



Creating User Interfaces with the awt

An awt Overview

- The basic idea behind the awt is that a graphical Java program is a set of nested components, starting from the outermost window all the way down to the smallest UI component. Components can include things you can actually see on the screen, such as windows, menu bars, buttons, and text fields, and they can also include containers, which in turn can contain other components.



Creating User Interfaces with the awt

An awt Overview

- These are the major components you can work with in the awt:
- *Containers.* Containers are generic awt components that can contain other components, including other containers. The most common form of container is the *panel*, which represents a container that can be displayed onscreen. Applets are a form of panel (in fact, the Applet class is a subclass of the Panel class).
- *Canvases.* A canvas is a simple drawing surface. Although you can draw on panels (as you've been doing all along), canvases are good for painting images or performing other graphics operations.



Creating User Interfaces with the awt

An awt Overview

- *UI components.* These can include buttons, lists, simple pop-up menus, check boxes, text fields, and other typical elements of a user interface.
- *Window construction components.* These include windows, frames, menu bars, and dialog boxes. They are listed separately from the other UI components because you'll use these less often-particularly in applets.



Creating User Interfaces with the awt The Basic User Interface Components

- In this section, you'll learn about the basic UI components: labels, buttons, check boxes, choice menus, and text fields.
- *public void init() {*
- *Button b = new Button("OK");*
- *add(b);*
- *}*



Creating User Interfaces with the awt The Basic User Interface Components

- **Labels**
- A *label* is an uneditable text string that acts as a description for other awt components.
- To create a label, use one of the following constructors:
- `Label()` creates an empty label, with its text aligned left.
- `Label(String)` creates a label with the given text string, also aligned left.
- `Label(String, int)` creates a label with the given text string and the given alignment. The available alignment numbers are stored in class variables in `Label`, making them easier to remember: `Label.RIGHT`, `Label.LEFT`, and `Label.CENTER`



Creating User Interfaces with the awt The Basic User Interface Components

- `import java.awt.*;`
- `public class LabelTest extends java.applet.Applet {`
- `public void init() {`
- `setFont(new Font ("Helvetica", Font.BOLD, 14));`
- `setLayout(new GridLayout(3,1));`
- `add(new Label("aligned left", Label.LEFT));`
- `add(new Label("aligned center", Label.CENTER));`
- `add(new Label("aligned right", Label.RIGHT));`
- `}`
- `}`



Creating User Interfaces with the awt

The Basic User Interface Components

- When you have a Label object, you can use methods defined in the Label class to get and set the values of the text, as shown in Table 13.1.

Table 13.1. Label methods.

Method	Action
■ getText()	Returns a string containing this label's text
■ setText(String)	Changes the text of this label
■ getAlignment()	Returns an integer representing the alignment of
■	0 is Label.LEFT
	1 is Label.CENTER
	2 is Label.RIGHT



Creating User Interfaces with the awt The Basic User Interface Components

- `setAlignment(int)` Changes the alignment of this label to
- the given integer-use the class variables
- listed in the `getAlignment()` method.



Creating User Interfaces with the awt The Basic User Interface Components

- **Buttons**
- The second user interface component to explore is the button. Buttons are simple UI components that trigger some action in your interface when they are pressed. For example, a calculator applet might have buttons for each number and operator, or a dialog box might have buttons for OK and Cancel.



Creating User Interfaces with the awt The Basic User Interface Components

- To create a button, use one of the following constructors:
- `Button()` creates an empty button with no label.
- `Button(String)` creates a button with the given string as a label.
- Once you have a `Button` object, you can get the value of the button's label by using the `getLabel()` method and set the label using the `setLabel(String)` method.



Creating User Interfaces with the awt The Basic User Interface Components

- `public class ButtonTest extends java.applet.Applet {`
- `public void init() {`
- `add(new Button("Rewind"));`
- `add(new Button("Play"));`
- `add(new Button("Fast Forward"));`
- `add(new Button("Stop"));`
- `}`
- `}`



Creating User Interfaces with the awt The Basic User Interface Components

- **Check Boxes**
- *Check boxes* are user-interface components that have two states: on and off (or checked and unchecked, selected and unselected, true and false, and so on). Unlike buttons, check boxes usually don't trigger direct actions in a UI, but instead are used to indicate optional features of some other action.
- Check boxes can be used in two ways:
 - Nonexclusive: Given a series of check boxes, any of them can be selected.
 - Exclusive: Given a series, only one check box can be selected at a time.



Creating User Interfaces with the awt

The Basic User Interface Components

- `Checkbox()` creates an empty check box, unselected.
- `Checkbox(String)` creates a check box with the given string as a label.
- `Checkbox(String, null, boolean)` creates a check box that is either selected or deselected based on whether the boolean argument is true or false, respectively. (The null is used as a placeholder for a group argument. Only radio buttons have groups, as you'll learn in the next section.)



