Java & Graphical User Interface II

Wang Yang
wyang AT njnet.edu.cn

Outline

- Review of GUI (first part)
- What is Event
- Basic Elements of Event Programming
- Secret Weapon - Inner Class
- Full version of Event Programming

Review of GUI Programming (first part)

GUI Programming

- Library
  - java.awt.*
    - Contains all of the classes for creating user interfaces and for painting graphics and images.
  - javax.swing.*
    - a set of "lightweight" (written in Java with no native code) components
  - sub packages:
    - java.awt.font.*
    - javax.swing.border.*

GUI Programming

- Why sub packages?
    - some feature is so important and complicated that need write a bundle of classes/interfaces for it

- **javax.swing** - Provides a set of "lightweight" (written in Java with no native code) components that, to the maximum degree possible, work the on all platforms.
- **javax.swing.border** - Provides classes and interfaces for drawing specialized borders around a Swing component.
- **javax.swing.colorchooser** - Contains classes and interfaces used by the `JColorChooser` component.
- **javax.swing.event** - Provides support for events fired by Swing components.
- **javax.swing.filechooser** - Contains classes and interfaces used by the `JFileChooser` component.
- **javax.swing.plaf** - Provides one interface and many abstract classes that Swing uses to provide its pluggable look and feel capabilities.
- **javax.swing.plaf.basic** - Provides user interface objects built according to the Basic look and feel.
- **javax.swing.plaf.metal** - Provides user interface objects built according to the Java look and feel (once codenamed *Metal*), which is the default and feel.
- **javax.swing.plaf.multi** - Provides user interface objects that combine two or more look and feels.
- **javax.swing.plaf.synth** - Provides user interface objects for a skinnable look and feel in which all painting is delegated.
- **javax.swing.table** - Provides classes and interfaces for dealing with `JTable`.
- **javax.swing.text** - Provides classes and interfaces that deal with editable and non-editable text components.
- **javax.swing.text.html** - Provides the class `HTMLEditorKit` and supporting classes for creating HTML text editors.
- **javax.swing.text.html.parser** - Provides the default HTML parser, along with support classes.
- **javax.swing.text.rtf** - Provides a class (`RTFEditorKit`) for creating Rich Text Format text editors.
- **javax.swing.tree** - Provides classes and interfaces for dealing with `JTree`.
- **javax.swing.undo** - Allows developers to provide support for undo/redo in applications such as text editors.

GUI Programming

- Container
  - Top-Container
    - JFrame, JApplet, JDialog
      - manage container & component
- Container
  - JPanel, JScrollPane, JSplitPane
  - manage container & component
  - container can be nested

GUI Programming

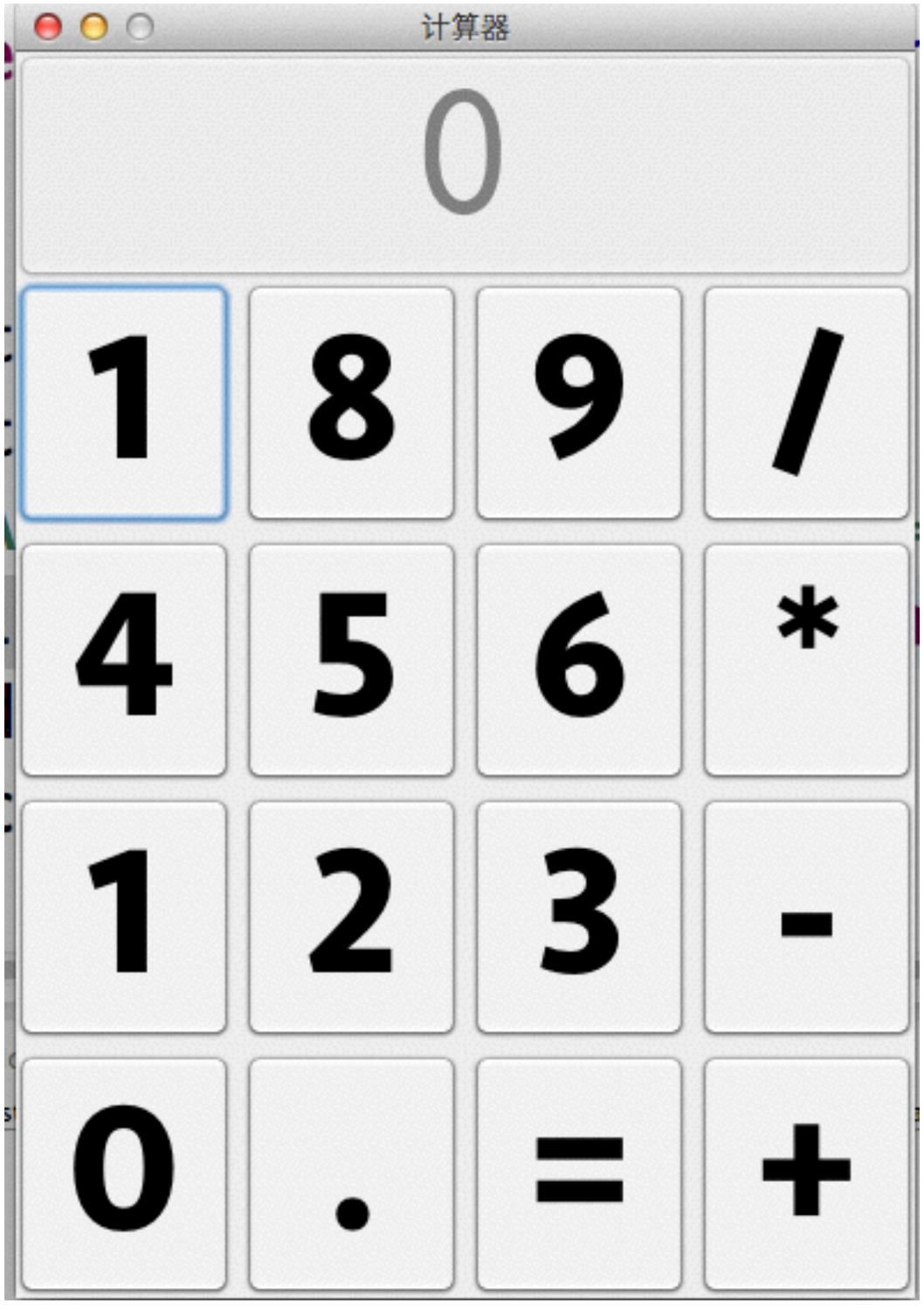
- Component:
  - Text Component : label ,TextField, TextArea
  - Button Component : Button, CheckBox, RadioButton
  - A lot of other Component:

GUI Progamming

- Layout Manager
- manage the position and the size of component automatically
- BorderLayout : JFrame
- FlowLayout : JPanel
- others : GridLayout, BoxLayout, ….

What is Event

We have a calculator



We want to use it

76+22

Different Way to input

- Mouse Input
  - move : we enter a component area or we leave a component area
    - we enter “7” button
    - we leave “7 button, and we enter “8” button
- click : we press the mouse button down and release
  - we click “7” button : we want input 7 into calculator

Different Way to input

- Keyboard
  - char typed : a Unicode character is entered
    - we typed “7”, we want input the 7 to the Cal.
  - code typed : combined key entered, not valid Unicode character
    - input “ALT+F4”, we want close the Cal.

what happened when we input ?

- when we click the “7” button
  - the JVM generate a ActionEvent()
- when we type the “7” in keyboard
  - the JVM generate a KeyEvent()

What is Event

- EventObject
  - The root class from which all event state objects
    - source : The object on which the Event initially occurred.
- AWTEvent
  - source + type :
    - who create : source component
    - what type : ACTION_EVENT_MASK, KEY_EVENT_MASK

More Specific Event

- Each Specific Event has its own fields and methods
- ActionEvent
  - When： the timestamp of when this event occurred
  - Command： the command string associated with this action
- KeyEvent
  - getKeyChar : Returns the character associated with the key in this event
  - getKeyCode : Returns the integer keyCode associated with the key in this event.

Semantic Event

- ActionEvent is a Semantic Event
  - high-level event is generated by a component
  - including series of low-level event
  - component-defined
    - action for button : click, type space key
    - semantic : the button is executed
  - Other Semantic Event : DocumentEvent

Low-level Event

- KeyEvent/MouseEvent is a Low-level Event
  - event generated by the input device
  - more detail than semantic event
    - MouseEvent : press, release, click, enter, and exit
    - when to use : provide tooltips

Basic Elements of Event Programming

We need Sth. to handle the Event

- Event Handler
  - The handler of events
  - An object receiving events and process them
  - response for change components' state
  - EventListener : A tagging interface that all event listener interfaces must extend.

