UNIT - IV

Unit - 4 Syllabus:

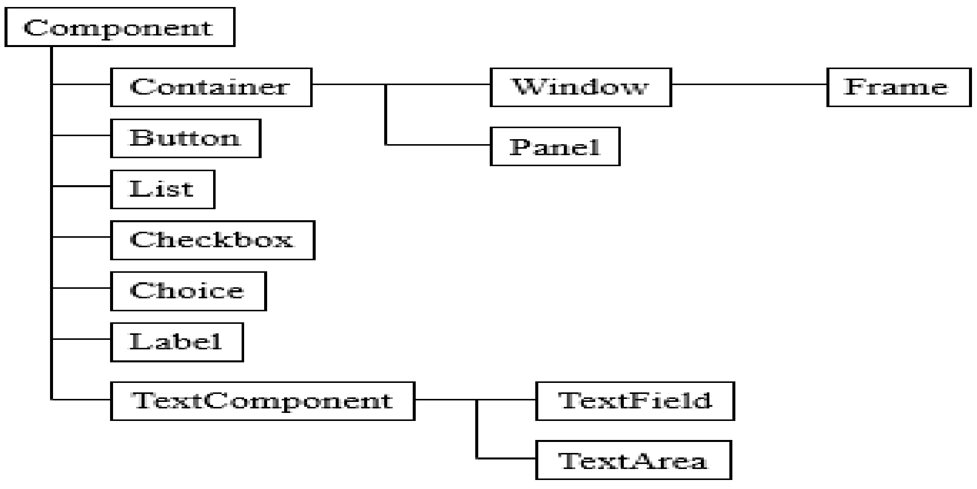
AWT: AWT Components, File Dialog boxes, Layout Managers, Event handling model of AWT, Adapter classes, Menu, Menu bar.
GUI with Swing– Swings introduction, JApplet, JFrame and JComponent, Icons and Labels, text fields, buttons – The JButton class, Check boxes, Radio buttons. Combo boxes, Tabbed Panes, Scroll Panes, Trees, and Tables.
Generics: Basics of Generic Methods, Generic Classes.
Collections: Collection Interfaces, Collection Classes, Accessing a Collection via an Iterator.

AWT

AWT Components:

- 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 TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

Java AWT class Hierarchy:



Frames:

- A frame is a container object in which GUI components(Textboxes,buttons etc.) can be placed in it.
- Frame is a class of the java.awt package.
- A Frame is the most important component of the Abstract Window Toolkit (AWT) that is treated as a complete window for specific software.

Constructor of Frame class

- **Frame():** It constructs a new Frame with no title that was initially invisible.

  public Frame()
- **Frame(String title):** It constructs a Frame with some title as specified by the programmer.

  public Frame(String title)

Methods of the Frame class

- **setTitle(String value):** This method provides a title to the Frame if not specified at the time of object declaration of the Frame.

  Frame object.setTitle("String value");
- **String getTitle():** This method returns the title of the Frame.

  String variable=Frame object.getTitle();
- **setSize(int row,int col):** This method sets the size of the Frame, row and column wise.

  void setSize(int row,int col);
- **setBounds(int x, int y, int w, int h):** Used to set the size and location of the component.

  void setBounds(int x, int y, int w, int h);
- **setLocation(int x, int y):** Used to set the size and location of the Window.

  void setLocation(int x, int y);

- **setVisible(boolean mode):** It makes the Frame visible to us. If the parameter ed is "true" then it will be visible otherwise if it is "False" then the Frame will not be visible.

  void setVisible(boolean mode);
- **setBackground(Color.color name):** It sets some specified color on the background of the Frame.

  Frame object.setBackground(Color.color name);
- **setTitle**: Sets the title for this frame to the specified title.

  public void setTitle(String title);
- **remove(Component c):** Deletes a component on this component.

  public void remove(Component c);
- **add(Component c):** Inserts a component on this component.

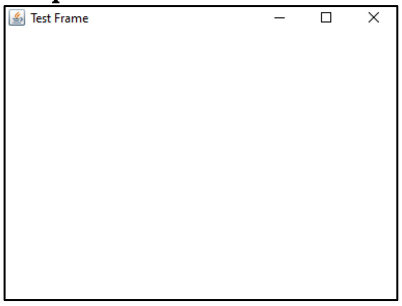
  public void add(Component c);
- **setLayout(Layout l):** Sets a layout to the container. setBounds doesn’t work if setLayout is not null.
  public void add(Component c);

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
KeyEvent	KeyListener
ItemEvent	ItemListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

Example:

```
import java.awt.*;
class MyFrame extends Frame{
	MyFrame(){
		super("Test Frame");
		setBounds(100,100,400,300);
		setVisible(true);
	}
	public static void main(String[] args){
		new MyFrame();
	}
}
```

Output:



We can't get the window closed by pressing X at the top.
To perform its specific operation we have WindowListener interface which contain some methods that takes WindowEvent as argument.

Java WindowListener Interface

The Java WindowListener is notified whenever you change the state of window. It is notified against WindowEvent. The WindowListener interface is found in java.awt.event package.

Sr. no.	Method signature	Description
1.	public abstract void windowActivated (WindowEvent e);	It is called when the Window is set to be an active Window.
2.	public abstract void windowClosed (WindowEvent e);	It is called when a window has been closed as the result of calling dispose on the window.
3.	public abstract void windowClosing (WindowEvent e);	It is called when the user attempts to close the window from the system menu of the window.
4.	public abstract void windowDeactivated (WindowEvent e);	It is called when a Window is not an active Window anymore.
5.	public abstract void windowDeiconified (WindowEvent e);	It is called when a window is changed from a minimized to a normal state.
6.	public abstract void windowIconified (WindowEvent e);	It is called when a window is changed from a normal to a minimized state.
7.	public abstract void windowOpened (WindowEvent e);	It is called when window is made visible for the first time.

