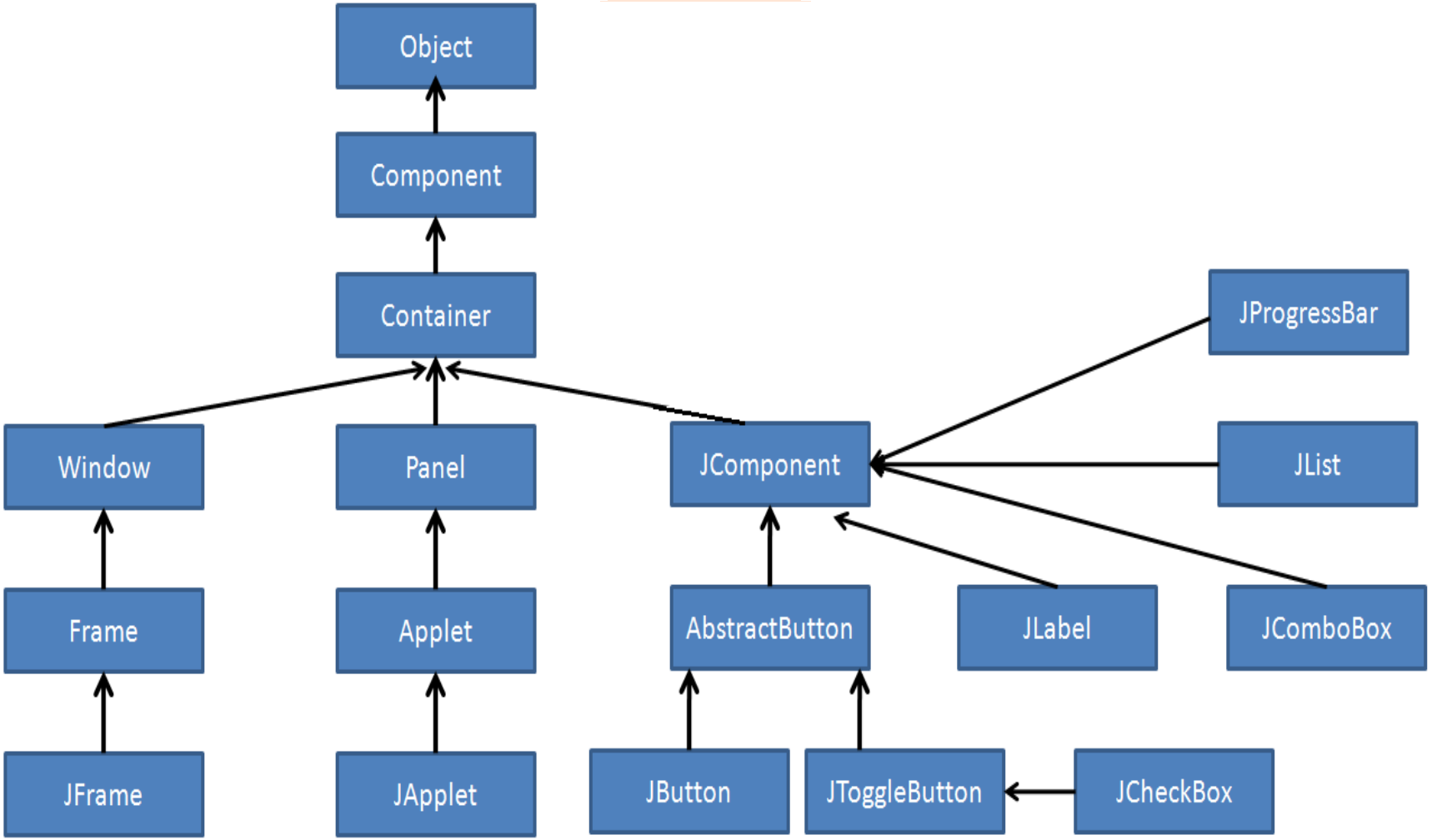
Swing Introduction

Mr. N. A. Anwat

Swing

1. Set of **API** (API –Set of Classes and Interfaces)
2. Provided to design a **Graphical User Interface**
3. Swing is part of the **Java Foundation Classes** (JFC)
4. An **extension library** to the AWT
5. Includes new and improved components that enhance the **look and functionality** of GUIs
6. Swing can be used to build **Standalone** swing GUI Apps as well as **Servlets and Applets.**
7. It employs **model/view** design architecture.
8. Swing is more **portable** and more **flexible** than AWT.
9. Swing is **built on top of AWT**
10. **Entirely written in Java** (means do not use OS resources)
11. Swing features:
    1. **Plugable look & feel**
    2. **Lightweight components**
    3. **Uses MVC architecture**
    4. **Platform Independant**
    5. **Advance features such as JTable, JTabbedPane, JScrollPane etc**