Creating User Interfaces with the awt

The Basic User Interface Components

- `import java.awt.*;`
- `public class CheckboxTest extends java.applet.Applet {`
- `public void init() {`
- `setLayout(new FlowLayout(FlowLayout.LEFT));`
- `add(new Checkbox("Shoes"));`
- `add(new Checkbox("Socks"));`
- `add(new Checkbox("Pants"));`
- `add(new Checkbox("Underwear", null, true));`
- `add(new Checkbox("Shirt"));`
- `}`
- `}`



Creating User Interfaces with the awt

The Basic User Interface Components

- **Radio Buttons**
- Radio buttons have the same appearance as check boxes, but only one in a series can be selected at a time. To create a series of radio buttons, first create an instance of `CheckboxGroup`:
- `CheckboxGroup cbg = new CheckboxGroup();`
- Then create and add the individual check boxes using the constructor with three arguments (the first is the label, the second is the group, and the third is whether that check box is selected). Note that because radio buttons, by definition, have only one in the group selected at a time, the last true to be added will be the one selected by default:
- `add(new Checkbox("Yes", cbg, true)); add(new Checkbox("No", cbg, false));`



Creating User Interfaces with the awt

The Basic User Interface Components

- `import java.awt.*;`
- `public class CheckboxGroupTest extends java.applet.Applet {`
- `public void init() {`
- `setLayout(new FlowLayout(FlowLayout.LEFT));`
- `CheckboxGroup cbg = new CheckboxGroup();`
- `add(new Checkbox("Red", cbg, false));`
- `add(new Checkbox("Blue", cbg, false));`
- `add(new Checkbox("Yellow", cbg, false));`
- `add(new Checkbox("Green", cbg, true));`
- `add(new Checkbox("Orange", cbg, false));`
- `add(new Checkbox("Purple", cbg, false));`
- `}`
- `}`



Creating User Interfaces with the awt

The Basic User Interface Components

- **Choice Menu**
- The choice menu is a more complex UI component than labels, buttons, or check boxes. Choice menus are pop-up (or pull-down) menus from which you can select an item. The menu then displays that choice on the screen. The function of a choice menu is the same across platforms, but its actual appearance may vary from platform to platform.



Creating User Interfaces with the awt

The Basic User Interface Components

- `import java.awt.*;`
- `public class ChoiceTest extends java.applet.Applet {`
- `public void init() {`
- `Choice c = new Choice();`
- `c.addItem("Apples");`
- `c.addItem("Oranges");`
- `c.addItem("Strawberries");`
- `c.addItem("Blueberries");`
- `c.addItem("Bananas");`
- `add(c);`
- `}`
- `}`



Creating User Interfaces with the awt

The Basic User Interface Components

- **Text Fields**
- Unlike the UI components up to this point, which only enable you to select among several options to perform an action, text fields allow you to enter and edit text. Text fields are generally only a single line and do not have scrollbars; text areas, which you'll learn about later today, are better for larger amounts of text.
- Text fields are different from labels in that they can be edited; labels are good for just displaying text, text fields for getting text input from the user.
- *Text fields* provide an area where you can enter and edit a single line of text.



Creating User Interfaces with the awt The Basic User Interface Components

- To create a text field, use one of the following constructors:
- `TextField()` creates an empty `TextField` that is 0 characters wide (it will be resized by the current layout manager).
- `TextField(int)` creates an empty text field. The integer argument indicates the minimum number of characters to display.
- `TextField(String)` creates a text field initialized with the given string. The field will be automatically resized by the current layout manager.



Creating User Interfaces with the awt

The Basic User Interface Components

- For example, the following line creates a text field 30 characters wide with the string "Enter Your Name" as its initial contents:
- `TextField tf = new TextField("Enter Your Name", 30);`
- `add(tf);`

- `add(new Label("Enter your Name"));`
- `add(new TextField("your name here", 45));`
- `add(new Label("Enter your phone number"));`
- `add(new TextField(12));`
- `add(new Label("Enter your password"));`
- `TextField t = new TextField(20);`
- `t.setEchoCharacter('*');`
- `add(t);`



Creating User Interfaces with the awt The Basic User Interface Components

Method	Action
■ getText()	Returns the text this text field contains (as a string)
■ setText(String)	Puts the given text string into the field
■ getColumn(s)	Returns the width of this text field
■ select(int, int) (positions start from 0)	Selects the text between the two integer positions
■ selectAll()	Selects all the text in the field
■ isEditable() editable	Returns true or false based on whether the text is
■ setEditable(boolean)	true (the default) enables text to be edited; false freezes the text
■ getEchoChar()	Returns the character used for masking input
■ echoCharIsSet() masking character	Returns true or false based on whether the field has a



Creating User Interfaces with the awt Panels and Layout

- awt panels can contain UI components or other panels. The question now is how those components are actually arranged and displayed onscreen.
- In other windowing systems, UI components are often arranged using hard-coded pixel measurements-put a text field at the position 10,30, for example-the same way you used the graphics operations to paint squares and ovals on the screen. In the awt, your UI design may be displayed on many different window systems on many different screens and with many different kinds of fonts with different font metrics.