*****If you want to implement WindowListener, we must implement each method in it. Else you will get error.**

Example:

```
import java.awt.*;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;

public class MyFrame extends Frame implements WindowListener {

	MyFrame() {
		addWindowListener(this);
		setSize (400, 400);
		setLayout (null);
		setVisible (true);
	}
public static void main(String[] args) {
	new MyFrame();
}
public void windowActivated (WindowEvent w) {}
public void windowClosed (WindowEvent w) {}
public void windowClosing (WindowEvent w) {
	dispose();	//Or we can use System.exit(0);
}
public void windowDeactivated (WindowEvent w) {}
public void windowDeiconified (WindowEvent w) {}
public void windowIconified(WindowEvent arg0) {}
public void windowOpened(WindowEvent w) {}
}
```

But implementing all these methods leads to inefficiency. So we use adapter classes to implement only the methods we need. Here is an Example:

```
import java.awt.*;
import java.awt.event.*;

class MyFrame extends Frame {
	MyFrame(){
		setSize(400, 400);
		setLayout(null);
		setVisible(true);
		addWindowListener( new WindowAdapter() {
		public void windowClosing(WindowEvent e) {
			System.exit(0);
	 }
}) ;
}
	 public static void main(String[] args){
       new MyFrame();
	}
}
```

Components of AWT

1. Label:

- The object of the Label class is a component for placing text in a container. It is used to display a single line of read only text. The text can be changed by a programmer but a user cannot edit it directly.
- It is called a passive control as it does not create any event when it is accessed. To create a label, we need to create the object of Label class.

Label class Constructors

1.	Label()	It constructs an empty label.
2.	Label(String text)	It constructs a label with the given string (left justified by default).

Label Class Methods

1.	void setText(String text)	It sets the texts for label with the specified text.
2.	String getText()	It gets the text of the label

2. TextField:

The object of a TextField class is a text component that allows a user to enter a single line text and edit it. It inherits TextComponent class, which further inherits Component class.
When we enter a key in the text field (like key pressed, key released or key typed), the event is sent to TextField. Then the KeyEvent is passed to the registered KeyListener. It can also be done using ActionEvent; if the ActionEvent is enabled on the text field, then the ActionEvent may be fired by pressing return key. The event is handled by the ActionListener interface.

TextField Class constructors

1.	TextField()	It constructs a new text field component.
2.	TextField(String text)	It constructs a new text field initialized with the given string text to be displayed.

TextField Methods

1.	void setText(String text)	It sets the texts for textfield with the specified text.
2.	String getText()	It gets the text of the label

3. Button:

A button is basically a control component with a label that generates an event when pushed. The Button class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed.

When we press a button and release it, AWT sends an instance of ActionEvent to that button by calling processEvent on the button. The processEvent method of the button receives the all the events, then it passes an action event by calling its own method processActionEvent. This method passes the action event on to action listeners that are interested in the action events generated by the button.

Button Class Constructors:

1.	Button( )	It constructs a new button with an empty string i.e. it has no label.
2.	Button (String text)	It constructs a new button with given string as its label.

Button Class Methods:

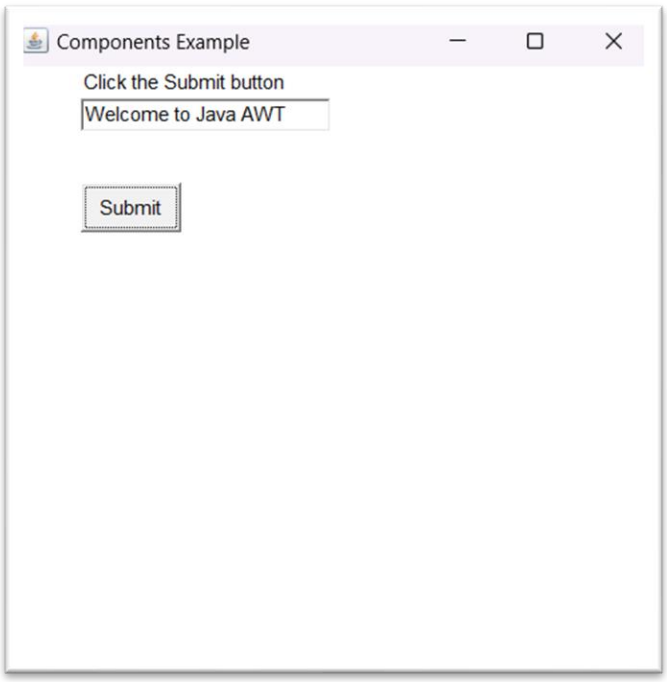
1.	void setText (String text)	It sets the string message on the button
2.	String getText()	It fetches the String message on the button.
3.	void setLabel (String label)	It sets the label of button with the specified string.
4.	String getLabel()	It fetches the label of the button.
5.	void addActionListener(ActionListener l)	It adds the specified action listener to get the action events from the button.

Example of above 3 AWT components:

```
import java.awt.*;
import java.awt.event.*;
public class Components{
public static void main(String[] args){
	Frame f = new Frame("Components Example");
	Label l = new Label("Click the Submit button");
	l.setBounds(50,30,150,20);
```

```
        final TextField tf=new TextField();
        tf.setBounds(50,50, 150,20);
        Button b=new Button("Submit");
        b.setBounds(50,100,60,30);
        b.addActionListener(new ActionListener() {
        public void actionPerformed (ActionEvent e) {
                tf.setText("Welcome to Java AWT");
            }
        });
        f.add(l);
        f.add(b);
        f.add(tf);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
}
}
```

Output:


Components Example

Click the Submit button

Welcome to Java AWT

4. Checkbox:

The Checkbox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a Checkbox changes its state from "on" to "off" or from "off" to "on".

Checkbox Class Constructors

1.	Checkbox()	It constructs a checkbox with no string as the label.
2.	Checkbox(String label)	It constructs a checkbox with the given label.

3.	Checkbox(String label, boolean state)	It constructs a checkbox with the given label and sets the given state.
4.	Checkbox(String label, CheckboxGroup group, boolean state)	It constructs a checkbox with the given label, in the given checkbox group and set to the specified state.

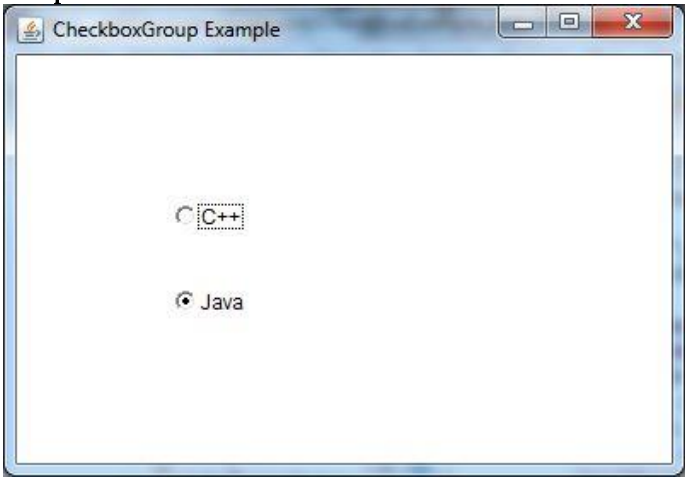
5. CheckboxGroup:

The object of CheckboxGroup class is used to group together a set of Checkbox. At a time only one check box button is allowed to be in "on" state and remaining check box button in "off" state. It inherits the object class.

Example:

```
import java.awt.*;
public class CheckboxGroupExample
{
        CheckboxGroupExample(){
        Frame f= new Frame("CheckboxGroup Example");
          CheckboxGroup cbg = new CheckboxGroup();
          Checkbox checkBox1 = new Checkbox("C++", cbg, false);
          checkBox1.setBounds(100,100, 50,50);
          Checkbox checkBox2 = new Checkbox("Java", cbg, true);
          checkBox2.setBounds(100,150, 50,50);
          f.add(checkBox1);
          f.add(checkBox2);
          f.setSize(400,400);
          f.setLayout(null);
          f.setVisible(true);
      }
public static void main(String args[])
{
     new CheckboxGroupExample();
}
}
```

Output:



6. Choice:

The object of Choice class is used to show popup menu of choices. Choice selected by user is shown on the top of a menu. It inherits Component class.

Choice Class constructor

1.	Choice()	It constructs a new choice menu.

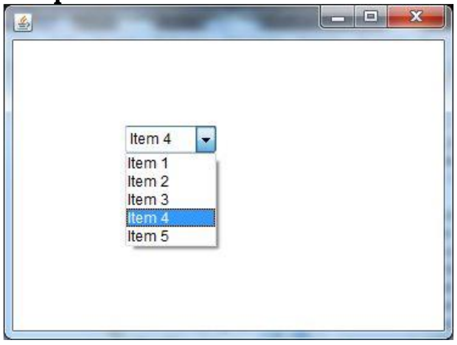
Choice Class Methods:

1.	void add(String item)	It adds an item to the choice menu.
2.	void addItemListener(ItemListener l)	It adds the item listener that receives item events from the choice menu.
3.	String getItem(int index)	It gets the item (string) at the given index position in the choice menu.
4.	int getItemCount()	It returns the number of items of the choice menu.
5.	int getSelectedIndex()	Returns the index of the currently selected item.
6.	String getSelectedItem()	Gets a representation of the current choice as a string.

Example:

```
import java.awt.*;
public class ChoiceExample1 {
        ChoiceExample1() {

        Frame f = new Frame();
        Choice c = new Choice();

        c.setBounds(100, 100, 75, 75);
        c.add("Item 1");
        c.add("Item 2");
        c.add("Item 3");
        c.add("Item 4");
        c.add("Item 5");

        f.add(c);
        f.setSize(400, 400);
        f.setLayout(null);
        f.setVisible(true);
    }

public static void main(String args[]){
    new ChoiceExample1();
}
}
```

Output:



7. List:

The object of List class represents a list of text items. With the help of the List class, user can choose either one item or multiple items. It inherits the Component class.

List Class Constructors

1.	List()	It constructs a new scrolling list.
2.	List(int row_num)	It constructs a new scrolling list initialized with the given number of rows visible.

List Class Methods

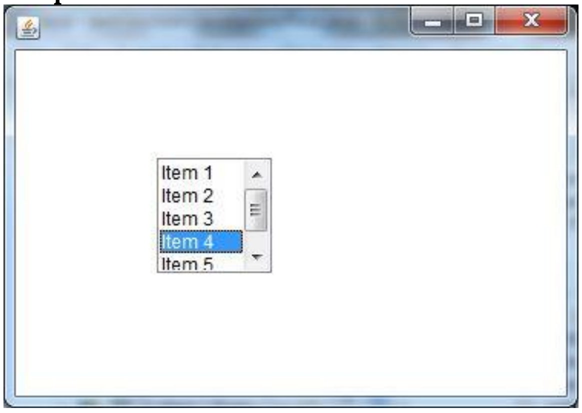
1.	void add(String item)	It adds the specified item into the end of scrolling list.
2.	void add(String item, int index)	It adds the specified item into list at the given index position.
3.	void addActionListener(ActionListener l)	It adds the specified action listener to receive action events from list.
4.	void addItemListener(ItemListener l)	It adds specified item listener to receive item events from list.

