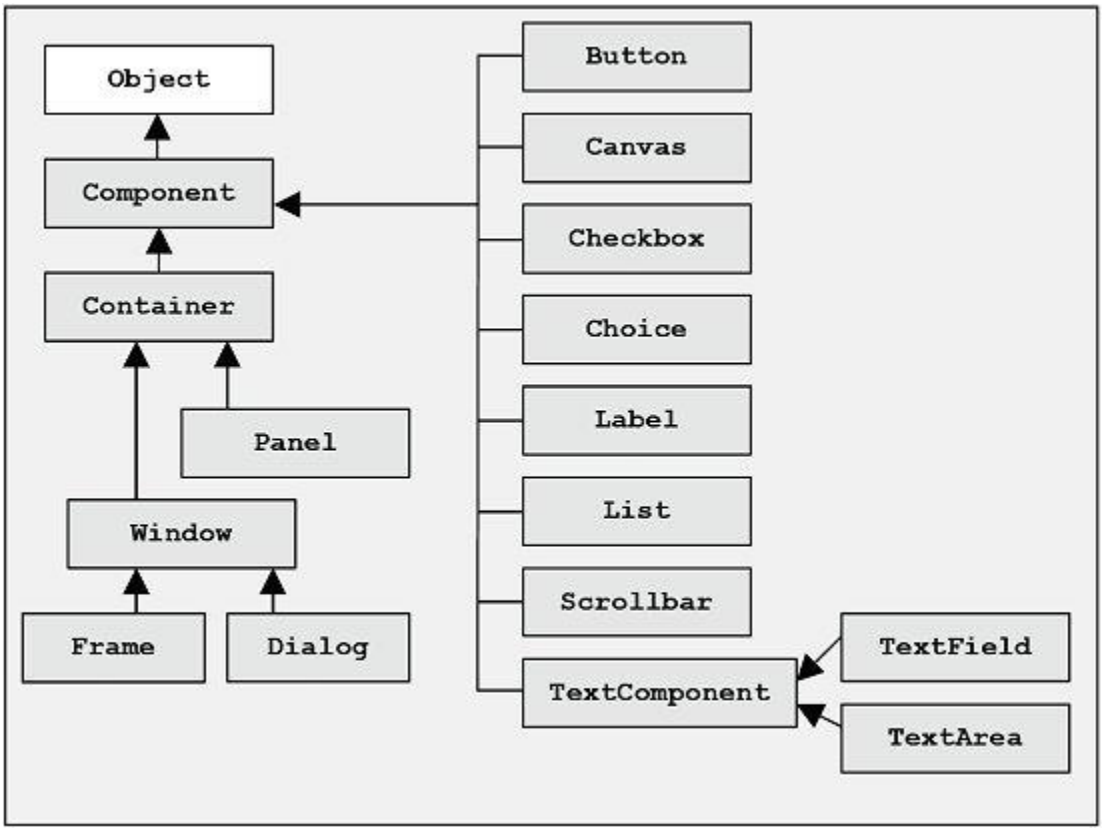
AWT - Abstract Windows Toolkit

The following diagram shows most of the class hierarchy of the AWT (Abstract Windows Toolkit), which comes as part of the core Java language (`java.awt` package and sub-packages.)



The key `Component` class provides the base class for all the AWT visual components and also for the Swing components. The class provides a very large number of public methods - see the API description for `Component`.

The `container` class provides for holding instances of other component classes. The `Window` subclass, for example, provides for the *top level* visible containers, `Frame` and `Dialog`, that hold various visible components.

Containers can hold other containers. The `Panel` class, in particular, is used within a top level container to arrange its sub-components, which often are also panels. An elaborate GUI display with lots of buttons, textfields, and other components will employ several panels and sub-panels to arrange the visible *atomic* (non-container) components.

As seen in the diagram above, the basic AWT includes several atomic components such as buttons, labels, and textfields. You can create fairly elaborate GUI displays with these tools.

Lightweight Wins Over Heavyweight

However, there are a number of limitations to these components. For example, simply creating a subclass of `Button` for a custom button that displays an icon is not practical. Java programs before version1.2 became know for a bland, dull appearance and limited capabilities.

The basic problem is that these components are closely tied to so-called *peer* component classes written in native code for the local operating system GUI. This means that Java portability required a *lowest-common-denominator* approach in which no visible component could provide more capability that what was available on all platforms. This results in very limited options in how the components can look and perform. These basic AWT components are called *heavyweight* because they drag along all the peer component coding.

A far more flexible approach is to open a heavyweight top level class, such as a frame, and then just let Java draw all the visible sub-components without involving any local peer components. Such *lightweight* components are very flexible, especially when combined with the more powerful event handling structure that came with Java on version 1.1.

The Swing set of classes (available in the `javax.swing` and related packages) consists primarily of lightweight components. Swing first became available as an independent set of code that would work with version 1.1 and then was included in the standard Java distribution for version 1.2.

Though there are some drawbacks to Swing), it is now generally recommended that everyone switch their Java user interface design from AWT to Swing for all serious program development for PC and equivalent platforms.

Note: In some cases, such as when developing programs for small platforms, it can be necessary to remain within the older AWT framework.