Swing Classes Hierarchy



AWT vs. Swing

AWT	Swing
Abstract window toolkit	Extended version of AWT
Heavyweight	Lightweight
Java.awt package	Javax.swing package
Platform dependant	Platform independent
OS dependant look & feel	Pluggable look & feel
Uses peer objects of OS for each components	Entirely written in java
Uses simple architecture	Uses MVC architecture
Simple and less portable	Portable and flexible

Swing – Important Points

Swing Is Built on the AWT

1. Although swing **eliminates a number of the limitations** inherent in the AWT, Swing *does not* replace it.
2. Instead, **Swing is built on the foundation of the AWT**.
3. This is why the **AWT is still a crucial part** of Java.
4. Swing also uses the **same event handling** mechanism as the AWT.
5. a basic understanding of the **AWT and of event handling** is required to use **Swing**

The MVC Connection

1. In general, a visual component is a composite of three distinct aspects:
    1. The way that the component **looks** when rendered on the screen
    2. The way that the component **reacts** to the user
    3. The **state** information associated with the component
2. Over the years, one component architecture has proven itself to be exceptionally effective: **Model-View-Controller or MVC** for short
3. In MVC terminology, the **model** corresponds to **the state information** associated with the component.
4. 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.
5. The **controller** determines how the component **reacts** to the user
6. 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

Swing – Components and Containers

1. A Swing GUI consists of two key items: ***components*** **and** ***containers***.
2. However, this distinction is mostly conceptual because **all containers** are also **components**.
3. The difference between the two is found in their intended purpose:
    1. A ***component*** is an independent visual control, such as a push button
    2. A ***container*** holds a group of components.
4. Thus, a container is a special type of component that is designed to hold other components
5. Furthermore, in order for a component to be displayed, it must be held within a container.
6. Thus, **all Swing GUIs will have at least one container.**
7. Because containers are components, a container can also hold other containers.
8. This enables Swing to define what is called **a *containment hierarchy***, at the top of which must be a ***top-level container***.

Swing – Components

1. Swing components are derived from the **JComponent** class
2. The only exceptions to this are the four top-level containers: **JFrame, JApplet, JWindow, JDialog**
3. **JComponent** provides the functionality that is common to all components
4. JComponent inherits the AWT classes **Container and Component.**
5. Thus, a Swing component is built on and compatible with an AWT

JApplet	JButton	JCheckBox	JCheckBoxMenuItem
JColorChooser	JComboBox	JComponent	JDesktopPane
JDialog	JEditorPane	JFileChooser	JFormattedTextField
JFrame	JInternalFrame	JLabel	JLayeredPane
JList	JMenu	JMenuBar	JMenuItem
JOptionPane	JPanel	JPasswordField	JPopupMenu
JProgressBar	JRadioButton	JRadioButtonMenuItem	JRootPane
JScrollBar	JScrollPane	JSeparator	JSlider
JSpinner	JSplitPane	JTabbedPane	JTable
JTextArea	JTextField	JTextPane	JTogglebutton
JToolBar	JToolTip	JTree	JViewport
JWindow			

