CIS2571 – Intro to Java

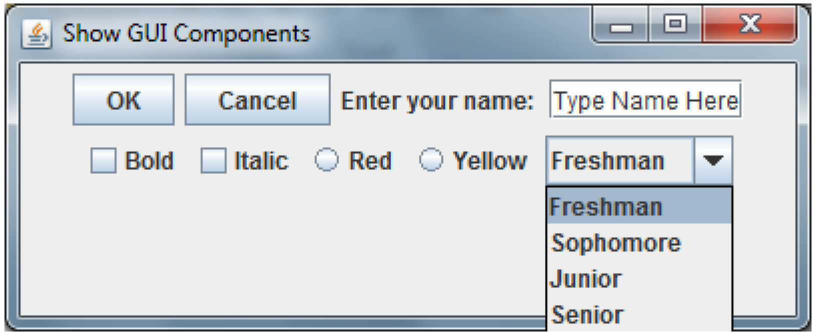
Chapter12 → GUI Basics

Information consolidated from a variety of textbook and online resources

Topic Objectives

- GUI Objects
- Swing vs AWT
- Java GUI API Classifications
  - Component, Container, Helper
- Containers
  - Frames
  - Panels
- Helper
  - Layout Managers
  - Color
  - Font
  - Image Icons
- Component
  - JButton
  - JCheckBox
  - JRadioButton
  - JLabel
  - JTextField

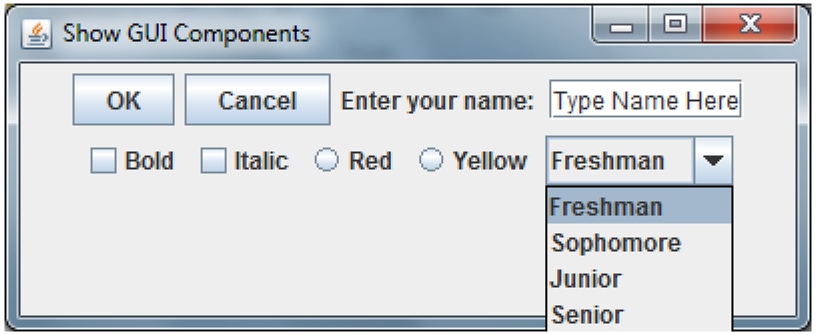
GUI Objects



```
// Create a button with text OK
JButton jbtOK = new JButton("OK");

// Create a button with text Cancel
JButton jbtCancel = new JButton("Cancel");

// Create a label with text "Enter your name: "
JLabel jlblName = new JLabel("Enter your name: ");

// Create a text field with text "Type Name Here"
JTextField jtfName = new JTextField("Type Name Here");
```

See 8.6 GUIComponents.java

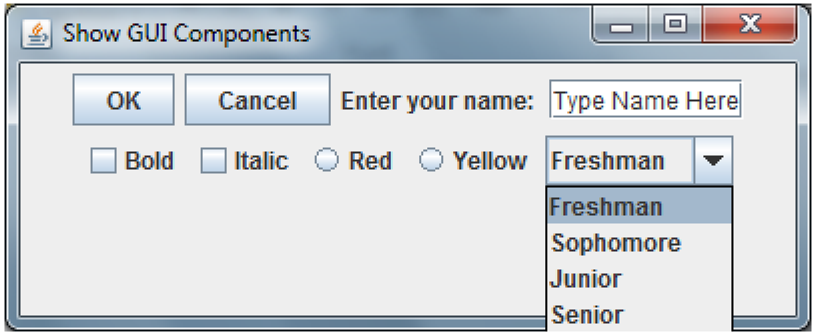
GUI Objects



```
// Create a check box with text bold
JCheckBox jchkBold = new JCheckBox("Bold");

// Create a check box with text italic
JCheckBox jchkItalic = new JCheckBox("Italic");

// Create a radio button with text red
JRadioButton jrbRed = new JRadioButton("Red");

// Create a radio button with text yellow
JRadioButton jrbYellow = new JRadioButton("Yellow");

// Create a combo box with several choices
JComboBox jcboColor = new JComboBox(new String[]{"Freshman", "Sophomore",
"Junior", "Senior"});
```

See 8.6 GUIComponents.java

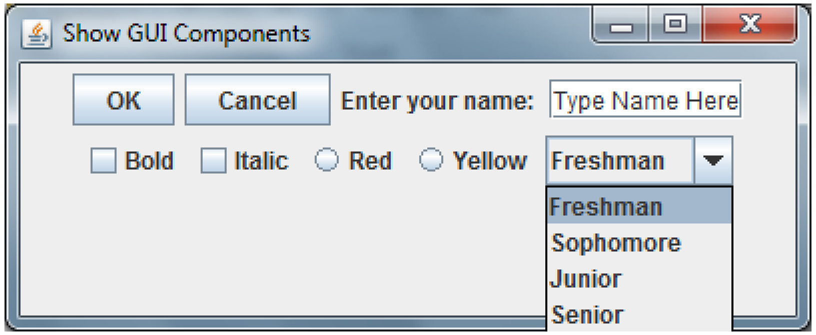
GUI Objects



```
// Create a panel to group components
JPanel panel = new JPanel();
panel.add(jbtOK); // Add the OK button to the panel
panel.add(jbtCancel); // Add the Cancel button to the panel
panel.add(jlblName); // Add the label to the panel
panel.add(jtfName); // Add the text field to the panel
panel.add(jchkBold); // Add the check box to the panel
panel.add(jchkItalic); // Add the check box to the panel
panel.add(jrbRed); // Add the radio button to the panel
panel.add(jrbYellow); // Add the radio button to the panel
panel.add(jcboColor); // Add the combo box to the panel
```

See 8.6 GUIComponents.java

GUI Objects

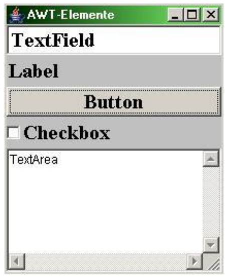


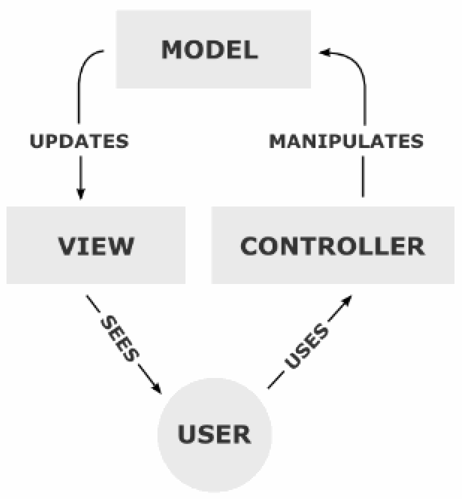
```
JFrame frame = new JFrame(); // Create a frame
frame.add(panel); // Add the panel to the frame
frame.setTitle("Show GUI Components");
frame.setSize(450, 100);
frame.setLocation(200, 100);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setVisible(true);
```

See 8.6 GUIComponents.java

Swing vs. AWT

- Originally, GUI classes bundled into library called Abstract Windows Toolkit (AWT)
  - Thin level of abstraction over native user interface
  - Differing displays on different platforms
    - Heavyweight components
- Swing Architecture part of Java Foundation Classes (JFC)
  - Rooted in the Model-View-Controller software architecture paradigm
    - A **model** that represents the data for the application.
    - The **view** that is the visual representation of that data.
    - A **controller** that takes user input on the view and translates that to changes in the model.

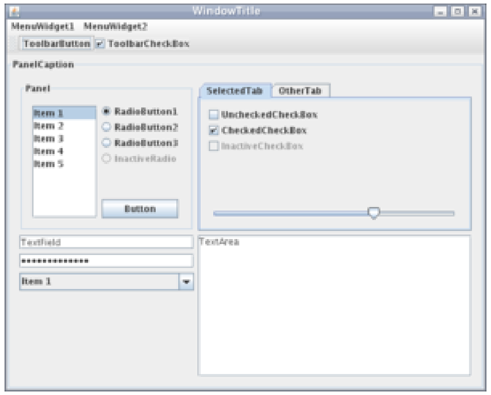




***images taken from wikipedia topics on AWT and Model-View-Controller*

Swing vs. AWT

- Swing is primary Java GUI widget toolkit
  - Emulates look and feel of several platforms
    - Cross platform, or lightweight, components
  - Tutorials for building GUI applications
  - Swing API is *complimentary* extension of AWT rather than replacement
    - Prefixed by "J" to distinguish from their AWT counterparts



