**Graphical User Interfaces**

- GUI Design considerations
- Putting GUI components together

---

**GUI Design**

- Remember that our goal is to solve a problem using the tools needed to put a GUI together

- The GUI designer should:
  - Know the users and their needs
  - Prevent user errors whenever possible
  - Optimize user abilities and make information readily available
  - Be consistent with placement of components and color schemes
A layout manager is an object that determines the manner in which components are arranged in a container.

There are several predefined layout managers defined in the Java standard class library:

<table>
<thead>
<tr>
<th>Defined in the AWT</th>
<th>Defined in Swing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow Layout</td>
<td>Box Layout</td>
</tr>
<tr>
<td>Border Layout</td>
<td>Grid Bag Layout</td>
</tr>
<tr>
<td>Card Layout</td>
<td></td>
</tr>
<tr>
<td>Grid Layout</td>
<td></td>
</tr>
<tr>
<td>Grid Bag Layout</td>
<td></td>
</tr>
<tr>
<td>Layout</td>
<td></td>
</tr>
<tr>
<td>Overlay Layout</td>
<td></td>
</tr>
</tbody>
</table>

Every container has a default layout manager, but we can explicitly set the layout manager as well.

Each layout manager has its own particular rules governing how the components will be arranged.

Some layout managers pay attention to a component's preferred size or alignment, while others do not.

A layout manager attempts to adjust the layout as components are added and as containers are resized.
Layout Managers

- We can use the `setLayout` method of a container to change its layout manager
  
  ```java
  JPanel panel = new JPanel();
  panel.setLayout (new BorderLayout());
  ```

- The following example uses a *tabbed pane*, a container which permits one of several panes to be selected

- See `LayoutDemo.java`

Flow Layout

- *Flow layout* puts as many components as possible on a row, and then moves to the next row

- Rows are created as needed to accommodate all of the components

- Components are displayed in the order they are added to the container

- Each row of components is centered horizontally in the window by default, but could also be aligned left or right

- The horizontal and vertical gaps between the components can be explicitly set also

- See `FlowPanel.java`
Border Layout

- A border layout defines five areas to which components can be added

<table>
<thead>
<tr>
<th>North</th>
<th>Center</th>
<th>East</th>
</tr>
</thead>
<tbody>
<tr>
<td>West</td>
<td></td>
<td></td>
</tr>
<tr>
<td>South</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Each area displays one component (which could be another container such as a JPanel)
- Each of the four outer areas enlarges as needed to accommodate the component added to it
- If nothing is added to the outer areas, they take up no space and other areas expand to fill the void
- The center area expands to fill space as needed
- See BorderPanel.java
Grid Layout

- A *grid layout* presents a container’s components in a rectangular grid of rows and columns
- One component is placed in each cell of the grid, and all cells have the same size
- As components are added to the container, they fill the grid from left-to-right and top-to-bottom (by default)
- The size of each cell is determined by the overall size of the container
- See `GridPanel.java`

Box Layout

- A *box layout* organizes components either horizontally (in one row) or vertically (in one column)
- Components are placed top-to-bottom or left-to-right in the order in which they are added to the container
- By combining multiple containers using box layout, many different configurations can be created
- Multiple containers with box layouts are often preferred to one container that uses the more complicated gridbag layout manager
Box Layout

- **Invisible components** can be added to a box layout container to take up space between components
  - *Rigid areas* have a fixed size
  - *Glue* specifies where excess space should go
- A rigid area is created using the `createRigidArea` method of the Box class
- Glue is created using the `createHorizontalGlue` or `createVerticalGlue` methods
- See `BoxPanel.java`

Containment Hierarchies

- The way components are grouped into containers and the way those containers are nested within each other establishes the *containment hierarchy* for the GUI
- Each container can have its own layout manager
- The appearance of a GUI is determined by:
  - the containment hierarchy
  - the layout manager of each container
  - the properties of individual components
- All of these issues work together to determine the final visual effect
Containment Hierarchies

Special Features

- Swing components offer special features to facilitate and enhance their use

<table>
<thead>
<tr>
<th>Special Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tool tip</td>
<td>Causes a line of text to appear when the mouse cursor pauses over a component</td>
</tr>
<tr>
<td>Mnemonic</td>
<td>Allows an action to occur in response to a keyboard key combination</td>
</tr>
<tr>
<td>Disable</td>
<td>Allows a component to be explicitly enabled or disabled</td>
</tr>
<tr>
<td>Border</td>
<td>Surrounds a component with a border</td>
</tr>
</tbody>
</table>
### Tool Tips

- **Tool tips** provide a short pop-up description when the mouse cursor rests momentarily on a component.
- A tool tip is assigned using the `setToolTipText` method of a Swing component.

```java
JButton button = new JButton("Compute");
button.setToolTipText("Calculate size.");
```

### Mnemonics

- A **mnemonic** provides a keyboard alternative for pushing a button or selecting a menu option.
- The mnemonic character should be chosen from the component’s label, and is underlined.
- The user activates the component by holding down the ALT key and pressing the mnemonic character.
- A mnemonic is established using the `setMnemonic` method.

```java
JButton button = new JButton("Calculate");
button.setMnemonic("C");
```
Disabled Components

- Components can be disabled if they should not be used
- A disabled component is "grayed out" and will not respond to user interaction
- The status is set using the `setEnabled` method:

  ```java
  JButton button = new JButton("Do It");
  button.setEnabled(false);
  ```

Special Features

- The right combination of special features and components can enhance the usefulness of a GUI
- See `LightBulb.java`
**Borders**

- A *border* can be put around any Swing component to define how the edges of the component should be drawn.
- The *BorderFactory* class is useful for creating border objects.
- It has methods for creating specific types of borders.
- A border is applied to a component using the `setBorder` method:

```java
JPanel myPanel = new JPanel();
Border myBorder = BorderFactory.createEtchedBorder();
myPanel.setBorder(myBorder);
```

**Borders**

- An *empty border*:
  - buffers the space around the edge of a component
  - otherwise has no visual effect
- A *line border*:
  - surrounds the component with a simple line
  - the line’s color and thickness can be specified
- An *etched border*:
  - creates the effect of an etched groove around a component
  - uses colors for the highlight and shadow
Borders

- A bevel border
  - can be raised or lowered
  - uses colors for the outer and inner highlights and shadows

- A titled border
  - places a title on or around the border
  - the title can be oriented in many ways

- A matte border
  - specifies the sizes of the top, left, bottom, and right edges of the border separately
  - uses either a solid color or an image

Borders

- A compound border
  - is a combination of two borders
  - one or both of the borders can be a compound border

See BorderDemo.java
Scroll Panes

- A **scroll pane** is useful for images or information too large to fit in a reasonably-sized area.
- A scroll pane offers a limited view of the component it contains.
- It provides vertical and/or horizontal scroll bars that allow the user to scroll to other areas of the component.
- No event listener is needed for a scroll pane.
- See *TransitMap.java*
Lists

- The Swing JList class represents a list of items from which the user can choose.
- The contents of a JList object can be specified using an array of objects.
- A JList object generates a list selection event when the current selection changes.
- See PickImage.java.

Lists

- A JList object can be set so that multiple items can be selected at the same time.
- The list selection mode can be one of three options:
  - single selection – only one item can be selected at a time
  - single interval selection – multiple, contiguous items can be selected at a time
  - multiple interval selection – any combination of items can be selected
- The list selection mode is defined by a ListSelectionModel object.
**Combo Boxes**

- A *combo box* provides a menu from which the user can choose one of several options.
- Currently selected option is shown in the combo box.
- Unlike a `JList`, a combo box shows its options only when the user presses it using the mouse.
- Options can be established using an array of strings or using the `addItem` method.
- A combo box generates an action event when the user makes a selection from it.

*See [JukeBox.java](#)*

**Sliders**

- A *slider* is a component that allows the user to specify a numeric value within a bounded range.
- The `JSlider` class has several methods that allow the programmer to tailor the look of a slider.
- Major and minor tick marks, and labels on the tick marks, can be explicitly set and displayed.
- A slider produces a *change event*.

*See [ViewColors.java](#)*
Events Revisited

- Some events are generated only by certain components
- But we can set up a listener on any component for any of the following events:
  - *component event* - changing a component’s size, position, or visibility
  - *focus event* - gaining or losing the keyboard focus
  - *key event* - pressing, releasing, or clicking keyboard keys
  - *mouse event* - clicking the mouse button and moving the mouse onto and off of a component
  - *mouse motion event* - moving or dragging a mouse over a component

More About GUIs

- We’ve only scratched the surface of GUIs – but we’ve established the core issues
- There are several other components and events to use
- Java supports *menus* and submenus
- A *tool bar* is a container that groups several components into a row or column
- An *internal frame* is a container that operates like a regular frame, but only within another window
More About GUIs

- A *layered panel* is a container that takes depth into consideration
- A *progress bar* indicates the progress of an activity
- A *table* displays data in table format
- A *tree* presents data in a hierarchical format
- Java also provides rich support for *text processing*

CIS 211 Final Exam

- Thursday, March 18, 3:15 PM
- Bring your Lewis&Loftus text book
- Final is 30% of grade
- Final is comprehensive
  - Emphasis on material since midterm
Final Exam Topics

- **Inheritance**
  - Subclasses, Super classes, Abstract classes, Interfaces
  - Overriding, overloading, polymorphism
  - Class hierarchies
  - Use of super

- **Software Design**
  - Software development cycle
  - Software reuse
  - Object relationships

Final Exam Topics

- **Java Graphical Programming**
  - GUI Components
  - Containers
  - Events and Listeners
    - Interfaces and Adapters
  - Forms of input
  - Applets versus Applications
Final Exam Topics

- Other topics
  - Unix commands
  - java, javac, javadoc
  - Exceptions
    - Syntax of try-catch, throw
  - UML Diagrams

Final Exam Format

- Multiple choice
  - Read the questions carefully!
- Design questions
  - Hierarchy of classes for a problem
  - Placement of data and methods in hierarchy
- Coding questions
  - Modifying or extending existing code
  - Coding new classes and methods