



GRAPHICAL USER INTERFACE

CSI 203: OBJECT ORIENTED PROGRAMMING

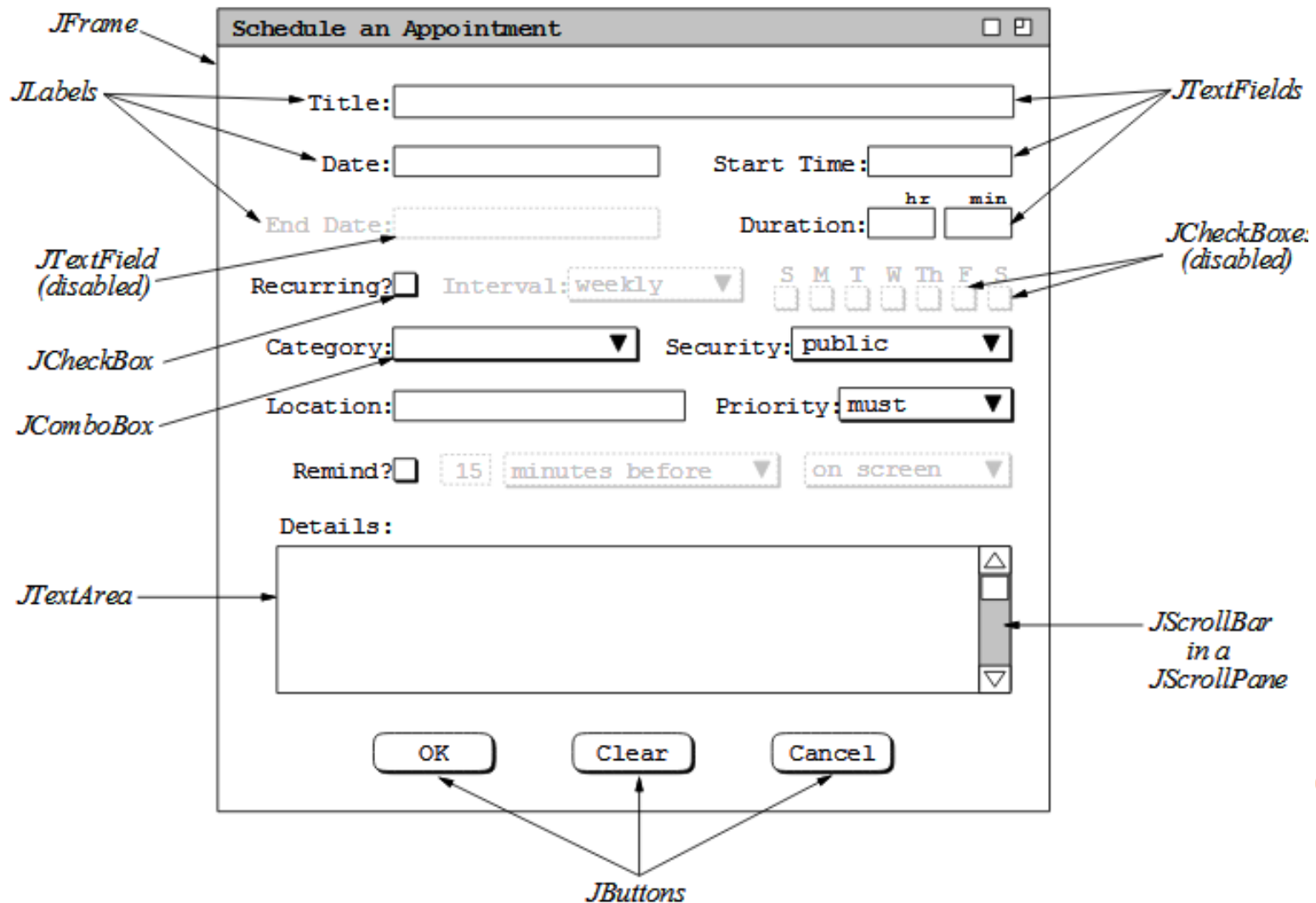
Tanjina Helaly

GRAPHICAL USE INTERFACE

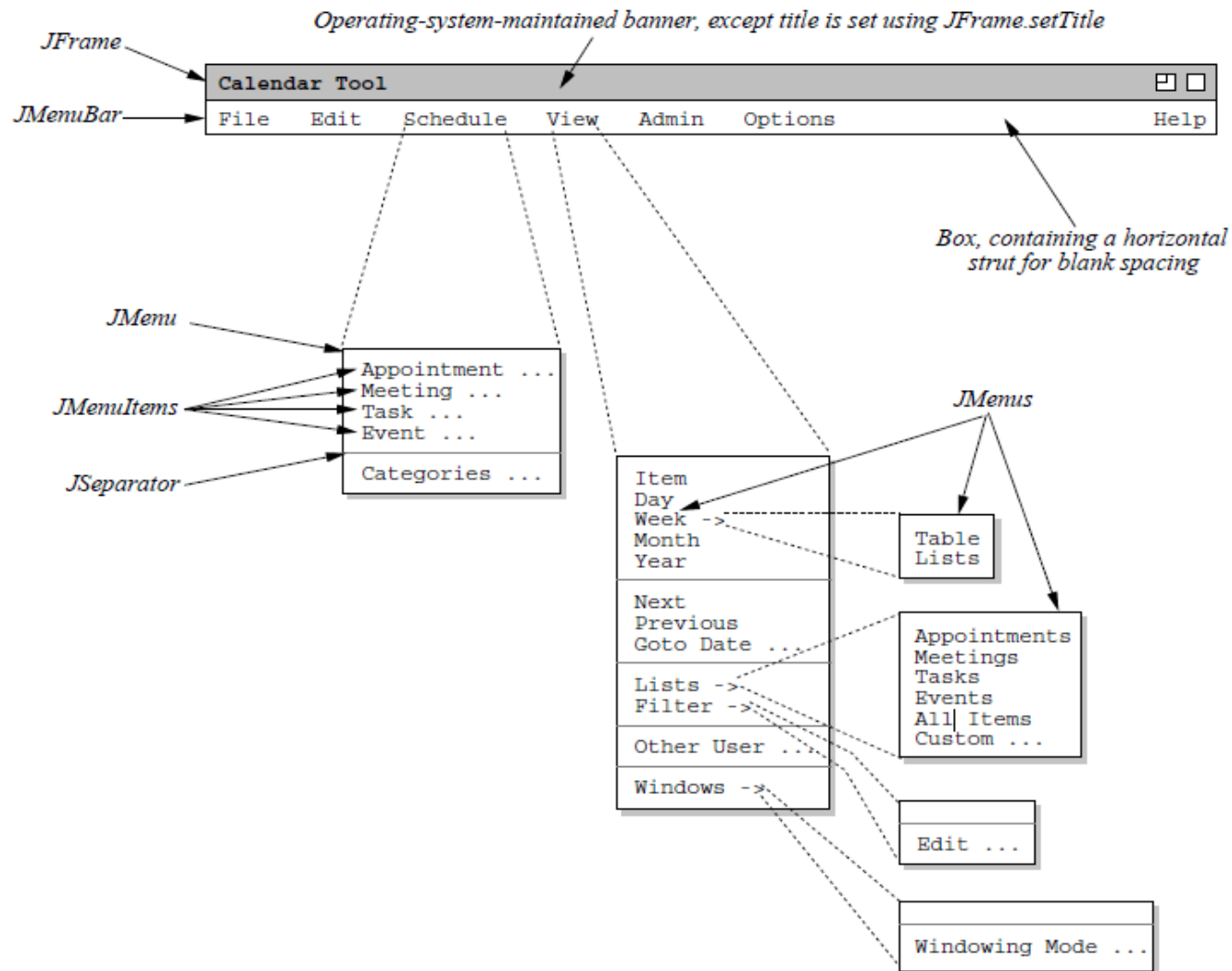
- Gives program distinctive “look” and “feel”
- Provides users with basic level of familiarity
- Built from GUI components (controls, widgets, etc.)
 - User interacts with GUI component via mouse, keyboard, etc.



