# Swings – A Study

**Tanya Sharma; Sumit Das &Achin Pal**
Computer science and engineering department, MDU Rohtak, India
Email: tanya.elegance1@gmail.com

**ABSTRACT**

*Swing API is set of extensible GUI Components to ease developer's life to create JAVA based Front End/ GUI Applications. It is built upon top of AWT API and acts as replacement of AWT API as it has almost every control corresponding to AWT controls. This paper helps us to understand the concept and features of SWINGS in java.*

## 1) INTRODUCTION

Swing basically consists of :

- A part of The JFC
- Swing Java consists of: Look and feel, Accessibility, Java 2D, Drag and Drop, etc
- Compiling & running programs
- if you do not explicitly add a GUI component to a container, the GUI component will not be displayed when the container appears on the screen.

Swing, which is an extension library to the AWT, includes new and improved components that enhance the look and functionality of GUIs. Swing can be used to build Standalone swing gui Apps as well as Servlets and Applets. It employs a model/view design architecture. Swing is moreportable and more flexible than AWT.

Swing Model/view design: The "view part" of the MV design is implemented with a component object and the UI object. The "model part" of the MV design is implemented by a model object and a change listener object.

Swing is built on top of AWT and is entirely written in Java, using AWT's lightweight component support. In particular, unlike AWT, the architecture of Swing components makes it easy to customize both their appearance and behavior. Components from AWT and Swing can be mixed, allowing you to add Swing support to existing AWT-based programs. For example, swing components such as JSlider, JButton and JCheckbox could be used in the same program with standard AWT labels, textfields and scrollbars. You could subclass the existing Swing UI, model, or change listener classes without having to reinvent the entire implementation. Swing also has the ability to replace these objects on-the-fly.

In Swing, classes that represent GUI components have names beginning with the letter J. Some examples are JButton, JLabel, and JSlider. Altogether there are more than 250 new classes and 75 interfaces in Swing — twice as many as in AWT.

### 1.1) JAVA SWING CLASS HIERARCHY

The class JComponent, descended directly from Container, is the root class for most of Swing's user interface components.

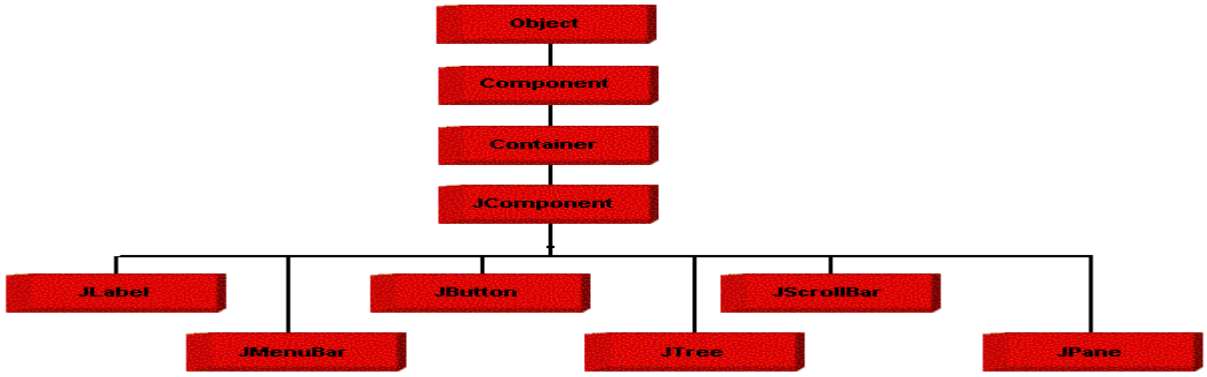


**Fig 1)**

## 2) HISTORY

The Internet Foundation Classes (IFC) were a graphics library for Java originally developed by Netscape Communications Corporation and first released on December 16, 1996. On April 2, 1997, Sun Microsystems and Netscape Communications Corporation announced their intention to incorporate IFC with other technologies to form the Java Foundation Classes. The "Java Foundation Classes" were later renamed "Swing."

Swing introduced a mechanism that allowed the look and feel of every component in an application to be altered without making substantial changes to the application code. The introduction of support for a pluggable look and feel allows Swing components to emulate the appearance of native components while still retaining the benefits of platform independence. Originally distributed as a separately downloadable library, Swing has been included as part of the Java Standard Edition since release 1.2.The Swing classes and components are contained in the javax.swingpackage hierarchy.

## 3) ARCHITECTURE

Swing is a platform-independent, Model-View-ControllerGUI framework for Java, which follows a single-threaded programming model.Additionally, this framework provides a layer of abstraction between the code structure and graphic presentation of a Swing-based GUI.

### 3.1) FOUNDATIONS

Swing is platform-independent because it is completely written in Java. Complete documentation for all Swing classes can be found in the Java API Guide.

### 3.2)EXTENSIBLE

Swing is a highly modular-based architecture, which allows for the "plugging" of various custom implementations of specified framework interfaces: Users can provide their own custom implementation(s) of these components to override the default implementations using Java's inheritance mechanism.

Swing is a component-based framework, whose components are all ultimately derived from the javax.swing.JComponent class. Swing objects asynchronously fire events, have bound properties, and respond to a documented set of methods specific to the component. Swing components are Java Beans components, compliant with the Java Beans Component Architecture specifications.

### 3.3) CUSTOMIZABLE

Given the programmatic rendering model of the Swing framework, fine control over the details of rendering of a component is possible. As a general pattern, the visual representation of a Swing component is a composition of a standard set of elements, such as a border, inset, decorations, and other properties. Typically, users will programmatically customize a standard Swing component (such as a JTable) by assigning specific borders, colors, backgrounds, opacities, etc. The core component will then use these properties to render itself. However, it is also completely possible to create unique GUI controls with highly customized visual representation.

### 3.4) CONFIGURABLE

Swing's heavy reliance on runtime mechanisms and indirect composition patterns allows it to respond at run time to fundamental changes in its settings. For example, a Swing-based application is capable of hot swapping its user-interface during runtime. Furthermore, users can provide their own look and feel implementation, which allows for uniform changes in the look and feel of existing Swing applications without any programmatic change to the application code.

### 3.5) LIGHTWEIGHT UI

Swing's high level of flexibility is reflected in its inherent ability to override the native host operating system (OS)'s GUI controls for displaying itself. Swing "paints" its controls using the Java 2D APIs, rather than calling a native user interface toolkit. Thus, a Swing component does not have a corresponding native OS GUI component, and is free to render itself in any way that is possible with the underlying graphics GUIs.

However, at its core, every Swing component relies on an AWT container, since (Swing's) JComponent extends (AWT's) Container. This allows Swing to plug into the host OS's GUI management framework, including the crucial device/screen mappings and user interactions, such as key presses or mouse movements. Swing simply "transposes" its own (OS-agnostic) semantics over the underlying (OS-specific) components. So, for example, every Swing component paints its rendition on the graphic device in response to a call to component.paint(), which is defined in (AWT) Container. But unlike AWT components, which delegated the painting to their OS-native "heavyweight" widget, Swing components are responsible for their own rendering.

This transposition and decoupling is not merely visual, and extends to Swing's management and application of its own OS-independent semantics for events fired within its component containment hierarchies. Generally speaking, the Swing architecture delegates the task of mapping the various flavors of OS GUI semantics onto a simple, but generalized, pattern to the AWT container. Building on that generalized platform, it establishes its own rich and complex GUI semantics in the form of the JComponent model.

### 3.6) LOOSELY COUPLED AND MVC

The Swing library makes heavy use of the Model/View/Controller software design pattern,which conceptually decouples the data being viewed from the user interface controls through which it is viewed. Because of this, most Swing components have associated *models* (which are specified in terms of Java interfaces), and the programmers can use various default implementations or provide their own. The framework provides default implementations of model interfaces for all of its concrete components. The typical use of the Swing framework does not require the creation of custom models, as the framework provides a set of default implementations that are transparently, by default, associated with the corresponding JComponent child class in the Swing library. In general, only complex components, such as tables, trees and sometimes lists, may require the custom model implementations around the application-specific data structures. To get a good sense of the potential that the Swing architecture makes possible, consider the hypothetical situation where custom models for tables and lists are wrappers over DAO and/or EJB services.

Typically, Swing component model objects are responsible for providing a concise interface defining events fired, and accessible properties for the (conceptual) data model for use by the associated JComponent. Given that the overall MVC pattern is a loosely coupled collaborative object relationship pattern, the model provides the programmatic means for attaching event listeners to the data model object. Typically, these events are model centric (ex: a "row inserted" event in a table model) and are mapped by the JComponentspecialization into a meaningful event for the GUI component.

For example, the JTable has a model called TableModel that describes an interface for how a table would access tabular data. A default implementation of this operates on a two-dimensional array.

The view component of a Swing JComponent is the object used to graphically represent the conceptual GUI control. A distinction of Swing, as a GUI framework, is in its reliance on programmatically rendered GUI controls (as opposed to the use of the native host OS's GUI controls). Prior to Java 6 Update 10, this distinction was a source of complications when mixing AWT controls, which use native controls, with Swing controls in a GUI (see Mixing AWT and Swing components).

Finally, in terms of visual composition and management, Swing favors relative layouts (which specify the positional relationships between components) as opposed to absolute layouts (which specify the exact location and size of components). This bias towards "fluid'" visual ordering is due to its origins in the applet operating environment that framed the design and development of the original Java GUI toolkit. (Conceptually, this view of the layout management is quite similar to that which informs the rendering of HTML content in browsers, and addresses the same set of concerns that motivated the former)

## 4) RELATIONSHIP TO AWT

Since early versions of Java, a portion of the Abstract Window Toolkit (AWT) has provided platform-independent APIs for user interface components. In AWT, each component is rendered and controlled by a native peer component specific to the underlying windowing system.

By contrast, Swing components are often described as lightweight because they do not require allocation of native resources in the operating system's windowing toolkit. The AWT components are referred to as heavyweight components.

Much of the Swing API is generally a complementary extension of the AWT rather than a direct replacement. In fact, every Swing lightweight interface ultimately exists within an AWT heavyweight component because all of the top-level components in Swing (JApplet, JDialog, JFrame, and JWindow) extend an AWT top-level container. Prior to Java 6 Update 10, the use of both lightweight and heavyweight components within the same window was generally discouraged due to Z-order incompatibilities. However, later versions of Java have fixed these issues, and both Swing and AWT components can now be used in one GUI without Z-order issues.

The core rendering functionality used by Swing to draw its lightweight components is provided by Java 2D, another part of JFC.

## 5) RELATIONSHIP TO SWT

The Standard Widget Toolkit (SWT) is a competing toolkit originally developed by IBM and now maintained by the Eclipsecommunity. SWT's implementation has more in common with the heavyweight components of AWT. This confers benefits such as more accurate fidelity with the underlying native windowing toolkit, at the cost of an increased exposure to the native platform in the programming model.

There has been significant debate and speculation about the performance of SWT versus Swing; some hinted that SWT's heavy dependence on JNI would make it slower when the GUI component and Java need to communicate data, but faster at rendering when the data model has been loaded into the GUI, but this has not been confirmed either way.A fairly thorough set of benchmarks in 2005 concluded that neither Swing nor SWT clearly outperformed the other in the general case.

## 6) EXAMPLE

Basically, the idea behind this Hello World program is to learn how to create a java program, compile and run it. To create your java source code you can use any editor( Text pad/Edit plus are my favorites) or you can use an IDE like Eclipse.

```
import javax.swing.JFrame;
import javax.swing.JLabel;

//import statements
//Check if window closes automatically. Otherwise add suitable code
public class HelloWorldFrame extends JFrame {

	public static void main(String args[]) {
		new HelloWorldFrame();
	}
	HelloWorldFrame() {
		JLabel jlbHelloWorld = new JLabel("Hello World");
		add(jlbHelloWorld);
		this.setSize(100, 100);
		// pack();
		setVisible(true);
	}
}
```

**Fig 2)**

**OUTPUT**


**Fig 3)**

## SUMMARY

**Swing** is a GUIwidget toolkit for Java . It is part of Oracle's Java Foundation Classes (JFC) — an API for providing a graphical user interface (GUI) for Java programs.

Swing was developed to provide a more sophisticated set of GUI components than the earlier Abstract Window Toolkit (AWT). Swing provides a native look and feel that emulates the look and feel of several platforms, and also supports a pluggable look and feel that allows applications to have a look and feel unrelated to the underlying platform. It has more powerful and flexible components than AWT. In addition to familiar components such as buttons, check boxes and labels, Swing provides several advanced components such as tabbed panel, scroll panes, trees, tables, and lists.Unlike AWT components, Swing components are not implemented by platform-specific code. Instead they are written entirely in Java and therefore are platform-independent. The term "lightweight" is used to describe such an element.

## DISCLOSURE STATEMENT

There is no financial support for this research work from the funding agency.

## ACKNOWLEDGMENTS

We thank our guide for his timely help, giving outstanding ideas and encouragement to finish this research work successfully.

## SIDE BAR

**Comparison:** it is an act of assessment or evaluation of things side by side in order to see to what extent they are similar or different. It is used to bring out similarities or differences between two things of same type mostly to discover essential features or meaning either scientifically or otherwise.

**Content:** The amount of things contained in something. Things written or spoken in a book, an article, a programme, a speech, etc.

## DEFINITION

- Mapping- an operation that associates each element of a given set (the domain) with one or more elements of a second set (the range).
- Interactions- a particular way in which matter, fields, and atomic and subatomic particles affect one another, e.g. through gravitation or electromagnetism.
- Toolkit- a set of tools, especially one kept in a bag or box and used for a particular purpose.
- Rendition- a performance or interpretation, especially of a dramatic role or piece of music.

## REFERENCE

[1] Matthew Robinson, PavelVorobiev: Swing, Second Edition, Manning, ISBN 1-930110-88-X
[2] David M. Geary: Graphic Java 2, Volume 2: Swing, Prentice Hall, ISBN 0-13-079667-0
[3] John Zukowski: The Definitive Guide to Java Swing, Third Edition, Apress, ISBN 1-59059-447-9
[4] James Elliott, Robert Eckstein, Marc Loy, David Wood, Brian Cole: Java Swing, O'Reilly, ISBN 0-596-00408-7
[5] Kathy Walrath, Mary Campione, Alison Huml, Sharon Zakhour: The JFC Swing Tutorial: A Guide to Constructing GUIs, Addison-Wesley Professional, ISBN 0-201-91467-0
[6] Joshua Marinacci, Chris Adamson: Swing Hacks, O'Reilly, ISBN 0-596-00907-0
[7] Ivan Portyankin, Swing, Effective User Interfaces (Russian)., 2nd Ed., 2010, Moscow, "Lory", ISBN 5-469-00005-2