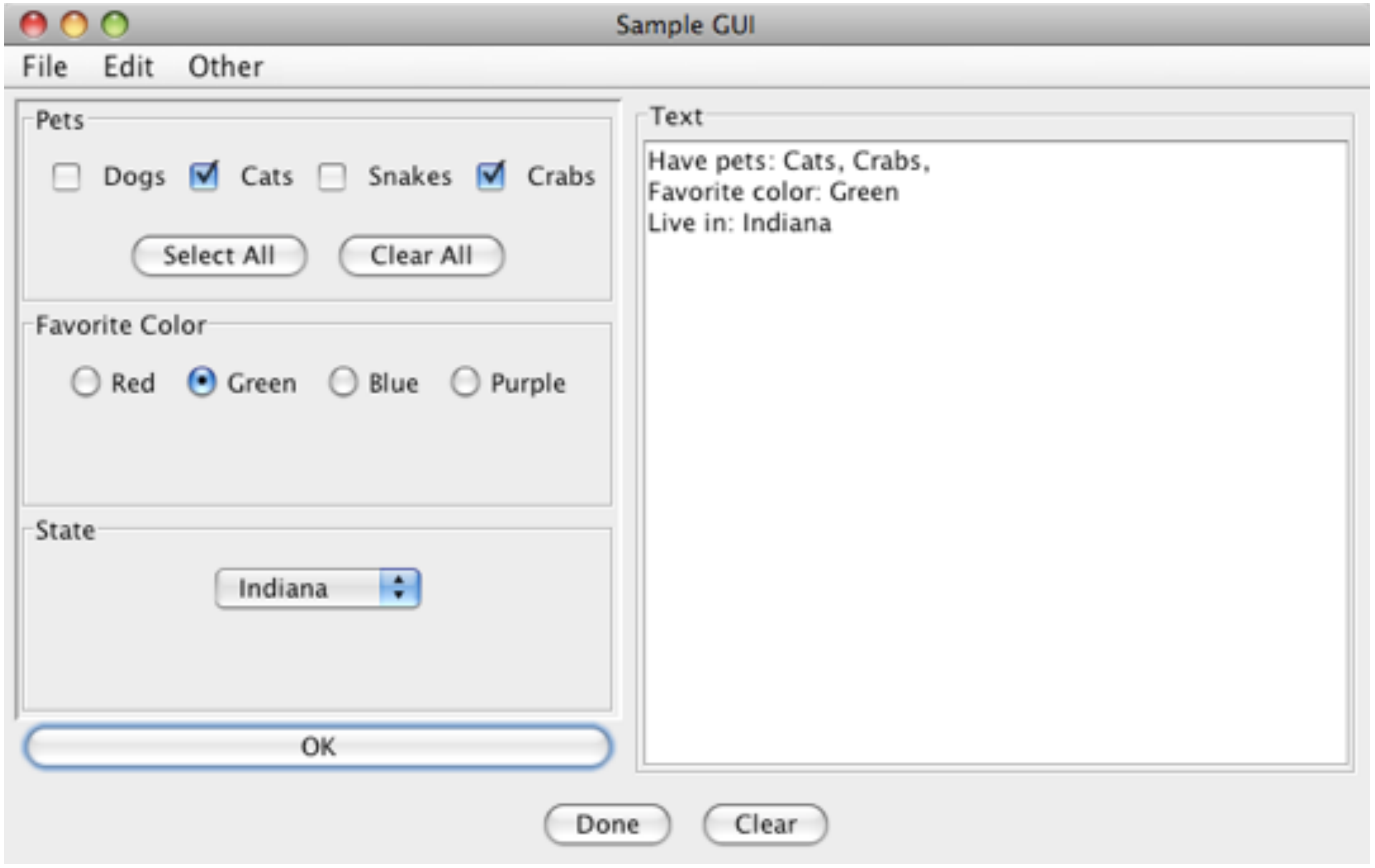
GUI and Event-Driven Programming

CS 18000
Prof. Sunil Prabhakar
Department of Computer Science
Purdue University

Graphical User Interfaces

- Input/output devices for computers
  - Printer, punch cards
  - Keyboard, Screen
  - Graphical interface with mouse input
- You have used GUIs for most of your interactions with the computer.
- GUIs consist of windows, buttons, menus, entry fields, …

Sample GUI



Have pets: Cats, Crabs,
Favorite color: Green
Live in: Indiana

GUI classes

- Java makes it very easy to create GUIs
- The two packages java.awt and javax.swing provide a large number of classes that can be used to construct GUIs
- By using these classes, we need not worry about the differences between operating systems or system details
- We will use classes from the swing package as they are more reliable across platforms
- The awt package provides support for swing classes

Creating a simple GUI

- Create a window object
- Add GUI elements to the window
- Write code to respond to the GUI elements

Creating a Window

- The JFrame class is a common starting point.
  - The JFrame class corresponds to a basic window for the given operating system
  - It behaves like most other windows
- We can either
  - create an object of the JFrame class, or
  - create a subclass of JFrame if we expect to create multiple windows with the same behavior

A simple **JFrame** object

```
class ShowWindow {
    public static void main( String[] args ) {
        JFrame myWindow;

        myWindow = new JFrame();
        myWindow.setSize(300,400);
        myWindow.setTitle("My Window");
        myWindow.setResizable(true);
        myWindow.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        myWindow.setVisible(true);
    }
}
```

A custom **JFrame**

```
import javax.swing.*;

class MyWindow extends JFrame {
    public MyWindow(String title) {
        this.setSize(300,400);
        this.setTitle(title);
        this.setVisible(true);
    }
}
```

```
import javax.swing.*;
class ShowWindow2 {
    public static void main( String[] args ) {
        MyWindow myWindow = new MyWindow("My Window");
        MyWindow window = new MyWindow("Another Window");
    }
}
```

Some GUI classes

- Frame
  - A special container corresponding to a window not contained in another window.
  - JFrame
  - JApplet (for web applets)
- Containers
  - GUI components that hold other GUI components.
  - JFrame, JApplet,
  - JPanel
    - An invisible container that can be nested.

