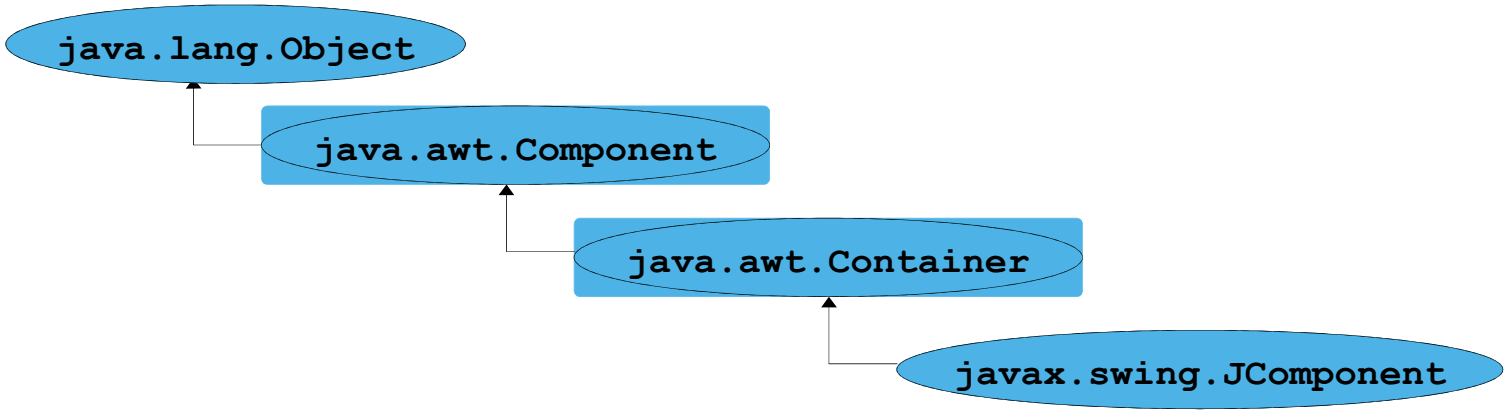
Java GUI Components: Details

- Swing Overview
- JLabel
- Event Handling Model
- JTextField and JPasswordField
- JTextArea
- JButton
- JCheckBox
- JComboBox
- Mouse Event Handling
- Layout Managers
  - FlowLayout
  - BorderLayout
  - GridLayout
- Panels
  - Creating a Self-Contained Subclass of JPanel
- Windows
- Using Menus with Frames

Swing Overview

- Swing GUI components
  - Defined in package `javax.swing`
  - Original GUI components from Abstract Windowing Toolkit in `java.awt`
    - Heavyweight components - rely on local platform's windowing system for look and feel
  - Swing components are lightweight
    - Written in Java, not weighed down by complex GUI capabilities of platform
    - More portable than heavyweight components
  - Swing components allow programmer to specify look and feel
    - Can change depending on platform
    - Can be the same across all platforms

Swing Overview



Swing Overview

- Swing component inheritance hierarchy
  - **Component** defines methods that can be used in its subclasses
  - **Container** - collection of related components
    - When using **JFrames**, attach components to the content panel (a **Container**)
    - Method **add** to add components to content pane
  - **JComponent** - superclass to most Swing components
  - Much of a component's functionality inherited from these classes

JLabel

- Labels
  - Provide text instructions on a GUI
  - Read-only text
  - Programs rarely change a label's contents
  - Class **JLabel** (subclass of **JComponent**)
- Methods
  - Can declare label text in constructor
  - **myLabel.setToolTipText("Text"**
- Displays **"Text"**in a tool tip when mouse over label
  - **myLabel.setText("Text")**
  - **myLabel.getText()**

Event Handling Model

- GUIs are event driven
  - Generate events when user interacts with GUI
    - Mouse movements, mouse clicks, typing in a text field, etc.
  - Event information stored in object that extends AWTEvent
- To process an event
  - Register an event listener
    - Object from a class that implements an event-listener interface (from java.awt.event or javax.swing.event)
    - "Listens" for events
  - Implement event handler
    - Method that is called in response to an event
    - Each event handling interface has one or more event handling methods that must be defined