Swing – Containers

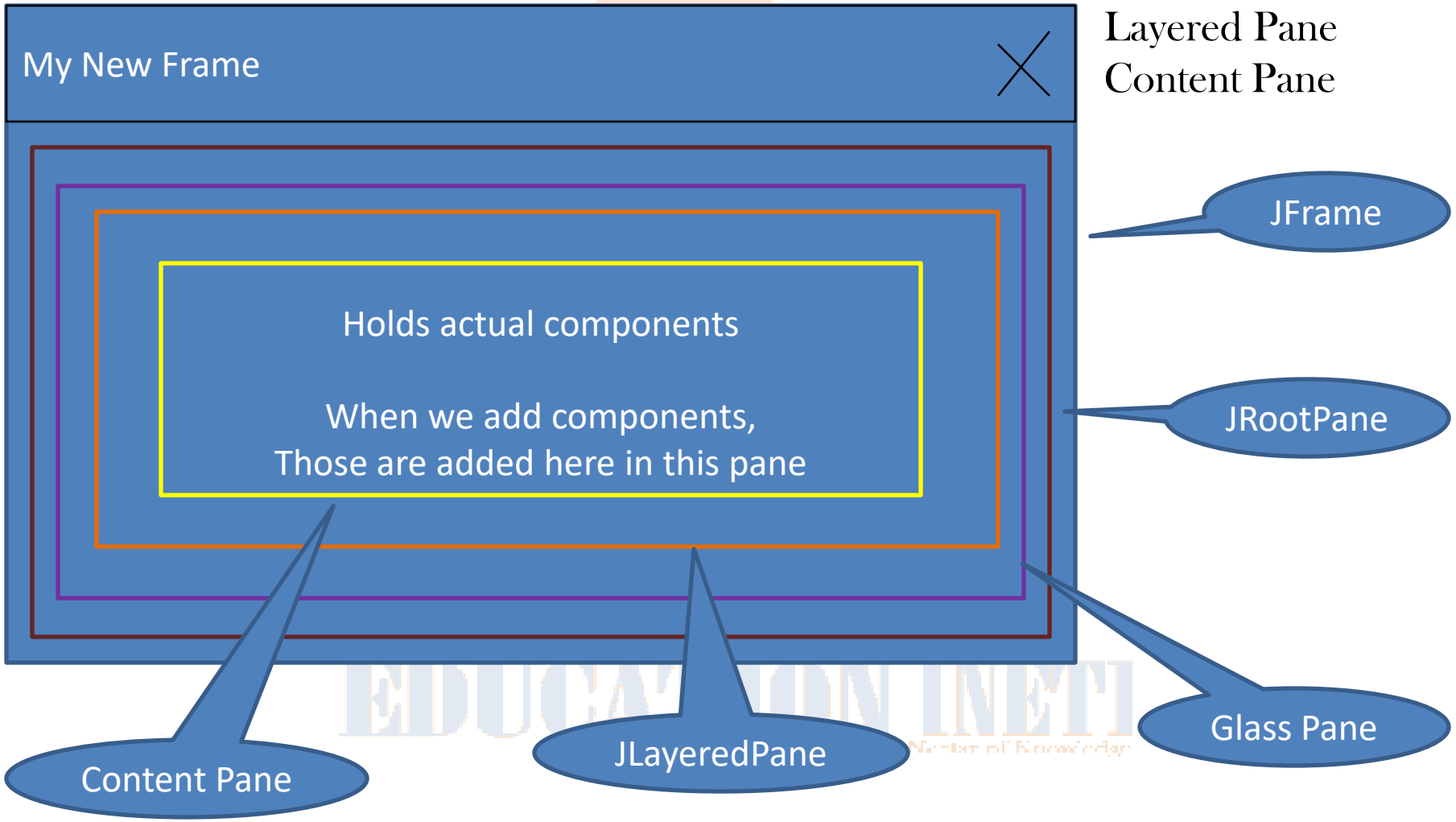
1. Swing defines two types of containers:
    1. The first are top-level containers: **JFrame, JApplet, JWindow**, and **JDialog**.
    2. The second type of containers are lightweight containers: **JPanel, JTabbedPane, JScrollPane etc.**

The Top-Level Container Panes

1. **Each top-level container defines a set of *panes***
2. At the top of the hierarchy is an instance of **JRootPane.**
3. **JRootPane** is a lightweight container whose purpose is to manage the other panes. It also helps manage the optional menu bar
4. **JRootPane contains:**
    1. **Glass Pane**: This enables you to manage mouse events that affect the entire container
    2. **Layered Pane**: This allows components to be given a depth value. This value determines which component overlays another. It contains Content Pane and Menu Bar.
    3. **Content Pane**: This pane holds the visual components such as buttons, labels, textfields
5. Generally we won't use glass pane and layered pane but it can serve the purpose when any one wants to use it.

Swing – Containers

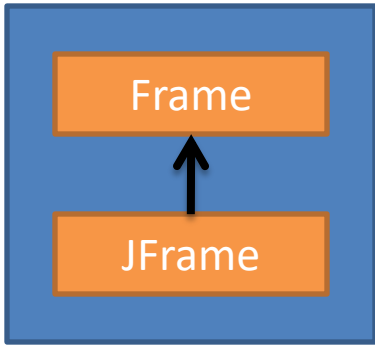
JRootPane contains:
Glass Pane
Layered Pane
Content Pane



My First Swing Frame

```
import java.awt.*;
import javax.swing.*;
 public class MySwingFrame extends JFrame {
            public MySwingFrame(){
                        setTitle("My Swing App");
                        setSize(800,600);
                        setVisible(true);
                        setLayout(new FlowLayout(FlowLayout.LEFT));
                        // no need to handle window closing event
                        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                                    //JFrame.DISPOSE_ON_CLOSE
                                    //JFrame.HIDE_ON_CLOSE
                                    //JFrame.DO_NOTHING_ON_CLOSE
            }
             public static void main(String[] args) {
                        MySwingFrame newFrame=new MySwingFrame();
            }
}
```

MySwingFrame Inherits properties of JFrame



Set basic things like AWT frame

Window closing event handler

Object of my own class

My First Swing Frame

```
import java.awt.*;
import javax.swing.*;
 public class MySwingFrame extends JFrame {
public MySwingFrame(){
                setTitle("My Swing App");
                setSize(800,600);
                setVisible(true);
                setLayout(new FlowLayout(FlowLayout.LEFT));
                // no need to handle window closing event
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                                //JFrame.DISPOSE_ON_CLOSE
                                //JFrame.HIDE_ON_CLOSE
                                //JFrame.DO_NOTHING_ON_CLOSE
}
 public static void main(String[] args) {
                SwingUtilities.invokeLater(new Runnable() {
                                public void run() {
                                                MySwingFrame newFrame=new MySwingFrame();
                                }
                });
}
}
```

In this example two threads working separately:

1. main() thread
2. Event dispatching thread

This code section starts swing frame on event dispatching thread which is separate from main() thread

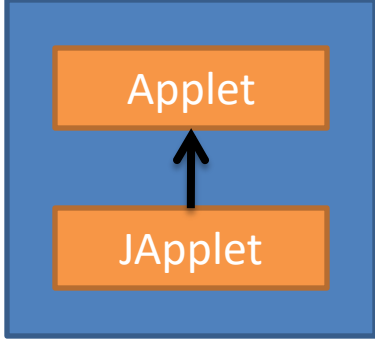
Event dispatching thread

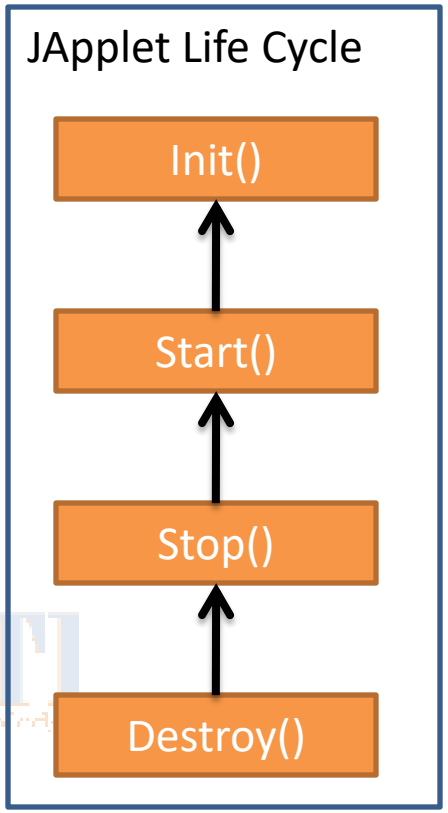
static void invokeLater(Runnable *obj*)
static void invokeAndWait(Runnable *obj*)

My Swing Applet

```
import javax.swing.*; import java.awt.*; import java.awt.event.*;
//      <object code="MySwingApplet" width=220 height=90>  </object>
public class MySwingApplet extends Japplet {
        JButton jbtnAlpha, JButton jbtnBeta;
        JLabel jlab;
        public void init() {
                setLayout(new FlowLayout());
                jbtnAlpha = new JButton("Alpha");
                jbtnBeta = new JButton("Beta");
                jbtnAlpha.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent le) {
                            jlab.setText("Alpha was pressed.");
                        }
                });
                jbtnBeta.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent le) {
                                jlab.setText("Beta was pressed.");
                        }
                });
                add(jbtnAlpha); add(jbtnBeta);
                jlab = new JLabel("Press a button."); add(jlab);
}}
```

Init() method of applet is overridden





Thank You

Any Queries ?