GRAPHICAL USE INTERFACE

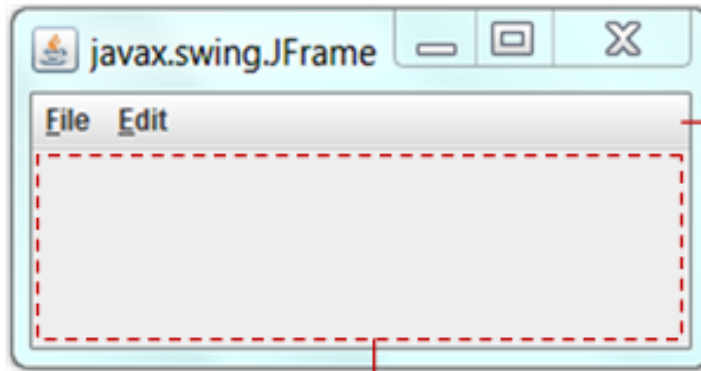


GRAPHICAL USE INTERFACE



TOP-LEVEL AND SECONDARY CONTAINERS

javax.swing.JFrame



Menu Bar
(Optional)

Content Pane

```
Container cp = aJFrame.getContentPane();  
aJFrame.setContentPane(aPanel);
```



THREE PARTS OF A GUI APPLICATION

- Components that make up the Graphical User Interface
- Listeners that receive the events and respond to them
- Application code that does useful work for the user

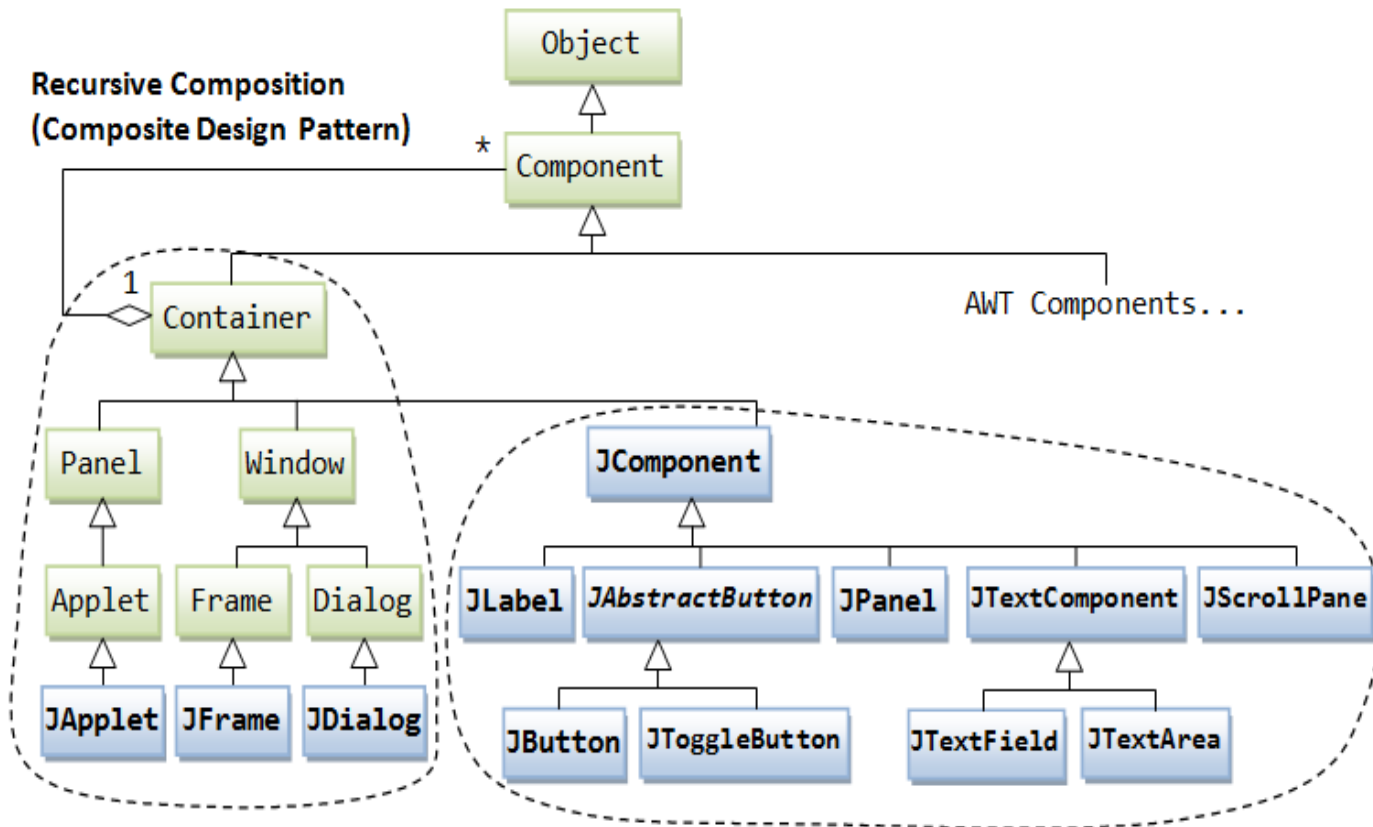


AWT & SWING

- AWT: Abstract Windowing Toolkit
 - Sun's initial effort to create a set of cross-platform GUI classes. (*JDK 1.0 - 1.1*)
 - Maps general Java code to each operating system's real GUI system.
 - Does not provide consistent, cross-platform look-and-feel
 - `import java.awt.*`
- Swing: new with Java2
 - A newer GUI library written from the ground up that allows much more powerful graphics and GUI construction. (*JDK 1.2+*)
 - Paints GUI controls itself pixel-by-pixel rather than handing off to OS.
 - *Benefits:* light weight, new features; compatibility; same look & feel across different platform.
 - `import javax.swing.*`

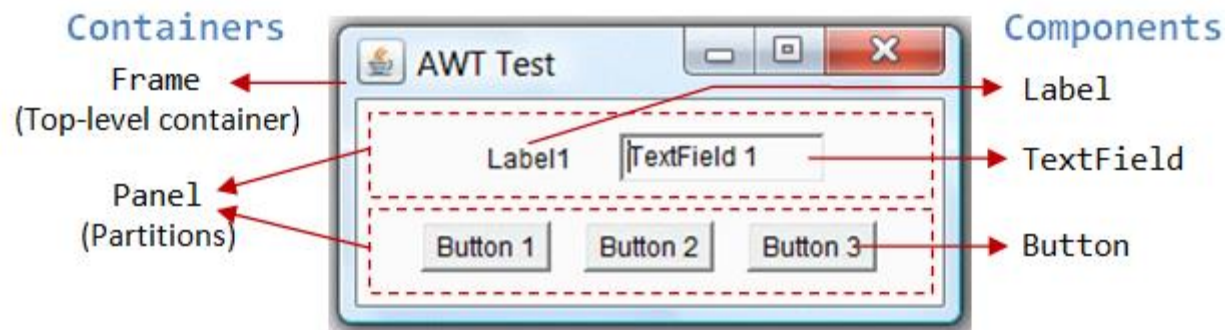


GUI HIERARCHY



CONTAINERS

- There are three basic *top-level* containers
 - **JWindow**: top-level window with no border
 - **JFrame**: top-level window with border and (optional) menu bar
 - **JDialog**: used for dialog windows
- Another important container
 - **JPanel**: used mostly to organize objects within other containers



GUI COMPONENT API

- Each component is a class. So, each component has
 - Properties/attributes
 - Methods
 - Events



STEPS TO CREATE GUI—BASIC WORKFLOW

- Create the top level container/window to hold the entire GUI.
- Add components to it or other lower level container.
 - Add individual component. Or
 - Group components into a lower level container and add that container to top window.
 - Organize these with
 - A layout manager, and
 - JPanel to group components
- Add event handling code.



STEPS TO CREATE GUI – ADDING COMPONENTS

1. Create it

- Instantiate object: `b = new JButton("Click here");`

2. Configure it

- Properties: `b.text = "Click here";` [avoided in java]
- Methods: `b.setText("Click here");`

3. Add it

- `panel.add(b);`

4. Listen to it

- Events: Listeners



GUI EXAMPLE

```
import javax.swing.*;
```

```
public class Hello {
```

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

```
        JFrame f = new JFrame("My Frame"); // Creating the top level  
        container
```

```
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
        f.setSize(200, 100);
```

```
        JPanel p = new JPanel();
```

```
        JButton b = new JButton("Click here"); //Create a JButton object
```

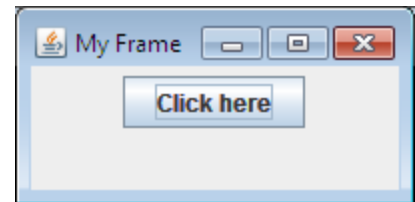
```
        p.add(b); // add button to panel
```

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

```
        f.setVisible(true);
```

```
    }
```

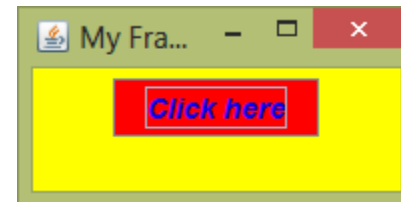
```
}
```



GUI EXAMPLE

```
import javax.swing.*;
```

```
public class Hello {  
    public static void main(String[] args) {  
        JFrame f = new JFrame("My Frame"); // Creating the top level container  
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        f.setSize(200, 100);  
        JPanel p = new JPanel();  
        p.setBackground(Color.cyan); // Setting the background color of the panel  
        JButton b = new JButton("Click here"); // Create a JButton object  
        b.setBackground(Color.red); // Set Button color  
        b.setForeground(Color.BLUE); // Setting the text color of the button  
        b.setFont(new Font("SansSerif", Font.BOLD + Font.ITALIC, 14));  
  
        p.add(b); // add button to panel  
        f.setContentPane(p); // add panel to frame  
  
        f.setVisible(true);  
    }  
}
```



GUI EXAMPLE

```
import javax.swing.*;
```

```
public class Hello {
```

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

```
        JFrame f = new JFrame("My Frame"); // Creating the top level  
        container
```

```
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
        f.setSize(200, 100);
```

```
        Container p = f.getContentPane();
```

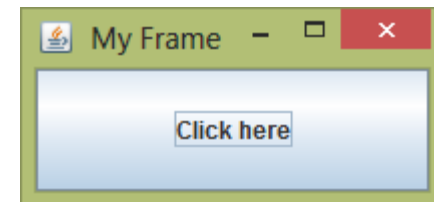
```
        JButton b = new JButton("Click here"); //Create a JButton  
        object
```

```
        p.add(b); // add button to panel
```

```
        f.setVisible(true);
```

```
    }
```

```
}
```



STEPS TO CREATE GUI

- Create:
 - Frame
 - Panel
 - Components
 - Listeners
- Add: (bottom up)
 - listeners into components
 - components into panel
 - panel into frame



JFRAME

a graphical window to hold other components

- `public JFrame()`
`public JFrame(String title)`
Creates a frame with an optional title.
 - Call `setVisible(true)` to make a frame appear on the screen after creating it.
- `public void add(Component comp)`
Places the given component or container inside the frame.



JFRAME

- `public void setDefaultCloseOperation(int op)`

Makes the frame perform the given action when it closes.

- Common value passed: `JFrame.EXIT_ON_CLOSE`
- If not set, the program will never exit even if the frame is closed.