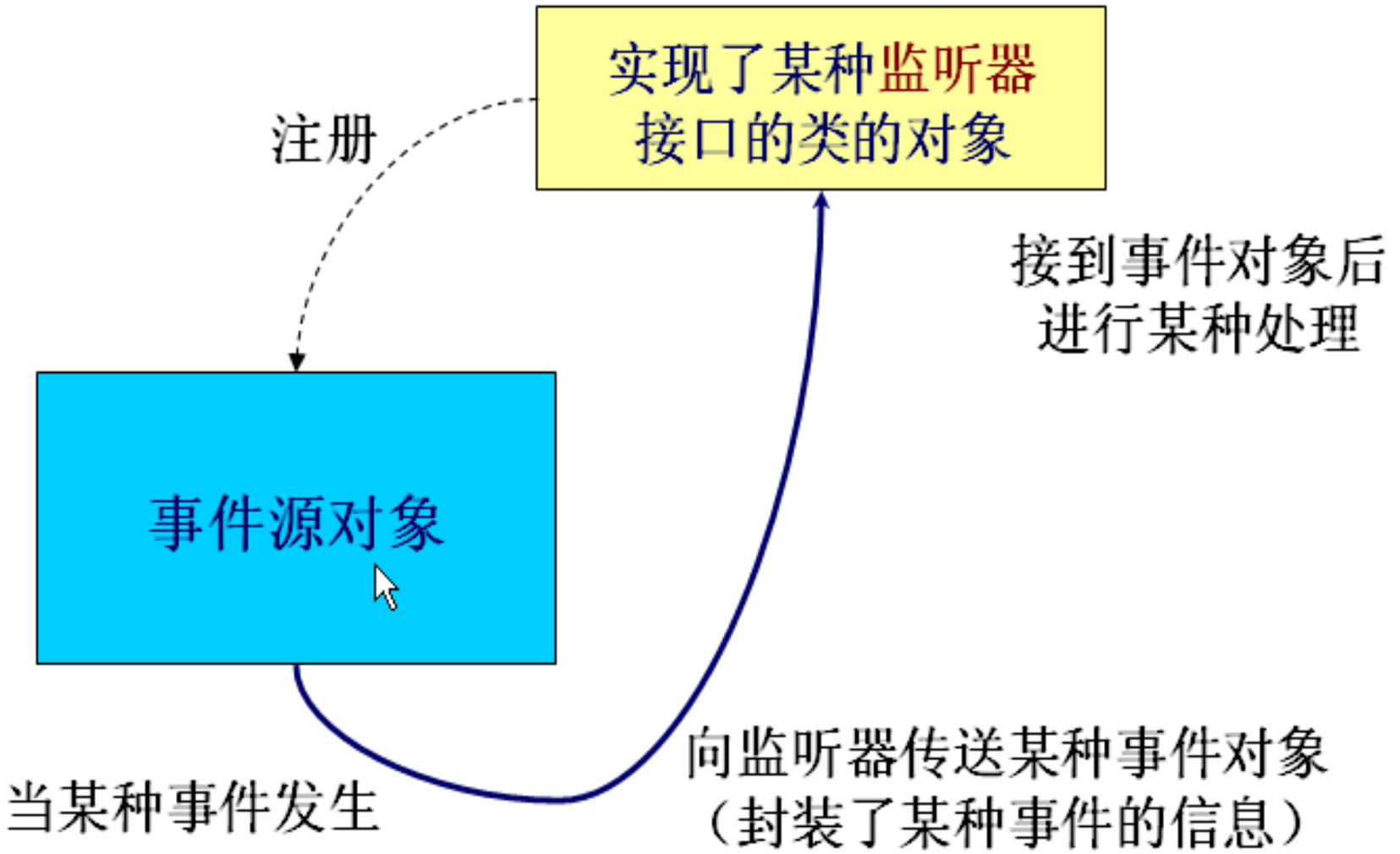
Event Handler

- Each Event has a corresponding Handler
  - ActionEvent : ActionListener
  - KeyEvent : KeyListener
  - MouseEvent : MouseListener
- Each Handler has specific methods.
  - ActionListener : actionPerformed()
  - MouseListener : MouseEnterer(), MousePressed() ….

Connect Handler to Source

- Each component has method register right handler for specific event
- JButton : addActionListener(ActionListener l)
- JTextField : addKeyListener(KeyListener k)

Event Handling Model



Example

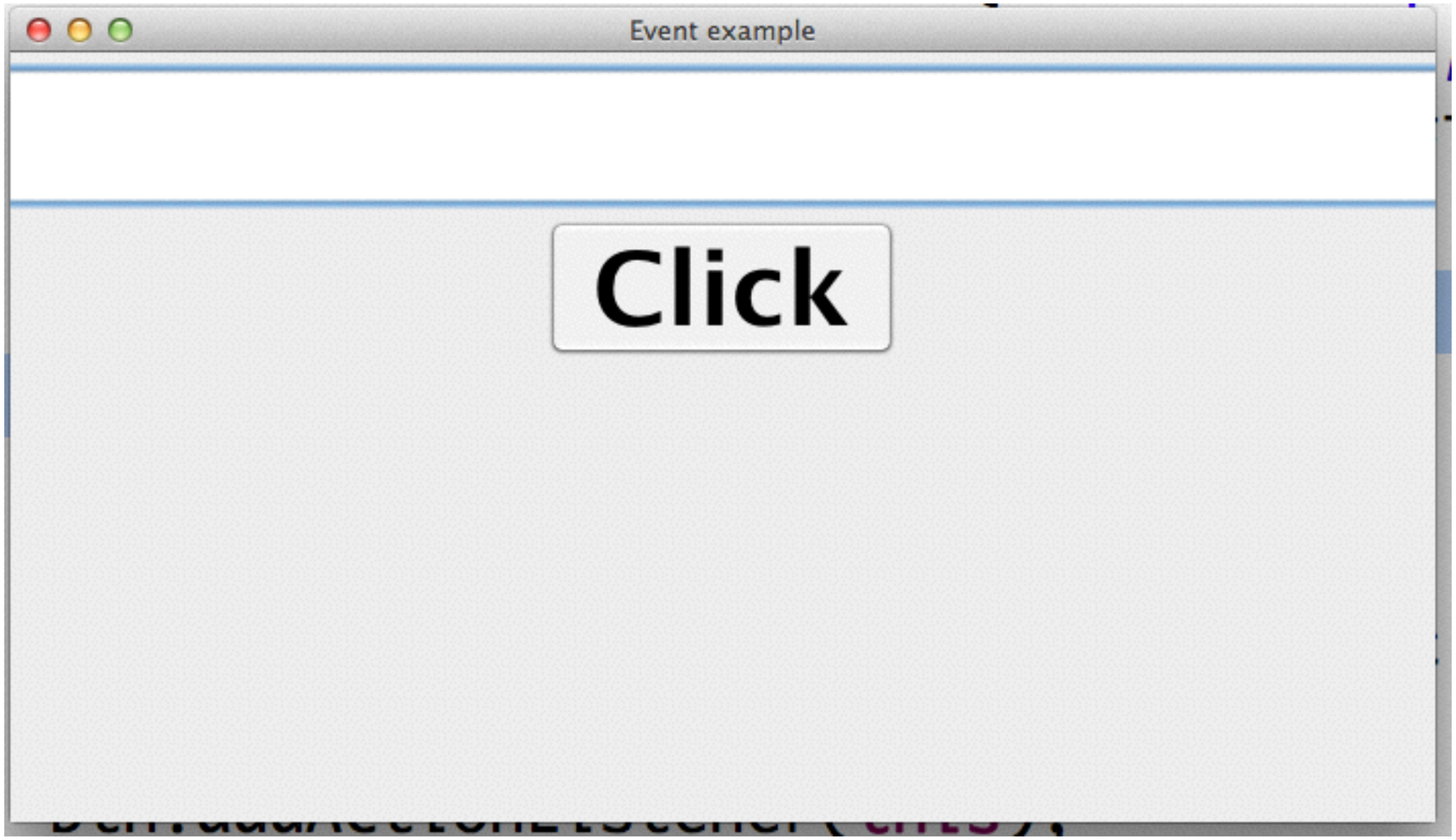
```
public class ActionDemo3 implements ActionListener{
private JTextField textField;

    public void actionPerformed(ActionEvent e){
        textField.setText("Button clicked");
    }
```

