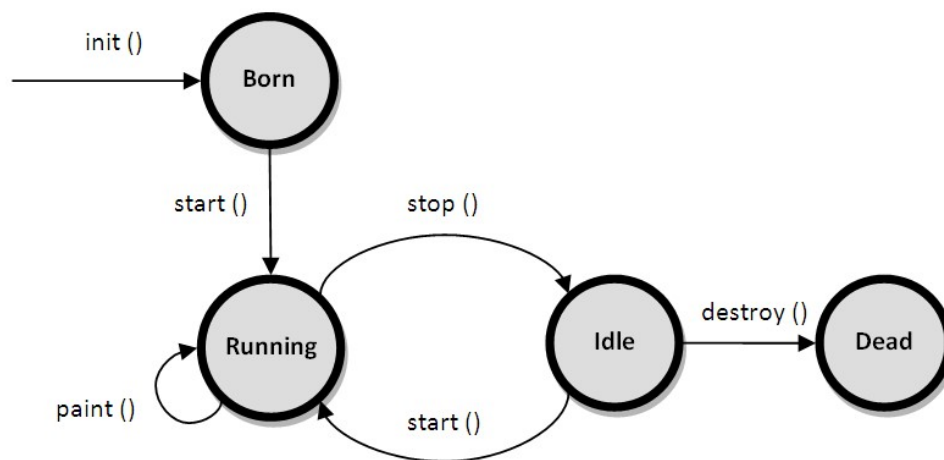


Unit-IV

Applet:

- Applet is a java program that is embedded in the webpage to generate the dynamic content.
- It runs inside the browser and works at client side.
- Applets are embedded in an HTML page using the APPLET or OBJECT tag and hosted on a web server.
- Applets are used to make the website more dynamic and entertaining.
- Applets have limited access to resources so that it can run complex computations without introducing the risk of viruses or breaching data integrity.
- Applet programs are primarily used in Internet programming.
- These programs are either developed in local systems or in remote systems and are executed by either a java compatible “**Web browser**” or “**appletviewer**”.
- There two types of applets-
 1. **Local Applet:** An applet developed locally and stored in a local system is known as a **Local Applet**. When a Web page is trying to find a local applet, it does not need to use the Internet and therefore the local system does not require the Internet connection. It simply searches the directories in the local system and locates and loads the specified applet.
 2. **Remote Applet:** An applet developed by someone else and stored on a remote computer is known as **Remote Applet**. If our system is connected to the Internet, we download the remote applet onto our system via the Internet and run it. In order to locate the remote applet, we must know the applet’s address on the Web. This address is known as **Uniform Resource Locator (URL)**.

Applet Life Cycle:



- Every java applet inherits a set of default behaviors from the Applet class defined in **java.applet** package.
- When an applet is loaded, it undergoes a series of changes in its states.
- The important states of the Applet Life Cycle are:
 1. **Born or Initialization State**

2. **Running State**

3. **Idle State**

4. **Dead or Destroyed State**

1. **Born or Initialization State**

- Applet enters the initialization state when it is first loaded.
- This is achieved by calling the **init()** method of **Applet** class. The applet is **born**.
- The initialization **occurs only once** in the applet's life cycle.
- Generally all the initialization variables are to be placed in the **init()** method.

• **Syntax:** public void init()
 {

 (Action)
 }

2. **Running State**

- An applet enters into running state when the system calls the **start()** method of **Applet** class.
- This occurs automatically after the applet is initialized.
- Starting can also **occurs** if the applet is already in “**Stopped State**”.
- The **start()** method may be called more than once.

• **Syntax:** public void start()
 {

 (Action)
 }

3. **Idle or Stopped State**

- An applet becomes idle when it is stopped from running.
- Stopping occurs automatically when we leave the page containing the currently running applet.
- This can also done by calling the **stop()** method explicitly.

• **Syntax:** public void stop()
 {

 (Action)
 }

4. **Dead State**

- An applet is said to be dead when it is removed from memory.
- This occurs automatically by invoking the **destroy()** method when we quit the browser.
- Destroying stage **occurs only once** in the applet life cycle.

• **Syntax:** public void destroy()
 {

 (Action)
 }

5. **Display State**

- Display state is useful to display the information on the output screen.
- This happens immediately after the applet enters into the running state.
- The **paint()** is called to accomplish this task. Almost every applet will have a **paint()** method.

• **Syntax:** public void paint(Graphics g)

Subject & Code: Object Oriented Programming & 20IT303

```
{  
    . . . . (Display Statements)  
}
```

Example: Demonstrate the applet life cycle process

Source code: **AppletDemo.java**

```
import java.awt.*;  
import java.applet.Applet;  
public class AppletDemo extends Applet  
{  
    String msg=" ";  
    public void init() {  
        msg=msg+"init( ) called";  
    }  
    public void start( ) {  
        msg=msg+"start( ) called";  
    }  
    public void paint(Graphics g) {  
        msg=msg+"paint( ) called";  
        g.drawString(msg,50,100);  
    }  
}  
  
//applet code  
//<applet code="AppletDemo.class" width="400" height="300">  
//</Applet>
```

The steps involved in developing and testing an applet are:

1. Building an applet code (.java file)
2. Creating an executable applet (.class file)
3. Create HTML page with the <APPLET> tag
4. Testing the Applet code with appletviewer

Output:



Displaying Graphics in Applet

java.awt.Graphics class provides many methods for graphics programming.

Commonly used methods of Graphics class:

1. **public abstract void drawString(String str, int x, int y):** is used to draw the specified string.
2. **public void drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.
3. **public abstract void fillRect(int x, int y, int width, int height):** is used to fill rectangle with the default color and specified width and height.
4. **public abstract void drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.
5. **public abstract void fillOval(int x, int y, int width, int height):** is used to fill oval with the default color and specified width and height.
6. **public abstract void drawLine(int x1, int y1, int x2, int y2):** is used to draw line between the points(x1, y1) and (x2, y2).
7. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.
8. **public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used draw a circular or elliptical arc.
9. **public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used to fill a circular or elliptical arc.
10. **public abstract void setColor(Color c):** is used to set the graphics current color to the specified color.

11. **public abstract void setFont(Font font):** is used to set the graphics current font to the specified font.

Example of Graphics in applet:

```
import java.applet.Applet;
import java.awt.*;

public class GraphicsDemo extends Applet{

    public void paint(Graphics g){
        g.setColor(Color.red);
        g.drawString("Welcome",50, 50);
        g.drawLine(20,30,20,300);
        g.drawRect(70,100,30,30);
        g.fillRect(170,100,30,30);
        g.drawOval(70,200,30,30);

        g.setColor(Color.pink);
        g.fillOval(170,200,30,30);
        g.drawArc(90,150,30,30,30,270);
        g.fillArc(270,150,30,30,0,180);

    }
}
```

myapplet.html

```
<html>
<body>
<applet code="GraphicsDemo.class" width="300" height="300">
</applet>
</body>
</html>
```

Displaying Image in Applet

Applet is mostly used in games and animation. For this purpose image is required to be displayed. The java.awt.Graphics class provide a method drawImage() to display the image.

Syntax of drawImage() method:

public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer): is used draw the specified image.

Subject & Code: Object Oriented Programming & 20IT303

How to get the object of Image:

The java.applet.Applet class provides getImage() method that returns the object of Image. Syntax:

```
public Image getImage(URL u, String image){}
```

Other required methods of Applet class to display image:

1. **public URL getDocumentBase():** is used to return the URL of the document in which applet is embedded.
2. **public URL getCodeBase():** is used to return the base URL.

Example of displaying image in applet:

```
import java.awt.*;  
import java.applet.*;  
public class DisplayImage extends Applet {
```

Image picture;

```
    public void init() {  
        picture = getImage(getDocumentBase(),"sonoo.jpg");  
    }  
    public void paint(Graphics g) {  
        g.drawImage(picture, 30,30, this);  
    }  
}
```

myapplet.html

```
<html>  
<body>  
<applet code="DisplayImage.class" width="300" height="300">  
</applet>  
</body>  
</html>
```

Parameter passing in Applet

Subject & Code: Object Oriented Programming & 20IT303

Parameters specify extra information that can be passed to an applet from the HTML page. Parameters are specified using the HTML's **param** tag.

Passing parameters to applets can be done using the *param* tag and retrieving the values of parameters using *getParameter* method.

Param Tag

The <param> tag is a sub tag of the <applet> tag. The <param> tag contains two attributes: name and value which are used to specify the name of the parameter and the value of the parameter respectively. For example, the param tags for passing name and age parameters looks as shown below:

Syntax:

```
<param name="name" value="Ramesh" />
```

```
<param name="age" value="25" />
```

Now, these two parameters can be accessed in the applet program using the `getParameter()` method of the Applet class.

getParameter() Method

The `getParameter()` method of the Applet class can be used to retrieve the parameters passed from the HTML page. The syntax of `getParameter()` method is as follows:

Syntax: `public String getParameter(String param-name)`

Example of using parameter in Applet:

```
import java.applet.Applet;
```

```
import java.awt.Graphics;
```

```
public class UseParam extends Applet{
```

```
public void paint(Graphics g){
```

```
String str=getParameter("msg");
```

```
g.drawString(str,50, 50);
```

```
}
```

```
}
```

myapplet.html

```
<html>
```

```
<body>
```

```
<applet code="UseParam.class" width="300" height="300">
```

```
<param name="msg" value="Welcome to applet">
```

Prepared By : Suresh Kumar Kallagunta, Asst.Professor, Dept.of IT

Subject & Code: Object Oriented Programming & 20IT303

</applet>

</body>

</html>

Subject & Code: Object Oriented Programming & 20IT303

AWT: Java AWT (Abstract Window Toolkit) is an API to develop Graphical User Interface (GUI) or windows-based applications in Java.

Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavy weight i.e. its components are using the resources of underlying operating system (OS).

