20CYS383

Java Programming Lab

Case Study on Java Swings

By,

Vishnu Vignesh

CH.EN.U4CYS21093

Semester – IV, Year – II

Acknowledgement

I, Vishnu Vignesh [CH.EN.U4CYS21093], of Cybersecurity Year – II, I would like to thank and express my gratitude to my teaching faculty, Mr. Durga Rao, who has given me the wonderful opportunity of working on creating a Graphical User Interface (GUI) using the concept of Java SWINGS.

His constant motivation and guidance throughout the span of my case study has really helped me a lot, and I am eternally grateful to him for this. His support has been invaluable and I do not think I can thank him enough for all he has done for me.

I would also like to extend my appreciation to our director, Shri. I.B. Manikandan, and our principal, Dr. V. Jayakumar, for always providing us with the right direction and guidance to excel in our studies.

Lastly, I would like to offer my humble pranams to the Almighty and Amma for their constant blessings upon us all.

Thank you once again for everything.

INDEX

Abstract:

The aim of this case study is to demonstrate the creation of a simple and functional graphical user interface (GUI) for a Library Management System using Java Swings. The GUI provides a user-friendly interface for library staff and patrons to interact with the system, allowing for efficient and easy management of library resources.

Through the use of Java Swings, the GUI is responsive and interactive, providing users with real-time updates and feedback. The case study also highlights the importance of proper planning and design in creating an effective GUI, as well as the benefits of using a platform-independent language such as Java.

What are Swings?

Swing is a GUI widget toolkit designed for Java programming language. It is a part of Oracle's Java Foundation Classes (JFC) API, providing a wide range of graphical user interface components to build sophisticated and user-friendly applications.

Swing was created to address the limitations of the earlier Abstract Window Toolkit (AWT) and offers more powerful and flexible components than its predecessor. Swing also provides a customizable look and feel that can emulate the appearance of several platforms or be independent of the underlying platform.

Swing offers an extensive set of components, including advanced components such as tabbed panels, scroll panes, trees, tables, and lists in addition to the standard components such as buttons, check boxes, and labels.

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

Architecture:

Swing is a framework that is based on components, with all components derived from the JComponent class. These components are designed to fire events asynchronously, possess bound properties, and respond to a well-defined set of methods specific to the component.

Swing components are also compliant with the JavaBeans specification, which means that they are JavaBeans components. This compliance allows the components to be easily integrated into other applications and environments that adhere to the JavaBeans standard.

The component-based approach of Swing makes it easier for developers to create complex user interfaces by building them from smaller, reusable components. This results in more modular and maintainable code, as well as providing greater flexibility and extensibility in application development.

Overall, Swing's component-based design, event-driven architecture, and compliance with the JavaBeans specification make it a powerful and versatile framework for building graphical user interfaces in Java applications.

Configurability:

Swing is known for its runtime adaptability, which is achieved through its heavy reliance on runtime mechanisms and indirect composition patterns. This allows Swing to respond to fundamental changes in its settings during runtime, such as swapping the user interface of a Swing-based application.

Swing's adaptability also extends to its ability to accommodate user-provided look and feel implementations. This feature enables uniform changes in the look and feel of existing Swing applications without any programmatic changes to the application code. This is particularly useful when there is a need to apply corporate branding or customized themes to an application.

Swing's ability to adapt to changes at runtime and its support for pluggable look and feel implementations make it a flexible and powerful tool for developing Java applications with dynamic and visually appealing user interfaces.

Lightweight UI:

Swing's flexibility is demonstrated by its ability to override the native host operating system's GUI controls for displaying itself. Instead of calling a native user interface toolkit, Swing "paints" its controls using the Java 2D APIs. This means that Swing components do not have a corresponding native OS GUI component and can render themselves in any way possible with the underlying graphics GUIs.

Although Swing components do not rely on native OS GUI components, they do rely on an AWT container, as Swing's JComponent extends AWT's Container. This allows Swing to plug into the host OS's GUI management framework, including the device/screen mappings and user interactions such as key presses or mouse movements. Swing transposes its own OS-agnostic semantics over the underlying OS-specific components. 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. However, unlike AWT components, which delegate painting to their OS-native "heavyweight" widget, Swing components are responsible for their own rendering.