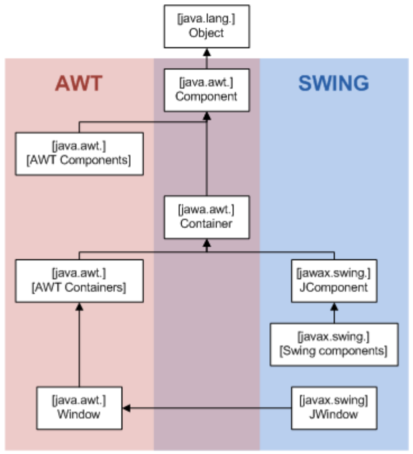
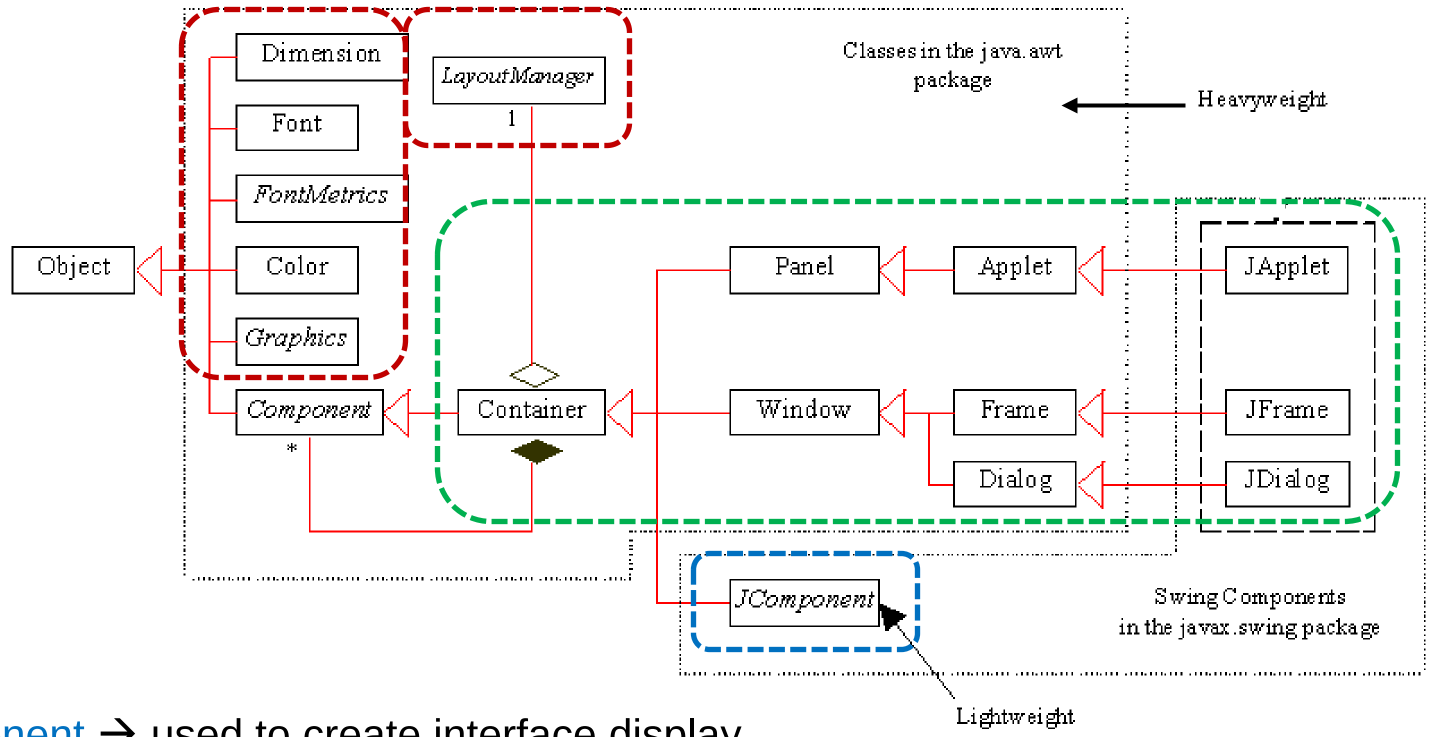

[java.lang.] Object
AWT
SWING
[java.awt.] Component
[java.awt.] [AWT Components]
[java.awt.] Container
[java.awt.] [AWT Containers]
[javax.swing.] JComponent
[javax.swing.] [Swing components]
[java.awt.] Window
[javax.swing] JWindow

***image taken from wikipedia topic on Swing*

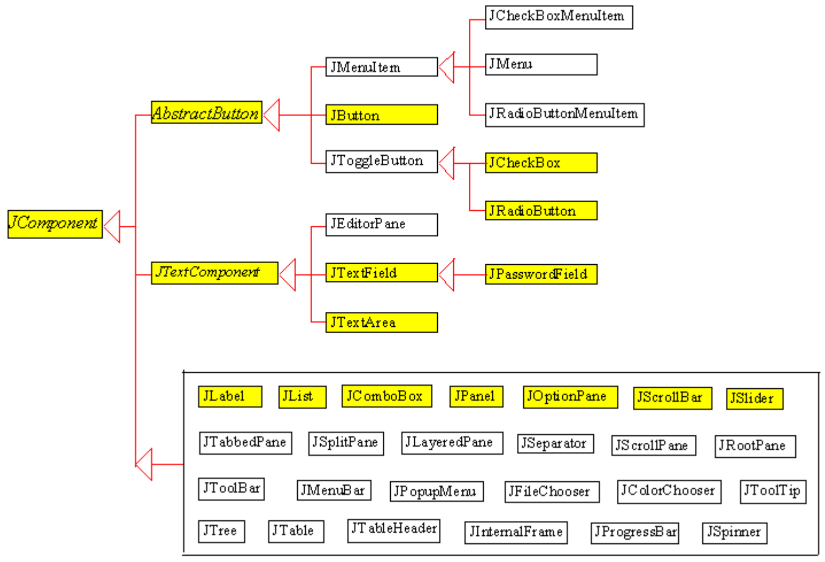
Java GUI API Classifications



Component → used to create interface display
Container → used to contain other components
Helper → used to support GUI components

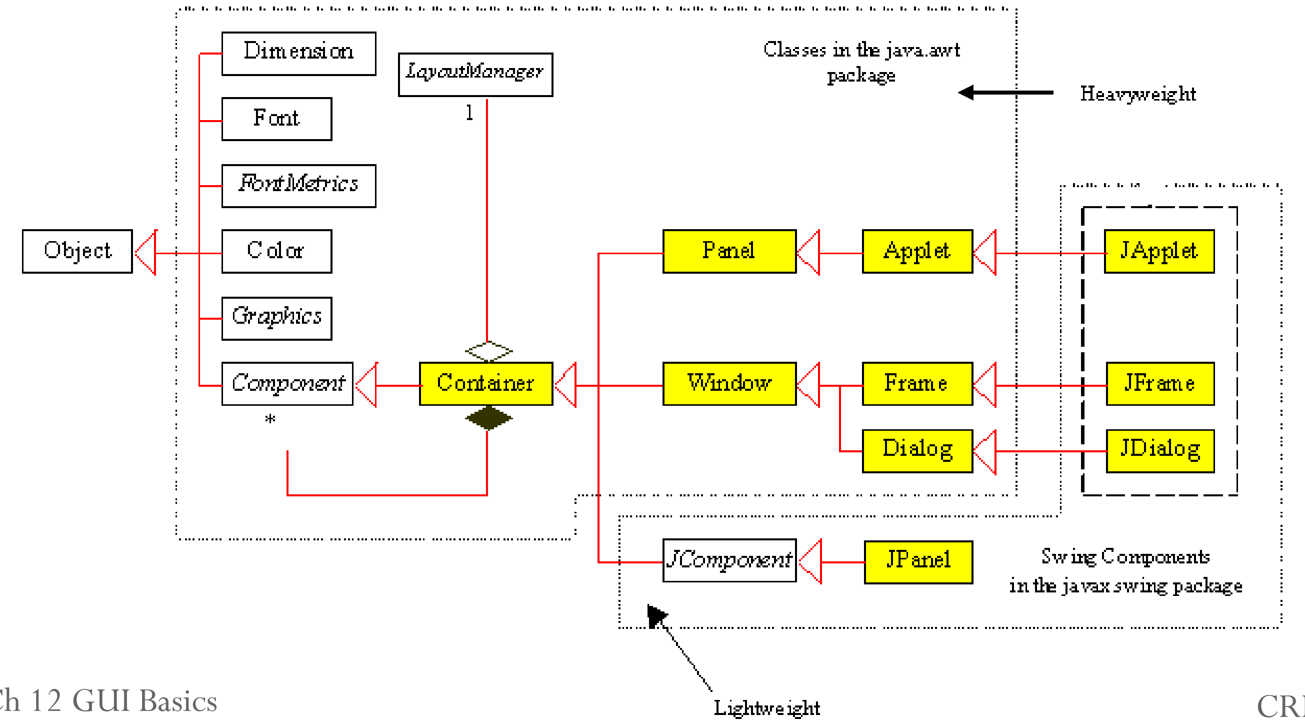
Java GUI API Classifications: Component

- Component classes → used for creating user interface
  - Instance can be displayed on screen
  - Abstract classes
    - Component is root class of all user-interface classes (*including container classes*)
    - JComponent is root class of all lightweight Swing components



