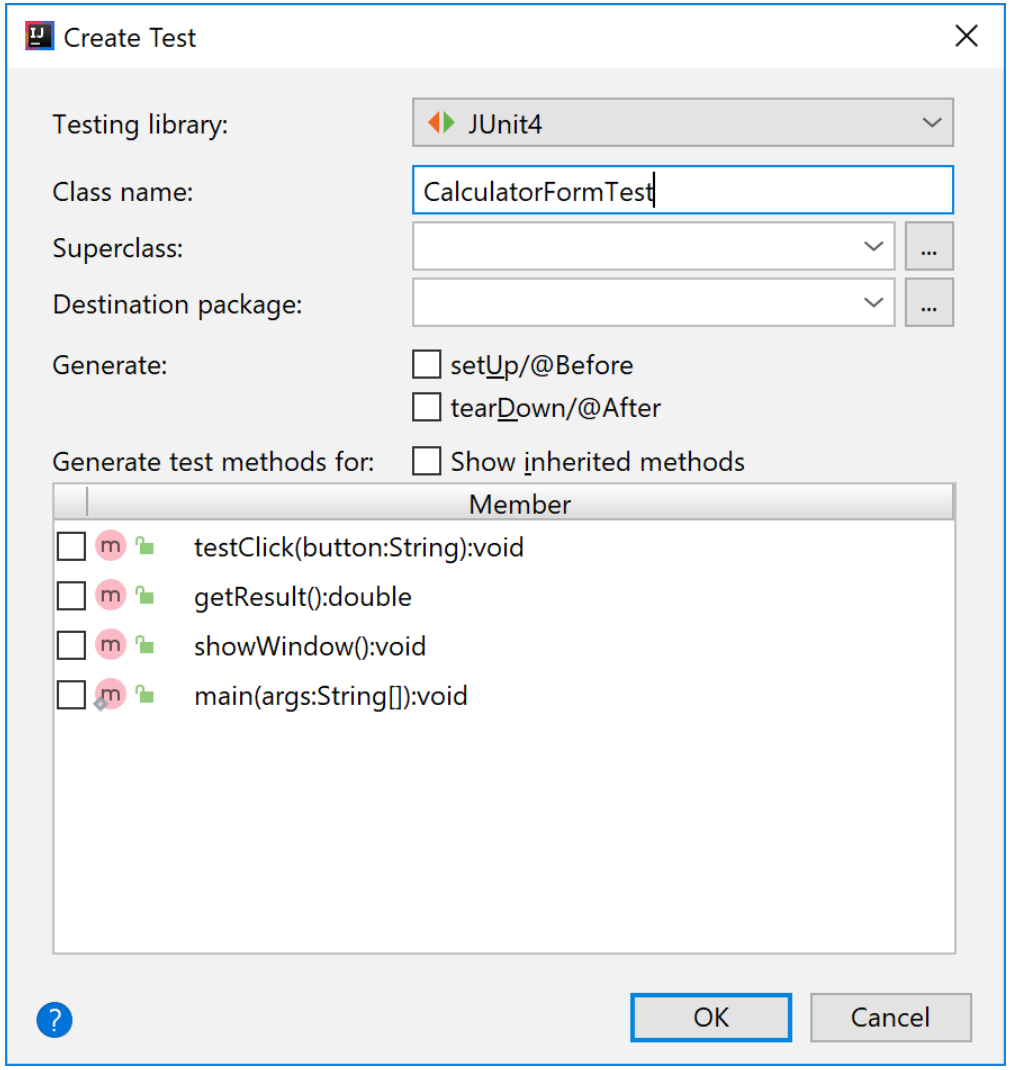
Select Your Class and Press “Alt + Enter”




```
1   import ...
5
6   public class CalculatorForm {
7       private JTextFiel
8       private JPanel Ca
9       private JButton b
10      private JButton b
11      private JButton b
12      private JButton b
13      private JButton b
14      private JButton b
15      private JButton button5;
16      private JButton button6;
17      private JButton button7;
18      private JButton button8;
19      private JButton button9;
20      private JButton buttonEqual;
21      private JButton buttonAdd;
22      private JButton buttonMultiply;
23      private JButton buttonSub;
24      private JButton buttonDivide;
25      private JButton buttonDot;
26      private JButton buttonMinusPlus;
27      private JButton buttonCLS;
28
29      enum CalcOP {NONE, ADD, SUB, MULTIPLY, DIVIDE};
30
```

Select Your Test Framework

- We use JUnit4 here.



Using Junit Test & Runner

Junit Test Unit

Junit Runner

```
import org.junit.Test;
import static org.junit.Assert.*;
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;

public class CalculatorFormTest {
    static CalculatorForm calc = new CalculatorForm();
    @ Test
    public void testAddSub() {
        try {
            calc.testClick("CLEAR");
            calc.testClick(“1"); calc.testClick(“+");
            calc.testClick(“2"); calc.testClick(“=");
        }
        catch (Exception e) {
            System.out.println(e.getMessage());
        }
        double result = calc.getResult();
        assertEquals(3, result, 0);
    }
    public static void main(String[] args) {
        calc.showWindow();
        Result result = JUnitCore.runClasses(CalculatorFormTest.class);
        for (Failure failure : result.getFailures()) {
             System.out.println(failure.toString());
        }
        System.out.println(result.wasSuccessful());
    }
}
```

Run Your Test

- Press “Alt + Tab” on your test class