Example

```
public ActionDemo3(){
      JFrame frame = new JFrame("Event example");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE
      frame.getContentPane().setLayout(new FlowLayout());
      textField = new JTextField();
      textField.setColumns(18);
      frame.add(textField);
      JButton btn = new JButton("Click");
      frame.add(btn);
      btn.addActionListener(this);
      frame.setSize(700, 400);
      frame.setVisible(true);
}
```

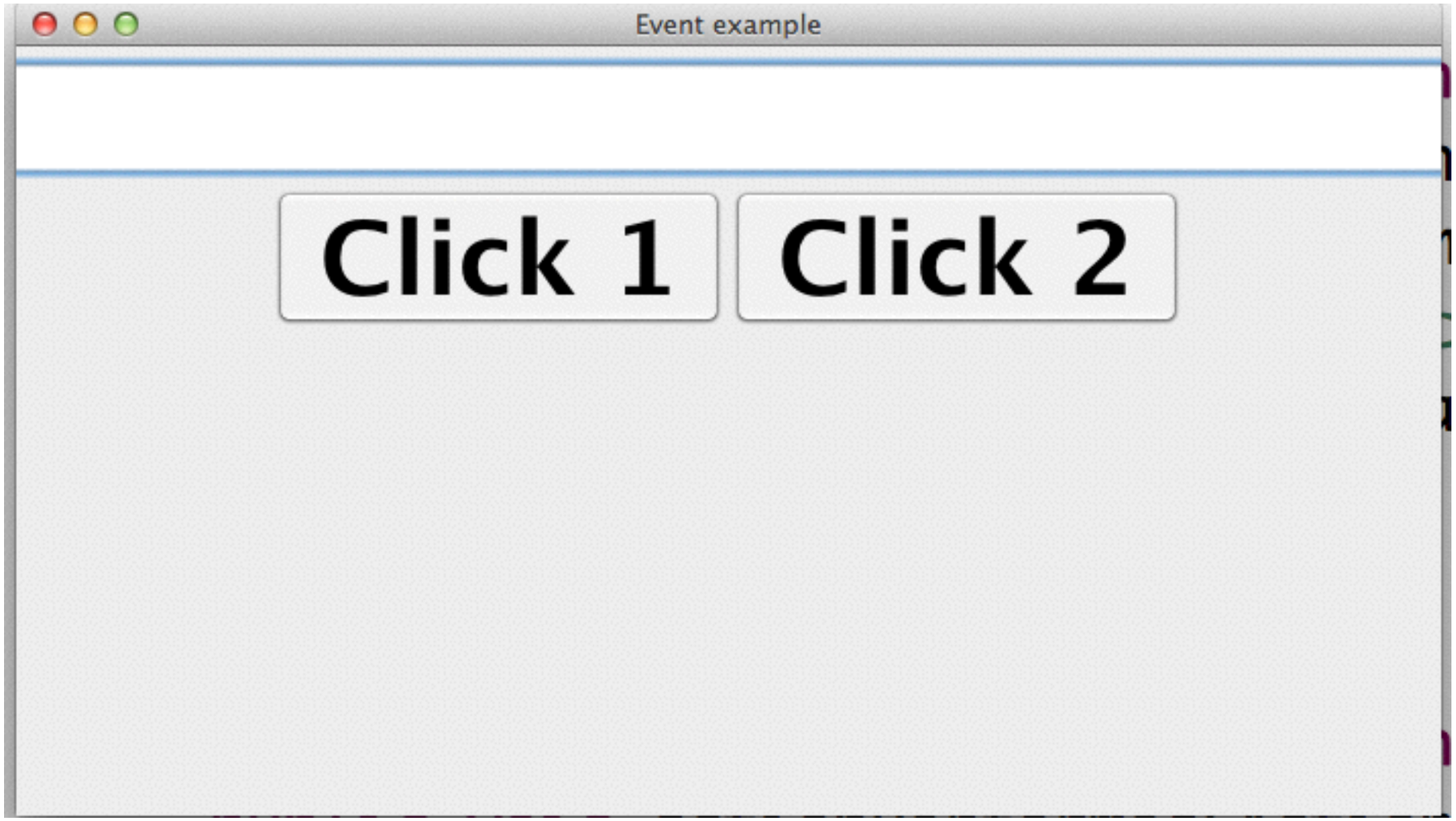
Example

```
public static void main(String[] args) {
    ActionDemo3 demo = new ActionDemo3();
}
```