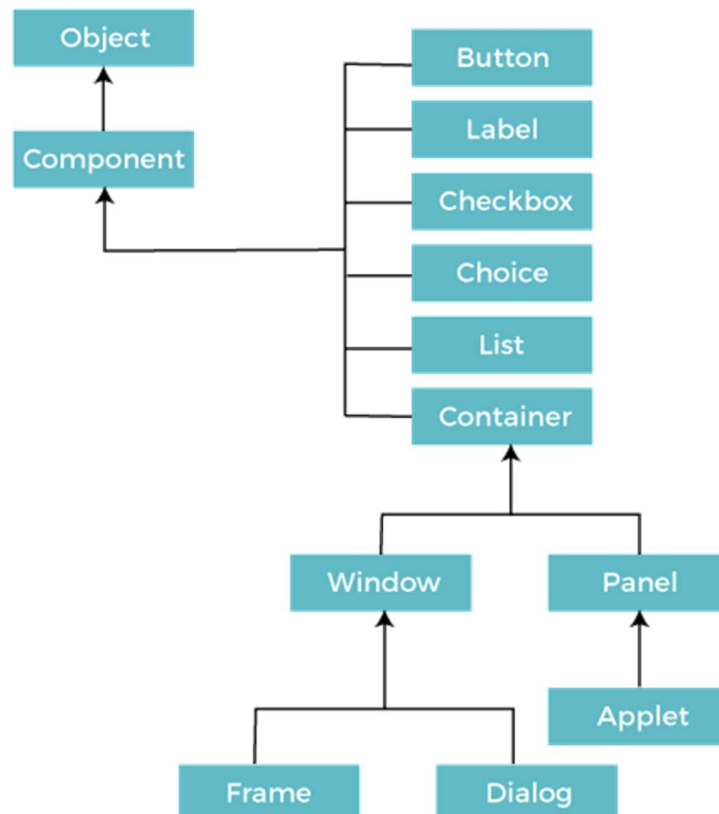
The java.awt package provides classes for AWT API such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

Why AWT is platform dependent?

Java AWT calls the native platform calls the native platform (operating systems) subroutine for creating API components like TextField, ChechBox, button, etc.

For example, an AWT GUI with components like TextField, label and button will have different look and feel for the different platforms like Windows, MAC OS, and Unix. The reason for this is the platforms have different view for their native components and AWT directly calls the native subroutine that creates those components.

Java AWT Hierarchy: The hierarchies of Java AWT classes are given below.



The AWT hierarchy is as follows:

- **Component:** This is the base class for all AWT components. It defines the basic properties and methods that are common to all AWT components.
- **Container:** This is a subclass of Component that can contain other components. Containers are used to organize and layout components in a GUI.
- **Window:** This is a subclass of Container that provides a top-level window for a GUI application.
- **Frame:** This is a subclass of Window that provides a basic window with a title bar, borders, and a close button.
- **Dialog:** This is a subclass of Window that provides a modal dialog box.
- **Panel:** This is a subclass of Container that is used to group and layout components in a GUI.
- **Button:** This is a subclass of Component that provides a button that can be clicked by the user.
- **Label:** This is a subclass of Component that provides a label for displaying text.
- **TextField:** This is a subclass of Component that provides a text field for the user to enter text.
- **Checkbox:** This is a subclass of Component that provides a checkbox that can be selected or deselected by the user.
- **Choice:** This is a subclass of Component that provides a drop-down list of choices for the user to select from.
- **List:** This is a subclass of Component that provides a list of items for the user to select from.

Components

All the elements like the button, text fields, scroll bars, etc. are called components. In Java AWT, there are classes for each component as shown in above diagram. In order to place every component in a particular position on a screen, we need to add them to a container.

Container

The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container such as Frame, Dialog and Panel.

Types of containers:

There are four types of containers in Java AWT:

1. Window
2. Panel
3. Frame
4. Dialog

Window

The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window. We need to create an instance of Window class to create this container.

Panel

The Panel is the container that doesn't contain title bar, border or menu bar. It is generic container for holding the components. It can have other components like button, text field etc. An instance of Panel class creates a container, in which we can add components.

Frame

The Frame is the container that contain title bar and border and can have menu bars. It can have other components like button, text field, scrollbar etc. Frame is most widely used container while developing an AWT application.

Useful Methods of Component Class

Method	Description
public void add (Component c)	Inserts a component on this component.
public void setSize (int width,int height)	Sets the size (width and height) of the component.
public void setLayout (LayoutManager m)	Defines the layout manager for the component.
public void setVisible (boolean status)	Changes the visibility of the component, by default false.

Java AWT Example:

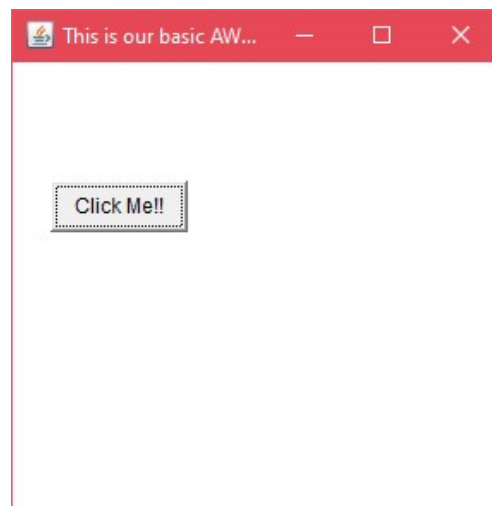
To create simple AWT example, we need a frame. There are two ways to create a GUI using Frame in AWT.

1. By extending Frame class (**inheritance**)
2. By creating the object of Frame class (**association**)

Subject & Code: Object Oriented Programming & 20IT303

Example program by extending Frame class (inheritance):

```
// importing Java AWT class
import java.awt.*;
// extending Frame class to our class AWTEExample1
public class AWTEExample1 extends Frame {
    // initializing using constructor
    AWTEExample1() {
        // creating a button
        Button b = new Button("Click Me!!");
        // setting button position on screen
        b.setBounds(30,100,80,30);
        // adding button into frame
        add(b);
        // frame size 300 width and 300 height
        setSize(300,300);
        // setting the title of Frame
        setTitle("This is our basic AWT example");
        // no layout manager
        setLayout(null);
        // now frame will be visible, by default it is not visible
        setVisible(true);
    }
    // main method
    public static void main(String args[]) {
        // creating instance of Frame class
        AWTEExample1 f = new AWTEExample1();
    } }
}
```



Creating the object of Frame class (association):

```
// importing Java AWT class
import java.awt.*;
// class AWTEmployee2 directly creates instance of Frame class
class AWTEmployee2 {
    // initializing using constructor
    AWTEmployee2() {
        // creating a Frame
        Frame f = new Frame();
        // creating a Label    Label l = new Label("Employee id:");
        // creating a Button
        Button b = new Button("Submit");
        // creating a TextField
        TextField t = new TextField();
        // setting position of above components in the frame
        l.setBounds(20, 80, 80, 30);
        t.setBounds(20, 100, 80, 30);
        b.setBounds(100, 100, 80, 30);
        // adding components into frame
        f.add(b);
        f.add(l);
        f.add(t);
        // frame size 300 width and 300 height
        f.setSize(400,300);
        // setting the title of frame
        f.setTitle("Employee info");
        // no layout
        f.setLayout(null);
        // setting visibility of frame
        f.setVisible(true);
    }
    // main method
    public static void main(String args[]) {
        // creating instance of Frame class
        AWTEmployee2 awt_obj = new AWTEmployee2();
    }
}
```

AWT Components

- An AWT component can be considered as an object that can be made visible on a graphical interface screen and through which interaction can be performed.

- Syntax:

```
public abstract class Component extends Object
                                implements ImageObserver, MenuContainer, Serializable
```

- AWT Control Fundamentals
- The AWT supports the following types of controls, these controls are subclasses of Component.
 1. Labels
 2. Push buttons
 3. Check boxes
 4. Text Field
 5. Choice lists
 6. Lists
 7. Scroll bars
 8. Text Area

1. Labels: A label is an object of type Label, and it contains a string, which it displays. Labels are passive controls that do not support any interaction with the user.

Constructors:	Description
Label()	creates a blank label
Label(String str)	creates a label that contains the string specified by str
Label(String str, int how)	creates a label that contains the string specified by str using the alignment specified by how. how: Label.LEFT, Label.RIGHT, or Label.CENTER.

Methods	Description
void setText(String str)	set or change the text in a label
String getText()	get the current label
void setAlignment(int how)	how must be one of the alignment constants shown earlier
int getAlignment()	get the alignment.

AWT Label Class Declaration

```
public class Label extends Component implements Accessible
```

AWT Label Fields

The java.awt.Component class has following fields:

1. **static int LEFT:** It specifies that the label should be left justified.
2. **static int RIGHT:** It specifies that the label should be right justified.
3. **static int CENTER:** It specifies that the label should be placed in center.

Exmple Program on AWT Label:

```
import java.awt.*;
public class LabelExample {
public static void main(String args[]){
    // creating the object of Frame class and Label class
    Frame f= new Frame ("Label example");
    Label l1, l2;
    // initializing the labels
    l1 = new Label ("First Label.");
    l2 = new Label ("Second Label.");
    // set the location of label
    l1.setBounds(50, 100, 100, 30);
    l2.setBounds(50, 150, 100, 30);
    // adding labels to the frame
    f.add(l1);
    f.add(l2);
    // setting size, layout and visibility of frame
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
}
```

Output:



2. Java AWT 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.

To perform an action on a button being pressed and released, the **ActionListener** interface needs to be implemented. The registered new listener can receive events from the button by calling **addActionListener** method of the button. The Java application can use the button's action command as a messaging protocol.

AWT Button Class Declaration

```
public class Button extends Component implements Accessible
```

Button Class Constructors

Following table shows the types of Button class constructors

Sr. no.	Constructor	Description
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

Sr. no.	Method	Description
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.

Subject & Code: Object Oriented Programming & 20IT303

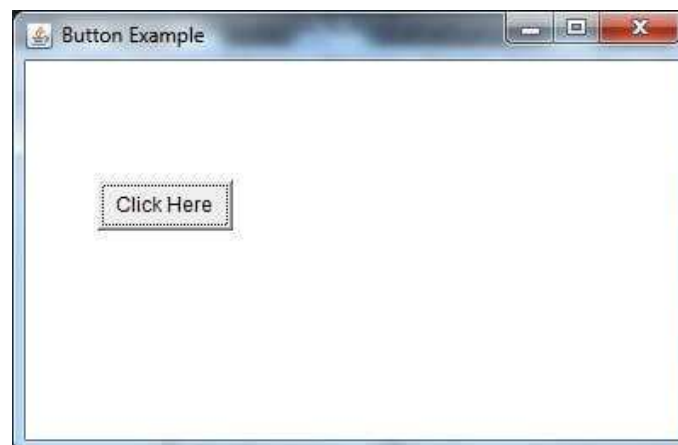
5.	<code>void addNotify()</code>	It creates the peer of the button.
6.	<code>AccessibleContext getAccessibleContext()</code>	It fetched the accessible context associated with the button.
7.	<code>void addActionListener(ActionListener l)</code>	It adds the specified action listener to get the action events from the button.
8.	<code>String getActionCommand()</code>	It returns the command name of the action event fired by the button.
9.	<code>ActionListener[] getActionListeners()</code>	It returns an array of all the action listeners registered on the button.
10.	<code>T[] getListeners(Class listenerType)</code>	It returns an array of all the objects currently registered as FooListeners upon this Button.
11.	<code>protected String paramString()</code>	It returns the string which represents the state of button.
12.	<code>protected void processActionEvent (ActionEvent e)</code>	It process the action events on the button by dispatching them to a registered ActionListener object.
13.	<code>protected void processEvent (AWTEvent e)</code>	It process the events on the button
14.	<code>void removeActionListener (ActionListener l)</code>	It removes the specified action listener so that it no longer receives action events from the button.
15.	<code>void setActionCommand(String command)</code>	It sets the command name for the action event given by the button.