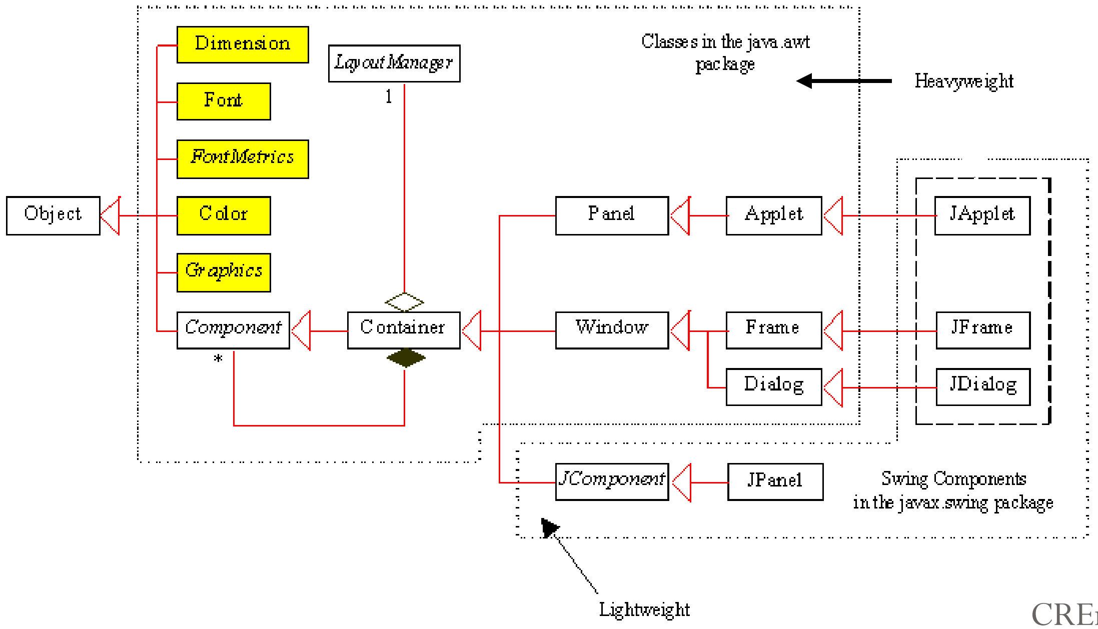
Java GUI API Classifications: Container

- Container classes → used to contain other components
  - Instance of Container class can hold instances of Component
  - Container, JFrame, JDialog, JApplet, JPanel *(see Table 12.1)*



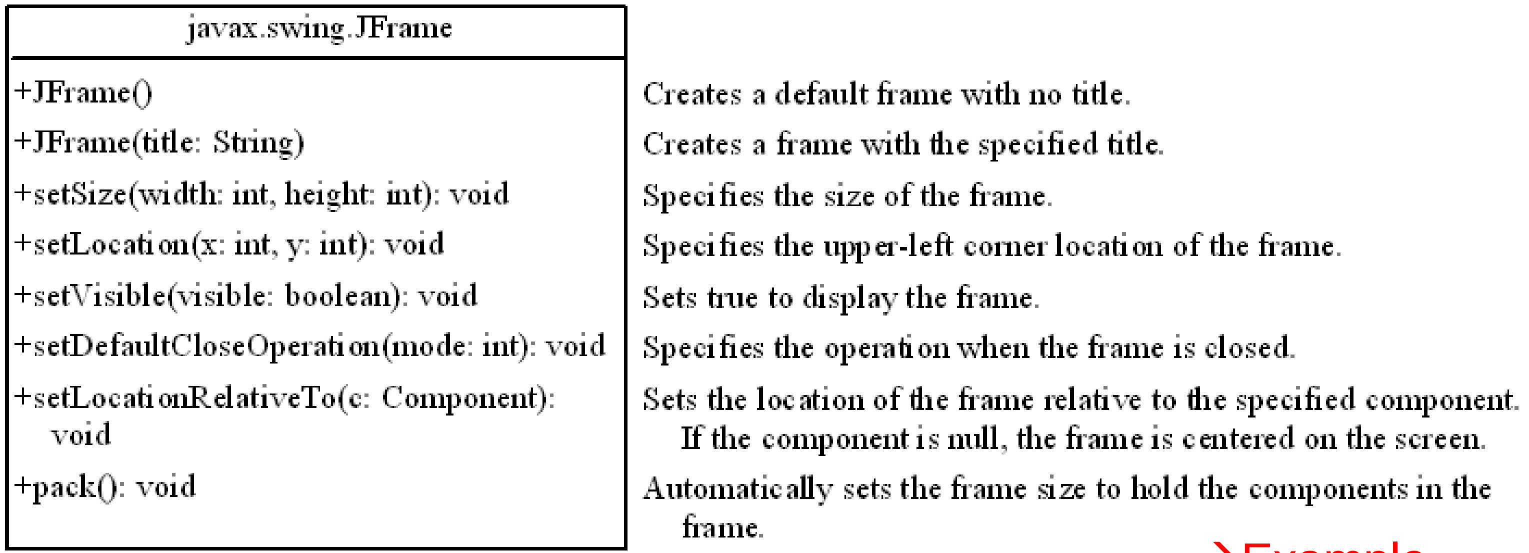
Java GUI API Classifications: Helper

- Helper classes → used to support GUI components
  - Used to describe properties of GUI components such as graphics context, colors, fonts, and dimensions
  - Graphics, Color, Font, FontMetrics, Dimension, and LayoutManager *(see Table 12.2)*



Container: Frames

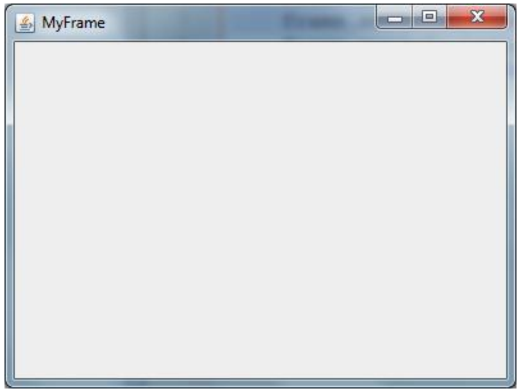
- Holds other user interface components in Java GUI applications
- For Swing GUI programs, use JFrame class to create windows
- Content-pane delegation → component is added to content pane of a frame (*or component is added to frame*)



javax.swing.JFrame	
+JFrame()	Creates a default frame with no title.
+JFrame(title: String)	Creates a frame with the specified title.
+setSize(width: int, height: int): void	Specifies the size of the frame.
+setLocation(x: int, y: int): void	Specifies the upper-left corner location of the frame.
+setVisible(visible: boolean): void	Sets true to display the frame.
+setDefaultCloseOperation(mode: int): void	Specifies the operation when the frame is closed.
+setLocationRelativeTo(c: Component): void	Sets the location of the frame relative to the specified component. If the component is null, the frame is centered on the screen.
+pack(): void	Automatically sets the frame size to hold the components in the frame.

→Example

Frames Example

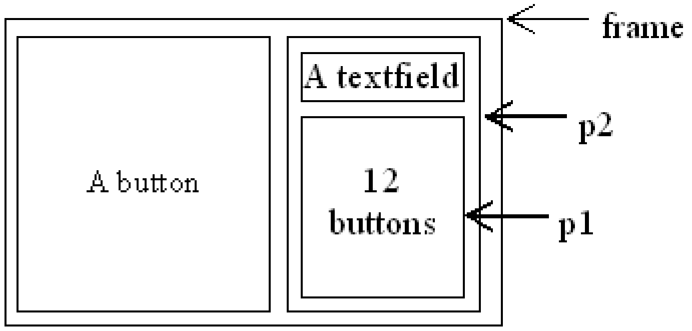


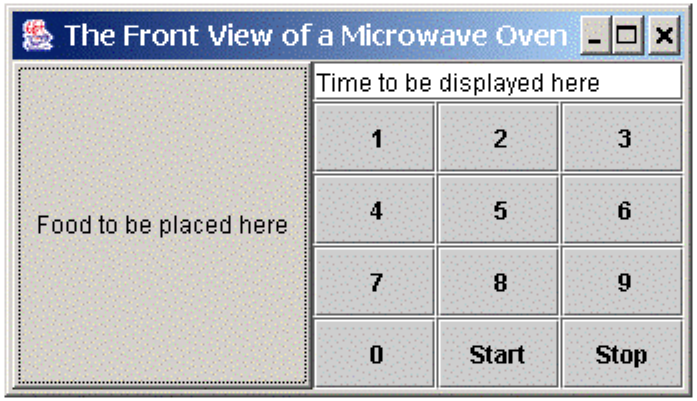
```
import javax.swing.*;

public class MyFrame {
  public static void main(String[] args) {
    JFrame frame = new JFrame("MyFrame"); // Create a frame
    frame.setSize(400, 300); // Set the frame size
    frame.setLocationRelativeTo(null); // New since JDK 1.4
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true); // Display the frame
  }
}
```

See 12.1 MyFrame.java

Container: Panels

- Panels (JPanel) act as subcontainer to group user-interface components
  - Add components to panel
  - Add panel to frame
  - Can also place panels in panel
- Panel is type of container class: panel has its own layout manager





See 12.6 TestPanels.java

Helper: Layout Managers

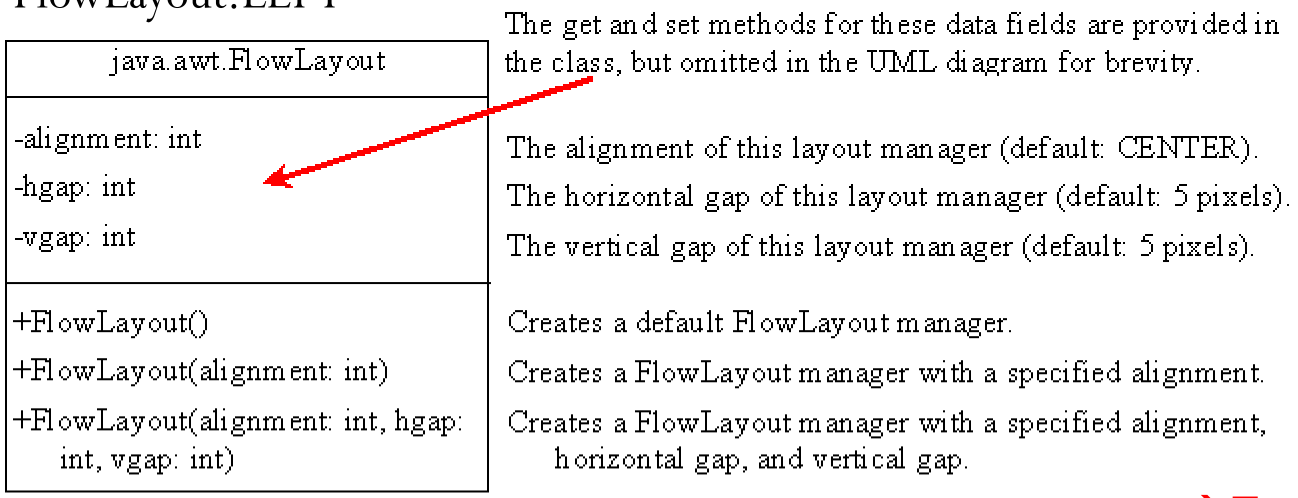
- Some window systems arrange UI components with hard-coded pixel measurements
  - Systems may display UI differently
  - Resizing is difficult
- Java GUI components placed in containers where arrangement is set by interface LayoutManager
  - Using one style* consistently makes programs easy to read
    - FlowLayout → arranged left to right, top to bottom
    - GridLayout → components arranged in matrix formation: left to right, starting with first row and continuing to next row
    - BorderLayout → components added to areas: East, South, West, North, and Center
  - Properties can be changed dynamically when layout explicitly reference by a variable
    - alignment, hgap, vgap

******these layouts are covered in textbook; they are* ***not*** *the only layouts available*

→Examples

Flow Layout Manager Example

- FlowLayout → arranged left to right, top to bottom
  - Default layout for JPanel class
  - Can specify pixel gap between components
  - Can specify how components are aligned using class static variables
    - FlowLayout.RIGHT
    - FlowLayout.CENTER
    - FlowLayout.LEFT



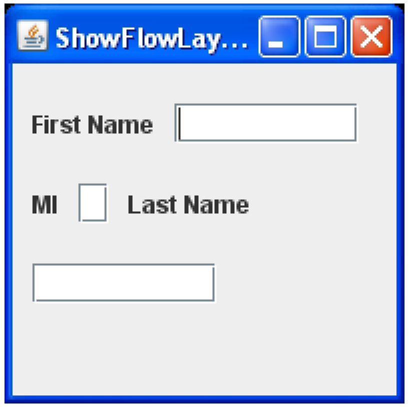
The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.

java.awt.FlowLayout	
-alignment: int	The alignment of this layout manager (default: CENTER).
-hgap: int	The horizontal gap of this layout manager (default: 5 pixels).
-vgap: int	The vertical gap of this layout manager (default: 5 pixels).
+FlowLayout()	Creates a default FlowLayout manager.
+FlowLayout(alignment: int)	Creates a FlowLayout manager with a specified alignment.
+FlowLayout(alignment: int, hgap: int, vgap: int)	Creates a FlowLayout manager with a specified alignment, horizontal gap, and vertical gap.

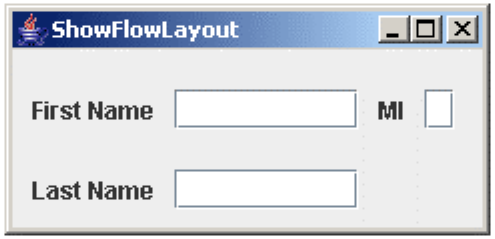
→Example

Flow Layout Manager Example

- FlowLayout → arranged left to right, top to bottom
  - Default layout for JPanel class
  - Can specify pixel gap between components
  - Can specify how components are aligned using class static variables
    - FlowLayout.RIGHT
    - FlowLayout.CENTER
    - FlowLayout.LEFT



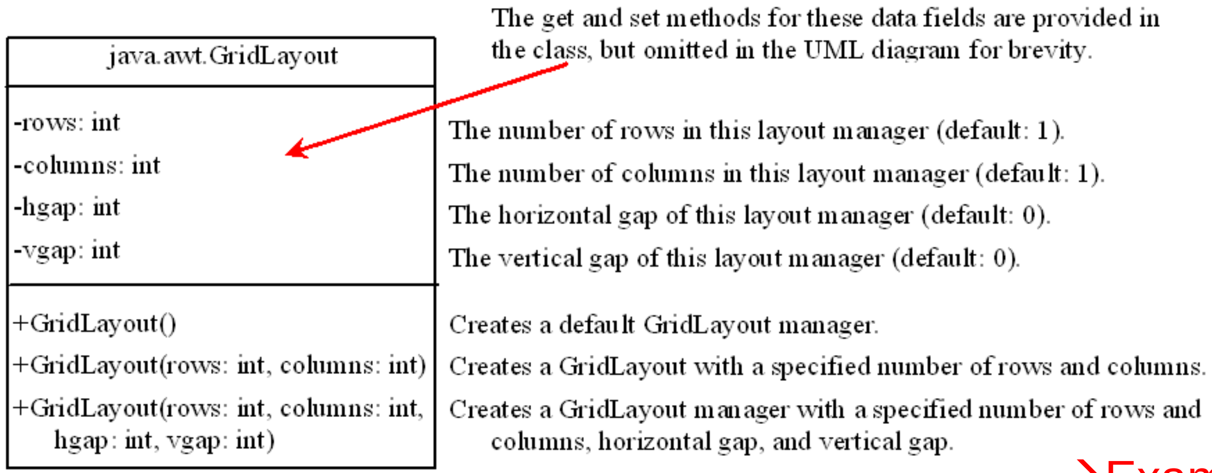
after resize



See 12.3 ShowFlowLayout.java

Grid Layout Example

- GridLayout → components arranged in matrix formation: left to right, starting with first row and continuing to next row
  - Specify rows and columns
    - rows **or** columns can be **zero** → **nonzero dimension** fixed and **zero dimension** dynamically determined by layout manager
    - rows **and** columns **nonzero** → number of **rows** fixed and number of **columns** dynamically determined by layout manager



The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.

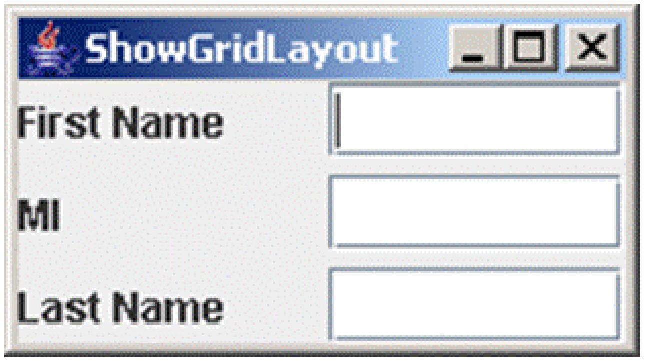
java.awt.GridLayout	
-rows: int	The number of rows in this layout manager (default: 1).
-columns: int	The number of columns in this layout manager (default: 1).
-hgap: int	The horizontal gap of this layout manager (default: 0).
-vgap: int	The vertical gap of this layout manager (default: 0).
+GridLayout()	Creates a default GridLayout manager.
+GridLayout(rows: int, columns: int)	Creates a GridLayout with a specified number of rows and columns.
+GridLayout(rows: int, columns: int, hgap: int, vgap: int)	Creates a GridLayout manager with a specified number of rows and columns, horizontal gap, and vertical gap.

→Example

Grid Layout Example

- GridLayout → components arranged in matrix formation: left to right, starting with first row and continuing to next row
  - Same example as previous with 3 rows and 2 columns: actual number of columns calculated by layout manager

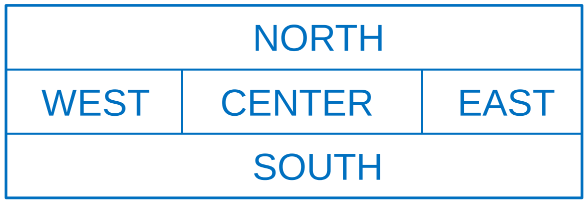
after resize, layout remains unchanged

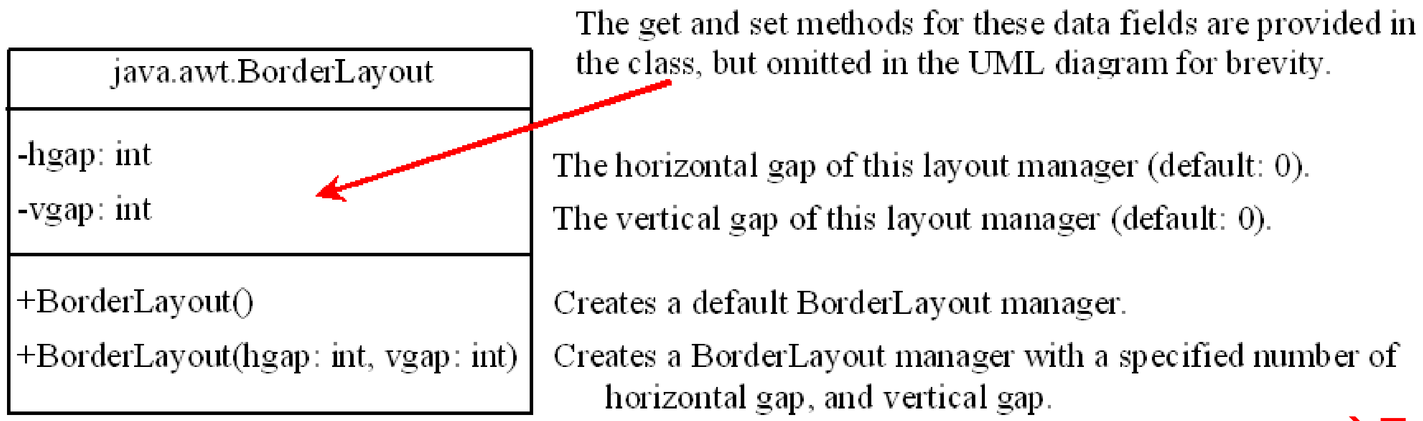


See 12.4 ShowGridLayout.java

Border Layout Example

- BorderLayout → components added to areas: East, South, West, North, and Center
  - Default layout for JFrame class
  - Components laid out according to their preferred sizes and placement in the container
    - North and South stretch **horizontally**
    - East and West stretch **vertically**
    - Center can stretch **horizontally** and **vertically**





The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.

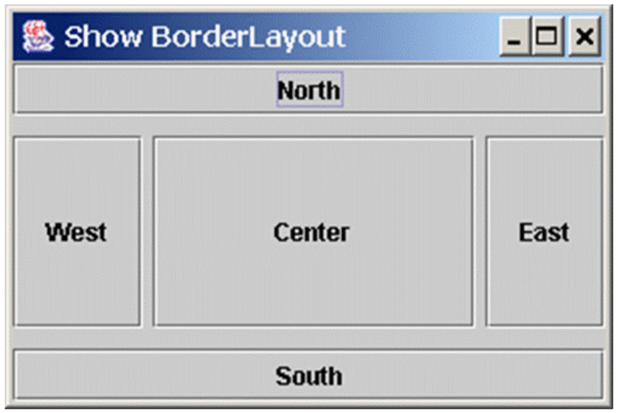
java.awt.BorderLayout	
-hgap: int	The horizontal gap of this layout manager (default: 0).
-vgap: int	The vertical gap of this layout manager (default: 0).
+BorderLayout()	Creates a default BorderLayout manager.
+BorderLayout(hgap: int, vgap: int)	Creates a BorderLayout manager with a specified number of horizontal gap, and vertical gap.

→Example

Border Layout Example

- BorderLayout → components added to areas: East, South, West, North, and Center
  - Example demonstrates 5 buttons labeled **East**, **South**, **West**, **North**, and **Center**

after resize, layout remains unchanged



See 12.5 ShowBorderLayout.java

Helper: Color Class

- Set colors for GUI components
- Standard Colors (java.awt.Color) *(static fields)*
  - BLACK, BLUE, CYAN, DARK_GRAY, GRAY, GREEN, LIGHT_GRAY, MAGENTA, ORANGE, PINK, RED, WHITE, and YELLOW
- Colors made of red, green, and blue components
  - alpha intensity ranging from **0** (no color) to **255** (true color)

```
public Color(int r, int g, int b);
```

- Example

```
JButton jbtOK = new JButton("OK");
jbtOK.setForeground(new Color(100, 1, 1));
jbtOK.setBackground(Color.WHITE);
```

RapidTables.com

Helper: Font Class

- Set fonts for GUI components

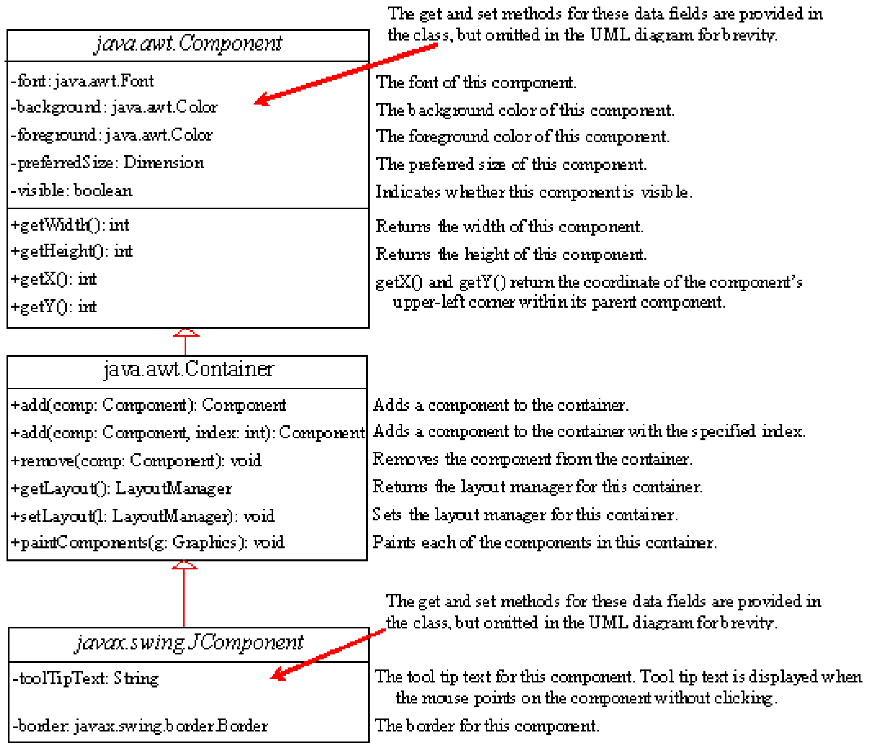
```
public Font(String name, int style, int
  pointSize);
```

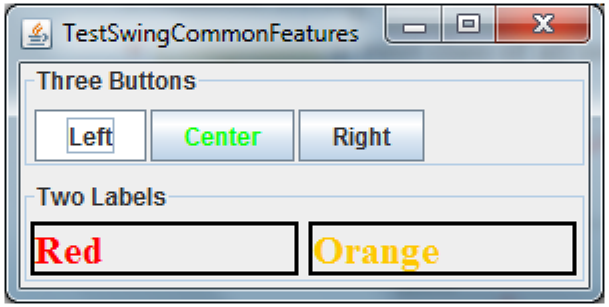
- Font Name *(static fields)*
  - SansSerif, Serif, Monospaced, Dialog, or Dialog Input
- Font Style *(static fields)*
  - Font.PLAIN(0), Font.BOLD(1), Font.ITALIC(2), and Font.BOLD+Font.ITALIC(3)
- Example

```
Font font1 = new Font("SansSerif",
  Font.BOLD, 16);
JButton jbtOK = new JButton("OK");
jbtOK.setFont(font1);
```

Java GUI API Classifications: Component

- Common Swing GUI Component Features
  - Component is root class of all user-interface classes (*including container classes*)


The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.
java.awt.Component
-font: java.awt.Font
-background: java.awt.Color
-foreground: java.awt.Color
-preferredSize: Dimension
-visible: boolean
+getWidth(): int
+getHeight(): int
+getX(): int
+getY(): int
The font of this component.
The background color of this component.
The foreground color of this component.
The preferred size of this component.
Indicates whether this component is visible.
Returns the width of this component.
Returns the height of this component.
getX() and getY() return the coordinate of the component's upper-left corner within its parent component.
java.awt.Container
+add(comp: Component): Component
+add(comp: Component, index: int): Component
+remove(comp: Component): void
+getLayout(): LayoutManager
+setLayout(l: LayoutManager): void
+paintComponents(g: Graphics): void
Adds a component to the container.
Adds a component to the container with the specified index.
Removes the component from the container.
Returns the layout manager for this container.
Sets the layout manager for this container.
Paints each of the components in this container.
The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.
javax.swing.JComponent
-toolTipText: String
-border: javax.swing.border.Border
The tool tip text for this component. Tool tip text is displayed when the mouse points on the component without clicking.
The border for this component.



See 12.7 TestSwingCommonFeatures.java

Component: Image Icons

- Icon is a fixed size picture
- Normally stored in image files
  - `.gif` or `.jpg` or `.png`
- To display an image,
  - create an ImageIcon object

```
ImageIcon usIcon = new
 ImageIcon(getClass().getResource("image/us.gif"));
```

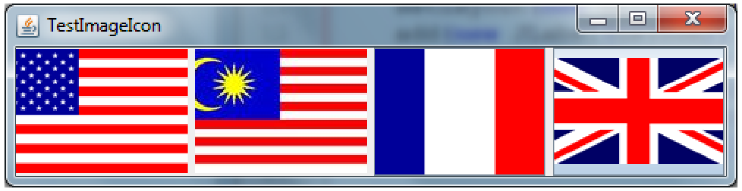
  - Use object in component:

```
JLabel jlblUS = new JLabel(usIcon);
```

- Borders and Icons **can be shared** between components

```
JButton jbtnUS = new JButton(usIcon);
```

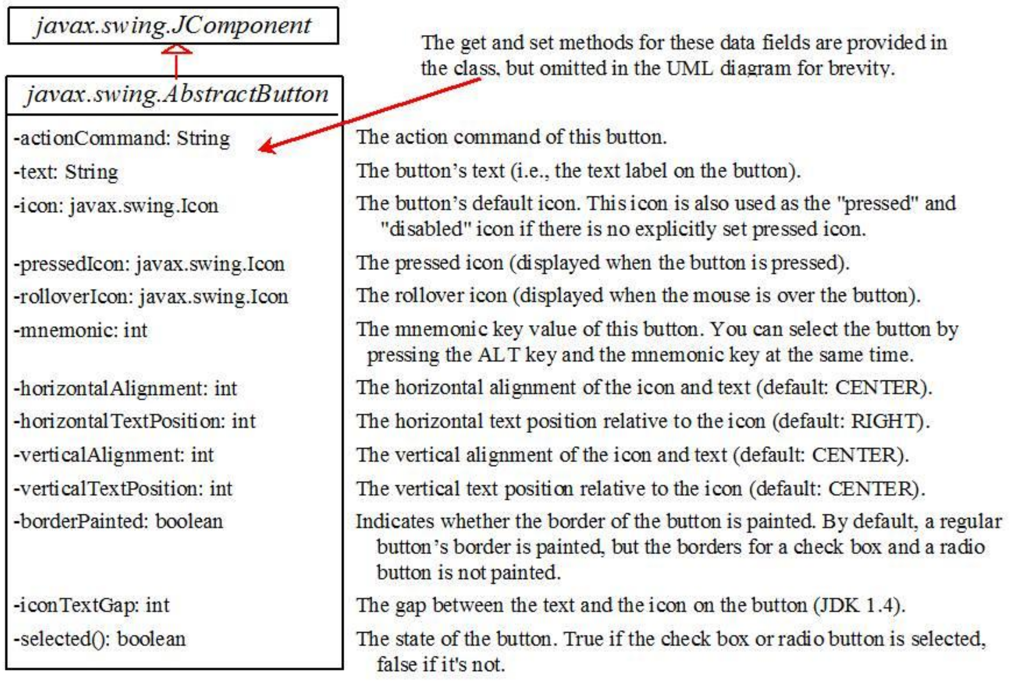
- GUI components **cannot be shared** by containers



See 12.8 TestImageIcon.java

Component: AbstractButton Class

- Common behaviors for buttons and menu items
  - Icons
    - Default
    - Pressed
    - Rollover
  - Alignments
  - Positions
- Types
  - Regular buttons
  - Toggle buttons
  - Check box buttons
  - Radio buttons



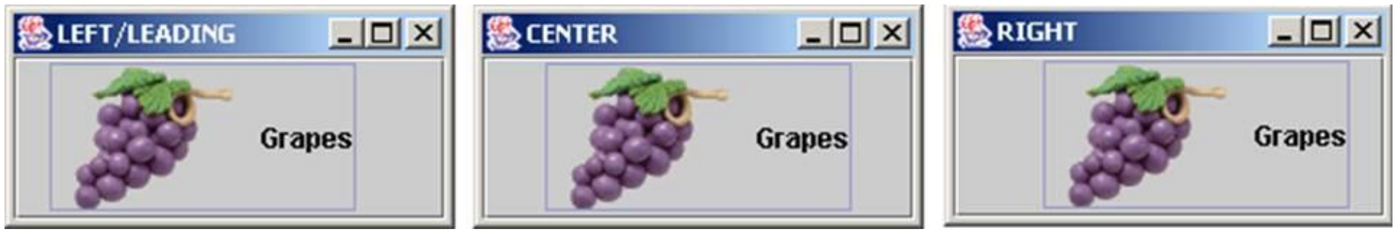
javax.swing.JComponent

javax.swing.AbstractButton

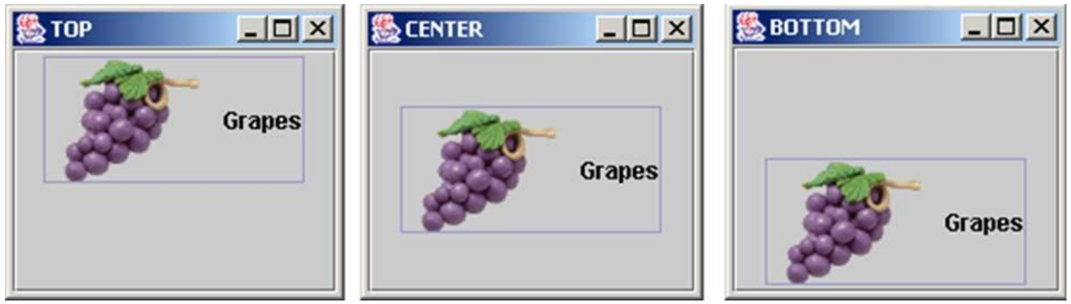
The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.

-actionCommand: String	The action command of this button.
-text: String	The button's text (i.e., the text label on the button).
-icon: javax.swing.Icon	The button's default icon. This icon is also used as the "pressed" and "disabled" icon if there is no explicitly set pressed icon.
-pressedIcon: javax.swing.Icon	The pressed icon (displayed when the button is pressed).
-rolloverIcon: javax.swing.Icon	The rollover icon (displayed when the mouse is over the button).
-mnemonic: int	The mnemonic key value of this button. You can select the button by pressing the ALT key and the mnemonic key at the same time.
-horizontalAlignment: int	The horizontal alignment of the icon and text (default: CENTER).
-horizontalTextPosition: int	The horizontal text position relative to the icon (default: RIGHT).
-verticalAlignment: int	The vertical alignment of the icon and text (default: CENTER).
-verticalTextPosition: int	The vertical text position relative to the icon (default: CENTER).
-borderPainted: boolean	Indicates whether the border of the button is painted. By default, a regular button's border is painted, but the borders for a check box and a radio button is not painted.
-iconTextGap: int	The gap between the text and the icon on the button (JDK 1.4).
-selected(): boolean	The state of the button. True if the check box or radio button is selected, false if it's not.

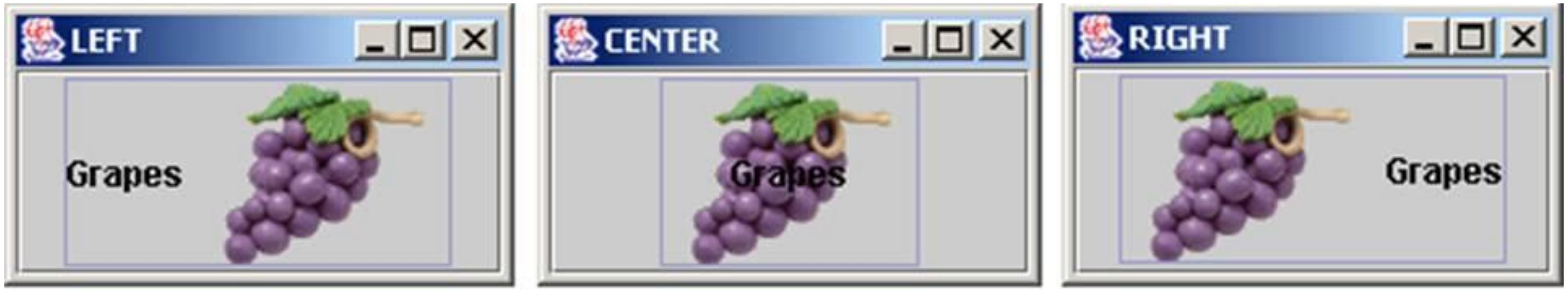
Component: AbstractButton Class



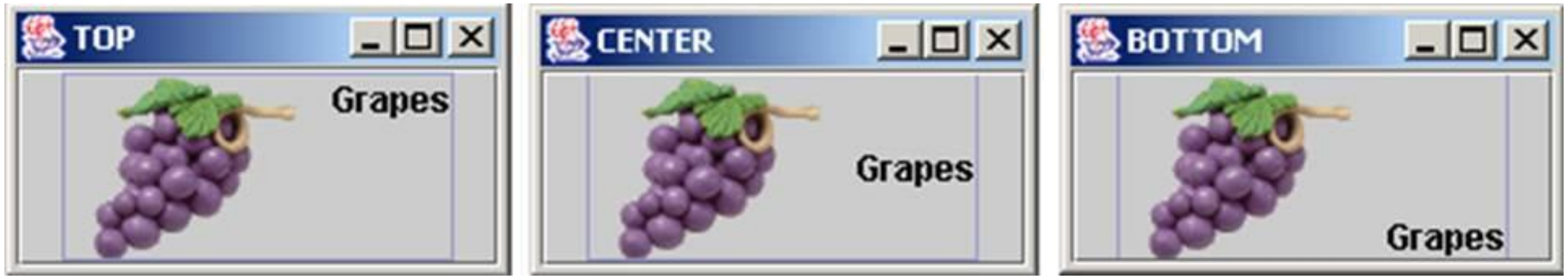
- Icon/Text Alignments *(icon and text on button)*
  - Horizontal
    - leading/left, center, right/trailing
  - Vertical
    - top, center, bottom



Component: AbstractButton Class

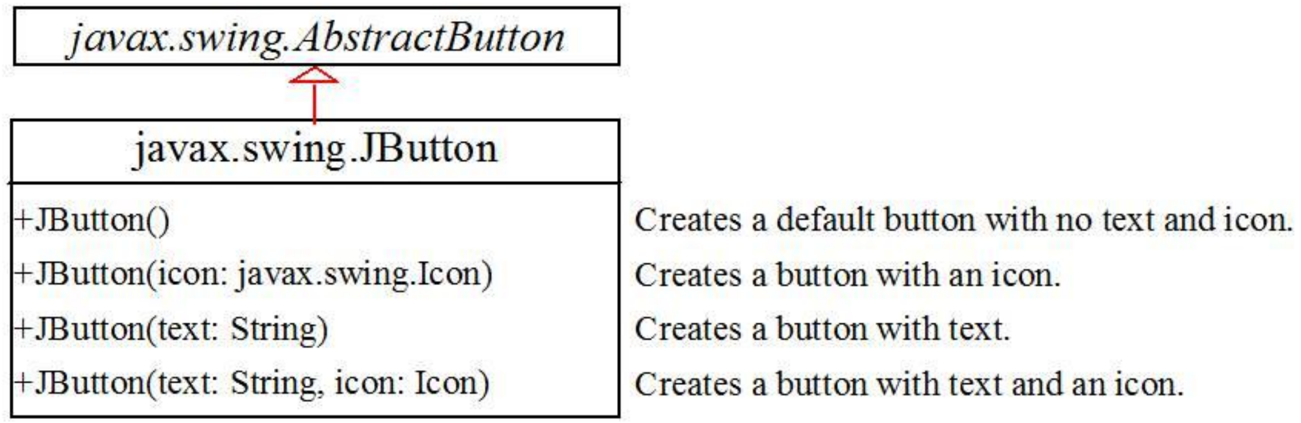


- Text Positions *(text relative to icon)*
  - Horizontal
    - leading/left, center, right/trailing
  - Vertical
    - top, center, bottom



Component: JButton Class

- Creates a push button component that triggers an action when clicked

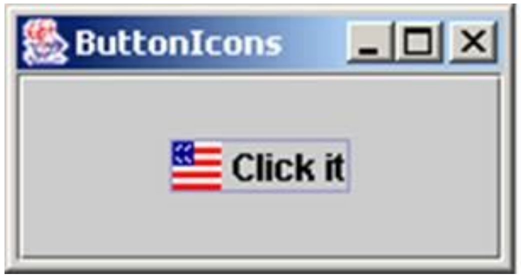


```
+JButton()                               Creates a default button with no text and icon.
+JButton(icon: javax.swing.Icon)         Creates a button with an icon.
+JButton(text: String)                   Creates a button with text.
+JButton(text: String, icon: Icon)       Creates a button with text and an icon.
```

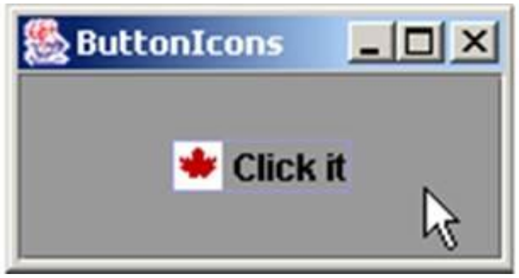
Component: JButton Class

- ***Example***:

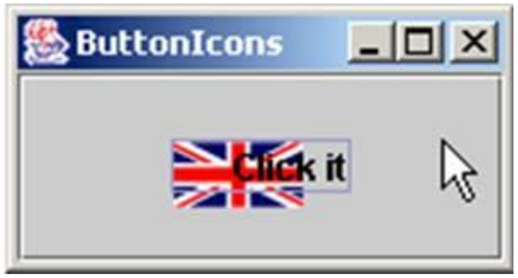
```
ImageIcon usIcon = new
  ImageIcon(getClass().getResource("image/usIcon.gif"));
ImageIcon caIcon = new
  ImageIcon(getClass().getResource("image/caIcon.gif"));
ImageIcon ukIcon = new
  ImageIcon(getClass().getResource("image/ukIcon.gif"));

JButton jbt = new JButton("Click it", usIcon);
jbt.setPressedIcon(caIcon);
jbt.setRolloverIcon(ukIcon);
```



Default Icon



Pressed Icon



Rollover Icon

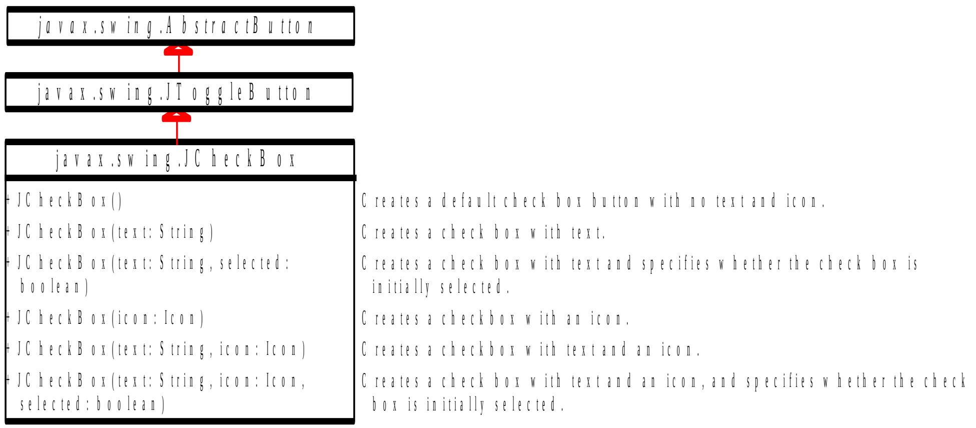
See 12.9 TestButtonIcons.java

Component: JCheckBox Class

- Creates a two-state, or toggle, button that operates like a light switch
- ***Example***:

```
JCheckBox jchk = new JCheckBox("Student", true);
jchk.setForeground(Color.RED);
jchk.setBackground(Color.WHITE);
jchk.setMnemonic('S');
```

use isSelected() method to see if check box is selected



javax.swing.AbstractButton

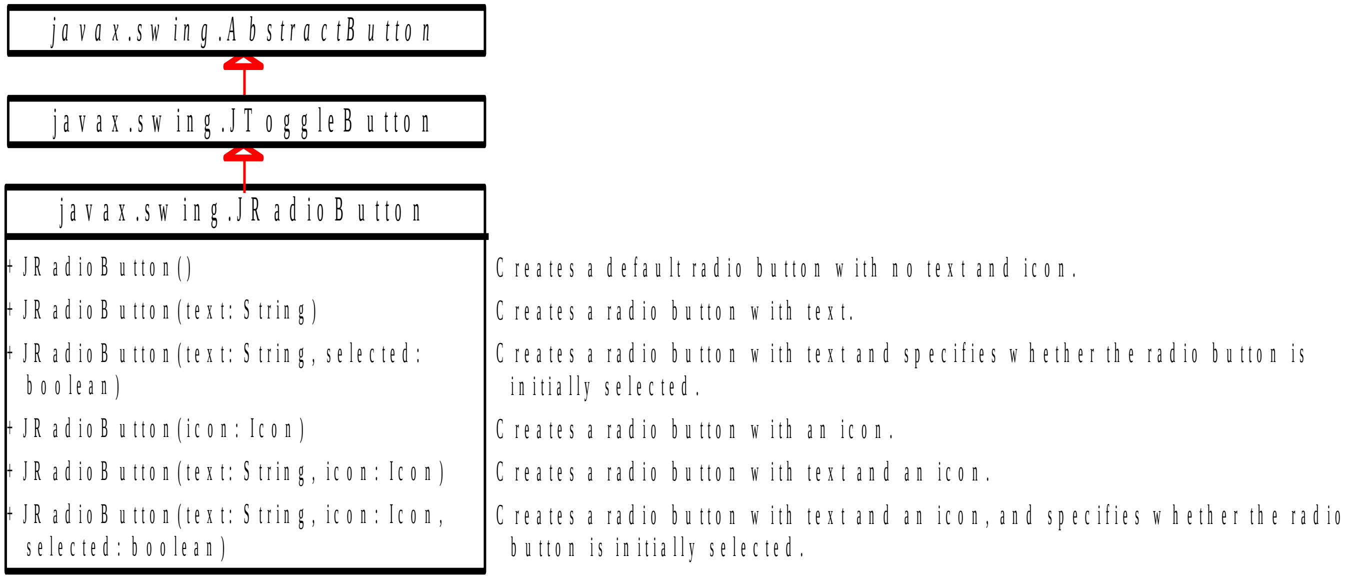
javax.swing.JToggleButton

javax.swing.JCheckBox

+JCheckBox()	Creates a default check box button with no text and icon.
+JCheckBox(text: String)	Creates a check box with text.
+JCheckBox(text: String, selected: boolean)	Creates a check box with text and specifies whether the check box is initially selected.
+JCheckBox(icon: Icon)	Creates a checkbox with an icon.
+JCheckBox(text: String, icon: Icon)	Creates a checkbox with text and an icon.
+JCheckBox(text: String, icon: Icon, selected: boolean)	Creates a check box with text and an icon, and specifies whether the check box is initially selected.

Component: JRadioButton Class

- Enables selection of a single item from a group of choices
- AKA option buttons
- Group individual JRadioButtons into ButtonGroup



+JRadioButton()	Creates a default radio button with no text and icon.
+JRadioButton(text: String)	Creates a radio button with text.
+JRadioButton(text: String, selected: boolean)	Creates a radio button with text and specifies whether the radio button is initially selected.
+JRadioButton(icon: Icon)	Creates a radio button with an icon.
+JRadioButton(text: String, icon: Icon)	Creates a radio button with text and an icon.
+JRadioButton(text: String, icon: Icon, selected: boolean)	Creates a radio button with text and an icon, and specifies whether the radio button is initially selected.

Component: JRadioButton Class

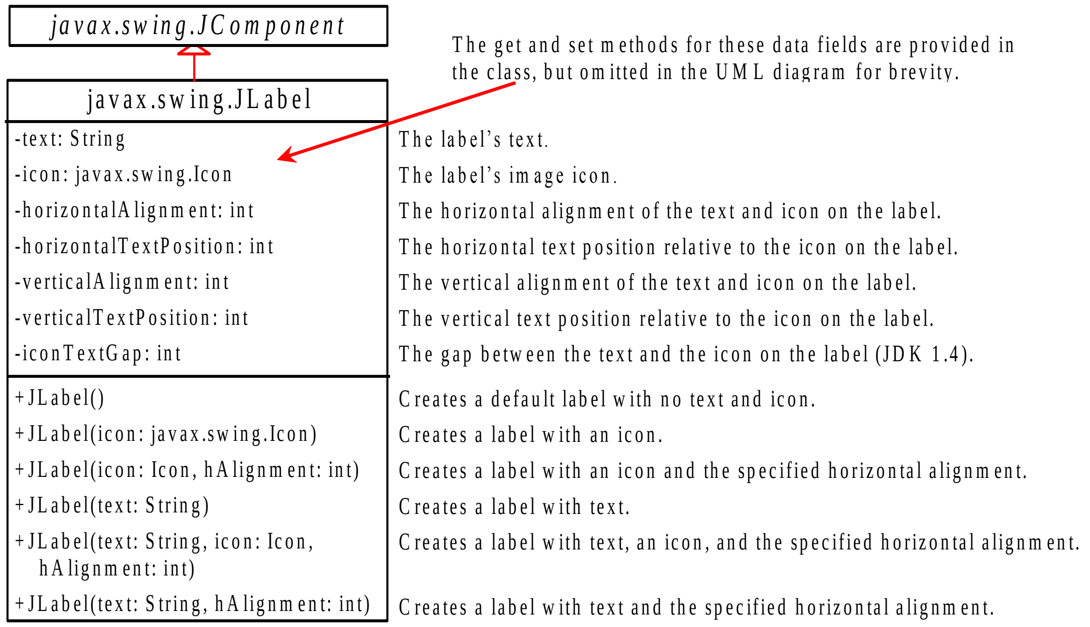
- ***Example***:

```
JRadioButton jrb1 = new JRadioButton("C++");
JRadioButton jrb2 = new JRadioButton("Java");
JRadioButton jrb3 = new JRadioButton("Python");
// group buttons so only one is selected
ButtonGroup bgroup = new ButtonGroup();
bgroup.add(jrb1);
bgroup.add(jrb2);
bgroup.add(jrb3);
jrb1.setSelected(true);
```

use isSelected() method to see if radio button is selected

Component: JLabel Class

- Creates display area for short text, image, or both
- Often used to label other components
- Inherits from JComponent and has many properties similar to JButton



javax.swing.JComponent

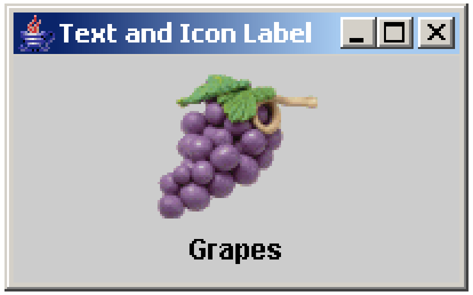
The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.

javax.swing.JLabel	
-text: String	The label's text.
-icon: javax.swing.Icon	The label's image icon.
-horizontalAlignment: int	The horizontal alignment of the text and icon on the label.
-horizontalTextPosition: int	The horizontal text position relative to the icon on the label.
-verticalAlignment: int	The vertical alignment of the text and icon on the label.
-verticalTextPosition: int	The vertical text position relative to the icon on the label.
-iconTextGap: int	The gap between the text and the icon on the label (JDK 1.4).
+JLabel()	Creates a default label with no text and icon.
+JLabel(icon: javax.swing.Icon)	Creates a label with an icon.
+JLabel(icon: Icon, hAlignment: int)	Creates a label with an icon and the specified horizontal alignment.
+JLabel(text: String)	Creates a label with text.
+JLabel(text: String, icon: Icon, hAlignment: int)	Creates a label with text, an icon, and the specified horizontal alignment.
+JLabel(text: String, hAlignment: int)	Creates a label with text and the specified horizontal alignment.

Component: JLabel Class

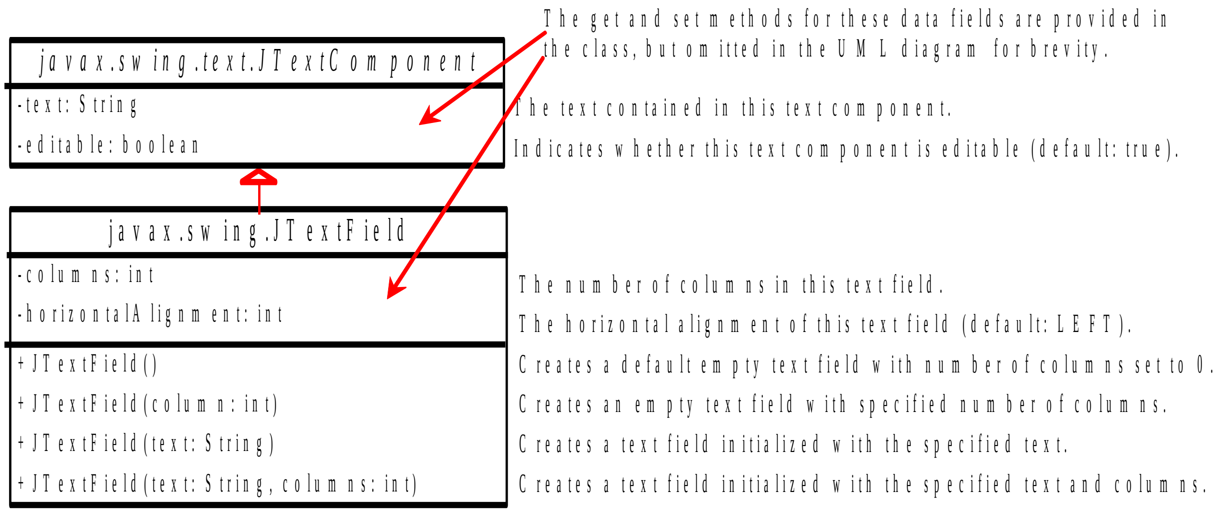
- ***Example*:**

```
// Create an image icon from image resource file
ImageIcon icon = new ImageIcon(getClass()
  .getResource("image/grapes.gif"));

// Create a label with text, an icon,
// with centered horizontal alignment
JLabel jlbl = new JLabel("Grapes", icon,
  SwingConstants.CENTER);

// Set label's text alignment and gap between text and icon
jlbl.setHorizontalTextPosition(SwingConstants.CENTER);
jlbl.setVerticalTextPosition(SwingConstants.BOTTOM);
jlbl.setIconTextGap(5);
```



Component: JTextField Class

- Used to enter or display a string
  - Enable user to enter data
- Subclass of JTextComponent


javax.swing.text.JTextComponent
-text: String
-editable: boolean
javax.swing.JTextField
-columns: int
-horizontalAlignment: int
+JTextField()
+JTextField(column: int)
+JTextField(text: String)
+JTextField(text: String, columns: int)

The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.

The text contained in this text component.

Indicates whether this text component is editable (default: true).

The number of columns in this text field.

The horizontal alignment of this text field (default: LEFT).

Creates a default empty text field with number of columns set to 0.

Creates an empty text field with specified number of columns.

Creates a text field initialized with the specified text.

Creates a text field initialized with the specified text and columns.

Component: JTextField Class

- ***Example***:

```
// text field with displayed text
JTextField jtfMessage = new JTextField("Illinois");
jtfMessage.setForeground(Color.RED);
jtfMessage.setHorizontalAlignment(JTextField.RIGHT);

// label with empty text
JLabel jlblName = new JLabel("Your Name:",
  SwingConstants.RIGHT);
JTextField jtfName = new JTextField("");
```

use setText() method to set new text; use getText() to get text