Other GUI classes

- Common elements
  - JButton, JCheckBox, JComboBox,JTextField, JTextArea
  - Graphics
  - Allows drawing of circles, strings, etc.
- Font
  - For selecting fonts for text
- Color
  - For selecting colors of GUI components
- Menu classes
  - JMenuBar, JMenu
- And many more …

Coverage

- There are way too many classes for us to consider each one
- We will see a sampling
- Use the online tutorial from Oracle for more examples, other details
- http://docs.oracle.com/javase/tutorial/ui/features/components.html

Essentials of a GUI

- We begin with a frame (e.g, JFrame, JApplet)
- We will use JFrame as our starting point.
- We can change the properties of the frame by calling several methods for it.
- We cannot add components to the JFrame directly. We have to add them to its Content Pane.
- We can add components from this pane.
  - These can be buttons, text fields, labels, lists, scroll bars, …. , and other panes.

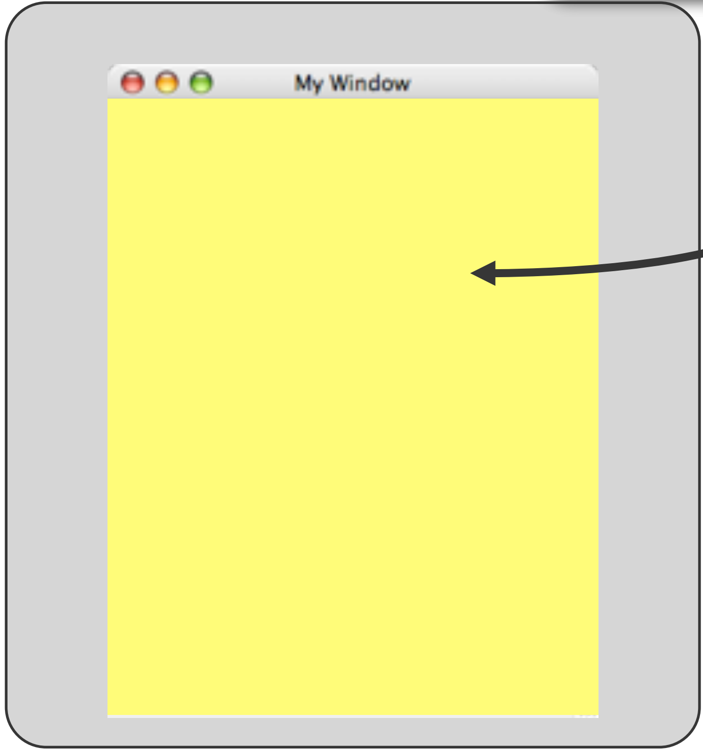
The Content Pane of a Frame

- We access the content pane by calling the frame's getContentPane() method.
- It belongs to the Container class

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

class MyWindow extends JFrame {
    public MyWindow(String title) {
        Container cPane;
        this.setSize(300,400);
        this.setTitle(title);
        this.setVisible(true);
        cPane =
           this.getContentPane();
        cPane.setBackground(
           Color.YELLOW);
    }
}
```

This yellow area is the content pane of this frame.



Adding Components

- We can add objects to a container object by using the add() method on the container
- We can add multiple objects to a single container
- Their placement is controlled by either
  - a layout manager, or
  - absolute positioning (rare)

Layout Managers

- A layout manager organizes the multiple components added to a single container.
- For now, we will use a FlowLayoutManager.
- The flow layout organizes objects similar to how (centered) text is written on a page
- We set the layout manager for a container by using the setLayout() method

Adding Buttons

- A JButton object is a GUI component that represents a pushbutton.

```
JButton loginButton = new JButton("Login");
JButton cancelButton = new JButton("Cancel");
```

CREATE NEW OBJECTS

```
Container contentPane;
contentPane = myFrame.getContentPane();
contentPane.setLayout(new FlowLayout());
```

GET CONTAINER PANEL,
SET LAYOUT MANAGER

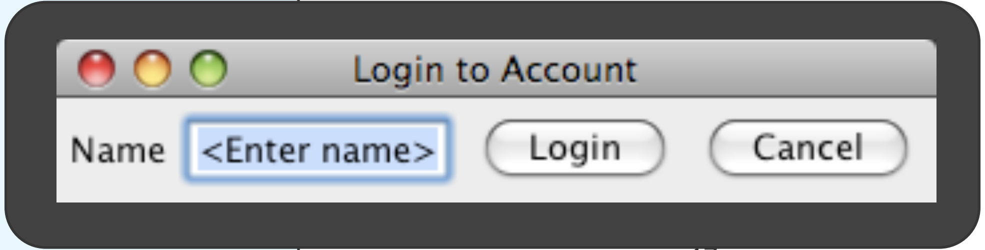
```
contentPane.add(loginButton);
contentPane.add(cancelButton);
```

ADD OBJECTS TO PANEL

Example

```
import javax.swing.*;
import java.awt.*;
class LoginWindow extends JFrame{
    JButton loginButton, cancelButton;
    JTextField nameInput;
    public LoginWindow(String title) {
        this.setTitle(title);
        this.setSize(200,100);
        loginButton = new JButton("Login");
        cancelButton = new JButton("Cancel");
        JLabel label = new JLabel("Name");
        nameInput = new JTextField("<Enter Name>");
        Container contentPane =
this.getContentPane();
        contentPane.setLayout(new FlowLayout());
        contentPane.add(label);
        contentPane.add(nameInput);
        contentPane.add(loginButton);
        contentPane.add(cancelButton);
        this.pack();
        this.setVisible(true);
    }
}
```

```
class OpenAccount {
    public static void main( String[] args ) {
        LoginWindow myWindow = new
            LoginWindow("Login to Account");
    }
}
```


Login to Account

Name <Enter name> Login Cancel

Control flow with GUI

- GUI components introduce a new type of control flow.
- In the earlier example, even though the main method ends, the window (and program) keep running.
- A separate thread is automatically created which handles the GUI components.
  - What code is running?
- The separate thread watches for user interactions with the GUI components
  - How does it know what to do, e.g., when a button is pressed?
  - Event handling

Event Handling

- An action involving a GUI object, such as clicking a button, is called an event.
- The mechanism to process events is called event handling.
- Event handling in Java is implemented by two types of objects:
  - event sources -- objects that create events
  - event listeners -- objects that handle events

Event Sources

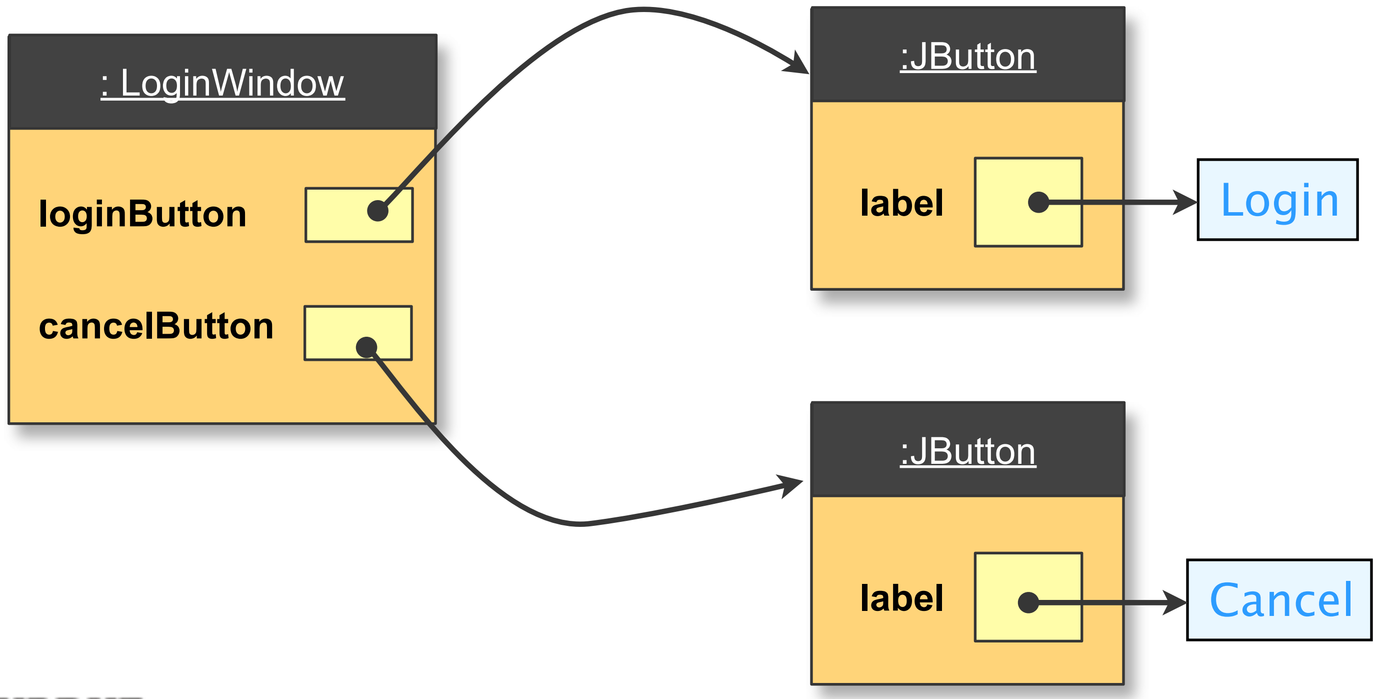
- An event source is a GUI object where an event occurs. We say an event source generates events