When we click the button

But if we want more component



First way, we try duplicate two methods

```
public void actionPerformed(ActionEvent e){
    textField.setText("Button 1 clicked");
}

public void actionPerformed(ActionEvent e){
    textField.setText("Button 2 clicked");
}
```

First way, we try duplicate two methods

```
public void actionPerformed(ActionEvent e){
    textField.setText("Button 1 clicked");
}

public void actionPerformed(ActionEvent e){
    textField.setText("Button 2 clicked");
}
```

The Compiler won't allow the duplicate method

Then we try see the source of event

```
public void actionPerformed(ActionEvent e){
    if(e.getSource() == btn1)
        textField.setText("Button 1 clicked");
    else
        textField.setText("Button 2 clicked");
}
```

Then we try see the source of event

```
public void actionPerformed(ActionEvent e){
    if(e.getSource() == btn1)
        textField.setText("Button 1 clicked");
    else
        textField.setText("Button 2 clicked");
}
```

It worked , but you combine different component logic together

Last we try get the handler out of GUI

```
public class ButtonListener1 implements ActionListener{
    public void actionPerformed(ActionEvent e){
        textField.setText("Button 1 clicked");
    }
}

public class ButtonListener2 implements ActionListener{
    public void actionPerformed(ActionEvent e){
        textField.setText("Button 2 clicked");
    }
}

public class ActionDemo3 {
private JTextField textField;
```

Last we try get the handler out of GUI

```
public class ButtonListener1 implements ActionListener{
    public void actionPerformed(ActionEvent e){
        textField.setText("Button 1 clicked");
    }
}
```

how can you access the textField…

```
public class ButtonListener2 implements ActionListener{
    public void actionPerformed(ActionEvent e){
        textField.setText("Button 2 clicked");
    }
}
```

```
public class ActionDemo3 {
private JTextField textField;
```

We want a help

- Each component has its own handler
- the handler can access other components easily
- the code will be easy to read and check

Secret Weapon - Inner Class

What is an Inner Class

- Inner Class :
- nested class
- define inside an outside class

```
public class Outside(){

    private class Inner(){

    }
}
```

Write in One file doesn't mean inner class

```
public class Outside(){

}

private class Inner(){

}
```

Not Inner Class !!!!

benefits of inner class

- An inner class gets a special pass to use the outer class's stuff. Even the private stuff.

- they have most of the benefits of a normal class. but with special access rights.

```
public class Outside{
    private int x;
    private class Inner{
        public go(){
            x = 42;
        }
    }
}
```

restrict of inner class

- An Inner class Must be tied to at 1 outer class instance

- you can’t new an inner class directly outside the outer class.

New Inner Class in the Outer Class

It is allowed

```
public class Outside{
    private int x;
    Inner i = new Inner();
    private class Inner{

    }
}
```

New Inner Class through an Outer Class

```
public class Outside{
    private int x;

    private class Inner{

    }
    public static void main(String args){
        Outside out = new Outside();
        Outside.Inner i = out.new Inner();
    }
}
```

It is allowed, but not suggest

New Inner Class Directly out of the Outer Class

```
public class Outside{
    private int x;
    public class Inner{
        public void go(){
            x = 42;
        }
    }
}
public class OtherClass{
    public static void main(String args){
        Outside.Inner i = new Outside.Inner();
    }
}
```

It is not allowed

New Inner Class Directly out of the Outer Class

```
public class Outside{
    private int x;
    public class Inner{
        public void go(){
            x = 42;
        }
    }
}
public class OtherClass{
    public static void main(String args){
        Outside.Inner i = new Outside.Inner();
    }
}
```

It is not allowed

What x should i access ?

