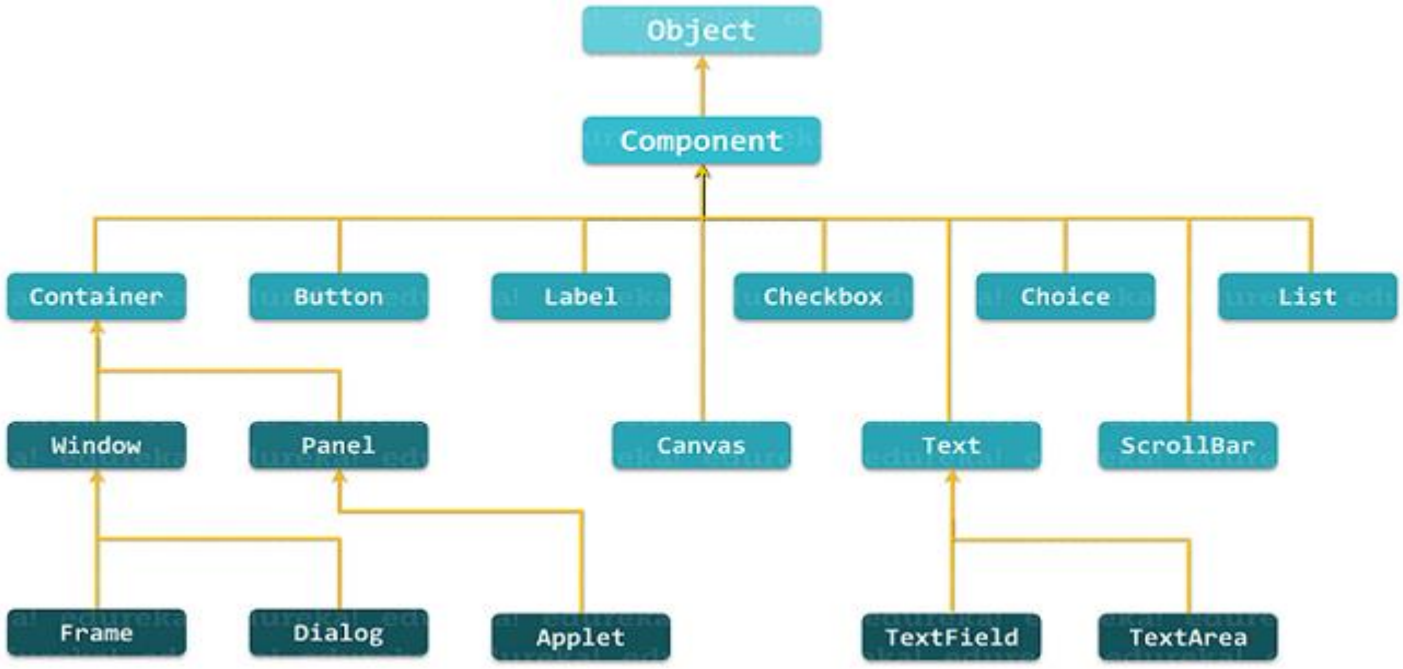
UNIT-V

GUI Programming with AWT

1. AWT stands for Abstract Window Toolkit
2. Java AWT is an API to develop Graphical User Interface (GUI) or windows-based applications in Java.
3. AWT is used to develop Desktop Application.
4. Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system.
5. AWT is heavy weight i.e. its components are using the resources of underlying operating system(OS).
6. The java.awt package provides classes for AWT API such as Text Field, Label, TextArea, RadioButton, Checkbox, Choice, List etc.

HIERARCHY OF AWT



Components

All the elements like the button, text fields, scroll bars, etc. are called components.

Containers:

1. A Container in AWT is a component itself and it adds the capability to add components such as text fields, buttons, etc. to self. It is a subclasses of Container are called as Container.
2. Container can add only Components itself.

Note: A container itself is a component (see the above diagram), therefore we can add a container inside container.

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 :

1. The Frame is the container that contain title bar and border and can have menu bars.
2. Frame is a container to manage some other GUI components like button, text field, scrollbar etc.
3. To represent Frame in GUI applications, JAVA has provided a separate predefined class in the form of java.awt.Frame.
4. Frame is most widely used container while developing an AWT application.

Constructors Frame class

1: public Frame() : It will create a Frame without title.

Example : Frame f = new Frame();

2: public Frame(String title) : It will create a Frame with specified title.

Example : Frame f = new Frame("Frame Class");

Methods of Frame Class

1. **public void setVisible(boolean b) :** This method returns true then it will provide visibleness to the frame, if returns false then it will hide the frame.
2. **public void setSize(int width, int height) :** In GUI Applications, when we create Frame then Frame will created with 0 width and 0 height, where to provide size to the Frame explicitly we have to use setSize() method.
3. **public void setTitle(String title) :** To set a particular title to the Frame explicitly we have to use setTitle() method.
4. **public void setLayout(LayoutManager m) :** Defines the layout manager for the component.

How to create a Frame class :

There are two ways to create a Form.

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

1. By creating the object of Frame class (association)

```
import java.awt.*;
class AwtExample1
{
        AwtExample1()
        {
                Frame f=new Frame();
                f.setVisible(true);
                f.setSize(500,500);
                f.setTitle("User Defined Frame");
                f.setLayout(null);
        }
        public static void main(String args[])
        {
                AwtExample1 a=new AwtExample1();
        }
}
```

2.By extending Frame class (inheritance) : In GUI applications, by creating the object of Frame class (association) is not suggestible to create Frames, because, we are unable to perform customization over the Frame. To implement customization over the Frame, we have to create user defined Frames.

Program :

```
import java.awt.*;
class MyFrame extends Frame
{
        MyFrame()
        {
                this.setVisible(true);
                this.setSize(500,500);
                f.setTitle("User Defined Frame");
                f.setLayout(null);
        }
        public static void main(String args[])
        {
                MyFrame f=new MyFrame();
        }
}
```

AWT COMPONENTS:

1. LABEL CLASS:

1. The object of the Label class is a component for placing text in a container.
2. It is used to display a single line of read only text.
3. The text can be changed by a programmer but a user cannot edit it directly.

AWT Label Fields (or) Constants in Label class

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

Label class Constructors

1. **public Label() :** It constructs an empty label.

   Syntax : Lable l1=new Label();
2. **public Label(String label) :** It constructs a label with the given string (left justified by default).

   Syntax : Lable l1=new Label("User Name");
3. **public Label(String label, int allignment) :** It constructs a label with the specified string and the specified alignment.

   Syntax : Lable l1=new Label("User Name",Label.CENTER);

Program:

```
import java.awt.*;

class LabelDemo extends Frame
{
  LabelDemo()
  {
          Label  l1=new Label("User Name");
          Label  l2=new Label("Password");
          l1.setBounds(50,50,100,30);
          l2.setBounds(50,100,100,30);
          this.add(l1);
          this.add(l2);
          this.setSize(300,300);
          this.setTitle("Label Class");
          this.setVisible(true);
  }
  public static void main(String args[])
  {
          LabelDemo ld=new LabelDemo ();
  }
}
```

2. TextField class:

1. The object of a TextField class is a text component that allows a user to enter a single line text and edit it.
2. To create a TextField, we need to create the object of TextField class.

Text Field Class constructors

1. public TextField() : It constructs a new text field component.

Example : TextField t1 = new TextField();

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

Example : TextField t1 = new TextField("Enter User Name");

3. public TextField(int columns) : It constructs a new text field (empty) with given number of columns.

Example : TextField t1 = new TextField(30);

Program :

```
import java.awt.*;
class TextFieldDemo extends Frame
{
        TextFieldDemo()
        {
                TextField  tf1 = new TextField("Student Name :");
                TextField  tf2 = new TextField("Student RollNo :");
                TextField  tf3 = new TextField("Student Age :");
                this.add(tf1);
                this.add(tf2);
                this.add(tf3);
                this.setSize(340,260);
                this.setLayout(new FlowLayout());
                this.setTitle("Text Field class");
                this.setVisible(true);
        }
        public static void main(String args[])
        {
                TextFieldDemo  tfd=new TextFieldDemo();
        }
}
```

3. BUTTON CLASS

1. The **Button** class is used to create a labeled button.
2. The application result in some action when the button is pushed.
3. When we press a button and release it, AWT sends an instance of **ActionEvent** to that button by calling **processEvent** on the button.
4. To create a Button, we need to create the object of Button class.

Button Class Constructors

1. **public Button( ) :** It constructs a new button with an empty string i.e. it has no label.

   Example : Button b = new Button();

2. **public Button (String text) :** It constructs a new button with given string as its label.

   Example : Button b = new Button("Login");

Program :

```
import java.awt.*;
class ButtonDemo extends Frame
{
        ButtonDemo()
        {
                this.setVisible(true);
                this.setTitle("Button Demo");
                this.setSize(500,500);
                this.setLayout(null);
                Button b=new Button();
                b.setLabel("Submit");
                b.setBounds(100,100,150,40);
                this.add(b);
                Button b1=new Button();
                b1.setLabel("Cancel");
                b1.setBounds(100,200,150,40);
                this.add(b1);
                System.out.println(b.getLabel());
        }
        public static void main(String args[])
        {
                ButtonDemo bd=new ButtonDemo();
        }
}
```

4. TEXTAREA CLASS

1. TextArea class is used to create a multi-line text-area, allowing a user to enter the text in multiple lines.
2. When the text in the text area becomes larger than the viewable area, the scroll bar appears automatically which helps us to scroll the text up and down, or right and left.

Class constructors:

1. **TextArea() :** It constructs a new and empty text area with no text in it.
2. **TextArea (int row, int column) :** It constructs a new text area with specified number of rows and columns and empty string as text.
3. **TextArea (String text) :** It constructs a new text area and displays the specified text in it.
4. **TextArea (String text, int row, int column) :** It constructs a new text area with the specified text in the text area and specified number of rows and columns.
5. **TextArea (String text, int row, int column, int scrollbars) :** It construcst a new text area with specified text in text area and specified number of rows and columns and visibility.

Program :

```
import java.awt.*;
class TextAreaDemo extends Frame
{
        TextAreaDemo ()
        {
                this.setVisible(true);
                this.setTitle("TextAreaDemo ");
                this.setSize(500,500);
                this.setLayout(null);
                Label l1=new Label(“Enter Address :”,Label.CENTER);
                TextArea ta = new TextArea(“Welcome to CSE Department”);
                this.add(l1);
                this.add(ta);
        }
        public static void main(String args[])
        {
                TextAreaDemo tad=new TextAreaDemo ();
        }
}
```

5. LIST CLASS

1. The List is a GUI component used to display a list of items.
2. It contains a set of text items that the user can choose from.
3. It is a list that allows the user to select one or more options.
4. The programmer has the choice to configure ‘Single select’ or ‘Multiple select’ option of a list.