Event Handling Model

- Delegation event model
  - Use of event listeners in event handling
  - Processing of event delegated to particular object
- When an event occurs
  - GUI component notifies its listeners
  - Calls listener's event handling method
- Example:
  - *Enter* pressed in a **JTextField**
  - Method **actionPerformed** called for registered listener

TextField and JPasswordField

- **JTextField**s and **JPasswordField**s
  - Single line areas in which text can be entered or displayed
  - **JPasswordField**s show inputted text as *
  - **JTextField** extends **JTextComponent**
    - **JPasswordField** extends **JTextField**
- When **Enter** pressed
  - **ActionEvent** occurs
  - Currently active field "has the focus"
- Methods
  - Constructor
    - **JTextField(10)** - sets textfield with 10 columns of text
    - **JTextField("Hi")** - sets text, width determined automatically

JTextField and JPasswordField

- Methods (continued)
  - **setEditable(boolean)**
    - If **true**, user can edit text
  - **getPassword**
    - Class **JPasswordField**
    - Returns password as an **array** of type **char**
- Example
  - Create **JTextField**s and a **JPasswordField**
  - Create and register an event handler
    - Displays a dialog box when *Enter* pressed

JTextArea

- Area for manipulating multiple lines of text
  - Like **JTextField**, inherits from **JTextComponent**
  - Many of the same methods
- **JScrollPane**
  - Provides scrolling
  - Initialize with component
    - **new JScrollPane(myComponent)**
  - Can set scrolling policies (always, as needed, never)
- **Box container**
  - Uses **BoxLayout** layout manager
  - Arrange GUI components horizontally or vertically
  - **Box b = Box.createHorizontalbox();**
    - Arranges components attached to it from left to right, in order attached

JButton

- **Button**
  - Component user clicks to trigger an action
  - Several types of buttons
    - Command buttons, toggle buttons, check boxes, radio buttons
- **Command button**
  - Generates **ActionEvent** when clicked
  - Created with class **JButton**
    - Inherits from class **AbstractButton**
- **Jbutton**
  - Text on face called button label
  - Each button should have a different label
  - Support display of **Icon**s

JButton

- Constructors

```
Jbutton myButton = new JButton( "Button" );
Jbutton myButton = new JButton( "Button",
  myIcon );
```

- Method
  - **`setRolloverIcon( myIcon )`**
    - Sets image to display when mouse over button

JCheckBox

- **State buttons**
  - **JToggleButton**
    - Subclasses **JCheckBox**, **JRadioButton**
  - Have on/off (true/false) values
  - We discuss **JCheckBox** in this section
- **Initialization**
  - **JCheckBox myBox = new JCheckBox("Title");**
- **When JCheckBox changes**
  - **ItemEvent** generated
    - Handled by an **ItemListener**, which must define **itemStateChanged**
  - Register with **addItemListener**

JCheckBox

- **ItemEvent** methods
  - **getStateChange**
    - Returns **ItemEvent.SELECTED** or **ItemEvent.DESELECTED**

JComboBox

- Combo box (drop down list)
  - List of items, user makes a selection
  - Class **JComboBox**
    - Generate **ItemEvents**
- **JComboBox**
  - Numeric index keeps track of elements
    - First element added at index **0**
    - First item added is appears as currently selected item when combo box appears

JComboBox

- Methods
  - **getSelectedIndex**
    - Returns the index of the currently selected item
  - **setMaximumRowCount(n)**
    - Set the maximum number of elements to display when user clicks combo box
    - Scrollbar automatically provided

Mouse Event Handling

- Mouse events
  - Can be trapped for any GUI component derived from **java.awt.Component**
  - Mouse event handling methods
    - Take a **MouseEvent** object
      - Contains info about event, including **x** and **y** coordinates
      - Methods **getX** and **getY**
  - **MouseListener** and **MouseMotionListener** methods called automatically (if component is registered)
    - **addMouseListener**
    - **addMouseMotionListener**

Layout Managers

- Layout managers
  - Arrange GUI components on a container
  - Provide basic layout capabilities
    - Easier to use than determining exact size and position of every component
    - Programmer concentrates on "look and feel" rather than details

FlowLayout

- Most basic layout manager
  - Components placed left to right in order added
  - When edge of container reached, continues on next line
  - Components can be left-aligned, centered (default), or right-aligned
- Method
  - **setAlignment**
    - **FlowLayout.LEFT, FlowLayout.CENTER, FlowLayout.RIGHT**
  - **layoutContainer(Container)**
    - Update **Container** specified with layout

