Java-Unit-4

17 April 2025 06:16

Q1. What is a Layout manager? Explain different types of Layouts
Q6. List and explain different types of Layout managers with suitable examples
Q3. Illustrate the use of Grid Bag layout

Ans:

Layout managers define how components are arranged within a container, such as a JFrame or Jpanel
In Java, graphical user interfaces (GUIs) play a vital role in creating interactive applications. To design a visually appealing and organized interface, the choice of layout manager becomes crucial.

Java provides several layout managers to suit various design needs.

1. FlowLayout
2. BorderLayout
3. GridLayout
4. CardLayout
5. GroupLayout
6. GridBagLayout

Layout Managers The LayoutManagers are used to arrange components in a particular manner.

1.Flow Layout
The FlowLayout is used to arrange the components in a line, one after another (in a flow).
It is the default layout of applet or panel.

Example program:

```
Import java.awt.*;
Import javax.swing.*;
public class Demo23
{
public static void main(String args[])
 {
JFrame f=new JFrame();
f.setLayout(new FlowLayout(FlowLayout.LEFT));
JButton b1=new JButton("1");
JButton b2=new JButton("2");
JButton b3=new JButton("3");
JButton b4=new JButton("4");
JButton b5=new JButton("5");
JButton b6=new JButton("6");
JButton b7=new JButton("7");
JButton b8=new JButton("8");
JButton b9=new JButton("9");
JButton b10=new JButton("10");
f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);
f.add(b6);f.add(b7);f.add(b8);f.add(b9);f.add(b10);
f.setSize(300,300);
f.setVisible(true);
 }
}
```

2. Border Layout :

   The BorderLayout is used to arrange the components in five regions: north, south, east, west and center.
   Each region (area) may contain one component only. It is the default layout of frame or window.

```
Example Program:
Import java.awt.*;
Import javax.swing.*;
class Demo21
{
public static void main(String[] args)
 {
JFrame f=new JFrame();
JButton b1=new JButton("A");;
JButton b2=new JButton("B");;
JButton b3=new JButton("C");;
JButton b4=new JButton("D");;
JButton b5=new JButton("E");;
f.add(b1,BorderLayout.NORTH);
f.add(b2,BorderLayout.SOUTH);
f.add(b3,BorderLayout.EAST);
f.add(b4,BorderLayout.WEST);
f.add(b5,BorderLayout.CENTER);
f.setSize(300,300);
f.setVisible(true);
 }
}
```

3. Grid Layout The GridLayout is used to arrange the components in rectangular grid. One component is displayed in each rectangle.

```
Example program:
importjava.awt.*;
importjavax.swing.*;
public class Demo22
{
public static void main(String[] args)
 {
JFrame f=new JFrame();
JButton b1=new JButton("1");
JButton b2=new JButton("2");
JButton b3=new JButton("3");
JButton b4=new JButton("4");
JButton b5=new JButton("5");
JButton b6=new JButton("6");
JButton b7=new JButton("7");
JButton b8=new JButton("8");
JButton b9=new JButton("9");
f.setLayout(new GridLayout(3,3));
f.add(b1);
f.add(b2);f.add(b3);f.add(b4);f.add(b5);
f.add(b6);f.add(b7);f.add(b8);f.add(b9);
f.setSize(300,300);
f.setVisible(true);
 }
}
```

4. Card Layout :

The CardLayout class manages the components in such a manner that only one component is visible at a time.
It treats each component as a card that is why it is known as CardLayout.

```
Example Program:
importjava.awt.*;
importjava.awt.event.*;
importjavax.swing.*;
public class Demo24 extends JFrame implements ActionListener
{
CardLayout card;
JButton b1,b2,b3;
Container c;
Demo24()
 {
c=getContentPane();
card=new CardLayout(40,30);
 //create CardLayout object with 40 hor space and 30 ver space
c.setLayout(card);
 b1=new JButton("Apple");
 b2=new JButton("Boy");
 b3=new JButton("Cat");
b1.addActionListener(this);
b2.addActionListener(this);
b3.addActionListener(this);
c.add("a",b1);c.add("b",b2);c.add("c",b3);
 }
public void actionPerformed(ActionEvent e)
 {
card.next(c);
 }
public static void main(String[] args)
 {
 Demo24 f = new Demo24();
f.setSize(400,400);
f.setVisible(true);
f.setDefaultCloseOperation(EXIT_ON_CLOSE);
 }
}
```