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.

3. List(int row_num, Boolean multipleMode) : It constructs a new scrolling list initialized which displays the given number of rows.

Program :

```
import java.awt.*;
class CreateList extends Frame
{
        CreateList ()
        {
                List list1 = new List(5, true);
                List list2 = new List(2);
                list1.add("Apple");
                list1.add("Banana");
                list1.add("Mango");
                list2.add("C++");
                list2.add("Java");
                list2.add("Ruby");
                list2.add("Javascript");
                list2.add("Python");
                list1.setBounds(50, 50, 100, 60);
                list2.setBounds(50, 200, 100, 60);
                this.setLayout(null);
                this.setVisible(true);
                this.setSize(300, 300);
                this.add(list1);
                this.add(list2);
        }
        public static void main(String[] args)
        {
                CreateList obj = new CreateList();
        }
}
```

Note: In the object constructor, the first parameter specifies the row size, and the second parameter is a boolean value that specifies whether the user can select single or multiple values. If the second parameter is false, then the user can select only one item from the list, as in the case of list 2. If the parameter is true, then the user can select multiple items from the list, as in the case of list 1.

6. CHECKBOX CLASS

1. The Checkbox class is used to create a checkbox.
2. 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.

   Example : CheckBox cb=new CheckBox();
2. **Checkbox(String label) :** It constructs a checkbox with the given label.

   Example : CheckBox cb=new CheckBox("Coffee");
3. **Checkbox(String label, boolean state):** It constructs a checkbox with the given label and sets the given state. (State means- it can be either true or false, true indicates checkbox checked, otherwise false.)

   Example : CheckBox cb=new CheckBox("Coffee",true);

Checkbox Class Methods

1. **void setLabel(String label):** It sets the checkbox's label to the string argument.
2. **String getLabel()** : It fetched the label of checkbox.
3. **void setState(boolean state) :** It sets the state of checkbox to the specified state.
4. **boolean getState() :** It returns true if the checkbox is on, else returns off.

Program :

```
import java.awt.*;
class CheckBoxDemo extends Frame
{
        CheckBoxDemo ()
        {
                this.setVisible(true);
                this.setTitle("CheckBoxDemo ");
                this.setSize(500,500);
                this.setLayout(null);
                Checkbox  cb1=new Checkbox();
                cb1.setLabel("JBREC");
                cb1.setBounds(100, 100,  50, 50);
                Checkbox  cb2=new Checkbox("JBIET");
                cb2.setBounds(100, 150,  50, 50);
                Checkbox  cb3=new Checkbox("VJIT",true);
                cb3.setBounds(100, 200,  50, 50);
                Checkbox  cb4=new Checkbox("KG REDDY");
                cb4.setBounds(100, 250,  50, 50);
                Checkbox  cb5=new Checkbox("GLOBAl");
                cb5.setBounds(100, 300,  50, 50);
                this.add(cb1);
                this.add(cb2);
                this.add(cb3);
                this.add(cb4);
                this.add(cb5);
        }
        public static void main(String args[])
        {
                CheckBoxDemo tad=new CheckBoxDemo();
        }
}
```

7. CHOICE CLASS

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

Choice Class Constructors

Choice() : It constructs a new choice menu.

Example : Choice ch1=new Choice();

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 getSelectedIndex()** : Returns the index of the currently selected item.
5. **String getSelectedItem() :** Gets a representation of the current choice as a string.

Program :

```
import java.awt.*;
class ChoiceDemo extends Frame
{
        ChoiceDemo ()
        {
                this.setVisible(true);
                this.setTitle("ChoiceDemo ");
                this.setSize(500,500);
                this.setLayout(null);
                Choice  ch1=new Choice();
                ch1.add("JAVA");
                ch1.setBounds(100, 100,  50, 50);
                ch1.add("DBMS");
                ch1.setBounds(100, 150,  50, 50);
                ch1.add("DM");
                ch1.setBounds(100, 200,  50, 50);
                ch1.add("COA");
                ch1.setBounds(100, 250,  50, 50);
                ch1.add("COI");
                ch1.setBounds(100, 300,  50, 50);
                this.add(ch1);
                this.add(ch1);
                this.add(ch1);
                this.add(ch1);
                this.add(ch1);
        }
        public static void main(String args[])
        {
                ChoiceDemo tad=new ChoiceDemo ();
        }
}
```

8. SCROLLBAR CLASS

The object of Scrollbar class is used to add horizontal and vertical scrollbar. Scrollbar is a GUI component allows us to see invisible number of rows and columns.

Scrollbar Class Fields

1. **static int HORIZONTAL -** It is a constant to indicate a horizontal scroll bar.
2. **static int VERTICAL -** It is a constant to indicate a vertical scroll bar.

Scrollbar Class Constructors

1. **Scrollbar() :** Constructs a new vertical scroll bar.
2. **Scrollbar(int orientation) :** Constructs a new scroll bar with the specified orientation.

Program :

```
import java.awt.*;
class ScrollbarExample extends Frame
{
  ScrollbarExample()
  {
        Scrollbar s = new Scrollbar();
        s.setBounds (100, 100, 50, 100);
        this.add(s);
        this.setSize(400, 400);
```

```
            this.setTitle("Scrollbar Example");
            this.setLayout(null);
            this.setVisible(true);
  }
  public static void main(String args[])
  {
                  ScrollbarExample sb=new ScrollbarExample();
  }
}
```

9. CANVAS CLASS

The Canvas class controls and represents a blank rectangular area where the application can draw or trap input events from the user.

```
import java.awt.*;
import java.awt.event.*;
class canvasExample extends Frame
{
  canvasExample()
  {
            Canvas c = new Canvas();
            c.setBackground(Color.red);
            this.add(c);
            this.setSize(100, 100);
            this.setTitle("Scrollbar Example");
            this.setLayout(null);
            this.setVisible(true);
  }

  public static void main(String args[])
  {
                  canvasExample sb=new canvasExample();
  }
}
```

LAYOUT MANAGERS:

The LayoutManagers are used to arrange components in a particular manner. LayoutManager is an interface that is implemented by all the classes of layout managers.

The following classes represent the layout managers in java

1. Flow Layout
2. Border Layout
3. Grid Layout
4. Card Layout
5. Grid Bag Layout.

1.FLOWLAYOUT CLASS

1. Flow Layout class is used to arrange the components in a line, one after another (in a flow).
2. When a line is filled with components, they are automatically placed in the next line.
3. This is the default layout of the applet or panel.
4. All rows in Flow Layout are center aligned default.
5. The default horizontal and vertical gap between components is 5 pixels.

Fields of FlowLayout class

1. **public static final int LEFT :** each row of components should be left-justified.
2. **public static final int RIGHT :** each row of components should be right-justified.
3. **public static final int CENTER :** each row of components should be centered.
4. **public static final int LEADING :** each row of components should be justified to the left in left to right orientation.
5. **public static final int TRAILING :** each row of components should be justified to the right in left to left orientation.

Constructors of FlowLayout class

1. **FlowLayout():** creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.

   Example : FlowLayout f1 = new FlowLayout();
2. **FlowLayout(int align):** creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.

   Example : FlowLayout f1 = new FlowLayout(FlowLayout.RIGHT);
3. **FlowLayout(int align, int hgap, int vgap):** creates a flow layout with the given alignment and the given horizontal and vertical gap.

   Example : FlowLayout f1 = new FlowLayout(FlowLayout.LEFT,15,15);

```
import java.awt.*;
class FlowLayoutExample extends Frame
{
        FlowLayoutExample()
        {
                this.setVisible(true);
                this.setTitle("FlowLayout Class");
                this.setSize(500,500);
                this.setBackground(new Color(0x009270));
                //FlowLayout f1=new FlowLayout();
                FlowLayout f1=new FlowLayout(FlowLayout.LEFT, 20, 25);
                this.setLayout(f1);
                Button b1 = new Button("Button-1");
                Button b2 = new Button("Button-2");
                Button b3 = new Button("Button-3");
                Button b4 = new Button("Button-4");
                Button b5 = new Button("Button-5");
                Button b6 = new Button("Button-6");
                Button b7 = new Button("Button-7");
                Button b8 = new Button("Button-8");
                Button b9 = new Button("Button-9");
                this.add(b1);
                this.add(b2);
                this.add(b3);
                this.add(b4);
                this.add(b5);
                this.add(b6);
                this.add(b7);
                this.add(b8);
                this.add(b9);
        }
        public static void main(String args[])
        {
                FlowLayoutExample fl=new FlowLayoutExample();
        }
}
```

2.BORDER LAYOUT:

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

The BorderLayout provides five constants for each region:

1. **public static final int NORTH** : It arranges that GUI component to fit in the north region.
2. **public static final int SOUTH**: It arranges that GUI component to fit in the south region.
3. **public static final int EAST**: It arranges that GUI component to fit in the east region.
4. **public static final int WEST**: It arranges that GUI component to fit in the west region.
5. **public static final int CENTER**: It arranges that GUI component to fit in the center region.

Constructors of BorderLayout class:

1. **BorderLayout():** creates a border layout but with no gaps between the components.

   Example : BorderLayout f1 = new BorderLayout();

2. **BorderLayout(int hgap, int vgap):** creates a border layout with the given horizontal and vertical gaps between the components.

   Example : BorderLayout f1 = new BorderLayout(20,15);

Program :

```
import java.awt.*;
class BorderLayoutExample extends Frame
{
        BorderLayoutExample()
        {
                this.setVisible(true);
                this.setTitle("BorderLayout Class");
                this.setSize(500,500);
                this.setBackground(new Color(0x009270));
                BorderLayout f1=new BorderLayout(20, 25);
                this.setLayout(f1);
                Button b1 = new Button("NORTH");
                Button b2 = new Button("SOUTH");
                Button b3 = new Button("EAST");
                Button b4 = new Button("WEST");
                Button b5 = new Button("CENTER");
                this.add(b1,BorderLayout.NORTH);
                this.add(b2, BorderLayout.SOUTH);
                this.add(b3, BorderLayout.EAST);
                this.add(b4, BorderLayout.WEST);
                this.add(b5, BorderLayout.CENTER);
        }
        public static void main(String args[])
        {
                BorderLayoutExample bl=new BorderLayoutExample();
        }
}
```

3.GRIDLAYOUT CLASS :

1. Grid Layout class is used to arrange the components in a rectangular grid or table format.
2. One component is displayed in each rectangle.
3. Grid Layout class is present in java.awt package.

Class Declaration :

Constructors of GridLayout class

1. **GridLayout():** creates a grid layout with one column per component in a row.

   Example : GridLayout g1 = new GridLayout();

2. **GridLayout(int rows, int columns):** creates a grid layout with the given rows and columns but no gaps between the components.

   Example : GridLayout g1 = new GridLayout(3,3);

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 : GridLayout g1 = new GridLayout(3,3,20,15);

Program :

```
import java.awt.*;
class GridLayoutExample extends Frame
{
        GridLayoutExample()
        {
                this.setVisible(true);
                this.setTitle("GridLayout Class");
                this.setSize(500,500);
                this.setBackground(Color.red);
                GridLayout g1=new GridLayout(3,3);
                this.setLayout(g1);
                Button b1 = new Button("Button-1");
                Button b2 = new Button("Button-2");
                Button b3 = new Button("Button-3");
                Button b4 = new Button("Button-4");
                Button b5 = new Button("Button-5");
                Button b6 = new Button("Button-6");
                Button b7 = new Button("Button-7");
                Button b8 = new Button("Button-8");
                Button b9 = new Button("Button-9");
                this.add(b1);
                this.add(b2);
                this.add(b3);
                this.add(b4);
                this.add(b5);
                this.add(b6);
                this.add(b7);
                this.add(b8);
                this.add(b9);
        }
        public static void main(String args[])
        {
                GridLayoutExample gl=new GridLayoutExample();
        }
}
```

4.CARDLAYOUT CLASS

1. CardLayout class manages the components in such a manner that only one component is visible at a time.
2. It treats each component as a card that is why it is known as CardLayout.
3. Only one card is visible at a time and the container acts as a stack of cards.
4. The first component added to a CardLayout object is the visible component, when the container is first displayed.

Constructors of CardLayout Class

1. **CardLayout():** creates a card layout with zero horizontal and vertical gap.

   Example : CardLayout g1 = new CardLayout();

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

   Example : CardLayout g1 = new CardLayout(20,15);

Methods of CardLayout class

1. **public void next(container parent) :** This method is used to the next card of the given container.
2. **public void previous(container parent) :** This method is used to the previous card of the given container.
3. **public void first(container parent) :** This method is used to the first card of the given container.
4. **public void last(container parent) :** This method is used to the last card of the given container.
5. **public void show(container parent,String name) :** This method is used to the specified card with the given name.

Program :

```
import java.awt.*;
import java.awt.event.*;
class CardLayoutExample extends Frame implements ActionListener
{
        CardLayout c1;
        CardLayoutExample()
        {
                this.setVisible(true);
                this.setTitle("CardLayout Class");
                this.setSize(500,500);
                this.setBackground(new Color(0x009270));
                c1=new CardLayout(50,10);
                this.setLayout(c1);
                Button b1 = new Button("Button-1");
                Button b2 = new Button("Button-2");
                Button b3 = new Button("Button-3");
                Button b4 = new Button("Button-4");
                this.add(b1);
                this.add(b2);
                this.add(b3);
                this.add(b4);
                b1.setBackground(new Color(0xf72585));
                b2.setBackground(new Color(0x1e96fc));
                b3.setBackground(new Color(0xfb8500));
                b4.setBackground(new Color(0xf72585));
                b1.addActionListener(this);
                b2.addActionListener(this);
                b3.addActionListener(this);
                b4.addActionListener(this);
        }
        public void actionPerformed(ActionEvent e)
        {
                c1.next(this);
        }
        public static void main(String args[])
        {
                CardLayoutExample gl=new CardLayoutExample();
        }
}
```

GRID BAG LAYOUT CLASS:

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

Example

```
import java.awt.Button;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import javax.swing.*;
public class GridBagLayoutExample extends JFrame
{
        public GridBagLayoutExample()
        {
                GridBagLayoutgrid = new GridBagLayout();
                GridBagConstraints gbc = new GridBagConstraints();
                setLayout(grid);
                setTitle("GridBag Layout Example");
                GridBagLayout layout = new GridBagLayout();
                this.setLayout(layout);
                gbc.fill = GridBagConstraints.HORIZONTAL;
                gbc.gridx = 0;
                gbc.gridy = 0;
                this.add(new Button("Button One"), gbc);
                gbc.gridx = 1;
                gbc.gridy = 0;
                this.add(new Button("Button two"), gbc);
                gbc.fill = GridBagConstraints.HORIZONTAL;
                gbc.ipady = 20;
                gbc.gridx = 0;
                gbc.gridy = 1;
                this.add(new Button("Button Three"), gbc);
                gbc.gridx = 1;
                gbc.gridy = 1;
                this.add(new Button("Button Four"), gbc);
                gbc.gridx = 0;
                gbc.gridy = 2;
                gbc.fill = GridBagConstraints.HORIZONTAL;
                gbc.gridwidth = 2;
                this.add(new Button("Button Five"), gbc);
                setSize(300, 300);
                setPreferredSize(getSize());
                setVisible(true);
                setDefaultCloseOperation(EXIT_ON_CLOSE);
        }
        public static void main(String[] args)
        {
                GridBagLayoutExample a = new GridBagLayoutExample();
        }

}
```

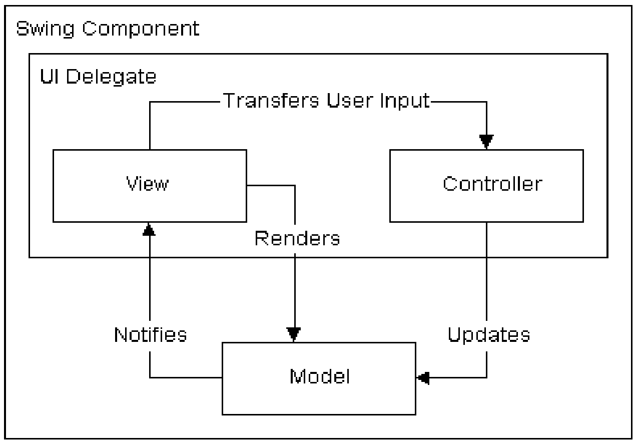
MVC Architecture

The MVC design pattern consists of three modules model, view and controller.

Model: The model represents the state (data) and business logic of the application. For example-in case of a check box, the model contains a field which indicates whether the box is checked or unchecked.

View: The view module is responsible to display data i.e. it represents the presentation.The view determines how a component has displayed on the screen, including any aspects of view that are affected by the current state of the model.

Controller: The controller determines how the component will react to the user.Controller The controller module acts as an interface between view and model. It intercepts all the requests i.e. receives input and commands to Model / View to change accordingly.



Advantage of MVC Architecture

1. Navigation control is centralized now only controller contains the logic to determine the next page.
2. Easy to maintain
3. Easy to extend
4. Easy to test
5. Better separation of concerns

Disadvantage of MVC Architecture

We need to write the controller code self. If we change the controller code, we need to recompile the class and redeploy the application.

Event Handling (or) The Delegation event model

In GUI applications, we may use the GUI components like Buttons, Checkboxes, Radio Buttons,... When we click on button or select item in checkbox , the respective GUI components are able to raise the respective events, here GUI components are not capable to handle the generated events, in this context to handle the generated events , we have to use another component internally that is Listener.

In GUI applications, Listeners will take events from GUI components, handle them, perform the required actions by executing Listener methods and return the result back to GUI application. Here in the process delegating events from GUI components to Listener in order to handle is called as **"Event Delegation Model" or Event Handling**.

To implement Event handling in GUI applications, java has provided predefined package is **"java.awt.event".**

Basically, an Event Model is based on the following three components:

- Events
- Events Sources
- Events Listeners

EVENTS:

1. An event is an object that describes a state change in a source or behavior by performing actions is referred to as an Event.
2. Events are generated as result of user interaction with the graphical user interface components.
3. Java provides a package **java.awt.event** that contains several event classes.

Types of Event

- Foreground Events
- Background Events

1. Foreground Events

Foregrounds events are the events that require user interaction to generate, i.e., foreground events are generated due to interaction by the user on components in Graphic User Interface (**GUI**). Interactions are nothing but clicking a button, entering a character via the keyboard, selecting an item in a list and clicking the mouse.

2. Background Events

Events may also occur that are not directly caused by interactions with a user interface are known as background events.

For Example, an event may be generated when a timer expires, a counter exceed a value, a software or hardware failure occurs or an operation is completed.

EVENT SOURCES: Events are generated from the source. There are various sources like buttons, checkboxes, list, menu-item, choice, scrollbar, text components, windows, etc., to generate events.
A source must register a listener to receive notifications for a specific event. Each event contains its registration method.

Syntax : public void addXXXListener(XXXListener e)

Example : rb.addActionListener(ActionListener e)

EVENT LISTENERS

1. The delegation event model has two parts: sources and listeners.
2. Sources are GUI components and Listeners are interfaces that handle the events generated by the components.
3. All the sources and listeners are defined in **java.awt.event** package.
4. When an event occurs, the source invokes an appropriate mehod defined by the listener.
5. The method takes an event object as its argument (like ActionEvent).

The following are the commonly used Listener Interfaces

1. ActionListener interface
2. Adjustment Listener interface
3. FocusListener interface
4. ItemListener interface
5. KeyListener interface
6. MouseListener interface
7. MouseMotionListener interface
8. TextListener interface
9. WindowListener interface

1. ActionListener interface : This interface defines only one method.

Syntax : void actionPerformed(ActionEvent ae)

The method actionPerformed ( ) that is invoked when an action event occurs

2. Adjustment Listener interface : This interface defines only one method.

Syntax: void adjustmentValueChanged (AdjustmentEvent e)

The method adjustmentValueChanged ( ) that is invoked when an adjustment event occurs

3. FocusListener interface: This interface has two methods.

1.void focusGained(FocusEvent e) : It is invoked when the component obtains keyboard focus.

2.**void focusLost(FocusEvent e) :** It is invoked when the component losses the keyboard focus.

4. ItemListener interface : This interface defines only one method.

Syntax : void itemStateChanged(ItemEvent e)

This method is invoked when the state of the item changes.

5. KeyListener interface : This interface has three methods.

1. public abstract void keyPressed (KeyEvent e) : It is invoked when a key has been pressed.

2. public abstract void keyReleased (KeyEvent e) : It is invoked when a key has been released.

3. public abstract void keyTyped (KeyEvent e) : It is invoked when a key has been typed.

6. MouseListener interface : This interface has five methods.

1. public abstract void mouseClicked(MouseEvent e) : When the mouse button has been clicked (pressed and released) on a component.

2. public abstract void mouseEntered(MouseEvent e): When the mouse enters a component.

3. public abstract void mouseExited(MouseEvent e) : When the mouse exits a component.

4. public abstract void mousePressed(MouseEvent e) : When the mouse button has been pressed on a component.

5. public abstract void mouseReleased(MouseEvent e) : When the mouse button has been released on a component.

7. MouseMotionListener interface : This interface has two methods.

1. void mouseMoved(MouseEvent e) : This method is invoked when the mouse is moved from one place to another

2**. void mouseDragged(MouseEvent e)** : This method is invoked when the mouse is dragged.

8. TextListener interface : This interface defines only one method.

Syntax : void textChanged(TextEvent e)

This method is invoked whenever there is a change in text field or text area.

9. WindowListener interface : This interface has seven methods.

1.void windowActivated(WindowEvent e) : This method is invoked when the window is activated.

2.void windowDeactivated(WindowEvent e) : It is invoked when the window is deactivated.

3.void windowOpened(WindowEvent e) : This method is invoked when the window is opened.

4.void windowClosed(WindowEvent e) : This method is invoked when the window is closed.

5.void windowClosing (WindowEvent e) : This method is invoked when the window is being closed.

EVENT CLASSES:

Event classes are the classes that represent events at the core of java's event handling mechanism.

At the root of the Java event class hierarchy is EventObject, which is in java.util. It is the superclass for all events. Its one constructor is

Syntax: EventObject(Object *src*)

EventObject defines two methods:

1. **Object getSource() :** This method returns the source of the event.
2. **toString() :** This method returns the string equivalent of the event.

Commonly Used Event Classes in java.awt.event package

1. ActionEvent Class
2. AdjustmentEvent Class
3. ComponentEvent Class
4. ContainerEvent Class
5. FocusEvent Class
6. InputEvent Class
7. ItemEvent Class
8. KeyEvent Class
9. MouseEvent Class
10. MouseWheelEvent Class
11. TextEvent Class
12. WindowEvent Class

1. ActionEvent Class : An ActionEvent class is generated when a button is pressed, a list item is double-clicked, or a menu item is selected. The ActionEvent class defines four integer constants that can be used to identify any modifiers associated with an action event : ALT_MASK, CTRL_MASK, META_MASK, and SHIFT_MASK. In addition, there is an integer constant, ACTION_PERFORMED, which can be used to identify action events.

ActionEvent has these three constructors:

1. ActionEvent(Object *src*, int *type*, String *cmd*)
2. ActionEvent(Object *src*, int *type*, String *cmd*, int *modifiers*)
3. ActionEvent(Object *src*, int *type*, String *cmd*, long *when*, int *modifiers*)

ActionEvent Class Methods :

1. **String getActionCommand() :** This method returns the command string associated with this action.
2. **int getModifiers() :** This method returns a value that indicates which modifier keys (alt, ctrl, meta, and/or shift) were pressed when the event was generated.
3. **long getWhen( ) :** This method returns the time at which the event took place. This is called the event's timestamp.

2. AdjustmentEvent Class : An AdjustmentEvent class is generated by a scroll bar. There are five types of adjustment events. The AdjustmentEvent class defines integer constants that can be used to identify them. The constants and their meanings are shown here:

BLOCK_DECREMENT	The user clicked inside the scroll bar to decrease its value.
BLOCK_INCREMENT	The user clicked inside the scroll bar to increase its value.
TRACK	The slider was dragged.
UINT_DECREMENT	The button at the end of scroll bar was clicked to decrease its value.
UNIT_INCREMENT	The button at the end of scroll bar was clicked to increase its value.

In addition there is an integer constant, ADJUSTMENT_VALUE_CHANGED that indicates that a change has occurred.

AdjustmentEvent constructor:

Syntax: AdjustmentEvent(Adjustable *src*, int *id*, int *type*, int *val*)

Adjustment Event Class Methods:

1. **Adjustable getAdjustable( ) :** This method returns object that generated the event.
2. **int getAdjustmentType() :** This method returns the type of adjustment event.
3. **long getValue() :** This method returns the amount of adjustment.

3.ComponentEvent Class : A **ComponentEvent class** is generated when the size, position, or visibility of a component is changed. There are four types of component events. The **ComponentEvent** class defines integer constants that can be used to identify them. The constants and their meanings are shown here:

1. COMPONENT_HIDDEN : The component was hidden.
2. **COMPONENT_MOVED :** The component was moved.
3. **COMPONENT_RESIZED :** The component was resized.
4. **COMPONENT_SHOWN :** The component became visible.

ComponentEvent constructor:

Syntax : ComponentEvent(Component *src*, int *type*)

ComponentEvent Class Methods :

Component getComponent( ) : This method returns the component that generated the event.

4. ContainerEvent Class A ContainerEvent is generated when a component is added to or removed from a container. There are two types of container events. The ContainerEvent class defines **int** constants that can be used to identify them: COMPONENT_ADDED and COMPONENT_REMOVED. They indicate that a component has been added to or removed from the container.

ContainerEvent Class Constructor

Sytnax : ContainerEvent(Component *src*, int *type*, Component *comp*)

ContainerEvent Class Methods

1. **Container getContainer( ) :** It returns a reference to the container that generated the event.
2. **Component getChild() :** This method returns a reference to the component that was added to or removed from the container.

5.InputEvent Class : The abstract class InputEvent is a subclass of ComponentEvent and is the superclass for component input events. Its subclasses are KeyEvent and MouseEvent.

InputEvent defines several integer constants that represent any modifiers, such as the control key being pressed, that might be associated with the event. Originally, the InputEvent class defined the following eight values to represent the modifiers:

ALT_MASK , BUTTON2_MASK , META_MASK ,ALT_GRAPH_MASK, BUTTON3_MASK , SHIFT_MASK

BUTTON1_MASK ,CTRL_MASK

However, because of possible conflicts between the modifiers used by keyboard events and mouse events, and other issues, the following extended modifier values were added:
ALT_DOWN_MASK, ALT_GRAPH_DOWN_MASK, BUTTON1_DOWN_MASK, BUTTON2_DOWN_MASK

BUTTON3_DOWN_MASK, CTRL_DOWN_MASK, META_DOWN_MASK, SHIFT_DOWN_MASK

InputEvent Class Methods :

1. **boolean isAltDown() :**
2. **boolean isAltGraphDown() :**
3. **boolean isControlDown() :**
4. **boolean isMetaDown():**
5. **boolean isShiftDown():**
6. **int getModifiers( ) :** It is used to obtain a value that contains all of the original modifier flags.
7. **int getModifiersEx( ):**It is used to obtain the extended modifiers.

6.ItemEvent Class : An ItemEvent is generated when a check box or a list item is clicked or when a checkable menu item is selected or deselected. There are two types of item events, which are identified by the following integer constants:

DESELECTED : The user deselected an item.

SELECTED : The user selected an item.

In addition, ItemEvent defines one integer constant, ITEM_STATE_CHANGED, that signifies a change of state.

ItemEvent Class constructor:

Syntax: ItemEvent(ItemSelectable *src*, int *type*, Object *entry*, int *state*)

ItemEvent Class Methods:

1. **Object getItem( ):** This method returns a reference to the item that generated an event.
2. **ItemSelectable getItemSelectable( ):** This method returns a reference to the ItemSelectable object that generated an event.
3. **int getStateChange( ) :** This method returns the state change(i.e, SELECTED or DESELECTED) for an event.

7. KeyEvent Class : A KeyEvent is generated when keyboard input occurs. There are three types of key events, which are identified by these integer constants:KEY_PRESSED, KEY_RELEASED, and KEY_TYPED. The first two events are generated when any key is pressed or released. The last event occurs only when a character is generated.

There are many other integer constants that are defined by **KeyEvent**. For example, **VK_0** through **VK_9** and **VK_A** through **VK_Z** define the ASCII equivalents of the numbers and letters. Here are some others:

VK_ALT, VK_CANCEL, VK_CONTROL, VK_DOWN, VK_ENTER, VK_ESCAPE, VK_LEFT, VK_PAGE_DOWN
VK_PAGE_UP, VK_RIGHT, VK_SHIFT, VK_UP

The **VK** constants specify *virtual key codes* and are independent of any modifiers, such as control, shift, or alt.

KeyEvent Class Constructor:

Syntax : KeyEvent (Component src, int type, long when, int modifiers, int code, char ch)

KeyEvent Class Methods :

1. **char getKeyChar () :** This method returns the character that was entered
2. **int getKeyCode () :** This method returns the key code.

8. MouseEvent Class : Generated when the mouse is dragged, moved, clicked, pressed, or released; also generated when the mouse enters or exits a component.

There are eight types of mouse events. The **MouseEvent** class defines the following integer constants that can be used to identify them:

MOUSE_CLICKED : The user clicked the mouse.

MOUSE_DRAGGED : The user dragged the mouse.

MOUSE_ENTERED : The mouse entered a component.

MOUSE_EXITED : The mouse exited from a component.

MOUSE_MOVED : The mouse moved.

MOUSE_PRESSED : The mouse was pressed.

MOUSE_RELEASED : The mouse was released.

MOUSE_WHEEL : The mouse wheel was moved.

MouseEvent Constructor :

MouseEvent(Component *src*, int *type*, long *when*, int *modifiers*, int *x*, int *y*, int *clicks*, boolean *triggersPopup*)

MouseEvent Methods:

int getClickCount( ) : Gets the number of mouse clicks.

boolean isPopupTrigger( ): This method returns whether this mouse event is the popup menu trigger event or not.

Point getLocationOnScreen( ): This method returns the absolute x, y position of the event.

int getXOnScreen( ): This method returns the absolute horizontal x position of the event.

int getYOnScreen( ): This method returns the absolute vertical y position of the event.

9.MouseWheelEvent Class : The MouseWheelEvent class encapsulates a mouse wheel event. It is a subclass of MouseEvent. Not all mice have wheels. If a mouse has a wheel, it is typically located between the left and right buttons. Mouse wheels are used for scrolling. MouseWheelEvent defines these two integer constants:

WHEEL_BLOCK_SCROLL : A page-up or page-down scroll event occurred.

WHEEL_UNIT_SCROLL : A line-up or line-down scroll event occurred.

MouseWheelEvent uses the constructor:

MouseWheelEvent (Component src, int type, long when, int modifier, int x,int y, int clicks, boolean triggersPopup, int scrollHow, int amont, int count)

MouseWheelEvent Methods :

int getWheelRotation() : This method returns number of rotational units

int getScrollType () : This method returns either **WHEEL_UNIT_SCROLL** or **WHEEL_BLOCK_SCROLL**.

10. TextEvent Class : Instances of this class describe text events. These are generated by text fields and text areas when characters are entered by a user or program. TextEvent defines the integer constant TEXT_VALUE_CHANGED.

11. WindowEvent Class : There are ten types of window events. The WindowEvent class defines integer constants that can be used to identify them. The constants and their meanings are shown here:

WINDOW_ACTIVATED : The window was activated.

WINDOW_CLOSED : The window has been closed.

WINDOW_CLOSING : The user requested that the window be closed.

WINDOW_DEACTIVATED : The window was deactivated.

WINDOW_DEICONIFIED : The window was deiconified.

WINDOW_GAINED_FOCUS : The window gained input focus.

WINDOW_ICONIFIED : The window was iconified.

WINDOW_LOST_FOCUS : The window lost input focus.

WINDOW_OPENED : The window was opened.

WINDOW_STATE_CHANGED : The state of the window changed.

12. FocusEvent Class : A **FocusEvent** is generated when a component gains or loses input focus. These events are identified by the integer constants **FOCUS_GAINED** and **FOCUS_LOST**.

FocusEvent Class Constructors

1. FocusEvent (Component src, int type)
2. FocusEvent (Component src, int type, boolean temporaryFlag)
3. FocusEvent (Component src, int type, boolean temporaryFlag, Component other)

FocusEvent Class Methods

1. **Component getOppositeComponent () :** This method used to determine the other component.
2. **boolean isTemporary() :**This method returns true if the change is temporary.

MOUSE EVENT HANDLING:

Mouse events are one of the most common events we come across always. Every movement of the mouse can generate an event. To catch the events of the mouse.

In java provided us with two Listeners.

1. MouseListener
2. MouseMotionListener

Mouse Listener interface : MouseListener interface registers an event whenever the mouse cursor is moved into the area of a component *listening* for mouse events. To add this listener to a component, developers can use the addMouseListener() method.

Methods of Mouse Listener Interface:

1. **public abstract void mouseClicked(MouseEvent e) :** When the mouse button has been clicked (pressed and released) on a component.
2. **public abstract void mouseEntered(MouseEvent e):** When the mouse enters a component.
3. **public abstract void mouseExited(MouseEvent e) :** When the mouse exits a component.
4. **public abstract void mousePressed(MouseEvent e) :** When the mouse button has been pressed on a component.
5. **public abstract void mouseReleased(MouseEvent e) :** When the mouse button has been released on a component.

MouseMotionListener interface : This interface has two methods.

1. void mouseMoved(MouseEvent e) : This method is invoked when the mouse is moved from one place to another

2**. void mouseDragged(MouseEvent e)** : This method is invoked when the mouse is dragged.

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

KEY EVEN HANDLING:

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.

Methods of KeyListener interface

1. public abstract void keyPressed (KeyEvent e) : It is invoked when a key has been pressed.

2. public abstract void keyReleased (KeyEvent e) : It is invoked when a key has been released.

3. public abstract void keyTyped (KeyEvent e) : It is invoked when a key has been typed.

Example

```
import java.awt.*;
import java.awt.event.*;
public class KeyListenerExample extends Frame implements KeyListener
{
        Label l;
        TextArea area;
        KeyListenerExample()
        {
                l=new Label();
                l.setBounds(20,50,100,20);
                area=new TextArea();
                area.setBounds(20,80,300, 300);
                area.addKeyListener(this);
                add(l);
                add(area);
                setSize(400,400);
                setLayout(null);
                setVisible(true);
        }
        public void keyPressed(KeyEvent e)
        {
              l.setText("Key Pressed");
        }
        public void keyReleased(KeyEvent e)
        {
              l.setText("Key Released");
        }
        public void keyTyped(KeyEvent e)
        {
              l.setText("Key Typed");
        }
        public static void main(String[] args)
        {
              KeyListenerExample  kl=new KeyListenerExample();
        }
}
```

Adapter Classes

1. In a program, when a listener has many abstract methods to override, it becomes complex for the programmer to override all of them.
2. For example, for closing a frame, we must override seven abstract methods of WindowListener, but we need only one method of them.
3. For reducing complexity, Java provides a class known as "adapters" or adapter class.
4. Adapters are abstract classes, that are already being overriden.

Adapter class	Listener interface
WindowAdapter	WindowListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
FocusAdapter	FocusListener
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
HierarchyBoundsAdapter	HierarchyBoundsListener

```
import java.awt.*;
import java.awt.event.*;
public class AdapterExample
{
        AdapterExample()
        {
                Frame f=new Frame ("Window Adapter");
                f.addWindowListener(new WindowAdapter()
                {
                        public void windowClosing(WindowEvent e)
                        {
                                f.dispose();
                        }
                });
                f.setSize(400,400);
                f.setVisible(true);
        }
        public static void main(String[] args)
        {
                AdapterExample  ae=new AdapterExample();
        }
}
```

Inner Classes

- Inner class means one class which is a member of another class.
- We use inner classes to logically group classes and interfaces in one place so that it can be more readable and maintainable.

Types of Inner classes

There are four types of inner classes.

1. Member Inner class
2. Local inner classes
3. Anonymous inner classes
4. Static nested classes

1. Member Inner class : A non-static class that is created inside a class but outside a method is called member inner class. It is also known as a regular inner class. It can be declared with access modifiers like public, default, private, and protected.

Syntax:

```
class Outer
{
        //code
        class Inner
        {
                //code
        }
}
```

Example

```
class TestMemberOuter
{
        private int data=30;
        class Inner
        {
                void msg()
                {
                        System.out.println("data is "+data);
                }
        }
        public static void main(String args[])
        {
                TestMemberOuter obj=new TestMemberOuter();
                TestMemberOuter.Inner in=obj.new Inner();
                in.msg();
        }
}
```

2.Anonymous inner class

1. In Java, a class can contain another class known as nested class. It's possible to create a nested class without giving any name.
2. A nested class that doesn't have any name is known as an anonymous class.
3. An anonymous class must be defined inside another class. Hence, it is also known as an anonymous inner class.

Example

```
abstract class Person
{
            abstract void eat();
}
class TestAnonymousInner
{
            public static void main(String args[])
            {
                     Person p=new Person()
                     {
                              void eat()
                              {
                                       System.out.println("nice fruits");
                               }
                     };
                     p.eat();
            }
}
```

3.Local inner class

1. A class i.e. created inside a method is called local inner class in java.
2. If you want to invoke the methods of local inner class, you must instantiate this class inside the method.

Example

```
public class localInner
{
        private int data=30;
        void display()
        {
                class Local
                {
                        void msg()
                        {
                                System.out.println(data);
                        }
                }
                Local l=new Local();
                 l.msg();
        }
        public static void main(String args[])
        {
                localInner obj=new localInner();
                 obj.display();
        }
}
```

4.static nested class

A static class is a class that is created inside a class, is called a static nested class in Java. It cannot access non-static data members and methods. It can be accessed by outer class name.

- It can access static data members of the outer class, including private.
- The static nested class cannot access non-static (instance) data members or method

Example

```
class TestOuter
{
        static int data=30;
        static class Inner
        {
                void msg()
                {
                        System.out.println("data is "+data);
                }
        }
        public static void main(String args[])
        {
                TestOuter.Inner obj=new TestOuter.Inner();
                obj.msg();
        }
}
```

In this example, you need to create the instance of static nested class because it has instance method msg(). But you don't need to create the object of Outer class because nested class is static and static properties, methods or classes can be accessed without object.

What is Applet in Java?

An applet may be a Java program that registers on the server and can be executed within the client. The applet may be a window that was constructed either by using Applet or JApplet class. Applets may be a special sort of program that's embedded within the webpage to get the dynamic content. It basically runs inside the browser and works on the client-side.

All applets are subclasses of (either directly or indirectly) Applet. Applets aren't standalone programs. Instead, they run within either an internet browser or an applet viewer. Execution of an applet doesn't begin at the main().

Types of Applets in Java:

It is important to state at the outset that there are two sorts of applets.

1. **Applet:** it's based directly on Applet class. These applets use the Abstract Window Toolkit (AWT) to supply the graphic user interface(GUI). This sort of applet has been available since Java was first created.
2. **JApplet:** it's supported Swing class. Swing applets use the Swing classes to supply the GUI. Swing offers a richer and ofter easier-to-use interface than does the AWT. Thus, Swing-based applets are now the foremost popular.

Advantages of Java Applet

There are many advantages to the applet. they're as follows:

1. It works on the client-side therefore the reaction time is extremely less.
2. It is Secured
3. It is often executed by browsers running under many platforms, including Linux, Windows, Mac Os, etc

Drawbacks of Applet in Java

1. The plugin is required at the client browser to execute the applet.

2. An applet cannot load libraries or define native methods.

3. An applet cannot read or write files on the execution host and certain system properties.

4. An applet cannot make network connections except to the host that it came from.

Basically, in Java we have two types of applications:

1. **Standalone Applications:** The applications that are executed in the context of a local machine are called standalone applications. Their applications use w-1 of system resources and the resources are not sharable. This kind of application contains the main() method.
2. **Distributed Applications:** The applications that are executed under the control of a browser are called distributed applications. The amount of resources required is very minimum and the resources are sharable. These applications will not contain the main() method. To develop a distributed GUI we use Applet

Difference between Applet and Application

Applet	Application
1 : Applets are inherently use the GUI. Example : Web Page Animation	1 : Gui is optional, mostly character based. Example : Network server.
2 : Method expected by the JVM is init(),start(),destroy(),stop() and paint() methods.	2 : JVM expected main() method or start up.
3 : it requires more memory and web browser	3 : Less memory is required.
4 : More built in security	4 : Less Security.
5 : Environmenatal input are supplied through PARAMETER embedded in host html.	5 : Mostly command line arguments.

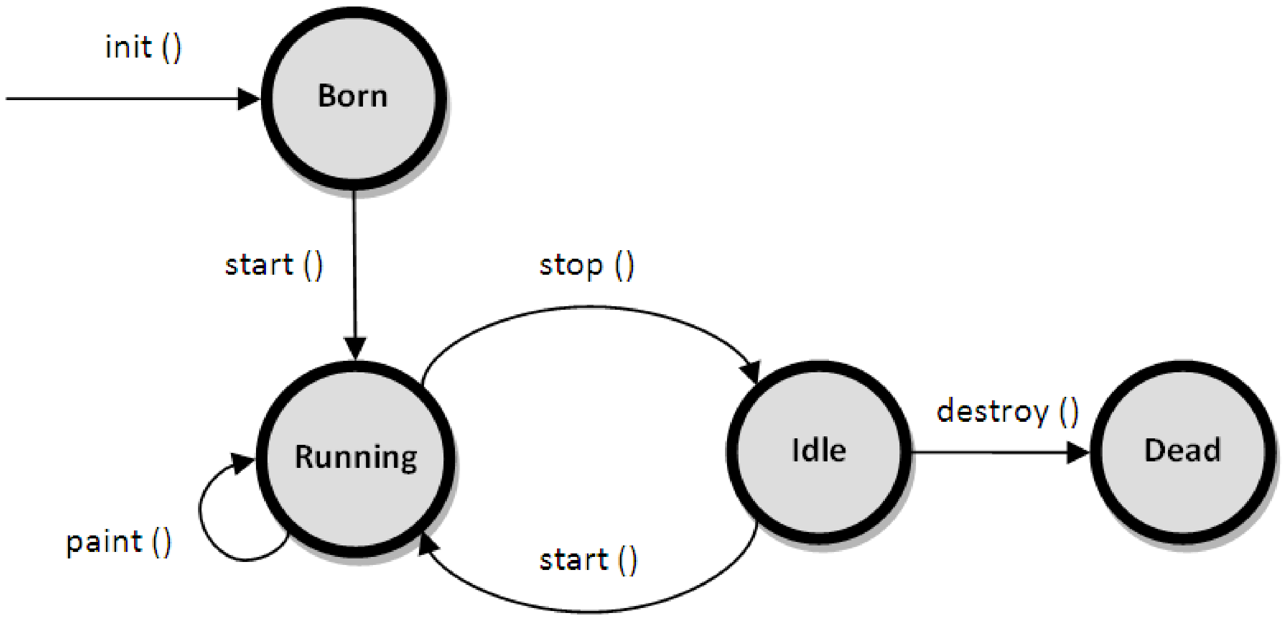
Applet LifeCycle

An applet undergoes various stages between its creation of objects and object removal as the work of the applet will get done. This cycle is known as Applet life cycle.

Following are the methods for a full applet cycle.

1. init() method
2. start() method
3. paint() method
4. stop() method
5. destroy() method

The life cycle of an applet is as shown in the figure below:



As shown in the above diagram the life cycle of an applet starts with init() method and ends with destroy() method. Other life cycle methods are start(), stop() and paint(). The methods to execute only once in the applet life cycle are init() and destroy(). Other methods execute multiple times.

Below is the description of each applet life cycle method:

1. **init():** The init() method is the first method to execute when the applet is executed. Variable declaration and initialization operations are performed in this method.

Syntax : public void init()

```
{
 //code;
}
```

2.start(): The start() method contains the actual code of the applet that should run. The start() method executes immediately after the init() method. It also executes whenever the applet is restored, maximized or moving from one tab to another tab in the browser. start() method can be called more than once.

Syntax : public void start()

```
{
 //code;
}
```

3.paint(): The paint() method is used to redraw the output on the applet display area. The paint() method executes after the execution of start() method and whenever the applet or browser is resized.

Syntax : public void paint()

```
{
 //code;
}
```

4.stop(): The stop() method stops the execution of the applet. The stop() method executes when the applet is minimized or when moving from one tab to another in the browser.

Syntax : public void stop()

```
{
 //code;
}
```

5.destroy(): The destroy() method executes when the applet window is closed or when the tab containing the webpage is closed. stop() method executes just before when destroy() method is invoked. The destroy() method removes the applet object from memory.

Syntax : public void destroy()

```
public void destroy()
{
 //code;
}
```

Example :

```
import java.awt.*;

import java.applet.*;

/*
<html>
        <applet code="AppletLife.class" width="200" height="300"></applet>
</html>
*/
public class AppletLife extends Applet
{
        public void init()
        {
                System.out.println("init()method");
        }
        public void start()
        {
                System.out.println("start() method");
        }
        public void paint(Graphics g)
        {
                System.out.println("from paint() method");
        }
        public void stop()
        {
                System.out.println("stop() method");
        }
        public void destroy()
        {
                System.out.println("destroy() method");
        }
}
```

PASSING PARAMETERS TO APPLET

1. We can supply user defined parameter to an applet using tag .
2. Param tag is used inside the applet tag.
3. Param tag has two attributes :
    1. Name
    2. Value

we will show you how to pass some parameters to an applet and how to read those parameters in an applet to display their values in the output.

Steps to accomplish this task -:

1. To pass the parameters to the Applet we need to use the **param** attribute of **<applet>** tag.
2. To retrieve a parameter's value, we need to use the **getParameter()** method of **Applet** class.

Signature of the getParamter() method

Syntax : public String getParameter(String *name*)

1. Method takes a String argument *name*, which represents the name of the parameter which was specified with the **param** attribute in the <applet> tag.
2. Method returns the value of the *name* parameter(if it was defined) else **null** is returned.

```
<applet  code="MyApplet.class"  height="300" width="500">
            <param  name="Name"  value="JBREC" />
            <param name="RollNumber" value="21J21A0550" />
            <param name="Age" value="20" />
            <param name="Mobile Number" value="1234567890" />
            <param name="Address" value="Hyderabad" />
</applet>
```

```
import java.awt.*;
import java.applet.*;
public class MyApplet extends Applet
{
            String Name;
            String RollNumber;
            String  Age;
            String  Mobile Number;
            String  Address ;
            public void init()
            {
                  Name = getParameter("Name ");
                  RollNumber = getParameter("RollNumber ");
                  Age = getParameter("Age ");
                  Mobile Number  = getParameter("Mobile Number ");
                  Address = getParameter("Address ");
            }
            public void paint(Graphics g)
            {
                  g.drawString("Name is: " + Name, 20, 20);
                  g.drawString("RollNumber is: " + RollNumber, 20, 40);
                  g.drawString("Age is: " + Age, 20, 60);
                  g.drawString("Mobile Number is: " + Mobile Number, 20, 80);
                  g.drawString("Address is: " + Address, 20, 100);
                  showStatus("PASSING PARAMETERS TO APPLETS");
            }
}
```

To execute the applet by appletviewer tool, write in command prompt:

```
c:\>javac MyApplet.java
c:\>appletviewer MyApplet.java
```

SWING

1. **Java Swing** is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*.
2. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.
3. Unlike AWT, Java Swing provides platform-independent and lightweight components.
4. The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

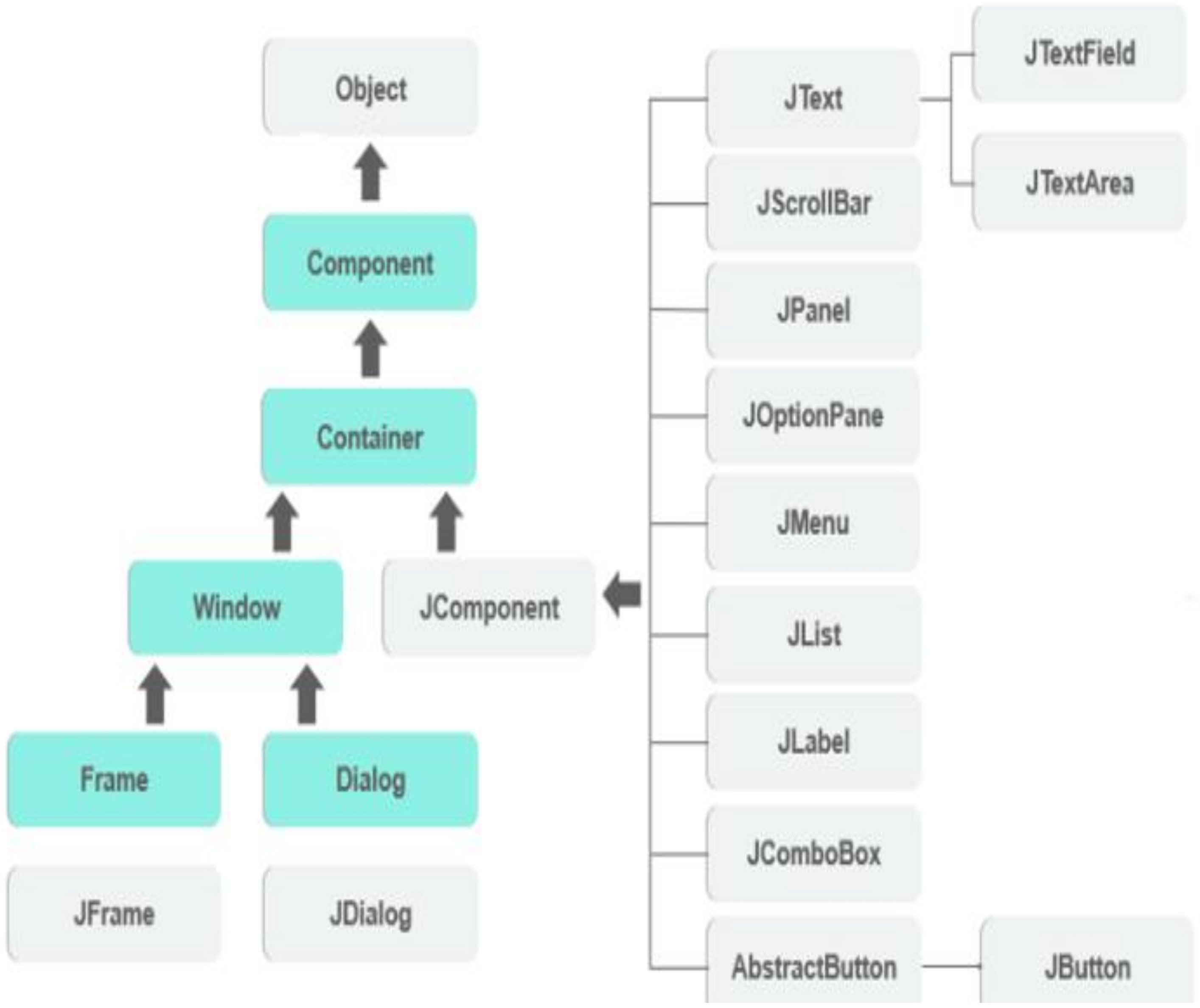
What is JFC

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

Difference between AWT and Swing

S.No	AWT	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 no of components than Swing.	Swing provides more no of 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	Swing follows MVC.

Hierarchy of Java Swing classes



Common Methods of Swing Component

1. **public void add(Component c):** It is used to add a component to another component.
2. **public void setSize(int width, int height):** It is used to set the size of the component.
3. **public void setLayout(LayoutManager m):** It is used to set the layout manager for the component.
4. **public void setVisible(boolean b):** It is used to set the visibility of the component. By default, the value is set to be false.

JFRAME:

The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

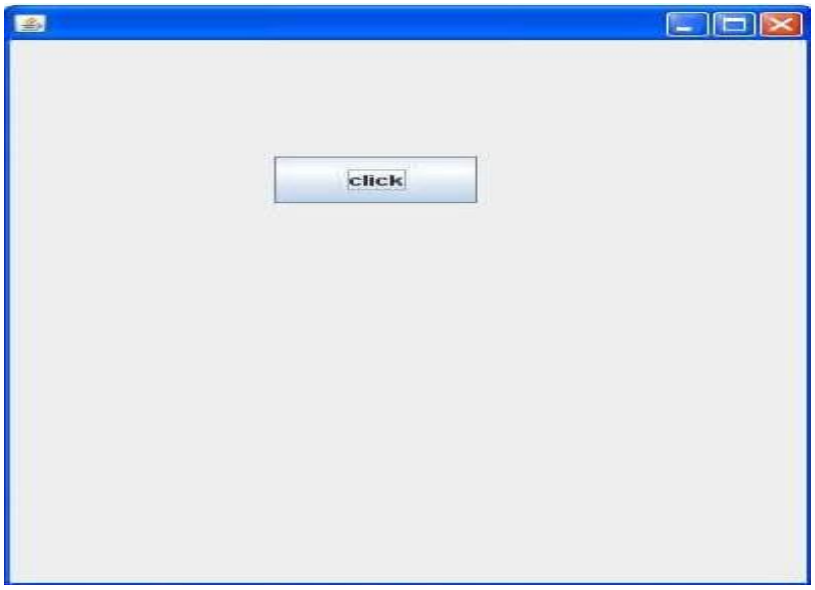
There are two ways to create a frame:

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

1.By creating the object of Frame class (association)

```
import javax.swing.*;
public class FirstSwingExample
{
        public static void main(String[] args)
        {
                JFrame f=new JFrame();
                JButton b=new JButton("click");
                b.setBounds(130,100,100, 40);
                f.add(b);
                f.setSize(400,500);
                f.setLayout(null);
                f.setVisible(true);
        }
}
```

Output:



2. By extending Frame class (inheritance)

We can also inherit the JFrame class, so there is no need to create the instance of JFrame class explicitly.

```
import javax.swing.*;
public class FirstSwingExample extends JFrame
{
        public static void main(String[] args)
        {
                JButton b=new JButton("click");
                b.setBounds(130,100,100, 40);
                add(b);
                setSize(400,500);
                setLayout(null);
                setVisible(true);
        }
}
```

SWINGS COMPONENTS

1. JLabel
2. JTextField
3. JTextArea
4. JButton
5. JCombobox
6. JCheckbox
7. JRadioButton
8. JScrollbar
9. JMenu
10. JOptionPane
11. JList

1. JLABEL CLASS

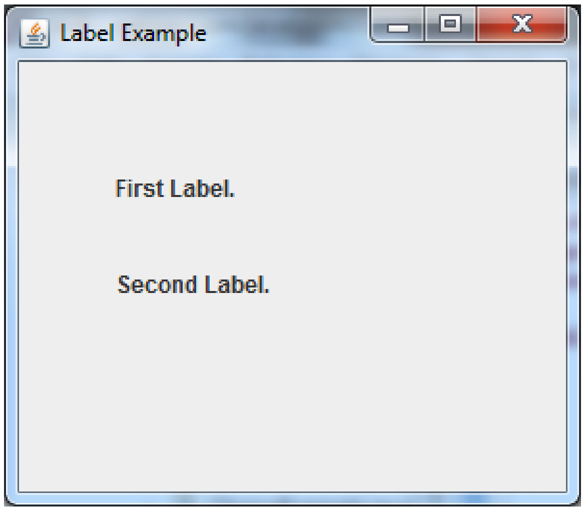
1. The object of JLabel class is a component for placing text in a container.
2. It is used to display a single line of read only text.
3. The text can be changed by an application but a user cannot edit it directly.

JLabel Constructors

1. **JLabel():** It is used to create a JLabel instance with no image and with an empty string for the title.
2. **JLabel(String s):** It is used to create a JLabel instance with the specified text.
3. **JLabel(Icon i):** It is used to create a JLabel instance with the specified image.
4. **JLabel(String s, Icon I, int horizontalAlignment):** It is used to create a JLabel instance with the specified text, image, and horizontal alignment.

```
import javax.swing.*;
class LabelExample
{
        public static void main(String args[])
        {
                JFrame f= new JFrame("Label Example");
                JLabel  l1=new JLabel("First Label.");
                l1.setBounds(50,50, 100,30);
                JLabel  l2=new JLabel("Second Label.");
                l2.setBounds(50,100, 100,30);
                f.add(l1);
                f.add(l2);
                f.setSize(300,300);
                f.setLayout(null);
                f.setVisible(true);
        }
  }
```

OUTPUT :



2. JTEXTFIELD

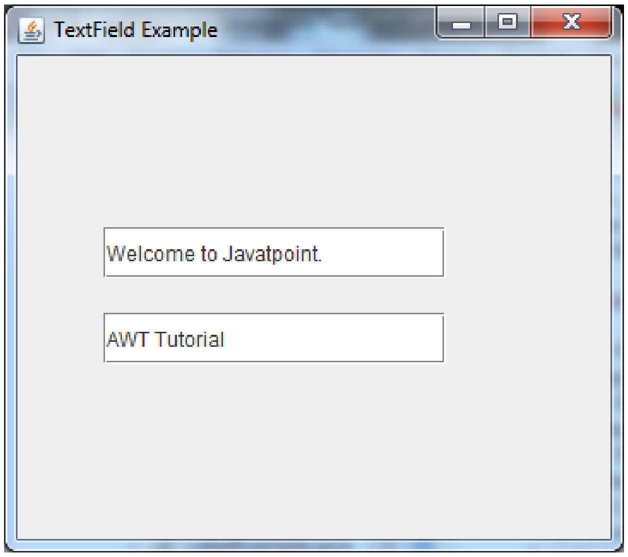
The JTextField component allows the user to type some text in a single line. It basically inherits the JTextComponent class.

JTextField Constructors

1. **JTextField():** It is used to create a new Text Field.
2. **JTextField(String text):** It is used to create a new Text Field initialized with the specified text.
3. **JTextField(String text, int columns):** It is used to create a new Text field initialized with the specified text and columns.
4. **JTextField(int columns):** It is used to create a new empty TextField with the specified number of columns.

```
import javax.swing.*;
class TextFieldExample
{
        public static void main(String args[])
        {
                JFrame f= new JFrame("TextField Example");
                JTextField t1=new JTextField("Welcome to Javatpoint.");
                t1.setBounds(50,100, 200,30);
                JTextField t2=new JTextField("AWT Tutorial");
                t2.setBounds(50,150, 200,30);
                f.add(t1);
                f.add(t2);
                f.setSize(400,400);
                f.setLayout(null);
                f.setVisible(true);
        }
}
```

OUTPUT :



3. JTEXTAREA

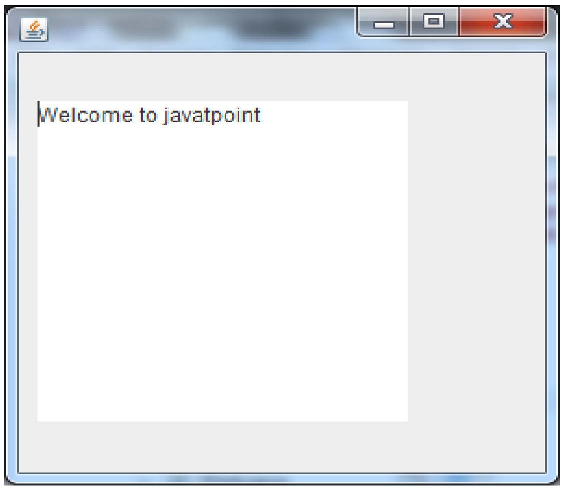
The JTextArea component allows the user to type the text in multiple lines. It also allows the editing of multiple-line text. It basically inherits the JTextComponent class.

TextArea Constructors

1. **JTextArea():** It is used to create a text area that displays no text initially.
2. **JTextarea(String s):** It is used to create a text area that displays specified text initially.
3. **JTextArea(int row, int column):** It is used to create a text area with the specified number of rows and columns that display no text initially.
4. **JTextarea(String s, int row, int column):** It is used to create a text area with the specified number of rows and columns that display specified text.

```
import javax.swing.*;
public class TextAreaExample
{
        TextAreaExample()
        {
                JFrame f= new JFrame();
                JTextArea area=new JTextArea("Welcome to javatpoint");
                area.setBounds(10,30, 200,200);
                f.add(area);
                f.setSize(300,300);
                f.setLayout(null);
                f.setVisible(true);
        }
        public static void main(String args[])
        {
                TextAreaExample  ta=new TextAreaExample();
        }
}
```

OUTPUT :



```
Welcome to javatpoint
```

4. JBUTTON

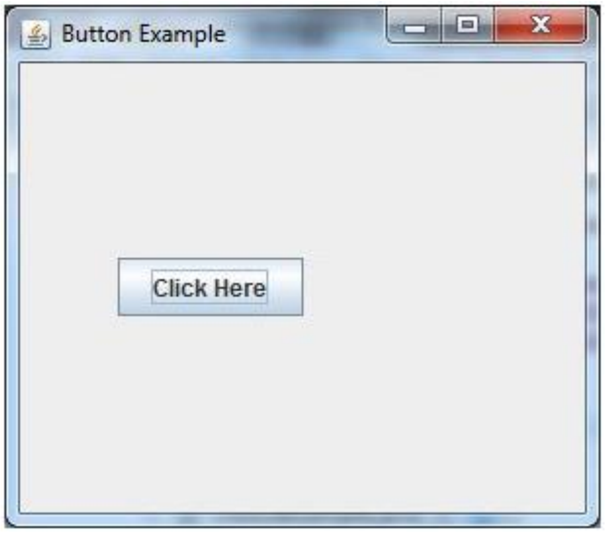
This component can be used to perform some operations when the user clicks on it. When the button is pushed, the application results in some action. It basically inherits the AbstractButton class.

JButton Constructors

1. **JButton():** It is used to create a button with no text and icon.
2. **JButton(String s):** It is used to create a button with the specified text.
3. **JButton(Icon i):** It is used to create a button with the specified icon object.

```
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:



5. JCOMBOBOX

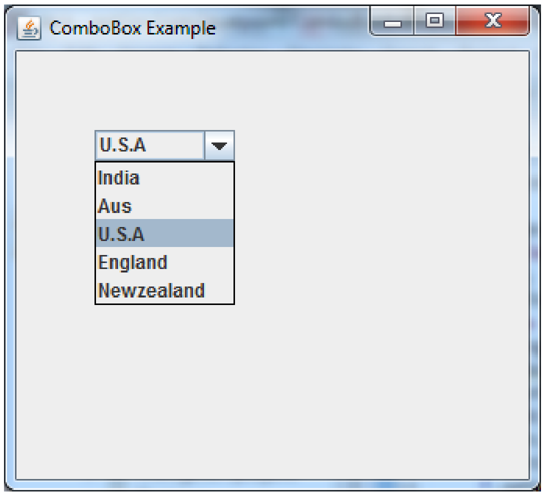
This component will display a group of items as a drop-down menu from which one item can be selected. At the top of the menu the choice selected by the user is shown.

JComboBox Constructors

1. **JComboBox():** It is used to create a JComboBox with a default data model.
2. **JComboBox(Object[] items):** It is used to create a JComboBox that contains the elements in the specified array.
3. **JComboBox(Vector<?> items):** It is used to create a JComboBox that contains the elements in the specified Vector.

```
import javax.swing.*;
public class ComboBoxExample
{
        ComboBoxExample()
        {
                JFrame 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)
        {
                ComboBoxExample  cb=new ComboBoxExample();
        }
}
```

OUTPUT :



6. JCHECKBOX

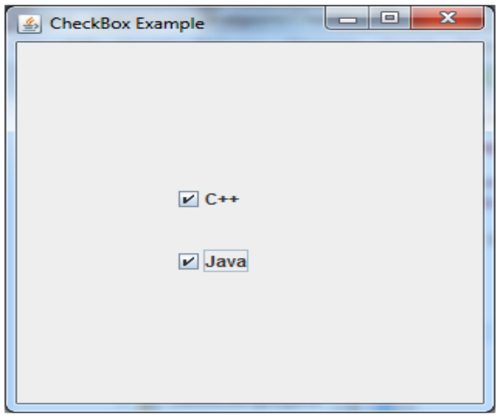
This component allows the user to select multiple items from a group of items. It is used to create a CheckBox. It is used to turn an option ON or OFF.

JCheckBox Constructors

1. **JCheckBox():** It is used to create an initially unselected checkbox button with no text, no icon.
2. **JCheckBox(Label):** It is used to create an initially unselected checkbox with text.
3. **JCheckBox(Label, boolean):** It is used to create a checkbox with text and specifies whether or not it is initially selected.
4. **JCheckBox(Action a):** It is used to create a checkbox where properties are taken from the Action supplied.

```
import javax.swing.*;
public class CheckBoxExample
{
        CheckBoxExample()
        {
                JFrame f= new JFrame("CheckBox Example");
                JCheckBox checkBox1 = new JCheckBox("C++");
                checkBox1.setBounds(100,100, 50,50);
                 JCheckBox checkBox2 = new JCheckBox("Java", 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[])
        {
                CheckBoxExample  cb=new CheckBoxExample();
        }
}
```

OUTPUT :



7. JRADIOBUTTON

Radio buttons are groups of buttons in which, by convention, only one button at a time can be selected. The Swing release supports radio buttons with the JRadioButton and ButtonGroup classes.

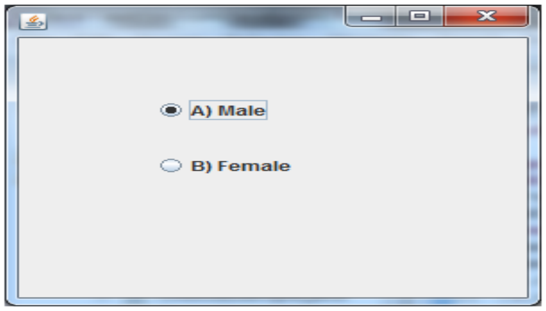
JRadioButton Constructors

1. **JRadioButton() :** Creates an unselected radio button with no text.
2. **JRadioButton(String s):** Creates an unselected radio button with specified text.
3. **JRadioButton(String s, boolean selected) :** Creates a radio button with the specified text and selected status.

```
import javax.swing.*;
public class RadioButtonExample
{
            RadioButtonExample()
            {
                        JFrame f=new JFrame();
                        JRadioButton r1=new JRadioButton("A) Male");
                        RadioButton r2=new JRadioButton("B) Female");
                        r1.setBounds(75,50,100,30);
                        r2.setBounds(75,100,100,30);
                        ButtonGroup bg=new ButtonGroup();
                        bg.add(r1);
                        bg.add(r2);
                        f.add(r1);
                        f.add(r2);
                        f.setSize(300,300);
                        f.setLayout(null);
                        f.setVisible(true);
        }
        public static void main(String[] args)
        {
                        RadioButtonExample  rb=new RadioButtonExample();
      }

}
```

OUTPUT ;



8. JSCROLLBAR CLASS

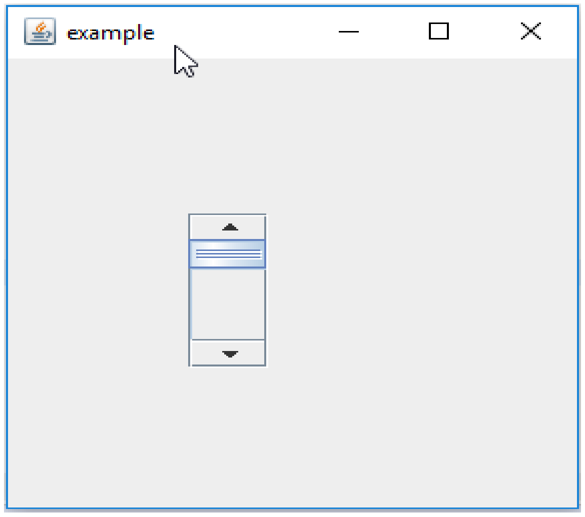
The object of JScrollbar class is used to add horizontal and vertical scrollbar. It is an implementation of a scrollbar. It inherits JComponent class.

JSCROLLBAR Constructors

1. **JScrollBar():**Creates a vertical scrollbar with the initial values.
2. **JScrollBar(int orientation):** Creates a scrollbar with the specified orientation and the initial values.
3. **JScrollBar(int orientation, int value, int extent, int min, int max) :** Creates a scrollbar with the specified orientation, value, extent, minimum, and maximum.

```
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[])
        {
                ScrollBarExample  sb=new ScrollBarExample();
        }
}
```

OUTPUT :

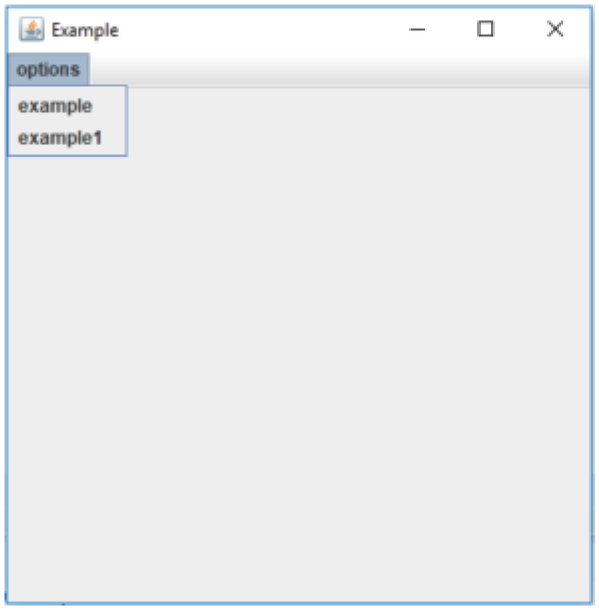


9. JMENU CLASS

It inherits the JMenuItem class, and is a pull down menu component which is displayed from the menu bar.

```
import javax.swing.*;
class JMenuDemo
{
        JMenuDemo ()
        {
                JFrame a = new JFrame("Example");
                JMenu menu = new JMenu("options");
                JMenuBar m1 = new JMenuBar();
                JMenuItem  a1 = new JMenuItem("example");
                JMenuItem  a2 = new JMenuItem("example1");
                menu.add(a1);
                menu.add(a2);
                m1.add(menu);
                a.setJMenuBar(m1);
                a.setSize(400,400);
                a.setLayout(null);
                a.setVisible(true);
        }
        public static void main(String args[])
        {
                JMenuDemo  e=new JMenuDemo ();
        }
}
```

OUTPUT:



10. JOPTIONPANE

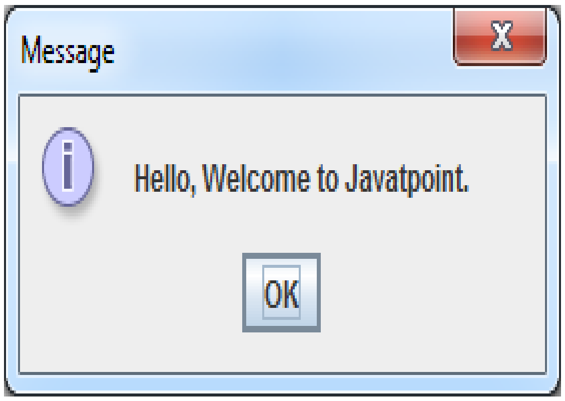
The JOptionPane class is used to provide standard dialog boxes such as message dialog box, confirm dialog box and input dialog box. These dialog boxes are used to display information or get input from the user. The JOptionPane class inherits JComponent class.

JOptionPane Constructors

1. **JOptionPane() :** It is used to create a JOptionPane with a test message.
2. **JOptionPane(Object message) :** It is used to create an instance of JOptionPane to display a message.
3. **JOptionPane(Object message, int messageType :** It is used to create an instance of JOptionPane to display a message with specified message type and default options.

```
import javax.swing.*;
public class OptionPaneExample
{
        OptionPaneExample()
        {
                JFrame   f=new JFrame();
                JOptionPane.showMessageDialog(f,"Hello, Welcome to Javatpoint.");
        }
        public static void main(String[] args)
        {
                OptionPaneExample  op=new OptionPaneExample();
        }
}
```

OUTPUT:



Hello, Welcome to Javatpoint.

11. JLIST CLASS

The object of JList class represents a list of text items. The list of text items can be set up so that the user can choose either one item or multiple items. It inherits JComponent class.

JLIST CLASS Constructors

1. JList() : Creates a JList with an empty, read-only, model.

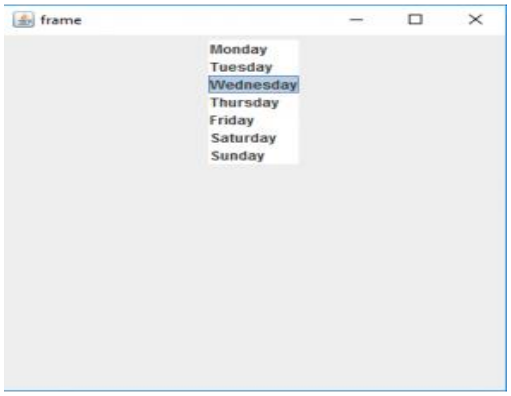
2. JList(ary[] listData) : Creates a JList that displays the elements in the specified array.

3. JList(ListModel<ary> dataModel) : Creates a JList that displays elements from the specified, non-null, model.

```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
class solve extends JFrame
{
            static JFrame f;
            static JList b;
            public static void main(String[] args)
            {
                        f = new JFrame("frame");
                        solve s=new solve();
                        JPanel p =new JPanel();
                        JLabel l= new JLabel("select the day of the week");
                        String week[]= { "Monday","Tuesday","Wednesday",
                                                  "Thursday","Friday","Saturday","Sunday"};

                        b= new JList(week);
                        p.add(b);
                        f.add(p);
                        f.setSize(400,400);
                        f.show();
            }
  }
```

OUTPUT :


frame
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday