**AWT and Swing (Unit 5)**

**AWT (abstract window toolkit) :**

Although the AWT is still a crucial part of Java, its component set is  
no longer widely used to create graphical user interfaces. Today, programmers typically use Swing for this purpose.

Swing is a framework that provides more powerful and flexible GUI components than does the AWT.

**What is awt?**

The AWT defines a basic set of controls, windows, and dialog boxes  
that support a usable, but limited graphical interface.

**Why is it not used much?**

One reason for the limited nature of the AWT is that it translates its various visual components into their corresponding, platform-specific equivalents**, or peers.**

This means that the look and feel of a component is defined by the platform, not by Java.

Because the AWT components **use native code resources, they are referred to as heavyweigh**t.

**Issues with heavyweight components/ native peers? \*\*\***

The use of native peers led to several problems. First, because of variations between operating systems, a component might look, or even act, differently on different platforms.

Second, the look and feel of each component was fixed (because it is defined by the platform) and could not be (easily) changed.

Third, the use of heavyweight components caused some frustrating restrictions. For example, a heavyweight component was always  
opaque.

To overcome these issues Swing is used.

**What is swing?**

Swing is a framework that provides more powerful and flexible GUI components than does the AWT.

**Features of Swing? /Advantage of swing over Awt? \*\*\*\*\*\* V.imp**

1. **Lightweight components**

Swing components are **lightweight.** This means that they are written entirely in Java and **do not map directly** to platform-specific peers.

Thus, lightweight components are more **efficient and more flexible.**

Furthermore, because lightweight components do not translate into native  
peers, the look and feel of each component **is determined by Swing**, not by the underlying operating system.

As a result, each component will **work in a** consistent manner across all platforms

1. **Pluggable look and feel.**

Swing supports a **pluggable look and feel (PLAF).** Because each Swing  
component is rendered by Java code rather than by native peers, the look and feel of a component **is under the control of Swing**. This fact means that it is possible to **separate the look and feel** of a component from the logic of the component, and this is what Swing does.

Separating out the look and feel provides a significant advantage: it becomes possible to change the way that a component is rendered without affecting any of its other aspects. In other words, it is possible to “**plug in**” a new look and feel for any given component without creating any side effects in the code that uses that component.

Moreover, it becomes possible to define entire sets of look-and-feels that  
represent different GUI styles. To use a specific style, its look and feel is  
simply “plugged in.” Once this is done, all components are automatically  
rendered using that style.  
Pluggable look-and-feels offer several important advantages. It is possible  
to define a look and feel that is consistent across all **platforms**. Conversely, it is possible to create a look and feel that acts like a specific platform. It is also possible to design a custom look and feel. Finally, the look and feel can be changed dynamically at run time.

\*\*\* IMP

**What is MVC Architecture?**

a visual component is a composite of three distinct aspects:  
• The way that the component looks when rendered on the screen  
• The way that the component reacts to the user  
• The state information associated with the component

The model view controller (MVC) architecture is used to handle these three aspects.

**the model corresponds** to the state information associated with the component. For example, in the case of a check box, the model contains a field that indicates if the box is checked or unchecked.

**The view determines** how the component is displayed on the screen, including any aspects of the view that are affected by the current state of the model.

**The controller determines** how the component reacts to the user. For example, when the user clicks a check box, the controller  
reacts by changing the model to reflect the user’s choice (checked or  
unchecked). This then results in the view being updated.

**Advantage of MVC:**

By separating a component into a model, a view, and a controller, the specific implementation of each can be changed without affecting the other two.

For instance, different view implementations can render the same component in different ways without affecting the model or the controller.

**Implementation of MVC in Swing**:

Swing doesn’t completely separate the view and controller.

Instead, Swing uses a modified version of MVC that **combines the view and the controller** into a single logical entity called **the UI delegate**.

So Model – (View + Controller) becomes

Model – UI Delegate in Swing

It is called either the **Model-Delegate architecture or the Separable Model architecture**.

\*\*\*Important  
**Advantage of UI delegate model:**

look and feel can be changed without affecting how the component is used within a program.

The model can be customized without affecting the way that the component appears on the screen or responds to user input.

To support the Model-Delegate architecture, most Swing components  
contain two objects. The first represents the model. The second represents the UI delegate.

Models are defined by interfaces. For example, the model for a button is defined by the ButtonModel interface.

UI delegates are classes that inherit ComponentUI. For example, the UI delegate for a button is ButtonUI.

**What are Components and Containers**  
A Swing GUI consists of two key items: components and containers.

A *component* is an independent visual control, such as a push button or slider.

A **container** holds a group of components. Thus, a container is a special type of component that is designed to hold other components.

In order for a component to be displayed, it must be held within a container. Thus, all Swing GUIs will have at least one container.

Swing components are derived from the JComponent class.

Swing defines two types of containers. The first are top-level containers:  
JFrame, JApplet, JWindow, and JDialog. These containers do not inherit  
JComponent.

The toplevel containers are heavyweight.

The second type of containers supported by Swing are lightweight  
containers. Lightweight containers do inherit JComponent. An example of a lightweight container is JPanel, which is a general-purpose container

|  |  |  |
| --- | --- | --- |
| **No.** | **Java AWT** | **Java Swing** |
| 1) | AWT components are **platform-dependent**. | Java swing components are **platform-independent**. |
| 2) | AWT components are **heavyweight**. | Swing components are **lightweight**. |
| 3) | AWT **doesn't support pluggable look and feel**. | Swing **supports pluggable look and feel**. |
| 4) | AWT provides **less components** than Swing. | Swing provides **more powerful components** such as tables, lists, scrollpanes, colorchooser, tabbedpane etc. |
| 5) | AWT **doesn't follows MVC**(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view. | Swing **follows MVC**. |