Example:

```
import java.awt,*;
public class ListExample1
{
        ListExample1() {
             Frame f = new Frame();
             List l1 = new List(5);
             l1.setBounds(100, 100, 75, 75);
             l1.add("Item 1");
             l1.add("Item 2");
             l1.add("Item 3");
             l1.add("Item 4");
             l1.add("Item 5");
             f.add(l1);
             f.setSize(400, 400);
```

```
            f.setLayout(null);
            f.setVisible(true);
        }
public static void main(String args[])      {
    new ListExample1();
}
}
```

Output:



Component

- Component is the superclass of most of the displayable classes defined within the AWT.   Note: it is abstract.
- Menu Component is another class which is similar to Component except it is the superclass for all GUI items which can be displayed within a drop-down menu.
- The Component class defines data and methods which are relevant to all Components

setBounds
setSize
setLocation
setFont
setEnabled
setVisible
setForeground   -- colour
setBackground   -- colour

Container

- Container is a subclass of Component. (ie. All containers are themselves, Components)
- Containers contain components
- For a component to be placed on the screen, it must be placed within a Container
- A container is responsible for laying out (that is, positioning) any components that it contains. It does this through the use of various layout managers
- The Container class defined all the data and methods necessary for managing groups of Components

add
getComponent
getMaximumSize
getMinimumSize
getPreferredSize
remove
removeAll

Windows and Frames

- The Window class defines a top-level Window with no Borders or Menu bar.
- Usually used for application splash screens
- A *top-level window* is not contained within any other object; it sits directly on the desktop.
- Generally, we won't create **Window** objects directly. Instead, we will use a subclass of **Window** called **Frame**
- Frame defines a top-level Window with Borders and a Menu Bar
    - Frames are more commonly used than Windows
- Once defined, a Frame is a Container which can contain Components

```
Frame aFrame = new Frame(□Hello World□);
aFrame.setSize(100,100);
aFrame.setLocation(10,10);
aFrame.setVisible(true);
```

Panels

- When writing a GUI application, the GUI portion can become quite complex.
- To manage the complexity, GUIs are broken down into groups of components. Each group generally provides a unit of functionality.
- A Panel is a rectangular Container whose sole purpose is to hold and manage components within a GUI.

```
Panel aPanel = new Panel();
aPanel.add(new Button("Ok"));
aPanel.add(new Button("Cancel"));
Frame aFrame = new Frame("Button Test");
aFrame.setSize(100,100);
aFrame.setLocation(10,10);
aFrame.add(aPanel);
```

Layout Managers

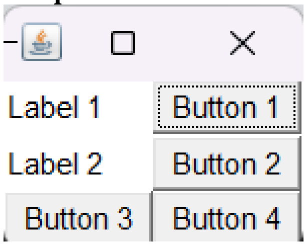
- All of the components that we have shown so far have been positioned by the default layout manager
- A layout manager automatically arranges our controls within a window by using some type of algorithm
- Each **Container** object has a layout manager associated with it. A layout manager is an instance of any class that implements the **LayoutManager** interface.
- The layout manager is set by the **setLayout( )** method.

- If no call to **setLayout( )** is made, then the default layout manager is used.
- Whenever a container is resized (or sized for the first time), the layout manager is used to position each of the components within it.

  void setLayout(new LayoutManager *layoutObj*);

Grid Layout Manager:
The GridLayout manager arranges components in a grid of rows and columns. It takes two arguments, the number of rows and the number of columns. Components are added to the grid one at a time, left to right, top to bottom. If there are more components than cells in the grid, the layout manager automatically adds additional rows and columns as needed.
Example:

```
import java.awt.*;
import java.awt.event.*;
class Components{
public static void main(String[] args){
Frame frame = new Frame("Grid Layout Example");
frame.setLayout(new GridLayout(3, 2));

Label label1 = new Label("Label 1");
Label label2 = new Label("Label 2");
Button button1 = new Button("Button 1");
Button button2 = new Button("Button 2");
Button button3 = new Button("Button 3");
Button button4 = new Button("Button 4");

frame.add(label1);
frame.add(button1);
frame.add(label2);
frame.add(button2);
frame.add(button3);
frame.add(button4);

frame.pack();
frame.setVisible(true);
   }
}
```

Output:


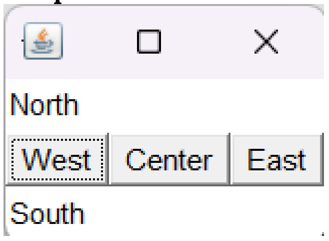
Border Layout Manager:
The BorderLayout manager divides the container into five areas: north, south, east, west, and center. Components can be added to any of these areas, and they

will automatically resize to fill the available space in that area. By default, components are added to the center area.

Example:

```
import java.awt.*;
import java.awt.event.*;
class Components{
public static void main(String[] args){
Frame frame = new Frame("Border Layout Example");
frame.setLayout(new BorderLayout());

Label label1 = new Label("North");
Label label2 = new Label("South");
Button button1 = new Button("West");
Button button2 = new Button("East");
Button button3 = new Button("Center");

frame.add(label1, BorderLayout.NORTH);
frame.add(label2, BorderLayout.SOUTH);
frame.add(button1, BorderLayout.WEST);
frame.add(button2, BorderLayout.EAST);
frame.add(button3, BorderLayout.CENTER);

frame.pack();
frame.setVisible(true);

   }
}
```

Output:

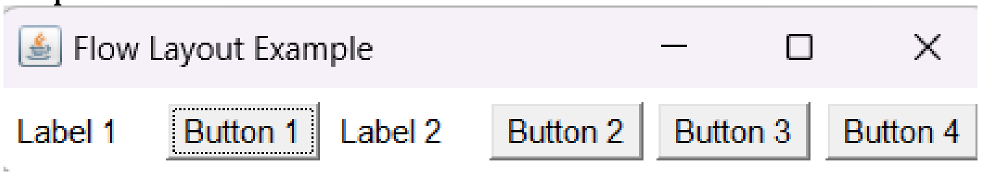


Flow Layout Manager:

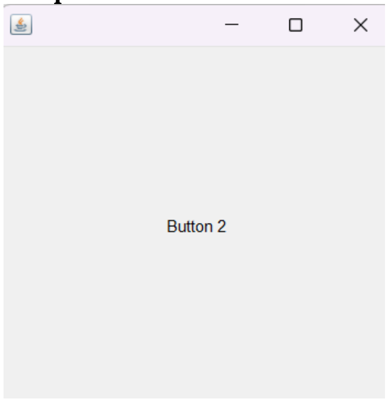
The FlowLayout manager arranges components in a left-to-right flow, wrapping to a new row if the container becomes too narrow. It takes three arguments: the alignment (left, center, or right), the horizontal gap between components, and the vertical gap between rows.