  - usually due to an action of the user (e.g., click)
- Buttons, text boxes, list boxes, and menus are common event sources in GUI-based applications.
- Each type of object produces events that are relevant to it.

Event Listener Objects

- An event listener is any object that is registered to respond ("listen") to events generated by some event source.
  - a listener is registered by calling one of the add listener methods on the source
- When an event is generated by the source, a special method is called for each listener
  - in order to be a listener, these methods must be defined

Handling a GUI Event

- A listener object registers with a source object.
- When the source generates an event, a handler method is called on the listener



Handling a GUI Event

- When an event (e.g., a click) takes place on the loginButton object
  - information about this event is sent to all objects that are listening to loginButton
- Who is listening?
  - all objects that registered as listeners
  - by being passed as an argument to a registration method of the login button: `addActionListener();`

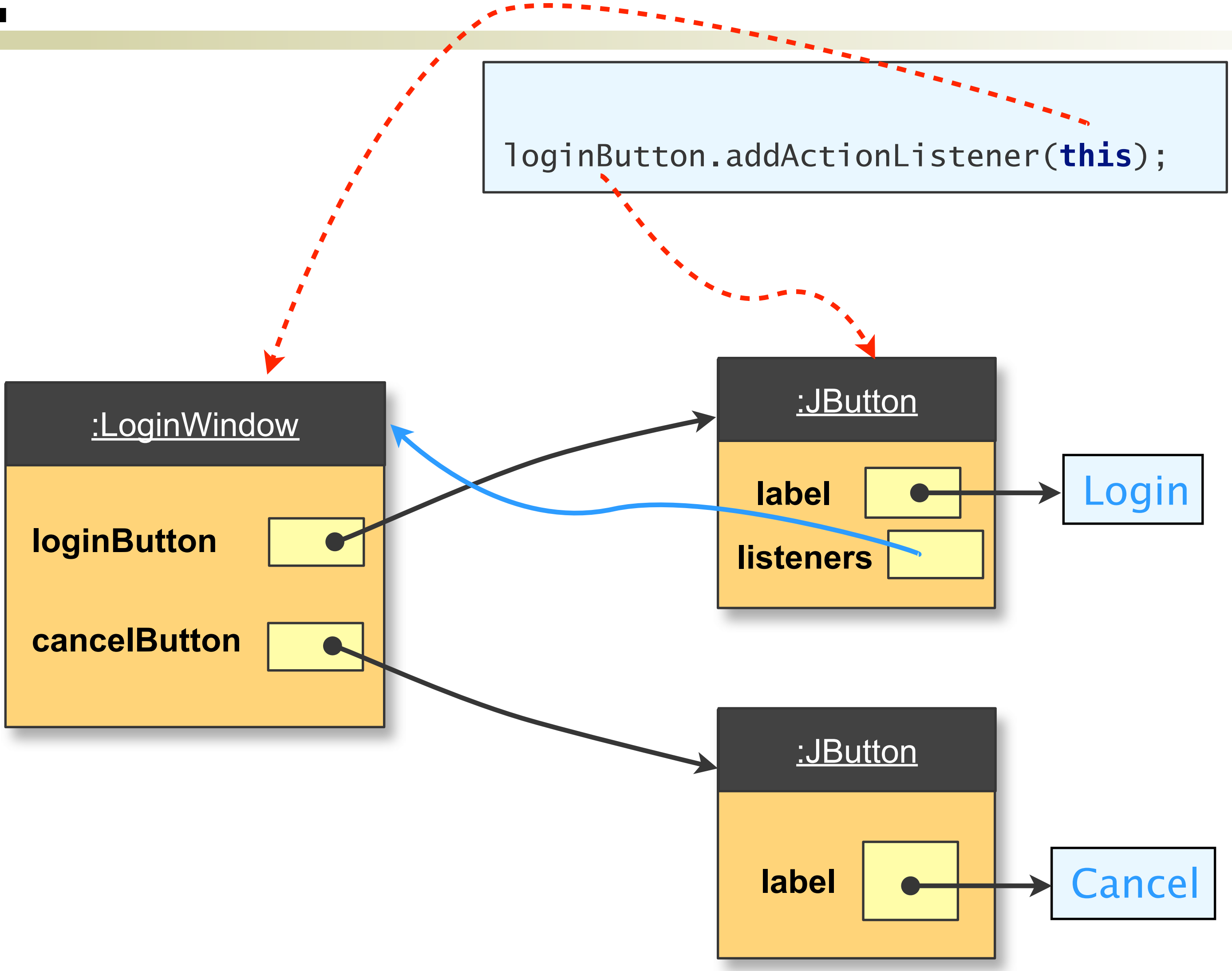
Becoming a Listener

```
import javax.swing.*;
class LoginWindow extends JFrame implements
ActionListener {
    public LoginWindow2(String title) {
. . .
        contentPane.add(loginButton);
        loginButton.addActionListener(this);
. . .
    }
    public void actionPerformed(ActionEvent e) {
        System.out.println("Login button pressed!");
    }
}
```

Add Listener

Registering as a listener

```
loginButton.addActionListener(this);
```



Handling a GUI Event 2

- When an event takes place all listeners will be notified.
- HOW?
  - A special method will be called on each listener: `actionPerformed(ActionEvent)`
  - The argument is an object containing details about the event that took place
  - Thus, *each listener must define this method*

Handling an event

```
import javax.swing.*;
class LoginWindow extends JFrame implements
ActionListener {
    public LoginWindow(String title) {
. . .
        contentPane.add(loginButton);
        loginButton.addActionListener(this);
. . .
    }
    public void actionPerformed(ActionEvent e) {
        System.out.println("Login button pressed!");
    }
}
```

HANDLER METHOD

Being a Listener

- What does it mean to be a listener?
- Being a listener implies that a special method of the listener object will be called when an event occurs.
- Each different event type results in a call to a different method.

  E.g., `actionPerformed(ActionEvent);`
- How do we ensure that the correct type of method has been defined? I.e., how do we enforce the signature of methods in classes we don't even know about?

ActionListener interface

Consider the `addActionListener()` method

- What is the type of its argument?
- Any object could be a listener

```
void addActionListener(??? listener){ }
```

- E.g,. a LoginWindow object or a Student object could be listeners.
- We will call the `actionPerformed(ActionEvent)` method on this listener, so we must ensure that this method exists for the listener object.
- How?

The Java Interface

- An interface is a guarantee of behavior (methods)
  - The interface only specifies the name, return type and arguments for methods. No body.
- An interface
  - is like a class since it is a data type
  - is unlike a class since we can't create objects of this type directly.
- For example: `ActionListener` is an interface.
  - `addActionListener` expects an argument of this type: `void addActionListener(ActionListener l)`
  - The interface requires one method: `void actionPerformed(ActionEvent)`

The Java Interface

- How do we get objects with type ActionListener?
- Objects of a given class are of the type of an interface (e.g., ActionListener) if that class promises to implement the methods of the interface.
- How?
  - by declaring it explicitly using the implements clause.

```
class LoginWindow implements ActionListener  {
```

- Any class can implement an interface.
- A class can implement multiple interfaces.

Being a Listener

- In order for an object of class X to be an action listener, we require that
  - Class X implements the ActionListener interface
    - `implements ActionListener` declaration and
    - defines `actionPerformed(ActionEvent) {...}`
  - Be registered as a listener for the appropriate object
    - by calling the `addActionListener()` method on that object with the listener as an argument.

Handling an event

```
import javax.swing.*;
import java.awt.event.*;
class LoginWindow extends JFrame implements ActionListener{
    public LoginWindow(String title) {
. . .
        contentPane.add(loginButton);
        loginButton.addActionListener(this);
. . .
    }
    public void actionPerformed(ActionEvent e) {
        System.out.println("Login button pressed!");
    }
}
```

Event parameter

- The event parameter that is passed to the listener object can be used to get more information about the source of the event.
- Common use: getSource()
- Used when a single object is listening to multiple GUI elements, to determine which object was the source of the event.

Handling Multiple Sources

```java
public LoginWindow3(String title) {
    . . .
    loginButton.addActionListener(this);
    cancelButton.addActionListener(this);
    . . .
}

public void actionPerformed(ActionEvent e) {
    JButton clickedButton = (JButton) e.getSource();
    if(clickedButton==loginButton){
        String name = nameInput.getText();
        System.out.println(name + " is logging in");
    } else {
        System.out.println("Login canceled");
    }
}
```

Responding to multiple events

```
public LoginWindow3(String title) {
    . . .
    loginButton.addActionListener(this);
    cancelButton.addActionListener(this);
    nameInput.addActionListener(this);
    . . .
}
public void actionPerformed(ActionEvent e) {
    Object source =  e.getSource();
    if(source instanceof JButton){
        JButton button = (JButton) source;
        . . .
    } else if (source instanceof JTextField) {
        String name = ((JTextField)source).getText();
    . . .
    }
}
```

Types of events

- There are several types of events that can be generated.
- A source must register for each specific type of event that it wants to handle
- A different method is called depending upon the type of event
    - ActionEvent (most common)
    - ItemEvent
    - MouseEvent …

3 Types of Listeners

- A separate, special event-handling class
- The same object as the container that holds the GUI elements (most common)
- A third option is to create an anonymous object to handle a single source

Anonymous inner class

```
public LoginWindow(String title) {
        . . .
        loginButton.addActionListener(
                new ActionListener() {
                    public void actionPerformed(ActionEvent e) {
                        String name = nameInput.getText();
                        System.out.println(name +
                                  " is logging in"););
                    }
                }
        );
       . . .
}
```

Anonymous inner classes

- This option essentially creates an instance of an unnamed class that implements the ActionListener interface.
- It provides the body of the method directly.
- This option avoids the need to figure out which object is the source of an action
- However, the class can’t be re-used

Layout Managers

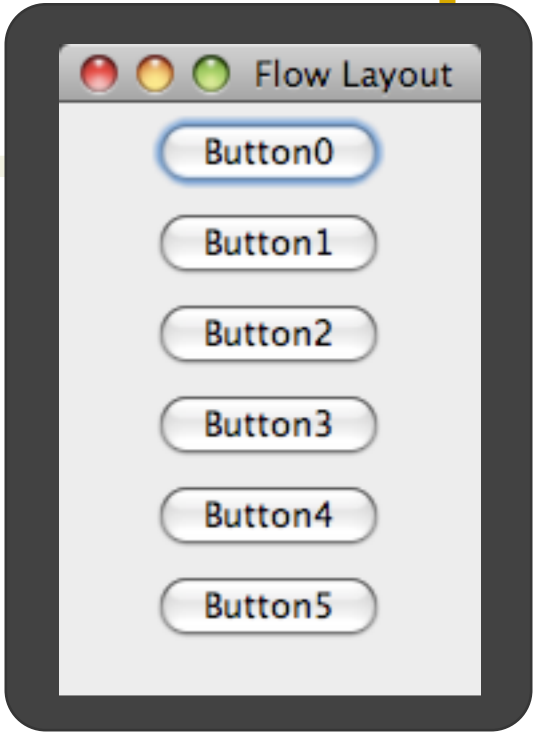
- The placement of GUI elements on a panel can be achieved using
  - absolute positioning (hard to do)
  - layout managers
- Layout managers work best when frames are resized
- Each container (e.g., JPanel, JFrame, etc.) can choose a different layout manager.
- Common managers
  - FlowLayout
  - GridLayout
  - BorderLayout

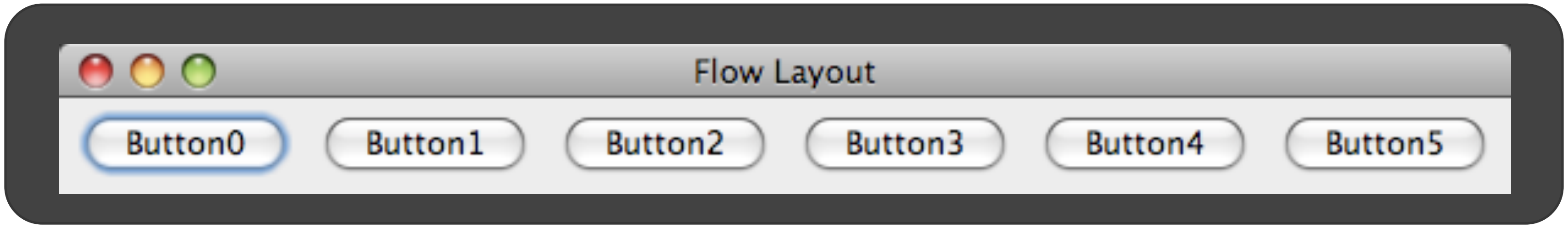
FlowLayout

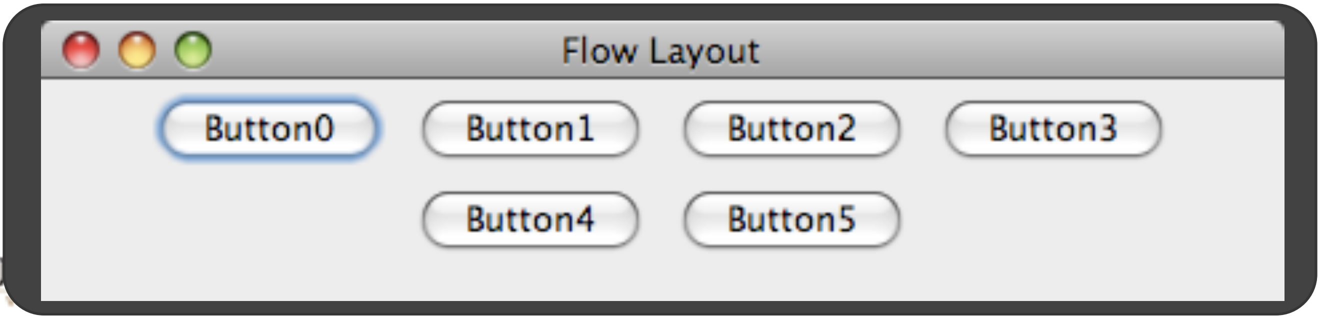
- Elements are added from left to right beginning at the top, similar to text.
- Elements can be justified, and the gaps can be adjusted:
  - FlowLayout(int align, int hGap, int vGap);
  - Align constants: FlowLayout.RIGHT
- Layout may change significantly when the frame is resized.

Flow Layout example

```
Container contentPane = this.getContentPane();
contentPane.setLayout(new FlowLayout());
buttons = new JButton[NUM_BUTTONS];
for(int i = 0;i < NUM_BUTTONS; i++){
    buttons[i] = new JButton("Button" + i);
    contentPane.add(buttons[i]);
}
```





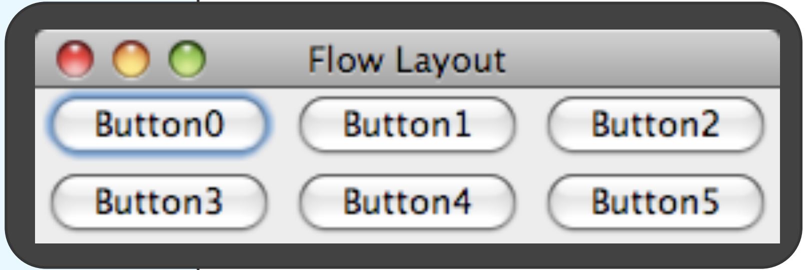


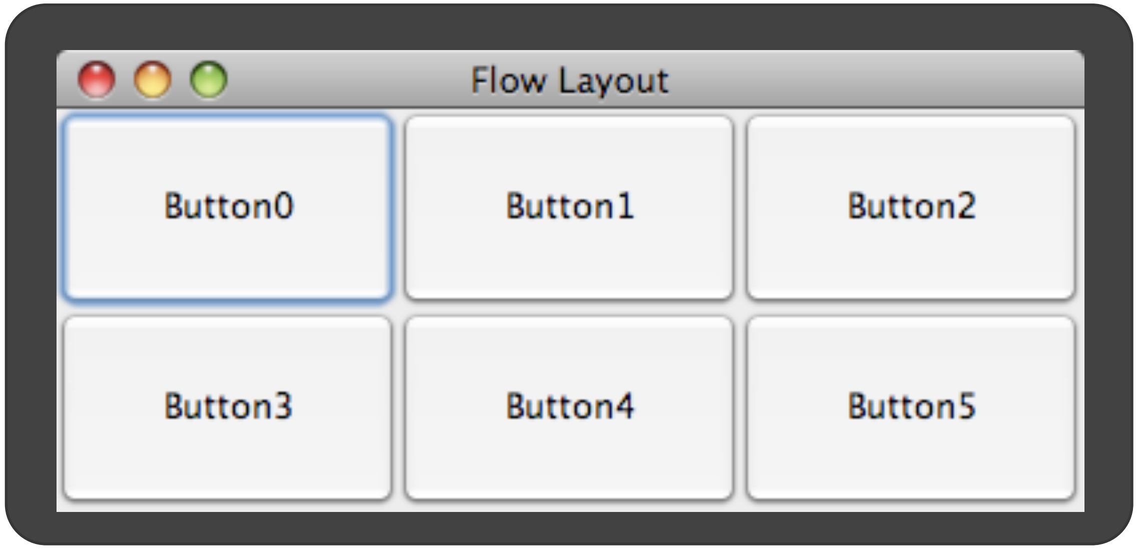
GridLayout

- This layout manager places GUI components on equal-size N by M grids.
- Number of rows and columns declared when creating the layout manager
  - `new GridLayout(nRows, nCols)`
- Components are placed in top-to-bottom, left-to-right order.
- The number of rows and columns remains the same after the frame is resized, but the width and height of each region will change.

GridLayout example

```
Container contentPane = this.getContentPane();
contentPane.setLayout(new GridLayout(2,3));
buttons = new JButton[NUM_BUTTONS];
for(int i=0;i<NUM_BUTTONS;i++){
    buttons[i] = new JButton("Button"+i);
    contentPane.add(buttons[i]);
}
```





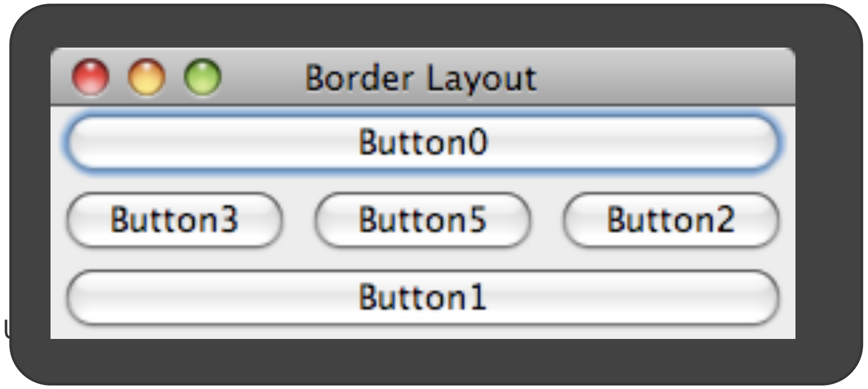
PURDUE UNIVERSITY

BorderLayout

- This layout manager divides the container into five regions: center, north, south, east, and west.
- The north and south regions expand or shrink in height only
- The east and west regions expand or shrink in width only
- The center region expands or shrinks on both height and width.
- Not all regions have to be occupied.

Border Layout example

```java
Container contentPane = this.getContentPane();
contentPane.setLayout(new BorderLayout());
buttons = new JButton[NUM_BUTTONS];
for(int i=0;i<NUM_BUTTONS;i++){
    buttons[i] = new JButton("Button"+i);
}
contentPane.add(buttons[2], BorderLayout.EAST);
contentPane.add(buttons[3], BorderLayout.WEST);
contentPane.add(buttons[0], BorderLayout.NORTH);
contentPane.add(buttons[1], BorderLayout.SOUTH);
contentPane.add(buttons[4], BorderLayout.CENTER);
contentPane.add(buttons[5], BorderLayout.CENTER);
```



Creating GUIs

- Often we need to use multiple panels that are placed within other panels to achieve the desired GUI
- Each panel can have a different layout manager
- Often, we use JPanel objects for this purpose
- The panels are invisible, but can have a visible border around them

Common GUI elements

- JButton
- JRadioButton
- JCheckBox
- JLabel
- JTextField
- JComboBox
- see http://docs.oracle.com/javase/tutorial/ui/features/components.html

Examples

- SampleGUITextArea
  - JTextArea
  - JScrollPane
- SampleGUICheckBox
  - JCheckBox
- SampleGUIRadioButton
  - JRadioButton
- SampleGUIComboBox
  - JComboBox

Nested Panels

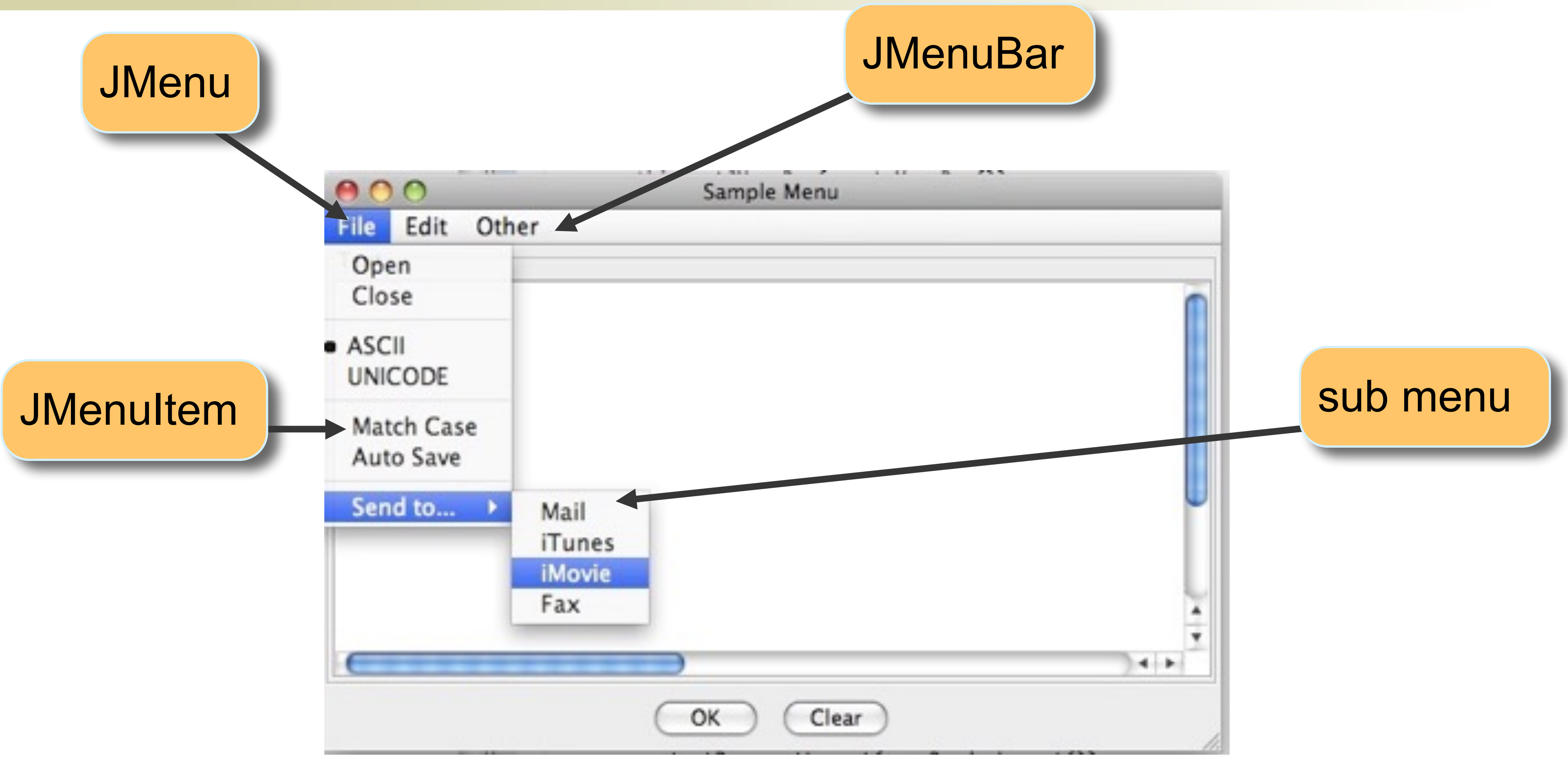
- Building more complex GUIs is achieved using nested panels.
- Instead of adding all components to a single content pane, we add components to panels, and then add these panels to other panels, …
- Each panel can have a different layout manager
- SampleGUI

Menus

Menus

- Menus are created using three classes: JMenuBar, JMenu, and JMenuItem.
- A JMenuBar object represents the entire menu that is attached to a single frame.
- The high-level entries in the menu bar correspond to JMenu objects (such as File or Edit)
- Each JMenu object can have
  - Selectable items that are JMenuItem objects (such as Copy, Cut, or Paste)
  - Submenus (another JMenu object)
- Only the JMenuItem objects generate events.

Menu elements



Other Features

- Using the setAccelerator() method, we can set keyboard shortcuts for menu items
- We can also attach Icons (objects from the class ImageIcon) to menu items
- More in recitation

Creating a Menu

1. Create a JMenuBar object;
2. Create JMenu objects
3. Create JMenuItem objects and add them to JMenu objects;
4. Add the JMenu objects to the menu bar
5. Attach the JMenuBar object to a frame

See example SampleGUIMenu

Event Types

- There are many types of events
  - Action events
  - Item events
  - Keyboard events
  - Mouse events
  - Mouse Motion events
  - Window events
  - Container events

Mouse Events

- Mouse events include such user interactions as
  - clicking mouse buttons
  - moving the mouse
  - dragging the mouse (moving the mouse while the mouse button is being pressed)
- The MouseListener interface handles mouse button events:

  `mouseClicked`, `mouseEntered`, `mouseExited`, `mousePressed`, and `mouseReleased`
- The MouseMotionListener interface handles mouse movement

  `mouseDragged` and `mouseMoved`.

Useful MouseEvent methods

- getClickCount()
- getX() , getY()
- getXOnScreen(), getYOnScreen()
- getButton()

- See API for details.

Other interesting classes

- Font
- Colors
- JFileChooser
- JApplet
- ImageIcon
- AudioClip