

Creating a GUI with JFC/Swing



<http://www.cp.eng.chula.ac.th/~chate/2110210/gui.pdf>

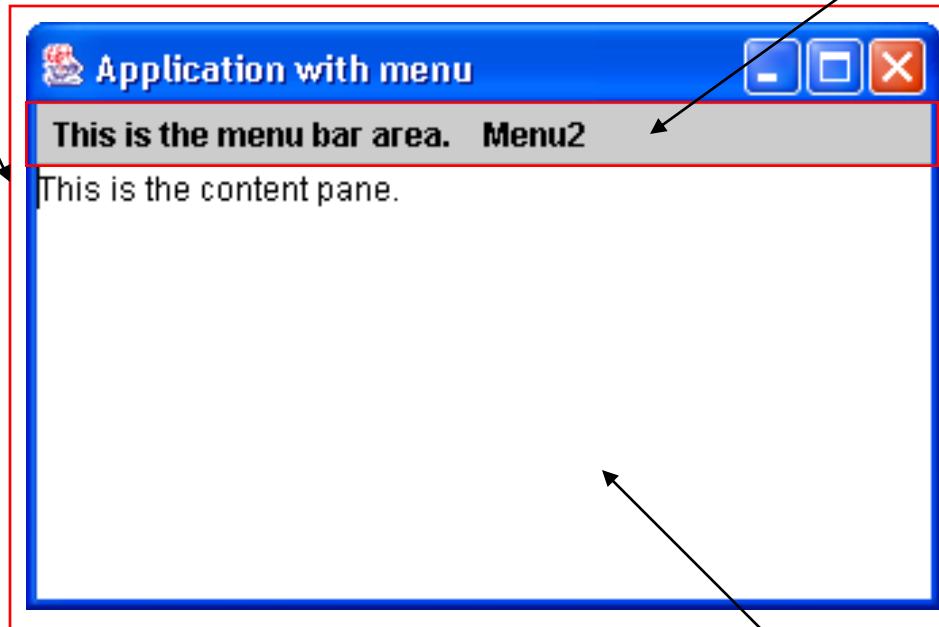
JFC and Swing คืออะไร?

- JFC
 - Java Foundation Classes
 - a group of features to help people build graphical user interfaces (GUIs)
- Swing
 - Components for GUIs
 - to use Swing, you have to import `javax.swing` package.

What are the objects in an application?

Frame – the top-level container class

Menu Bar -- optional



Content Pane – contains the visible components in the top-level container's GUI

How to make frames (main windows)?

- A frame, an instance of the **JFrame**, is a window that typically has decorations such as a border, a title, and a buttons for closing and iconifying the window.
- Every GUI components must be put into a **container**.
- Each GUI components can be contained only once.
- A frame has a **content pane** that contains the visible components
- An optional menu bar can be added to a top-level container

```
import java.awt.*;
import javax.swing.*;

public class App1 {

    public static void main(String[] args) {

        // 1. Optional: Specify who draws the window
        //     decorations. (default: native window system)
        JFrame.setDefaultLookAndFeelDecorated(true);

        // 2. Create a top-level frame
        JFrame frame = new JFrame("Application 1");

        // 3. Optional: What happens when the frame closes?
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // 4. Optional: How the components are put in the
        //     frame?
        Container cp = frame.getContentPane();
        cp.setLayout(new FlowLayout());
```

```
// 5. Create GUI/Swing components
```

```
JButton button1 = new JButton("A JButton");
```

```
JButton exitButton = new JButton(" Exit ");
```

```
JTextField text =
```

```
    new JTextField("This is a text field.", 20);
```

```
// 6. Put the components in the frame
```

```
cp.add(button1, BorderLayout.WEST);
```

```
cp.add(text, BorderLayout.CENTER);
```

```
cp.add(exitButton, BorderLayout.EAST);
```

```
// 7. Set frame size
```

```
//    frame.setSize(int width, int hieght);
```

```
frame.pack();
```

```
// 8. Show it
```

```
frame.setVisible(true);
```

```
}
```

```
}
```

frame.pack() ;