Full version of Event Programming

Use inner class define the Handler

- Use Inner class define the different Handler
- you can define different Handler classes for different Listener Class
- you also can define different Handler classes for same listener class

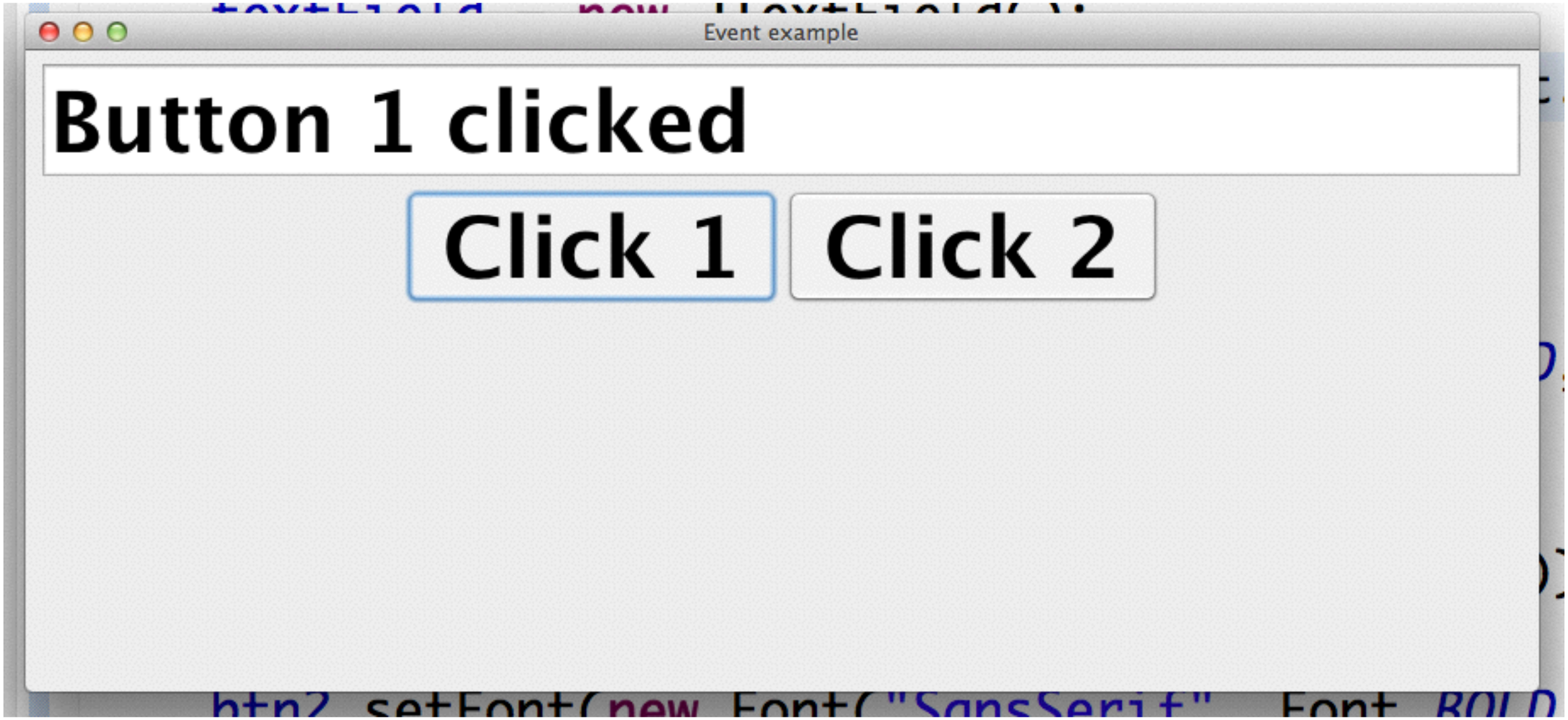
Example

```
public class ActionDemo2{
    private JTextField textField;
    private JButton btn1;
    private JButton btn2;
    private class ButtonListener1 implements ActionList
        public void actionPerformed(ActionEvent e){
            textField.setText("Button 1 clicked");
        }
    }
    private class ButtonListener2 implements ActionList
        public void actionPerformed(ActionEvent e){
            textField.setText("Button 2 clicked");
        }
    }
```

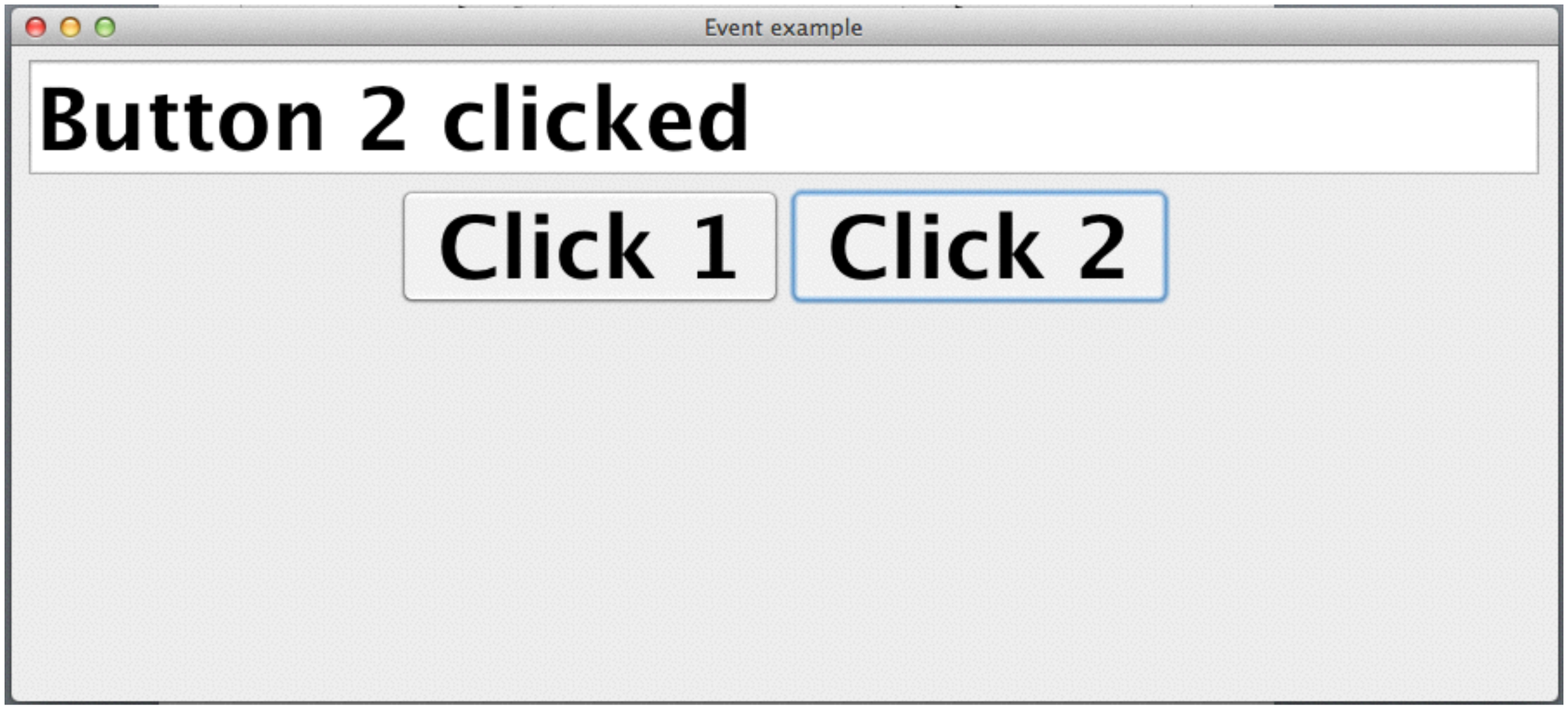
Example

```
public ActionDemo2(){
      JFrame frame = new JFrame("Event example");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_C
      frame.getContentPane().setLayout(new FlowLayout
      textField = new JTextField();
      textField.setColumns(18);
      frame.add(textField);
      btn1 = new JButton("Click 1");
      frame.add(btn1);
      btn1.addActionListener(new ButtonListen1());
      btn2 = new JButton("Click 2");
      frame.add(btn2);
      btn2.addActionListener(new ButtonListen2());
      frame.setSize(700, 400);
```

