Graphical Applications
- So far we’ve focused on command-line applications, which interact with the user using simple text prompts.
- In this lecture, we’re beginning to explore Java applications that have graphical components.
- These components will serve as a foundation to programs that have true graphical user interfaces (GUIs).

GUI Components
- A GUI component is an object that represents a screen element such as a button or a text field.
- GUI-related classes are defined primarily in the java.awt and the javax.swing packages.
- The Abstract Windowing Toolkit (AWT) was the original Java GUI package.
- The Swing package provides additional and more versatile components.
- Both packages are needed to create a Java GUI-based program.

GUI Containers
- A GUI container is a component that is used to hold and organize other components.
- A frame is a container that is used to display a GUI-based Java application.
- A frame is displayed as a separate window with a title bar – it can be repositioned and resized on the screen as needed.
- A panel is a container that cannot be displayed on its own but is used to organize other components.
- A panel must be added to another container to be displayed.

GUI Containers
- A GUI container can be classified as either heavyweight or lightweight.
- A heavyweight container is one that is managed by the underlying operating system.
- A lightweight container is managed by the Java program itself.
- Occasionally this distinction is important.
- A frame is a heavyweight container and a panel is a lightweight container.
Graphical User Interfaces

- A Graphical User Interface (GUI) in Java is created with at least three kinds of objects:
  - components
  - events
  - listeners

- First, components are objects that represent screen elements
  - labels, buttons, text fields, menus, etc.

- Some components are containers that hold and organize other components
  - frames, panels, applets, dialog boxes

Labels

- A label is a GUI component that displays a line of text
- Labels are usually used to display information or identify other components in the interface
- DEMO

Buttons

- A push button is a component that allows the user to initiate an action by pressing a graphical button using the mouse
- A push button is defined by the JButton class
- It generates an action event
- The PushCounter example displays a push button that increments a counter each time it is pushed
- DEMO

Text Fields

- Let’s look at another GUI example that uses another type of component
- A text field allows the user to enter one line of input
- If the cursor is in the text field, the text field component generates an action event when the enter key is pressed
- DEMO

Dialog Boxes

- A dialog box is a window that appears on top of any currently active window
- It may be used to:
  - convey information
  - confirm an action
  - allow the user to enter data
  - pick a color
  - choose a file
- A dialog box usually has a specific, solitary purpose, and the user interaction with it is brief
Dialog Boxes

- The JOptionPane class provides methods that simplify the creation of some types of dialog boxes
- DEMO

Check Boxes

- A check box is a button that can be toggled on or off
- It is represented by the JCheckBox class
- Unlike a push button, which generates an action event, a check box generates an item event whenever it changes state (is checked on or off)
- The ItemListener interface is used to define item event listeners
- The check box calls the itemStateChanged method of the listener when it is toggled

Radio Buttons

- A group of radio buttons represents a set of mutually exclusive options – only one can be selected at any given time
- When a radio button from a group is selected, the button that is currently "on" in the group is automatically toggled off
- To define the group of radio buttons that will work together, each radio button is added to a ButtonGroup object
- A radio button generates an action event

File Choosers

- Situations often arise where we want the user to select a file stored on a disk drive, usually so that its contents can be read and processed
- A file chooser, represented by the JFileChooser class, simplifies this process
- The user can browse the disk and filter the file types displayed
- DEMO

Sliders

- A slider is a GUI component that allows the user to specify a value within a numeric range
- A slider can be oriented vertically or horizontally and can have optional tick marks and labels
- The minimum and maximum values for the slider are set using the JSlider constructor
- A slider produces a change event when the slider is moved, indicating that the slider and the value it represents has changed
- DEMO

Nested Panels

- Containers that contain other components make up the containment hierarchy of an interface
- This hierarchy can be as intricate as needed to create the visual effect desired
- DEMO
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

Split Panes
- A split pane (JSplitPane) is a container that displays two components separated by a moveable divider bar
- The two components can be displayed side by side, or one on top of the other

Split Panes
- The orientation of the split pane is set using the HORIZONTAL_SPLIT or VERTICAL_SPLIT constants
- The divider bar can be set so that it can be fully expanded with one click of the mouse
- The components can be continuously adjusted as the divider bar is moved, or wait until it stops moving
- Split panes can be nested

Events
- An event is an object that represents some activity to which we may want to respond
- For example, we may want our program to perform some action when the following occurs:
  - the mouse is moved
  - the mouse is dragged
  - a mouse button is clicked
  - a graphical button is clicked
  - a keyboard key is pressed
  - a timer expires
- Events often correspond to user actions, but not always

Events and Listeners
- The Java standard class library contains several classes that represent typical events
- Components, such as a graphical button, generate (or fire) an event when it occurs
- A listener object “waits” for an event to occur and responds accordingly
- We can design listener objects to take whatever actions are appropriate when an event occurs
# Events and Listeners

A component object may generate an event. A corresponding listener object is designed to respond to the event. When the event occurs, the component calls the appropriate method of the listener, passing an object that describes the event.

## Mouse Events

- Events related to the mouse are separated into mouse events and mouse motion events.

### Mouse Events:

<table>
<thead>
<tr>
<th>Mouse Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mouse pressed</td>
<td>the mouse button is pressed down</td>
</tr>
<tr>
<td>mouse released</td>
<td>the mouse button is released</td>
</tr>
<tr>
<td>mouse clicked</td>
<td>the mouse button is pressed down and released</td>
</tr>
<tr>
<td>mouse in between</td>
<td>released without moving the mouse</td>
</tr>
<tr>
<td>mouse entered</td>
<td>the mouse pointer is moved onto (over) a component</td>
</tr>
<tr>
<td>mouse exited</td>
<td>the mouse pointer is moved off of a component</td>
</tr>
</tbody>
</table>

- **Rubberbanding** is the visual effect in which a shape is "stretched" as it is drawn using the mouse.

## Key Events

- A key event is generated when the user types on the keyboard.

<table>
<thead>
<tr>
<th>Key Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key pressed</td>
<td>a key on the keyboard is pressed down</td>
</tr>
<tr>
<td>key released</td>
<td>a key on the keyboard is released</td>
</tr>
<tr>
<td>key typed</td>
<td>a key on the keyboard is pressed down and released</td>
</tr>
</tbody>
</table>

- Listeners for key events are created by implementing the `KeyListener` interface.
- A `KeyEvent` object is passed to the appropriate method when a key event occurs.

---

**Notes:**

- Mouse Events:
  - Mouse Motion Events:
    - mouse moved: the mouse is moved
    - mouse dragged: the mouse is moved while the mouse button is pressed down
  - Listeners for mouse events are created using the `MouseListener` and `MouseMotionListener` interfaces.
  - A `MouseEvent` object is passed to the appropriate method when a mouse event occurs.

- Key Events:
  - A key event is generated when the user types on the keyboard.
  - Listeners for key events are created by implementing the `KeyListener` interface.
  - A `KeyEvent` object is passed to the appropriate method when a key event occurs.
Key Events

- The component that generates a key event is the one that has the current keyboard focus
- Constants in the KeyEvent class can be used to determine which key was pressed

Layout Managers

- A layout manager is an object that determines the way that components are arranged in a container
- There are several predefined layout managers defined in the Java standard class library:
  - Flow Layout
  - BorderLayout
  - Card Layout
  - Grid Layout
  - GridBag Layout
  - Box Layout
  - Overlay Layout

  Defined in the AWT
  Defined in Swing

Layout Managers

- 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

Flow Layout

- Flow layout puts as many components as possible on a row, 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
- Also, the horizontal and vertical gaps between the components can be explicitly set

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());
  ```
Border Layout

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

```
+-----------------+-------+-----------------+
| North           | West  | East            |
|                 |       |                 |
|                 | Center|                 |
|                 |       |                 |
| South           |       |                 |
```

- Each area displays one component (which could be a 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

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

Box Layout

- A box layout organizes components 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

- 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

Borders

- A border can be put around any Swing component to define how the edges of the component should be drawn
- Borders can be used effectively to group components visually
- The BorderFactory class contains several static methods for creating border objects
- A border is applied to a component using the setBorder method
**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

- **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

---

**GUI DEVELOPMENT**

**GUI Development**

- Generally we use components and events that are predefined by classes in the Java class library

- Therefore, to create a Java program that uses a GUI we must:
  - instantiate and set up the necessary components
  - implement listener classes for any events we care about
  - establish the relationship between listeners and components that generate the corresponding events

- Let’s now explore some new components and see how this all comes together

---

**GUI Design Principles**

- We must remember that the goal of software is to help the user solve the problem

- To that end, the GUI designer should:
  - Know the user
  - Prevent user errors
  - Optimize user abilities
  - Be consistent

- Let’s discuss each of these in more detail
Know the User

- Knowing the user implies an understanding of:
  - the user’s true needs
  - the user’s common activities
  - the user’s level of expertise in the problem domain and in computer processing
- We should also realize these issues may differ for different users
- Remember, to the user, the interface is the program

Prevent User Errors

- Whenever possible, we should design user interfaces that minimize possible user mistakes
- We should choose the best GUI components for each task
- For example, in a situation where there are only a few valid options, using a menu or radio buttons would be better than an open text field
- Error messages should guide the user appropriately

Optimize User Abilities

- Not all users are alike – some may be more familiar with the system than others
- Knowledgeable users are sometimes called power users
- We should provide multiple ways to accomplish a task whenever reasonable
  - “wizards” to walk a user through a process
  - short cuts for power users
- Help facilities should be available but not intrusive

Be Consistent

- Consistency is important – users get used to things appearing and working in certain ways
- Colors should be used consistently to indicate similar types of information or processing
- Screen layout should be consistent from one part of a system to another
- For example, error messages should appear in consistent locations

The Conditional Operator

- Java has a conditional operator that uses a boolean condition to determine which of two expressions is evaluated
- Its syntax is:

  \[
  \text{condition} \ ? \text{expression1} : \text{expression2}
  \]
- If the condition is true, expression1 is evaluated; if it is false, expression2 is evaluated
- The value of the entire conditional operator is the value of the selected expression
The Conditional Operator

- The conditional operator is similar to an if-else statement, except that it is an expression that returns a value.
- For example:
  
  ```java
  larger = ((num1 > num2) ? num1 : num2);
  ```

- If `num1` is greater than `num2`, then `num1` is assigned to `larger`; otherwise, `num2` is assigned to `larger`.
- The conditional operator is ternary because it requires three operands.

The Conditional Operator

- Another example:
  
  ```java
  System.out.println("Your change is " + count +
                   (count == 1) ? "Dime" : "Dimes");
  ```

- If `count` equals 1, then "Dime" is printed.
- If `count` is anything other than 1, then "Dimes" is printed.