Chapter - 4
Awt Controls

Introduction

- The AWT supports the following types of controls:
  - Labels
  - Buttons
  - Check boxes
  - Choice lists
  - Lists
  - Scroll bars
  - Text Editing
- These controls are subclasses of Component.
- Although this is not a particularly rich set of controls, it is sufficient for simple applications.

Adding and Removing Controls

- To include a control in a window, you must add it to the window.
- To do this, you must first create an instance of the desired control and then add it to a window by calling add( ), which is defined by Container.
- The add( ) method has several forms. The following form is the one that is used for the first part of this chapter:
- Component add(Component compRef)

  Here, compRef is a reference to an instance of the control that you want to add. A reference to the object is returned. Once a control has been added, it will automatically be visible whenever its parent window is displayed.

- Sometimes you will want to remove a control from a window when the control is no longer needed. To do this, call remove( ). This method is also defined by Container. Here is one of its forms:
    - void remove(Component compRef)
- Here, compRef is a reference to the control you want to remove. You can remove all controls by calling removeAll( ).

Responding to Controls

- Except for labels, which are passive, all other controls generate events when they are accessed by the user.
- For example, when the user clicks on a push button, an event is sent that identifies the push button.
- In general, your program simply implements the appropriate interface and then registers an event listener for each control that you need to monitor.
- Once a listener has been installed, events are automatically sent to it.

Label

- The easiest control to use is a label.
- A label is an object of type Label, and it contains a string, which it displays.
- Labels are passive controls that do not support any interaction with the user.
- Label defines the following constructors:
  - Label( ) throws HeadlessException
  - Label(String str) throws HeadlessException
  - Label(String str, int how) throws HeadlessException
- The first version creates a blank label.
- The second version creates a label that contains the string specified by str. This string is left-justified.
- The third version creates a label that contains the string specified by str using the alignment specified by how. The value of how must be one of these three constants: Label.LEFT, Label.RIGHT, or Label.CENTER.

- You can set or change the text in a label by using the setText( ) method. You can obtain the current label by calling getText( ). These methods are shown here:
  - void setText(String str)
  - String getText( )
- You can set the alignment of the string within the label by calling setAlignment( ). To obtain the current alignment, call getAlignment( ). The methods are as follows:
  - void setAlignment(int how)
  - int getAlignment( )

Buttons

- Perhaps the most widely used control is the push button.
- A push button is a component that contains a label and that generates an event when it is pressed.
- Push buttons are objects of type Button. Button defines these two constructors:
  - Button( ) throws HeadlessException
  - Button(String str) throws HeadlessException
- The first version creates an empty button.
- The second creates a button that contains str as a label.
- After a button has been created, you can set its label by calling setLabel( ). You can retrieve its label by calling getLabel( ). These methods are as follows:
  - void setLabel(String str)
  - String getLabel( )
- Here, str becomes the new label for the button.

Check Boxes

- A check box is a control that is used to turn an option on or off.
- It consists of a small box that can either contain a check mark or not.
- There is a label associated with each checkbox that describes what option the box represents.
- You change the state of a checkbox by clicking on it.
- Check boxes can be used individually or as part of a group.
- Check boxes are objects of the Checkbox class.
    - Checkbox( ) throws HeadlessException
    - Checkbox(String str) throws HeadlessException
    - Checkbox(String str, boolean on) throws HeadlessException
    - Checkbox(String str, boolean on, CheckboxGroup cbGroup) throws HeadlessException
    - Checkbox(String str, CheckboxGroup cbGroup, boolean on) throws HeadlessException

- The first form creates a checkbox whose label is initially blank. The state of the checkbox is unchecked.
- The second form creates a checkbox whose label is specified by str. The state of the checkbox is unchecked.
- The third form allows you to set the initial state of the check box. If on is true, the check box is initially checked; otherwise, it is cleared.
- The fourth and fifth forms create a checkbox whose label is specified by str and whose group is specified by cbGroup.
- If this check box is not part of a group, then cbGroup must be null.