- `public void setSize(int width, int height)`

Gives the frame a fixed size in pixels.

- `public void pack()`

Resizes the frame to fit the components inside it snugly.



SOME COMPONENTS

- ButtonGroup -- for grouping buttons, particular radio buttons.
- JButton -- a standard command button; used all over the place.
- JCheckBox -- a typical on/off checkbox.
- JCheckBoxMenuItem -- a menu item with a checkbox next to it.
- JColorChooser -- a standard-looking color selection dialog.
- JComboBox -- a pulldown that allows typing too.
- JComponent -- the top-level of the Swing component hierarchy.
- JDialog -- a handy pop-up dialog.
- JEditorPane -- a way to view and edit text, in particular HTML.
- JFileChooser -- a standard-looking file chooser.
- JFrame -- the outermost container for a GUI window.
- JLabel -- a simple piece of text within a GUI.
- JList -- a list of GUI components, typically with a scroll bar.
- JMenu -- a pulldown or pop-up menu
- JMenuBar -- a standard menubar, typically at the top of a JFrame.
- JMenuItem -- an item in a JMenu.



SOME COMPONENTS

- JOptionPane -- a parent class for a set of standard option dialogs.
- JPanel -- Typically the inner-wrapper of a JFrame, for managing GUI layout.
- JPasswordField -- a way to enter passwords without echoing.
- JProgressBar -- a typical-looking "throbber"
- JRadioButton -- a typical-looking radio button
- JScrollBar -- horizontal or vertical scrollbar, typically in a JScrollPane.
- JSeparator -- spacing in a menu.
- JSlider -- typical-looking slider, typically for numeric input.
- JTabbedPane -- tabbing pane for organizing things like preferences.
- JTable -- a two-dimensional table, with many display options.
- JTextArea -- a simple, unformatted multi-line text area.
- JTextField -- a single-line text field.
- JToggleButton -- an on/off button.
- JToolBar -- a container for buttons that select other tools.
- JToolTip -- roll-over help for tool buttons or menu items.
- JTree -- a hierarchical tree display, in a Windows Explorer style.



EXAMPLE - WITH MENU

```
import javax.swing.*;
class MenuFrame extends JFrame {
String msg = "";
JCheckBoxMenuItem debug, test;
MenuFrame(String title) {
    super(title);
    setSize(300, 200);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    // create menu bar and add it to frame
    JMenuBar mbar = new JMenuBar();
    setJMenuBar(mbar);

    // create the menu items add to the menubar
    JMenu file = new JMenu("File");
    JMenuItem item1 = new JMenuItem("New...");
    JMenuItem item2 = new JMenuItem("Open...");
    JMenuItem item3 = new JMenuItem("Close");
    JMenuItem item5 = new JMenuItem("Quit...");
    file.add(item1);
    file.add(item2);
    file.add(item3);
    file.addSeparator();
    file.add(item5);
    mbar.add(file);
```



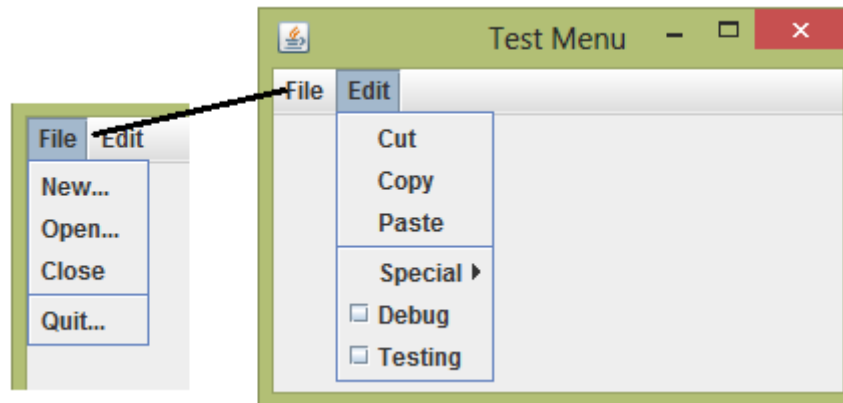
EXAMPLE - WITH MENU

```
// Adding another menu item
JMenu edit = new JMenu("Edit");
JMenuItem item6 = new JMenuItem("Cut");
JMenuItem item7 = new JMenuItem("Copy");
JMenuItem item8 = new JMenuItem("Paste");
edit.add(item6);
edit.add(item7);
edit.add(item8);
edit.addSeparator();
JMenu sub = new JMenu("Special");
JMenuItem item10 = new JMenuItem("First");
JMenuItem item11 = new JMenuItem("Second");
JMenuItem item12 = new JMenuItem("Third");
sub.add(item10);
sub.add(item11);
sub.add(item12);
edit.add(sub);
// these are checkable menu items
debug = new JCheckBoxMenuItem("Debug");
edit.add(debug);
test = new JCheckBoxMenuItem("Testing");
edit.add(test);
mbar.add(edit);
setVisible(true);
}
}
```



EXAMPLE - WITH MENU

```
public class MenuDemo {  
    public static void main(String[] args){  
        new MenuFrame("Test Menu");  
    }  
}
```



EXAMPLE – WITH SCROLLBAR

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

public class GUITest extends JFrame{
    public GUITest(){
        super();
        setTitle("Create Account");
        setSize(300,300);
        setLayout(null);