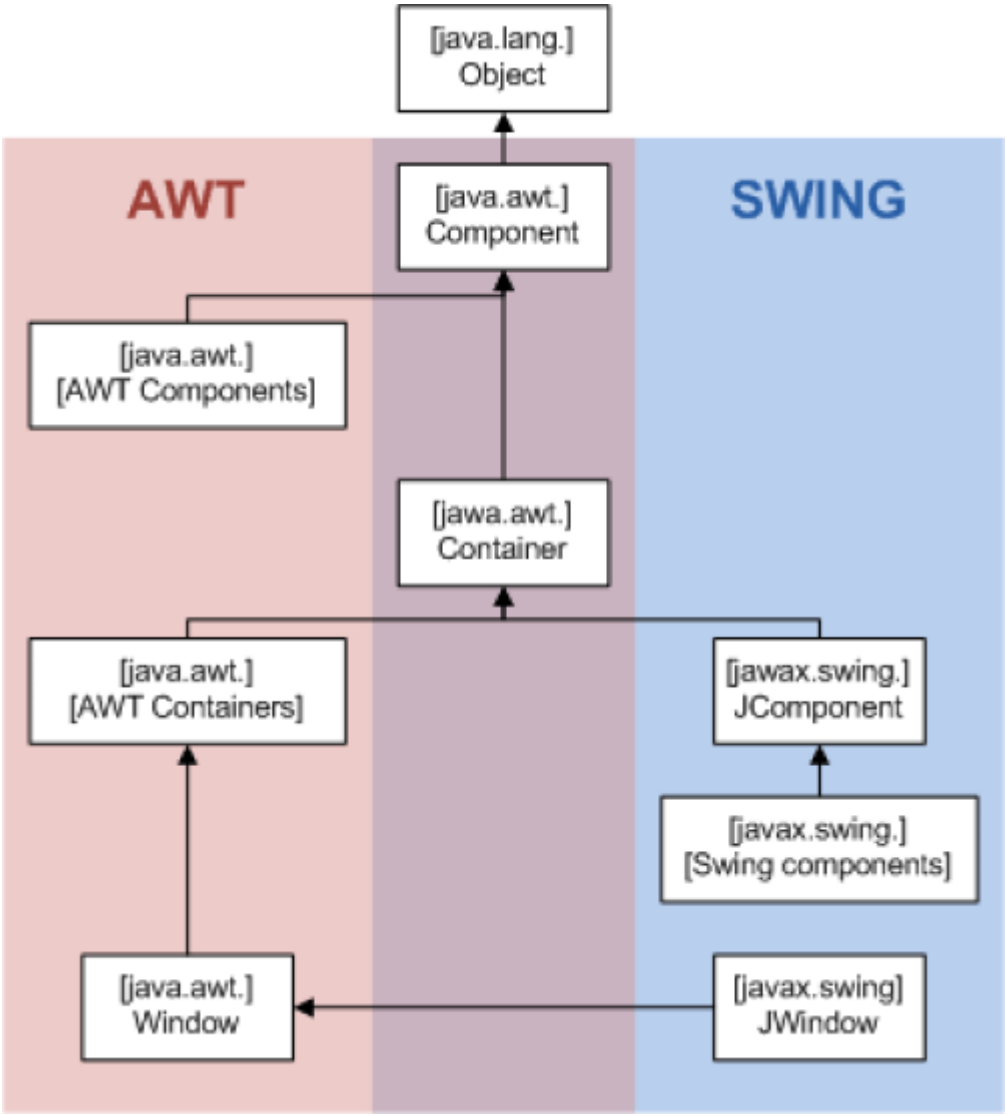
Swing's transposition and decoupling extend beyond the visual display of its components to the management and application of its own OS-independent semantics for events fired within its component containment hierarchies. Swing architecture delegates the mapping of various OS GUI semantics onto a simple, generalized pattern to the AWT container. Building on that platform, Swing establishes its own rich and complex GUI semantics in the form of the JComponent model.

Relationship between Swings and AWT:

Swing is built on top of AWT (Abstract Window Toolkit) and extends AWT. At the core of Swing, 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.

However, Swing does not use AWT's heavyweight components for painting. Instead, Swing uses the Java 2D APIs for painting its controls, allowing it to override the native host operating system's GUI controls for displaying itself.

Thus, Swing components do not have corresponding native OS GUI components and are free to render themselves in any way that is possible with the underlying graphics GUIs.



Components of Swing Class and the Task's Percentage:

Head	Description
Component	A Component is the Abstract base class for about the non menu user-interface controls of SWING. Components are represents an object with graphical representation.
Container	A Container is a component that can container SWING Components.
JComponent	A JComponent is a base class for all swing UI Components In order to use a swing component that inherits from

	JComponent, component must be in a containment hierarchy whose root is a top-level Swing container
JLabel	A JLabel is an object component for placing text in a container.
JButton	This class creates a labeled button.
JColorChooser	A JColorChooser provides a pane of controls designed to allow the user to manipulate and select a color
JCheckBox	A JCheckBox is a graphical (GUI) component that can be in either an on-(true) or off-(false) state.
JRadioButton	The JRadioButton class is a graphical(GUI) component that can be in either an on-(true) or off-(false) state in the group.
JList	A JList component represents the user with the scrolling list of text items.
JComboBox	A JComboBox component is Presents the User with a show up Menu of choices.
JTextField	A JTextField object is a text component that will allow for the editing of a single line of text.
JPasswordField	A JPasswordField object it is a text component specialized for password entry.
JTextArea	A JTextArea object s a text component that allows for the editing of multiple lines of text.
ImageIcon	A ImageIcon control is an implementation of the Icon interface that paints Icons from Images.
JScrollbar	A JScrollbar control represents a scroll bar component in order to enable users to Select from range values.
JOptionPane	JOptionPane provides set of standard dialog boxes that prompt users for a value or something.
JFileChooser	A JFileChooser it Controls represents a dialog window from which the user can select a file.
JProgressBar	As the task progresses towards completion, the progress bar displays the tasks percentage on its completion.
JSlider	A JSlider this class is lets the user graphically (GUI) select by using a value by sliding a knob within a bounded interval.
JSpinner	A JSpinner this class is a single line input where the field that lets the user select by using.
JComboBox	A JComboBox component is Presents the User with a show up Menu of choices.

Code:

https://github.com/VishnuV7845/JavaSem4_CaseStudy_21093

Output: