#### Section 12.1 Introduction

• A graphical user interface (GUI; p. [474](http://proquest.safaribooksonline.com/9780133813036/ch12_html#page_474)) presents a user-friendly mechanism for interacting with an application. A GUI gives an application a distinctive look-and-feel (p. [474](http://proquest.safaribooksonline.com/9780133813036/ch12_html#page_474)).

• Providing different applications with consistent, intuitive user-interface components gives users a sense of familarity with a new application, so that they can learn it more quickly.

• GUIs are built from GUI components (p. [474](http://proquest.safaribooksonline.com/9780133813036/ch12_html#page_474))—sometimes called controls or widgets.

#### Section 12.2 Java’s Nimbus Look-and-Feel

• As of Java SE 6 update 10, Java comes bundled with a new, elegant, cross-platform look-and-feel known as Nimbus (p. [476](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec2_html#page_476)).

• To set Nimbus as the default for all Java applications, create a swing.properties text file in the lib folder of your JDK and JRE installation folders. Place the following line of code in the file:

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p0543pro01a)

swing.defaultlaf=com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel

• To select Nimbus on an application-by-application basis, place the following command-line argument after the java command and before the application’s name when you run the application:

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p0543pro02a)

-Dswing.defaultlaf=com.sun.java.swing.plaf.nimbus.NimbusLookA

#### Section 12.3 Simple GUI-Based Input/Output with JOptionPane

• Most applications use windows or dialog boxes (p. [476](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec2_html#page_476)) to interact with the user.

• Class JOptionPane (p. [476](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec2_html#page_476)) of package javax.swing(p. [474](http://proquest.safaribooksonline.com/9780133813036/ch12_html#page_474)) provides prebuilt dialog boxes for both input and output. JOptionPane static methodshowInputDialog (p. [477](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec3_html#page_477)) displays an input dialog (p.[476](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec2_html#page_476)).

• A prompt typically uses sentence-style capitalization—capitalizing only the first letter of the first word in the text unless the word is a proper noun.

• An input dialog can input only input Strings. This is typical of most GUI components.

• JOptionPane static method showMessageDialog (p.[478](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec3_html#page_478)) displays a message dialog (p. [476](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec2_html#page_476)).

#### Section 12.4 Overview of Swing Components

• Most Swing GUI components (p. [474](http://proquest.safaribooksonline.com/9780133813036/ch12_html#page_474)) are located in package javax.swing.

• Together, the appearance and the way in which the user interacts with the application are known as that application’s look-and-feel. Swing GUI components allow you to specify a uniform look-and-feel for your application across all platforms or to use each platform’s custom look-and-feel.

• Lightweight Swing components are not tied to actual GUI components supported by the under lying platform on which an application executes.

• Several Swing components are heavyweight components (p. [480](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec7_html#page_480)) that require direct interaction with the local windowing system (p. [480](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec7_html#page_480)), which may restrict their appearance and functionality.

• Class Component (p. [480](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec7_html#page_480)) of package java.awtdeclares many of the attributes and behaviors common to the GUI components in packages java.awt (p. [479](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec6_html#page_479)) and javax.swing.

• Class Container (p. [480](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec7_html#page_480)) of package java.awt is a subclass of Component. Components are attached toContainers so the Components can be organized and displayed on the screen.

• Class JComponent (p. [480](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec7_html#page_480)) of package javax.swing is a subclass of Container. JComponent is the superclass of all lightweight Swing components and declares their common attributes and behaviors.

• Some common JComponent features include a pluggable look-and-feel (p. [480](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec7_html#page_480)), shortcut keys called mnemonics (p. [480](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec7_html#page_480)), tool tips (p. [480](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec7_html#page_480)), support for assistive technologies and support for user-interface localization (p. [480](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec7_html#page_480)).

#### Section 12.5 Displaying Text and Images in a Window

• Class JFrame provides the basic attributes and behaviors of a window.

• A JLabel (p. [481](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec5_html#page_481)) displays read-only text, an image, or both text and an image. Text in a JLabel normally uses sentence-style capitalization.

• Each GUI component must be attached to a container, such as a window created with a JFrame (p. [483](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec10_html#page_483)).

• Many IDEs provide GUI design tools (p. [529](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec18_html#page_529)) in which you can specify the exact size and location of a component by using the mouse; then the IDE will generate the GUI code for you.

• JComponent method setToolTipText (p. [483](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec10_html#page_483)) specifies the tool tip that’s displayed when the user positions the mouse cursor over a lightweight component (p. [480](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec7_html#page_480)).

• Container method add attaches a GUI component to aContainer.

• Class ImageIcon (p. [484](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec13_html#page_484)) supports several image formats, including GIF, PNG and JPEG.

• Method getClass of class Object (p. [484](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec13_html#page_484)) retrieves a reference to the Class object that represents the the class declaration for the object on which the method is called.

• Class method getResource (p. [484](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec13_html#page_484)) returns the location of its argument as a URL. The methodgetResource uses the Class object’s class loader to determine the location of the resource.

• The horizontal and vertical alignments of a JLabel can be set with methods setHorizontal Alignment (p.[484](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec13_html#page_484)) and setVerticalAlignment (p. [484](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec13_html#page_484)), respectively.

• JLabel methods setText (p. [484](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec13_html#page_484)) and getText (p.[484](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec13_html#page_484)) set and get the text displayed on a label.

• JLabel methods setIcon (p. [484](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec13_html#page_484)) and getIcon (p.[484](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec13_html#page_484)) set and get the Icon (p. [484](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec13_html#page_484)) on a label.

• JLabel methods setHorizontalTextPosition (p. [484](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec13_html#page_484)) and setVerticalTextPosition (p. [484](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec13_html#page_484)) specify the text position in the label.

• JFrame method setDefaultCloseOperation (p. [485](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec16_html#page_485)) with constant JFrame.EXIT\_ON\_CLOSE as the argument indicates that the program should terminate when the window is closed by the user.

• Component method setSize (p. [485](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec16_html#page_485)) specifies the width and height of a component.

• Component method setVisible (p. [485](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec16_html#page_485)) with the argument true displays a JFrame on the screen.

#### Section 12.6 Text Fields and an Introduction to Event Handling with Nested Classes

• GUIs are event driven—when the user interacts with a GUI component, events (p. [485](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec16_html#page_485)) drive the program to perform tasks.

• An event handler (p. [485](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec16_html#page_485)) performs a task in response to an event.

• Class JTextField (p. [485](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec16_html#page_485)) extends JTextComponent (p. [485](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec16_html#page_485)) of package javax.swing.text, which provides common text-based component features. Class JPasswordField (p. [485](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec16_html#page_485)) extends JTextField and adds several methods that are specific to processing passwords.

• A JPasswordField (p. [485](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec16_html#page_485)) shows that characters are being typed as the user enters them, but hides the actual characters with echo characters (p. [485](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec16_html#page_485)).

• A component receives the focus (p. [486](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec6_html#page_486)) when the user clicks the component.

• JTextComponent method setEditable (p. [488](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec18_html#page_488)) can be used to make a text field uneditable.

• To respond to an event for a particular GUI component, you must create a class that represents the event handler and implements an appropriate event-listener interface (p. [488](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec18_html#page_488)), then register an object of the event-handling class as the event handler (p. [488](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec18_html#page_488)).

• Non-static nested classes (p. [488](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec18_html#page_488)) are called inner classes and are frequently used for event handling.

• An object of a non-static inner class (p. [488](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec18_html#page_488)) must be created by an object of the top-level class (p. [488](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec18_html#page_488)) that contains the inner class.

• An inner-class object can directly access the instance variables and methods of its top-level class.

• A nested class that’s static does not require an object of its top-level class and does not implicitly have a reference to an object of the top-level class.

• Pressing Enter in a JTextField or JPasswordField generates an ActionEvent (p. [489](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec21_html#page_489)) that can be handled by an ActionListener (p. [489](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec21_html#page_489)) of package java.awt.event.

• JTextField method addActionListener (p. [489](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec21_html#page_489)) registers an event handler for a text field’sActionEvent.

• The GUI component with which the user interacts is the event source (p. [490](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec23_html#page_490)).

• An ActionEvent object contains information about the event that just occurred, such as the event source and the text in the text field.

• ActionEvent method getSource returns a reference to the event source. ActionEvent methodgetActionCommand (p. [490](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec23_html#page_490)) returns the text the user typed in a text field or the label on aJButton.

• JPasswordField method getPassword (p. [490](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec23_html#page_490)) returns the password the user typed.

#### Section 12.7 Common GUI Event Types and Listener Interfaces

• Each event-object type typically has a corresponding event-listener interface that specifies one or more event-handling methods, which must be declared in the class that implements the interface.

#### Section 12.8 How Event Handling Works

• When an event occurs, the GUI component with which the user interacted notifies its registered listeners by calling each listener’s appropriate event-handling method.

• Every GUI component supports several event types. When an event occurs, the event is dispatched (p. [494](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec26_html#page_494)) only to the event listeners of the appropriate type.

#### Section 12.9 JButton

• A button is a component the user clicks to trigger an action. All the button types are subclasses of AbstractButton (p. [495](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec27_html#page_495); package javax.swing). Button labels (p. [495](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec27_html#page_495)) typically use book-title capitalization (p. [495](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec27_html#page_495)).

• Command buttons (p. [495](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec27_html#page_495)) are created with class JButton.

• A JButton can display an Icon. A JButton can also have a rollover Icon (p. [497](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec9_html#page_497))—an Iconthat’s displayed when the user positions the mouse over the button.

• Method setRolloverIcon (p. [498](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec9_html#page_498)) of class AbstractButton specifies the image displayed on a button when the user positions the mouse over it.

#### Section 12.10 Buttons That Maintain State

• There are three Swing state button types—JToggleButton (p. [498](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec9_html#page_498)), JCheckBox (p. [498](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec9_html#page_498)) andJRadioButton (p. [498](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec9_html#page_498)).

• Classes JCheckBox and JRadioButton are subclasses ofJToggleButton.

• Component method setFont (p. [500](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec29_html#page_500)) sets the component’s font to a new Font object (p. [500](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec29_html#page_500)) of package java.awt.

• Clicking a JCheckBox causes an ItemEvent (p. [501](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec29_html#page_501)) that can be handled by an ItemListener (p. 501) which defines method itemStateChanged (p. 501). Method addItemListener registers the listener for theItemEvent of a JCheckBox or JRadioButton object.

• JCheckBox method isSelected determines whether aJCheckBox is selected.

• JRadioButtons have two states—selected and not selected. Radio buttons (p. [495](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec27_html#page_495)) normally ap pear as a group (p. [501](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec29_html#page_501)) in which only one button can be selected at a time.

• JRadioButtons are used to represent mutually exclusive options (p. [501](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec29_html#page_501)).

• The logical relationship between JRadioButtons is maintained by a ButtonGroup object (p. [501](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec29_html#page_501)).

• ButtonGroup method add (p. [504](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec30_html#page_504)) associates eachJRadioButton with a ButtonGroup. If more than one selected JRadioButton object is added to a group, the selected one that was added first will be selected when the GUI is displayed.

• JRadioButtons generate ItemEvents when they’re clicked.

#### Section 12.11 JComboBox; Using an Anonymous Inner Class for Event Handling

• A JComboBox (p. [504](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec30_html#page_504)) provides a list of items from which the user can make a single selection.JComboBoxes generate ItemEvents.

• Each item in a JComboBox has an index (p. [507](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec11_html#page_507)). The first item added to a JComboBox appears as the currently selected item when the JComboBox is displayed.

• JComboBox method setMaximumRowCount (p. [507](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec11_html#page_507)) sets the maximum number of elements that are displayed when the user clicks the JComboBox.

• An anonymous inner class (p. [507](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec11_html#page_507)) is a class without a name and typically appears inside a method declaration. One object of the anonymous inner class must be created when the class is declared.

• JComboBox method getSelectedIndex (p. [508](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec31_html#page_508)) returns the index of the selected item.

#### Section 12.12 JList

• A JList displays a series of items from which the user may select one or more items. Class JList supports single-selection lists (p. [508](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec31_html#page_508)) and multiple-selection lists.

• When the user clicks an item in a JList, aListSelectionEvent (p. [508](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec31_html#page_508)) occurs. JList methodaddListSelectionListener (p. [510](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec12_html#page_510)) registers aListSelectionListener (p. [510](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec12_html#page_510)) for a JList’s selection events. A ListSelectionListener of packagejavax.swing.event(p. [492](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec7_html#page_492)) must implement methodvalueChanged.

• JList method setVisibleRowCount (p. [510](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec12_html#page_510)) specifies the number of visible items in the list.

• JList method setSelectionMode (p. [510](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec12_html#page_510)) specifies a list’s selection mode.

• A JList can be attached to a JScrollPane (p. [510](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec12_html#page_510)) to provide a scrollbar for the JList.

• JFrame method getContentPane (p. [510](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec12_html#page_510)) returns a reference to the JFrame’s content pane where GUI components are displayed.

• JList method getSelectedIndex (p. [511](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec12_html#page_511)) returns the selected item’s index.

#### Section 12.13 Multiple-Selection Lists

• A multiple-selection list (p. [511](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec12_html#page_511)) enables the user to select many items from a JList.

• JList method setFixedCellWidth (p. [513](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec13_html#page_513)) sets aJList’s width. Method setFixedCellHeight (p. [513](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec13_html#page_513)) sets the height of each item in a JList.

• Normally, an external event (p. [513](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec13_html#page_513)) generated by another GUI component (such as a JButton) specifies when the multiple selections in a JList should be processed.

• JList method setListData (p. [513](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec13_html#page_513)) sets the items displayed in a JList. JList method getSe lectedValues (p. [513](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec13_html#page_513)) returns an array of Objects representing the selected items in a JList.