        JLabel nm = new JLabel("Name");
        JTextField nmtf = new JTextField(20);
        JLabel gen = new JLabel("Gender");
        JCheckBox ml = new JCheckBox("Male");
        JCheckBox fml = new JCheckBox("Female");
        JLabel note = new JLabel("Note");
        JTextArea nt = new JTextArea(5,10);
        JButton submit = new JButton("Create");
        JScrollPane jsp = new JScrollPane(nt,
            ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS,
            ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED);
```



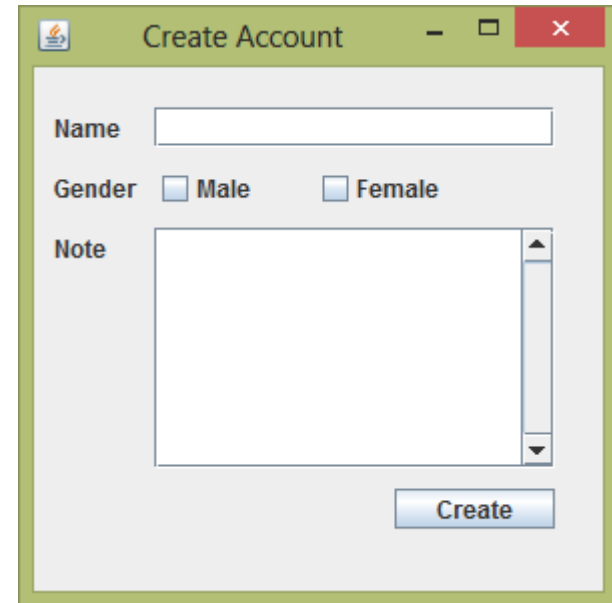
EXAMPLE – WITH SCROLLBAR

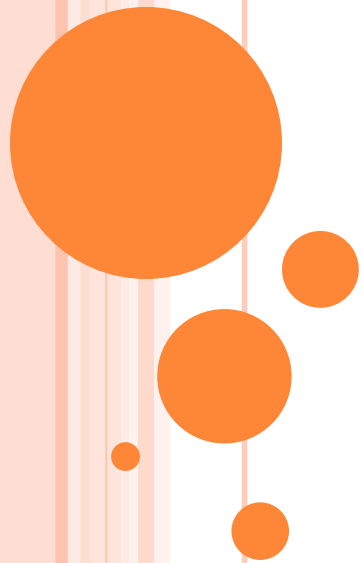
```
nm.setBounds(10, 20, 50, 20);  
add(nm);  
nmtf.setBounds(60, 20, 200, 20);  
add(nmtf);  
gen.setBounds(10, 50, 50, 20);  
add(gen);  
ml.setBounds(60, 50, 80, 20);  
add(ml);  
fml.setBounds(140, 50, 100, 20);  
add(fml);  
note.setBounds(10, 80, 40, 20);  
add(note);  
jsp.setBounds(60, 80, 200, 120);  
add(jsp);  
submit.setBounds(180, 210, 80, 20);  
add(submit);  
setVisible(true);
```

```
}
```

```
public static void main(String[] args) {  
    new GUITest();  
}
```

```
}
```





LAYOUT

WHAT IS LAYOUT

- The LayoutManagers are used to arrange components in a particular manner.
- AWT provides the following layout managers (in package `java.awt`):
 - FlowLayout,
 - GridLayout,
 - BorderLayout,
 - GridBagLayout,
 - BoxLayout,
 - CardLayout,
 - and others.



WHAT IS LAYOUT

- Each **Container** object has a layout manager associated with it.
- A layout manager is an instance of any class that implements the **LayoutManager** interface.
- The layout manager is set by the **setLayout()** method.
- If you do not specify a layout manager, the container will use a default:
 - **JPanel** default = **FlowLayout**
 - **JFrame** default = **BorderLayout**



FLOW LAYOUT

- In the `java.awt.FlowLayout`, components are arranged
 - from left-to-right inside the container in the order that they are added
 - When one row is filled, a new row will be started.
- Constructors:
 - `FlowLayout()`
 - creates the default layout, which centers components and leaves five pixels of space between each component.
 - `FlowLayout(int how)`
 - specify how each line is aligned.
 - Valid values for *how* are as follows:
 - `FlowLayout.LEFT`
 - `FlowLayout.CENTER`
 - `FlowLayout.RIGHT`
 - `FlowLayout.LEADING`
 - `FlowLayout.TRAILING`

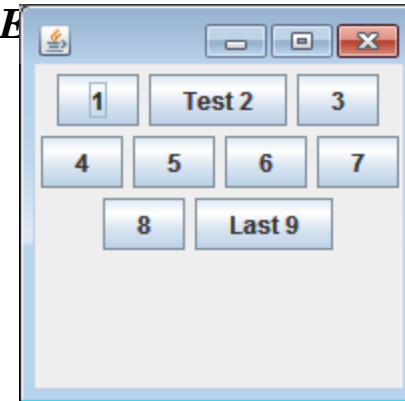


FLOW LAYOUT

```
public class MyLayout{
    JFrame f;
    MyLayout(){
        f=new JFrame();
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setLayout(new FlowLayout());
        f.setSize(200,200);

        JButton b1=new JButton("1");
        JButton b2=new JButton("Test 2");
        JButton b3=new JButton("3");
        JButton b4=new JButton("4");
        JButton b5=new JButton("5");
        JButton b6=new JButton("6");
        JButton b7=new JButton("7");
        JButton b8=new JButton("8");
        JButton b9=new JButton("Last 9");

        f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);
        f.add(b6);f.add(b7);f.add(b8);f.add(b9);
        f.setVisible(true);
    }
    public static void main(String[] args) {
        new MyLayout();
    }
}
```

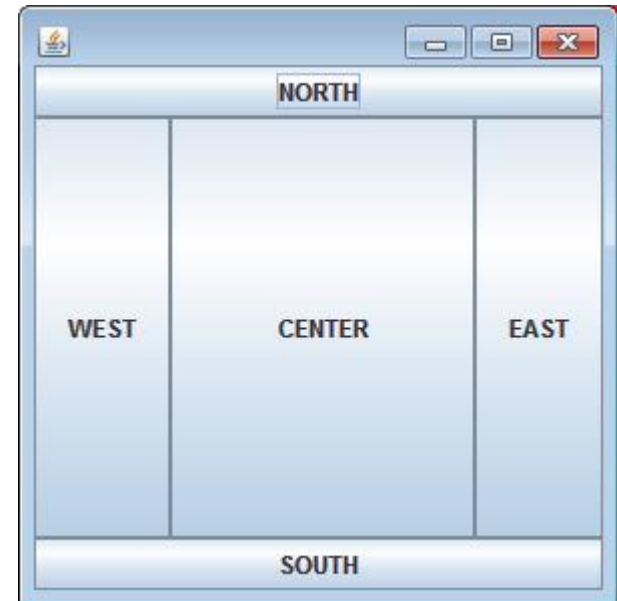


BORDER LAYOUT

- Default layout for a frame and divides area into named regions:

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

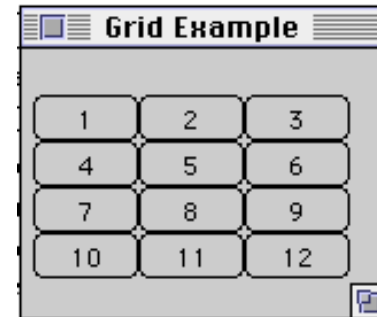
```
public class Border {  
    JFrame f;  
    Border(){  
        f=new JFrame();  
  
        JButton b1=new JButton("NORTH");  
        JButton b2=new JButton("SOUTH");  
        JButton b3=new JButton("EAST");  
        JButton b4=new JButton("WEST");  
        JButton b5=new JButton("CENTER");  
  
        f.add(b1,BorderLayout.NORTH);  
        f.add(b2,BorderLayout.SOUTH);  
        f.add(b3,BorderLayout.EAST);  
        f.add(b4,BorderLayout.WEST);  
        f.add(b5,BorderLayout.CENTER);  
  
        f.setSize(300,300);  
        f.setVisible(true);  
    }  
  
    public static void main(String[] args) {  
        new Border();  
    }  
}
```



GRID LAYOUT

- In GridLayout, components are arranged in a grid (matrix) of rows and columns inside the Container.
- Components are added in a left-to-right, top-to-bottom manner in the order they are added.

