Chapter 25

Introducing the AWT: Working with Windows, Graphics, and Text

Here, you will learn how to manage windows, work with fonts, output text, and utilize graphics. Java's newest GUI framework is JavaFX. It is anticipated that, at some point in the future, JavaFX will replace Swing as Java's most popular GUI.

1. AWT Classes

The AWT classes are contained in the java.awt package. Beginning with JDK 9, java.awt is part of the java.desktop module.

Class	Description
AWTEvent	Encapsulates AWT events.
AWTEventMulticaster	Dispatches events to multiple listeners.
BorderLayout	The border layout manager. Border layouts use five components: North, South, East, West, and Center.
Button	Creates a push button control.
Canvas	A blank, semantics-free window.
CardLayout	The card layout manager. Card layouts emulate index cards. Only the one on top is showing.
Checkbox	Creates a check box control.
CheckboxGroup	Creates a group of check box controls.
CheckboxMenuItem	Creates an on/off menu item.
Choice	Creates a pop-up list.
Color	Manages colors in a portable, platform-independent fashion.
Component	An abstract superclass for various AWT components.
Container	A subclass of **Component** that can hold other components.
Cursor	Encapsulates a bitmapped cursor.
Dialog	Creates a dialog window.
Dimension	Specifies the dimensions of an object. The width is stored in **width**, and the height is stored in **height**.
EventQueue	Queues events.
FileDialog	Creates a window from which a file can be selected.

FlowLayout	The flow layout manager. Flow layout positions components left to right, top to bottom.
Font	Encapsulates a type font.
FontMetrics	Encapsulates various information related to a font. This information helps you display text in a window.
Frame	Creates a standard window that has a title bar, resize corners, and a menu bar.
Graphics	Encapsulates the graphics context. This context is used by the various output methods to display output in a window.
GraphicsDevice	Describes a graphics device such as a screen or printer.
GraphicsEnvironment	Describes the collection of available **Font** and **GraphicsDevice** objects.
GridBagConstraints	Defines various constraints relating to the **GridBagLayout** class.
GridBagLayout	The grid bag layout manager. Grid bag layout displays components subject to the constraints specified by **GridBagConstraints**.
GridLayout	The grid layout manager. Grid layout displays components in a two-dimensional grid.
Image	Encapsulates graphical images.
Insets	Encapsulates the borders of a container.
Label	Creates a label that displays a string.
List	Creates a list from which the user can choose. Similar to the standard Windows list box.
MediaTracker	Manages media objects.
Menu	Creates a pull-down menu.
MenuBar	Creates a menu bar.
MenuComponent	An abstract class implemented by various menu classes.
MenuItem	Creates a menu item.
MenuShortcut	Encapsulates a keyboard shortcut for a menu item.
Panel	The simplest concrete subclass of **Container**.
Point	Encapsulates a Cartesian coordinate pair, stored in **x** and **y**.
Polygon	Encapsulates a polygon.
PopupMenu	Encapsulates a pop-up menu.
PrintJob	An abstract class that represents a print job.
Rectangle	Encapsulates a rectangle.
Robot	Supports automated testing of AWT-based applications.
Scrollbar	Creates a scroll bar control.
ScrollPane	A container that provides horizontal and/or vertical scroll bars for another component.
SystemColor	Contains the colors of GUI widgets such as windows, scroll bars, text, and others.
TextArea	Creates a multiline edit control.
TextComponent	A superclass for **TextArea** and **TextField**.
TextField	Creates a single-line edit control.
Toolkit	Abstract class implemented by the AWT.
Window	Creates a window with no frame, no menu bar, and no title.

Table 25-1 A Sampling of AWT Classes

Reference--Schildt, H. (2018). Java: The Complete Reference. 10th edition. McGraw-Hill Education.

2. Window Fundamentals

The AWT defines windows according to a class hierarchy that adds functionality and specificity with each level. Arguably the two most important window-related classes are Frame and Panel. Frame encapsulates a top-level window and it is typically used to create what would be thought of as a standard application window. Panel provides a container to which other components can be added. Panel is also a superclass for Applet (deprecated as of JDK 9).

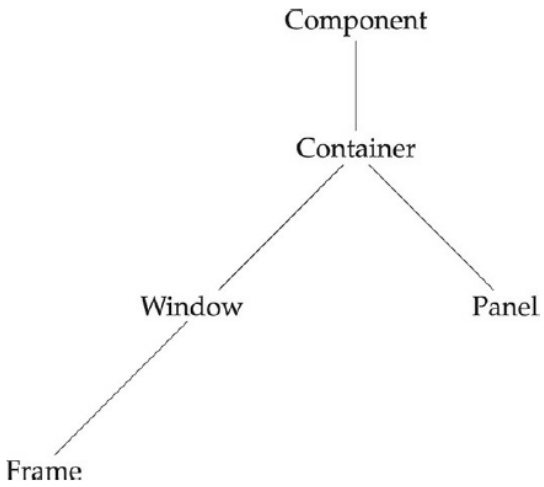

Figure 25-1 The class hierarchy for **Panel** and **Frame**

- **Component** is an abstract class that encapsulates all of the attributes of a visual component (Except for menus). It defines over a hundred public methods that are responsible for managing events, such as mouse and keyboard input, positioning and sizing the window, and repainting. A Component object is responsible for remembering the current foreground and background colors and the currently selected text font.
- **Container** class is a subclass of Component. A container is responsible for laying out (that is, positioning) any components that it contains. It does this through the use of various layout managers.
- **Panel** class is a concrete subclass of Container. Other components can be added to a Panel object by its add( ) method. you can position and resize them manually using the setLocation( ), setSize( ), or setBounds( ) methods defined by Component.
- **Window** class creates a top-level window. Generally, you won't create Window objects directly. Instead, you will use a subclass of Window called Frame.
- **Frame** is a subclass of Window and has a title bar, menu bar, borders, and resizing corners.
- **Canvas** is not part of the hierarchy for Panel or Frame. It is derived from Component, Canvas encapsulates a blank window upon which you can draw.