Choice Controls

- The Choice class is used to create a pop-up list of items from which the user may choose.
- Thus, a Choice control is a form of menu. When inactive, a Choice component takes up only enough space to show the currently selected item.
- When the user clicks on it, the whole list of choices pops up, and a new selection can be made.
- Each item in the list is a string that appears as a left-justified label in the order it is added to the Choice object.
- Choice defines only the default constructor, which creates an empty list.
- To add a selection to the list, call add( ). It has this general form:
    - void add(String name)
- Here, name is the name of the item being added. Items are added to the list in the order in which calls to add( ) occur.

- To determine which item is currently selected, you may call either getSelectedItem( ) or getSelectedIndex( ). These methods are shown here:
  - String getSelectedItem( )
  - int getSelectedIndex( )
- The getSelectedItem( ) method returns a string containing the name of the item. getSelectedIndex( ) returns the index of the item. The first item is at index 0. By default, the first item added to the list is selected.
- To obtain the number of items in the list, call getItemCount( ). You can set the currently selected item using the select( ) method with either a zero-based integer index or a string that will match a name in the list. These methods are shown here:
  - int getItemCount( )
  - void select(int index)
  - void select(String name)
- Given an index, you can obtain the name associated with the item at that index by calling getItem( ), which has this general form:
  - String getItem(int index)
- Here, index specifies the index of the desired item.

Using a TextField

- The TextField class implements a single-line text-entry area, usually called an edit control.
- Text fields allow the user to enter strings and to edit the text using the arrow keys, cut and paste keys, and mouse selections.
- TextField is a subclass of TextComponent. TextField defines the following constructors:
  - TextField( ) throws HeadlessException
  - TextField(int numChars) throws HeadlessException
  - TextField(String str) throws HeadlessException
  - TextField(String str, int numChars) throws HeadlessException
- The first version creates a default text field. The second form creates a text field that is numChars characters wide. The third form initializes the text field with the string contained in str. The fourth form initializes a text field and sets its width.

- TextField (and its superclass TextComponent) provides several methods that allow you to utilize a text field. To obtain the string currently contained in the text field, call getText( ). To set the text, call setText( ). These methods are as follows:
  - String getText( )
  - void setText(String str)
- The user can select a portion of the text in a text field. Also, you can select a portion of text under program control by using select( ). Your program can obtain the currently selected text by calling getSelectedText( ). These methods are shown here:
  - String getSelectedText( )
  - void select(int startIndex, int endIndex)
- getSelectedText( ) returns the selected text. The select( ) method selects the characters beginning at startIndex and ending at endIndex –1.

- You can control whether the contents of a text field may be modified by the user by calling setEditable( ). You can determine editability by calling isEditable( ). These methods are shown here:
  - boolean isEditable( )
  - void setEditable(boolean canEdit)
- isEditable( ) returns true if the text may be changed and false if not. In setEditable( ), if canEdit is true, the text may be changed. If it is false, the text cannot be altered.
- There may be times when you will want the user to enter text that is not displayed, such as a password.
- You can disable the echoing of the characters as they are typed by calling setEchoChar( ).
- This method specifies a single character that the TextField will display when characters are entered (thus, the actual characters typed will not be shown).

- You can check a text field to see if it is in this mode with the echoCharIsSet( ) method. You can retrieve the echo character by calling the getEchoChar( ) method. These methods are as follows:
- void setEchoChar(char ch)
- boolean echoCharIsSet( )
- char getEchoChar( )
- Here, ch specifies the character to be echoed. If ch is zero, then normal echoing is restored.

Using a TextArea

- Sometimes a single line of text input is not enough for a given task. To handle these situations, the AWT includes a simple multiline editor called TextArea.
- Following are the constructors for TextArea:
  - TextArea( ) throws HeadlessException
  - TextArea(int numLines, int numChars) throws HeadlessException
  - TextArea(String str) throws HeadlessException
  - TextArea(String str, int numLines, int numChars) throws HeadlessException
  - TextArea(String str, int numLines, int numChars, int sBars) throws HeadlessException

Here, numLines specifies the height, in lines, of the text area, and numChars specifies its width, in characters.

Initial text can be specified by str.