```
import java.awt.*;
class GridLayoutExample extends Frame {
    public GridLayoutExample( int widthInPixels, int heightInPixels ) {
        setTitle( "Grid Example" );
        setSize( widthInPixels, heightInPixels );
        int numberOfRows = 4;
        int numberOfColumns = 3;
        setLayout( new GridLayout( numberOfRows,
                                   numberOfColumns ) );
        for ( int label = 1; label < 13; label++ )
            add( new Button( String.valueOf( label ) ) );
        show();
    }
    public static void main( String args[] ) {
        new GridLayoutExample( 175, 100 );
    }
}
```

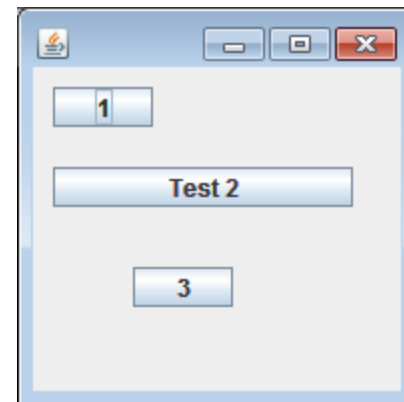


No LAYOUT

- When want to add component to a specific location.
- Set the Layout to null

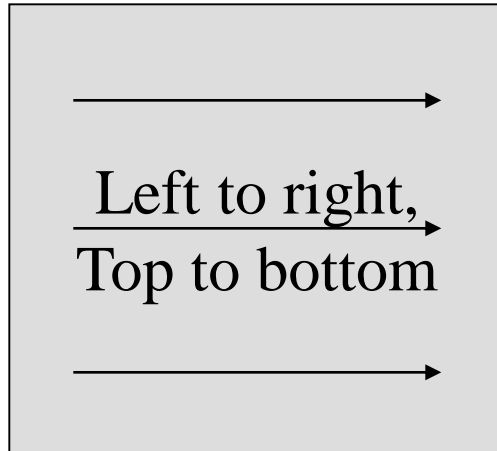
```
public class MyLayout{
    JFrame f;
    MyLayout(){
        f=new JFrame();
        f.setSize(200,200);
        f.setLayout(null);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JButton b1=new JButton("1");
        JButton b2=new JButton("Test 2");
        JButton b3=new JButton("3");
        b1.setBounds(10, 10, 50, 20);
        b2.setBounds(10, 50, 150, 20);
        b3.setBounds(50, 100, 50, 20);
        f.add(b1);f.add(b2);f.add(b3);
        f.setVisible(true);
    }
    public static void main(String[] args) {
        new MyLayout();
    }
}
```

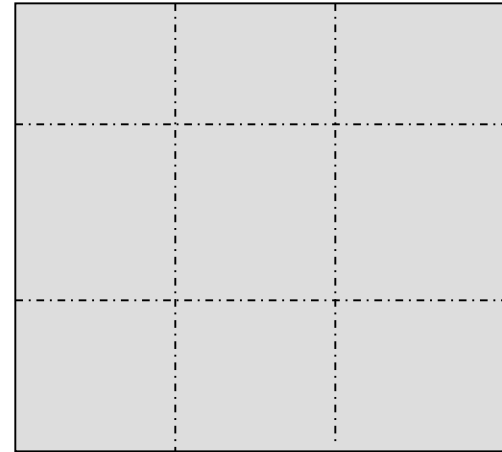


LAYOUTS - SUMMARY

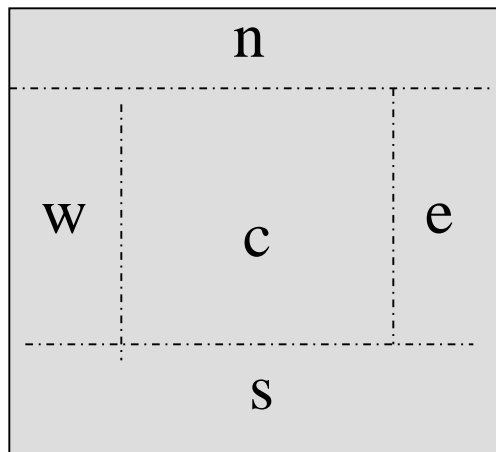
FlowLayout



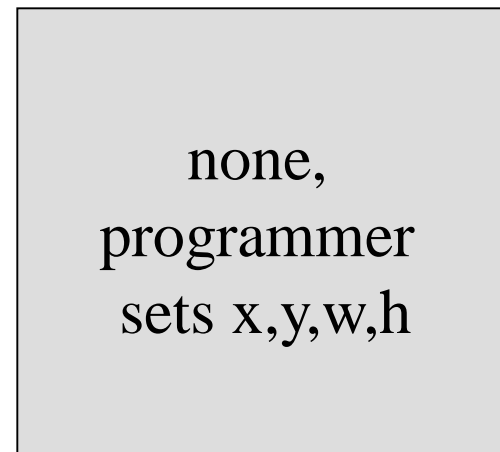
GridLayout



BorderLayout



null





EVENT HANDLING

AWT EVENT-HANDLING

- Event handling is a basic concept of graphical user interfaces.
- In event-driven programming,
 - a piece of event-handling codes is executed (or called back by the graphics subsystem)
 - when an event has been fired
 - in response to an user input
 - Example:
 - mouse clicks
 - keyboard entries
 - time intervals
 - Button click
- This is unlike the procedural model, where codes are executed in a sequential manner.



MAIN COMPONENTS

- Three objects are involved in the event-handling:
 - a *source* –
 - *GUI component with which user interact.*
 - generates an event and sends it to one or more listeners.
 - *listener(s)* -
 - simply waits until it receives an event.
 - Receives event object when notified, then responds(processes the event and then returns)
 - an *event* object. –
 - Encapsulates information about event that occurred
 - Describe the state change of a source



PROGRAMMER'S RESPONSIBILITY

- Programmer must perform two tasks
 - Register event listener for event source
 - Implement event-handling method (event handler)



EVENTS

- In the delegation model, an event is an object that describes a **state change** in a source.

○
an event can be **generated as a consequence of a person interacting** with the elements in a graphical user interface

- Example:
 - pressing a button,
 - entering a character via the keyboard,
 - selecting an item in a list, and
 - clicking the mouse.

○
Events may also occur that are not directly caused by interactions with a user interface. For example, an event may be generated when

- a timer expires,
- a counter exceeds a value,
- a software or hardware failure occurs,
- or an operation is completed.



EVENT SOURCE

- A source is an object that generates an event.
- This occurs when the internal state of that object changes in some way.
- Sources may generate more than one type of event.
 - A source must register listeners in order for the listeners to receive notifications about a specific type of event.
 - Each type of event has its own registration method.
 - Here is the general form:

