

# Graphical User Interfaces (GUIs)

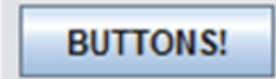


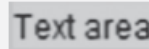

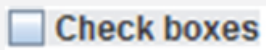





# GUIs and Swing

- GUIs are a contemporary technique to interact with computer applications
- More “user friendly” than text-based
- Swing is a toolkit to provide graphical interactive elements (“widgets”)
- Other GUI toolkits are available for Java, as described in the text

# Widgets

CLASS	USED AS	LOOKS LIKE
JButton	<pre>JComponent component = new JButton("BUTTONS!");</pre>	
JLabel	<pre>JComponent component = new JLabel("A label");</pre>	
JTextField	<pre>JComponent component = new JTextField("Text field");</pre>	
JTextArea	<pre>JComponent component = new JTextArea("Text area");</pre>	
JComboBox	<pre>JComponent component = new JComboBox&lt;String&gt;( new String[]{ "---1---", "---2---" });</pre>	
JCheckBox	<pre>JComponent component = new JCheckBox("Check boxes");</pre>	
JRadioButton	<pre>JComponent component = new JRadioButton( "And radio buttons");</pre>	

# Event Handling

- Interacting with a widget causes an event object to be created and dispatched
- Handling that event involves writing code to “listen” for specific events
- The listener must be registered to the widget
- Different widgets cause different events
- Event listeners must implement the corresponding interface and methods

# Event Handling (continued)

- JButton, JCheckBox, JComboBox, JRadioButton
  - Fires:(ActionEvent) event objects
  - Listener: ActionListener interface
  - Register using: addActionListener method
- Any component
  - Fires: MouseEvent events for clicks
  - Listener: MouseListener interface
  - Register using: addMouseListener method
  - See also
    - MouseMotionListener for mouse movement
    - KeyListener for keyboard events (e.g., pressing a key)

# JFrame

- Represents a GUI window with:
  - A title bar
  - A (possibly) resizable border
  - Buttons to minimize, maximize, and close
- GUI-based applications “extend” this class to take advantage of its existing features, and add customizations



# JPanel

- A container used to arrange widgets
- Can also contain other panels
- Alternatively, can be used to “draw” custom shapes and other graphic elements



# paintComponent

- This method is called when its component needs “repainting” (e.g., when its state or appearance changes)
- Can be overridden (re-defined) to create a custom appearance (e.g., draw shapes)
- Do not call this method directly
- Call `repaint()` instead





# MouseClicked Example

- Demonstrates:
  - Basic principles of inheritance
  - The purpose of overriding the `paintComponent` method
  - Handling events by registering a listener
- Code presented in lecture

# Layout Managers

- Every component must have its position and size specified and updated – very tedious
- Layout managers automate this process
- Each manager arranges components using
  - Specifications from the programmer
  - The state of each component
  - Its own style rules
- Each JPanel has a layout manager
  - JPanel can contain zero or more JPanel objects, each with a different manager to create complex layouts

# FlowLayout

- Arranged left-to-right, top-to-bottom
- “Flow” to the next line if needed
- Sizes based on “preferred” size



FIGURE 11-9

# BorderLayout

- Components added using compass directions
- Ignores preferred size

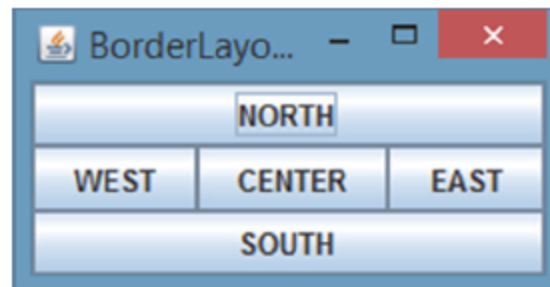


FIGURE 11-10

# GridLayout

- Components arranged in a grid
- Ignores preferred size



FIGURE 11-12

# BoxLayout

- Like FlowLayout, but with more options:
  - Specify alignment
  - Specify spacer components that can stretch or stay rigid to accommodate resizing the JFrame

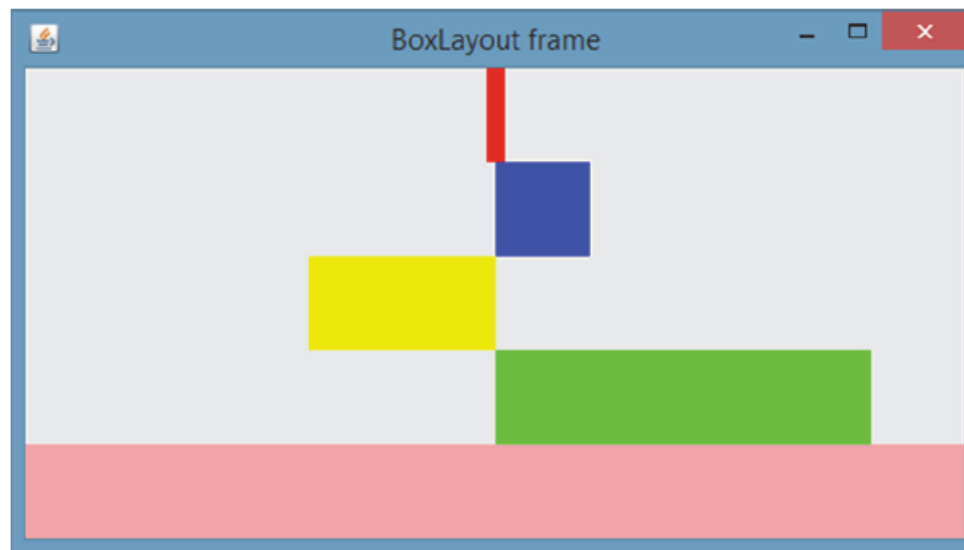


FIGURE 11-17

# Additional Examples

- A visual guide to layout managers
  - <https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html>
- FlowLayout
  - <https://docs.oracle.com/javase/tutorial/uiswing/layout/flow.html>
- BorderLayout
  - <https://docs.oracle.com/javase/tutorial/uiswing/layout/border.html>
- GridLayout
  - <https://docs.oracle.com/javase/tutorial/uiswing/layout/grid.html>
- BoxLayout
  - <https://docs.oracle.com/javase/tutorial/uiswing/layout/box.html>

# Separating View from Logic

- Listener code can be very lengthy
- For organization and maintenance, separate widget layout code from listener code
- The “model-view-controller” (MVC) pattern separates code based on responsibility:
  - Model: the current state of the application
  - View: the appearance of the application (GUI layout)
  - Controller: how the GUI behaves (the listener(s))
- MVC concept developed further in EECS 2030