In the fifth form, you can specify the scroll bars that you want the control to have. sBars must be one of these values:

SCROLLBARS_BOTH	SCROLLBARS_NONE
SCROLLBARS_HORIZONTAL_ONLY	SCROLLBARS_VERTICAL_ONLY

TextArea is a subclass of TextComponent. Therefore, it supports the getText( ), setText( ), getSelectedText( ), select( ), isEditable( ), and setEditable( ) methods described in the preceding section.

Using Lists

- The List class provides a compact, multiple-choice, scrolling selection list.
- Unlike the Choice object, which shows only the single selected item in the menu, a List object can be constructed to show any number of choices in the visible window.
- It can also be created to allow multiple selections. List provides these constructors:
  - List( ) throws HeadlessException
  - List(int numRows) throws HeadlessException
  - List(int numRows, boolean multipleSelect) throws HeadlessException
- The first version creates a List control that allows only one item to be selected at any one time.

- For lists that allow only single selection, you can determine which item is currently selected by calling either getSelectedItem( ) or getSelectedIndex( ). These methods are shown here:
  - String getSelectedItem( )
  - int getSelectedIndex( )
- The getSelectedItem( ) method returns a string containing the name of the item.
- getSelectedIndex( ) returns the index of the item. The first item is at index 0.

- In the second form, the value of numRows specifies the number of entries in the list that will always be visible (others can be scrolled into view as needed).
- In the third form, if multipleSelect is true, then the user may select two or more items at a time. If it is false, then only one item may be selected.
- For lists that allow multiple selection, you must use either getSelectedItems( ) or getSelectedIndexes( ), shown here, to determine the current selections:
    - String[ ] getSelectedItems( )
    - int[ ] getSelectedIndexes( )
- getSelectedItems( ) returns an array containing the names of the currently selected items. getSelectedIndexes( ) returns an array containing the indexes of the currently selected items.

- To add a selection to the list, call add( ). It has the following two forms:
  - void add(String name)
  - void add(String name, int index)
- Here, name is the name of the item added to the list.
- The first form adds items to the end of the list. The second form adds the item at the index specified by index.
- Indexing begins at zero. You can specify –1 to add the item to the end of the list.
- To obtain the number of items in the list, call getItemCount( ).
- You can set the currently selected item by using the select( ) method with a zero-based integer index. These methods are shown here:
  - int getItemCount( )
  - void select(int index)
- Given an index, you can obtain the name associated with the item at that index by calling getItem( ), which has this general form:
  - String getItem(int index)
- Here, index specifies the index of the desired item.

Managing Scroll Bars

- Scroll bars are used to select continuous values between a specified minimum and maximum.
- Scroll bars may be oriented horizontally or vertically.
- A scroll bar is actually a composite of several individual parts. Each end has an arrow that you can click to move the current value of the scroll bar one unit in the direction of the arrow.
- The current value of the scroll bar relative to its minimum and maximum values is indicated by the slider box (or thumb) for the scroll bar.
- The slider box can be dragged by the user to a new position. The scroll bar will then reflect this value.

- In the background space on either side of the thumb, the user can click to cause the thumb to jump in that direction by some increment larger than 1.
- Typically, this action translates into some form of page up and page down. Scroll bars are encapsulated by the Scrollbar class.
- Scrollbar defines the following constructors:
    - Scrollbar() throws Headless Exception
    - Scrollbar(int style) throws Headless Exception
    - Scrollbar(int style, int initialValue, int thumbSize,int min,int max) throws Headless Exception

- The first form creates a vertical scroll bar.
- The second and third forms allow you to specify the orientation of the scroll bar.
- If style is Scrollbar.VERTICAL, a vertical scroll bar is created.
- If style is Scrollbar.HORIZONTAL, the scroll bar is horizontal.
- In the third form of the constructor, the initial value of the scroll bar is passed in initialValue.
- The number of units represented by the height of the thumb is passed in thumbSize.
- The minimum and maximum values for the scroll bar are specified by min and max.

