Introduction to GUI Programming (Part I)

188230 Advanced Computer Programming

Asst. Prof. Dr. Kanda Runapongsa Saikaew
(krunapon@kku.ac.th)
Department of Computer Engineering
Khon Kaen University

Agenda

- The Basic GUI Application
- Basic Components
- Basic Layout
- Menus

Introduction to GUI

- Computer users today expect to interact with their computers using a graphical user interface (GUI)
- Java can be used to write GUI programs ranging from simple applets which run on a Web page to sophisticated stand-alone applications.
- One big difference is that GUI programs are event-driven

Event-driven Programming

- Events are user actions
  - Clicking on a button
  - Pressing a key on the keyboard generate events
- The program must respond to these events as they occur
- Objects are everywhere in GUI programming
  - Events are objects
  - GUI components such as buttons and menus are objects

The Basic GUI Application

- There are two basic types of GUI program in Java
  - Stand-alone applications
  - Applets
- A stand-alone application is a program that runs on its own, without depending on a Web browser
- An applet is a program that runs in a rectangular area on a Web page
  - Applets are generally small programs

A GUI “Hello World” Program

```
import javax.swing.JOptionPane;
public class HelloWorldGUI1 {
   public static void main(String[] args) {
      JOptionPane.showMessageDialog( null, "Hello World!" );
   }
}
```

Program Output

- When this program is run, a window appears on the screen that contains the message “Hello World!”
- The window also contains an “OK” button for the user to click after reading the message
- When the user clicks this button, the window closes and the program ends

HelloWorldGUI1 Explanation

- The reason the program was so easy to write is that all the work is done by

  showMessageDialog(), a static method in the built-in class JOptionPane
- The source code “imports” the class javax.swing.JOptionPane to make it possible to refer to the JOptionPane class using its simple name
  - JOptionPane.showMessageDialog

HelloWorldGUI2 (1/4)

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class HelloWorldGUI2 {
   private static class HelloWorldDisplay extends
JPanel {
```

HelloWorldGUI2 (2/4)

```
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.drawString("Hello World!", 20, 30);
    }
}
private static class ButtonHandler implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        System.exit(0);
    }
}
```

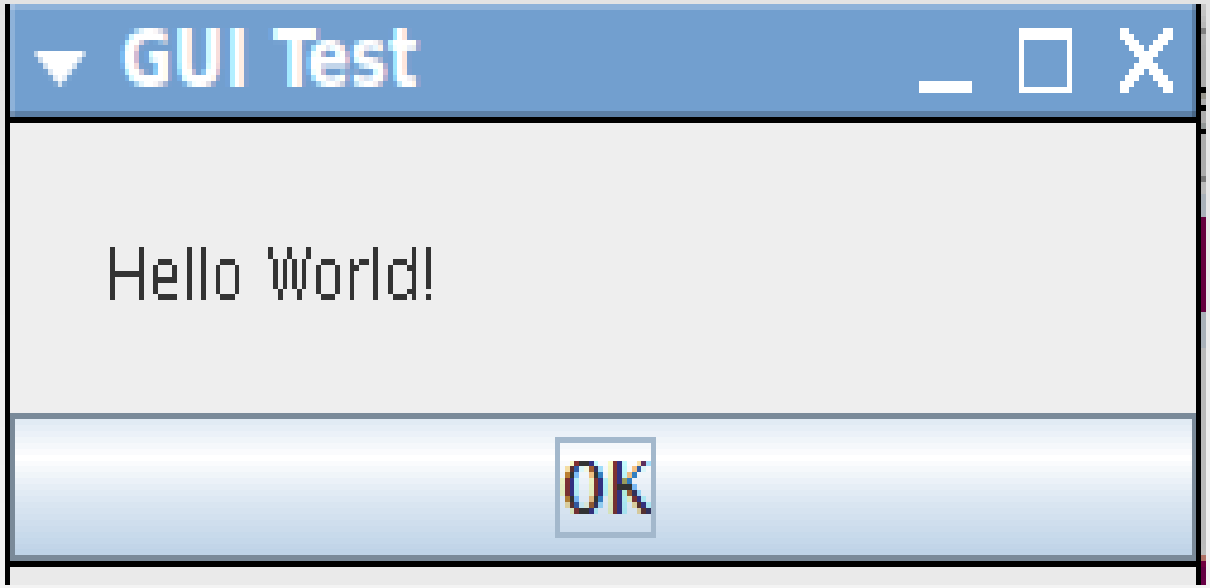
HelloWorldGUI2 (3/4)

```
public static void main(String[] args) {
   HelloWorldDisplay displayPanel = new HelloWorldDisplay();
   JButton okButton = new JButton("OK");
   ButtonHandler listener = new ButtonHandler();
   okButton.addActionListener(listener);
   JPanel content = new JPanel();
   content.setLayout(new BorderLayout());
   content.add(displayPanel, BorderLayout.CENTER);
   content.add(okButton, BorderLayout.SOUTH);
```

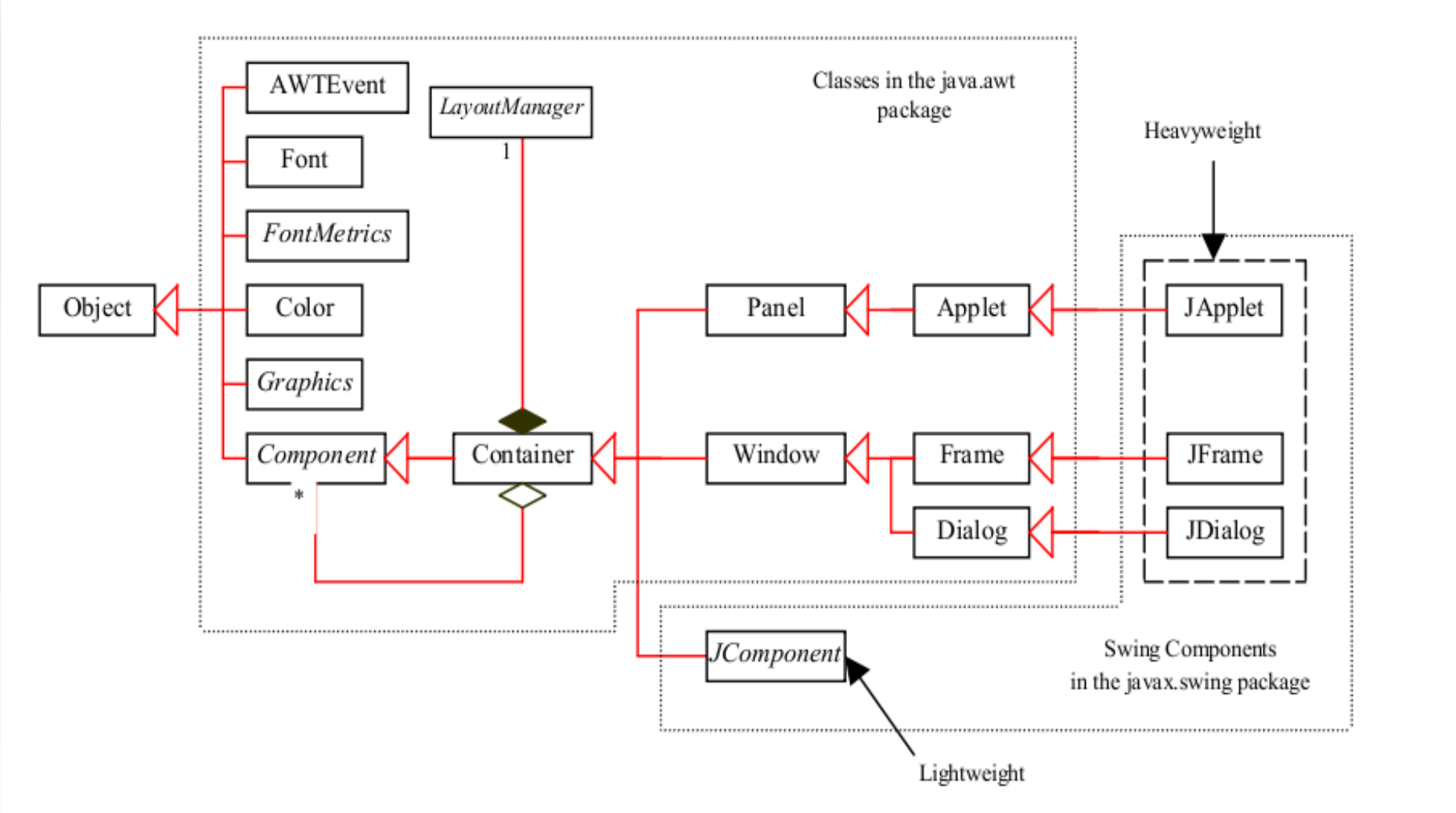
HelloWorldGUI2 (4/4)

```
    JFrame window = new JFrame("GUI Test");
    window.setContentPane(content);
    window.setSize(250,100);
    window.setLocation(100,100);
    window.setVisible(true);
  }
}
```

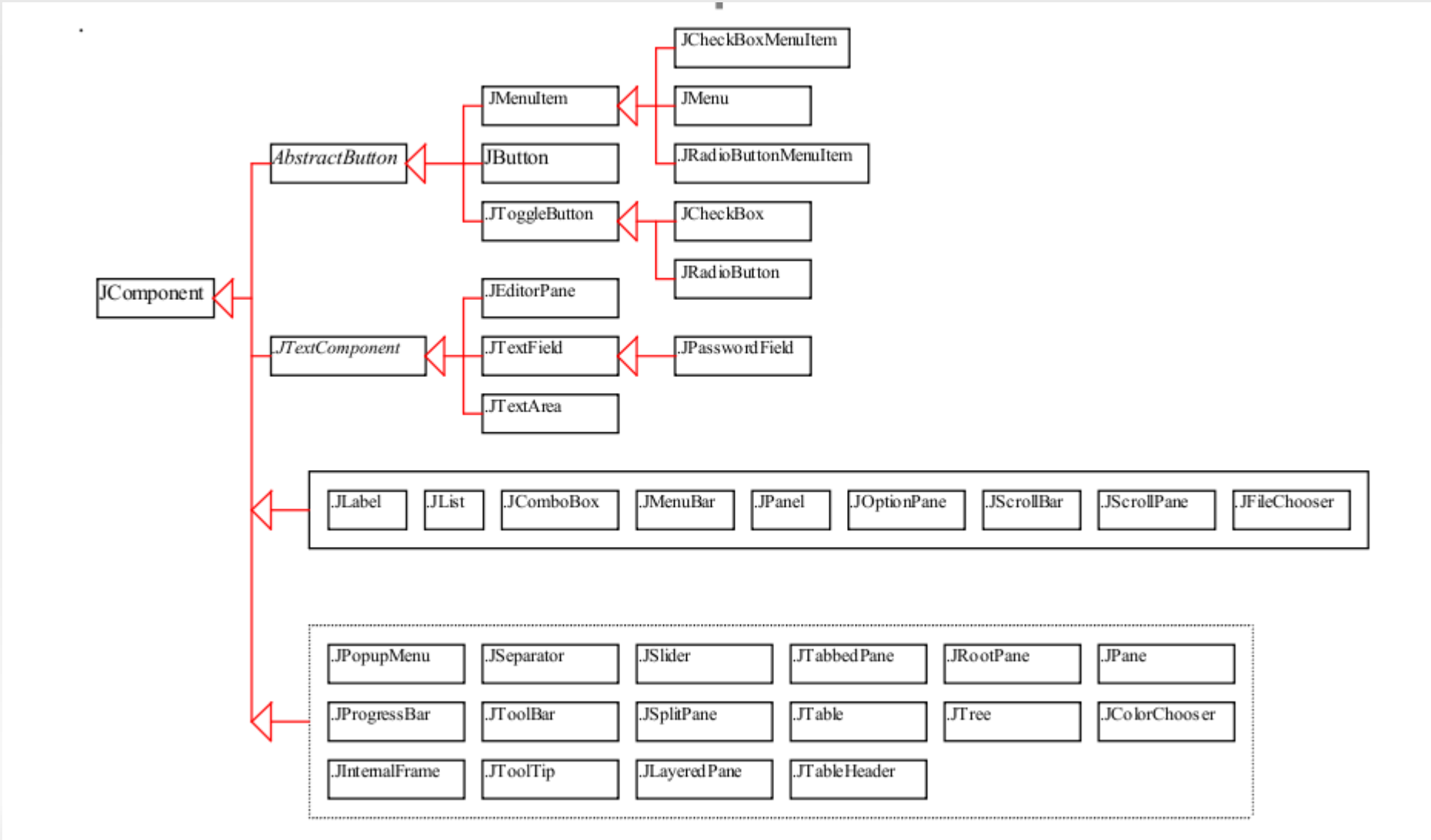
HelloWorldGUI2 Output



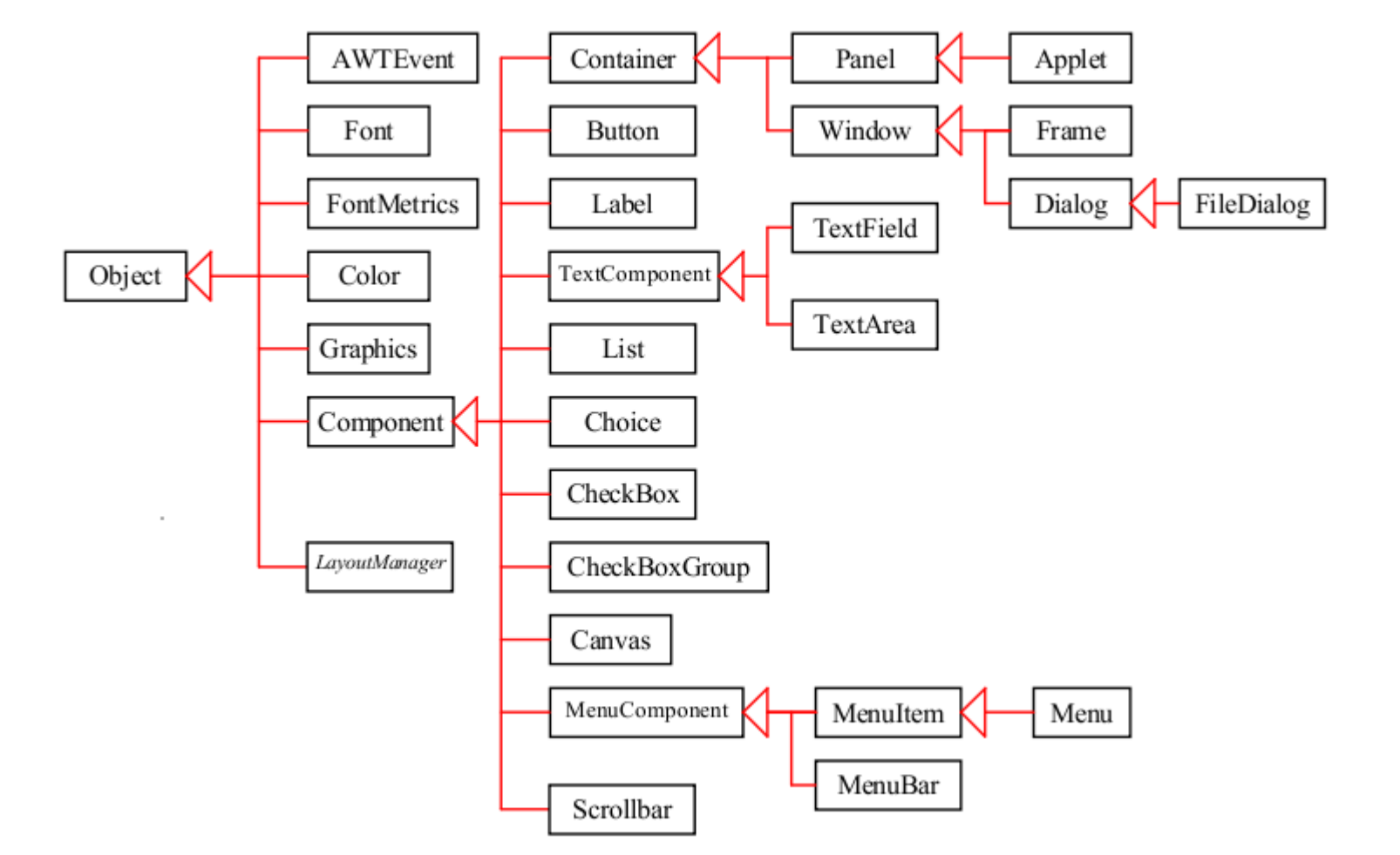
Graphics Class Hierarchy



JComponent and Its Subclasses



AWT


AWTEvent
Font
FontMetrics
Object
Color
Graphics
Component
LayoutManager
Container
Button
Label
TextComponent
List
Choice
CheckBox
CheckBoxGroup
Canvas
MenuComponent
Scrollbar
Panel
Applet
Window
Frame
Dialog
FileDialog
TextField
TextArea
MenuItem
Menu
MenuBar

AWT and swing (1/2)

- There have been three major versions of the graphical user interface
- The GUI that Java offers has changed with each version of Java.
- Java 1.0
  - This offered the AWT (Abstract Windowing Toolkit) which was a basic set of tools for a GUI, but it lacked some important features
  - It is now considered obsolete and should not be used.

AWT and swing (2/2)

- Java 1.1
  - This version made a basic change in event handling of the AWT
  - It is still commonly used in applets so that the applets work on all versions of browsers
  - However, using these AWT components is not a good idea for new applications because they are much more limited.
- Java 1.2/2.0
  - This introduced the Swing GUI, which provides the facilities to create a really good interface

Using Swing Components, not AWT

- Never mix the components from Swing and AWT
  - The Swing component names generally start with 'J' (eg JButton), and the AWT components don't (eg Button).
- Use Swing components because they are "lighter", faster, and use less memory
- Use some AWT classes, such as Graphics, Color, Font, ...

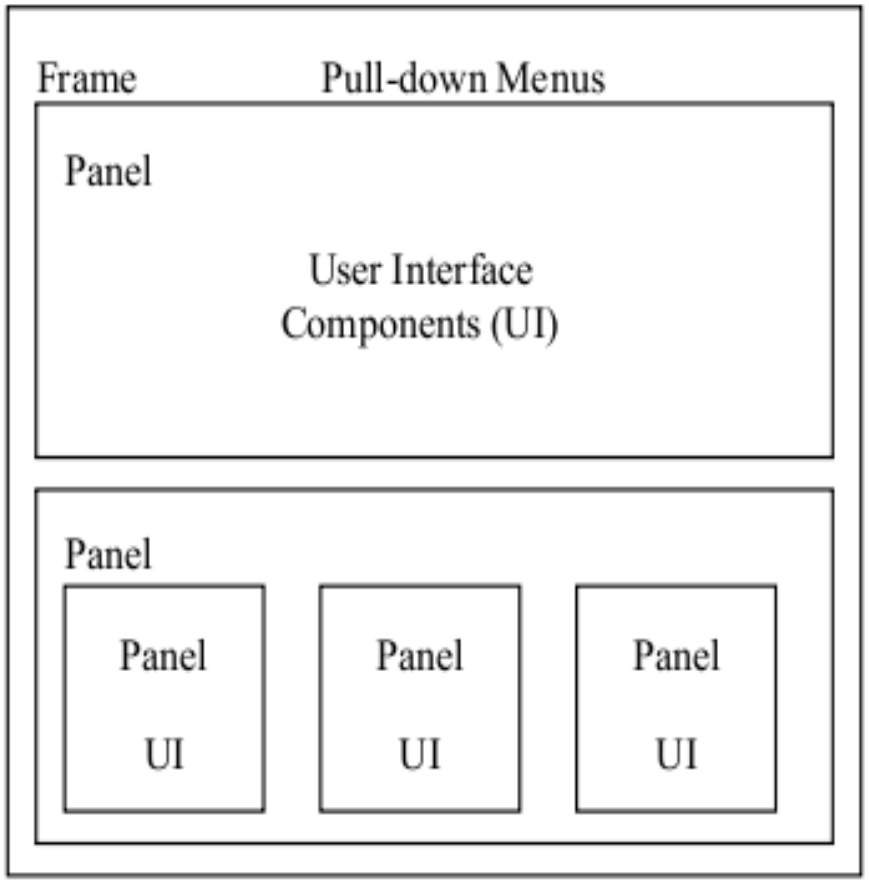
Packages Needed to Import

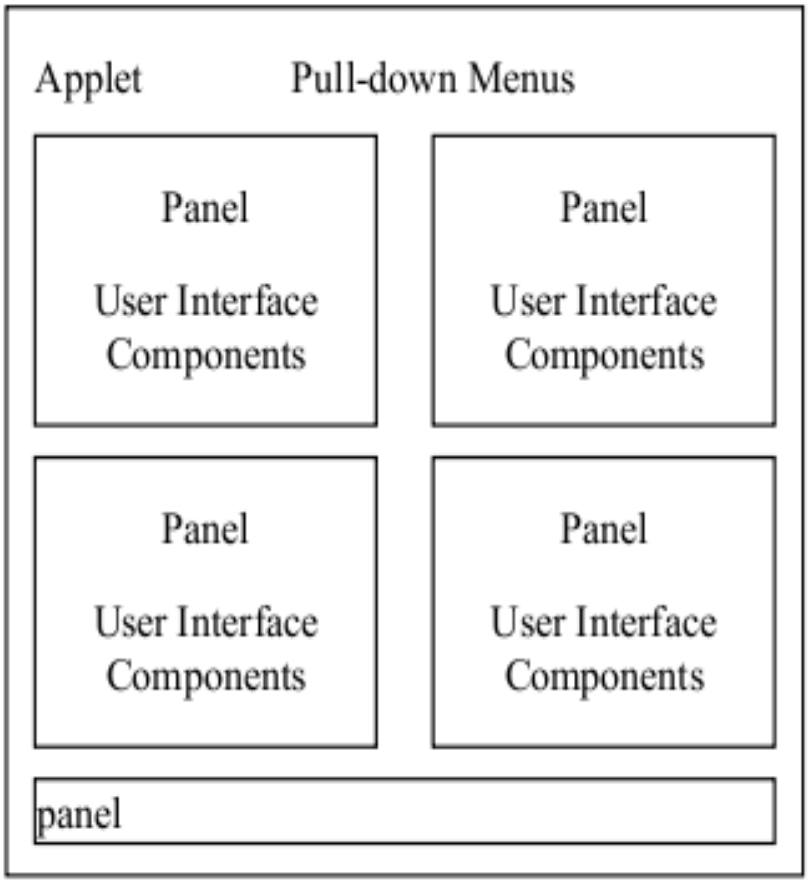
- For a graphical user interface, you typically have these import statements:
  - import java.awt.*;
  - import java.awt.event.*;
  - import javax.swing.*;
  - import javax.swing.event.*;
- There are many other packages, but these are the most common. Note that the AWT packages are in "java" and the Swing classes are in "javax"

GUI Components

- In a Java GUI program, each GUI component in the interface is represented by an object
- One of the most fundamental types of component is the window
  - Windows have many behaviors
  - They can be opened and closed
  - They can be resized
  - They have “titles” that are displayed in the title bar above the window
  - They can contain other GUI components such as buttons and menus

UI Components





GUI Overview

- GUI is easy to create after you have a few basic elements from these groups
  - Top-level Containers (eg, JFrame)
  - Intermediate Containers (eg, JPanel, JMenuBar)
  - Layouts (eg., FlowLayout, BorderLayout)
  - Components (eg, JButton, JLabel)
  - Listeners (eg, ActionListener)

Top-level Containers

- Top-level Containers are windows (JFrame, JDialog)
- Each window has two intermediate containers
  - A menubar which contains the menus and
  - The content pane which holds the components

Intermediate Containers

- Intermediate Containers (eg, JPanel) contain components
- Components are added to the intermediate container
- Every container has a layout manager, which specifies how the components are arranged in the container

Layouts

- Layouts specify how to arrange components in a JPanel (or other intermediate container)
- Every JPanel starts with a default layout manager but it's better to set it explicitly
- Use one of Java's layout managers
  - FlowLayout
  - BorderLayout
  - GridLayout

Components

- Components are the user interface controls like buttons (JButton), text boxes (JTextField, JTextArea), menus
- These components are added to a container with the add() method, and the place they appear on a container is determined by the layout which is used for that container

Listeners

- Listeners are attached to components and contain the code that is called when the component is used (eg., a button is clicked)
- This is how a user action on a component(such as a click on a JButton or moving a JSlider) is connected to a Java method

Optional Graphics Setting

- Setting Color by using class Color
- Setting Font by using class Font
- Draw many shapes by using drawxxx() methods
- Setting Animation
- Setting Mouse Pointer
- Setting Sound

JFrame (1/2)

- There are actually several different types of window, but the most common type is represented by the JFrame class (which is included in the package javax.swing)
- A JFrame is an independent window that can, for example, act as the main window of an application
- One of the most important things to understand is that a JFrame object comes with many of the behaviors of windows already programmed in

JFrame (2/2)

- JFrame comes with the basic properties shared by all windows, such as a titlebar and the ability to be opened and closed
- Since a JFrame comes with these behaviors,

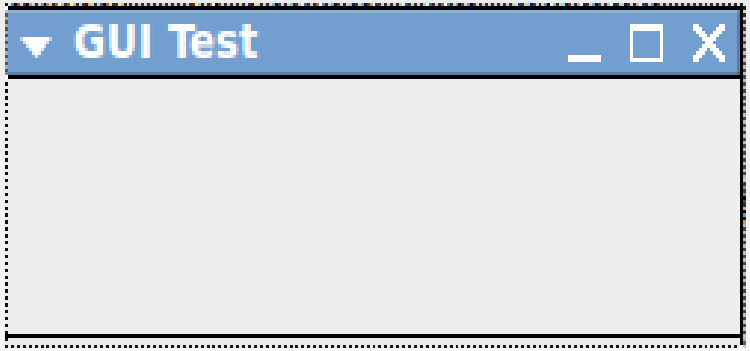
  you don't have to program them yourself!
  - This is, of course, one of the central ideas of object-oriented programming
- What a JFrame doesn't come with is content
  - If you don't add any other content to a JFrame, it will just display a large blank area.

Creating JFrame Object

- // Declares a variable, window, of type JFrame and sets it to refer to a new window object with the statement:
  - JFrame window = new JFrame("GUI Test");
- The parameter in the constructor, “GUI Test”, specifies the title that will be displayed in the titlebar of the window
- This line creates the window object, but the window itself is not yet visible on the screen

JFrame1

```
import javax.swing.*;
public class JFrame1 {
public static void main(String[] args) {
JFrame window = new JFrame("GUI Test");
window.setSize(250,100);
window.setLocation(100,100);
window.setVisible(true);
}
}
```



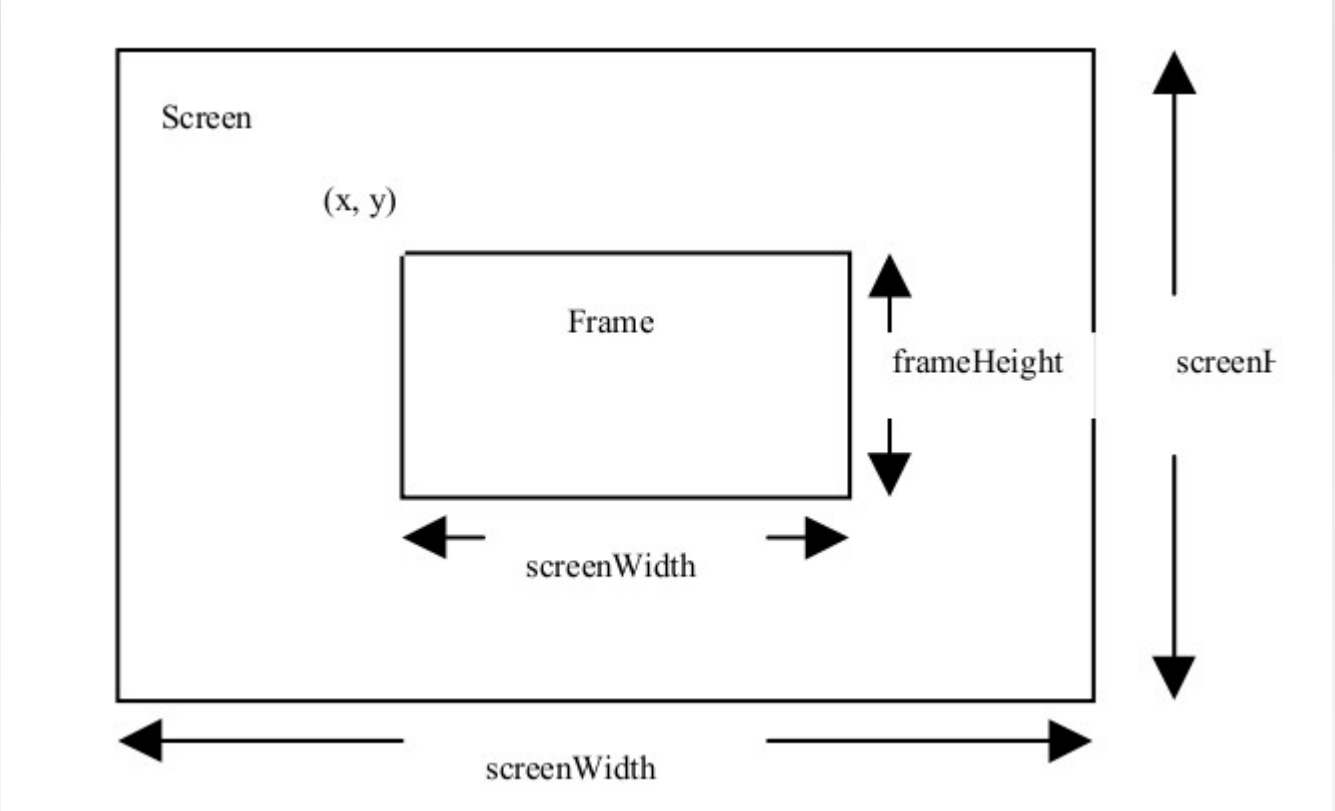
JFrame1 Explanation

- // window will be 250 pixels wide and 100 pixels high
  window.setSize(250,100);
- // the upper left corner of the window will be 100 pixels
  // over from the left edge of the screen and 100 pixels
  // down from the top
  window.setLocation(100,100);
- // window is actually made visible on the screen with
  // the command:
  window.setVisible(true);

Centering Frame

- By default, a frame is displayed in the upper-left corner of the screen
- To display a frame at a specified location, you can use the setLocation(x,y) method in the JFrame class
- This method places the upper-left corner of a frame at location (x,y)

Frame Size and Location



Content Pane

- The content that is displayed in a JFrame is called its content pane
- The content can be a component of type JPanel
- Then we can replace the original blank content pane with a JPanel object
  - window.setContentPane(content)

Using Panels as Containers

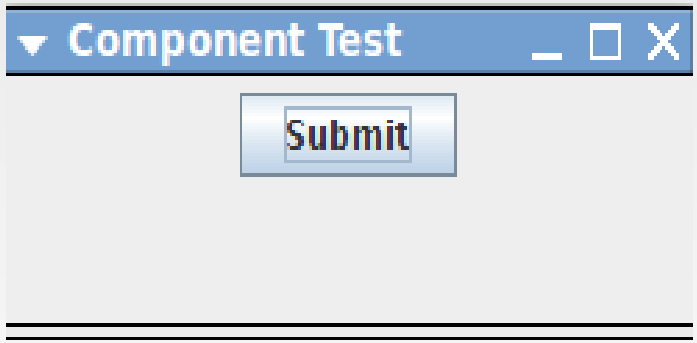
- Panels act as smaller containers for grouping user interface components
- It is recommended that you place the user interface components in panels and place the panels in a frame
- You can also place panels in a panel

JPanel

- JPanel is another of the fundamental classes in Swing
- The basic JPanel is, again, just a blank rectangle
- There are two ways to make a useful JPanel
  - The first is to add other components to the panel
  - The second is to draw something in the panel.

Adding a Component

```
public class JFrame2 {
public static void main(String[] args) {
JFrame window = new JFrame("Component Test");
JPanel panel = new JPanel();
JButton button = new JButton("Submit");
panel.add(button);
window.add(panel);
window.setSize(250,100);
window.setLocation(100,100);
window.setVisible(true);
}}
```



Layout Managers

- Java's layout managers provide a level of abstraction to automatically map your user interface on all windowing systems.
- The UI components are placed in containers
- Each container has a layout manager to arrange the UI components within the container

Kinds of Layouts Managers (1/2)

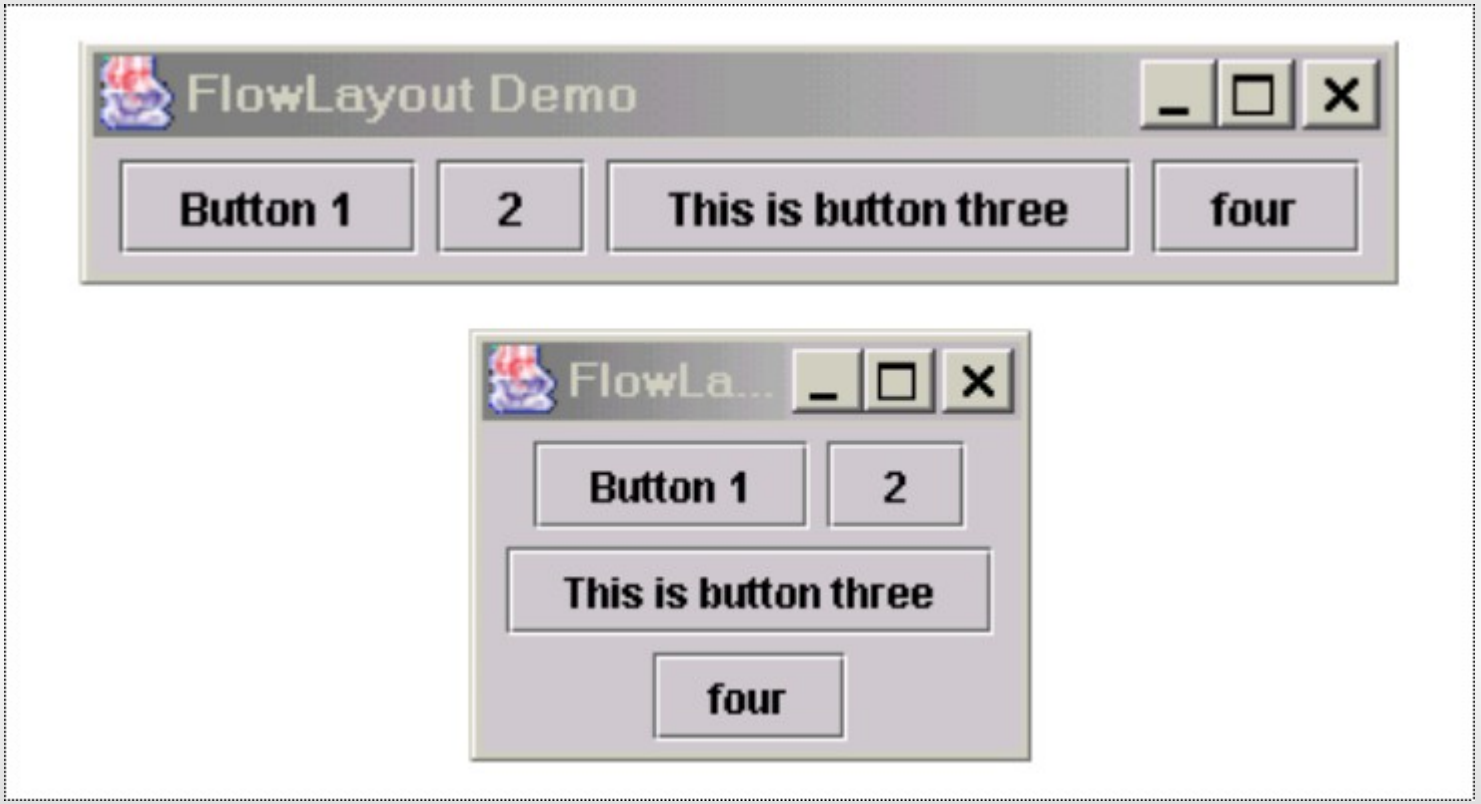
- FlowLayout
  - Arranges components in the order left-to-right, top-to-bottom (by default)
  - We can set components in the order right-to-left as well
- BorderLayout
  - Lays out components in BorderLayout.NORTH, EAST, SOUTH, WEST, and CENTER sections.

Kinds of Layout Managers (2/2)

- GridLayout
  - Lays out components in equal sized rectangular grid, added r-t-l, top-to-bottom.
- CardLayout
  - Panels are placed on top of each other like stack of cards -- only one visible at a time
- GridBagLayout
  - Panel divided into rows&cols of possibly unequal size. Overall best, but difficult

FlowLayout Examples

- The components are arranged in the container from left to right in the order in which they were added
- When one row becomes filled, a new row is started.



FlowLayoutDemo (1/2)

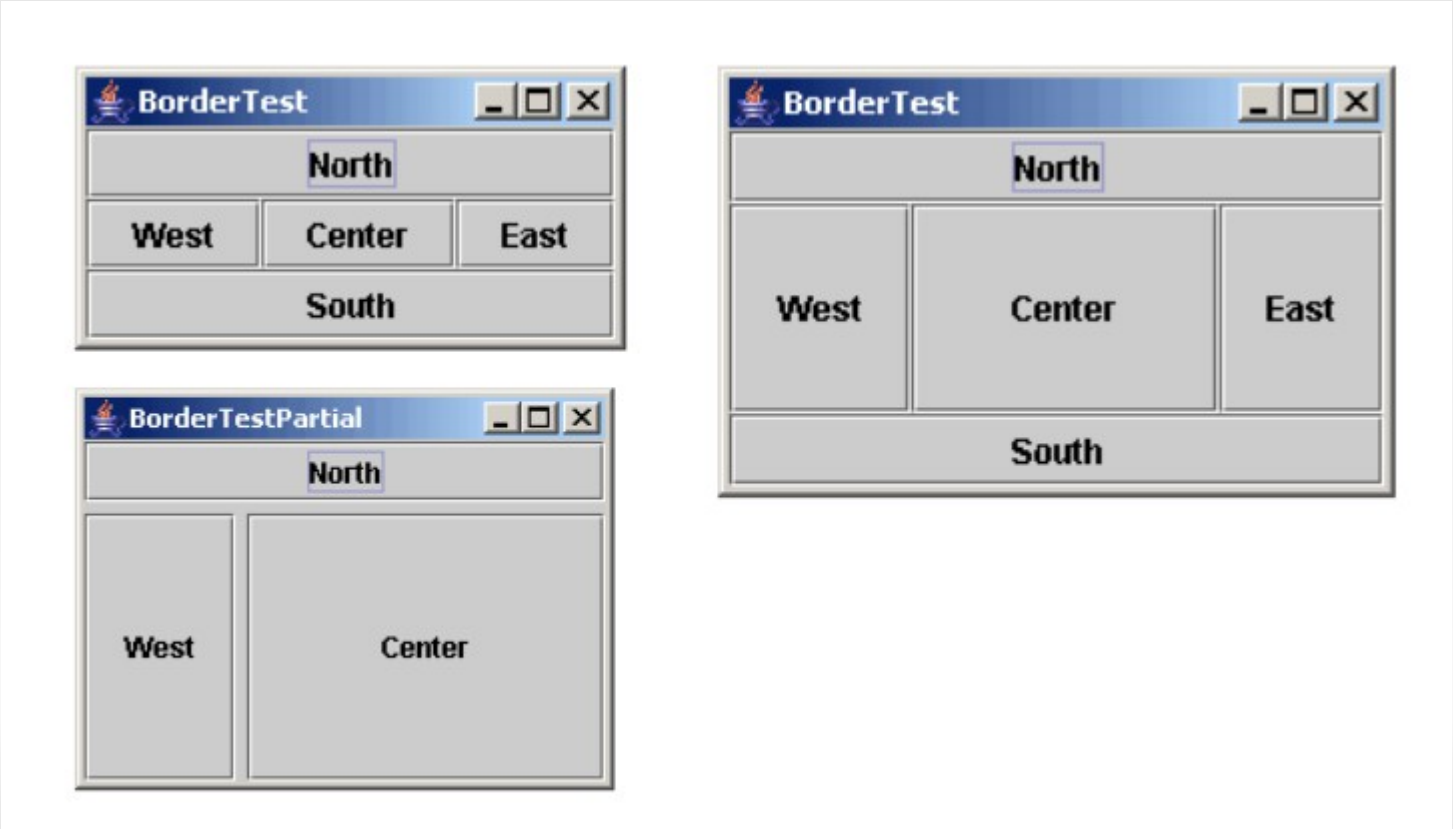
```
import java.awt.*;
import javax.swing.*;
public class FlowLayoutDemo {
public static void addComponents(Container contentPane) {
        contentPane.setLayout(new FlowLayout());
        contentPane.add(new JButton("Button 1"));
        contentPane.add(new JButton("2"));
        contentPane.add(new JButton("This is button three"));
        contentPane.add(new JButton("four"));
}
```

FlowLayoutDemo (2/2)

```
public static void main(String[] args) {
JFrame frame = new JFrame("FlowLayoutDemo");
addComponents(frame.getContentPane());
        // adjust frame size to fit with
       // all components inside the frame
        frame.pack();
        frame.setLocation(100,100);
        frame.setVisible(true);
}
}
```

BorderLayout Examples

- BorderLayout manager divides the window into five areas: East, South, West, North, and Center



BorderLayoutDemo

```
import java.awt.*;
import javax.swing.*;
public class BorderLayoutDemo {
public static void addComponents(Container cp) {
        cp.setLayout(new BorderLayout());
        cp.add(new Button("North"), BorderLayout.NORTH);
        cp.add(new Button("South"), BorderLayout.SOUTH);
        cp.add(new Button("East"), BorderLayout.EAST);
        cp.add(new Button("West"), BorderLayout.WEST);
        cp.add(new Button("Center"), BorderLayout.CENTER);
}
```

GridLayout Examples

- GridLayout arranges components in a grid (matrix) formation with the number of rows and columns defined by the constructor
- The components are placed in the grid from left to right starting with the first row, then the second, and so on.

GridLayoutDemo

```
import java.awt.*;
import javax.swing.*;
public class GridLayoutDemo {
public static void addComponents(Container cp) {
        cp.setLayout(new GridLayout(2,2));
        cp.add(new Button("Button 1"));
        cp.add(new Button("2"));
        cp.add(new Button(""));
        cp.add(new Button("This is button three"));
}
```

Basic GUI Components

- JButton: Standard clickable button
- JLabel: Displaying fixed text
- JTextField: Box that accepts one line of text
- JTextArea: Box that contains multiple lines of text separated by '\n'
- JCheckBox: Check box followed by text
- JList: List of string values
- JMenuBar, JMenu, JMenuItem: Components for creating menus

JButton

The JButton class has a constructor that takes a string as a parameter.

- This string becomes the text displayed on the button
- Example:
  - JButton stopGoButton = new JButton("Go")
- This creates a button object that will display the text, “Go” (but remember that the button must still be added to a container before it can appear on the screen)

JLabel

JLabel exists just to display a line of text

- The constructor for a JLabel specifies the text to be displayed:
  - JLabel message = new JLabel("Hello World!");
- The text cannot be edited by the user, although it can be changed by using setText(String newText) method
  - message.setText(“Goodbye World!”);

JTextField and JTextArea

- JTextField and JTextArea classes represent components that contain text that can be edited
- A JTextField holds a single line of text, while a JTextArea can hold multiple lines
- The contents of the component can be retrieved by calling its getText() instance method, which returns a value of type String
- Its content also can be set by using setText()
- If you want to stop the user from modifying the text, you can call setEditable(false)

JTextField

- You can declare a text field by specifying the number of characters that should be visible in the text field
  - JTextField name = new JTextField(10);
- You can use the following constructors, which specify the initial contents of the text field:
  - public JTextField(String contents);
  - public JTextField(String contents, int columns);

JTextFieldDemo (1/2)

```
3 import java.awt.*;
4 import javax.swing.*;
5 public class JTextFieldDemo {
6     public static void addComponents(Container cp) {
7         cp.setLayout(new FlowLayout());
8         JLabel name = new JLabel("name:");
9         JTextField nameInput =
10            new JTextField("Kanda ", 15);
11        cp.add(name);
12        cp.add(nameInput);
13    }
```

JTextFieldDemo (2/2)

```
15      public static void main(String[] args) {
16          JFrame frame = new JFrame("JTextFieldDemo");
17          JPanel content = new JPanel();
18          addComponents(content);
19          JPanel buttonPane = new JPanel();
20          buttonPane.add(new JButton("Submit"));
21          frame.setLayout(new BorderLayout());
22          frame.add(content, BorderLayout.NORTH);
23          frame.add(buttonPane,
24                  BorderLayout.AFTER_LAST_LINE);
25          // Display the window.
26          frame.pack();
27          frame.setLocation(100, 100);
28          frame.setVisible(true);
29      }
30  }
```

JTextArea

- JTextArea can be created by specifying the number of rows and columns in the text area
  - JTextArea ta = new JTextArea(rows, cols);
- JTextArea also has gets and sets methods
  - String s = ta.getText();
  - ta.setText(s); //Sets text to s.
- JTextArea supports appending & inserting text
  - ta.append(s); //Adds s to end of existing text
  - ta.insert(s, pos); //Inserts s at position pos.

JScrollPane

- To scroll a text area to see all the text, a text area should come with scroll bars
- To get scroll bars for a text area, you have to put the JTextArea inside another component,

  called a JScrollPane
- This can be done as follows:
  - JTextArea inputArea = new JTextArea();
  - JScrollPane scroller = new JScrollPane(inputArea);

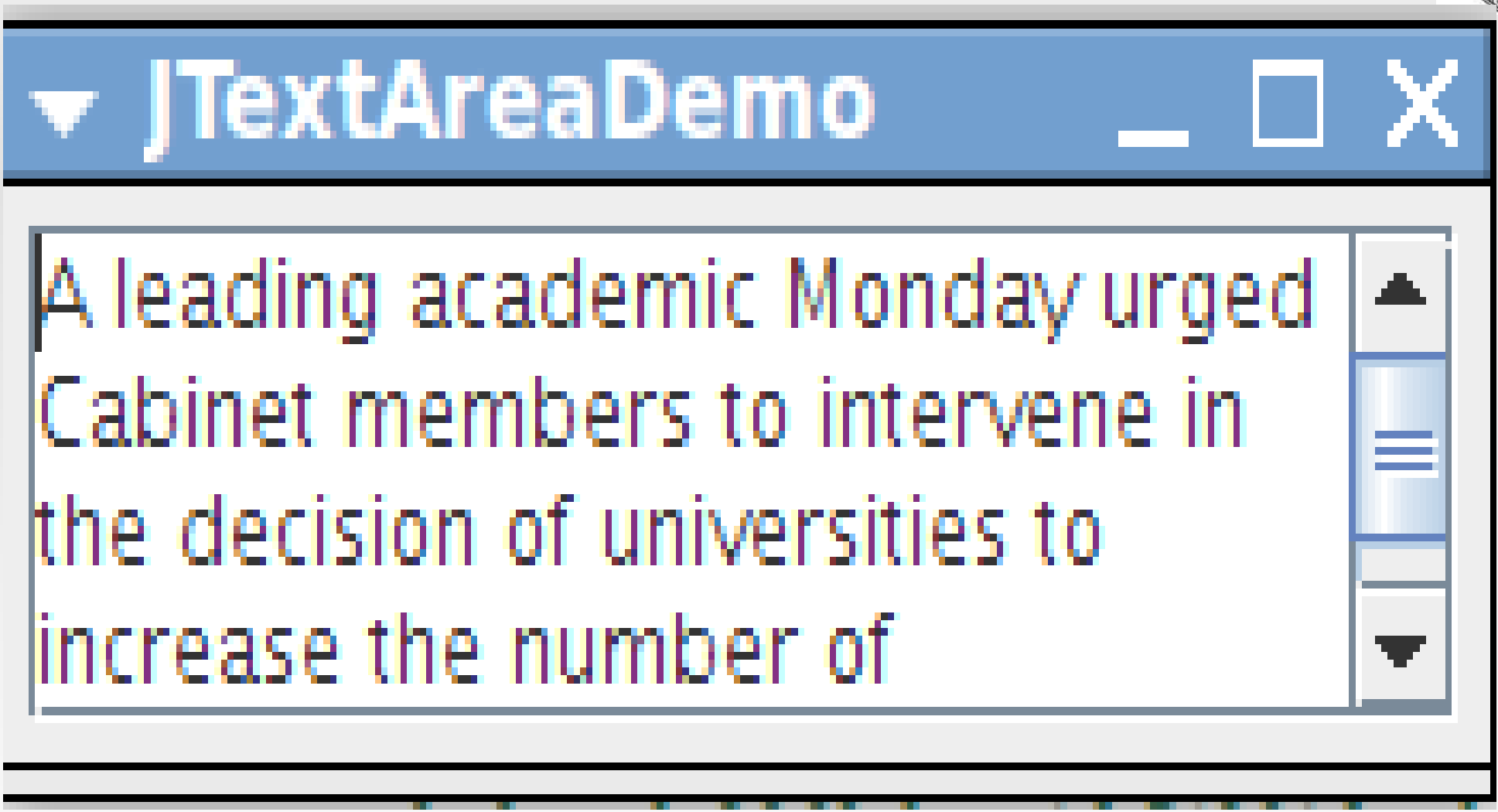
JScrollPane and JTextArea

- The scroll pane provides scroll bars that can be used to scroll the text in the text area
- The scroll bars will appear only when needed, that is when the size of the text exceeds the size of the text area
- Note that when you want to put the text area into a container, you should add the scroll pane, not the text area itself, to the container

JTextAreaDemo

```
6     public static void addComponents(Container cp) {
7         cp.setLayout(new FlowLayout());
8         // text area with 4 rows and 20 columns
9         JTextArea news = new JTextArea(4, 20);
10        news.setLineWrap(true);
11        news.setWrapStyleWord(true);
12        news.setText("A leading academic Monday urged Cabinet ");
13        news.append("members to intervene in the decision of ");
14        news.append("universities to increase the number of ");
15        news.append("Professional Aptitude Tests (PATs).");
16        JScrollPane sp = new JScrollPane(news);
17        cp.add(sp);
18    }
19    public static void main(String[] args) {
20        JFrame frame = new JFrame("JTextAreaDemo");
21        addComponents(frame.getContentPane());
22        frame.pack();
23        frame.setLocation(100, 100);
24        frame.setVisible(true);
25    }
```

JTextAreaDemo Result



JTextArea setLineWrap Method

- setLineWrap(boolean wrap)
  - Tells what should happen when a line of text is too long to be displayed in the text area
  - If wrap is true, then any line that is too long will be “wrapped” onto the next line
  - If wrap is false, the line will simply extend outside the text area, and the user will have to scroll the text area horizontally to see the entire line
  - The default value of wrap is false.

JTextArea setWrapStyleWord

- Sets the style of wrapping used if the text area is wrapping lines
- If set to true the lines will be wrapped at word boundaries (whitespace)
- If they are too long to fit within the allocated width. If set to false, the lines will be wrapped at character boundaries
- By default this property is false.

CheckBox and Radio Buttons

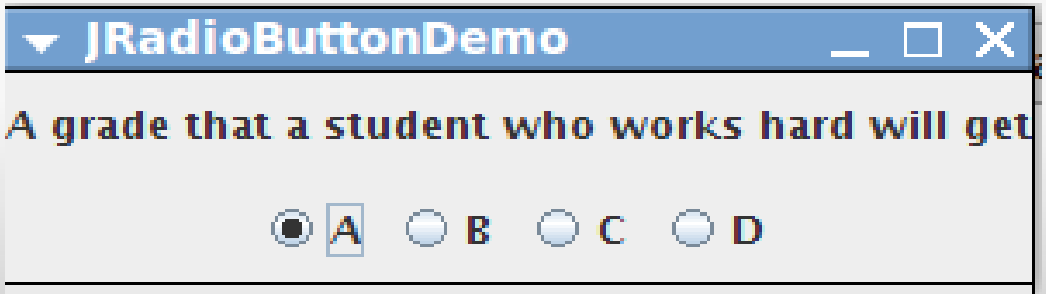
- Any number of check boxes in a group — none, some, or all — can be selected
- A group of radio buttons, on the other hand, can have only one button selected
- JCheckBox can be created using these constructors
  - new JCheckBox(String name)
  - new JCheckBox(String name,boolean isSelected)

JCheckBoxDemo

```
 6  public static void addComponents(Container cp) {
 7      cp.setLayout(new GridLayout(2,1));
 8      JLabel label = new JLabel("Languages that you know well");
 9      JPanel boxes = new JPanel();
10      boxes.setLayout(new FlowLayout());
11      JCheckBox cPlusPlus = new JCheckBox("C++", true);
12      JCheckBox php = new JCheckBox("PHP");
13      JCheckBox java = new JCheckBox("Java", true);
14      JCheckBox javascript = new JCheckBox("JavaScript");
15      boxes.add(cPlusPlus);
16      boxes.add(php);
17      boxes.add(java);
18      boxes.add(javascript);
19      cp.add(label);
20      cp.add(boxes);
21  }
22  public static void main(String[] args) {
23      JFrame frame = new JFrame("JCheckBoxDemo");
24      addComponents(frame.getContentPane());
25      frame.pack();
```

JRadioButtonDemo

```
6  public class JRadioButtonDemo {
7      public static void addComponents(Container cp) {
8          cp.setLayout(new GridLayout(2,1));
9          JLabel label = new JLabel("A grade that a student who w
10         JPanel grades = new JPanel();
11         JRadioButton a = new JRadioButton("A", true);
12         JRadioButton b = new JRadioButton("B");
13         JRadioButton c = new JRadioButton("C");
14         JRadioButton d = new JRadioButton("D");
15         grades.add(a);
16         grades.add(b);
17         grades.add(c);
18         grades.add(d);
19         cp.add(label);
20         cp.add(grades);
21     }
22     public static void main(String[] args) {
23         JFrame frame = new JFrame("JRadioButtonDemo");
24         addComponents(frame.getContentPane());
25         frame.pack();
```



JComboBox

- JComboBox class provides a way to let the user select one option from a list of options.
- The options are presented as a kind of pop-up menu, and only the currently selected option is visible
- When a JComboBox object is first constructed, it initially contains no items
- An item is added to the bottom of the menu by calling the combo box's instance method, addItem(str), where str is the string that will be displayed in the menu

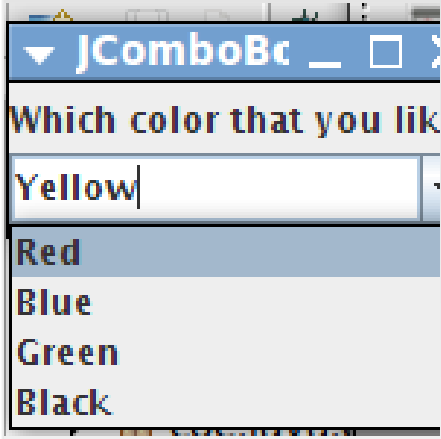
JComboBox

- JComboBoxes have a nifty feature, which is probably not all that useful in practice. You can

  make a JComboBox “editable” by calling its method setEditable(true)
- If you do this, the user can edit the selection by clicking on the JComboBox and typing
- This allows the user to make a selection that is not in the pre-configured list that you provide
- The “Combo” in the name “JComboBox” refers to the fact that it’s a kind of combination of menu and text-input box.)

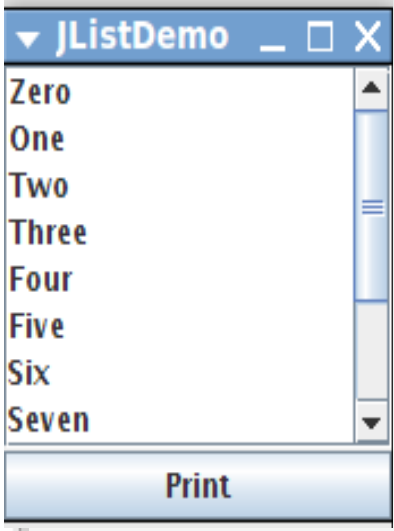
JComboBoxDemo

```
 5 public class JComboBoxDemo {
 6     public static void addComponents(Container cp) {
 7         cp.setLayout(new GridLayout(2,1));
 8         JLabel label = new JLabel("Which color that you li
 9         cp.add(label);
10         JComboBox colorChoice = new JComboBox();
11         colorChoice.addItem("Red");
12         colorChoice.addItem("Blue");
13         colorChoice.addItem("Green");
14         colorChoice.addItem("Black");
15         colorChoice.setEditable(true);
16         cp.add(colorChoice);
17     }
18     public static void main(String[] args) {
19         JFrame frame = new JFrame("JComboBoxDemo");
20         addComponents(frame.getContentPane());
21         frame.pack();
22         frame.setLocation(100, 100);
23         frame.setVisible(true);
24     }
```



JListDemo

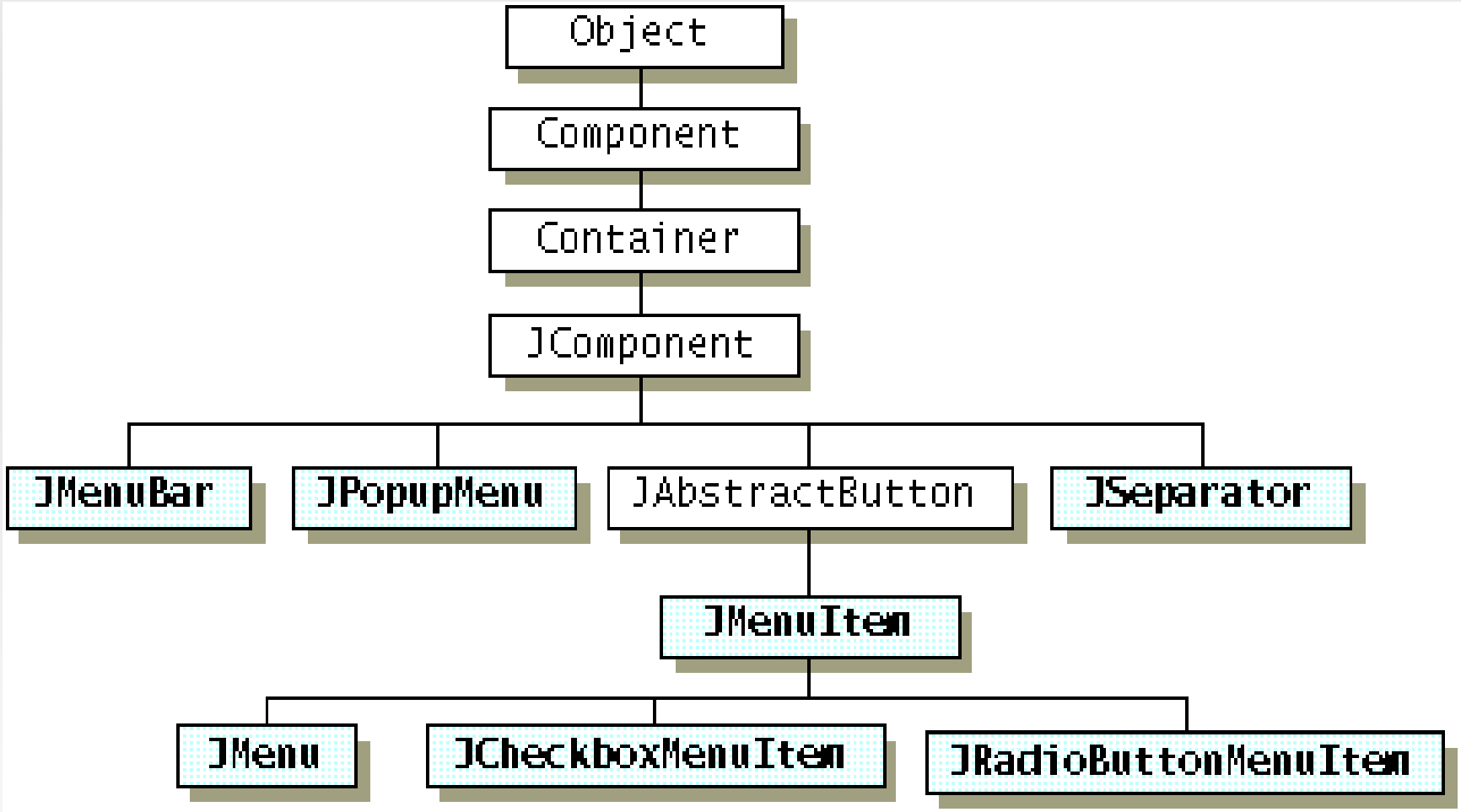
```
3  import java.awt.*;
4  import javax.swing.*;
5  public class JListDemo {
6      public static void addComponents(Container cp) {
7          cp.setLayout(new BorderLayout());
8          String labels[] = { "Zero", "One", "Two", "Three",
9                  "Four", "Five", "Six",
10                   "Seven", "Eight", "Nine", "Ten", "Eleven" };
11         JList jlist = new JList(labels);
12         JScrollPane scrollPane1 = new JScrollPane(jlist);
13         cp.add(scrollPane1, BorderLayout.NORTH);
14         cp.add(new JButton("Print"), BorderLayout.SOUTH);
15     }
16     public static void main(String[] args) {
17         JFrame frame = new JFrame("JListDemo");
18         addComponents(frame.getContentPane());
19         frame.pack();
20         frame.setLocation(100, 100);
21         frame.setVisible(true);
22     }
```



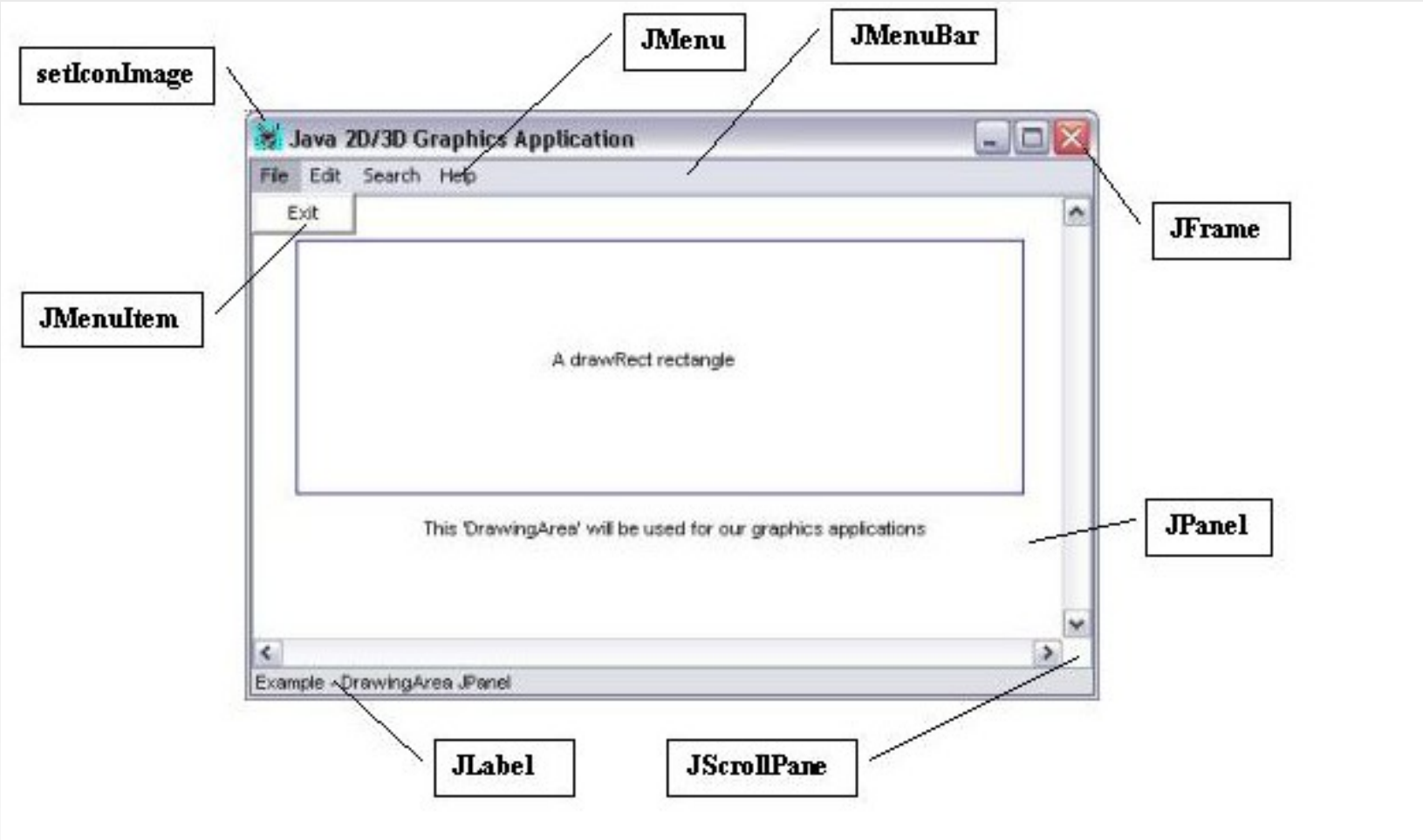
Menus

- Menus are unique in that, by convention, they aren't placed with the other components in the UI
- Instead, a menu usually appears either in a menu bar or as a popup menu
- A menu bar contains one or more menus and has a customary, platform-dependent location — usually along the top of a window

JMenu Hierarchy



JComponent in Action



JMenuBar

- Once a menu has been created, it must be added to a menu bar
- A menu bar is represented by the class JMenuBar and it's just a container for menus
- It has an add(JMenu) method that can be used to add menus to the menu bar
- Example:
  - JMenuBar menuBar = new JMenuBar();
  - JMenu menu = new JMenu(“File”);
  - menuBar.add(menu);

JMenuItem

- The items in a menu are represented by the class JMenuItem
- Menu items are used in almost exactly the same way as buttons
- Constructors of JMenuItem
  - JMenuItem menuItem = new JMenuItem("New");
  - menu.add(menuItem);

JMenuItem with Icon

- We can use the constructor that accepts the object in class Icon
- How to create an icon by loading an image file
- There are many ways to load resource files in java app
- The recommended way to do is
  - Class.getResource("resource_name") or Class.getResourceAsStream("resource_name") then you will get URL and InputStream respectively.

Using function getResource()

- If we want to use it in an instance method, we can use “this”
- Example
  - this.getClass().getResource("/resource/buttons1.png")
  - this.getClass().
    getResourceAsStream("/resource/TX_Jello2.ttf")
- If we want to use it in a static method, we can use the static member “class” of the class name
- Example
  JMenuLookDemo.**class**.getResource("/images/open.png")

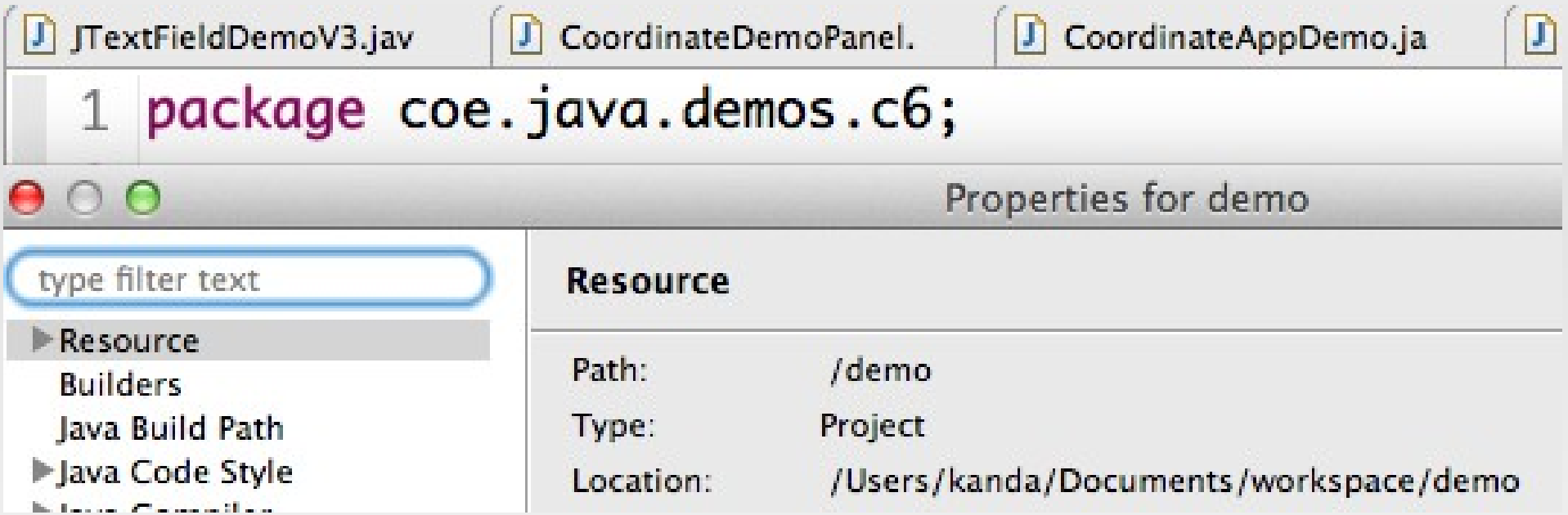
  - In this example, JMenuLookDemo is the class name

Example of Image Location

- Suppose that “Sample” class in in package “coe.acp” and is located at and you use eclipse
- You need to go to the project folder and then go to the “bin” directory of that folder
- Then put the image files there or create the folder that contains the images there
- Example:
  - The class location is at /home/kanda/workspace/labs/bin/coe/acp/Sample
  - The image file is at

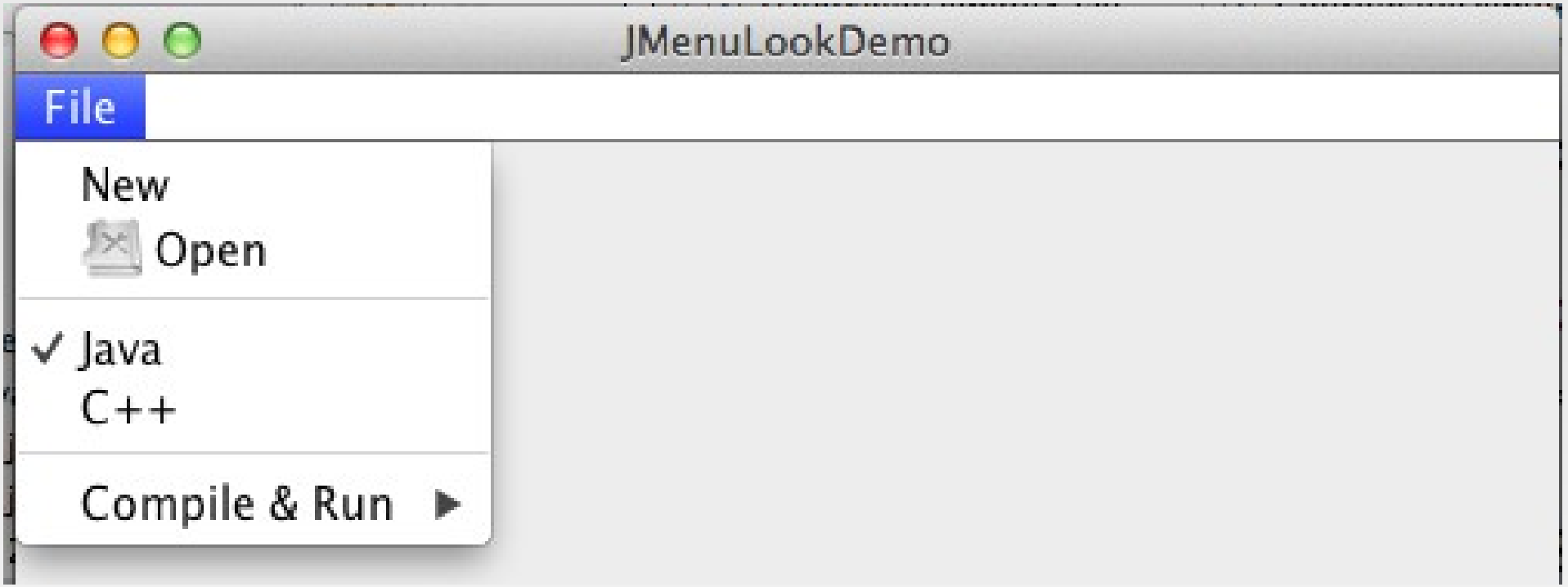
    /home/kanda/workspace/labs/bin/images/open-icon.png

Sample Project and Images Location




```
Kandas-MacBook-Air:images kanda$ pwd
/Users/kanda/Documents/workspace/demo/bin/images
Kandas-MacBook-Air:images kanda$ ls
open.png
```

Sample Code to Display the MenuItem with Icon




```
25          ImageIcon icon =new
26                  ImageIcon(JMenuLookDemo.class.getResource("/images/open.png"));
27          menuItem = new JMenuItem("Open", icon);
28          menu.add(menuItem);
```

Using Submenus

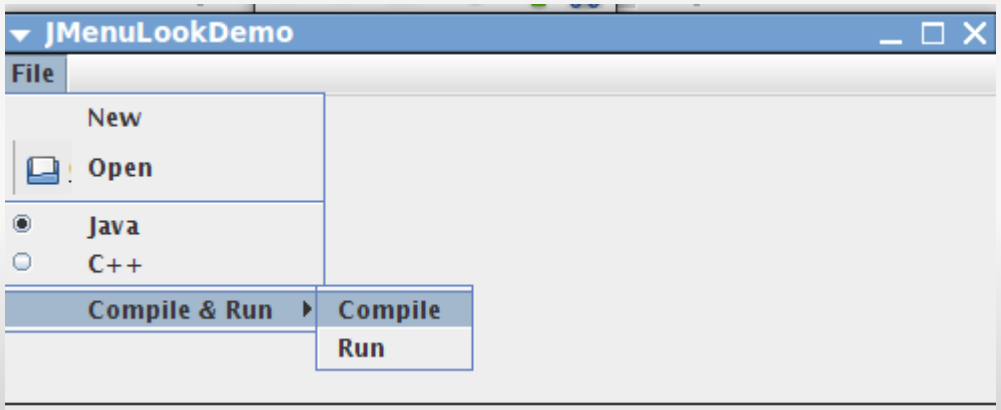
- You can add submenus into menu items
- First, you need to create a JMenu object that contains many JMenuItem objects
- Then, add the JMenuObject as a JMenuItem of the main menu
- Example:
  - JMenu submenu = new JMenu("Compile & Run");
  - submenu.add(new JMenuItem("Compile"));
  - submenu.add(new JMenuItem("Run"));
  - menu.add(submenu)

Using Menu Bar in Window

- The final step in using menus is to use the menu bar in a JApplet or JFrame
- We have already seen that an applet or frame has a "content pane."
- The menu bar is another component of the applet or frame, not contained inside the content pane
- Example:
  - frame.setJMenuBar(menuBar);

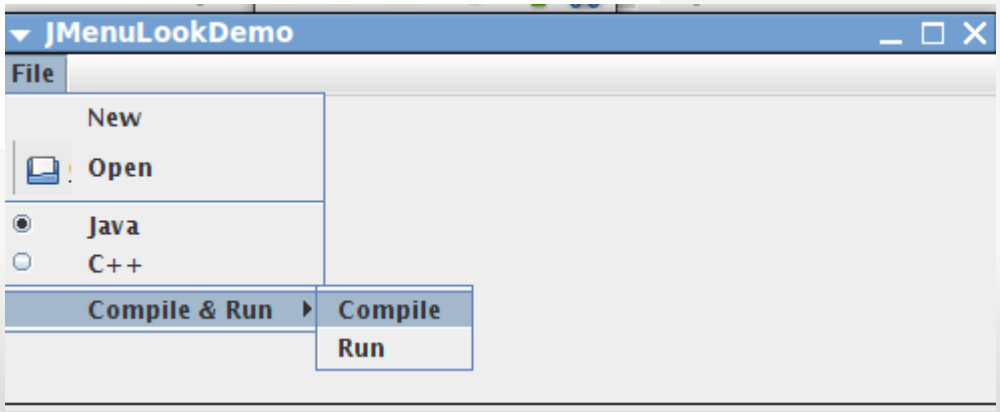
MenuLookDemo (1/2)

```
 3 import javax.swing.*;
 4 public class JMenuLookDemo {
 5     public static void main(String[] args) {
 6         JFrame frame = new JFrame("JMenuLookDemo");
 7
 8         JMenuBar menuBar = new JMenuBar();
 9         JMenu menu = new JMenu("File");
10         menuBar.add(menu);
11
12         JMenuItem menuItem = new JMenuItem("New");
13         menu.add(menuItem);
14
15         menuItem = new JMenuItem("Open", new ImageIcon(
16                 "images/openIcon.png"));
17         menu.add(menuItem);
```



MenuLookDemo (2/2)

```
19          // a group of radio button menu items
20          menu.addSeparator();
21          ButtonGroup group = new ButtonGroup();
22          JRadioButtonMenuItem rb1 =
23              new JRadioButtonMenuItem("Java");
24          rb1.setSelected(true);
25          JRadioButtonMenuItem rb2 =
26              new JRadioButtonMenuItem("C++");
27          group.add(rb1);
28          group.add(rb2);
29          menu.add(rb1);
30          menu.add(rb2);
31          // a submenu
32          menu.addSeparator();
33          JMenu submenu = new JMenu("Compile & Run");
34          submenu.add(new JMenuItem("Compile"));
35          submenu.add(new JMenuItem("Run"));
36          menu.add(submenu);
37
38          frame.setJMenuBar(menuBar);
```



How to position the window at the center of the screen

```
// Get the size of the screen
Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();

// Determine the new location of the window
int w = window.getSize().width;
int h = window.getSize().height;
int x = (dim.width-w)/2;
int y = (dim.height-h)/2;

// Move the window
window.setLocation(x, y);
```

References

- David J. Eck, ”Introduction to Programming Using Java”, Version 5.0, December 2006 http://math.hws.edu/javanotes/
- http://www.computing.northampton.ac.uk/~gary/csy3019/ir