#### Section 12.13 Multiple-Selection Lists

• A multiple-selection list (p. [511](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec12_html#page_511)) enables the user to select many items from a JList.

• JList method setFixedCellWidth (p. [513](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec13_html#page_513)) sets aJList’s width. Method setFixedCellHeight (p. [513](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec13_html#page_513)) sets the height of each item in a JList.

• Normally, an external event (p. [513](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec13_html#page_513)) generated by another GUI component (such as a JButton) specifies when the multiple selections in a JList should be processed.

• JList method setListData (p. [513](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec13_html#page_513)) sets the items displayed in a JList. JList method getSe lectedValues (p. [513](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec13_html#page_513)) returns an array of Objects representing the selected items in a JList.

#### Section 12.15 Adapter Classes

• An adapter class (p. [518](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec33_html#page_518)) implements an interface and provides default implementations of its methods. When you extend an adapter class, you can override just the method(s) you need.

• MouseEvent method getClickCount (p. [521](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec34_html#page_521)) returns the number of consecutive mouse-button clicks. Methods isMetaDown (p. [528](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec17_html#page_528)) and isAltDown (p. [521](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec34_html#page_521)) determine which button was clicked.

#### Section 12.16 JPanel Subclass for Drawing with the Mouse

• JComponents method paintComponent (p. [522](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec16_html#page_522)) is called when a lightweight Swing component is displayed. Override this method to specify how to draw shapes using Java’s graphics capabilities.

• When overriding paintComponent, call the superclass version as the first statement in the body.

• Subclasses of JComponent support transparency. When a component is opaque (p. [522](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec16_html#page_522)), paintComponentclears its background before the component is displayed.

• The transparency of a Swing lightweight component can be set with method setOpaque (p. [522](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec16_html#page_522); a falseargument indicates that the component is transparent).

• Class Point (p. [523](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec36_html#page_523)) package java.awt represents anx-y coordinate.

• Class Graphics (p. [523](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec36_html#page_523)) is used to draw.

• MouseEvent method getPoint (p. [524](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec36_html#page_524)) obtains thePoint where a mouse event occurred.

• Method repaint (p. [524](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec36_html#page_524)), inherited indirectly from class Component, indicates that a component should be refreshed on the screen as soon as possible.

• Method paintComponent receives a Graphicsparameter and is called automatically whenever a lightweight component needs to be displayed on the screen.

• Graphics method fillOval (p. [524](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec36_html#page_524)) draws a solid oval. The first two arguments are the upper-left x-ycoordinate of the bounding box, and the last two are the bounding box’s width and height.

#### Section 12.17 Key Event Handling

• Interface KeyListener (p. [494](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec26_html#page_494)) is used to handle key events (p. [494](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec26_html#page_494)) that are generated when keys on the keyboard are pressed and released. MethodaddKeyListener of class Component (p. [525](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec17_html#page_525)) registers aKeyListener.

• KeyEvent (p. [494](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec26_html#page_494)) method getKeyCode (p. [528](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec17_html#page_528)) gets the virtual key code (p. [528](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec17_html#page_528)) of the pressed key. ClassKeyEvent contains virtual key-code constants that represent every key on the keyboard.

• KeyEvent method getKeyText (p. [528](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec17_html#page_528)) returns a string containing the name of the pressed key.

• KeyEvent method getKeyChar (p. [528](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec17_html#page_528)) gets the Unicode value of the character typed.

• KeyEvent method isActionKey (p. [528](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec17_html#page_528)) determines whether the key in an event was an action key (p. [525](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec17_html#page_525)).

• InputEvent method getModifiers (p. [528](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec17_html#page_528)) determines whether any modifier keys (such as Shift, Alt and Ctrl) were pressed when the key event occurred.

• KeyEvent method getKeyModifiersText (p. [528](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec17_html#page_528)) returns a string containing the pressed modifier keys.

#### Section 12.18 Introduction to Layout Managers

• Layout managers (p. [528](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec17_html#page_528)) arrange GUI components in a container for presentation purposes.

• All layout managers implement the interfaceLayoutManager (p. [528](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec17_html#page_528)) of package java.awt.

• Container method setLayout specifies the layout of a container.

• FlowLayout places components left to right in the order in which they’re added to the container. When the container’s edge is reached, components continue to display on the next line. FlowLayout allows GUI components to be left aligned, centered (the default) and right aligned.

• FlowLayout method setAlignment (p. [532](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec38_html#page_532)) changes the alignment for a FlowLayout.

• BorderLayout (p. [532](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec38_html#page_532)) the default for a JFrame) arranges components into five regions: NORTH, SOUTH,EAST, WEST and CENTER. NORTH corresponds to the top of the container.

• A BorderLayout limits a Container to containing at most five components—one in each region.

• GridLayout (p. [536](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec40_html#page_536)) divides a container into a grid of rows and columns.

• Container method validate (p. [537](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec40_html#page_537)) recomputes a container’s layout based on the current layout manager for the Container and the current set of displayed GUI components.

#### Section 12.19 Using Panels to Manage More Complex Layouts

• Complex GUIs often consist of multiple panels with different layouts. Every JPanel may have components, including other panels, attached to it with Containermethod add.

#### Section 12.20 JTextArea

• A JTextArea (p. [539](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec19_html#page_539))—a subclass of JTextComponent—may contain multiple lines of text.

• Class Box (p. [540](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec20_html#page_540)) is a subclass of Container that uses a BoxLayout layout manager (p. [541](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec20_html#page_541)) to arrange the GUI components either horizontally or vertically.

• Box static method createHorizontalBox (p. [541](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec20_html#page_541)) creates a Box that arranges components from left to right in the order that they’re attached.

• Method getSelectedText (p. [541](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec20_html#page_541)) returns the selected text from a JTextArea.

• You can set the horizontal and vertical scrollbar policies (p. [542](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec20_html#page_542)) of a JScrollPane when it’s constructed.JScrollPane methods setHorizontalScrollBarPolicy(p. [542](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec20_html#page_542)) and setVertical ScrollBarPolicy (p. [542](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec20_html#page_542)) can be used to change the scrollbar policies at any time.

**Self-Review Exercises**

[**12.1**](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec24_html#ch12ans1) Fill in the blanks in each of the following statements:

a) Method \_\_\_\_\_\_ is called when the mouse is moved with no buttons pressed and an event listener is registered to handle the event.

b) Text that cannot be modified by the user is called \_\_\_\_\_\_ text.

c) A(n) \_\_\_\_\_\_ arranges GUI components in a Container.

d) The add method for attaching GUI components is a method of class \_\_\_\_\_\_.

e) GUI is an acronym for \_\_\_\_\_\_.

f) Method \_\_\_\_\_\_ is used to specify the layout manager for a container.

g) A mouseDragged method call is preceded by a(n) \_\_\_\_\_\_ method call and followed by a(n) \_\_\_\_\_\_ method call.

h) Class \_\_\_\_\_\_ contains methods that display message dialogs and input dialogs.

i) An input dialog capable of receiving input from the user is displayed with method \_\_\_\_\_\_ of class \_\_\_\_\_\_.

j) A dialog capable of displaying a message to the user is displayed with method of class \_\_\_\_\_\_.

k) Both JTextFields and JTextAreas directly extend class \_\_\_\_\_\_.

[**12.2**](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec24_html#ch12ans2) Determine whether each statement is *true* or *false*. If *false*, explain why.

a) BorderLayout is the default layout manager for a JFrame’s content pane.

b) When the mouse cursor is moved into the bounds of a GUI component, method mouseOver is called.

c) A JPanel cannot be added to another JPanel.

d) In a BorderLayout, two buttons added to the NORTH region will be placed side by side.

e) A maximum of five components can be added to a BorderLayout.

f) Inner classes are not allowed to access the members of the enclosing class.

g) A JTextArea’s text is always read-only.

h) Class JTextArea is a direct subclass of class Component.

[**12.3**](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec24_html#ch12ans3) Find the error(s) in each of the following statements, and explain how to correct it (them):

a) buttonName = JButton("Caption");

b) JLabel aLabel, JLabel;

c) txtField = new JTextField(50, "Default Text");

d)

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p549pro01a)

setLayout(new BorderLayout());  
button1 = new JButton("North Star");  
button2 = new JButton("South Pole");  
add(button1);  
add(button2);

### Answers to Self-Review Exercises

[**12.1**](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec23_html#ch12que1)

a) mouseMoved.

b) uneditable (read-only).

c) layout manager.

d) Container.

e) graphical user interface.

f) setLayout.

g) mousePressed, mouseReleased.

h) JOptionPane.

i) showInputDialog, JOptionPane.

j) showMessageDialog, JOptionPane.

k) JTextComponent.

[**12.2**](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec23_html#ch12que2)

a) True.

b) False. Method mouseEntered is called.

c) False. A JPanel can be added to another JPanel, because JPanel is an indirect subclass ofComponent. So, a JPanel is a Component. Any Component can be added to a Container.

d) False. Only the last button added will be displayed. Remember that only one component should be added to each region in a BorderLayout.

e) True. [Note: Panels containing multiple components can be added to each region.]

f) False. Inner classes have access to all members of the enclosing class declaration.

g) False. JTextAreas are editable by default.

h) False. JTextArea derives from class JTextComponent.

[**12.3**](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec23_html#ch12que3)

a) new is needed to create an object.

b) JLabel is a class name and cannot be used as a variable name.

c) The arguments passed to the constructor are reversed. The String must be passed first.

d) BorderLayout has been set, and components are being added without specifying the region, so both are added to the center region. Proper add statements might be

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p549pro02a)

add(button1, BorderLayout.NORTH);  
add(button2, BorderLayout.SOUTH);

**Exercises**

**12.4** Fill in the blanks in each of the following statements:

a) The JTextField class directly extends class \_\_\_\_\_\_\_.

b) Container method \_\_\_\_\_\_\_\_ attaches a GUI component to a container.

c) Method \_\_\_\_\_\_\_\_ is called when a mouse button is released (without moving the mouse).

d) The \_\_\_\_\_\_\_\_ class is used to create a group of JRadioButtons.

**12.5** Determine whether each statement is *true* or *false*. If *false*, explain why.

a) Only one layout manager can be used per Container.

b) GUI components can be added to a Container in any order in a BorderLayout.

c) JRadioButtons provide a series of mutually exclusive options (i.e., only one can be true at a time).

d) Graphics method setFont is used to set the font for text fields.

e) A JList displays a scrollbar if there are more items in the list than can be displayed.

f) A Mouse object has a method called mouseDragged.

**12.6** Determine whether each statement is *true* or *false*. If *false*, explain why.

a) A JPanel is a JComponent.

b) A JPanel is a Component.

c) A JLabel is a Container.

d) A JList is a JPanel.

e) An AbstractButton is a JButton.

f) A JTextField is an Object.

g) ButtonGroup is a subclass of JComponent.

**12.7** Find any errors in each of the following lines of code, and explain how to correct them.

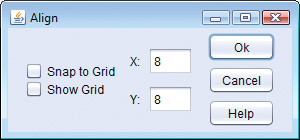
a) import javax.swing.JFrame

b) panelObject.GridLayout(8, 8);

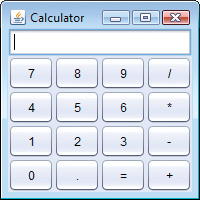
c) container.setLayout(new FlowLayout(FlowLayout.DEFAULT));

d) container.add(eastButton, EAST); face

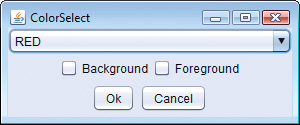
**12.8** Create the following GUI. You do not have to provide any functionality.



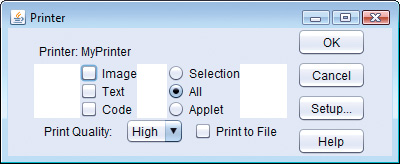
**12.9** Create the following GUI. You do not have to provide any functionality.



**12.10** Create the following GUI. You do not have to provide any functionality.



**12.11** Create the following GUI. You do not have to provide any functionality.



**12.12 *(Temperature Conversion)*** Write a temperature-conversion application that converts from Fahrenheit to Celsius. The Fahrenheit temperature should be entered from the keyboard (via aJTextField). A JLabel should be used to display the converted temperature. Use the following formula for the conversion:

Image

**12.13 *(Temperature-Conversion Modification)*** Enhance the temperature-conversion application of [Exercise 12.12](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec25_html#ch12que12) by adding the Kelvin temperature scale. The application should also allow the user to make conversions between any two scales. Use the following formula for the conversion between Kelvin and Celsius (in addition to the formula in [Exercise 12.12](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec25_html#ch12que12)):

*Kelvin* = *Celsius* + 273.15

**12.14 *(Guess-the-Number Game)*** Write an application that plays “guess the number” as follows: Your application chooses the number to be guessed by selecting an integer at random in the range 1–1000. The application then displays the following in a label:

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p0551pro02a)

I have a number between 1 and 1000. Can you guess my number?  
Please enter your first guess.

A JTextField should be used to input the guess. As each guess is input, the background color should change to either red or blue. Red indicates that the user is getting “warmer,” and blue, “colder.” AJLabel should display either "Too High" or "Too Low" to help the user zero in. When the user gets the correct answer, "Correct!" should be displayed, and the JTextField used for input should be changed to be uneditable. A JButton should be provided to allow the user to play the game again. When the JButton is clicked, a new random number should be generated and the input JTextFieldchanged to be editable.

**12.15 *(Displaying Events)*** It’s often useful to display the events that occur during the execution of an application. This can help you understand when the events occur and how they’re generated. Write an application that enables the user to generate and process every event discussed in this chapter. The application should provide methods from the ActionListener, ItemListener,ListSelectionListener, MouseListener, MouseMotionListener and KeyListener interfaces to display messages when the events occur. Use method toString to convert the event objects received in each event handler into Strings that can be displayed. Method toString creates aString containing all the information in the event object.

**12.16 *(GUI-Based Craps Game)*** Modify the application of [Section 6.10](http://proquest.safaribooksonline.com/9780133813036/ch06lev1sec10_html#ch06lev1sec10) to provide a GUI that enables the user to click a JButton to roll the dice. The application should also display four JLabels and four JTextFields, with one JLabel for each JTextField. The JTextFields should be used to display the values of each die and the sum of the dice after each roll. The point should be displayed in the fourth JTextField when the user does not win or lose on the first roll and should continue to be displayed until the game is lost.

#### (Optional) GUI and Graphics Case Study Exercise: Expanding the Interface

**12.17****(Interactive Drawing Application)** In this exercise, you’ll implement a GUI application that uses the MyShape hierarchy from GUI and Graphics Case Study [Exercise 10.2](http://proquest.safaribooksonline.com/9780133813036/ch10lev1sec14_html#ch10que2) to create an interactive drawing application. You’ll create two classes for the GUI and provide a test class that launches the application. The classes of the MyShape hierarchy require no additional changes.

The first class to create is a subclass of JPanel called DrawPanel, which represents the area on which the user draws the shapes. Class DrawPanel should have the following instance variables:

a) An array shapes of type MyShape that will store all the shapes the user draws.

b) An integer shapeCount that counts the number of shapes in the array.

c) An integer shapeType that determines the type of shape to draw.

d) A MyShape currentShape that represents the current shape the user is drawing.

e) A Color currentColor that represents the current drawing color.

f) A boolean filledShape that determines whether to draw a filled shape.

g) A JLabel statusLabel that represents the status bar. The status bar will display the co ordinates of the current mouse position.

Class DrawPanel should also declare the following methods:

a) Overridden method paintComponent that draws the shapes in the array. Use instance variable shapeCount to determine how many shapes to draw. Method paintComponent should also call currentShape’s draw method, provided that currentShape is not null.

b) Set methods for the shapeType, currentColor and filledShape.

c) Method clearLastShape should clear the last shape drawn by decrementing instance variableshapeCount. Ensure that shapeCount is never less than zero.

d) Method clearDrawing should remove all the shapes in the current drawing by settingshapeCount to zero.

Methods clearLastShape and clearDrawing should call repaint (inherited from JPanel) to refresh the drawing on the DrawPanel by indicating that the system should call method paintComponent.

Class DrawPanel should also provide event handling to enable the user to draw with the mouse. Create a single inner class that both extends MouseAdapter and implements MouseMotionListenerto handle all mouse events in one class.

In the inner class, override method mousePressed so that it assigns currentShape a new shape of the type specified by shapeType and initializes both points to the mouse position. Next, override method mouseReleased to finish drawing the current shape and place it in the array. Set the second point of currentShape to the current mouse position and add currentShape to the array. Instance variable shapeCount determines the insertion index. Set currentShape to null and call methodrepaint to update the drawing with the new shape.

Override method mouseMoved to set the text of the statusLabel so that it displays the mouse coordinates—this will update the label with the coordinates every time the user moves (but does not drag) the mouse within the DrawPanel. Next, override method mouseDragged so that it sets the second point of the currentShape to the current mouse position and calls method repaint. This will allow the user to see the shape while dragging the mouse. Also, update the JLabel in mouseDraggedwith the current position of the mouse.

Create a constructor for DrawPanel that has a single JLabel parameter. In the constructor, initialize statusLabel with the value passed to the parameter. Also initialize array shapes with 100 entries, shapeCount to 0, shapeType to the value that represents a line, currentShape to null andcurrentColor to Color.BLACK. The constructor should then set the background color of theDrawPanel to Color.WHITE and register the MouseListener and MouseMotionListener so theJPanel properly handles mouse events.

Next, create a JFrame subclass called DrawFrame that provides a GUI that enables the user to control various aspects of drawing. For the layout of the DrawFrame, we recommend a BorderLayout,with the components in the NORTH region, the main drawing panel in the CENTER region, and a status bar in the SOUTH region, as in [Fig. 12.49](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec62_html#ch12fig49). In the top panel, create the components listed below. Each component’s event handler should call the appropriate method in class DrawPanel.

a) A button to undo the last shape drawn.

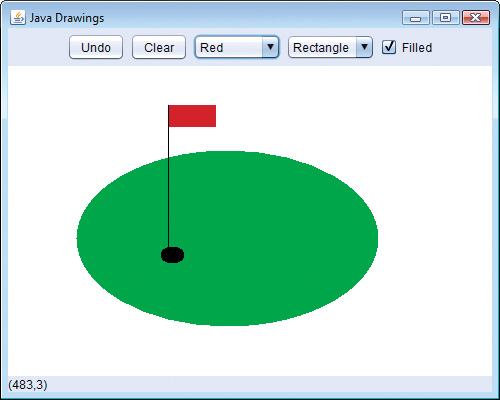
b) A button to clear all shapes from the drawing.

c) A combo box for selecting the color from the 13 predefined colors.

d) A combo box for selecting the shape to draw.

e) A checkbox that specifies whether a shape should be filled or unfilled.

Declare and create the interface components in DrawFrame’s constructor. You’ll need to create the status bar JLabel before you create the DrawPanel, so you can pass the JLabel as an argument toDrawPanel’s constructor. Finally, create a test class that initializes and displays the DrawFrame to execute the application.



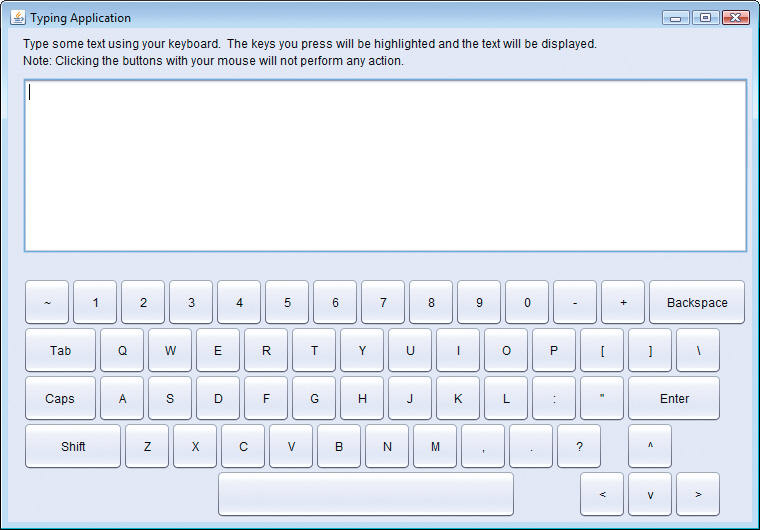
**Fig. 12.49** | Interface for drawing shapes.

**12.18 (GUI-Based Version of the ATM Case Study)** Reimplement the Optional ATM Case Study of Chapters 33–34 as a GUI-based application. Use GUI components to approximate the ATM user interface shown in Fig. 33.1. For the cash dispenser and the deposit slot use JButtons labeled**Remove Cash** and **Insert Envelope**. This will enable the application to receive events indicating when the user takes the cash and inserts a deposit envelope, respectively.

**Making a Difference**

**12.19 *(Ecofont)*** Ecofont ([www.ecofont.eu/ecofont\_en.html](http://www.ecofont.eu/ecofont_en.html))—developed by SPRANQ (a Netherlands-based company)—is a free, open-source computer font designed to reduce by as much as 20% the amount of ink used for printing, thus reducing also the number of ink cartridges used and the environmental impact of the manufacturing and shipping processes (using less energy, less fuel for shipping, and so on). The font, based on sans-serif Verdana, has small circular “holes” in the letters that are not visible in smaller sizes—such as the 9- or 10-point type frequently used. Download Ecofont, then install the font file Spranq\_eco\_sans\_regular.ttf using the instructions from the Ecofont website. Next, develop a GUI-based program that allows you to type in a text string to be displayed in the Ecofont. Create **Increase Font Size** and **Decrease Font Size** buttons that allow you to scale up or down by one point at a time. Start with a default font size of 9 points. As you scale up, you’ll be able to see the holes in the letters more clearly. As you scale down, the holes will be less apparent. What is the smallest font size at which you begin to notice the holes?

**12.20*****(Typing Tutor: Tuning a Crucial Skill in the Computer Age)*** Typing quickly and correctly is an essential skill for working effectively with computers and the Internet. In this exercise, you’ll build a GUI application that can help users learn to “touch type” (i.e., type correctly without looking at the keyboard). The application should display a *virtual keyboard* ([Fig. 12.50](http://proquest.safaribooksonline.com/9780133813036/ch12lev1sec26_html#ch12fig50)) and should allow the user to watch what he or she is typing on the screen without looking at the *actual keyboard*. UseJButtons to represent the keys. As the user presses each key, the application highlights the corresponding JButton on the GUI and adds the character to a JTextArea that shows what the user has typed so far. [*Hint:* To highlight a JButton, use its setBackground method to change its background color. When the key is released, reset its original background color. You can obtain theJButton’s original background color with the getBackground method before you change its color.]



**Fig. 12.50** | Typing tutor.

You can test your program by typing a pangram—a phrase that contains every letter of the alphabet at least once—such as “The quick brown fox jumped over a lazy dog.” You can find other pangrams on the web.

To make the program more interesting you could monitor the user’s accuracy. You could have the user type specific phrases that you’ve prestored in your program and that you display on the screen above the virtual keyboard. You could keep track of how many keystrokes the user types correctly and how many are typed incorrectly. You could also keep track of which keys the user is having difficulty with and display a report showing those keys.