Creating User Interfaces with the awt Panels and Layout

- Therefore, you need a more flexible method of arranging components on the screen so that a layout that looks nice on one platform isn't a jumbled, unusable mess on another.
- For just this purpose, Java has layout managers, insets, and hints that each component can provide to help dynamically lay out the screen.
- Note that the nice thing about awt components and user-interface items is that you don't have to paint them-the awt system manages all that for you. If you have graphical components or images, or you want to create animation inside panels, you still have to do that by hand, but for most of the basic components, all you have to do is put them on the screen and Java will handle the rest.



Creating User Interfaces with the awt Panels and Layout

- **Layout Managers: An Overview**
- The *layout manager* determines how awt components are dynamically arranged on the screen.
- The awt provides five basic layout managers: FlowLayout, GridLayout, BorderLayout, CardLayout, and GridBagLayout. To create a layout manager for a given panel, create an instance of that layout manager and then use the `setLayout()` method for that panel. This example sets the layout manager of the entire enclosing applet panel:



Creating User Interfaces with the awt Panels and Layout

- `public void init() {`
- `setLayout(new FlowLayout());`
- `}`
- Setting the default layout manager, like creating user-interface components, is best done during the applet's initialization, which is why it's included here.



Creating User Interfaces with the awt Panels and Layout

- **The FlowLayout Class**
- The FlowLayout class is the most basic of layouts. Using flow layout, components are added to the panel one at a time, row by row. If a component doesn't fit onto a row, it's wrapped onto the next row. The flow layout also has an alignment, which determines the alignment of each row. By default, each row is centered.
- *Flow layout* arranges components from left to right in rows. The rows are aligned left, right, or centered.
- To create a basic flow layout with a centered alignment, use the following line of code in your panel's initialization (because this is the default pane layout, you don't need to include this line if that is your intent):



Creating User Interfaces with the awt Panels and Layout

- `setLayout(new FlowLayout());` With the layout set, the order in which you add elements to the layout determines their position. The following code creates a simple row of six buttons in a centered flow layout



Creating User Interfaces with the awt Panels and Layout

- `import java.awt.*;`
- `public class FlowLayoutTest extends java.applet.Applet {`
- `public void init() {`
- `setLayout(new FlowLayout());`
- `add(new Button("One"));`
- `add(new Button("Two"));`
- `add(new Button("Three"));`
- `add(new Button("Four"));`
- `add(new Button("Five"));`
- `add(new Button("Six"));`
- `}`
- `}`



Creating User Interfaces with the awt Panels and Layout

- To create a flow layout with an alignment other than centered, add the `FlowLayout.RIGHT` or `FlowLayout.LEFT` class variable as an argument:
- `setLayout(new FlowLayout(FlowLayout.LEFT));` You can also set horizontal and vertical gap values by using flow layouts. The *gap* is the number of pixels between components in a panel; by default, the horizontal and vertical gap values are three pixels, which can be very close indeed. Horizontal gap spreads out components to the left and to the right; vertical gap spreads them to the top and bottom of each component. Add integer arguments to the flow layout constructor to increase the gap. Figure 13.10 shows the result of adding a gap of 30 points in the horizontal and 10 in the vertical directions, like this:
`setLayout(new FlowLayout(FlowLayout.LEFT, 30, 10));`



Creating User Interfaces with the awt Panels and Layout

- **Grid Layouts**
- *Grid layouts* offer more control over the placement of components inside a panel. Using a grid layout, you portion off the display area of the panel into rows and columns. Each component you then add to the panel is placed in a *cell* of the grid, starting from the top row and progressing through each row from left to right (here's where the order of calls to the `add()` method are very relevant to how the screen is laid out).
- To create a grid layout, indicate the number of rows and columns you want the grid to have when you create a new instance of the `GridLayout` class. Here's a grid layout with three rows and two columns



Creating User Interfaces with the awt Panels and Layout

- `import java.awt.*;`
- `public class GridLayoutTest extends java.applet.Applet {`
- `public void init() {`
- `setLayout(new GridLayout(3,2);`
- `add(new Button("One"));`
- `add(new Button("Two"));`
- `add(new Button("Three"));`
- `add(new Button("Four"));`
- `add(new Button("Five"));`
- `add(new Button("Six"));`
- `}`
- `}`



Creating User Interfaces with the awt Panels and Layout

- Grid layouts can also have a horizontal and vertical gap between components. To create gaps, add those pixel values:
- `setLayout(new GridLayout(3, 3, 10, 30));` It shows a grid layout with a 10-pixel horizontal gap and a 30-pixel vertical gap.



