JAVA™ PROGRAMMING

Chapter 15: Advanced GUI Topics

Objectives

- Use content panes
- Use color
- Learn more about layout managers
- Use `JPanel`s to increase layout options
- Create `JScrollPane`s

Objectives (cont'd.)

- Understand events and event handling more thoroughly
- Use the `AWTEvent` class methods
- Handle mouse events
- Use menus

Understanding the Content Pane

- **Top-level container**
  - `JFrame`
  - Contains a **content pane**, **menu bar**, and **glass pane**
- **Containment hierarchy**
  - A tree of components that has a top-level container as its root
- **Content pane**
  - Contains all the visible components in the container's user interface

Understanding the Content Pane (cont'd.)

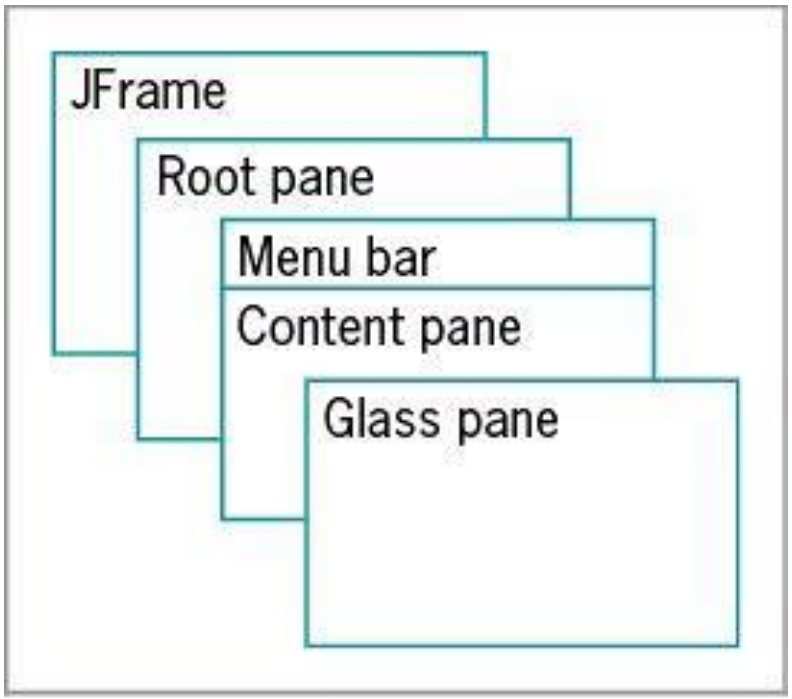

Figure 15-1 Parts of a `JFrame`

Understanding the Content Pane (cont'd.)

- **`getContentPane()` method**
  - You can add components directly to `JFrame`s without explicitly calling `getContentPane()` using the `add()` method
    - You can use the `remove()` and `setLayout()` methods also
  - Other methods must use `getContentPane()`
    - `setBackground()`

Using Color

- **`Color` class**
  - Defines colors for you to use in your applications
  - Use with `setBackground()` and `setForeground()`
  - Defines named constants that represent 13 colors
- Create your own `Color` object
  - `Color someColor = new Color(r, g, b);`
- Discover the red, green, or blue components of any existing color with `getRed()`, `getGreen()`, and `getBlue()`

Learning More About Layout Managers

- Layout manager
  - Controls the size and position of components inside a `Container` object
  - Determines how the components are sized and positioned within it
  - Is an interface class that is part of Java SDK
  - Aligns components so that they do not:
    - Crowd each other
    - Overlap

Learning More About Layout Managers (cont'd.)

- Layout manager (cont'd.)
  - Arranges components within a `Container`
    - Each component you place within a `Container` can also be a `Container` itself
    - You can assign layout managers within layout managers
- Java platform–supplied layout managers
  - `FlowLayout` and `GridLayout`
  - `BorderLayout` and `CardLayout`
  - `GridBagLayout` and `BoxLayout`

Learning More About Layout Managers (cont'd.)

Layout Manager	When to Use
`BorderLayout`	Use when you add components to a maximum of five sections arranged in north, south, east, west, and center positions
`FlowLayout`	Use when you need to add components from left to right; `FlowLayout` automatically moves to the next row when needed, and each component takes its preferred size
`GridLayout`	Use when you need to add components into a grid of rows and columns; each component is the same size
`CardLayout`	Use when you need to add components that are displayed one at a time
`BoxLayout`	Use when you need to add components into a single row or a single column
`GridBagLayout`	Use when you need to set size, placement, and alignment constraints for every component that you add

Table 15-2 Java layout managers

Using `BorderLayout`

- **`BorderLayout` manager**
  - The default for all content panes
  - Use with any container that has five or fewer components
    - Component containers can hold more components
  - Components fill the screen in five regions:
    - North
    - South
    - East
    - West
    - Center

Using `BorderLayout` (cont'd.)

Figure 15-6 Output of the `JDemoBorderLayout` application

Using `FlowLayout`

- **`FlowLayout` manager**
  - Arranges components in rows across the width of a `Container`
  - When you add a `Component`:
    - It is placed to the right of previously added components in a row
    - If the current row is filled, the `Component` is placed to start a new row
  - Each `Component` retains its **preferred size**

Using `FlowLayout` (cont'd.)

- You can use three constants to align `Components` with a `Container`:
  - `FlowLayout.LEFT`
  - `FlowLayout.CENTER`
  - `FlowLayout.RIGHT`
- If the alignment is not specified, `Components` are center-aligned

Using `FlowLayout` (cont'd.)

- `invalidate()`
  - Marks the container as needing to be laid out
- `validate()`
  - Causes components to be rearranged based on the newly assigned layout

Using `GridLayout`

- **`GridLayout` manager** class
  - Arranges components into equal rows and columns
- When you create a `GridLayout` object:
  - Indicate the numbers of rows and columns
    - Specify rows first and then columns
  - The container surface is divided into a grid
- The following statement establishes a `GridLayout`:

```
con.setLayout(new GridLayout(4, 5));
```

Using `GridLayout` (cont'd.)

- The following statement establishes a `GridLayout` with three horizontal rows, two vertical columns, and vertical gaps of five pixels each:

```
private GridLayout layout = new
GridLayout(3, 2, 5, 5);
```

- You can use 0 for the number of columns or rows

Using `GridLayout` (cont'd.)

Figure 15-11 Output of the `JDemoGridLayout` program

Using `CardLayout`

- **`CardLayout` manager**
  - Generates a stack of containers or components one on top of another
  - Each component in the group is referred to as a card
  - Multiple components share the same display space

Using `CardLayout` (cont'd.)

- To create a card layout, use one of two constructors:
  - `CardLayout()`
    - Creates a card layout without a horizontal or vertical gap
  - `CardLayout(int hgap, int vgap)`
    - Creates a card layout with the specified horizontal and vertical gaps
  - To add a component to a content pane, use:

```
add(aString, aContainer);
```

Using `CardLayout` (cont'd.)

Figure 15-13 Output of `JDemoCardLayout` when it first appears on the screen, after the user clicks or taps once, and after the user clicks or taps twice

Using Advanced Layout Managers

- **`GridBagLayout` manager**
  - Adds `Components` to precise locations within the grid
  - Indicates that specific `Components` should span multiple rows or columns within the grid
  - You must set the position and size for each component
  - You must customize one or more `GridBagConstraints` objects
- **`BoxLayout` manager**
  - Allows multiple components to be laid out either vertically or horizontally

Using the `JPanel` Class

- **`JPanel`**
  - A plain, borderless surface
  - Can hold lightweight UI components
- **Double buffering**
  - Additional memory space will be used to draw the `JPanel` offscreen when it is updated

Using the `JPanel` Class (cont'd.)

- Constructors
  - `JPanel()`
  - `JPanel(LayoutManager layout)`
  - `JPanel(Boolean isDoubleBuffered)`
  - `JPanel(LayoutManager layout, Boolean isDoubleBuffered)`
- To add a component to `JPanel`:
  - Call the container's `add()` method
  - Use the component as an argument

Using the `JPanel` Class (cont'd.)

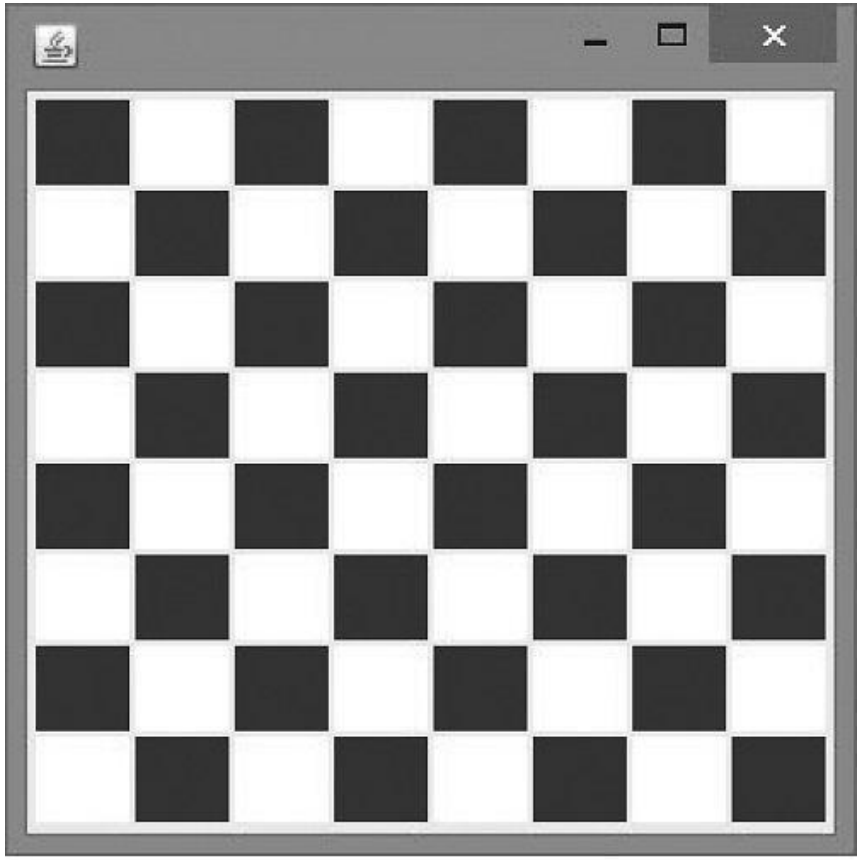

Figure 15-26 Output of the Checkerboard application

Creating `JScrollPanes`

- **`JScrollPane`**
  - Provides scroll bars along the side or bottom of a pane
  - Enables the user to scroll initially invisible parts of the pane into view
- Constructors
  - `JScrollPane()`
  - `JScrollPane(Component)`
  - `JScrollPane(Component, int, int)`
  - `JScrollPane(int, int)`

Creating JScrollPanes (cont'd.)

- To force the display of the scroll bar, use the following ScrollPaneConstants class variables:
  - ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED
  - ScrollPaneConstants.HORIZONTAL_SCROLLBAR_ALWAYS
  - ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER
  - ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED
  - ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS
  - ScrollPaneConstants.VERTICAL_SCROLLBAR_NEVER

Creating JScrollPanes (cont'd.)

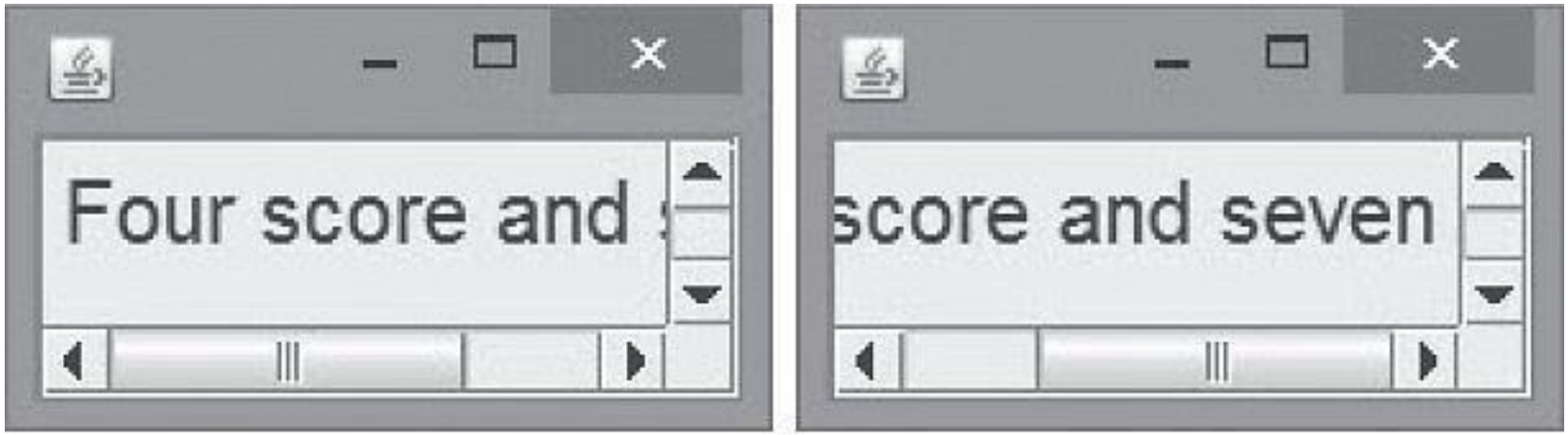

Figure 15-29 Output of the JScrollDemo application

A Closer Look at Events and Event Handling

- Events
  - Objects that the user initiates
- `EventObject`
  - The parent class for all event objects
  - Descends from the `Object` class
  - The parent of the `AWTEvent` class
    - `AWTEvent` is the parent of `ActionEvent` and `ComponentEvent`

A Closer Look at Events and Event Handling (cont'd.)

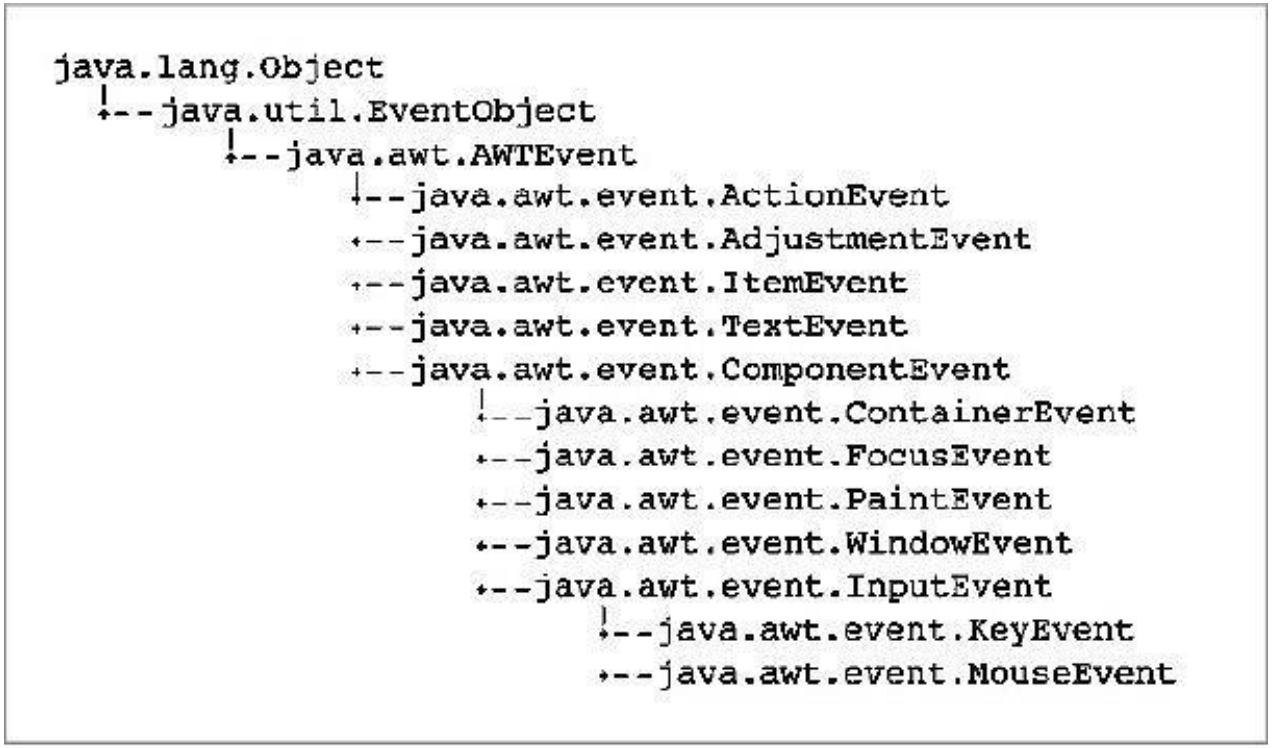

Figure 15-30 The inheritance hierarchy of event classes

A Closer Look at Events and Event Handling (cont'd.)

- `ActionEvents`
  - Focus on changes in a component
- `MouseEvents`
  - Focus on what the user does manually with the mouse
- The computer's operating system notifies the user when an `AWTEvent` occurs
  - You can ignore `AWTEvents`
  - You must implement an appropriate interface for your class to receive events

A Closer Look at Events and Event Handling (cont'd.)

User Action	Resulting Event Type
Click a button	`ActionEvent`
Click a component	`MouseEvent`
Click an item in a list box	`ItemEvent`
Click an item in a check box	`ItemEvent`
Change text in a text field	`TextEvent`
Open a window	`WindowEvent`
Iconify a window	`WindowEvent`
Press a key	`KeyEvent`

Table 15-3 Examples of user actions and their resulting event types

A Closer Look at Events and Event Handling (cont'd.)

- Event handler
  - An interface method such as `actionPerformed()`
  - Called automatically when an appropriate event occurs
- **Adapter class**
  - Implements all methods in an interface
    - Provides an empty body for each method
- When you extend the adapter class, you need to write only the methods you want to use
  - Do not bother creating empty methods for the others

A Closer Look at Events and Event Handling (cont'd.)

- You create an event handler when you write code for the listener methods
  - Tell the class how to handle events
- You must register an instance of the class with the component that the event affects
  - For any `<name>Listener`, use:

```
object.add<name>Listener(Component)
```

Event	Listener(s)	Handler(s)
ActionEvent	ActionListener	actionPerformed(ActionEvent)
ItemEvent	ItemListener	itemStateChanged(ItemEvent)
TextEvent	TextListener	textValueChanged(TextEvent)
AdjustmentEvent	AdjustmentListener	adjustmentValueChanged (AdjustmentEvent)
ContainerEvent	ContainerListener	componentAdded(ContainerEvent) componentRemoved(ContainerEvent)
ComponentEvent	ComponentListener	componentMoved(ComponentEvent) componentHidden(ComponentEvent) componentResized(ComponentEvent) componentShown(ComponentEvent)
FocusEvent	FocusListener	focusGained(FocusEvent) focusLost(FocusEvent)
MouseEvent	MouseListener MouseMotionListener	mousePressed(MouseEvent) mouseReleased(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mouseClicked(MouseEvent) mouseDragged(MouseEvent) mouseMoved(MouseEvent)
KeyEvent	KeyListener	keyPressed(KeyEvent) keyTyped(KeyEvent) keyReleased(KeyEvent)
WindowEvent	WindowListener	windowActivated(WindowEvent) windowClosing(WindowEvent) windowClosed(WindowEvent) windowDeiconified(WindowEvent) windowIconified(WindowEvent) windowOpened(WindowEvent)
MouseWheelEvent	MouseWheelListener	mouseWheelMoved(MouseWheelEvent)

Table 15-4 Events with their related listeners and handlers

An Event-Handling Example: `KeyListener`

- **`KeyListener` interface**
  - Use to receive actions that the user initiates from the keyboard
- `KeyListener` contains three methods:
  - `keyPressed()`
  - `keyTyped()`
    - Use to discover which character was typed
    - Does not execute calls from action keys
  - `keyReleased()`
    - Does not take action while the user holds down the key

An Event-Handling Example: `KeyListener` (cont'd.)

- `KeyEvent` class
  - Contains **virtual key codes**
    - Represent keyboard keys that have been pressed
    - Virtual key code constants have names such as `VK_SHIFT` and `VK_ALT`

Using `AWTEvent` Class Methods

- `AWTEvent` classes
  - Contain methods that return information about an event
  - All `Component`s have these methods:
    - `addComponentListener()`
    - `addFocusListener()`
    - `addMouseListener()`
    - `addMouseMotionListener()`

Class	Method	Purpose
EventObject	Object getSource()	Returns the Object involved in the event
ComponentEvent	Component getComponent()	Returns the Component involved in the event
WindowEvent	Window getWindow()	Returns the Window involved in the event
ItemEvent	Object getItem()	Returns the Object that was selected or deselected
ItemEvent	int getStateChange()	Returns an integer named ItemEvent.SELECTED or ItemEvent.DESELECTED
InputEvent	int getModifiers()	Returns an integer to indicate which mouse button was clicked
InputEvent	int getWhen()	Returns a time indicating when the event occurred
InputEvent	boolean isAltDown()	Returns whether the Alt key was pressed when the event occurred
InputEvent	boolean isControlDown()	Returns whether the Ctrl key was pressed when the event occurred
InputEvent	boolean isShiftDown()	Returns whether the Shift key was pressed when the event occurred
KeyEvent	int getKeyChar()	Returns the Unicode character entered from the keyboard
MouseEvent	int getClickCount()	Returns the number of mouse clicks; lets you identify the user's double-clicks
MouseEvent	int getX()	Returns the x-coordinate of the mouse pointer
MouseEvent	int getY()	Returns the y-coordinate of the mouse pointer
MouseEvent	Point getPoint()	Returns the Point Object that contains the x- and y-coordinates of the mouse location

Table 15-5 Useful event class methods

Using `AWTEvent` Class Methods (cont'd.)

- To call `Event` class methods, use the object-dot-method format
  - For example, if you have a `KeyEvent` named `inputEvent` and an integer named `unicodeVal`, the following statement is valid:

```
unicodeVal = inputEvent.getKeyChar();
```

- When you use an event object within a handler method to obtain information, add a dot and the appropriate method name
- When you use an event, you can use any methods that belong to any superclass of the event

Understanding x- and y-Coordinates

- **x-axis**
  - Horizontal position
- **y-axis**
  - Vertical position
- 0, 0
  - Upper-left corner of any display
- **x-coordinate**
- **y-coordinate**

Handling Mouse Events

- **`MouseMotionListener` interface**
  - `mouseDragged()` and `mouseMoved()`
    - Detect the mouse being rolled or dragged across a component surface
- **`MouseListener` interface**
  - `mousePressed()`, `mouseClicked()`, and `mouseReleased()`
    - Analogous to keyboard event methods
  - `mouseEntered()` and `mouseExited()`
    - Inform you when the user positions the mouse over a component (entered) or moves the mouse off a component (exited)

Handling Mouse Events (cont'd.)

Method	Description
`void mouseClicked(MouseEvent e)`	Invoked when the mouse button has been clicked (pressed and released) on a component
`void mouseEntered(MouseEvent e)`	Invoked when the mouse pointer enters a component
`void mouseExited(MouseEvent e)`	Invoked when the mouse pointer exits a component
`void mousePressed(MouseEvent e)`	Invoked when a mouse button has been pressed on a component
`void mouseReleased(MouseEvent e)`	Invoked when a mouse button has been released on a component

Table 15-6 `MouseListener` methods

Handling Mouse Events (cont'd.)

Method	Description
`void mouseDragged(MouseEvent e)`	Invoked when a mouse button is pressed on a component and then dragged
`void mouseMoved(MouseEvent e)`	Invoked when the mouse pointer has been moved onto a component but no buttons have been pressed

Table 15-7 `MouseMotionListener` methods

Handling Mouse Events (cont'd.)

- **`MouseInputListener` interface**
  - Implements all methods in both `MouseListener` and `MouseMotionListener` interfaces
  - Has no methods of its own
  - Handles many different types of mouse events
- **`MouseEvent`**
  - The type of event generated by mouse manipulation
  - Contains instance methods and fields
    - Useful in describing mouse-generated events

Using Menus

- **Menus**
  - Lists of user options
- Classes:
  - `JMenuBar`
  - `JMenu`
  - `JMenuItem`
- `setJMenuBar()` method
  - Adds the `JMenuBar` to a `JFrame`
- `add()` method
  - Adds the `JMenu` to the `JMenuBar`

Using Specialized Menu Items

- `JCheckBoxMenuItem` objects appear with a check box next to them
- `JRadioButtonMenuItem` objects appear with a round radio button next to them
- `isSelected()` method
  - Determines the state of a `JCheckBoxMenuItem` or `JRadioButtonMenuItem`

Using `addSeparator()`

- Adds a horizontal line to menus in order to visually separate groups for your users
- Does not change the functionality of the menu

Using `setMnemonic()`

- **Mnemonic**
  - A key that causes an already-visible menu item to be chosen
- `setMnemonic()` method
  - Provides a shortcut menu key for any visible menu item
  - Use a different mnemonic for each menu item
- **Accelerator**
  - A key combination that causes a menu item to be chosen, whether or not it is visible
  - Only **leaf menu items** can have accelerators

You Do It

- Using `BorderLayout`
- Using Fewer than Five Components with the `BorderLayout` Manager
- Using `FlowLayout`
- Using `GridLayout`
- Using `CardLayout`
- Viewing All the Cards in `CardLayout`
- Using a Menu Bar and `JPanels`

Don’t Do It

- Don’t forget that the content pane is operating behind the scenes
- Don’t forget that when you create a custom `Color` object, 0 represents the darkest shade and 255 represents the lightest
- Don’t forget to set a layout manager
- Don’t forget to use a region when adding a component to a `BorderLayout`

Don’t Do It (cont’d.)

- Don’t use `add()` to place a `JFrame`’s menu bar
- Don’t use the same mnemonic for multiple menu items

Summary

- Layout manager
  - An object that controls the size and position of components inside a `Container` object
    - `BorderLayout`
    - `FlowLayout`
    - `GridLayout`
    - `CardLayout`
    - `GridBagLayout`
- Use `JPanels` within other `JPanels`
  - Create an infinite variety of screen layouts

Summary (cont'd.)

- Events are `Objects` that the user initiates
- Implement the appropriate listener interface for your class
- Event handlers
  - Interface methods automatically called when an event occurs
- `KeyListener` interface
  - Handles keyboard events
- `MouseListener` interface
  - Handles mouse events