- If you construct a scroll bar by using one of the first two constructors, then you need to set its parameters by using setValues( ), shown here, before it can be used:
  - void setValues(int initialValue, int thumbSize, int min, int max)
- The parameters have the same meaning as they have in the third constructor just described.
- To obtain the current value of the scroll bar, call getValue( ). It returns the current setting.
- To set the current value, call setValue( ). These methods are as follows:
  - int getValue( )
  - void setValue(int newValue)
- Here, newValue specifies the new value for the scroll bar.
- When you set a value, the slider box inside the scroll bar will be positioned to reflect the new value.

- You can also retrieve the minimum and maximum values via getMinimum( ) and getMaximum( ), shown here:
  - int getMinimum( )
  - int getMaximum( )
- They return the requested quantity.
- By default, 1 is the increment added to or subtracted from the scroll bar each time it is scrolled up or down one line. You can change this increment by calling setUnitIncrement( ).
- By default, page-up and page-down increments are 10. You can change this value by calling setBlockIncrement( ). These methods are shown here:
  - void setUnitIncrement(int newIncr)
  - void setBlockIncrement(int newIncr)

Layout manager

- Layout means the arrangement of components within the container.
- In other way we can say that placing the components at a particular position within the container.
- The task of layouting the controls is done automatically by the Layout Manager.
- A layout manager automatically arranges your controls within a window by using some type of algorithm.
- Each Container object has a layout manager associated with it.
- The layout manager is set by the setLayout( ) method. If no call to setLayout( ) is made, then the default layout manager is used.
- Whenever a container is resized (or sized for the first time), the layout manager is used to position each of the components within it.

- The setLayout( ) method has the following general form:
  - void setLayout(LayoutManager layoutObj)
- Here, layoutObj is a reference to the desired layout manager.
- If you wish to disable the layout manager and position components manually, pass null for layoutObj
- If you do this, you will need to determine the shape and position of each component manually, using the setBounds( ) method defined by Component.

Flow Layout

- FlowLayout is the default layout manager.
- FlowLayout implements a simple layout style, which is similar to how words flow in a text editor.
- The direction of the layout is governed by the container’s component orientation property, which, by default, is left to right, top to bottom.
- Therefore, by default, components are laid out line-by-line beginning at the upper-left corner. In all cases, when a line is filled, layout advances to the next line.
- A small space is left between each component, above and below, as well as left and right. Here are the constructors for FlowLayout:
    - FlowLayout( )
    - FlowLayout(int how)
    - FlowLayout(int how, int horz, int vert)

- The first form creates the default layout, which centers components and leaves five pixels of space between each component.
- The second form lets you specify how each line is aligned. Valid values for how are as follows:
  - FlowLayout.LEFT
  - FlowLayout.CENTER
  - FlowLayout.RIGHT
  - FlowLayout.LEADING
  - FlowLayout.TRAILING
- These values specify left, center, right, leading edge, and trailing edge alignment, respectively.
- The third constructor allows you to specify the horizontal and vertical space left between components in horz and vert, respectively.

BorderLayout

- The BorderLayout class implements a common layout style for top-level windows.
- It has four narrow, fixed-width components at the edges and one large area in the center.
- The four sides are referred to as north, south, east, and west. The middle area is called the center.
- Here are the constructors defined by BorderLayout:
  - BorderLayout( )
  - BorderLayout(int horz, int vert)
- The first form creates a default border layout.
- The second allows you to specify the horizontal and vertical space left between components in horz and vert, respectively.

- BorderLayout defines the following constants that specify the regions:
  - BorderLayout.CENTER
  - BorderLayout.EAST
  - BorderLayout.WEST
  - BorderLayout.NORTH
  - BorderLayout.SOUTH
- When adding components, you will use these constants with the following form of add( ), which is defined by Container:
  - void add(Component compRef, Object region)
- Here, compRef is a reference to the component to be added, and region specifies where the component will be added.

Grid Layout

- GridLayout lays out components in a two-dimensional grid.
- When you instantiate a GridLayout, you define the number of rows and columns.
- The constructors supported by GridLayout are shown here:
  - GridLayout( )
  - GridLayout(int numRows, int numColumns)
  - GridLayout(int numRows, int numColumns, int horz, int vert)
- The first form creates a single-column grid layout.
- The second form creates a grid layout with the specified number of rows and columns.
- The third form allows you to specify the horizontal and vertical space left between components in horz and vert, respectively.
- Either numRows or numColumns can be zero. Specifying numRows as zero allows for unlimited-length columns. Specifying numColumns as zero allows for unlimited-length rows.

GridBagLayout

- We can specify the relative placement of components by specifying their positions within cells inside a grid.
- The key to the grid bag is that each component can be a different size, and each row in the grid can have a different number of columns. This is why the layout is called a grid bag. It’s a collection of small grids joined together.
- The location and size of each component in a grid bag are determined by a set of constraints linked to it. The constraints are contained in an object of type GridBagConstraints. Constraints include the height and width of a cell, and the placement of a component, its alignment, and its anchor point within the cell.
- The general procedure for using a grid bag is to first create a new GridBagLayout object and to make it the current layout manager.
- Then, set the constraints that apply to each component that will be added to the grid bag.
- Finally, add the components to the layout manager.

Constraints

- **gridx**, **gridy**
  - Specify the row and column at the upper left of the component. The leftmost column has address gridx=0 and the top row has address gridy=0.
- **gridwidth**, **gridheight**
  - Specify the number of columns (for gridwidth) or rows (for gridheight) in the component's display area. These constraints specify the number of cells the component uses, *not* the number of pixels it uses. The default value is 1.
- **ipadx**, **ipady**
  - Specifies the internal padding: how much to add to the size of the component. The default value is zero.

- fill
  - This field is used when the component's display area is larger than the component's requested size. It determines whether to resize the component, and if so, how.
  - The following values are valid for fill:
    - NONE: Do not resize the component.
    - HORIZONTAL: Make the component wide enough to fill its display area horizontally, but do not change its height.
    - VERTICAL: Make the component tall enough to fill its display area vertically, but do not change its width.
    - BOTH: Make the component fill its display area entirely.

Card Layout

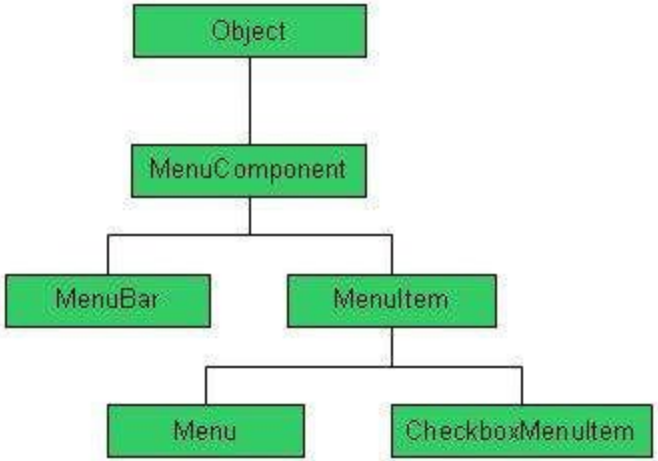
- The CardLayout class manages the components in such a manner that only one component is visible at a time.
- It treats each component as a card that is why it is known as CardLayout
- The CardLayout class is unique among the other layout managers in that it stores several different layouts.
- You can prepare the other layouts and have them hidden, ready to be activated when needed.
- CardLayout provides these two constructors:
- CardLayout( )
- CardLayout(int horz, int vert)
- The first form creates a default card layout.
- The second form allows you to specify the horizontal and vertical space left between components in horz and vert, respectively.

- When card panels are added to a panel, they are usually given a name.
- Thus, most of the time, you will use this form of add( ) when adding cards to a panel:
    - void add(Component comp, Object name)

Here, name is a string that specifies the name of the card whoseAfter you have created a deck, your program activates a card by calling one of the following methods defined by CardLayout:

- void first(Container container)
- void last(Container container)
- void next(Container container)
- void previous(Container container)
- void show(Container container, String cardName)

Menu


Object
MenuComponent
MenuBar
MenuItem
Menu
CheckboxMenuItem

- A top-level window can have a menu bar associated with it.
- A menu bar displays a list of top-level menu choices. Each choice is associated with a drop-down menu.
- This concept is implemented in the AWT by the following classes: MenuBar, Menu, and MenuItem.
- In general, a menu bar contains one or more Menu objects. Each Menu object contains a list of MenuItem objects. Each MenuItem object represents something that can be selected by the user.
- Since Menu is a subclass of MenuItem, a hierarchy of nested submenus can be created. It is also possible to include checkable menu items. These are menu options of type CheckboxMenuItem and will have a check mark next to them when they are selected.

Steps to Create Menu

- Create menu bar
- Add menu bar to the frame
- Create menus
- Add menus to menu bar
- Create menu items
- Add menu items to menus
- Event handling(If required)

1. Create MenuBar

To create a menu bar, first create an instance of MenuBar. This class defines only the default constructor.

- MenuBar()

2. Create menu

Following are the constructors for Menu:

- Menu( ) throws HeadlessException
- Menu(String optionName) throws HeadlessException
- Menu(String optionName, boolean removable) throws HeadlessException
- Here, optionName specifies the name of the menu selection. If removable is true, the menu can be removed and allowed to float free. Otherwise, it will remain attached to the menu bar. (Removable menus are implementation-dependent.)
- The first form creates an empty menu.

3. Create Menu Item

- Individual menu items are of type MenuItem.
- It defines these constructors:
  - MenuItem( ) throws HeadlessException
  - MenuItem(String itemName) throws HeadlessException
  - MenuItem(String itemName, MenuShortcut keyAccel) throws HeadlessException
  - Here, itemName is the name shown in the menu, and keyAccel is the menu shortcut for this item.
- You can disable or enable a menu item by using the setEnabled( ) method. Its form is shown here:
  - void setEnabled(boolean enabledFlag)
  - If the argument enabledFlag is true, the menu item is enabled. If false, the menu item is disabled.
- You can determine an item's status by calling isEnabled( ). This method is shown here:
  - boolean isEnabled( )

    isEnabled( ) returns true if the menu item on which it is called is enabled. Otherwise, it returns false.

Dailog Box

- Often, you will want to use a dialog box to hold a set of related controls.
- Dialog boxes are primarily used to obtain user input and are often child windows of a top-level window.
- Dialog boxes don’t have menu bars, but in other respects, they function like frame windows. (You can add controls to them, for example, in the same way that you add controls to a frame window.)
- Dialog boxes may be modal or modeless. When a modal dialog box is active, all input is directed to it until it is closed. This means that you cannot access other parts of your program until you have closed the dialog box.
- When a modeless dialog box is active, input focus can be directed to another window in your program. Thus, other parts of your program remain active and accessible.
- In the AWT, dialog boxes are of type Dialog. Two commonly used constructors are shown here:
- Dialog(Frame parentWindow, boolean mode)
- Dialog(Frame parentWindow, String title, boolean mode)

- Here, parentWindow is the owner of the dialog box. If mode is true, the dialog box is modal. Otherwise, it is modeless. The title of the dialog box can be passed in title. Generally, you will subclass Dialog, adding the functionality required by your application.

FileDialog

- Java provides a built-in dialog box that lets the user specify a file.
- To create a file dialog box, instantiate an object of type FileDialog. This causes a file dialog box to be displayed.
- Usually, this is the standard file dialog box provided by the operating system. Here are three FileDialog constructors:
  - FileDialog(Frame parent)
  - FileDialog(Frame parent, String boxName)
  - FileDialog(Frame parent, String boxName, int how)
- Here, parent is the owner of the dialog box.
- The boxName parameter specifies the name displayed in the box's title bar. If boxName is omitted, the title of the dialog box is empty.
- If how is FileDialog.LOAD, then the box is selecting a file for reading.
- If how is FileDialog.SAVE, the box is selecting a file for writing.
- If how is omitted, the box is selecting a file for reading.

- FileDialog provides methods that allow you to determine the name of the file and its path as selected by the user. Here are two examples:
  - String getDirectory( )
  - String getFile( )
- These methods return the directory and the filename, respectively.