Creating User Interfaces with the awt Panels and Layout

- **Border Layouts**
- *Border layouts* behave differently from flow and grid layouts. When you add a component to a panel that uses a border layout, you indicate its placement as a geographic direction: north, south, east, west, or center. (See Figure 13.13.) The components around all the edges are laid out with as much size as they need; the component in the center, if any, gets any space left over.



Creating User Interfaces with the awt Panels and Layout

- `import java.awt.*;`
- `public class BorderLayoutTest extends java.applet.Applet {`
- `public void init() {`
- `setLayout(new BorderLayout());`
- `add("North", new Button("One"));`
- `add("East", new Button("Two"));`
- `add("South", new Button("Three"));`
- `add("West", new Button("Four"));`
- `add("Center", new Button("Five"));`
- `add(new Button("Six"));`
- `}`
- `}`



Creating User Interfaces with the awt Panels and Layout

- Border layouts can also have horizontal and vertical gaps. Note that the north and south components extend all the way to the edge of the panel, so the gap will result in less vertical space for the east, right, and center components. To add gaps to a border layout, include those pixel values in the constructor as with the other layout managers:
- `setLayout(new BorderLayout(10, 10));`



Creating User Interfaces with the awt Panels and Layout

- **Card Layouts**
- **Grid Bag Layouts**



Creating User Interfaces with the awt

More UI Components

- ***Text areas*** are larger, scrollable text-entry components. Whereas text fields only provide one line of text, text areas can hold any amount of editable text.
- To create a text area, use one of the following constructors:
- `TextArea()` creates an empty text area 0 rows long and 0 characters wide (the text area will be automatically resized based on the layout manager).
- `TextArea(int, int)` creates an empty text area with the given number of rows and columns (characters).
- `TextArea(String)` creates a text area displaying the given string, which will be sized according to the current layout manager.
- `TextArea(String, int, int)` creates a text area displaying the given string and with the given dimensions.



Creating User Interfaces with the awt

More UI Components

- **Text area methods** **MethodAction**
- `getColumns()`
- Returns the width of the text area, in characters or columns
- `getRows()`
- Returns the number of rows in the text area (not the number of rows of text that the text area contains)
- `insertText(String, int)`
- Inserts the string at the given position in the text (text positions start at 0)
- `replaceText(String, int, int)`
- Replaces the text between the given integer positions with the new string



Creating User Interfaces with the awt

More UI Components

- **Scrolling Lists**
- ***Scrolling lists* provide a menu of items that can be selected or deselected. Unlike choice menus, scrolling lists are not pop-up menus and can be defined to allow multiple selections.**
- **To create a scrolling list, create an instance of the List class and then add individual items to that list. The List class has two constructors:**
- **List() creates an empty scrolling list that enables only one selection at a time.**
- **List(int, boolean) creates a scrolling list with the given number of visible lines on the screen (you're unlimited as to the number of actual items you can add to the list). The boolean argument indicates whether this list enables multiple selections (true) or not (false).**



Creating User Interfaces with the awt

More UI Components

- import java.awt.*;
- public class ListsTest extends java.applet.Applet {
- public void init() {
- List lst = new List(5, true);
- lst.addItem("Hamlet");
- lst.addItem("Claudius");
- lst.addItem("Gertrude");
- lst.addItem("Polonius");
- lst.addItem("Horatio");
- lst.addItem("Laertes");
- lst.addItem("Ophelia");
-
- add(lst);
- }
- }



Creating User Interfaces with the awt

More UI Components

- `getItem(int)`
 - Returns the string item at the given position
- `countItems()`
 - Returns the number of items in the menu
- `getSelectedIndex()`
 - Returns the index position of the item that's selected (used for lists that allow only single selections)
- `getSelectedIndexes()`
 - Returns an array of index positions (used for lists that allow multiple selections)
- `getSelectedItem()`
 - Returns the currently selected item as a string



Creating User Interfaces with the awt More UI Components

- `getSelectedItems()`
- Returns an array of strings containing all the selected items
- `select(int)`
- Selects the item at the given position
- `select(String)`
- Selects the item with that string