BorderLayout

- **BorderLayout**
  - Default manager for content pane
  - Arrange components into 5 regions
    - North, south, east, west, center
  - Up to 5 components can be added directly
    - One for each region
  - Components placed in
    - North/South - Region is as tall as component
    - East/West - Region is as wide as component
    - Center - Region expands to take all remaining space

BorderLayout

- Methods
  - Constructor: **BorderLayout(hGap, vGap);**
    - **hGap** - horizontal gap space between regions
    - **vGap** - vertical gap space between regions
    - Default is **0** for both
  - Adding components
    - **myLayout.add(component, position)**
    - **component** - component to add
    - **position** - **BorderLayout.NORTH**
      - **SOUTH, EAST, WEST, CENTER** similar
  - **setVisible(boolean)** (in class **Jbutton**)
    - If **false**, hides component
  - **layoutContainer(container)** - updates container, as before

GridLayout

- **GridLayout**
  - Divides container into a grid
  - Components placed in rows and columns
  - All components have same width and height
    - Added starting from top left, then from left to right
    - When row full, continues on next row, left to right
- **Constructors**
  - **GridLayout(rows, columns, hGap, vGap)**
    - Specify number of rows and columns, and horizontal and vertical gaps between elements (in pixels)
  - **GridLayout(rows, columns)**
    - Default **0** for **hGap** and **vGap**

GridLayout

- Updating containers
  - **Container** method **validate**
    - Re-layouts a container for which the layout has changed
  - Example:

```
Container c = getContentPane;
c.setLayout( myLayout );
if ( x = 3 ){
   c.setLayout( myLayout2 );
   c.validate();
}
```

    - Changes layout and updates **c** if condition met

Panels

- **Complex GUIs**
  - Each component needs to be placed in an exact location
  - Can use multiple panels
    - Each panel's components arranged in a specific layout
- **Panels**
  - Class **JPanel** inherits from **JComponent**, which inherits from **java.awt.Container**
    - Every **JPanel** is a **Container**
  - **JPanels** can have components (and other **JPanels**) added to them
    - **JPanel** sized to components it contains
    - Grows to accommodate components as they are added

Panels

- Usage
  - Create panels, and set the layout for each
  - Add components to the panels as needed
  - Add the panels to the content pane (default **BorderLayout**)

Creating a Self-Contained Subclass of JPanel -1-

- **JPanel**
  - Can be used as a dedicated drawing area
    - Receive mouse events
    - Can extend to create new components
  - Combining Swing GUI components and drawing can lead to improper display
    - GUI may cover drawing, or may be able to draw over GUI components
  - Solution: separate the GUI and graphics
    - Create dedicated drawing areas as subclasses of **JPanel**

Creating a Self-Contained Subclass of JPanel -2-

- Swing components inheriting from **`JComponent`**
  - Contain method **`paintComponent`**
    - Helps to draw properly in a Swing GUI
  - When customizing a JPanel, override **`paintComponent`**

```
public void paintComponent( Graphics g )
 {
    super.paintComponent( g );
    //additional drawing code
 }
```

  - Call to superclass **`paintComponent`** ensures painting occurs in proper order
    - The call should be the first statement - otherwise, it will erase any drawings before it

Creating a Self-Contained Subclass of JPanel -3-

- **`JFrame`** and **`JApplet`**
  - Not subclasses of **`JComponent`**
    - Do not contain **`paintComponent`**
  - Override **`paint`** to draw directly on subclasses
- Events
  - **`JPanels`** do not create events like buttons
  - Can recognize lower-level events
    - Mouse and key events

Creating a Self-Contained Subclass of JPanel-4-

- Example
  - Create a subclass of **`JPanel`** named **`SelfContainedPanel`** that listens for its own mouse events
    - Draws an oval on itself (overrides **`paintComponent`**)
  - Import **`SelfContainedPanel`** into another class
    - The other class contains its own mouse handlers
  - Add an instance of **`SelfContainedPanel`** to the content panel

Windows

- **JFrame**
  - Inherits from **java.awt.Frame**, which inherits from **java.awt.Window**
  - **JFrame** is a window with a title bar and a border
    - Not a lightweight component - not written completely in Java
    - Window part of local platform's GUI components
      - Different for Windows, Macintosh, and UNIX
- **JFrame** operations when user closes window
  - Controlled with method **setDefaultCloseOperation**
    - Interface **`WindowConstants`** (**`javax.swing`**) has three constants to use
    - **`DISPOSE_ON_CLOSE`**, **`DO_NOTHING_ON_CLOSE`**, **`HIDE_ON_CLOSE`** (default)

Windows -2-

- Windows take up valuable resources
  - Explicitly remove windows when not needed with method **dispose** (of class **Window**, indirect superclass of **JFrame**)
    - Or, use **setDefaultCloseOperation**
  - **DO_NOTHING_ON_CLOSE** - you determine what happens when user wants to close window
- Display
  - By default, window not displayed until method **show** called
  - Can display by calling method **setVisible(true)**
  - Method **setSize** - make sure to set a window's size, otherwise only the title bar will appear

Windows -3-

- All windows generate window events
  - **addWindowListener**
  - **WindowListener** interface has 7 methods
    - **windowActivated**
    - **windowClosed** (called after window closed)
    - **windowClosing** (called when user initiates closing)
    - **windowDeactivated**
    - **windowIconified** (minimized)
    - **windowDeiconified**
    - **windowOpened**

Using Menus with Frames

- **Menus**
  - Important part of GUIs
  - Perform actions without cluttering GUI
  - Attached to objects of classes that have method **setJMenuBar**
    - **JFrame** and **JApplet**
- **Classes used to define menus**
  - **JMenuBar** - container for menus, manages menu bar
  - **JMenuItem** - manages menu items
    - Menu items - GUI components inside a menu
    - Can initiate an action or be a submenu

Using Menus with Frames -2-

- **Classes used to define menus** (continued)
  - **JMenu** - manages menus
    - Menus contain menu items, and are added to menu bars
    - Can be added to other menus as submenus
    - When clicked, expands to show list of menu items
  - **JCheckBoxMenuItem**
    - Manages menu items that can be toggled
    - When selected, check appears to left of item
  - **JRadioButtonMenuItem**
    - Manages menu items that can be toggled
    - When multiple **JRadioButtonMenuItem**s are part of a group, only one can be selected at a time
    - When selected, filled circle appears to left of item

Using Menus with Frames -3-

- **Mnemonics**
  - Provide quick access to menu items (File)
    - Can be used with classes that have subclass **javax.swing.AbstractButton**
  - Use method **setMnemonic**

    JMenu fileMenu = new JMenu("File")

    fileMenu.setMnemonic('F');
    - Press **Alt + F** to access menu
- **Methods**
  - **setSelected(true)**
    - Of class **AbstractButton**
    - Sets button/item to selected state

Using Menus with Frames -3-

- **Methods** (continued)
  - addSeparator()
    - Class JMenu
    - Inserts separator line into menu
- **Dialog boxes**
  - Modal - No other window can be accessed while it is open (default)
    - Modeless - other windows can be accessed
  - JOptionPane.showMessageDialog(parentWindow, String, title, messageType)
  - parentWindow - determines where dialog box appears
    - null - displayed at center of screen
    - window specified - dialog box centered horizontally over parent

Using Menus with Frames-4-

- **Using menus**
  - Create menu bar
    - Set menu bar for **JFrame** (**setJMenuBar(myBar);**)
  - Create menus
    - Set Mnemonics
  - Create menu items
    - Set Mnemonics
    - Set event handlers
  - If using **JRadioButtonMenuItem**s
    - Create a group: **myGroup = new ButtonGroup();**
    - Add **JRadioButtonMenuItem**s to the group

Using Menus with Frames-5-

- **Using menus** (continued)
  - Add menu items to appropriate menus
    - **myMenu.add(myItem);**
    - Insert separators if necessary: **myMenu.addSeparator();**
  - If creating submenus, add submenu to menu
    - **myMenu.add(mySubMenu);**
  - Add menus to menu bar
    - **myMenuBar.add(myMenu);**
- Example
  - Use menus to alter text in a **JLabel**
  - Change color, font, style
  - Have a "File" menu with a "About" and "Exit" items