Example



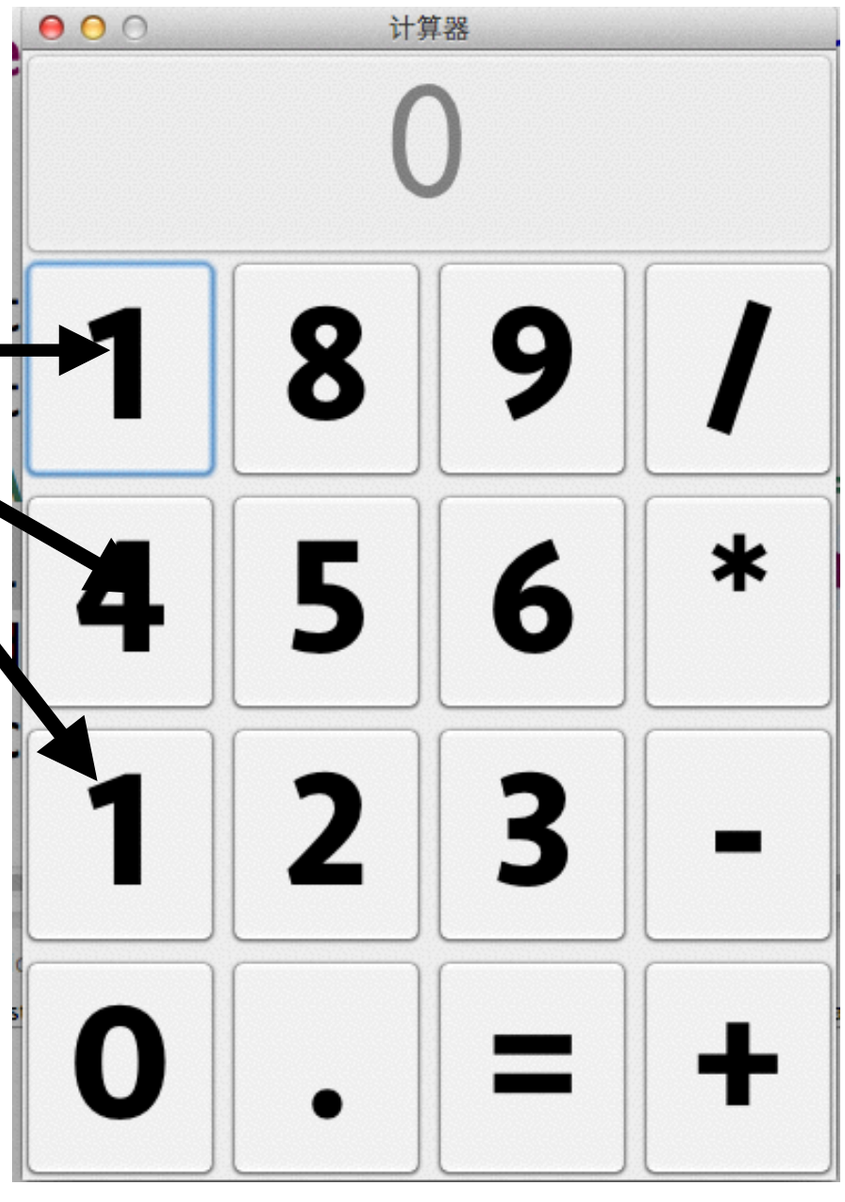
Example



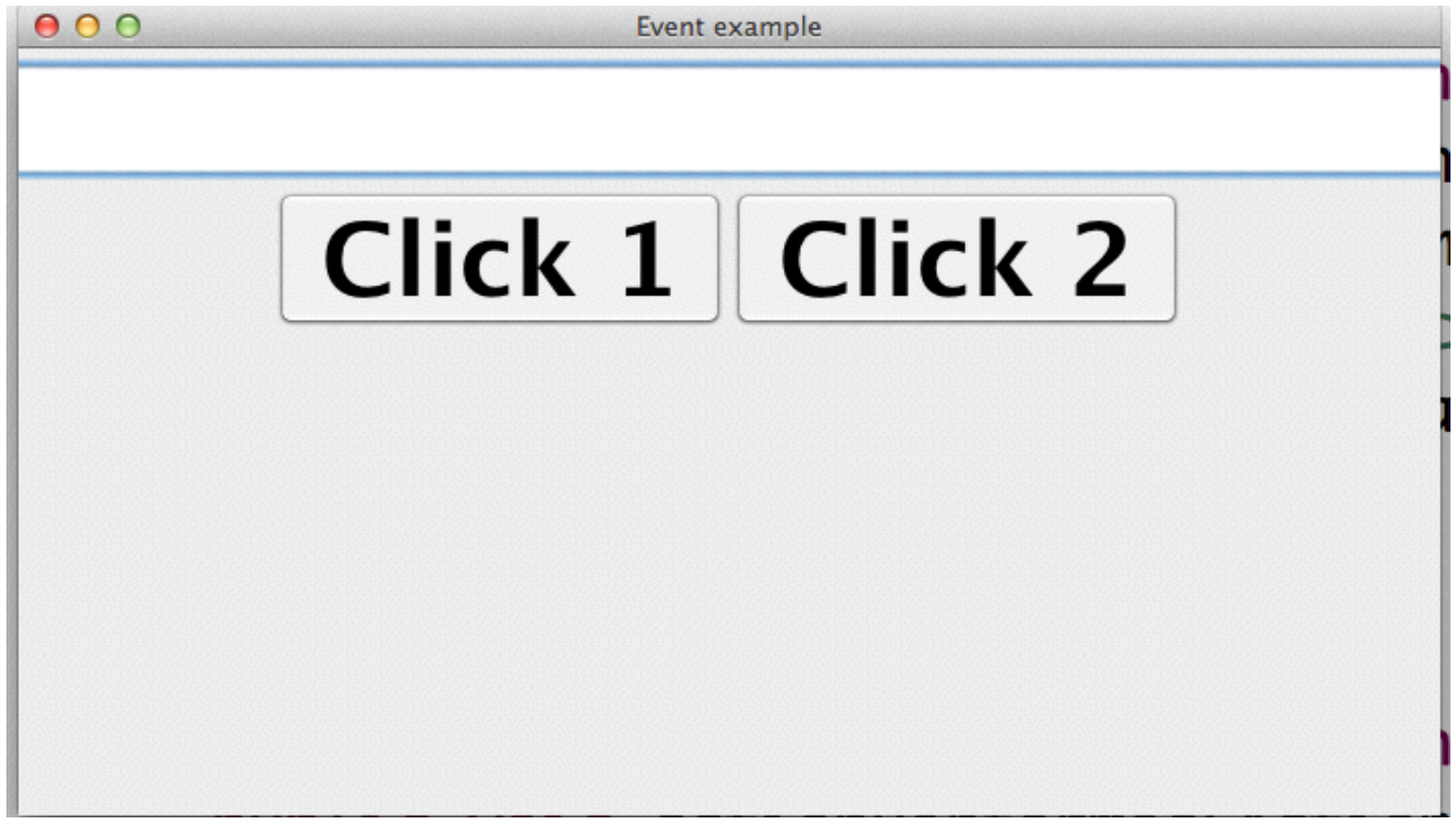
Inner Class is not the end

- Use event's field or method to combine components for same use

all the number buttons do the same thing except the "command"



Example




```
private class ButtonListenSimple implements ActionListe
    public void actionPerformed(ActionEvent e){
        textField.setText(e.getActionCommand());
    }
}
```