GridBagLayout

The Java GridBagLayout class is used to align components vertically, horizontally or along their baseline.
The components may not be of same size. Each GridBagLayout object maintains a dynamic, rectangular grid of cells.
Each component occupies one or more cells known as its display area.
Each component associates an instance of GridBagConstraints.
With the help of constraints object we arrange component's display area on the grid.
The GridBagLayout manages each component's minimum and preferred sizes in order to determine component's size.

Example program:

```
importjava.awt.Button;
importjava.awt.GridBagConstraints;
importjava.awt.GridBagLayout;
importjavax.swing.*;
class Demo25 extends JFrame
{
public static void main(String args[])
 {
JFrame f=new JFrame();
GridBagLayout grid = new GridBagLayout();
GridBagConstraintsgbc = new GridBagConstraints();
f.setLayout(grid);
gbc.fill = GridBagConstraints.HORIZONTAL;
gbc.gridx = 0;
gbc.gridy = 0;
gbc.ipady=20;
f.add(new Button("Button One"), gbc);
gbc.gridx = 1;
gbc.gridy = 0;
JButton b1=new JButton("Button One");
f.add(b1, gbc);
gbc.fill = GridBagConstraints.HORIZONTAL;
gbc.gridx = 0;
gbc.gridy = 1;
f.add(new Button("Button Three"), gbc);
gbc.gridx = 1;
gbc.gridy = 1;
JButton b4=new JButton("Button Four");
f.add(b4, gbc);
gbc.gridx = 0;
gbc.gridy = 2;
gbc.fill = GridBagConstraints.HORIZONTAL;
gbc.gridwidth = 2;
f.add(new Button("Button Five"), gbc);
f.setSize(300, 300);
f.setVisible(true);
f.setDefaultCloseOperation(EXIT_ON_CLOSE);
 }
}
```

===

Q4. What are the subclasses of JButton class of swing package?

The JButton class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed. It inherits AbstractButton class.

Direct Known Subclasses:

BasicArrowButton,

MetalComboBoxButton

A key element of graphical user interfaces (GUIs) in Java that is used to create interactive buttons is the JButton class

It is descended from the AbstractButton class, which offers shared functionality for all button kinds in the Swing GUI framework and Java's Abstract Window Toolkit (AWT).

===

Q5. Write a short note on delegation event model.
Q10. Explain the event-handling mechanism in java with an example

An **event** is one of the most important concepts in Java. The change in the state of an object or behavior by performing actions is referred to as **an Event** in Java. Actions include button click, keypress, page scrolling, or cursor movement.
Java provides a package **java.awt.event** that contains several event classes.
We can classify the events in the following two categories:

1. Foreground Events
2. Background Events

Foreground Events

Foreground events are those events that require user interaction to generate. In order to generate these foreground events, the user interacts with components in GUI. When a user clicks on a button, moves the cursor, and scrolls the scrollbar, an event will be fired.

Background Events

Background events don't require any user interaction. These events automatically generate in the background. OS failure, OS interrupts, operation completion, etc., are examples of background events.

Delegation Event Model

A mechanism for controlling the events and deciding what should happen after an event occur is referred to as event handling. Java follows the **Delegation Event Model** for handling the events.
The **Delegation Event Model** consists **of Source** and **Listener**.
Source
Buttons, checkboxes, list, menu-item, choice, scrollbar, etc., are the sources from which events are generated.
Listeners
The events which are generated from the source are handled by the listeners. Each and every listener represents interfaces that are responsible for handling events.

We need to register the source with the listener for handling events. Different types of classes provide different registration methods.

Examples:

Handling Mouse Events

```
importjava.awt.*;
importjava.awt.event.*;
importjava.applet.*;
/*
<applet code="evnt0" width=300 height=100>
</applet>
*/
public class evnt0 extends Applet implements MouseListener
{
String msg = "";
intmouseX = 0, mouseY = 0;
public void init()
 {
this.addMouseListener(this);
 }
public void mouseClicked(MouseEvent me)
 {
mouseX = me.getX();
mouseY = me.getY();
msg = "PAVAN";
repaint();
 }
public void mouseEntered(MouseEvent me) { }
public void mouseExited(MouseEvent me) { }
public void mousePressed(MouseEvent me) { }
public void mouseReleased(MouseEvent me) { }
public void paint(Graphics g)
 {
g.drawString(msg, mouseX, mouseY);
 }
}
```

Handling Keyboard Events

```
importjava.awt.*;
importjava.awt.event.*;
importjava.applet.*;
/*
<applet code="evnt2" width=300 height=100>
</applet>
*/
public class evnt2 extends Applet implements KeyListener
{
int X=20,Y=30;
 String msg="chars :";
public void init()
 {
addKeyListener(this);
 }
public void keyPressed(KeyEventke) { }
public void keyReleased(KeyEventke) { }
public void keyTyped(KeyEventke)
 {
msg += ke.getKeyChar();
repaint();
 }
public void paint(Graphics g)
 {
g.drawString(msg, X, Y);
 }
}
```

==

Q7. What are the various components of Swing? Explain.

Java Swing tutorial is a part of Java Foundation Classes (JFC) that is used to create window-based applications. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in Java.
.,m

Key Features of Java Swing:

1. Platform independent
2. Rich set of components
3. Customizability
4. Event Handling
5. Layout Managers

Modern Applications of Java Swing

1. Desktop Applications
2. Legacy Systems
3. Educational Purposes

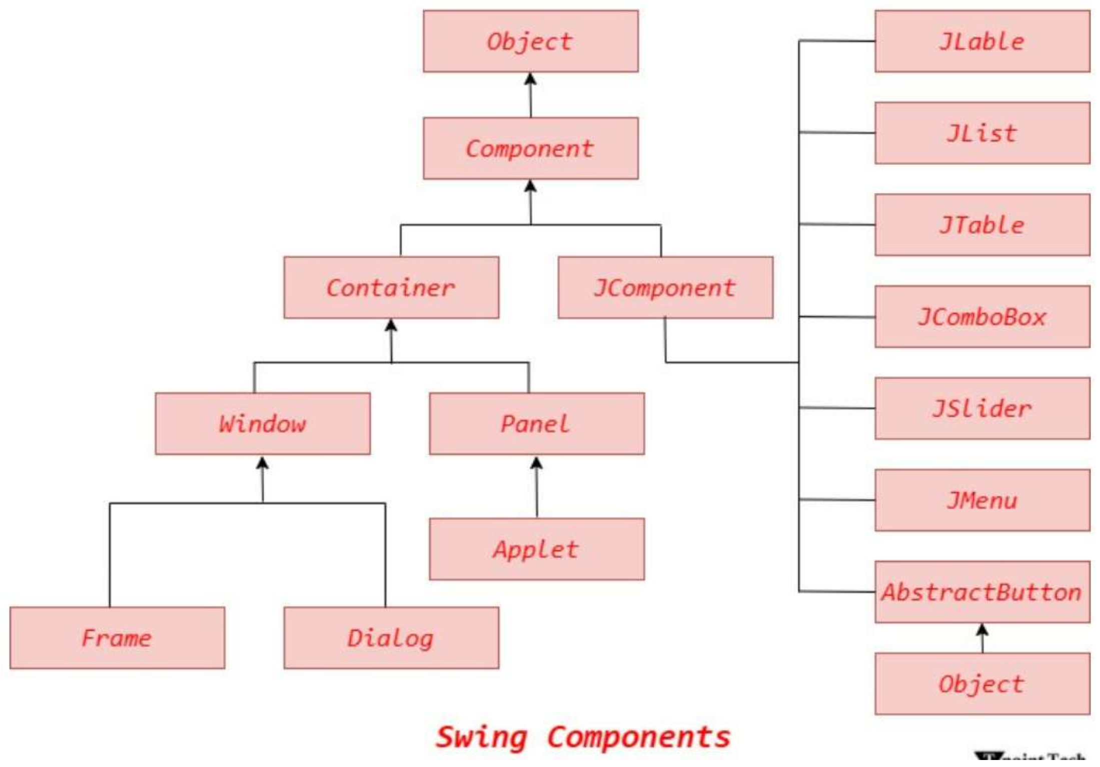
What is JFC

The Java Foundation Classes (JFC) are a set of GUI components which simplify the development of desktop applications.

Hierarchy of Java Swing classes

The hierarchy of Java swing API is given below.

Swing Components:



==

Difference Between AWT and Swing

The following table describes the key differences between AWT and Swing.

Java AWT	Java Swing

AWT components are **platform-dependent**.	Java swing components are **platform-independent**.
AWT components are **heavyweight**.	Swing components are **lightweight**.
AWT **does not support pluggable look and feel**.	Swing **supports pluggable look and feel**.
AWT provides **less components** than Swing.	Swing provides **more powerful components** such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
AWT **does not follows MVC**(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	

Difference Between AWT and Swing

We have discussed the major differences between AWT and Swing in the following table.

S.N.	Aspect	AWT	Swing
1	**Component Weight**	It offers heavyweight components that depend on native OS resources.	It offers lightweight components rendered by Java, reducing OS dependency.
2	**Number of Components**	It provides a smaller number of components in comparison to Swing.	It provides a greater number of components than AWT, such as list, scroll panes, tables, color choosers, etc.
3	**Platform Dependency**	The Components used in AWT are mainly dependent on the operating system.	The Components used in Swing are not dependent on the operating system. It is completely scripted in Java.
4	**Appearance**	The Appearance of AWT Components is mainly not configurable. It generally depends on the operating system's look and feels.	The Swing Components are configurable and mainly support pluggable look and feel.
5	**Peers**	Java AWT has 21 peers. There is one peer for each control and one peer for the dialogue. Peers are provided by the operating system in the form of widgets themselves.	Java Swing has only one peer in the form of OS's window object, which provides the drawing surface used to draw the Swing's widgets (label, button, entry fields, etc.) developed directly by Java Swing Package.
6	**Functionality and Implementati on**	Java AWT many features that are completely developed by the developer. It serves as a thin layer of development on the top of the OS.	Swing components provide the higher-level inbuilt functions for the developer that facilitates the coder to write less code.
7	**Memory**	Java AWT needs a higher amount of memory for the execution.	Java Swing needs less memory space as compared to Java AWT.
8	**Extensibility**	It offers limited customization options due to native component dependency.	It is highly extensible, allows modification and creation of custom components.
9	**Event Handling Model**	It uses an older event-handling model that is less efficient.	It uses a delegation event model for better event management.
10	**Thread Safety**	It is not thread-safe, requiring manual thread handling.	It supports built-in thread safety mechanisms.
11	**Data Binding**	There is no direct support for data binding.	It offers built-in data binding support for smoother UI updates.
12	**Tooltips**	It does not support tooltips for UI elements.	It provides tooltips to enhance user interaction.
13	**Double Buffering**	It requires manual double buffering to avoid flickering.	Automatically supports double buffering for smooth graphics rendering.
14	**Integration with MVC**	It does not follow the Model-View-Controller (MVC) pattern.	It follows the MVC architecture, improving separation of concerns.
15	**Resource Consumption**	It provides higher resource consumption due to reliance on native components.	It is more efficient in memory and CPU usage because of lightweight components.
16	**Accessibility Support**	It has limited accessibility features.	It offers enhanced accessibility features for assistive technology users.
17	**Internationali zation**	It provides basic support for internationalization.	It provides stronger internationalization features, enabling easier localization.
18	**Drag and Drop Support**	It provides limited drag-and-drop functionality.	It provides advanced drag-and-drop features for better UI interactions.
19	**HTML Content**	It cannot render HTML content in components.	It supports rendering simple HTML in components like JLabel.

	Rendering		
20	**Animation Support**	It provides basic animation capabilities with manual implementation.	It provides better animation support, improving user experience.
21	**Printing Support**	It provides limited printing options with basic functionality.	It provides advanced printing API for better document control.
22	**Development Community and Updates**	Less active community and rare updates.	Actively maintained with frequent updates and improvements.
23	**Learning Curve**	It is easier to learn and suitable for simple applications.	It is more complex due to extensive features, requiring a deeper understanding.

Q13. What are the limitations of AWT?

Q18. What is Swing in Java? How is it different from AWT.

AWT- Introduction:
Java AWT (Abstract Window Toolkit) is an API to develop GUI or window-based applications
in java.
Java AWT components are platform-dependent i.e. components are displayed according to the
view of operating system.
AWT is heavyweight i.e. its components are using the resources of OS.
The java.awt package provides classes for AWT API such as Text Field, Label, Tex tArea,
Radio Button, Check Box, Choice, List etc

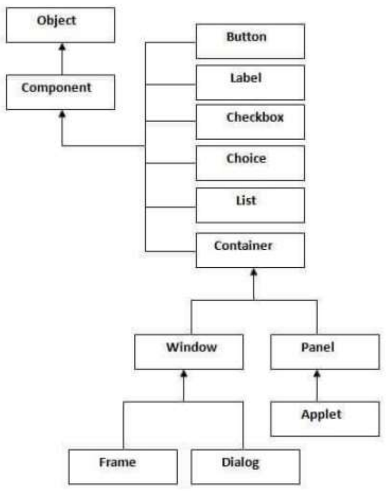
Limitations of AWT :

The AWT defines a basic set of controls, windows, and dialog boxes that support a reusable, but
limited graphical interface.
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 heavy weight.
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 and could not be changed.
Third, the use of heavyweight components caused some frustrating restrictions

==

Q.14 Give the AWT hierarchy.

Java AWT Hierarchy(Components,
Containers)
A component is an independent visual control, such as a push button.
A container holds a group of components. Examples of containers are Frame, Dialog, Applet, Panel, Window.



==

Q.15 .What are the various classes used in creating a swing menu?

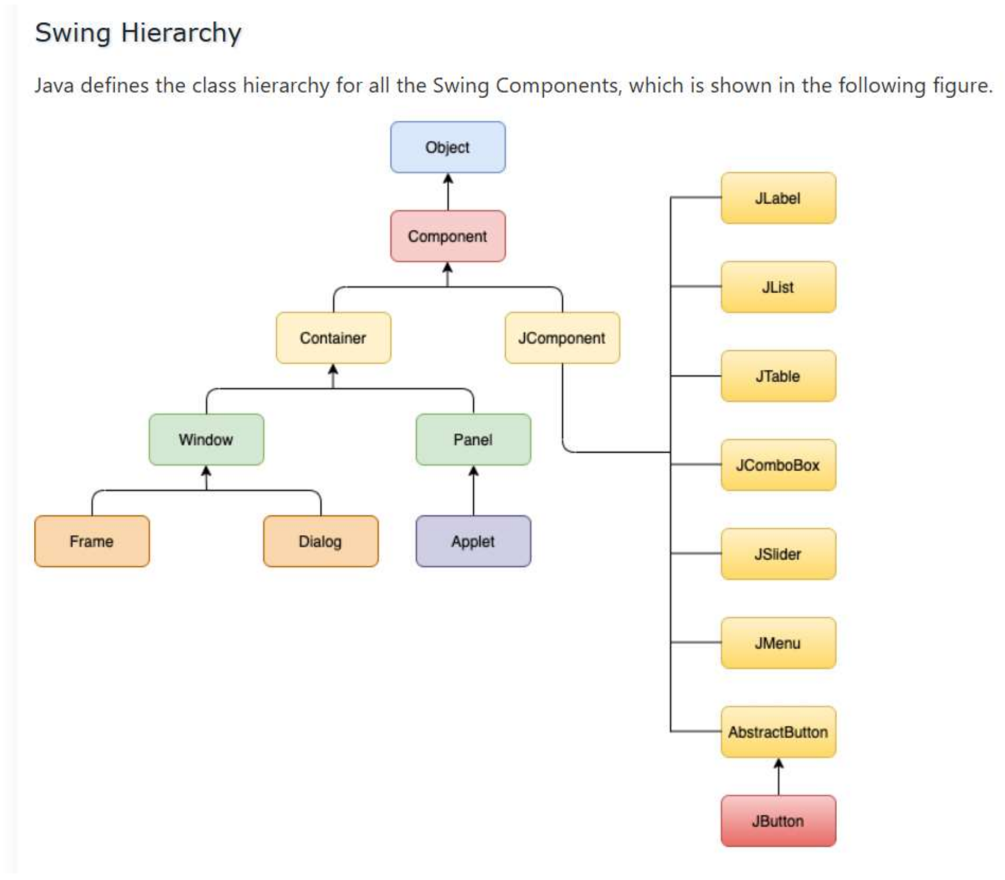
Java Swing

Swing is an enhanced GUI library introduced in JDK 1.2. It is a part of JFC (Java Foundation Classes). It is built on the top of AWT and written entirely in Java. It offers a richer set of components and a more flexible architecture for building user interfaces.

The **javax.swing** API provides all the component classes like

1. JButton,
2. JTextField,
3. JCheckbox,
4. JMenu, etc.

The components of Swing are platform-independent, i.e., swing does not depend on the operating system to show the components. Also, the Swing's components are lightweight.



Swing Hierarchy

Java defines the class hierarchy for all the Swing Components, which is shown in the following figure.

Screen clipping taken: 18-04-2025 17:30

===

Q16.

1. What are the differences between JToggle buttion and Radio buttion?

Java JRadioButton

17 Mar 2025 | 2 min read

The JRadioButton class is used to create a radio button. It is used to choose one option from multiple options. It is widely used in exam systems or quiz. It should be added in ButtonGroup to select one radio button only.

JRadioButton class declaration

Let's see the declaration for javax.swing.JRadioButton class

Java JToggleButton

JToggleButton is used to create toggle button, it is two-states button to switch on or off.

==

Q17. What is an adapter class? Explain with an example.

Java Adapter Classes

17 Mar 2025 | 4 min read

Java adapter classes *provide the default implementation of listener* *interfaces*. If you inherit the adapter class, you will not be forced to provide the implementation of all the methods of listener interfaces. So it *saves code*.

Pros of using Adapter classes:

- It assists the unrelated classes to work combinedly.
- It provides ways to use classes in different ways.

- It increases the transparency of classes.
- It provides a way to include related patterns in the class.
- It provides a pluggable kit for developing an application.
- It increases the reusability of the class.

The adapter classes are found in **java.awt.event, java.awt.dnd** and **javax.swing.event** packages. The Adapter classes with their corresponding listener interfaces are given below.

Java WindowAdapter Example

In the following example, we are implementing the WindowAdapter class of AWT and one its methods windowClosing() to close the frame window.

AdapterExample.java

```java
// importing the necessary libraries
import java.awt.*;
import java.awt.event.*;

public class AdapterExample {
// object of Frame
    Frame f;
// class constructor
    AdapterExample() {
// creating a frame with the title
        f = new Frame ("Window Adapter");
// adding the WindowListener to the frame
// overriding the windowClosing() method
        f.addWindowListener (new WindowAdapter() {
            public void windowClosing (WindowEvent e) {
                f.dispose();
            }
        });
         // setting the size, layout and
        f.setSize (400, 400);
        f.setLayout (null);
        f.setVisible (true);
    }

// main method
public static void main(String[] args) {
    new AdapterExample();
}
}
```

==

Q11. Write a short note on the following (i) JFrame (ii) JTabbedPane

Java JFrame

The javax.swing.JFrame class is a type of container which inherits the java.awt.Frame class. JFrame works like the main window where components like labels, buttons, textfields are added to create a GUI.
Unlike Frame, JFrame has the option to hide or close the window with the help of setDefaultCloseOperation(int) method.

JFrame is a flexible option for creating sophisticated graphical user interfaces for interactive Java applications.

Advantages of JFrame in Java Applications

1. Developing Desktop Applications
2. Graphical user interfaces
3. Application windows
4. Boxes of dialogue and pop-up windows
5. Data Visualization

Java JTabbedPane

The JTabbedPane class is used to switch between a group of components by clicking on a tab with a given title or icon. It inherits JComponent class.

JTabbedPane class declaration

Let's see the declaration for javax.swing.JTabbedPane class.

==