Example 1: Java AWT Button Example

ButtonExample.java

```
import java.awt.*;  
  
public class ButtonExample {  
public static void main (String[] args) {  
  
    // create instance of frame with the label  
    Frame f = new Frame("Button Example");  
  
    // create instance of button with label  
    Button b = new Button("Click Here");  
  
    // set the position for the button in frame  
    b.setBounds(50,100,80,30);  
  
    // add button to the frame  
    f.add(b);  
    // set size, layout and visibility of frame  
    f.setSize(400,400);  
    f.setLayout(null);  
    f.setVisible(true);  
}  
}
```

Output:



Subject & Code: Object Oriented Programming & 20IT303

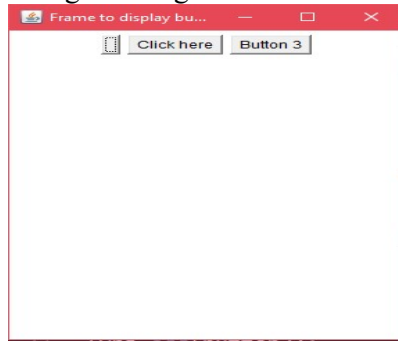
Example 2:

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

public class ButtonExample2
{
// creating instances of Frame class and Button class
Frame fObj;
Button button1, button2, button3;
// instantiating using the constructor
ButtonExample2() {
fObj = new Frame ("Frame to display buttons");
button1 = new Button();
button2 = new Button ("Click here");
button3 = new Button();
button3.setLabel("Button 3");
fObj.add(button1);
fObj.add(button2);
fObj.add(button3);
fObj.setLayout(new FlowLayout());
fObj.setSize(300,400);
fObj.setVisible(true);
}
// main method
public static void main (String args[])
{
new ButtonExample2();
}
}
```

Output:

Subject & Code: Object Oriented Programming & 20IT303



Example: Java AWT Button Example with ActionListener

In the following example, we are handling the button click events by implementing ActionListener Interface.

ButtonExample3.java

```
// importing necessary libraries
import java.awt.*;
import java.awt.event.*;
public class ButtonExample3 {
public static void main(String[] args) {
    // create instance of frame with the label
    Frame f = new Frame("Button Example");
    final TextField tf=new TextField();
    tf.setBounds(50,50, 150,20);
    // create instance of button with label
    Button b=new Button("Click Here");
    // set the position for the button in frame
    b.setBounds(50,100,60,30);
    b.addActionListener(new ActionListener() {
        public void actionPerformed (ActionEvent e) {
            tf.setText("Welcome to Javatpoint.");
        }
    });
    // adding button the frame
    f.add(b);
    // adding textfield the frame
    f.add(tf);
    // setting size, layout and visibility
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
```

} }

3. Check boxes

- A check box is a control that is used to turn an option on or off.
- It consists of a small box that can either contain a check mark or not.
- There is a label associated with each check box that describes what option the box represents.
- You change the state of a check box by clicking on it. Check boxes can be used individually or as part of a group.
- Check boxes are objects of the Checkbox class.

Constructors:	Description
Checkbox()	creates a blank check box and state is unchecked.
Checkbox(String str)	creates a check box that specified by str as label state is unchecked.
Checkbox(String str, boolean on)	allows you to set the initial state of the check box. If true, the check box is initially checked; otherwise, it is cleared.
Checkbox(String str, boolean on, CheckboxGroup cbGroup)	create a check box whose label is specified by str and whose group is specified by cbGroup. If this check box is not part of a group, then cbGroup must be null.
Checkbox(String str, CheckboxGroup cbGroup, boolean on)	create a check box whose label is specified by str and whose group is specified by cbGroup. If this check box is not part of a group, then cbGroup must be null.

Methods	Description
void setState(boolean on)	set or change the state
boolean getState()	get the current state
void setLabel(String str)	set or change the label
String getLabel()	get the current label

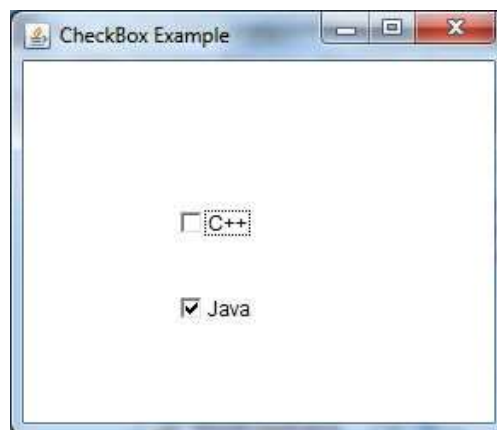
Example:

```
import java.awt.*;
public class CheckboxExample1
{
// constructor to initialize
    CheckboxExample1() {
// creating the frame with the title
        Frame f = new Frame("Checkbox Example");
// creating the checkboxes
        Checkbox checkbox1 = new Checkbox("C++");
        checkbox1.setBounds(100, 100, 50, 50);
        Checkbox checkbox2 = new Checkbox("Java", true);
```

Subject & Code: Object Oriented Programming & 20IT303

```
// setting location of checkbox in frame
checkbox2.setBounds(100, 150, 50, 50);
// adding checkboxes to frame
f.add(checkbox1);    f.add(checkbox2);
// setting size, layout and visibility of frame
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
}
// main method
public static void main (String args[])
{
    new CheckboxExample1();
}
}
```

Output:



4. Text Field

- The TextField class implements a single-line text-entry area, usually called an edit control.
- Text fields allow the user to enter strings and to edit the text using the arrow keys, cut and paste keys, and mouse selections.

Constructors:	Description
TextField()	creates a default text field
TextField(int numChars)	creates a text field that is numChars width
TextField(String str)	initializes the text field with the string contained in str
TextField(String str, int numChars)	initializes a text field and sets its width

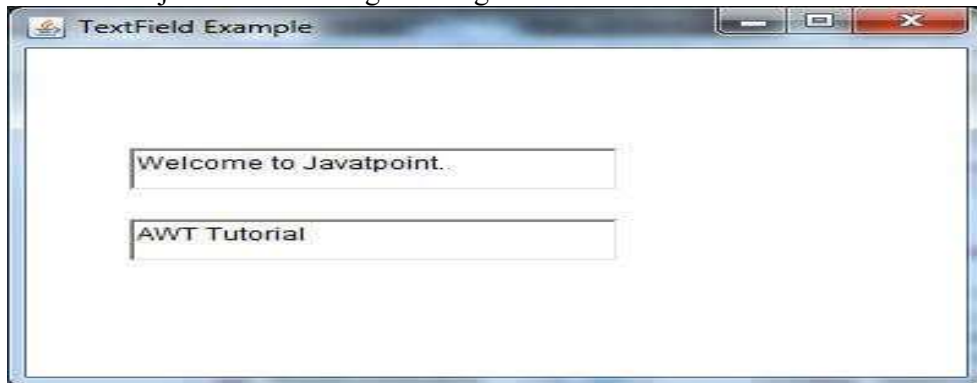
Methods	Description
void setText(String str)	Set the text specified in str
String getText()	Get the text with in text field.
void select(int startIndex, int endIndex)	select a portion of text
String getSelectedText()	returns the selected text
void setEditable(boolean canEdit)	if canEdit is true, the text may be changed. If it is false, the text cannot be altered.
boolean isEditable()	isEditable() returns true if the text may be changed and false if not.
void setEchoChar(char ch)	disable the echoing of the characters as they are typed
boolean echoCharIsSet()	check a text field to see if it is in this mode
char getEchoChar()	retrieve the echo character

Example:

```
import java.awt.*;

public class TextFieldExample1 {
    // main method
    public static void main(String args[]) {
        // creating a frame
        Frame f = new Frame("TextField Example");
        // creating objects of textfield
        TextField t1, t2;
        // instantiating the textfield objects
        // setting the location of those objects in the frame
        t1 = new TextField("Welcome to Javatpoint.");
        t1.setBounds(50, 100, 200, 30);
        t2 = new TextField("AWT Tutorial");
        t2.setBounds(50, 150, 200, 30);
        // adding the components to frame
        f.add(t1);
        f.add(t2);
        // setting size, layout and visibility of frame
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```

Output:



Event Handling in Java:

Changing the state of an object is known as an event. For example, click on button, dragging mouse etc. The *java.awt.event* package provides many event classes and Listener interfaces for event handling.

Types of Events: The events can be broadly classified into two categories:

Foreground Events - Those events which require the direct interaction of user. They are generated as consequences of a person interacting with the graphical components in Graphical User Interface. For example, **clicking on a button, moving the mouse, entering a character** through keyboard, selecting an item from list, scrolling the page etc.

Background Events - Those events that require the interaction of end user are known as background events. Operating system interrupts, hardware or software failure, timer expires, an operation completion are the example of background events

Event Handling: Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism has the code which is known as event handler that is executed when an event occurs. Java Uses the **Delegation Event Model** to handle the events. This model defines the standard mechanism to generate and handle the events. Let's have a brief introduction to this model.

Delegation Event Model: The Delegation Event model is defined to handle events in GUI programming languages. The GUI stands for Graphical User Interface, where a user graphically/visually interacts with the system.

The **delegation model** is a programming design pattern that is used to handle events and event-driven programming in graphical user interfaces (GUIs). The delegation model works by separating the concerns of event generation and event handling, allowing for greater modularity and flexibility in GUI programming.

The **Event model** is based on two things, i.e., Event Source and Event Listeners.

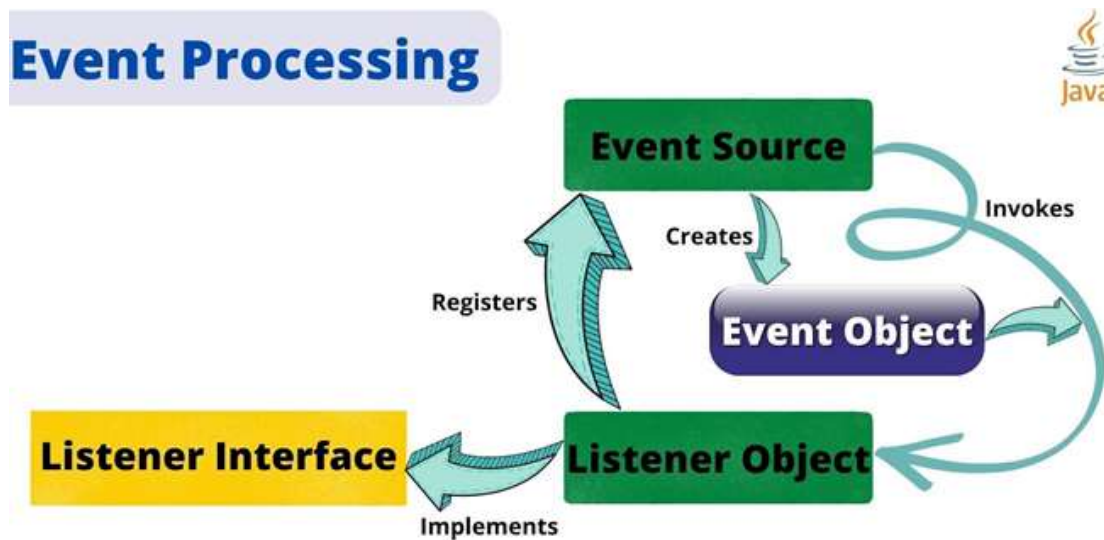
- Event Source means any object which creates the message or event.
- Event Listeners are the object which receives the message or event.

Subject & Code: Object Oriented Programming & 20IT303

The **Delegation event model** is based upon three things: Event Source, Event Listeners, and Event Objects.

- Event Source (component) is the class used to broadcast the events.
- Event Listeners are the classes that receive notifications of events.
- Event Object is the class object which describes the event.

In the **Delegation model**, a source generates an event and forwards it to one or more listeners, where the listener waits until they receive an event. Once the listener gets the event, it is processed by the listener, and then they return it. The UI elements can delegate an event's processing to a separate function.



Java Event classes and Listener interfaces

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener

Subject & Code: Object Oriented Programming & 20IT303

KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

Steps to perform Event Handling

Following steps are required to perform event handling:

1. Implement an appropriate interface in the class.
2. Register the component with the Listener

Registration Methods

For registering the component with the Listener, many classes provide the registration methods. For example

- **Button**
 - `public void addActionListener(ActionListener a){}`
- **MenuItem**
 - `public void addActionListener(ActionListener a){}`
- **TextField**
 - `public void addActionListener(ActionListener a){}`
 - `public void addTextListener(TextListener a){}`
- **TextArea**
 - `public void addTextListener(TextListener a){}`

Subject & Code: Object Oriented Programming & 20IT303

- **Checkbox**
 - public void addItemListener(ItemListener a){}
- **Choice**
 - public void addItemListener(ItemListener a){}
- **List**
 - public void addActionListener(ActionListener a){}
 - public void addItemListener(ItemListener a){}

Java Event Handling Code

We can put the event handling code into one of the following places:

1. Within class
2. Other class
3. Anonymous class

Java event handling by implementing ActionListener

```
import java.awt.*;
import java.awt.event.*;
class AEvent extends Frame implements ActionListener{
    TextField tf;
    AEvent(){

//create components
    tf=new TextField();
    tf.setBounds(60,50,170,20);
    Button b=new Button("click me");
    b.setBounds(100,120,80,30);

//register listener
    b.addActionListener(this);//passing current instance

//add components and set size, layout and visibility
    add(b);add(tf);
    setSize(300,300);
    setLayout(null);
    setVisible(true);
```

Subject & Code: Object Oriented Programming & 20IT303

```
}  
public void actionPerformed(ActionEvent e){  
tf.setText("Welcome");  
}  
public static void main(String args[]){  
new AEvent();  
}  
}
```

public void setBounds(int xaxis, int yaxis, int width, int height); have been used in the above example that sets the position of the component it may be button, textfield etc.



2) Java event handling by outer class

```
import java.awt.*;  
import java.awt.event.*;  
class AEvent2 extends Frame{  
TextField tf;  
AEvent2(){  
//create components  
tf=new TextField();  
tf.setBounds(60,50,170,20);  
Button b=new Button("click me");  
b.setBounds(100,120,80,30);  
//register listener  
Outer o=new Outer(this);  
b.addActionListener(o);//passing outer class instance  
//add components and set size, layout and visibility  
add(b);add(tf);  
setSize(300,300);
```

Prepared By : Suresh Kumar Kallagunta, Asst.Professor, Dept.of IT

Subject & Code: Object Oriented Programming & 20IT303

```
setLayout(null);
setVisible(true);
}
public static void main(String args[]){
new AEvent2();
}
}
import java.awt.event.*;
class Outer implements ActionListener{
AEvent2 obj;
Outer(AEvent2 obj){
this.obj=obj;
}
public void actionPerformed(ActionEvent e){
obj.tf.setText("welcome");
}
}
```

3) Java event handling by anonymous class

```
import java.awt.*;
import java.awt.event.*;
class AEvent3 extends Frame{
TextField tf;
AEvent3(){
tf=new TextField();
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(50,120,80,30);

b.addActionListener(new ActionListener(){
public void actionPerformed(){
tf.setText("hello");
}
});
add(b);add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);
```

Subject & Code: Object Oriented Programming & 20IT303

```
}  
public static void main(String args[]){  
    new AEvent3();  
}  
}
```

Java MouseListener Interface

The Java MouseListener is notified whenever you change the state of mouse. It is notified against MouseEvent. The MouseListener interface is found in java.awt.event package. It has five methods.

Methods of MouseListener interface

There are 5 significant methods found in **MouseListener interface**. They are:

- **public abstract void mouseClicked**(MouseEvent e);
- **public abstract void mouseEntered**(MouseEvent e);
- **public abstract void mouseExited**(MouseEvent e);
- **public abstract void mousePressed**(MouseEvent e);
- **public abstract void mouseReleased**(MouseEvent e);

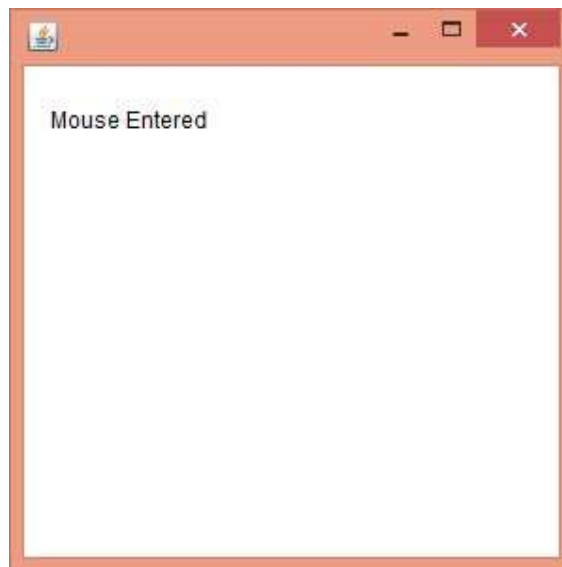
Mouse Listener Example:

```
import java.awt.*;  
import java.awt.event.*;  
public class MouseListenerExample extends Frame implements MouseListener{  
    Label l;  
    MouseListenerExample(){  
        addMouseListener(this);  
  
        l=new Label();  
        l.setBounds(20,50,100,20);  
        add(l);  
        setSize(300,300);  
        setLayout(null);  
    }  
}
```

Prepared By : Suresh Kumar Kallagunta, Asst.Professor, Dept.of IT

Subject & Code: Object Oriented Programming & 20IT303

```
setVisible(true);
}
public void mouseClicked(MouseEvent e) {
    l.setText("Mouse Clicked");
}
public void mouseEntered(MouseEvent e) {
    l.setText("Mouse Entered");
}
public void mouseExited(MouseEvent e) {
    l.setText("Mouse Exited");
}
public void mousePressed(MouseEvent e) {
    l.setText("Mouse Pressed");
}
public void mouseReleased(MouseEvent e) {
    l.setText("Mouse Released");
}
public static void main(String[] args) {
    new MouseListenerExample();
}
}
```



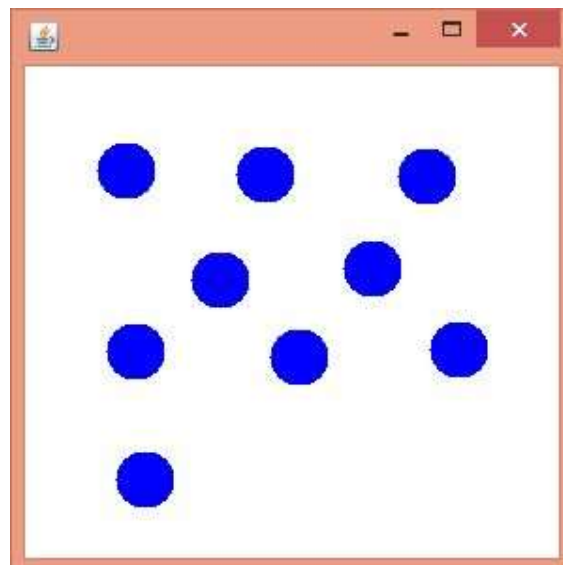
Java MouseListener Example 2

```
import java.awt.*;
import java.awt.event.*;
public class MouseListenerExample2 extends Frame implements MouseListener{
    MouseListenerExample2(){
        addMouseListener(this);

        setSize(300,300);
        setLayout(null);
        setVisible(true);
    }

    public void mouseClicked(MouseEvent e) {
        Graphics g=getGraphics();
        g.setColor(Color.BLUE);
        g.fillOval(e.getX(),e.getY(),30,30);
    }
    public void mouseEntered(MouseEvent e) {}
    public void mouseExited(MouseEvent e) {}
    public void mousePressed(MouseEvent e) {}
    public void mouseReleased(MouseEvent e) {}

    public static void main(String[] args) {
        new MouseListenerExample2();
    }
}
```



Java MouseMotionListener Interface

The Java MouseMotionListener is notified whenever you move or drag mouse. It is notified against MouseEvent. The MouseMotionListener interface is found in java.awt.event package. It has two methods.