```
el )    public void addTypeListener (TypeListener
```



EVENT LISTENERS

- A listener is an object that is notified when an event occurs. It has two major requirements.
 - First, it **must have been registered** with one or more **sources** to receive notifications about specific types of events.
 - For a program to respond to an event there must be an event listener object in the GUI program that listens to that type of event
 - Second, it **must implement methods** to receive and process these notifications.
 - When an event is generated by the GUI component this method in the listener object is invoked to respond to the event
- The methods that receive and process events are defined in a set of interfaces.

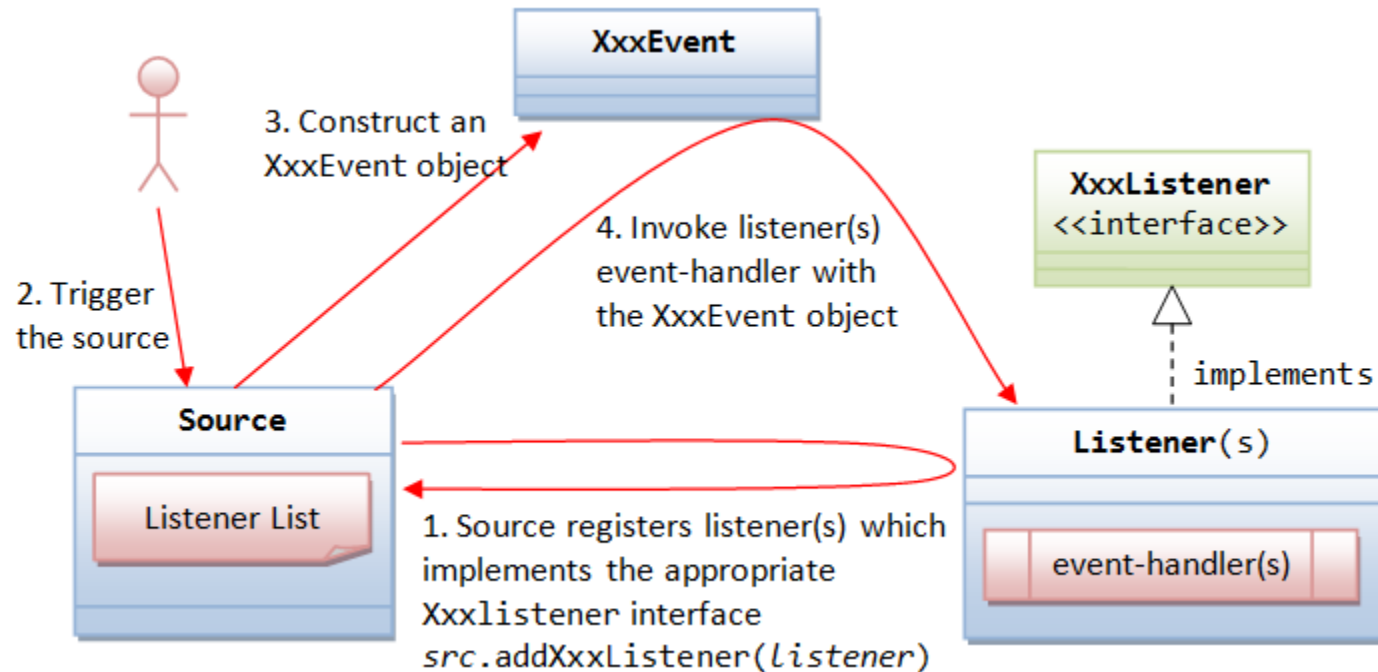


WHAT IF ...?

- When there is no event listener for an event
 - A program can ignore events
 - If there is no listener for an event, the event is just ignored



THE SEQUENCE OF STEPS IN EVENT HANDLING



IMPLEMENT LISTENER IN INNER CLASS

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class Counter{
    JFrame f;
    JTextField tf;
    public Counter(){
        f = new JFrame("Counter");
        f.setLayout(new FlowLayout());
        f.setSize(200, 100);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.add(new JLabel("Counter"));
        tf = new JTextField(10);
        f.add(tf);
        tf.setText("0");
        JButton b = new JButton("Count");
        f.add(b);

        b.addActionListener(new CounterAction());

        f.setVisible(true);
    }
    public static void main(String[] args) {
        new Counter();
    }
}
```

Register the listener to the source

```
public class CounterAction implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        tf.setText(Integer.parseInt(tf.getText()) + 1 + "");
    }
}
```

Implement the Listener Inner Class



IMPLEMENT LISTENER IN OWN CLASS

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class Counter implements ActionListener{
    JFrame f;
    JTextField tf;
    public Counter(){
        f = new JFrame("Counter");
        f.setLayout(new FlowLayout());
        f.setSize(200, 100);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.add(new JLabel("Counter"));
        tf = new JTextField(10);
        f.add(tf);
        tf.setText("0");
        JButton b = new JButton("Count");
        f.add(b);

        b.addActionListener(this);

        f.setVisible(true);
    }


    public void actionPerformed(ActionEvent e) {
        tf.setText(Integer.parseInt(tf.getText()) + 1 + "");
    }

    public static void main(String[] args) {
        new Counter();
    }
}
```

Implement the listener

Register the listener to the source

Implement the method to handle the event



IMPLEMENT LISTENER IN ANONYMOUS CLASS

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class Counter{
    JFrame f;
    JTextField tf;
    public Counter(){
        f = new JFrame("Counter");
        f.setLayout(new FlowLayout());
        f.setSize(200, 100);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.add(new JLabel("Counter"));
        tf = new JTextField(10);
        tf.setText("0");
        JButton b = new JButton("Count");
        f.add(tf); f.add(b);
```

```
        b.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e) {
                tf.setText(Integer.parseInt(tf.getText()) + 1 + "");
            }
        });
```

Register the anonymous listener object to the source

```
        f.setVisible(true);
    }
    public static void main(String[] args) {
        new Counter();
    }
}
```



MULTIPLE BUTTONS

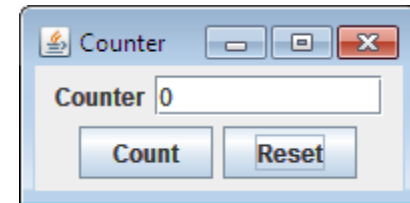
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Counter extends JFrame implements ActionListener{
    JTextField tf;
    JButton b, r;
    public Counter(){
        super("Counter"); setSize(200, 100);
        setLayout(new FlowLayout());
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        add(new JLabel("Counter"));
        tf = new JTextField("0", 10);
        b = new JButton("Count");
        r = new JButton("Reset");
        add(tf); add(b); add(r);
        setVisible(true);

        b.addActionListener(this);
        r.addActionListener(this);
    }

    public void actionPerformed(ActionEvent e) {
        Object obj = e.getSource();
        if (obj == b)
            tf.setText(Integer.parseInt(tf.getText()) + 1 + "");
        else if (obj == r)
            tf.setText("0");
    }

    public static void main(String[] args) { new Counter(); }
}
```

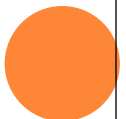


**Identify the source first and
then implement the logic
depending on the source**



LIST OF SOME COMMON SOURCE, EVENT AND LISTENER

Source	Event	Interface	Methods
Button (Click) List (Double clicked) MenuItem (Clicked) Timer	ActionEvent	ActionListener	void actionPerformed(ActionEvent ae)
CheckBox	ItemEvent	ItemListener	void itemStateChanged(ItemEvent ie)
Frame(via Keyboard)	KeyEvent	KeyListener	void keyPressed(KeyEvent ke) void keyReleased(KeyEvent ke) void keyTyped(KeyEvent ke)
Frame(Mouse action)	MouseEvent	MouseListener	void mouseClicked(MouseEvent me) void mouseEntered(MouseEvent me) void mouseExited(MouseEvent me) void mousePressed(MouseEvent me) void mouseReleased(MouseEvent me)
Frame(Mouse Move)	MouseEvent	MouseMotionListener	void mouseDragged(MouseEvent me) void mouseMoved(MouseEvent me)
TextField/TextArea(changing text)	TextEvent	TextListener	void textValueChanged(TextEvent te)
Window/Frame	WindowEvent	WindowListener	void windowActivated(WindowEvent we) void windowClosed(WindowEvent we) void windowClosing(WindowEvent we) void windowDeactivated(WindowEvent we) void windowDeiconified(WindowEvent we) void windowIconified(WindowEvent we) void windowOpened(WindowEvent we)



ADAPTER CLASSES

- Suppose your class directly implements **MouseListener**
 - Then you must implement all five **MouseListener** methods.
 - Even if you care only about mouse clicks
- Methods for those events you don't care about can have empty bodies.
 - Resulting collection of empty method bodies can make code harder to read and maintain
- How to avoid?
 - Use adapter classes
 - An adapter class implements empty versions of all its interface's methods.
 - For example, the **MouseAdapter** class implements the **MouseListener** interface.

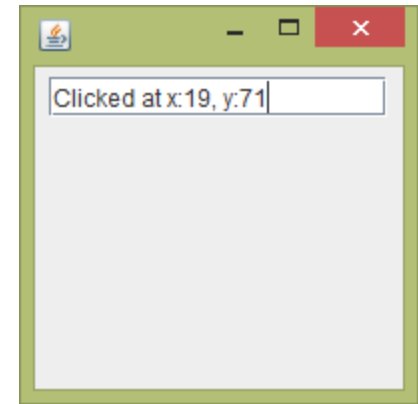


GUI EXAMPLE – MOUSE EVENT WITH ADAPTER

```
import java.awt.FlowLayout;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import javax.swing.JFrame;
import javax.swing.JTextField;

public class TestMouseEvent {
    public static void main(String[] args) {
        JFrame f = new JFrame();
        f.setLayout(new FlowLayout());
        f.setSize(200, 200);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JTextField tf = new JTextField(15);
        f.add(tf);
        f.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                tf.setText("Clicked at x:"+e.getX()+" y:"+e.getY());
            }
        });

        f.setVisible(true);
    }
}
```



GUI EXAMPLE – MOUSE EVENT WITH LISTENER & ANONYMOUS CLASS

```
import java.awt.FlowLayout;
import java.awt.event.*;
import javax.swing.JFrame;
import javax.swing.JTextField;
public class TestMouseListener{
    public static void main(String[] args) {
        JFrame f = new JFrame();
        f.setLayout(new FlowLayout());
        f.setSize(200, 200);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JTextField tf = new JTextField(15);
        f.add(tf);
        f.addMouseListener(new MouseListener() {
            public void mouseReleased(MouseEvent e) {}
            public void mousePressed(MouseEvent e) {}
            public void mouseExited(MouseEvent e) {}
            public void mouseEntered(MouseEvent e) {}
            public void mouseClicked(MouseEvent e) {
                tf.setText("Clicked at x:"+e.getX()+" y:"+e.getY());
            }
        });
        f.setVisible(true);
    }
}
```



GUI EXAMPLE – MOUSE EVENT WITH LISTENER

```
import java.awt.FlowLayout;
import java.awt.event.*;
import javax.swing.JFrame;
import javax.swing.JTextField;
public class TestMouseListener implements MouseListener{
    JTextField tf ;
    public TestMouseListener()    {
        JFrame f = new JFrame();
        f.setLayout(new FlowLayout());
        f.setSize(200, 200);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        tf = new JTextField(15);
        f.add(tf);
        f.addMouseListener(this);
        f.setVisible(true);
    }

    public static void main(String[] args) {
        new TestMouseListener();
    }

    public void mouseClicked(MouseEvent e) {
        tf.setText("Clicked at x:"+e.getX()+" , y:"+e.getY());
    }
    public void mousePressed(MouseEvent e) {}
    public void mouseReleased(MouseEvent e) {}
    public void mouseEntered(MouseEvent e) {}
    public void mouseExited(MouseEvent e) { }
```



REFERENCE

- Java: Complete Reference - Chapter 24-26, 31-33
- Java: How to Program – Chapter 12