Example:

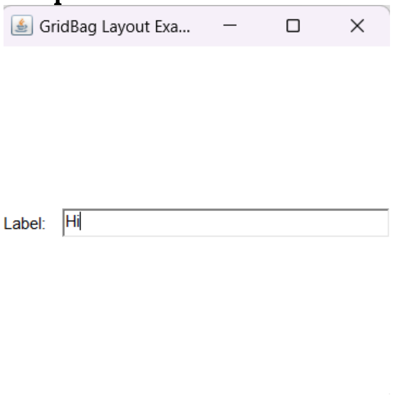
```
import java.awt.*;
import java.awt.event.*;
class Components{
public static void main(String[] args){
Frame frame = new Frame("Flow Layout Example");
frame.setLayout(new FlowLayout());

Label label1 = new Label("Label 1");
Label label2 = new Label("Label 2");
Button button1 = new Button("Button 1");
Button button2 = new Button("Button 2");
Button button3 = new Button("Button 3");
Button button4 = new Button("Button 4");

frame.add(label1);
frame.add(button1);
frame.add(label2);
frame.add(button2);
frame.add(button3);
frame.add(button4);

frame.pack();
frame.setVisible(true);

   }
}
```
Output:



Card Layout Manager:
The CardLayout manager allows multiple components to be stacked on top of each other, with only one component visible at a time. Components are added to the layout with a unique name, and can be switched between using the CardLayout.show() method.
Example:
```
import java.awt.*;
import java.awt.event.*;
public class Components extends Frame implements ActionListener {
    Panel cards;
    Button btn1, btn2, btn3;
    CardLayout cl;
    public Components() {
        cl = new CardLayout();
        cards = new Panel();
        cards.setLayout(cl);
        btn1 = new Button("Button 1");
        btn2 = new Button("Button 2");
        btn3 = new Button("Button 3");
        btn1.addActionListener(this);
        btn2.addActionListener(this);
        btn3.addActionListener(this);
        cards.add(btn1, "Button 1");
        cards.add(btn2, "Button 2");
        cards.add(btn3, "Button 3");
        add(cards, BorderLayout.CENTER);
        setSize(300,300);
        setVisible(true);
    }
    public void actionPerformed(ActionEvent e) {
        cl.next(cards);
    }
    public static void main(String args[]) {
        new Components();
    }
}
```
Output: If we click on a button we get moved to next button.



GridBag Layout Manager:
The GridBagLayout manager is a more flexible layout manager that allows you to specify how components should be arranged in a grid of cells. You can specify the row and column that a component should be placed in, as well as how many cells it should span horizontally and vertically. You can also specify the amount of space that should be left between components.
Example:
```
import java.awt.*;
import java.awt.event.*;
public class Components{
    public static void main(String args[]) {
     Frame frame = new Frame("GridBag Layout Example");
        Panel panel = new Panel(new GridBagLayout());
     GridBagConstraints c = new GridBagConstraints();
     c.gridx = 0;
     c.gridy = 0;
     c.anchor = GridBagConstraints.LINE_END;
     panel.add(new Label("Label:"), c);
     c.gridx = 1;
     c.gridy = 0;
     c.fill = GridBagConstraints.HORIZONTAL;
     c.weightx = 1.0;
     panel.add(new TextField(), c);
     frame.add(panel);
     frame.setSize(300,300);
     frame.setVisible(true);
    }
}
```
Output:



MenuItem and Menu

- The object of MenuItem class adds a simple labeled menu item on menu. The items used in a menu must belong to the MenuItem or any of its subclass.
- The object of Menu class is a pull down menu component which is displayed on the menu bar. It inherits the MenuItem class.
- Following is the declaration for java.awt.Menu class:

public class Menu extends MenuItem implements MenuContainer, Accessible

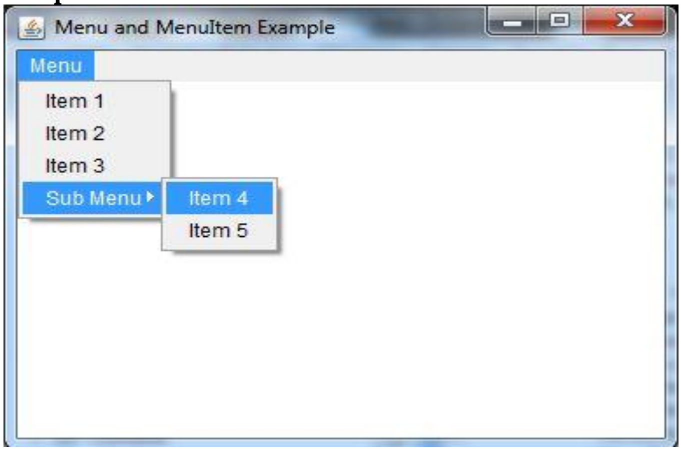
- **Constructors are:**
    - Menu()
    - Menu(String label)
    - Menu(String label, boolean tearOff)
- **Methods inherited**
    - java.awt.MenuItem
    - java.awt.MenuComponent
    - java.lang.Object

Example:

```
import java.awt.*;
class MenuExample
{
   MenuExample(){
     Frame f= new Frame("Menu and MenuItem Example");
     MenuBar mb=new MenuBar();
     Menu menu=new Menu("Menu");
     Menu submenu=new Menu("Sub Menu");
     MenuItem i1=new MenuItem("Item 1");
     MenuItem i2=new MenuItem("Item 2");
     MenuItem i3=new MenuItem("Item 3");
     MenuItem i4=new MenuItem("Item 4");
     MenuItem i5=new MenuItem("Item 5");
     menu.add(i1);
     menu.add(i2);
     menu.add(i3);
     submenu.add(i4);
     submenu.add(i5);
```

```
        menu.add(submenu);
        mb.add(menu);
        f.setMenuBar(mb);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
}
public static void main(String args[])
{
new MenuExample();
}
}
```

Output:



GUI with Swing

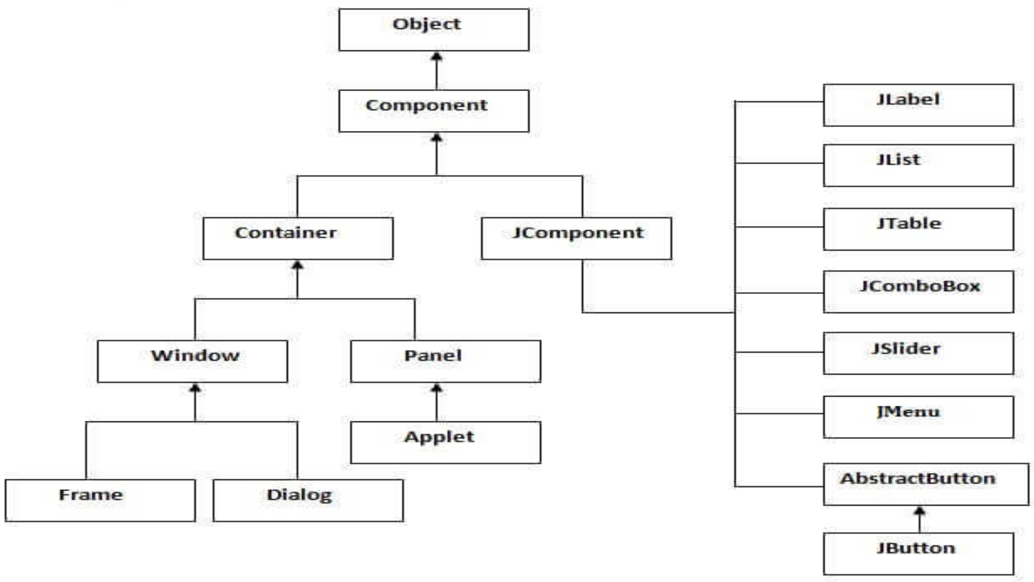
Introduction

- Swing is a framework that provides more powerful and flexible GUI components than does the AWT.
- **Java Swing** is *used to create window-based applications*.
- It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.
- Unlike AWT, Java Swing provides platform-independent and lightweight components.
- The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Difference between AWT and Swing

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.**

Hierarchy of Java Swing classes



Components

- A *component* is an independent visual control, such as a push button or slider.
- Swing components are derived from the **JComponent** class.
- **JComponent** inherits the AWT classes **Container** and **Component.**
- All of Swing's components are represented by classes defined within the package **javax.swing.**

Containers

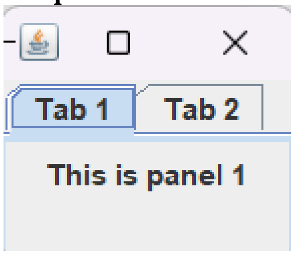
- A container holds a group of components.
- Thus, a container is a special type of component that is designed to hold other components.
- All Swing GUIs will have at least one container

- Swing defines two types of containers.
    1. Top-level containers
    2. Lightweight containers.

Top-level containers:
- The top-level containers are heavyweight.
- The first are top-level containers: **JFrame**, **JApplet**, **JWindow**, and **JDialog**.
- These containers do not inherit **Jcomponent** They do, however, inherit the AWT classes **Component** and **Container**.
- The most commonly used for applications is **Jframe,** for applets is **JApplet**.

Light-weight containers:
- Lightweight containers *do* inherit **JComponent.**
- An example of a lightweight container is **Jpanel.**
- Lightweight containers are often used to organize and manage groups of related components because a lightweight container can be contained within another container.

Swing Comppponents

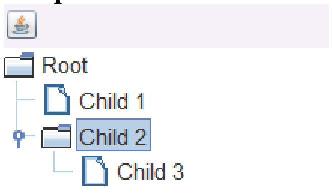
1. JTabbedPane:
JTabbedPane is a swing component that allows you to create a tabbed pane, which is a container that can hold multiple components and displays them one at a time. Each component is associated with a tab, and you can switch between the components by clicking on the tabs.
Example:
```
import javax.swing.*;
public class TabbedPaneExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        JTabbedPane tabbedPane = new JTabbedPane();
        JPanel panel1 = new JPanel();
        panel1.add(new JLabel("This is panel 1"));
        JPanel panel2 = new JPanel();
        panel2.add(new JLabel("This is panel 2"));
        tabbedPane.addTab("Tab 1", panel1);
        tabbedPane.addTab("Tab 2", panel2);
        frame.add(tabbedPane);
        frame.pack();
        frame.setVisible(true);
    }
}
```
Output:


This is panel 1

2. JTree:
JTree is a swing component that allows you to display hierarchical data in a tree structure. Each node in the tree can have zero or more child nodes. You can expand and collapse nodes to show or hide their children.
Example:

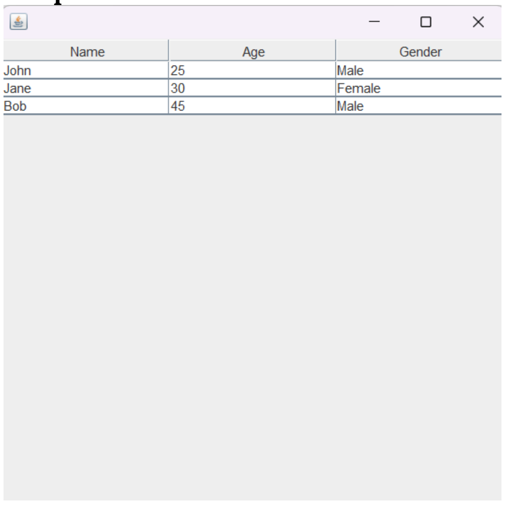
```
import javax.swing.*;
import javax.swing.tree.DefaultMutableTreeNode;
public class TreeExample{
	public static void main(String[] args){
		JFrame frame = new JFrame();
		DefaultMutableTreeNode root=new DefaultMutableTreeNode("Root");
		DefaultMutableTreeNode child1=new DefaultMutableTreeNode("Child
1");
		DefaultMutableTreeNode child2=new DefaultMutableTreeNode("Child
2");
		DefaultMutableTreeNode child3=new DefaultMutableTreeNode("Child
3");
		root.add(child1);
		root.add(child2);
		child2.add(child3);
		JTree tree = new JTree(root);
		frame.add(tree);
		frame.pack();
		frame.setVisible(true);
	}
}
```

Output:



3. JTable:
J Table is a swing component that allows you to display tabular data. It allows you to create a table with columns and rows, and you can customize the cells to display different types of data, such as text, numbers, or images.
Example:

```
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
public class TableExample {
	public static void main(String[] args) {
		JFrame frame = new JFrame();
		String[] columns = {"Name", "Age", "Gender"};
		Object[][] data = {
			{"John", 25, "Male"},
			{"Jane", 30, "Female"},
                {"Bob", 45, "Male"}
            };
            JTable table = new JTable(new DefaultTableModel(data, columns));
            frame.add(new JScrollPane(table));
            frame.pack();
            frame.setVisible(true);
        }
}
```
Output:



Name	Age	Gender
John	25	Male
Jane	30	Female
Bob	45	Male

4. JButton: JButton is a swing component that represents a clickable button. It is used to perform an action when it is clicked. For example, a "Submit" button in a form, a "Cancel" button in a dialog box, or a "Next" button in a wizard.

5. JLabel: JLabel is a swing component used to display text or an image. It is often used to provide a description or label for another component. For example, a label for a text field, a title for a window, or a caption for an image.

6. JList: JList is a swing component used to display a list of items. It allows the user to select one or more items from the list. For example, a list of contacts, a list of files, or a list of options.

7. JCheckBox: JCheckBox is a swing component used to represent a checkbox. It allows the user to select one or more options from a set of choices. For example, a checkbox to enable or disable a feature, or a set of checkboxes to select multiple items.

8. JRadioButton: JRadioButton is a swing component used to represent a radio button. It allows the user to select one option from a set of mutually exclusive choices. For example, a set of radio buttons to select a gender or a set of options.

9. JTextField: JTextField is a swing component used to allow the user to enter and edit text. It is often used to collect input from the user. For example, a text field to enter a username or password, or a search box to enter a query.

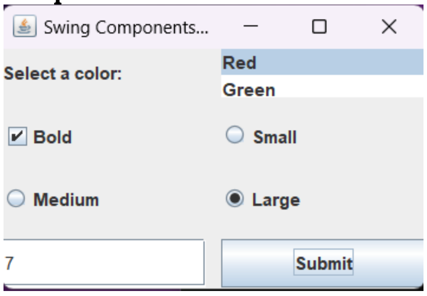
10. JPanel: JPanel is a swing component that is used as a container to hold other components. It is often used to group related components together and to provide a layout for them. JPanel can also be used as a background or a separator for other components.

11. JScrollPane: JScrollPane is a swing component used to provide scrolling capability to other components. It is commonly used to provide scrolling capability to large text areas, tables, and other components. JScrollPane wraps the other component and provides horizontal and vertical scroll bars as needed.

Example containing above swing components:

```
import javax.swing.*;
import java.awt.*;
public class SwingComponents extends JFrame {
public SwingComponents() {
        // Setting the title of the frame
        setTitle("Swing Components Example");
        // Creating a panel for holding the components
        JPanel panel = new JPanel();
        // Creating a label
        JLabel label = new JLabel("Select a color:");
        // Creating a list
        String[] colors = {"Red", "Green", "Blue", "Yellow"};
        JList<String> list = new JList<>(colors);
        // Creating a checkbox
        JCheckBox checkBox = new JCheckBox("Bold");
        // Creating radio buttons
        JRadioButton radioButton1 = new JRadioButton("Small");
        JRadioButton radioButton2 = new JRadioButton("Medium");
        JRadioButton radioButton3 = new JRadioButton("Large");
        // Creating a group for the radio buttons
        ButtonGroup group = new ButtonGroup();
        group.add(radioButton1);
        group.add(radioButton2);
        group.add(radioButton3);
        // Creating a text field
        JTextField textField = new JTextField(10);
        // Creating a button
        JButton button = new JButton("Submit");
        // Adding the components to the panel
        panel.add(label);
        panel.add(list);
        panel.add(checkBox);
        panel.add(radioButton1);
        panel.add(radioButton2);
        panel.add(radioButton3);
        panel.add(textField);
        panel.add(button);
          //Creating a Scroll pane
          JScrollPane jsp = new JScrollPane(panel);
          add(jsp, BorderLayout.CENTER);
        // Adding the panel to the frame
        add(panel);
        // Setting the size of the frame
        setSize(300, 200);
        // Setting the layout of the panel
        panel.setLayout(new GridLayout(4, 2, 10, 10));
        // Setting the default close operation
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // Making the frame visible
        setVisible(true);
    }
    public static void main(String[] args) {
        new SwingComponents();
    }
```

Output:



12. JApplet:

JApplet is a Swing component that is used to create applets in Java. Applets are small applications that run inside a web browser. The JApplet class extends the Applet class and provides additional functionality specific to Swing.

import javax.swing.*;

Example:

```
public class MyApplet extends JApplet {
    public void init() {
        JLabel label = new JLabel("Hello, World!");
        getContentPane().add(label);
    }
}
```

13. JScrollBar:

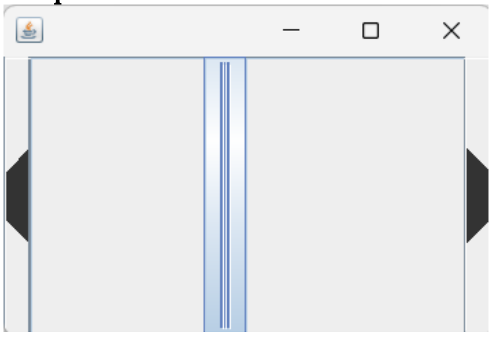
JScrollBar is a Swing component that provides a horizontal or vertical scrollbar for scrolling through a range of values. It is typically used in conjunction with another component, such as a JScrollPane, to provide scrolling functionality.

Example:

```
import javax.swing.*;

public class MyScrollBar extends JFrame {
  public MyScrollBar() {
    // Create a horizontal scrollbar
    JScrollBar scrollbar = new JScrollBar(JScrollBar.HORIZONTAL);
    add(scrollbar);
```

```
        setSize(300, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }

    public static void main(String[] args) {
        new MyScrollBar();
    }
}
```

Output:



Generics

Generics in Java allow you to define classes, interfaces, and methods that can work with different data types. This means that you can create reusable code that can be used with a variety of data types without having to rewrite the code for each type.
Generics were introduced in Java 5 and are represented using angle brackets (<>). The type parameter is defined inside the angle brackets and can be any valid Java identifier

Example:

```
class Gen<T> {
T ob;
Gen(T o) {
ob = o;
}
T getob() {
return ob;
}
void showType() {
System.out.println("Type of T is " +ob.getClass().getName());
}
}
```

Now we will see how to create generic classes and generic methods.
In Java, generic classes and methods are used to create reusable code that can work with different data types.

Syntax:

```
class ClassName<T>{
        ……..
}
```

Example:

```
class MyClass<T> {
        private T value;
        public MyClass(T value) {
                this.value = value;
        }
        public T getValue() {
                return value;
        }
        public void setValue(T value) {
                this.value = value;
        }
        public void print(){
                System.out.print("The Data Type is:
"+value.getClass().getName()+"\n");
        }
}
class Main{
        public static void main(String[] args){
                MyClass<String> myString = new MyClass<>("Hello World");
                myString.print();
                MyClass<Integer> myInt = new MyClass<>(42);
                myInt.print();
        }
}
```

Output:

```
The Data Type is: java.lang.String
The Data Type is: java.lang.Integer
```

Similarly, we can create generic methods that can work with different data types. The syntax for creating a generic method is as follows:

Syntax:

```
public static <T> T MethodName(parameter 1,    parameter 2,…..) {
        …………..
}
```

Example:

```
public class GenericMethodTest {
      public static < E > void printArray( E[] inputArray ) {
            for(E element : inputArray) {
                  System.out.printf("%s ", element);
            }
            System.out.println();
      }
      public static void main(String args[]) {
            Integer[] intArray = { 1, 2, 3, 4, 5 };
            Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4 };
```

```
        Character[] charArray = { 'H', 'E', 'L', 'L', 'O' };
        System.out.println("Array integerArray contains:");
        printArray(intArray);
        System.out.println("\nArray doubleArray contains:");
        printArray(doubleArray);
        System.out.println("\nArray characterArray contains:");
        printArray(charArray);
    }
}
```

Output:

```
Array integerArray contains:1 2 3 4 5
Array doubleArray contains:1.1 2.2 3.3 4.4
Array characterArray contains:H E L L O
```

Collections

Java Collections is a group of objects. It includes a set of interfaces and classes that provide various data structures, such as lists, sets, maps, and queues. These data structures are used to store and organize collections of elements, such as numbers, strings, or objects. Here are some examples of Java Collections:

1. List: List is an interface that extends Collection and represents an ordered sequence of elements. It allows duplicate elements and provides methods to add, remove, and access elements by their index. ArrayList and LinkedList are the two most commonly used implementations of the List interface.

Example:

```
List<String> names = new ArrayList<>();
names.add("John");
names.add("Jane");
names.add("Bob");
System.out.println(names); // Output: [John, Jane, Bob]
```

2. Set: Set is an interface that extends Collection and represents a collection of unique elements. It does not allow duplicate elements and provides methods to add, remove, and check if an element is present. HashSet, TreeSet, and LinkedHashSet are the three most commonly used implementations of the Set interface.

Example:

```
Set<String> colors = new HashSet<>();
colors.add("Red");
colors.add("Green");
colors.add("Blue");
System.out.println(colors); // Output: [Blue, Green, Red]
```

3. Map: Map is an interface that represents a collection of key-value pairs. It does not allow duplicate keys and provides methods to add, remove, and access elements by their key. HashMap, TreeMap, and LinkedHashMap are the three most commonly used implementations of the Map interface.

Example:

```
Map<String, String> countryCodes = new HashMap<>();
countryCodes.put("USA", "001");
countryCodes.put("UK", "044");
countryCodes.put("India", "091");
System.out.println(countryCodes.get("USA")); // Output: 001
```

4. Queue: Queue is an interface that represents a collection of elements that can be added and removed in a specific order. It provides methods to add, remove, and access elements based on their order. LinkedList and PriorityQueue are the two most commonly used implementations of the Queue interface.

Example:

```
Queue<String> queue = new LinkedList<>();
queue.add("John");
queue.add("Jane");
queue.add("Bob");
System.out.println(queue.poll()); // Output: John
```

For more info. refer Collections PPT

****