Programming in Java

Building Java GUIs Using the Swing API

Hua Huang, Ph.D.

Spring 2019

Objectives

- Describe the **JFC Swing** technology
- Define Swing
- Identify the **Swing packages**
- Describe the GUI building blocks: **containers**, **components**, and **layout managers**
- Examine **top-level**, **general-purpose**, and **special-purpose properties** of **container**
- Examine **components**
- Examine **layout managers**
- Describe the Swing single-threaded model
- **Build a GUI** using Swing components

What Are the Java Foundation Classes(JFC)?

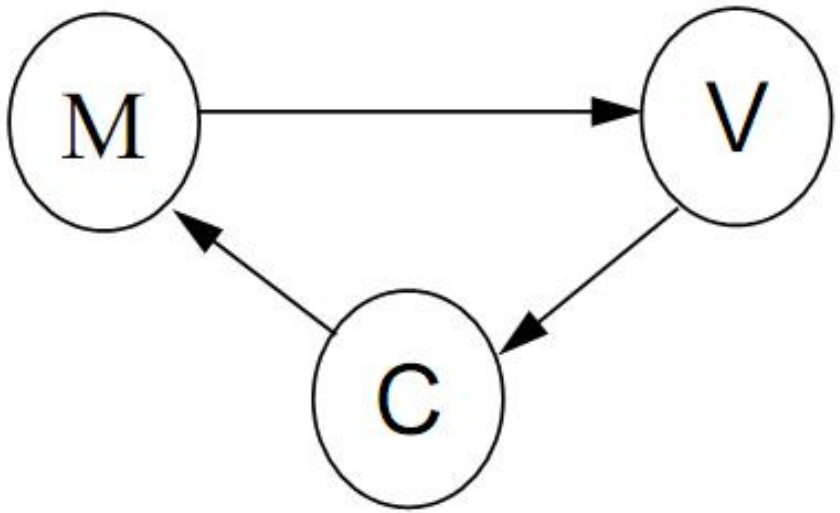
- Java Foundation Classes are a set of Graphical User Interface (GUI) support packages, including:
  - Abstract Window Toolkit (**AWT**)
  - The Swing component set
  - **2D** graphics
  - **Pluggable look-and-feel**
  - Accessibility
  - Drag-and-drop
  - Internationalization

What Is Swing?

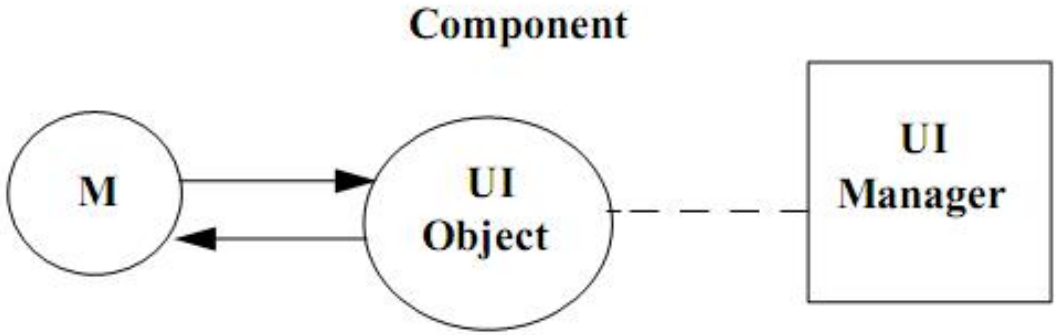
- An **enhanced GUI component set**
- Provides **replacement components** for those in the original **AWT**
- Has special features, such as a pluggable look-and feel

Swing Architecture

- Has its roots in the Model-View-Controller (MVC) architecture



- The Swing components follow Separable Model Architecture



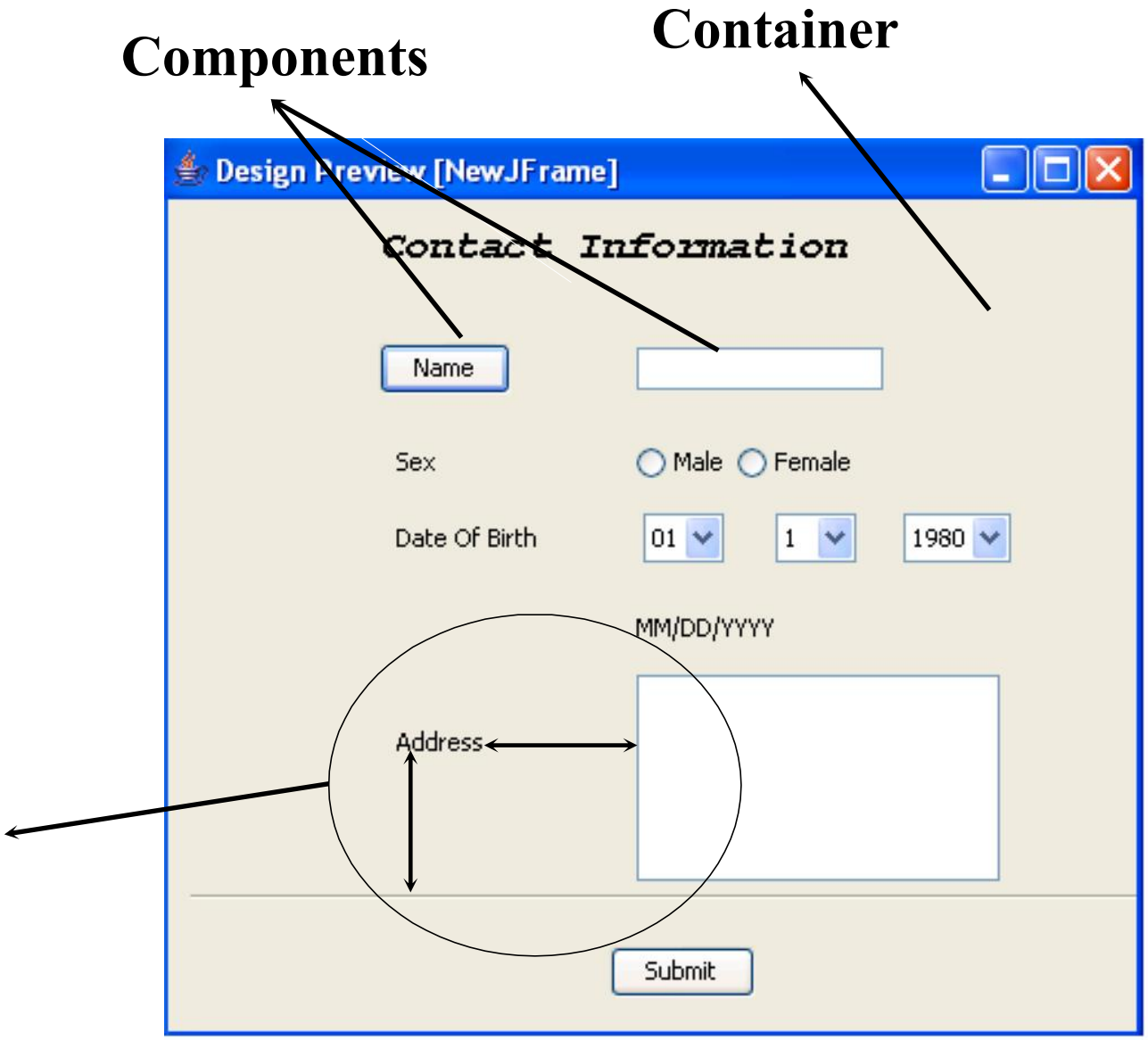
Swing Packages

Package Name	Package Name
javax.swing	javax.swing.**colorchooser**
javax.swing.**border**	javax.swing.**filechooser**
javax.swing.**event**	javax.swing.**table**
javax.swing.**undo**	javax.swing.**tree**
javax.swing.plaf	javax.swing.**text**
javax.swing.plaf.basic	javax.swing.**text.html**
javax.swing.plaf.metal	javax.swing.**text.html.parser**
javax.swing.plaf.multi	javax.swing.**text.rtf**
javax.swing.plaf.synth	

Examining the Composition of a Java Technology GUI

- A Swing API-based GUI is composed of the following elements:
  - **Containers** – Are on **top** of the GUI containment hierarchy.
  - **Components** – Contain all the **GUI components** that are derived from the `JComponent` class.
  - **Layout Managers** – Are responsible for **laying out components** in a container.

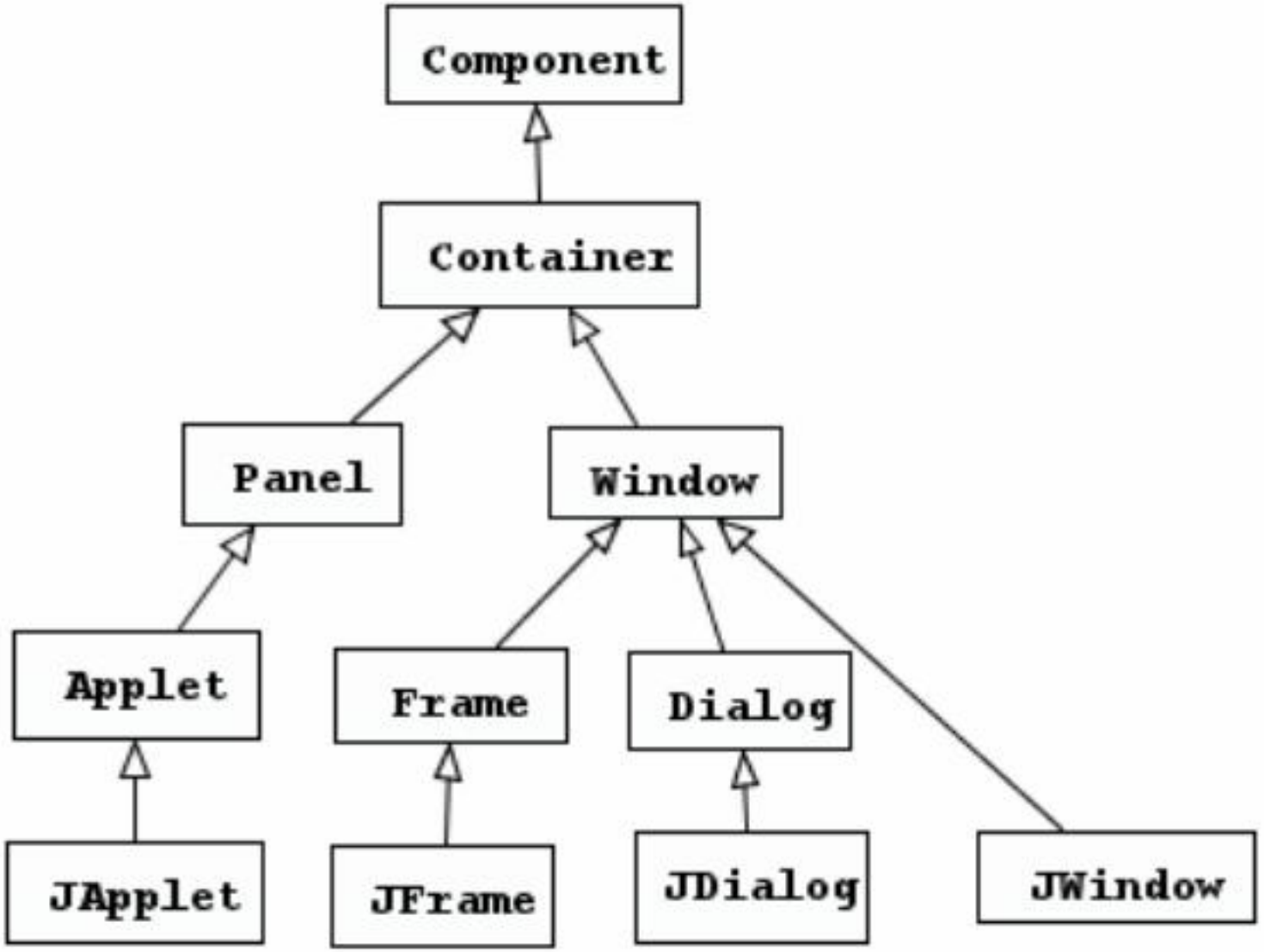
Examining the Composition of a Java Technology GUI(Cont.)



Swing Containers

- Swing containers can be classified into **three** main **categories**:
  - **Top-level** containers:
    - `JFrame`, `JWindow`, and `Jdialog`, (`Japplet`)
  - **General-purpose** containers:
    - `JPanel`, `JScrollPane`, `JToolBar`, `JSplitPane`, and `JTabbedPane`
  - **Special-purpose** containers:
    - `JInternalFrame` and `JLayeredPane`

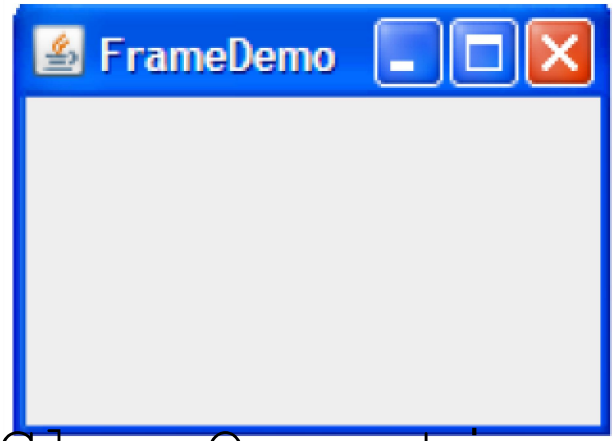
Top-Level Containers



Top-Level Containers(Cont.)

A Frame is the **basic window** used in most GUI applications.

- Has a **border** and a **title**.
- **Other components can be added** to the frame.
- **Menus** can be added to the frame.
- The `javax.swing.JFrame` class is used to create frames.
- The `JFrame` container permits you to set one of four reactions for the **Close Window menu button**.
  - DO_NOTHING_ON_CLOSE
  - HIDE_ON_CLOSE
  - DISPOSE_ON_CLOSE
  - EXIT_ON_CLOSE
- Set the option by invoking the `setDefaultCloseOperation` method on the `JFrame` instance.



Top-Level Containers(Cont.)

`JDialog` is used to create a dialog window.

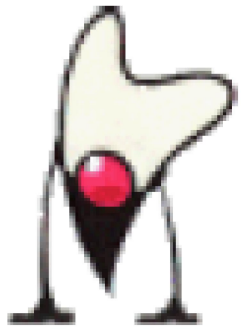
- Has different versions of constructors for defining dialogs.
- Are **dependent on the frames**.
- Can be used to **take input** from the user and **confirm** any critical actions.
- Can be used to display warnings, errors, questions, and information to the user.

Top-Level Containers(Cont.)

- The `JWindow` container is similar to `JFrame` but it does not have a **border** or **title** bar.
- There are **no window management services**.

Top-Level Containers(Cont.)

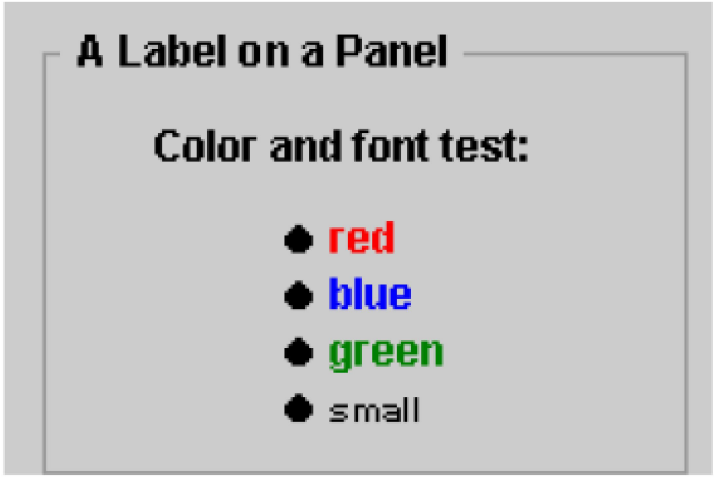
- The `JApplet` container is used to create a UI that is run in a web browser.
- `JApplets` are usually **embedded in a web page** and can be used to **run animations**.
- Other **components** and **menus** can be added to this container.
- **Draw** in an applet is allowed.



General-purpose containers

Panels are containers into which several components can be added. They also provide a surface to draw on.

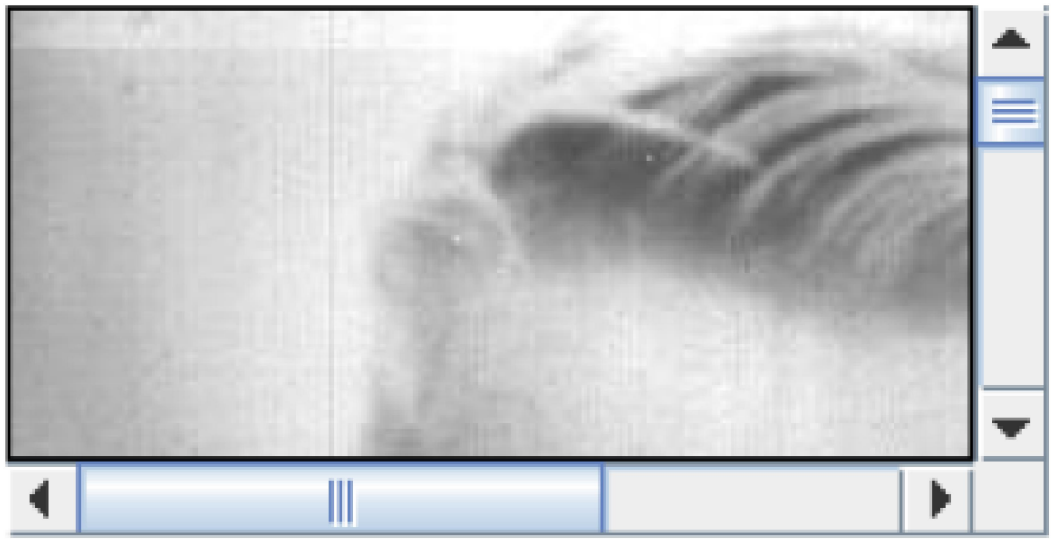
- **Not top-level containers.**
- Should be contained in top-level containers.
- The `javax.swing.JPanel` class is used to create panels. It extends `Jcomponent` and not `java.awt.Panel`.



General-purpose containers(Cont.)

Scrollpanes are very handy when the amount of space is limited.

- Used to display large components or images.
- Have two scrollbars, a row header, and a column header.
- The `javax.swing.JScrollpane` class is used to create scrollpanes.



General-purpose containers(Cont.)

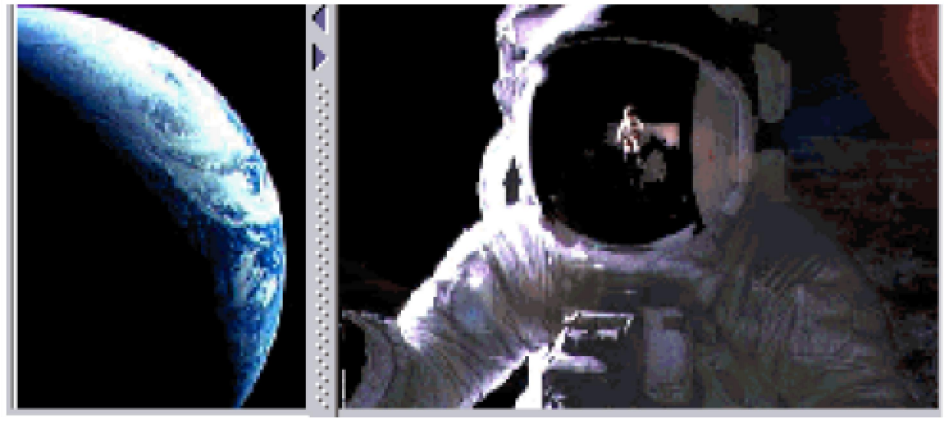
Toolbars are a group of buttons with icons for easily accessing the frequently used functions.

- Can be considered as shortcuts to the actions in menus.
- The `javax.swing.JToolbar` class is used to create toolbars.

General-purpose containers(Cont.)

Splitpanes display two or more components separated by a divider.

- Components can be displayed side by side or one over the other.
- By dragging the divider the amount of space for each component can be adjusted.
- The `javax.swing.JSplitPane` class is used to create splitpanes.



General-purpose containers(Cont.)

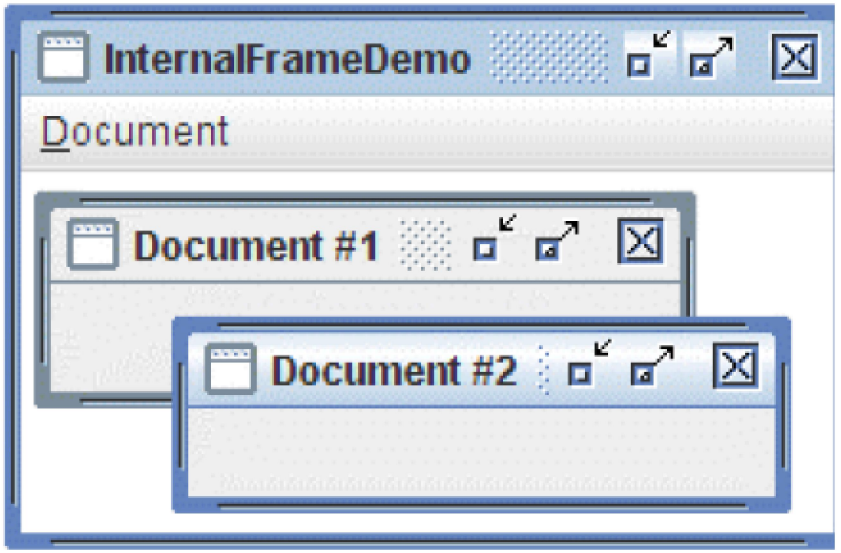
TabbedPanes are also useful when the space is limited.

- Several tabs share the same space. At any one time only one tab is visible.
- To be displayed, a tab needs to be selected by the user.
- The `javax.swing.JTabbedPane` class is used to create tabbedpanes.

Special-purpose containers

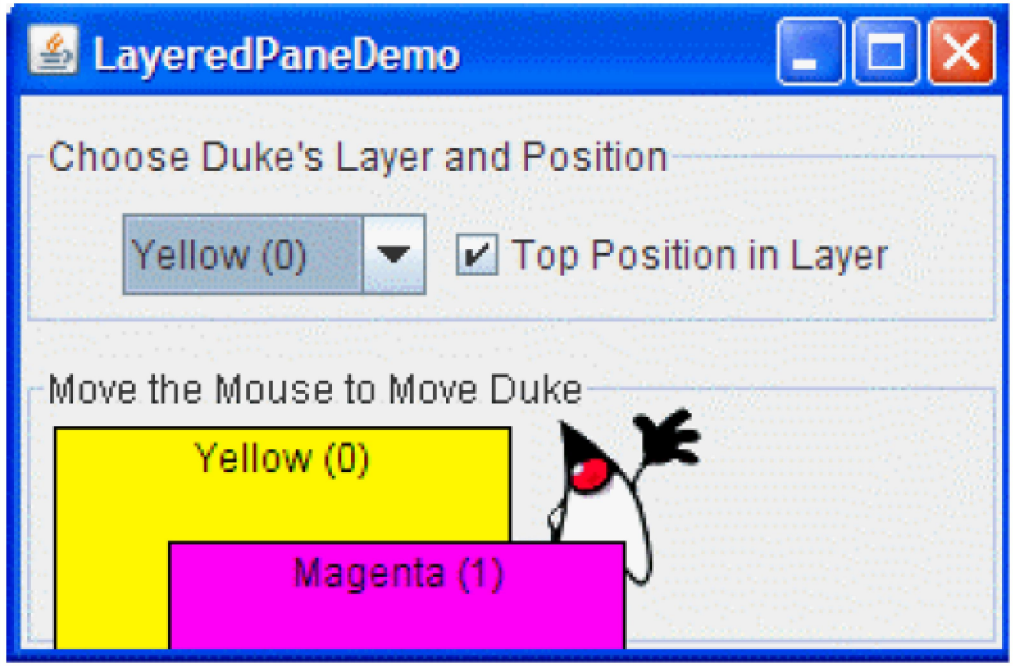
Internal frames are **non** top-level containers.

- Features similar to `JFrames` such as dragging, resizing, iconifying, and maximizing.
- Class `javax.swing.JInternalFrame` are created using internal frames.
- As with regular frames, components can be added to a `JInternalFrame`.



Special-purpose containers

- **Layered panes** allow the addition of components at required depths. The depth is specified as an integer value.
- The `javax.swing.JLayeredPane` class is used to create layered panes. Standard layers defined by this class:
  - DEFAULT_LAYER(bottommost),
  - PALETTE_LAYER,
  - MODAL_LAYER,
  - POPUP_LAYER,
  - DRAG_LAYER(topmost).



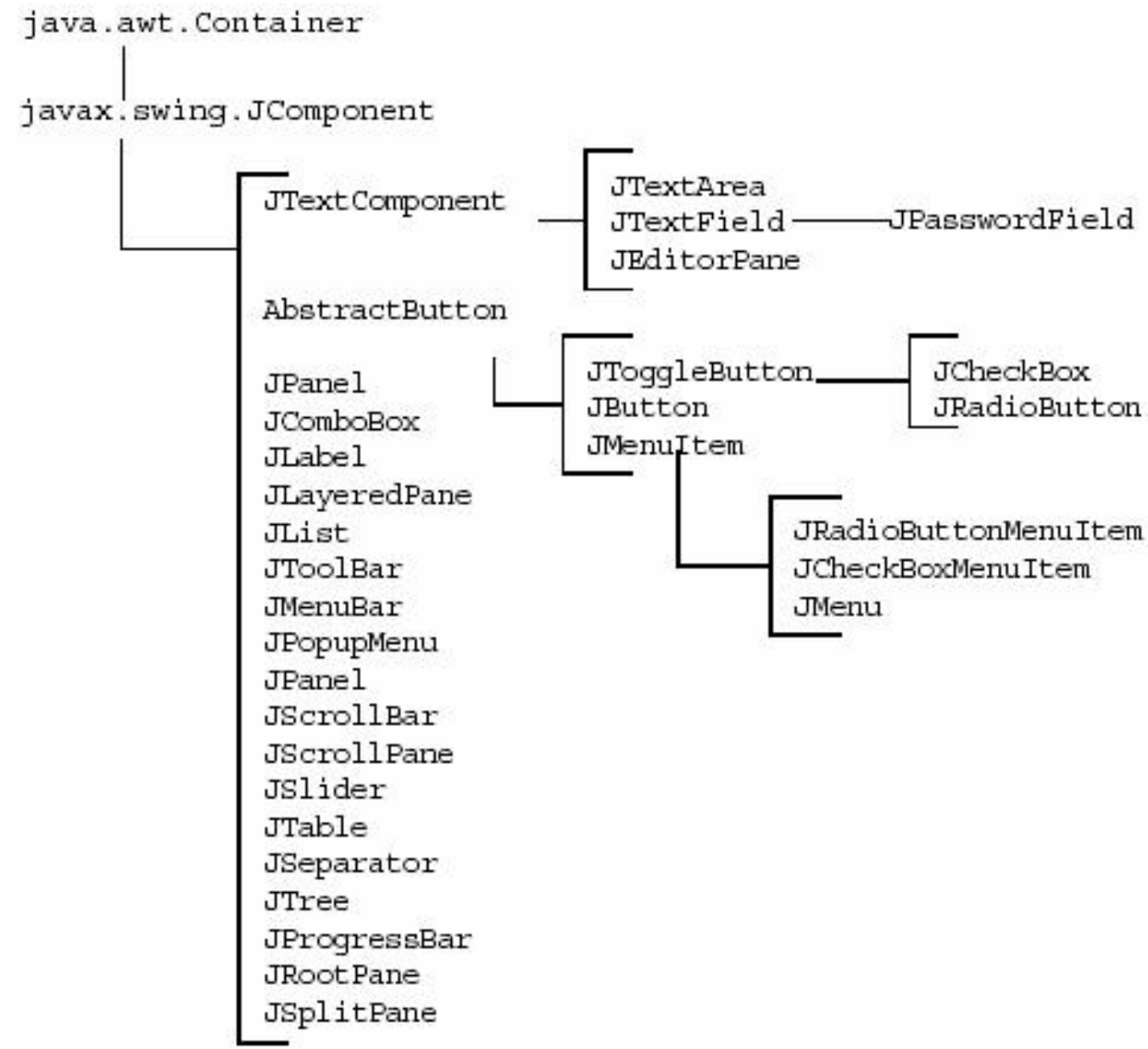
Rules for using container classes

- GUI components will only be **displayed** when they are in a containment hierarchy.
- A GUI component object instance **can appear only once** in a containment tree.
- Each top-level container has a **content pane** that contains (directly or indirectly) the visible components in that top level container's GUI.
- **Menu bar** can be optionally added to a top-level container.

Swing Components

- Swing components can be broadly classified as:
  - **Buttons**
  - **Text components**
  - **Uneditable** information display components
  - **Menus**
  - **Formatted** display components
  - Other basic controls

Swing Component Hierarchy

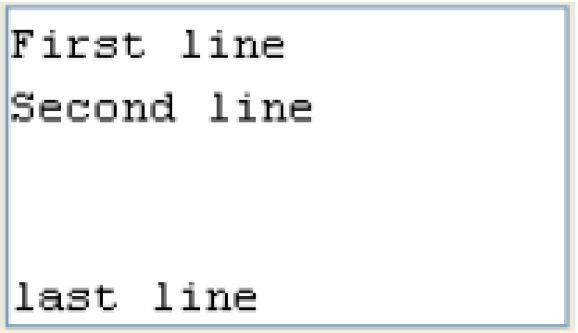


Text Components

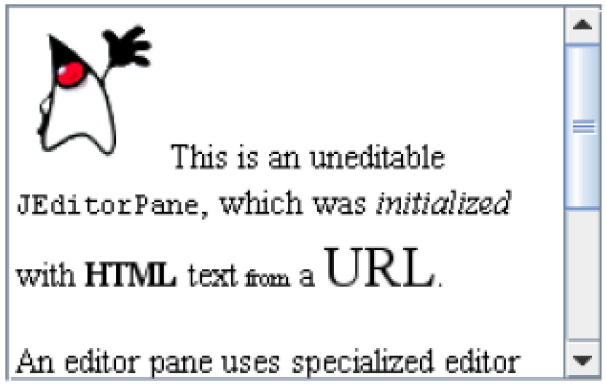
- Swing text components can be broadly divided into three categories.

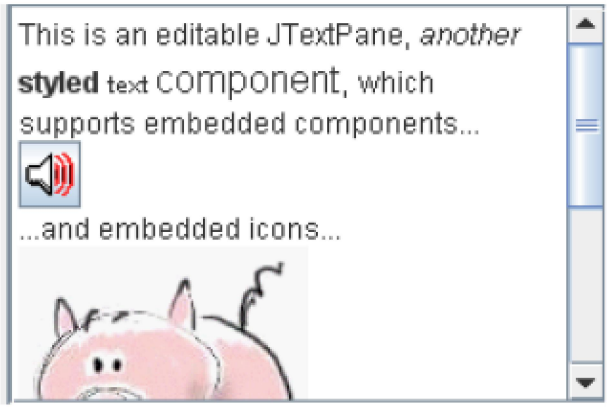
- Text controls – **`JTextField`**, **`JPasswordField`**(for user input)

- **Plain** text areas – **`JTextArea`**(displays text in plain text, also for multi-line user input



- **Styled** text areas – **`JEditorPane`**, **`JTextPane`** (displays formatted text)



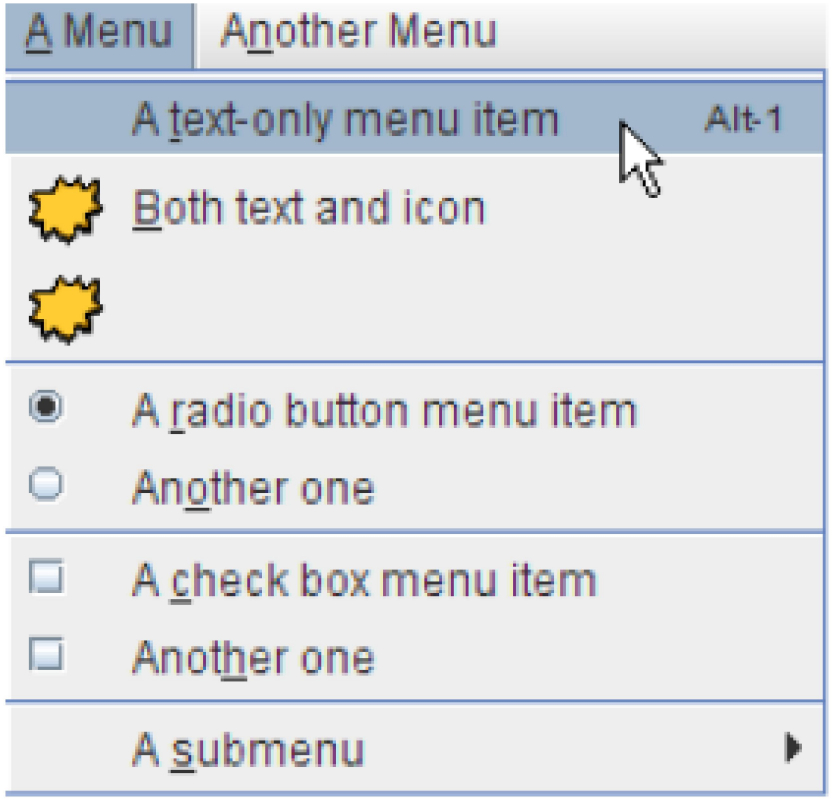


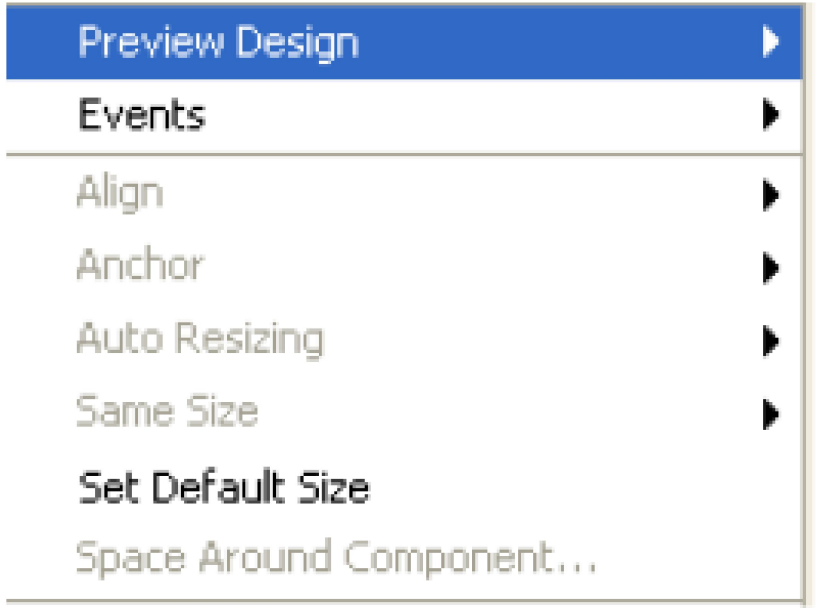
Uneditable Information Display Components

- Component
  - **Labels**: `javax.swing.JLabel`, can also be used to display images.
  - **ToolTip**: `javax.swing.JToolTip`, `setToolTipText`(Jcomponent) method can be used to set the string to be displayed.
  - `JProgressBar`: helpful in indicating the progress of any long-running tasks.

Uneditable Information Display Components(Cont.)

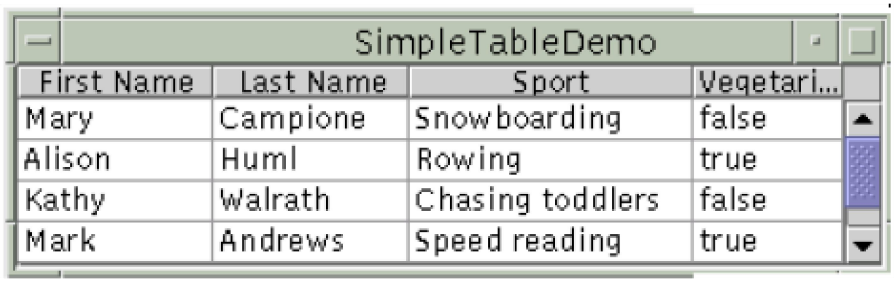
- Menu Components
  - menu bar
    - `javax.swing.JMenu`,
    - `javax.swing.JMenuItem`
  - popup menu
    - `javax.swing.JPopupMenu`

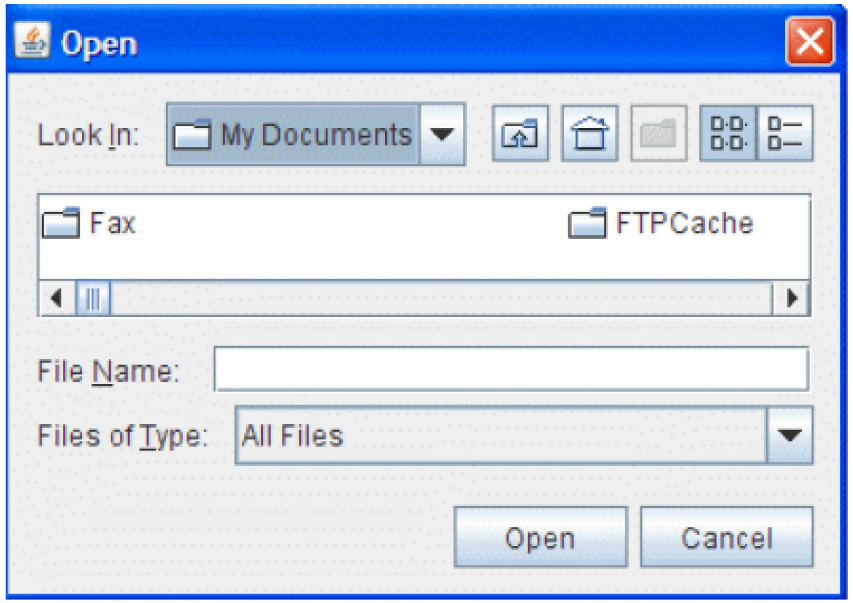


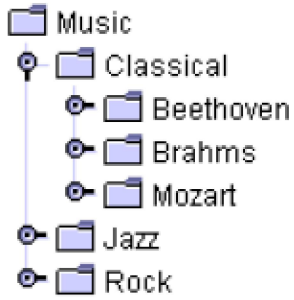


Formatted Display Components

- Tables: **`javax.swing.JTable`**
  - JTable does **not store the data**, but only **displays the data from the table model**.
- Tree: **`Jtree`**
  - Presents the data in the **`TreeModel`** instance.
  - Data can also be provided as a collection of elements such as a **`Hashtable`** or a **`Vector`**.
- **`FileChooser`**
- **`ColorChooser`**







Other Basic Controls

- ComboBox: **`JcomboBox`**, Can be **Editable** and **UnEditable**
- Lists: **`Jlist`**, Can create its own **ListModel** if needed by putting the data items into either a **Vector** or an **array** of Objects
- Spinners: **`Jspinner`**, Has three subcomponents: an up arrow, a down arrow, and an editor
- Slider: have a finite range.

Swing Component Properties

- Common component properties:
  - All the Swing components share some **common properties** because they all extend **`JComponent`**.
- Component-specific properties:
  - Each component defines more **specific** properties.

Common Component Properties

Property	Methods
Border	`Border getBorder()` `void setBorder(Border b)`
Background and foreground color	`void setBackground(Color bg)` `void setForeground(Color bg)`
Font	`void setFont(Font f)`
Opaque	`void setOpaque(boolean isOpaque)`
Maximum and minimum size	`void setMaximumSize(Dimension d)` `void setMinimumSize(Dimension d)`
Alignment	`void setAlignmentX(float ax)` `void setAlignmentY(float ay)`
Preferred size	`void setPreferredSize(Dimension ps)`

Component-Specific Properties

- The following shows properties specific to **JComboBox**.

Property	Methods
Maximum row count	`void setMaximumRowCount(int count)`
Model	`void setModal(ComboBoxModel cbm)`
Selected index	`int getSelectedIndex()`
Selected Item	`Object getSelectedItem()`
Item count	`int getItemCount()`
Renderer	`void setRenderer(ListCellRenderer ar)`
Editable	`void setEditable(boolean flag)`

Layout Managers

Determines the **size** and **position** of the components **within** a container. (Absolute positioning)

- Handle problems caused by:
  - GUI **resizing** by user
  - **Operating system differences in fonts**
  - Locale-specific **text layout** requirements
- Layout manager classes:
  - **`BorderLayout`**
  - **`FlowLayout`**
  - **`BoxLayout`**
  - **`CardLayout`**
  - **`GridLayout`**
  - **`GridBagLayout`**
  - ……

The `BorderLayout` Manager

- The **`BorderLayout`** manager places components in **top**, **bottom**, **left**, **right** and **center** locations.
- **`BorderLayout`** is the default layout for **`JFrame`**, **`JDialog`**, and **`JApplet`**.

BorderLayout Example

```
import java.awt.*;
import javax.swing.*;
public class BorderExample {
  private JFrame f;
  private JButton bn, bs, bw, be, bc;
  public BorderExample() {
    f = new JFrame("Border Layout"); bn = new JButton("Button 1");
    bc = new JButton("Button 2"); bw = new JButton("Button 3");
    bs = new JButton("Button 4"); be = new JButton("Button 5");
  }
  public void launchFrame() {
    f.add(bn, BorderLayout.NORTH); f.add(bs, BorderLayout.SOUTH);
    f.add(bw, BorderLayout.WEST); f.add(be, BorderLayout.EAST);
    f.add(bc, BorderLayout.CENTER); f.setSize(400,200);
    f.setVisible(true);
  }
  public static void main(String args[]) {
    BorderExample guiWindow2 = new BorderExample();
    guiWindow2.launchFrame();
  }
}
```

The `FlowLayout` Manager

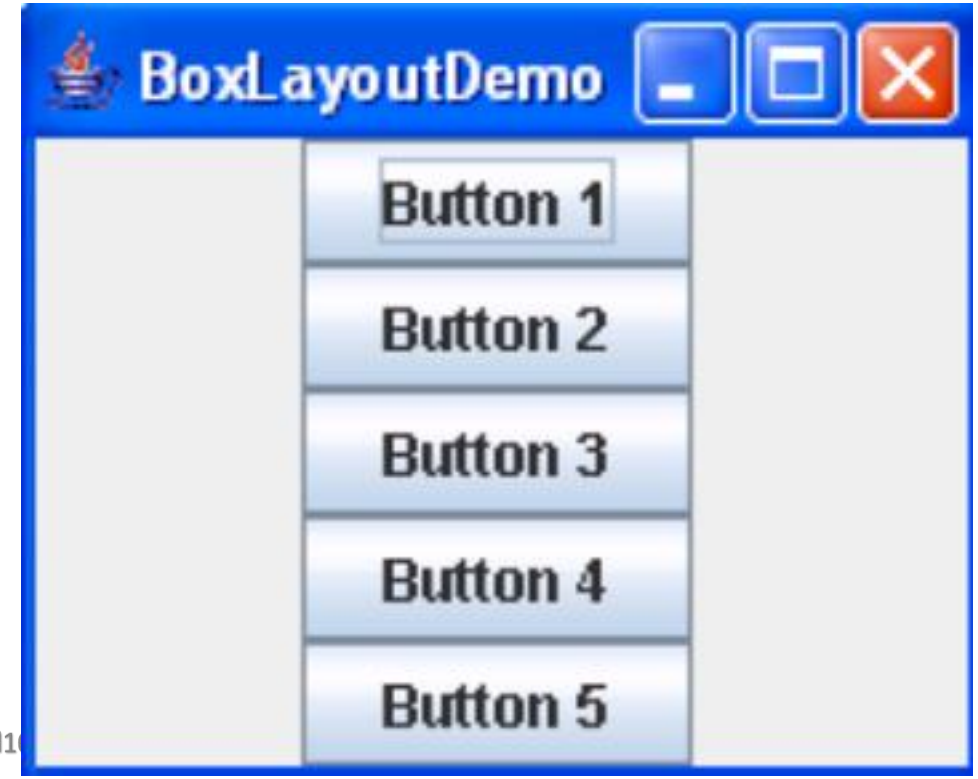
- The **`FlowLayout`** manager places components **in a row**, and if the row fills, components are placed in the next row.
- By default, it arranges the components from **LEFT_TO_RIGHT**. It can be changed using the **`ComponentOrientation`** property RIGHT_TO_LEFT

FlowLayout Example

```
import javax.swing.*;
import java.awt.*;
public class LayoutExample {
  private JFrame f; private JButton b1;
  private JButton b2; private JButton b3;
  private JButton b4; private JButton b5;
  public LayoutExample() {
    f = new JFrame("GUI example"); b1 = new JButton("Button 1");
    b2 = new JButton("Button 2"); b3 = new JButton("Button 3");
    b4 = new JButton("Button 4"); b5 = new JButton("Button 5");
  }
  public void launchFrame() {
    f.setLayout(new FlowLayout());
    f.add(b1); f.add(b2);
    f.add(b3); f.add(b4);
    f.add(b5); f.pack(); f.setVisible(true);
  }
  public static void main(String args[]){
    LayoutExample guiWindow = new LayoutExample();
    guiWindow.launchFrame();
  }
} // end of LayoutExample class
```

The `BoxLayout` Manager

- The **`BoxLayout`** manager arranges the components either vertically or horizontally.
- The **`BoxLayout`** constructor takes a parameter called **axis**, where the **direction to align the components** should be specified. Values can be:
  - X_AXIS
  - Y_AXIS
  - LINE_AXIS
  - PAGE_AXIS



The `CardLayout` Manager

- The **`CardLayout`** manager places the components in different cards. Cards are usually controlled by a combo box.

The `GridLayout` Manager

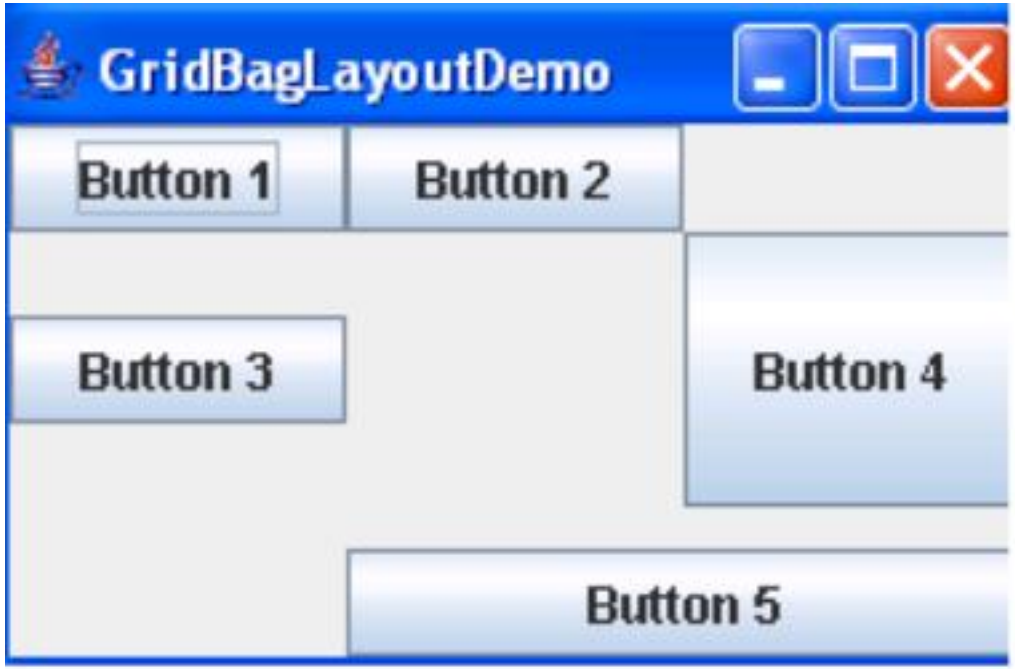
- The **`GridLayout`** manager places components in **rows and columns** in the form of a grid.

GridLayout Example

```
import java.awt.*;
import javax.swing.*;
public class GridExample {
  private JFrame f;
  private JButton b1, b2, b3, b4, b5;
  public GridExample() {
    f = new JFrame("Grid Example");    b1 = new JButton("Button 1");
    b2 = new JButton("Button 2");     b3 = new JButton("Button 3");
    b4 = new JButton("Button 4");     b5 = new JButton("Button 5");
  }
  public void launchFrame() {
    f.setLayout (new GridLayout(3,2));
    f.add(b1);      f.add(b2);
    f.add(b3);      f.add(b4);
    f.add(b5);      f.pack();
    f.setVisible(true);
  }
  public static void main(String args[]) {
    GridExample grid = new GridExample();
    grid.launchFrame();
  }
}
```

The `GridBagLayout` Manager

- The **`GridBagLayout`** manager arranges components in **rows and columns**, similar to a **grid layout**, but provides a wide variety of options for resizing and positioning the components.
- **`GroupLayout`** was added to Java SE version 6. This layout manager was added for use by toolmakers. **Used in the NetBeans IDE GUI builder tool**.



GUI Construction

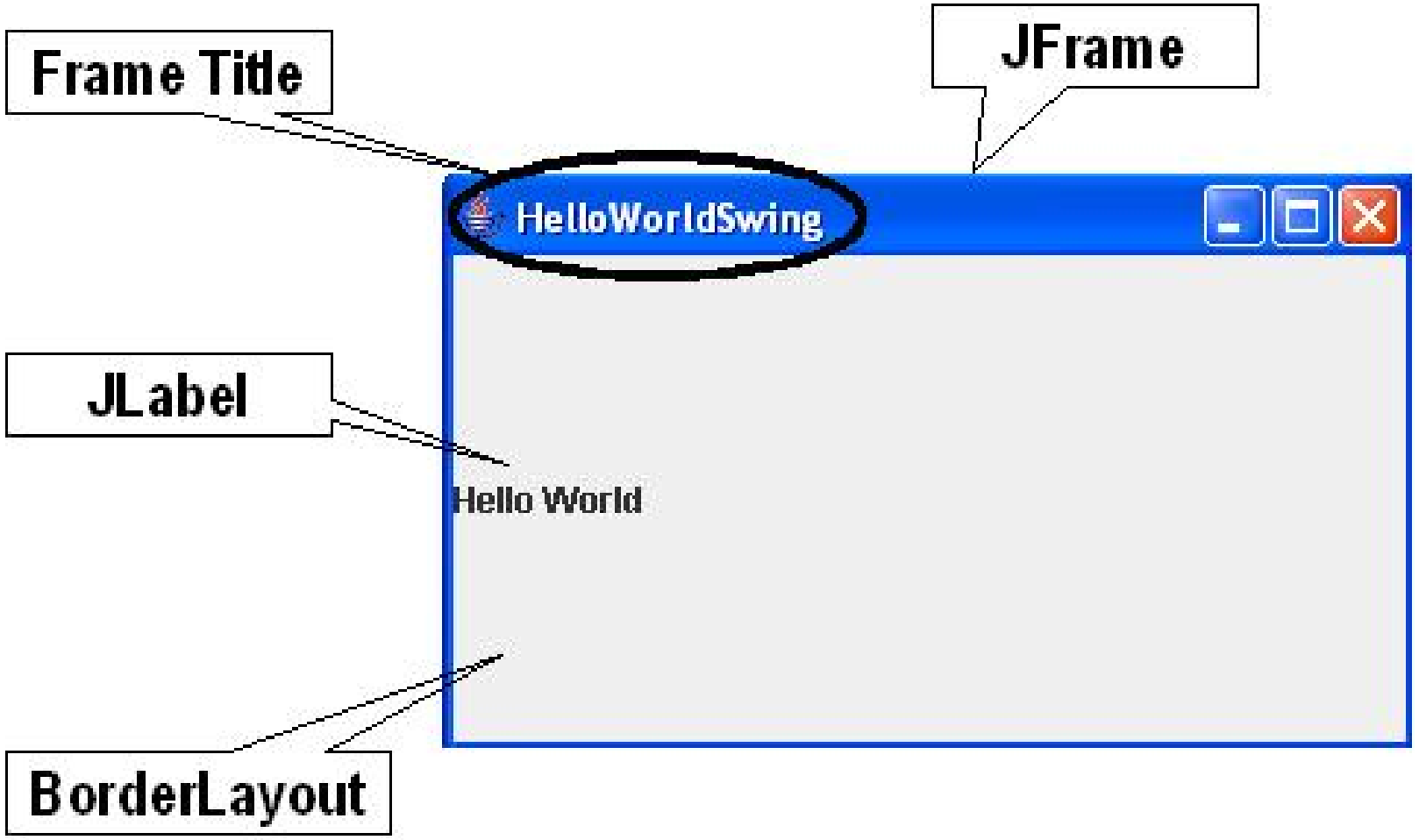
- Programmatic
  - Useful for learning GUI construction. However, it is very laborious for use in production environments.
- GUI builder tool
  - Uses a visual approach to drag-and-drop containers and components to a work area.
  - **Eclipse**: Windows Buider Pro/MyEclipse visual swing designer/Eclipse Visual Editor
  - **Netbeans**: Swing GUI Builder (Matisse)
  - **IntelliJ IDEA**: UI Designer plugin

Programmatic Construction

```
import javax.swing.*;
public class HelloWorldSwing {
  private static void createAndShowGUI() {
    JFrame frame = new JFrame("HelloWorldSwing");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);      //Set up the window.
    JLabel label = new JLabel("Hello World");   // Add Label
    frame.add(label);
    frame.setSize(300,200);
    frame.setVisible(true);     // Display Window
  }
  public static void main(String[] args) {
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
      //Schedule for the event-dispatching thread:
      //creating,showing this app's GUI.
      public void run() {
        createAndShowGUI();
      }
    });
  }
}
```

Programmatic Construction(Cont.)

- The output generated from the program



Key Methods

- Methods for setting up the **`JFrame`** and adding **`JLabel`**:
  - `setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)`
    - Creates the program to exit when the close button is clicked.
    - Four possible ways of handling:

      `DO_NOTHING_ON_CLOSE` `HIDE_ON_CLOSE` `DISPOSE_ON_CLOSE` `EXIT_ON_CLOSE`
  - `pack()`: Inherited from `java.awt.Window`. Helps to resize the window, so that all the components' layout and their preferred sizes are preserved.
  - `setVisible(true)` – Makes the `JFrame` visible.
  - `add(label)` – `JLabel` is added to the content pane not to the `JFrame` directly.

Key Methods(Cont.)

- Tasks involved in displaying the GUI **efficiently**:
  - Executing GUI **application code**, such as rendering
  - Handling **GUI events**
  - Handling **time consuming (background) processes**
- The `SwingUtilities` class:
  - `SwingUtilites.invokeLater(new Runnable())`

Summary

- JFC **Swing** technology
- The **Swing packages**
- GUI building blocks: **containers**, **components**, and **layout managers**
- Examine top-level, general-purpose, and special- purpose properties of **container**
- **Components**
- **Layout managers**
- The Swing single-threaded model
- **Build GUIs** using Swing components

Questions or Comments?