Java Programming, 9e

Chapter 14

Introduction to `Swing` Components

CENGAGE

Objectives (1 of 2)

- Describe `Swing` components
- Use the `JFrame` class
- Use the `JLabel` class
- Use a layout manager
- Extend the `JFrame` class

Objectives (2 of 2)

- Add `JTextField`s and `JButton`s to a `JFrame`
- Learn about event-driven programming
- Understand `Swing` event listeners
- Use the `JCheckBox`, `ButtonGroup`, and `JComboBox` classes

Understanding `Swing` Components (1 of 3)

- **Abstract Windows Toolkit (AWT)**
  - Older framework, not as portable
- **GUI components**
  - Buttons, text fields, and other components with which the user can interact
- **`Swing` components:**
  - Part of **Java Foundation Classes** (**JFC**)
  - Are descendants of `JComponent`
  - Inherit from the `java.awt.Container` class
- To take advantage of the `Swing` GUI components and their methods, insert: `import javax.swing.*;`
- **Lightweight components** (independent of OS)
- **Heavyweight components** (interact with OS)

Understanding `Swing` Components (2 of 3)

- **Container**
  - A type of component that holds other components
  - Allows a group to be treated as a single entity
  - Defined in the `Container` class
  - Often takes the form of a window that you can:
    - Drag
    - Resize
    - Minimize
    - Restore
    - Close

Understanding `Swing` Components (3 of 3)

- **`Window`** class
  - A child of `Container`
  - Does not have title bars or borders
  - Is rarely used
  - Instead, use the following subclasses:
    - **`Frame`**
    - **`JFrame`**

Using the `JFrame` Class (1 of 6)

```
java.lang.Object
   |
   +--java.awt.Component
           |
           +--java.awt.Container
                   |
                   +--java.awt.Window
                           |
                           +--java.awt.Frame
                                   |
                                   +--javax.swing.JFrame
```

Figure 14-1 Relationship of the `JFrame` class to its ancestors

Using the `JFrame` Class (2 of 6)

- Create a `JFrame` so you can place other objects within it for display
- The `JFrame` class has four constructors:
  - `JFrame()`
  - `JFrame(String title)`
  - `JFrame(GraphicsConfiguration gc)`
  - `JFrame(String title, GraphicsConfiguration gc)`

Using the JFrame Class (3 of 6)

Table 14-1 Useful methods inherited by the JFrame class	
Method	**Purpose**
void setTitle(String)	Sets a JFrame’s title using the String argument
void setSize(int, int)	Sets a JFrame’s size in pixels with the width and height as arguments
void setSize(Dimension)	Sets a JFrame’s size using a Dimension class object; the Dimension(int, int) constructor creates an object that represents both a width and a height
String getTitle()	Returns a JFrame’s title
void setResizable(boolean)	Sets the JFrame to be resizable by passing true to the method, or sets the JFrame not to be resizable by passing false to the method
boolean isResizable()	Returns true or false to indicate whether the JFrame is resizable
void setVisible(boolean)	Sets a JFrame to be visible using the boolean argument true and invisible using the boolean argument false
void setBounds(int, int, int, int)	Overrides the default behavior for the JFrame to be positioned in the upper-left corner of the computer screen’s desktop; the first two arguments are the horizontal and vertical positions of the JFrame’s upper-left corner on the desktop, and the final two arguments set the width and height

Using the `JFrame` Class (4 of 6)

- Create `JFrame`

```
JFrame firstFrame = new JFrame("Hello");
```

- Set size and title

```
firstFrame.setSize(200, 100);
firstFrame.setTitle("My frame");
```

Using the `JFrame` Class (5 of 6)

```
import javax.swing.*;
public class JFrame1
{
   public static void main(String[] args)
   {
      JFrame aFrame = new JFrame("First frame");
      aFrame.setSize(250, 100);
      aFrame.setVisible(true);
   }
}
```

Figure 14-2 The `JFrame1` application

Using the `JFrame` Class (6 of 6)

- To close `JFrame`, click the Close button
  - Default behavior
    - `JFrame` becomes hidden and the application keeps running
  - To change this behavior, use the `setDefaultCloseOperation()` method

Customizing a `JFrame`'s Appearance (1 of 2)

- **Window decorations**
  - Icon and buttons
- **Look and feel**
  - The default appearance and behavior of a user interface
  - The `setDefaultLookAndFeelDecorated()` method sets `JFrame`'s look and feel

Customizing a `JFrame`'s Appearance (2 of 2)

```
import javax.swing.*;
public class JFrame2
{
   public static void main(String[] args)
   {
      JFrame.setDefaultLookAndFeelDecorated(true);
      JFrame aFrame = new JFrame("Second frame");
      aFrame.setSize(250, 100);
      aFrame.setVisible(true);
   }
}
```

Figure 14-4 The JFrame2 class

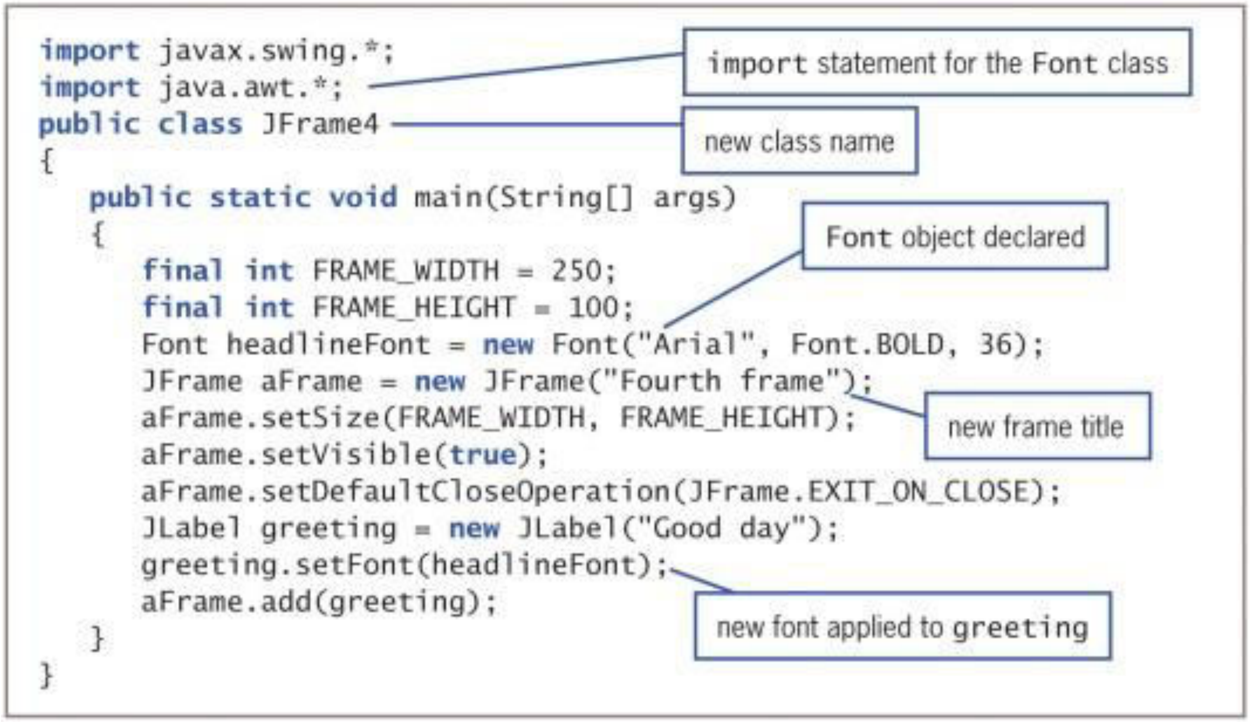
Using the `JLabel` Class

- **`JLabel`**
  - Holds text you can display
  - Six available constructors
  - Methods
    - **`add()` method**
    - **`remove()` method**
    - **`setText()` method**
    - **`getText()` method**

Changing a `JLabel`'s Font (1 of 2)

- **`Font` class**
  - Creates an object that holds typeface and size information
  - To construct a `Font` object, you need three arguments:
    - Typeface
    - Style
    - Point size
- **`setFont()` method**
  - A `Font` object argument is required

Changing a `JLabel`'s Font (2 of 2)



```
import javax.swing.*;
import java.awt.*;
public class JFrame4
{
   public static void main(String[] args)
   {
      final int FRAME_WIDTH = 250;
      final int FRAME_HEIGHT = 100;
      Font headlineFont = new Font("Arial", Font.BOLD, 36);
      JFrame aFrame = new JFrame("Fourth frame");
      aFrame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
      aFrame.setVisible(true);
      aFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      JLabel greeting = new JLabel("Good day");
      greeting.setFont(headlineFont);
      aFrame.add(greeting);
   }
}
```

import statement for the Font class
new class name
Font object declared
new frame title
new font applied to greeting

Figure 14-10 The `JFrame4` program

Using a Layout Manager

- **Layout manager**
  - A class that controls component positioning
- **`BorderLayout`**
  - The normal (default) behavior of a `JFrame`
  - Divides a container into regions
- **Flow layout manager**
  - Places components in a row

Extending the `JFrame` Class (1 of 2)

- When you create a class that descends from the `JFrame` class:
  - You can set the `JFrame`’s properties within your object’s constructor
  - Then, when the `JFrame` child object is created, it is automatically endowed with the features you specified
- Create a child class using the keyword `extends`
- Call the parent class’s constructor method using the keyword `super`

Extending the `JFrame` Class (2 of 2)

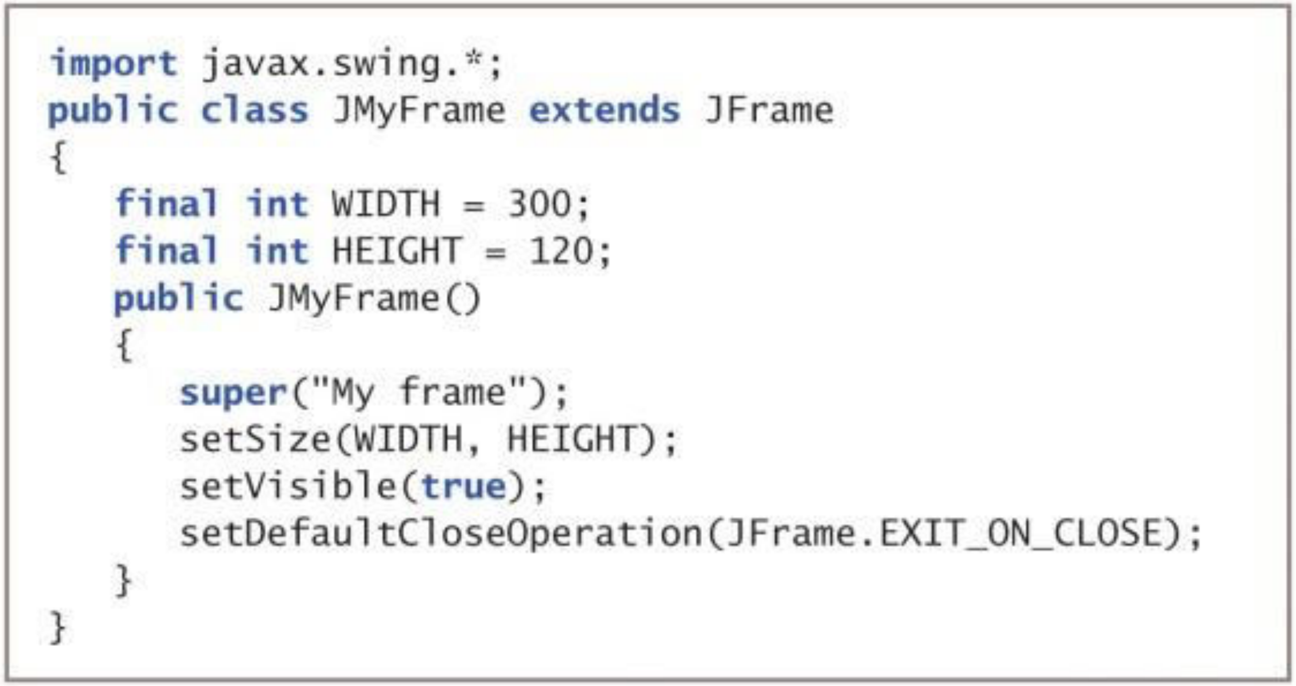

```
import javax.swing.*;
public class JMyFrame extends JFrame
{
    final int WIDTH = 300;
    final int HEIGHT = 120;
    public JMyFrame()
    {
        super("My frame");
        setSize(WIDTH, HEIGHT);
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Figure 14-16 The JMyFrame class

Adding JTextFields and JButtons to a JFrame

- In addition to including JLabel objects, JFrames often contain other window features, such as:
  - JTextFields
  - JButtons

Adding `JTextFields`

- **`JTextField`**
  - A component into which a user can type a single line of text data
  - Has several constructors
  - Methods
    - `setText()` method
    - `getText()` method
    - **`setEditable()` method**

Adding `JButtons`

- **`JButton`**
  - Click with a mouse to make a selection
  - Has five constructors
  - Methods include: `setText()` and `getText()`
- `add()` method
  - Adds a `JButton` to a `Jframe`
- When clicked, no resulting actions occur
  - The code has not yet been written to handle user-initiated events

Learning About Event-Driven Programming (1 of 2)

- **Event**
  - Occurs when a user takes action on a component, such as clicking the mouse on a `JButton` object
- **Event-driven program**
  - A program in which the user might initiate any number of events in any order
- **Source**
  - The component on which an event is generated
- **Listener**
  - The object that is interested in an event
  - **Register** the object as a listener

Learning About Event-Driven Programming (2 of 2)

- To respond to user events within any class you create, you must:
  - Prepare your class to accept event messages
  - Tell your class to expect events to happen
  - Tell your class how to respond to events

Preparing Your Class to Accept Event Messages

- Import the `java.awt.event` package
- Add the phrase `implements ActionListener` to the class header
- Implementing `ActionListener` provides you with standard event method specifications that allow your listener to work with `ActionEvents`

Telling Your Class to Expect Events to Happen

- **`addActionListener()` method**
- `aButton.addActionListener(this);`
  - Causes any `ActionEvent` messages (button clicks) that come from `aButton` to be sent to “this current object”

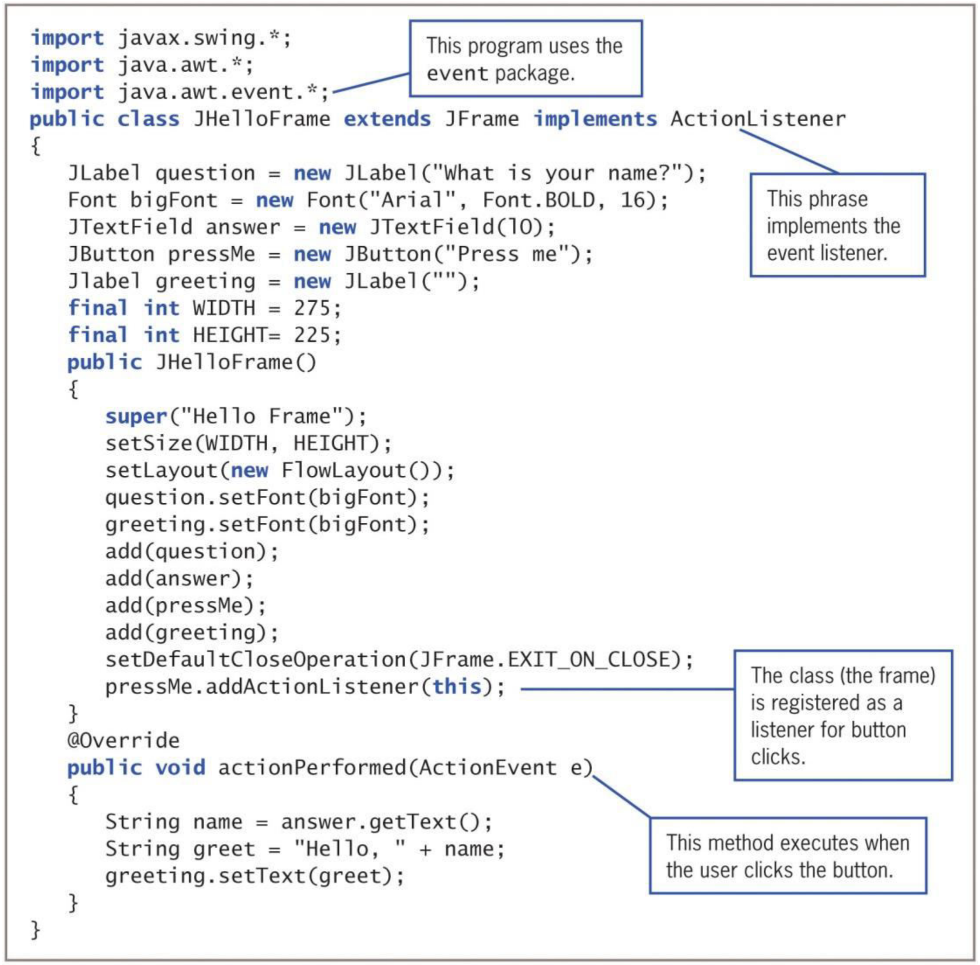
Telling Your Class How to Respond to Events

- The `ActionListener` interface contains the **`actionPerformed(ActionEvent e)` method** specification
  - **Event handler**
  - The body of the method contains any statements that you want to execute when the action occurs
- When more than one component is added and registered to a `JFrame`, it might be necessary to determine which component was used
  - Find the source of the event using `getSource();`

An Event-Driven Program (1 of 2)

- Figure 14-25 shows a `JFrame` that reacts to a button click
  - Import the `event` package
  - Within the `actionPerformed()` method, the `String` that a user has typed into the `JTextField` is retrieved and stored in the `name` variable and then used in the text of a second `JLabel`

An Event-Driven Program (2 of 2)



```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class JHelloFrame extends JFrame implements ActionListener
{
   JLabel question = new JLabel("What is your name?");
   Font bigFont = new Font("Arial", Font.BOLD, 16);
   JTextField answer = new JTextField(10);
   JButton pressMe = new JButton("Press me");
   Jlabel greeting = new JLabel("");
   final int WIDTH = 275;
   final int HEIGHT= 225;
   public JHelloFrame()
   {
      super("Hello Frame");
      setSize(WIDTH, HEIGHT);
      setLayout(new FlowLayout());
      question.setFont(bigFont);
      greeting.setFont(bigFont);
      add(question);
      add(answer);
      add(pressMe);
      add(greeting);
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      pressMe.addActionListener(this);
   }
   @Override
   public void actionPerformed(ActionEvent e)
   {
      String name = answer.getText();
      String greet = "Hello, " + name;
      greeting.setText(greet);
   }
}
```

This program uses the event package.

This phrase implements the event listener.

The class (the frame) is registered as a listener for button clicks.

This method executes when the user clicks the button.

Figure 14-25 The JHelloFrame class that produces output when the user clicks the JButton

Using Multiple Event Sources (1 of 3)

- Can add more than one event source to a listener
- Figure 14-28 shows a `JFrame` that reacts to either of two buttons
- Alternatively, you can use `instanceof` to determine the event source (Figure 14-29)

Using Multiple Event Sources (2 of 3)

```
@Override
public void actionPerformed(ActionEvent e)
{
   Object source = e.getSource();
   if(source == option1)
      //execute these statements when user clicks option1
   else
      //execute these statements when user clicks any other option
}
```

Figure 14-28 An `actionPerformed()` method that takes one of two possible actions

Using Multiple Event Sources (3 of 3)

```
@Override
void actionPerformed(ActionEvent e)
{
   Object source = e.getSource();
   if(source instanceof JTextField)
   {
      // execute these statements when any JTextField
      // generates the event
      // but not when a JButton or other Component does
   }
}
```

Figure 14-29 An `actionPerformed()` method that executes a block of statements when a user generates an event from any `JTextField`

Using the `setEnabled()` Method

- **`setEnabled()` method**
  - Makes a component unavailable, and then makes it available again in turn
  - Use after a specific series of actions has taken place

Understanding `Swing` Event Listeners (1 of 6)

- Classes that respond to user-initiated events must implement an interface that deals with events called event listeners
- Many types of listeners exist in Java
  - Each can handle a specific event type
- A class can implement as many event listeners as it needs
- An event occurs every time the user types a character or clicks the mouse button

Understanding `Swing` Event Listeners (2 of 6)

Table 14-2 Alphabetical list of some event listeners		
Listener	**Type of Events**	**Example**
ActionListener	Action events	Button clicks
AdjustmentListener	Adjustment events	Scroll bar moves
ChangeListener	Change events	Slider is repositioned
FocusListener	Keyboard focus events	Text field gains or loses focus
ItemListener	Item events	Check box changes status
KeyListener	Keyboard events	Text is entered
MouseListener	Mouse events	Mouse clicks
MouseMotionListener	Mouse movement events	Mouse rolls
WindowListener	Window events	Window closes

Understanding Swing Event Listeners (3 of 6)

- Create relationships between Swing components and classes that react to users' manipulations of them
- JCheckBox responds to the user's clicks
  - addItemListener() method
  - Register JCheckBox as a type of object that can create an ItemEvent
  - Format

```
theSourceOfTheEvent.addListenerMethod
(theClassThatShouldRespond);
```

Understanding Swing Event Listeners (4 of 6)

Table 14-3 Some Swing components and their associated listener-registering methods	
Component(s)	**Associated Listener-Registering Method(s)**
JButton, JCheckBox, JComboBox, JTextField, and JRadioButton	addActionListener()
JScrollBar	addAdjustmentListener()
All Swing components	addFocusListener(), addKeyListener(), addMouseListener(), and addMouseMotionListener()
JButton, JCheckBox, JComboBox, and JRadioButton	addItemListener()
All JWindow and JFrame components	addWindowListener()
JSlider and JCheckBox	addChangeListener()

Understanding `Swing` Event Listeners (5 of 6)

- The class of the object that responds to an event contains a method that accepts the event object created by the user’s action
  - Specific methods react to specific event types
- If you declare a class that handles an event, create a class to do one of the following:
  - Implement a listener interface
  - Extend a class that implements a listener interface

Understanding `Swing` Event Listeners (6 of 6)

- If you declare a class that extends `MyFrame`, you need not include `implements ItemListener` in its header
- You must register each instance of the event-handling class as a listener for one or more components

Using the JCheckBox, ButtonGroup, and JComboBox Classes

- Besides JButtons and JTextFields, several other Java components allow a user to make selections in a GUI environment

The `JCheckBox` Class (1 of 4)

- **`JCheckBox`**
  - **Check box** consists of a label positioned beside a square
  - Click the square to display or remove a check mark
  - Use to allow the user to turn an option on or off
- Constructors

```
JCheckBox()
JCheckBox("Check here")
JCheckBox("Check here", false)
```

The `JCheckBox` Class (2 of 4)

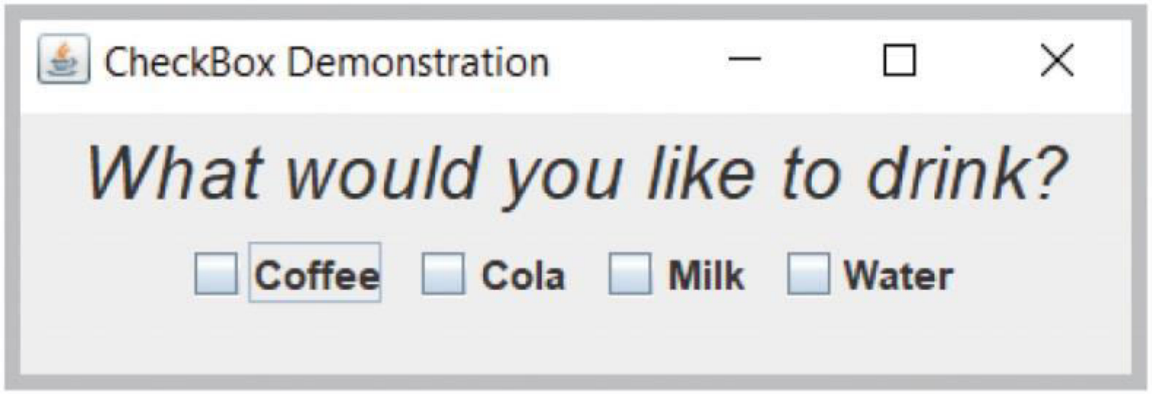


Figure 14-33 Execution of the `CheckBoxDemonstration` class

The JCheckBox Class (3 of 4)

Table 14-5 Frequently used JCheckBox methods	
Method	**Purpose**
void setText(String)	Sets the text for the JCheckBox
String getText()	Returns the JCheckBox text
void setSelected(boolean)	Sets the state of the JCheckBox to true for selected or false for unselected
boolean isSelected()	Gets the current state (checked or unchecked) of the JCheckBox

CENGAGE

The `JCheckBox` Class (4 of 4)

- Methods
  - `setText()`
  - `setSelected()`
  - `isSelected()`
- When the status of `JCheckBox` changes from unchecked to checked:
  - An `ItemEvent` is generated
  - The `itemStateChanged()` method executes

The `ButtonGroup` Class (1 of 2)

- **`ButtonGroup`**
  - Groups several components so that the user can select only one at a time
- When you group `JCheckBox` objects, all of the other `JCheckBox`es are automatically turned off when the user selects any one check box

The `ButtonGroup` Class (2 of 2)

- To create a `ButtonGroup` in a `JFrame` and then add `JCheckBox`:
  - Create a `ButtonGroup`

    ```
    ButtonGroup aGroup = new ButtonGroup();
    ```

  - Create a `JCheckBox`

    ```
    JCheckBox aBox = new JCheckBox();
    ```

  - Add `aBox` to `aGroup`

    ```
    aGroup.add(aBox);
    ```

The `JComboBox` Class (1 of 5)

- **`JComboBox`**
  - **Combo box** is a component that combines two features:
    - A display area showing a default option
    - A list box containing additional , alternate options
  - When the user clicks the `JComboBox`, a list of alternative items drops down
    - If the user selects one, it replaces the box's displayed item

The JComboBox Class (2 of 5)

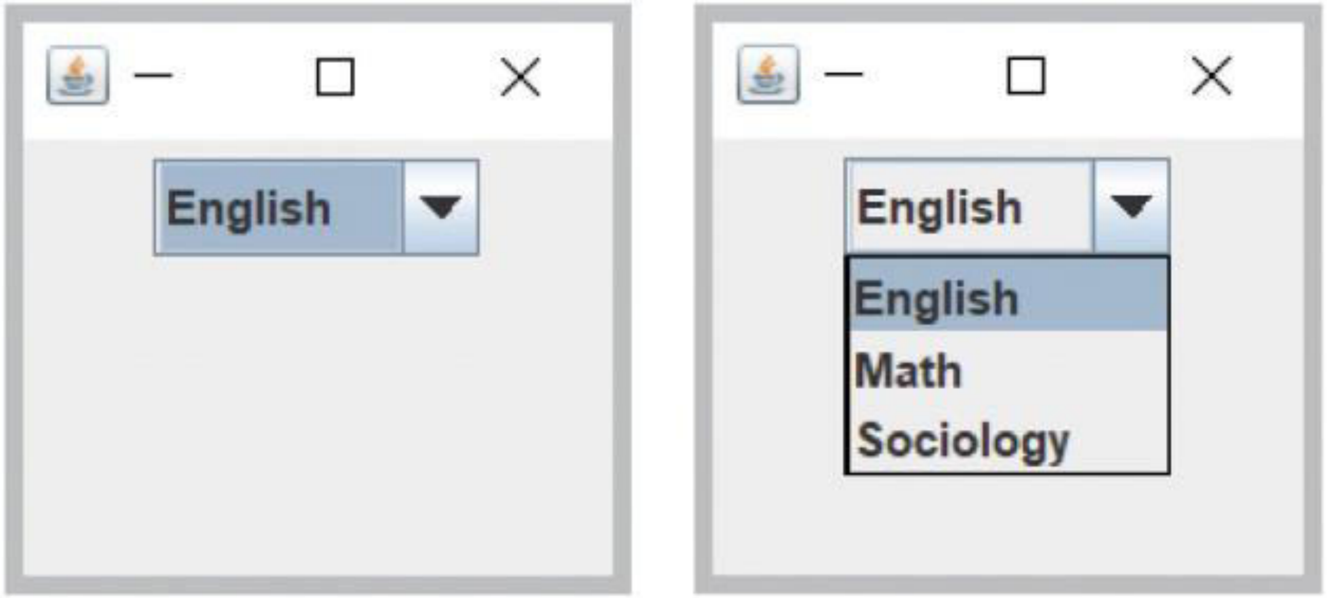


Figure 14-37 A JComboBox before and after the user clicks it

The JComboBox Class (3 of 5)

- To build a JComboBox:
  - Use a constructor with no arguments and then add items with the addItem() method
  - Alternatively, use an array of Objects as the constructor argument

```
String[] majorArray = {"English", "Math", "Sociology"};
JComboBox majorChoice = new JComboBox(majorArray);
```

The JComboBox Class (4 of 5)

Table 14-6 Some JComboBox class methods	
Method	**Purpose**
void addItem(Object)	Adds an item to the list
void removeItem(Object)	Removes an item from the list
void removeAllItems()	Removes all items from the list
Object getItemAt(Int)	Returns the list item at the index position specified by the integer argument
int getItemCount()	Returns the number of items in the list
int getMaximumRowCount()	Returns the maximum number of items the combo box can display without a scroll bar
int getSelectedIndex()	Returns the position of the currently selected item
Object getSelectedItem()	Returns the currently selected item
Object[] getSelectedObjects()	Returns an array containing selected Objects
void setEditable(boolean)	Sets the field to be editable or not editable
void setMaximumRowCount(int)	Sets the number of rows in the combo box that can be displayed at one time
void setSelectedIndex(int)	Sets the index at the position indicated by the argument
void setSelectedItem(Object)	Sets the selected item in the combo box display area to be the Object argument

CENGAGE

The JComboBox Class (5 of 5)

- setSelectedItem() or setSelectedIndex() method
  - Choose one item in the JComboBox to be the selected item
- getSelectedItem() or getSelectedIndex() method
  - Discover which item is currently selected
- Treat the list of items in a JComboBox object as an array
  - The first item is at position 0
  - The second item is at position 1
  - And so on

Don’t Do It

- Don’t forget the `x` in `javax` when you import `Swing` components into an application
- Don’t forget to use a `JFrame`’s `setVisible()` method if you want the `JFrame` to be visible
- Don’t forget to use `setLayout()` when you add multiple components to a `Jframe`
- Don’t forget to call `validate()` and `repaint()` after you add or remove a component from a container that has been made visible
- Don’t forget that `itemStateChanged()` executes when an ItemEvent is generated in response to a check box action ending in *d*
- Don’t forget that creating a `ButtonGroup` does not cause components to be grouped; each component that should be in the group must be added explicitly
- Don’t forget that the `ButtonGroup` class does not begin with a *J*

Summary (1 of 2)

- Each `Swing` component descends from `JComponent`
- `JFrame`
  - A `Swing` container that resembles a window
  - Has a title bar and borders, and the ability to be resized, minimized, restored, and closed
- `Jlabel` holds text
- Layout managers control component positioning
- Many types of listeners exist in Java
  - Each can handle a specific event type
  - Register a listener with the event source
  - Handle an event in the event-handling method

Summary (2 of 2)

- `JCheckBox`
  - Consists of a label positioned beside a square
- `ButtonGroup`
  - Groups several components so the user can select only one at a time
- `JComboBox`
  - Displays an area showing an option combined with a list box containing additional options