- without

```
JFrame.setDefaultLookAndFeelDecorated(true) ;
```

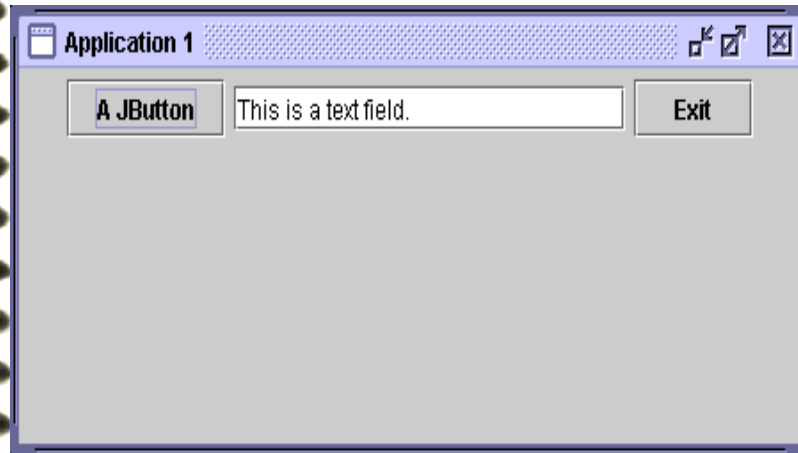


- with

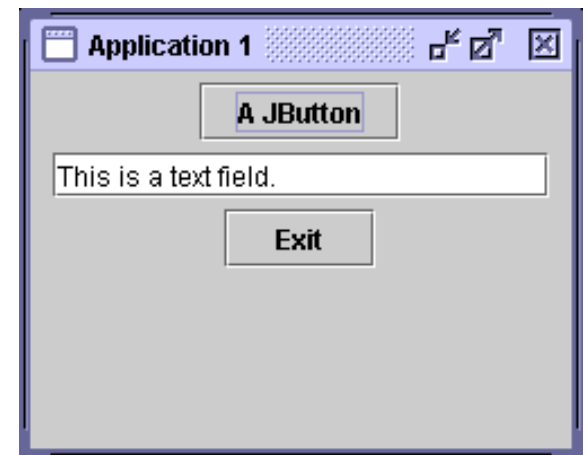
```
JFrame.setDefaultLookAndFeelDecorated(true) ;
```



```
frame.setSize(450, 200);
```



```
frame.setSize(250, 200);
```



Run the application

- เมื่อกดปุ่ม หรือพิมพ์ข้อความลงใน text field แล้วกด enter, จะมีเหตุการณ์ (event) เกิดขึ้น
- ไม่มีอะไรเกิดขึ้น ทำไม?
- ต้องสร้าง listener object เพื่อรอรับ event และจัดการเกี่ยวกับ event นั้น

Example of actions

Act that results in the event	Listener Type
User clicks a button, presses Return while typing in a text field, or chooses a menu item	ActionListener
User closes a frame (main window)	WindowListener
User presses a mouse button while the cursor is over a component	MouseListener
User moves the mouse over a component	MouseMotionListener

How to implement an event handler ?

1. Define a new class that either implements a listener interface or extends a class that implements a listener interface (adapter class)

```
public class MyListener implements ActionListener {  
    .  
    .  
    .  
}
```

How to implement an event handler ?

2. Register an instance of the event handler class as a listener upon one or more components

```
someComponent.addActionListener(anInstanceOfMyListener);
```

How to implement an event handler ?

3. Implement all methods in the listener interface.

```
public void actionPerformed(ActionEvent e) {  
    // code that reacts to the action  
    . . .  
}
```

Example

```
// Program App3 with event handling

public class App3 {

    public static JTextField text;

    public static void main(String[] args) {
        . . .
        // 6.1. Register an event handler
        exitButton.addActionListener(
            new MyExitButtonListener());
        button1.addActionListener(new MyButtonListener());
        . . .
    }
}
```

```
// New classes that implement ActionListener
```

```
class MyExitButtonListener implements ActionListener {  
    public void actionPerformed(ActionEvent e) {  
        System.exit(0);  
    }  
}
```

```
class MyButtonListener implements ActionListener {  
    static int count = 0;  
    public void actionPerformed(ActionEvent e) {  
        App2.text.setText("A JButton has been pressed " +  
            ++count + " times.");  
    }  
}
```



click
2
times



Using inner class

- A class defined inside any class.
- Advantages:
 - Be able to access instance variables from the enclosing class.
 - Keep the event-handling code close to where event occurs.
 - Your event-handling class can inherit from other class.
- Disadvantages:
 - lengthy class
 - longer loading time
 - increase memory requirements


```
// Example
// Program App4 with event handling using inner class

public class App4 {
    public static void main(String[] args) {
        . . .
        class MyExitButtonListener implements ActionListener {
            public void actionPerformed(ActionEvent e) {
                System.exit(0);
            }

            class MyButtonListener implements ActionListener {
                // code for event handling
            }

            // 6.1. Register an event handler
            exitButton.addActionListener(
                new MyExitButtonListener());
            button1.addActionListener(new MyButtonListener());
            . . .
        }
    }
}
```

```
// Example
// Program App5 with event handling using
//   anonymous inner class

public class App5 {
    public static void main(String[] args) {
        . . .

        // 6.1. Register an event handler
        exitButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                System.exit(0);
            }
        });
        button1.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                // code for event handling
            }
        });
        . . .
    }
}
```

Writing Event Listeners

- The event-listener methods should execute quickly. Because all event-handling and drawing methods are executed in the same thread.
- If the action that need to be performed will take a long time, initialize as required and perform the rest in a new thread (will cover later).

Getting Event Info.: Event Objects

- EventObject – the root class of all event state objects
- Useful methods:

getSource

public Object **getSource()**

Returns:

The object on which the Event initially occurred.

toString

public String **toString()**

Overrides:

toString in class Object

Returns:

A a String representation of this EventObject.

Listeners supported by Swing

- Component listener – changes in the component's size, position, or visibility.
- Focus listener – whether the component gained or lost the ability to receive keyboard input.
- Key listener – keypresses; key events are fired only by the component that has the current keyboard focus.
- Mouse events – mouse clicks and movement into or out of the component's drawing area.
- Mouse-motion events – changes in the cursor's position over the component

Common Event-Handling Problem

- A component does not generate the events it should.
 - Did you register the right kind of listener to detect the events?
 - Did you register the listener to the right object?
 - Did you implement the event handler correctly?

Exercise

1. Create a calculator application that has buttons for 0 – 9, +, –, and = signs. It should have a display area that shows the result.
2. Modify the program from (1) to have more functions such as, *, /, %, or handle real number.