Reference--Schildt, H. (2018). Java: The Complete Reference. 10th edition. McGraw-Hill Education.

3. Working with Frame Windows

It creates a standard-style, top-level window that has all of the features normally associated with an application window, such as a close box and title. Here are two of Frame's constructors:

Frame( ) throws HeadlessException -- creates a standard window that does not contain a title.

Frame(String title) throws HeadlessException--creates a window with the title specified by title.

You must set the size of the window after it has been created. A HeadlessException is thrown if an attempt is made to create a Frame instance in an environment that does not support user interaction.

3.1 Setting the Window's Dimensions

The setSize( ) method is used to set the dimensions of the window. It is shown here:

- void setSize(int newWidth, int newHeight)
- void setSize(Dimension newSize)

The new size of the window is specified by newWidth and newHeight, or by the width and height fields of the Dimension object passed in newSize. **The dimensions are specified in terms of pixels.**

The getSize( ) method is used to obtain the current size of a window. One of its forms is shown here:

- Dimension getSize( )

3.2 Hiding and Showing a Window

- void setVisible(boolean visibleFlag)

The component is visible if the argument to this method is true. Otherwise, it is hidden.

3.3 Setting a Window's Title

You can change the title in a frame window using setTitle( ), which has this general form:

- void setTitle(String newTitle)

Here, newTitle is the new title for the window.

3.4 Closing a Frame Window

For the main application window, you can simply terminate the program by calling System.exit( ). To intercept a window-close event, you must implement the windowClosing( ) method of the WindowListener interface.

Reference--Schildt, H. (2018). Java: The Complete Reference. 10th edition. McGraw-Hill Education.

3.5 The paint( ) Method

This method is defined by Component and overridden by Container and Window. Thus, it is available to instances of Frame. The paint( ) method is called each time an AWT-based application's output must be redrawn. This situation can occur for several reasons. For example, the window may be minimized and then restored. paint( ) is also called when the window is first displayed. Whatever the cause, whenever the window must redraw its output, paint( ) is called. The paint( ) method is shown here:

- void paint(Graphics context)

3.6 Displaying a String

To output a string to a Frame, use drawString( ), which is a member of the Graphics class. This is the form we will use:

- void drawString(String message, int x, int y)

Here, message is the string to be output beginning at x,y. In a Java window, the upper-left corner is location 0,0. ***The drawString( ) method will not recognize newline characters.*** If you want to start a line of text on another line, you must do so manually, specifying the precise X,Y location where you want the line to begin.

3.7 Setting the Foreground and Background Colors

To set the background color, use setBackground( ). To set the foreground color use setForeground( ). These methods are defined by Component, and they have the following general forms:

- void setBackground(Color newColor)
- void setForeground(Color newColor)

Here, newColor specifies the new color. The class Color defines the constants shown here that can be used to specify colors.

For example, the following sets the background color to green and the foreground color to red:

```
setBackground(Color.green);
setForeground(Color.red);
```

You can obtain the current settings for the background and foreground colors by calling getBackground( ) and getForeground( ), respectively. They are also defined by Component and are shown here:

- Color getBackground( )

- Color getForeground( )

3.8 Requesting Repainting

An application writes to its window only when its paint( ) method is called by the AWT. **This raises an interesting question: How can the program itself cause its window to be updated to display new output?** For example, if a program displays a moving banner, what mechanism does it use to update the window each time the banner scrolls? **Remember, one of the fundamental architectural constraints imposed on a GUI program is that it must quickly return control to the run-time system.** It cannot create a loop inside paint( ) that repeatedly scrolls the banner, for example. This would prevent control from passing back to the AWT. Given this constraint, it may seem that output to your window will be difficult at best.

Whenever your program needs to update the information displayed in its window, it simply calls repaint( ). The repaint( ) method is defined by Component. As it relates to Frame, this method causes the AWT run-time system to execute a call to the update( ) method (also defined by Component). However, the default implementation of update( ) calls paint( ). Thus, to output to a window, simply store the output and then call repaint( ). The AWT will then execute a call to paint( ), which can display the stored information. For example, if part of your program needs to output a string, it can store this string in a String variable and then call repaint( ). Inside paint( ), you will output the string using drawString( ). The repaint( ) method has four forms. Let's look at each one in turn. The simplest version of repaint( ) is shown here:

- void repaint( )

This version causes the entire window to be repainted. The following version specifies a region that will be repainted:

- void repaint(int left, int top, int width, int height)

Here, the coordinates of the upper-left corner of the region are specified by left and top, and the width and height of the region are passed in width and height. These dimensions are specified in pixels. You save time by specifying a region to repaint. Window updates are costly in terms of time. If you need to update only a small portion of the window, it is more efficient to repaint only that region. Calling repaint( ) is essentially a request that a window be repainted sometime

soon. However, if your system is slow or busy, update( ) might not be called immediately. Multiple requests for repainting that occur within a short time can be collapsed by the AWT in a manner such that update( ) is only

called sporadically. This can be a problem in many situations, including animation, in which a consistent update time is necessary. One solution to this problem is to use the following forms of repaint( ):

- void repaint(long maxDelay)
- void repaint(long maxDelay, int x, int y, int width, int height)

Here, maxDelay specifies the maximum number of milliseconds that can elapse before update( ) is called.

Reference--Schildt, H. (2018). Java: The Complete Reference. 10th edition. McGraw-Hill Education.