Methods of MouseMotionListener interface

The signature of 2 methods found in MouseMotionListener interface are given below:

- **public abstract void** mouseDragged(MouseEvent e);
- **public abstract void** mouseMoved(MouseEvent e);

Java MouseMotionListener Example

```
import java.awt.*;
import java.awt.event.*;
public class MouseMotionListenerExample extends Frame implements MouseMotionListener {
    MouseMotionListenerExample() {
        addMouseMotionListener(this);

        setSize(300,300);
        setLayout(null);
        setVisible(true);
    }
    public void mouseDragged(MouseEvent e) {
        Graphics g=getGraphics();
        g.setColor(Color.BLUE);
        g.fillOval(e.getX(),e.getY(),20,20);
    }
    public void mouseMoved(MouseEvent e) {}

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

Java KeyListener Interface

The Java **KeyListener** is notified whenever you change the state of key. It is notified against **KeyEvent**. The **KeyListener** interface is found in `java.awt.event` package, and it has three methods.

Interface declaration

Following is the declaration for `java.awt.event.KeyListener` interface:

- **public interface** `KeyListener` **extends** `EventListener`

Methods of KeyListener interface

The signature of 3 methods found in `KeyListener` interface are given below:

Sr. no.	Method name	Description
1.	<code>public abstract void keyPressed (KeyEvent e);</code>	It is invoked when a key has been pressed.
2.	<code>public abstract void keyReleased (KeyEvent e);</code>	It is invoked when a key has been released.
3.	<code>public abstract void keyTyped (KeyEvent e);</code>	It is invoked when a key has been typed.

Java KeyListener Example

In the following example, we are implementing the methods of the `KeyListener` interface.

`KeyListenerExample.java`

```
// importing awt libraries
import java.awt.*;
import java.awt.event.*;
// class which inherits Frame class and implements KeyListener interface
public class KeyListenerExample extends Frame implements KeyListener {
// creating object of Label class and TextArea class
Label l;
    TextArea area;
// class constructor
    KeyListenerExample() {
```

Subject & Code: Object Oriented Programming & 20IT303

```
// creating the label
l = new Label();
// setting the location of the label in frame
l.setBounds (20, 50, 100, 20);
// creating the text area
area = new TextArea();
// setting the location of text area
area.setBounds (20, 80, 300, 300);
// adding the KeyListener to the text area
area.addKeyListener(this);
// adding the label and text area to the frame
add(l);
add(area);
// setting the size, layout and visibility of frame
setSize (400, 400);
setLayout (null);
setVisible (true);
}
// overriding the keyPressed() method of KeyListener interface where we set the text of the label whn key is pres
sed
public void keyPressed (KeyEvent e) {
    l.setText ("Key Pressed");
}
// overriding the keyReleased() method of KeyListener interface where we set the text of the label wen key is rele
ased
public void keyReleased (KeyEvent e) {
    l.setText ("Key Released");
}
// overriding the keyTyped() method of KeyListener interface where we set the text of the label when a key is typ
ed
public void keyTyped (KeyEvent e) {
    l.setText ("Key Typed");
}
public static void main(String[] args) {
    new KeyListenerExample();
}
}
```

Output:



Layout management

- A layout manager automatically arranges your controls within a window by using some type of algorithm.
- While it is possible to lay out Java controls by hand, too, you generally won't want to, for two main reasons.
 - ✓ First, it is very tedious to manually lay out a large number of components.
 - ✓ Second, sometimes the width and height information is not yet available when you need to arrange some control, because the native toolkit components haven't been realized.
- 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.

Syntax: `void setLayout(LayoutManager layoutObj)`

There are the following classes that represent the layout managers:

1. `java.awt.BorderLayout`
2. `java.awt.FlowLayout`
3. `java.awt.GridLayout`
4. `java.awt.CardLayout`
5. `java.awt.GridBagLayout`
6. `javax.swing.BoxLayout`
7. `javax.swing.GroupLayout`
8. `javax.swing.ScrollPaneLayout`
9. `javax.swing.SpringLayout` etc.

Java BorderLayout

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 a frame or window. The `BorderLayout` provides five constants for each region:

1. **`public static final int NORTH`**
2. **`public static final int SOUTH`**
3. **`public static final int EAST`**
4. **`public static final int WEST`**
5. **`public static final int CENTER`**

Constructors of BorderLayout class:

- **BorderLayout():** creates a border layout but with no gaps between the components.
- **BorderLayout(int hgap, int vgap):** creates a border layout with the given horizontal and vertical gaps between the components.

Example of BorderLayout class: Using BorderLayout() constructor

FileName: Border.java

```
import java.awt.*;
import javax.swing.*;
public class Border
{
    JFrame f;
    BorderLayout()
    {
        f = new JFrame();
        // creating buttons
        JButton b1 = new JButton("NORTH"); // the button will be labeled as NORTH
        JButton b2 = new JButton("SOUTH"); // the button will be labeled as SOUTH
        JButton b3 = new JButton("EAST"); // the button will be labeled as EAST
        JButton b4 = new JButton("WEST"); // the button will be labeled as WEST
        JButton b5 = new JButton("CENTER"); // the button will be labeled as CENTER

        f.add(b1, BorderLayout.NORTH); // b1 will be placed in the North Direction
        f.add(b2, BorderLayout.SOUTH); // b2 will be placed in the South Direction
        f.add(b3, BorderLayout.EAST); // b2 will be placed in the East Direction
        f.add(b4, BorderLayout.WEST); // b2 will be placed in the West Direction
        f.add(b5, BorderLayout.CENTER); // b2 will be placed in the Center

        f.setSize(300, 300);
        f.setVisible(true);
    }
    public static void main(String[] args) {
        new Border();
    }
}
```

Output:



Example of BorderLayout class: Using BorderLayout(int hgap, int vgap) constructor

The following example inserts horizontal and vertical gaps between buttons using the parameterized constructor BorderLayout(int hgap, int vgap)

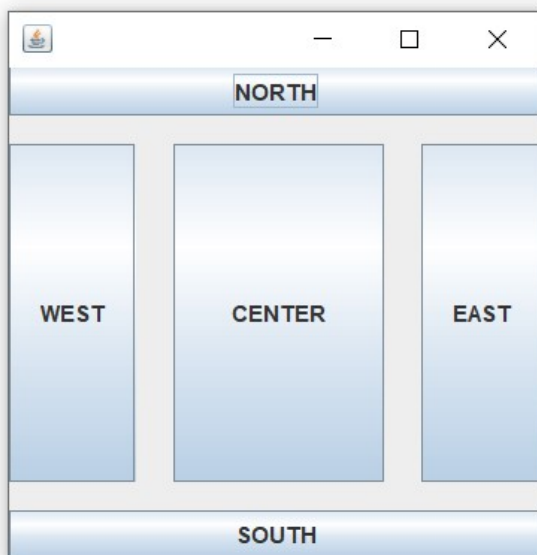
FileName: BorderLayoutExample.java

```
// import statement
import java.awt.*;
import javax.swing.*;
public class BorderLayoutExample
{
JFrame jframe;
// constructor
BorderLayoutExample()
{
// creating a Frame
jframe = new JFrame();
// create buttons
JButton btn1 = new JButton("NORTH");
JButton btn2 = new JButton("SOUTH");
JButton btn3 = new JButton("EAST");
JButton btn4 = new JButton("WEST");
```

Subject & Code: Object Oriented Programming & 20IT303

```
JButton btn5 = new JButton("CENTER");  
    // creating an object of the BorderLayout class using  
    // the parameterized constructor where the horizontal gap is 20  
    // and vertical gap is 15. The gap will be evident when buttons are placed  
    // in the frame  
jframe.setLayout(new BorderLayout(20, 15));  
jframe.add(btn1, BorderLayout.NORTH);  
jframe.add(btn2, BorderLayout.SOUTH);  
jframe.add(btn3, BorderLayout.EAST);  
jframe.add(btn4, BorderLayout.WEST);  
jframe.add(btn5, BorderLayout.CENTER);  
jframe.setSize(300,300);  
jframe.setVisible(true);  
}  
// main method  
public static void main(String argsv[])  
{  
    new BorderLayoutExample();  
}  
}
```

Output:



Java GridLayout

The Java GridLayout class is used to arrange the components in a rectangular grid. One component is displayed in each rectangle.

Constructors of GridLayout class

1. **GridLayout():** creates a grid layout with one column per component in a row.
2. **GridLayout(int rows, int columns):** creates a grid layout with the given rows and columns but no gaps between the components.
3. **GridLayout(int rows, int columns, int hgap, int vgap):** creates a grid layout with the given rows and columns along with given horizontal and vertical gaps.

Example of GridLayout class: Using GridLayout() Constructor

The GridLayout() constructor creates only one row. The following example shows the usage of the parameterless constructor.

FileName: GridLayoutExample.java

```
// import statements
import java.awt.*;
import javax.swing.*;

public class GridLayoutExample
{
    JFrame frameObj;

    // constructor
    GridLayoutExample()
    {
        frameObj = new JFrame();

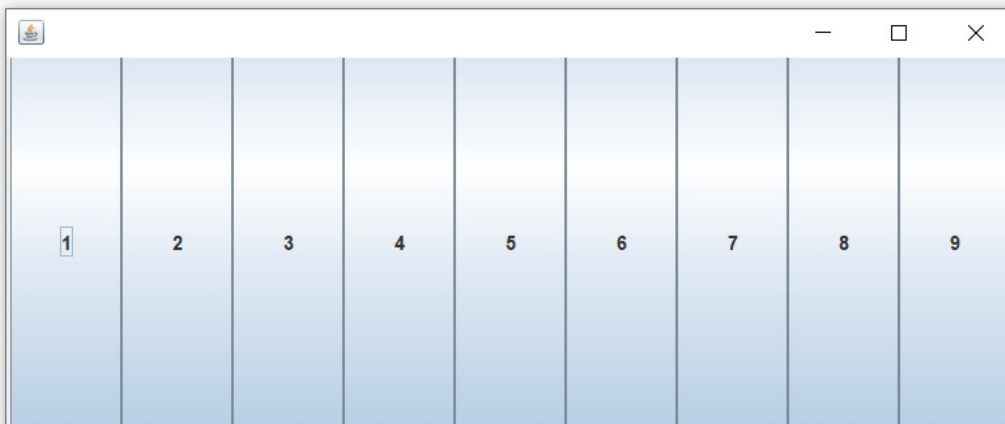
        // creating 9 buttons
        JButton btn1 = new JButton("1");
        JButton btn2 = new JButton("2");
        JButton btn3 = new JButton("3");
        JButton btn4 = new JButton("4");
        JButton btn5 = new JButton("5");
        JButton btn6 = new JButton("6");
        JButton btn7 = new JButton("7");
        JButton btn8 = new JButton("8");
        JButton btn9 = new JButton("9");
    }
}
```

Subject & Code: Object Oriented Programming & 20IT303

```
// adding buttons to the frame
// since, we are using the parameterless constructor, therefore;
// the number of columns is equal to the number of buttons we
// are adding to the frame. The row count remains one.
frameObj.add(btn1); frameObj.add(btn2); frameObj.add(btn3);
frameObj.add(btn4); frameObj.add(btn5); frameObj.add(btn6);
frameObj.add(btn7); frameObj.add(btn8); frameObj.add(btn9);

// setting the grid layout using the parameterless constructor
frameObj.setLayout(new GridLayout());
frameObj.setSize(300, 300);
frameObj.setVisible(true);
}
// main method
public static void main(String argsv[])
{
new GridLayoutExample();
}
}
```

Output:



Subject & Code: Object Oriented Programming & 20IT303

Example of GridLayout class: Using GridLayout(int rows, int columns) Constructor

FileName: MyGridLayout.java

```
import java.awt.*;
import javax.swing.*;
public class MyGridLayout {
    JFrame f;
    MyGridLayout() {
        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");
        // adding buttons to the frame
        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);
        // setting grid layout of 3 rows and 3 columns
        f.setLayout(new GridLayout(3,3));
        f.setSize(300,300);
        f.setVisible(true);
    }
    public static void main(String[] args) {
        new MyGridLayout(); } }
}
```

Output:



Prepared By : Suresh Kumar Kallagunta, Asst.Professor, Dept.of IT

Subject & Code: Object Oriented Programming & 20IT303

Example of GridLayout class: Using GridLayout(int rows, int columns, int hgap, int vgap) Constructor

The following example inserts horizontal and vertical gaps between buttons using the parameterized constructor GridLayout(int rows, int columns, int hgap, int vgap).

FileName: GridLayoutExample1.java

```
// import statements
import java.awt.*;
import javax.swing.*;

public class GridLayoutExample1
{
    JFrame frameObj;

    // constructor
    GridLayoutExample1()
    {
        frameObj = new JFrame();

        // creating 9 buttons
        JButton btn1 = new JButton("1");
        JButton btn2 = new JButton("2");
        JButton btn3 = new JButton("3");
        JButton btn4 = new JButton("4");
        JButton btn5 = new JButton("5");
        JButton btn6 = new JButton("6");
        JButton btn7 = new JButton("7");
        JButton btn8 = new JButton("8");
        JButton btn9 = new JButton("9");

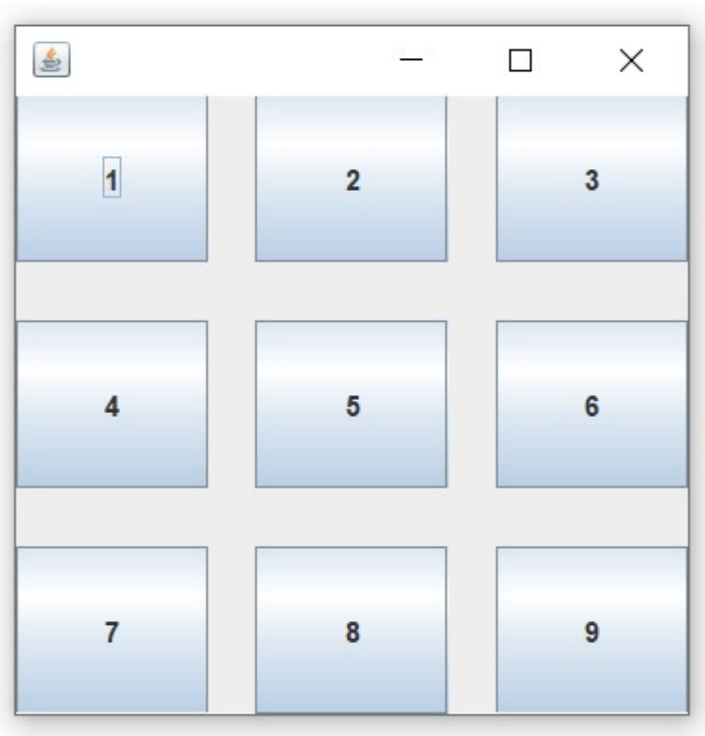
        // adding buttons to the frame
        // since, we are using the parameterless constructor, therefore;
        // the number of columns is equal to the number of buttons we
        // are adding to the frame. The row count remains one.
        frameObj.add(btn1); frameObj.add(btn2); frameObj.add(btn3);
        frameObj.add(btn4); frameObj.add(btn5); frameObj.add(btn6);
        frameObj.add(btn7); frameObj.add(btn8); frameObj.add(btn9);
        // setting the grid layout
```

Prepared By : Suresh Kumar Kallagunta, Asst.Professor, Dept.of IT

Subject & Code: Object Oriented Programming & 20IT303

```
// a 3 * 3 grid is created with the horizontal gap 20
// and vertical gap 25
frameObj.setLayout(new GridLayout(3, 3, 20, 25));
frameObj.setSize(300, 300);
frameObj.setVisible(true);
}
// main method
public static void main(String argsv[])
{
new GridLayoutExample();
}
}
```

Output:



Java FlowLayout

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

Fields of FlowLayout class

1. **public static final int LEFT**
2. **public static final int RIGHT**

Prepared By : Suresh Kumar Kallagunta, Asst.Professor, Dept.of IT

3. **public static final int CENTER**
4. **public static final int LEADING**
5. **public static final int TRAILING**

Constructors of FlowLayout class

1. **FlowLayout():** creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.
2. **FlowLayout(int align):** creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.
3. **FlowLayout(int align, int hgap, int vgap):** creates a flow layout with the given alignment and the given horizontal and vertical gap.

Example of FlowLayout class: Using FlowLayout() constructor

FileName: FlowLayoutExample.java

```
// import statements
import java.awt.*;
import javax.swing.*;

public class FlowLayoutExample
{

JFrame frameObj;

// constructor
FlowLayoutExample()
{
    // creating a frame object
    frameObj = new JFrame();

    // creating the buttons
    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");
```

Subject & Code: Object Oriented Programming & 20IT303

```
JButton b8 = new JButton("8");
JButton b9 = new JButton("9");
JButton b10 = new JButton("10");

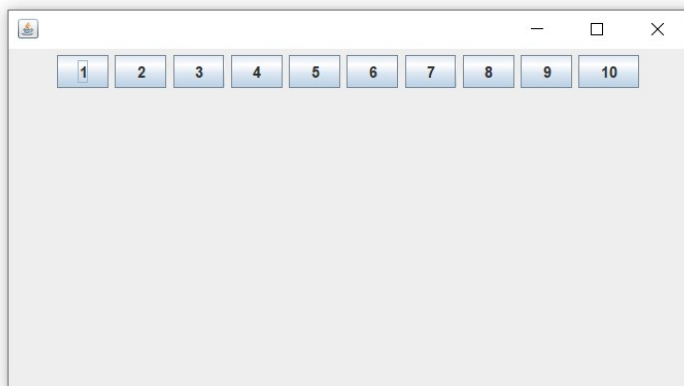
// adding the buttons to frame
frameObj.add(b1); frameObj.add(b2); frameObj.add(b3); frameObj.add(b4);
frameObj.add(b5); frameObj.add(b6); frameObj.add(b7); frameObj.add(b8);
frameObj.add(b9); frameObj.add(b10);

// parameter less constructor is used
// therefore, alignment is center
// horizontal as well as the vertical gap is 5 units.
frameObj.setLayout(new FlowLayout());

frameObj.setSize(300, 300);
frameObj.setVisible(true);
}

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

Output:



Java BorderLayout

The **Java BorderLayout class** is used to arrange the components either vertically or horizontally. For this purpose, the BorderLayout class provides four constants. They are as follows:

Note: The BorderLayout class is found in javax.swing package

Fields of BorderLayout Class

1. **public static final int X_AXIS:** Alignment of the components are horizontal from left to right.
2. **public static final int Y_AXIS:** Alignment of the components are vertical from top to bottom.
3. **public static final int LINE_AXIS:** Alignment of the components is similar to the way words are aligned in a line, which is based on the ComponentOrientation property of the container. If the ComponentOrientation property of the container is horizontal, then the components are aligned horizontally; otherwise, the components are aligned vertically.

For horizontal orientations, we have two cases: left to right, and right to left. If the value ComponentOrientation property of the container is from left to right, then the components are rendered from left to right, and for right to left, the rendering of components is also from right to left. In the case of vertical orientations, the components are always rendered from top to bottom.

4. **public static final int PAGE_AXIS:** Alignment of the components is similar to the way text lines are put on a page, which is based on the ComponentOrientation property of the container. If the ComponentOrientation property of the container is horizontal, then components are aligned vertically; otherwise, the components are aligned horizontally.

For horizontal orientations, we have two cases: left to right, and right to left. If the value ComponentOrientation property of the container is also from left to right, then the components are rendered from left to right, and for right to left, the rendering of components is from right to left. In the case of vertical orientations, the components are always rendered from top to bottom.

Constructor of BorderLayout class

1. **BoxLayout(Container c, int axis):** creates a box layout that arranges the components with the given axis.

Example of BorderLayout class with Y-AXIS:

FileName: **BoxLayoutExample1.java**

```
import java.awt.*;  
import javax.swing.*;
```

```
public class BorderLayoutExample1 extends Frame {
```


Subject & Code: Object Oriented Programming & 20IT303

```
Button buttons[];
```

```
public BoxLayoutExample1 () {  
    buttons = new Button [5];  
  
    for (int i = 0;i<5;i++) {  
        buttons[i] = new Button ("Button " + (i + 1));  
        // adding the buttons so that it can be displayed  
        add (buttons[i]);  
    }  
    // the buttons will be placed horizontally  
    setLayout (new BoxLayout (this, BoxLayout.Y_AXIS));  
    setSize(400,400);  
    setVisible(true);  
}  
// main method  
public static void main(String args[]){  
    BoxLayoutExample1 b=new BoxLayoutExample1();  
}  
}
```

Output:



Java CardLayout

The **Java 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.

Constructors of CardLayout Class

1. **CardLayout():** creates a card layout with zero horizontal and vertical gap.
2. **CardLayout(int hgap, int vgap):** creates a card layout with the given horizontal and vertical gap.

Commonly Used Methods of CardLayout Class

- **public void next(Container parent):** is used to flip to the next card of the given container.
- **public void previous(Container parent):** is used to flip to the previous card of the given container.
- **public void first(Container parent):** is used to flip to the first card of the given container.
- **public void last(Container parent):** is used to flip to the last card of the given container.
- **public void show(Container parent, String name):** is used to flip to the specified card with the given name.

Example of CardLayout Class: Using Default Constructor

The following program uses the next() method to move to the next card of the container.

FileName: CardLayoutExample1.java

```
// import statements
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class CardLayoutExample1 extends JFrame implements ActionListener
{

    CardLayout crd;

    // button variables to hold the references of buttons
    JButton btn1, btn2, btn3;
    Container cPane;

    // constructor of the class
```

Subject & Code: Object Oriented Programming & 20IT303

```
CardLayoutExample1()
```

```
{
```

```
cPane = getContentPane();
```

```
//default constructor used
```

```
// therefore, components will
```

```
// cover the whole area
```

```
crd = new CardLayout();
```

```
cPane.setLayout(crd);
```

```
// creating the buttons
```

```
btn1 = new JButton("Apple");
```

```
btn2 = new JButton("Boy");
```

```
btn3 = new JButton("Cat");
```

```
// adding listeners to it
```

```
btn1.addActionListener(this);
```

```
btn2.addActionListener(this);
```

```
btn3.addActionListener(this);
```

```
cPane.add("a", btn1); // first card is the button btn1
```

```
cPane.add("b", btn2); // first card is the button btn2
```

```
cPane.add("c", btn3); // first card is the button btn3
```

```
}
```

```
public void actionPerformed(ActionEvent e)
```

```
{
```

```
// Upon clicking the button, the next card of the container is shown
```

```
// after the last card, again, the first card of the container is shown upon clicking
```

```
crd.next(cPane);
```

```
}
```

```
// main method
```

```
public static void main(String argsv[])
```

```
{
```

```
// creating an object of the class CardLayoutExample1
```

Subject & Code: Object Oriented Programming & 20IT303

```
CardLayoutExample1 crdl = new CardLayoutExample1();
```

```
// size is 300 * 300
```

```
crdl.setSize(300, 300);
```

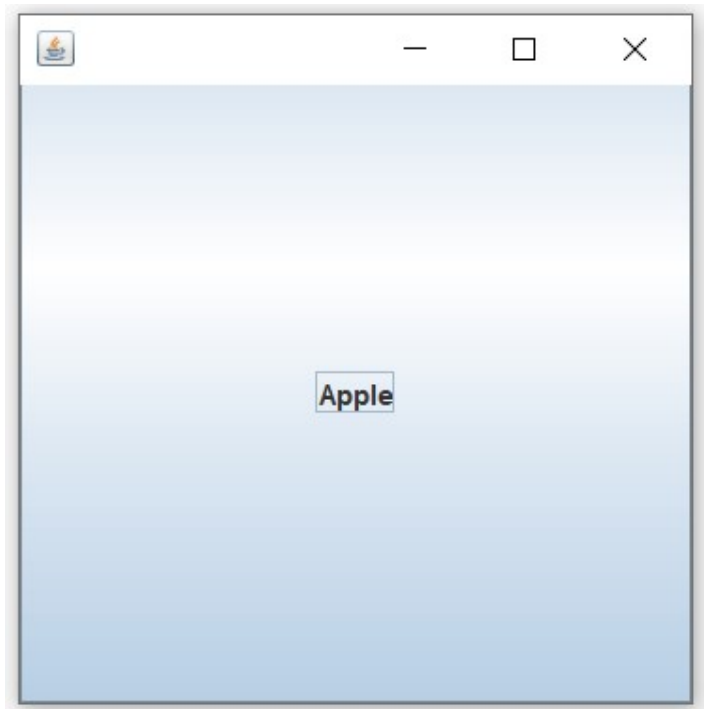
```
crdl.setVisible(true);
```

```
crdl.setDefaultCloseOperation(EXIT_ON_CLOSE);
```

```
}
```

```
}
```

Output:



Java Swing Tutorial

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.

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:

There are many differences between java awt and swing that are given below.

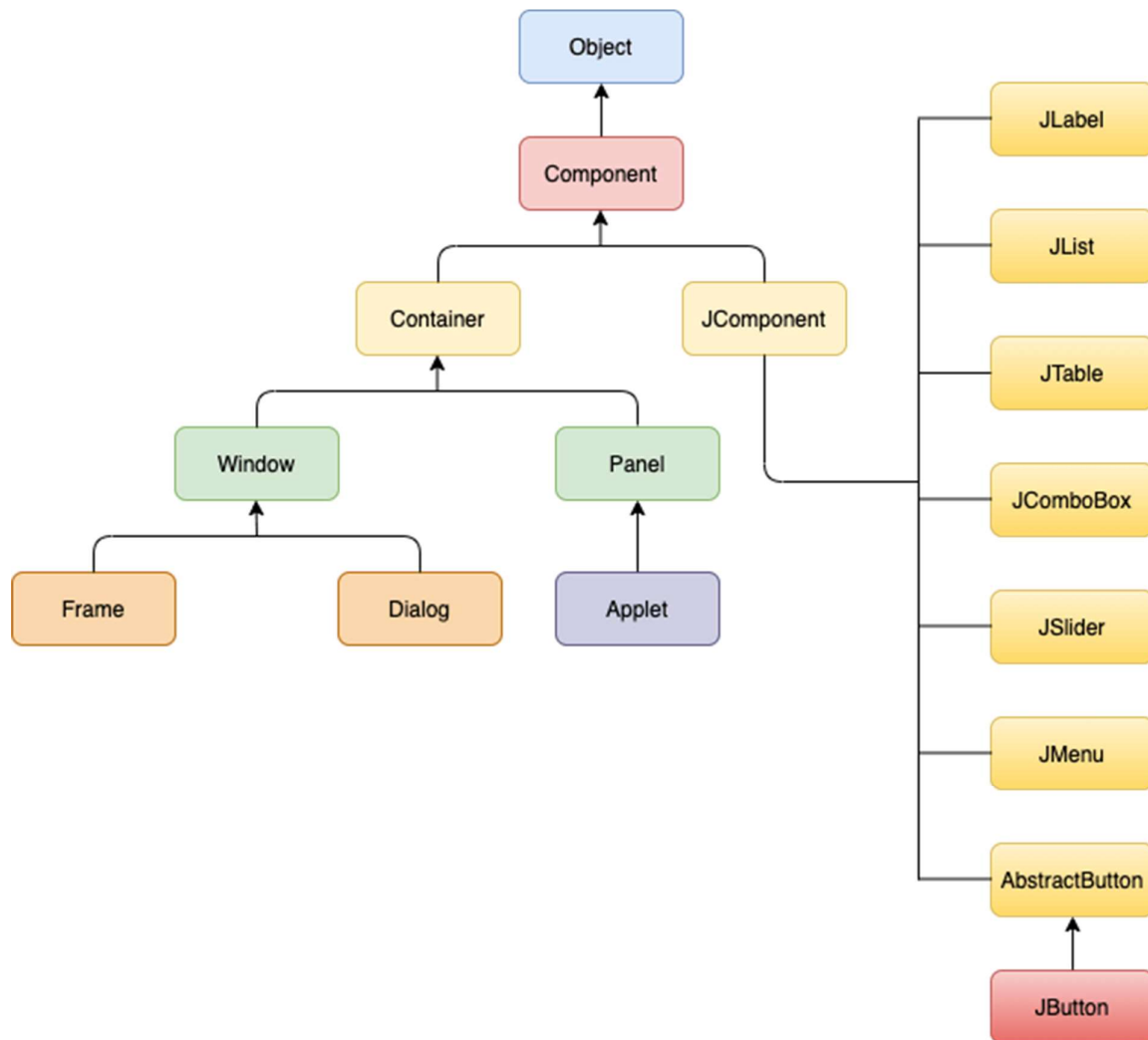
No.	Java AWT	Java Swing
1)	AWT components are platform-dependent .	Java swing components are platform-independent .
2)	AWT components are heavyweight .	Swing components are lightweight .
3)	AWT doesn't support pluggable look and feel .	Swing supports pluggable look and feel .
4)	AWT provides less components 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 .

Subject & Code: Object Oriented Programming & 20IT303

Context	AWT	Swing
API Package	The AWT Component classes are provided by the java.awt package.	The Swing component classes are provided by the javax.swing package.
Operating System	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.
Weightiness	The AWT is heavyweight since it uses the resources of the operating system.	The Swing is mostly lightweight since it doesn't need any Operating system object for processing. The Swing Components are built on the top of AWT.
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.
Number of Components	The Java AWT provides a smaller number of components in comparison to Swing.	Java Swing provides a greater number of components than AWT, such as list, scroll panes, tables, color choosers, etc.
Full-Form	Java AWT stands for Abstract Window Toolkit.	Java Swing is mainly referred to as Java Foundation Classes (JFC).
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.
Functionality and Implementation	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.
Memory	Java AWT needs a higher amount of memory for the execution.	Java Swing needs less memory space as compared to Java AWT.
Speed	Java AWT is slower than swing in terms of performance.	Java Swing is faster than the AWT.

Java Swing Hierarchy

Java defines the class hierarchy for all the Swing Components, which is shown in the following image:



Commonly used Methods of Component class

The methods of Component class are widely used in java swing that are given below.

Method	Description
public void add(Component c)	add a component on another component.

Subject & Code: Object Oriented Programming & 20IT303

<code>public void setSize(int width,int height)</code>	sets size of the component.
<code>public void setLayout(LayoutManager m)</code>	sets the layout manager for the component.
<code>public void setVisible(boolean b)</code>	sets the visibility of the component. It is by default false.

Java Swing Examples

There are two ways to create a frame:

- By creating the object of Frame class (association)
- By extending Frame class (inheritance)

We can write the code of swing inside the main(), constructor or any other method.

Java Swing Example

In the following example, we have created a User form by using the swing component classes provided by the javax.swing package. Consider the example.

```
import javax.swing.*;
public class SwingApp {
    SwingApp(){
        JFrame f = new JFrame();

        JLabel firstName = new JLabel("First Name");
        firstName.setBounds(20, 50, 80, 20);

        JLabel lastName = new JLabel("Last Name");
        lastName.setBounds(20, 80, 80, 20);

        JLabel dob = new JLabel("Date of Birth");
        dob.setBounds(20, 110, 80, 20);

        JTextField firstNameTF = new JTextField();
        firstNameTF.setBounds(120, 50, 100, 20);

        JTextField lastNameTF = new JTextField();
        lastNameTF.setBounds(120, 80, 100, 20);
```


Subject & Code: Object Oriented Programming & 20IT303

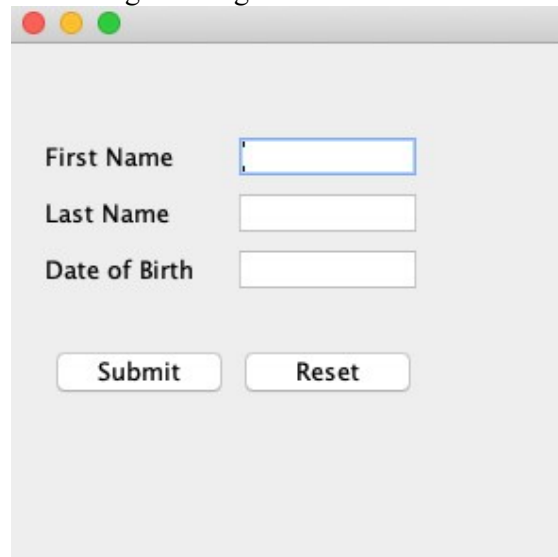
```
JTextField dobTF = new JTextField();  
dobTF.setBounds(120, 110, 100, 20);  
JButton sbmt = new JButton("Submit");  
sbmt.setBounds(20, 160, 100, 30);
```

```
JButton reset = new JButton("Reset");  
reset.setBounds(120,160,100,30);
```

```
f.add(firstName);  
f.add(lastName);  
f.add(dob);  
f.add(firstNameTF);  
f.add(lastNameTF);  
f.add(dobTF);  
f.add(sbmt);  
f.add(reset);
```

```
f.setSize(300,300);  
f.setLayout(null);  
f.setVisible(true);  
}  
public static void main(String[] args) {  
// TODO Auto-generated method stub  
SwingApp s = new SwingApp();  
}  
}
```

Output:



Swing JButton

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.

JButton class declaration

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

1. **public class** JButton **extends** AbstractButton **implements** Accessible

Commonly used Constructors:

Constructor	Description
JButton()	It creates a button with no text and icon.
JButton(String s)	It creates a button with the specified text.
JButton(Icon i)	It creates a button with the specified icon object.

Commonly used Methods of AbstractButton class:

Methods	Description
void setText(String s)	It is used to set specified text on button

Subject & Code: Object Oriented Programming & 20IT303

String getText()	It is used to return the text of the button.
void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	It is used to get the Icon of the button.
void setMnemonic(int a)	It is used to set the mnemonic on the button.
void addActionListener(ActionListener a)	It is used to add the action listener to this object.

Java JButton Example

```
import javax.swing.*;
public class ButtonExample {
public static void main(String[] args) {
    JFrame f=new JFrame("Button Example");
    JButton b=new JButton("Click Here");
    b.setBounds(50,100,95,30);
    f.add(b);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
}
```

Output:



Example of displaying image on the button:

```
import javax.swing.*;
public class ButtonExample {
    ButtonExample() {
        JFrame f=new JFrame("Button Example");
        JButton b=new JButton(new ImageIcon("D:\\icon.png"));
        b.setBounds(100,100,100, 40);
        f.add(b);
        f.setSize(300,400);
        f.setLayout(null);
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] args) {
        new ButtonExample();
    }
}
```

Output:



Swing JComboBox

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 [JComponent](#) class.

JComboBox class declaration

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

1. **public class** JComboBox **extends** JComponent **implements** ItemSelectable, ListDataListener, ActionListener, Accessible

Commonly used Constructors:

Constructor	Description
JComboBox()	Creates a JComboBox with a default data model.
JComboBox(Object[] items)	Creates a JComboBox that contains the elements in the specified array .
JComboBox(Vector<?> items)	Creates a JComboBox that contains the elements in the specified Vector .

Commonly used Methods:

Methods	Description
void addItem(Object anObject)	It is used to add an item to the item list.
void removeItem(Object anObject)	It is used to delete an item to the item list.
void removeAllItems()	It is used to remove all the items from the list.
void setEditable(boolean b)	It is used to determine whether the JComboBox is editable.
void addActionListener(ActionListener a)	It is used to add the ActionListener .
void addItemListener(ItemListener i)	It is used to add the ItemListener .

Swing JComboBox Example

```
import javax.swing.*;
public class ComboBoxExample {
    JFrame f;
    ComboBoxExample(){
        f=new JFrame("ComboBox Example");
        String country[]={"India", "Aus", "U.S.A", "England", "Newzealand"};
        JComboBox cb=new JComboBox(country);
        cb.setBounds(50, 50,90,20);
        f.add(cb);
        f.setLayout(null);
        f.setSize(400,500);
        f.setVisible(true);
    }
    public static void main(String[] args) {
        new ComboBoxExample();
    }
}
```

Output:



Swing JTable

The JTable class is used to display data in tabular form. It is composed of rows and columns.

JTable class declaration

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

Commonly used Constructors:

Constructor	Description
JTable()	Creates a table with empty cells.
JTable(Object[][] rows, Object[] columns)	Creates a table with the specified data.


Swing JTable Example

```
import javax.swing.*;
public class TableExample {
    JFrame f;
    TableExample(){
        f=new JFrame();
        String data[][]={ {"101","Amit","670000"},
                           {"102","Jai","780000"},
                           {"101","Sachin","700000"} };
    }
}
```

Subject & Code: Object Oriented Programming & 20IT303

```
String column[]={"ID","NAME","SALARY"};
JTable jt=new JTable(data,column);
jt.setBounds(30,40,200,300);
JScrollPane sp=new JScrollPane(jt);
f.add(sp);
f.setSize(300,400);
f.setVisible(true);
}
public static void main(String[] args) {
    new TableExample();
}
}
```

Output:



ID	NAME	SALARY
101	Amit	670000
102	Jai	780000
101	Sachin	700000

Swing JScrollBar

The object of JScrollBar class is used to add horizontal and vertical scrollbar. It is an implementation of a scrollbar. It inherits JComponent class.

JScrollBar class declaration

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

```
public class JScrollBar extends JComponent implements Adjustable, Accessible
```

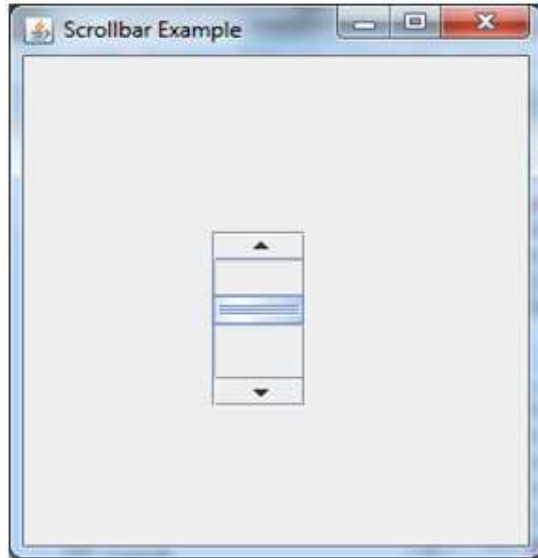
Commonly used Constructors:

Constructor	Description
JScrollBar()	Creates a vertical scrollbar with the initial values.
JScrollBar(int orientation)	Creates a scrollbar with the specified orientation and the initial values.
JScrollBar(int orientation, int value, int extent, int min, int max)	Creates a scrollbar with the specified orientation, value, extent, minimum, and maximum.

Java JScrollBar Example

```
import javax.swing.*;  
class ScrollBarExample  
{  
    ScrollBarExample(){  
        JFrame f= new JFrame("Scrollbar Example");  
        JScrollBar s=new JScrollBar();  
        s.setBounds(100,100, 50,100);  
        f.add(s);  
        f.setSize(400,400);  
        f.setLayout(null);  
        f.setVisible(true);  
    }  
    public static void main(String args[])  
    {  
        new ScrollBarExample();  
    }  
}
```

Output:



Java JMenuBar, JMenu and JMenuItem

The JMenuBar class is used to display menubar on the window or frame. It may have several menus.

The object of JMenu class is a pull down menu component which is displayed from the menu bar. It inherits the JMenuItem class.

The object of JMenuItem class adds a simple labeled menu item. The items used in a menu must belong to the JMenuItem or any of its subclass.

JMenuBar class declaration

```
public class JMenuBar extends JComponent implements MenuElement, Accessible
```

JMenu class declaration

```
public class JMenu extends JMenuItem implements MenuElement, Accessible
```

JMenuItem class declaration

```
public class JMenuItem extends AbstractButton implements Accessible, MenuElement
```

Java JMenuItem and JMenu Example

```
import javax.swing.*;
```

```
class MenuExample
```

```
{
```

```
    JMenu menu, submenu;
```

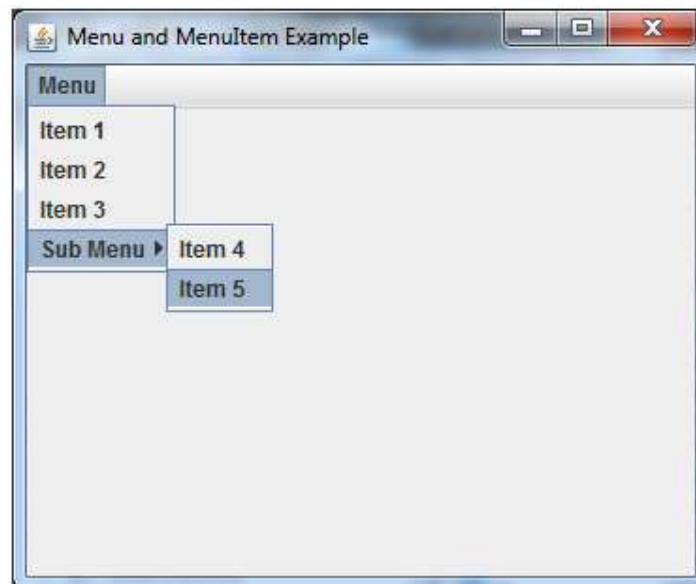
```
    JMenuItem i1, i2, i3, i4, i5;
```

```
    MenuExample(){
```

Subject & Code: Object Oriented Programming & 20IT303

```
JFrame f= new JFrame("Menu and MenuItem Example");
JMenuBar mb=new JMenuBar();
menu=new JMenu("Menu");
submenu=new JMenu("Sub Menu");
i1=new JMenuItem("Item 1");
i2=new JMenuItem("Item 2");
i3=new JMenuItem("Item 3");
i4=new JMenuItem("Item 4");
i5=new JMenuItem("Item 5");
menu.add(i1); menu.add(i2); menu.add(i3);
submenu.add(i4); submenu.add(i5);
menu.add(submenu);
mb.add(menu);
f.setJMenuBar(mb);
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
}
public static void main(String args[])
{
new MenuExample();
}}
```

Output:



Java JTree

The JTree class is used to display the tree structured data or hierarchical data. JTree is a complex component. It has a 'root node' at the top most which is a parent for all nodes in the tree. It inherits JComponent class.

JTree class declaration

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

1. **public class** JTree **extends** JComponent **implements** Scrollable, Accessible

Commonly used Constructors:

Constructor	Description
JTree()	Creates a JTree with a sample model.
JTree(Object[] value)	Creates a JTree with every element of the specified array as the child of a new root node.
JTree(TreeNode root)	Creates a JTree with the specified TreeNode as its root, which displays the root node.

Java JTree Example

```
import javax.swing.*;
import javax.swing.tree.DefaultMutableTreeNode;
public class TreeExample {
    JFrame f;
    TreeExample(){
        f=new JFrame();
        DefaultMutableTreeNode style=new DefaultMutableTreeNode("Style");
        DefaultMutableTreeNode color=new DefaultMutableTreeNode("color");
        DefaultMutableTreeNode font=new DefaultMutableTreeNode("font");
        style.add(color);
        style.add(font);
        DefaultMutableTreeNode red=new DefaultMutableTreeNode("red");
        DefaultMutableTreeNode blue=new DefaultMutableTreeNode("blue");
        DefaultMutableTreeNode black=new DefaultMutableTreeNode("black");
```

Subject & Code: Object Oriented Programming & 20IT303

```
DefaultMutableTreeNode green=new DefaultMutableTreeNode("green");
color.add(red); color.add(blue); color.add(black); color.add(green);
JTree jt=new JTree(style);
f.add(jt);
f.setSize(200,200);
f.setVisible(true);
}
public static void main(String[] args) {
    new TreeExample();
}}
```

Output:

