Unit-I
Abstract Window Toolkit & Swing

Unit	Topic Name	Topics	Marks
1	**AWT**	Introduction to AWT, AWT Classes: Component, Container, Window, Panel, Frame etc., Top level window, Layout Managers: FlowLayout, BorderLayout, GridLayout, CardLayout, GridbagLayout, AWT Control Classes: Button, Label, TextField, TextArea, Checkbox, ScrollBar etc., Menu Classes: MenuBar, Menu, MenuItem etc., Dialog, FileDialog.	**16**
	Swing	Introduction to Swing, AWT vs. Swing, MVC Architecture, Swing Classes: JButton, JLabel, JTextField, JComboBox, JScrollPane, JTabbedPane, JTree etc.	

A. Introduction

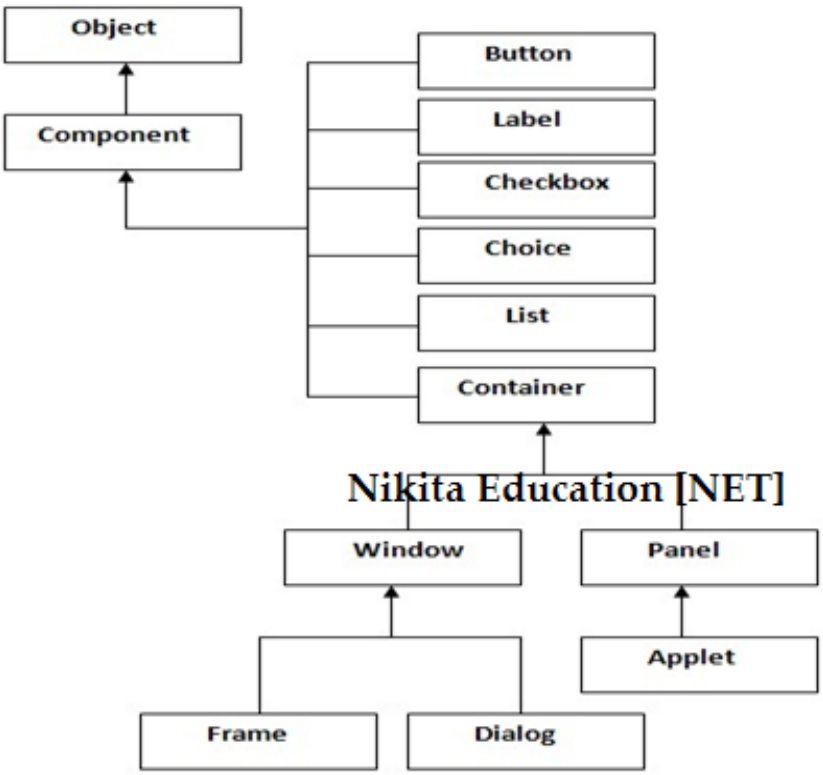
Graphical User Interface

Graphical User Interface (GUI) offers user interaction via some graphical components. GUI provides result to end user in response to raised events. GUI is entirely based on events.

AWT (Abstract Window Toolkit)

Abstract Window Toolkit (AWT) is a set of application program interfaces (**API - collection of classes and interfaces**) used by Java programmers to create graphical user interface (GUI) objects, such as buttons, scroll bars, and windows. The AWT is a class library that provides components for graphical user interfaces, e.g., buttons, labels, textboxes. Java AWT (Abstract Windowing Toolkit) is an API to develop GUI or window-based application in java. Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components uses the resources of system. The java.awt package provides classes for AWT API such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

AWT Classes Hierarchy

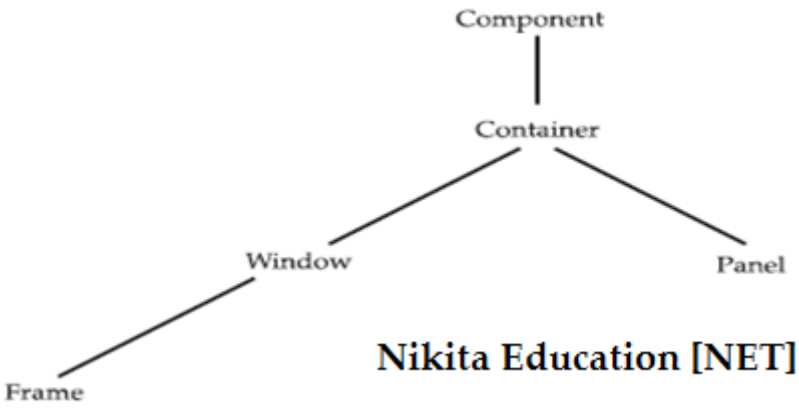


AWT Classes

Sr. No.	Control & Description
1	**Component** A Component is an abstract super class for GUI controls and it represents any object of its subclasses with graphical representation.
2	**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 containers such as Frame, Dialog and Panel.
3	**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.
4	**Panel** The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc
5	**Frame** The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.
6	**Label** A Label object is a component for placing text in a container.
7	**Button** This class creates a labeled button.
8	**Check Box** A check box is a graphical component that can be in either an **on** (true) or **off** (false) state.
11	**Text Field** A TextField object is a text component that allows for the editing of a single line of text.
12	**Text Area** A TextArea object is a text component that allows for the editing of a multiple lines of text.
13	**Choice** A Choice control is used to show pop up menu of choices. Selected choice is shown on the top of the menu.
15	**Scroll Bar** A Scrollbar control represents a scroll bar component in order to enable user to select from range of values.
16	**Dialog** A Dialog control represents a top-level window with a title and a border used to take some form of input from the user.
17	**File Dialog** A FileDialog control represents a dialog window from which the user can select a file.

Windows Fundamental:

The two most common windows are those derived from Panel, which is used by applets, and those derived from Frame, which creates a standard application window. Following are the window fundamental classes used to create **top level windows** that can open on OS desktops or Web Browsers and hold other components and containers.



- **Component**

At the top of the AWT hierarchy is the **Component** class. **Component** is an abstract class that encapsulates all of the attributes of a visual component. All user interface elements that are displayed on the screen and that interact with the user are subclasses of **Component**. It defines over a hundred public methods.

- **Container**

The **Container** class is a subclass of **Component**. It has additional methods that allow other **Component** objects to be nested within it. A container is responsible for laying out (that is, positioning) any components that it contains. It does this through the use of various layout managers.

- **Panel**

The **Panel** class is a concrete subclass of **Container**. **Panel** is the super class for **Applet**. When screen output is directed to an applet, it is drawn on the surface of a **Panel** object. A **Panel** is a window that does not contain a title bar, menu bar, or border. Other components can be added to a **Panel** object by its **add( )** method.

- **Window**

The **Window** class creates a top-level window. A*top-level window* is not contained within any other object; it sits directly on the desktop. Generally, you won't create **Window** objects directly. Instead, you will use a subclass of **Window** called **Frame**, described next.

- **Frame**

Frame is a subclass of **Window** and has a title bar, menu bar, borders, and resizing corners. Frame class is used to create **standard top level windows** that can open on desktop. It provides more features than Window class to design a standard GUI application window.

B. Creating Frame as a top level window

B.1. Important constructors of Frame

- **public Frame()** Constructs a new instance of Frame that is initially invisible.
- **public Frame(String title)** Constructs a new invisible Frame object with the specified title.

B.2. Important methods for Frame

- **setSize**(int width, int height) Resizes this component so that it has width and height.
- **setVisible**(boolean b) Shows or hides this component depending on the value of parameter b.
- **setTitle**(**String** title) Sets the title for this frame to the specified string.
- **addWindowListener**(**WindowListener** wl) Adds the specified window listener to receive window events from this window.
- **setLayout**(**LayoutManager** mgr) Sets the layout manager for this container.

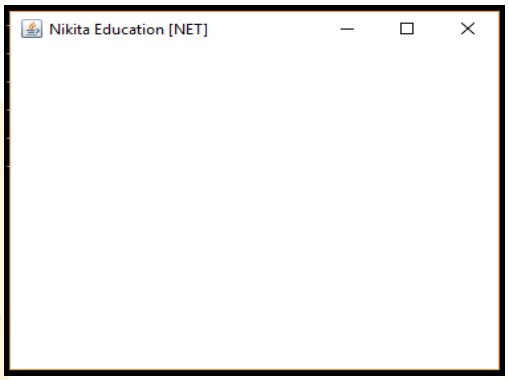
B.3. Simple Frame Examples

- Creating instance of Frame
- Inheriting properties of Frame

1. Creating instance of Frame

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

class MyNewFrameWindow {

        public static void main(String[] args) {

                Frame newFrame=new Frame();

                newFrame.setTitle("Nikita Education");
                newFrame.setSize(500,500);
                newFrame.setVisible(true);
        }
}
```

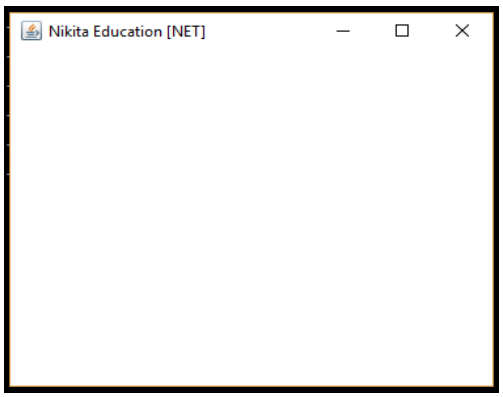
To Run:
Javac MyNewFrameWindow.java
Java MyNewFrameWindow



2. Inheriting properties of Frame

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

public class MyNewApp extends Frame
{
        public MyNewApp()
        {
                setTitle("Nikita Education [NET]");
                setSize(500,500);
                setVisible(true);
                setLayout(new FlowLayout());
        }

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

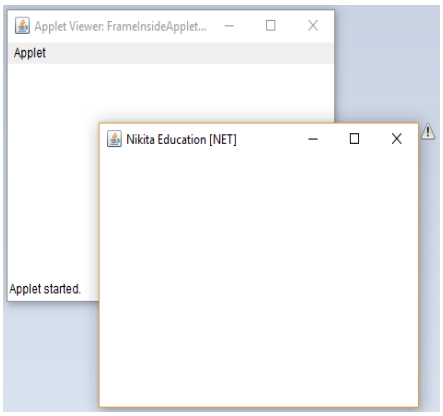
To Run:
Javac MyNewApp.java
Java MyNewApp



3. Creating Frame inside Applet

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

/*      <applet code="MyAppletFrame.class"
        Height=500 Width=500></applet>          */

public class MyAppletFrame extends Applet
{
        Frame newFrame;

        public void init()
        {
                newFrame=new Frame();
                newFrame.setTitle("Nikita Education");
                newFrame.setSize(500,500);
        }

        public void start()
        {
                newFrame.setVisible(true);
        }

        public void stop()
        {
                newFrame.setVisible(false);
        }
}
```

To Run:
Javac MyAppletFrame.java
Appletviewer MyAppletFrame.java



4. Frame close event

```
import java.awt.*;
import java.awt.event.*;
public class FrameCloseEvent extends Frame implements WindowListener{
        public FrameCloseEvent(){
                setTitle("Frame Window");
                setSize(500,500);
                setVisible(true);
                addWindowListener(this);
        }
        public void windowClosing(WindowEvent we){
                System.exit(0);
        }
        public void windowOpened(WindowEvent we){}
        public void windowClosed(WindowEvent we){}
        public void windowIconified(WindowEvent we){}
        public void windowDeiconified(WindowEvent we){}
        public void windowActivated(WindowEvent we){}
        public void windowDeactivated(WindowEvent we){}
        public static void main(String [] args){
                FrameCloseEvent newFrame=new FrameCloseEvent();
        }
}
```

Implements event handler for WindowEvent

Handler registration with source i.e. Frame

Event handler method defines action steps

It is compulsory to define all the methods of interface in its subclass

C. AWT Controls

Controls are **components** that allow a user to interact with your application in various ways. All AWT controls are placed inside a container and a **layout manager** automatically positions components within a container. Thus, the appearance of a window is determined by a combination of the **controls** that it contains and the **layout manager** used to position them. The AWT supports the following types of controls: **Labels, Push buttons, Check boxes, Choice lists, Lists, Scroll bars, Text editing etc**. These controls are subclasses of **Component** class**.** The **HeadlessException** is thrown by all AWT Controls.

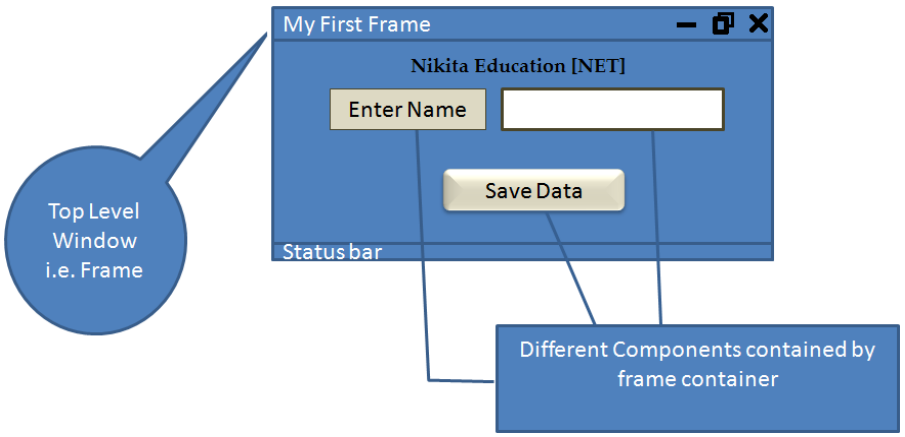
C.1. Adding and Removing Controls inside a container i.e. Frame or Panel

To show control in a window, you must add it to the **Frame or Panel**. To do this, **you must first create an instance of the desired control and then add it to a window by calling add( ) method,** which is defined by **Container** class**.**

- Component add(Component *compObj)*
- void remove(Component *obj)*

C.2. Standard Steps to Add Controls inside a container i.e. Frame or Panel:

- ***Create instance of Control for eg.***
  1. Label control: **Label name=new Label("Enter your name");**
  2. TextField control: **TextField t1=new TextField(50);**
  3. Button control: **Button btnSave=new Button("Save Data");**
- ***Set basic properties of control for eg.***
  1. Label control: **name.setAlignment(Label.RIGHT);**
  2. Button control: **btnSave.setBounds(100,150,50,30);**
- ***Register event with control for eg.***
  1. Window event: **addWindowListener(this);**
  2. Button control: **btnSave.addActionListener(this);**
- ***Add control on Frame or Panel for eg.***
  1. Any Control: **add(name); add(t1); add(btnSave);**



D. Label

D.1. Important Constants

Label.CENTER -- Indicates that the label should be centered. **Label.LEFT** -- Indicates that the label should be left justified. **Label.RIGHT** -- Indicates that the label should be right justified.

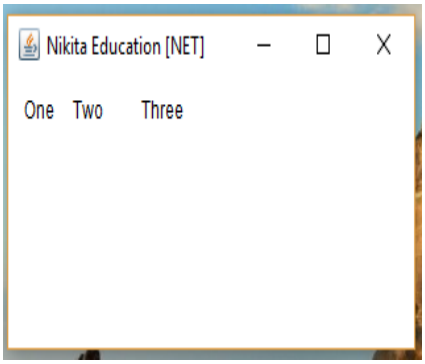
D.2. Important Constructors

Label() Constructs an empty label.
Label(String text) Constructs a new label with the specified string of text, left justified.
Label(String text, int alignment) Constructs a new label that presents the specified string of text with the specified alignment.

D.3. Important Methods

int getAlignment() Gets the current alignment of this label.
String getText() Gets the text of this label.
void setAlignment(int alignment) Sets the alignment for this label to the specified alignment.
void setText(String text) Sets the text for this label to the specified text.

```
import java.awt.*;
import java.awt.event.*;
public class LabelDemo extends Frame{
	public LabelDemo(){
		setTitle("Nikita Education [NET]");
		setSize(500,500);
		setVisible(true);
		setLayout(new FlowLayout(FlowLayout.LEFT));

		Label one = new Label();
		Label two = new Label("Two");
		Label three = new Label("Three",Label.RIGHT);
		one.setText("One");
		one.setAlignment(Label.CENTER);
		String lblValue=two.getText();
		int algn=two.getAlignment();
		// add labels to frame window
		add(one);
		add(two);
		add(three);
	}
	public static void main(String [] args){
		LabelDemo obj1=new LabelDemo();
	}
}
```



E. Button

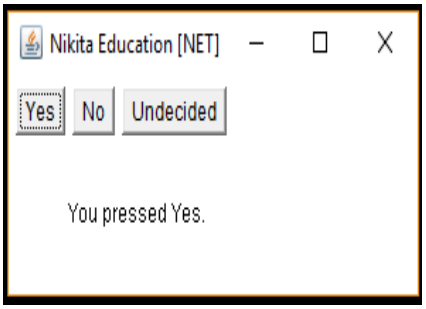
E.1. Important Constructors

Button() Constructs a button with an empty string for its label.
Button(String text) Constructs a new button with specified label.

E.2. Important Methods

void addActionListener(ActionListener l) Adds the specified action listener to receive action events from this button.
String getActionCommand() Returns the command name of the action event fired by this button.
String getLabel() Gets the label of this button.
void setActionCommand(String command) Sets the command name for the action event fired by this button.
void setLabel(String label) Sets the button's label to be the specified string.

```
import java.awt.*;
import java.awt.event.*;
public class ButtonDemo extends Frame implements ActionListener{
	String msg = "";
	Button yes, no, maybe;
	public ButtonDemo(){
		setTitle("Nikita Education [NET]");
		setSize(500,500);
		setVisible(true);
		setLayout(new FlowLayout(FlowLayout.LEFT));

		yes = new Button("Yes");
		no = new Button("No");
		maybe = new Button("Undecided");
		add(yes);
		add(no);
		add(maybe);
		yes.addActionListener(this);
		no.addActionListener(this);
		maybe.addActionListener(this);
	}
	public void actionPerformed(ActionEvent ae){
		String str = ae.getActionCommand();
		if(str.equals("Yes")){
			msg = "You pressed Yes.";
		}
		else if(str.equals("No")){
			msg = "You pressed No.";
		}
		else{
			msg = "You pressed Undecided.";
		}
		repaint();
	}
	public void paint(Graphics g) {
		g.drawString(msg, 50, 100);
	}
	public static void main(String [] args){
		ButtonDemo obj1=new ButtonDemo();
	}
}
```



F. Checkbox

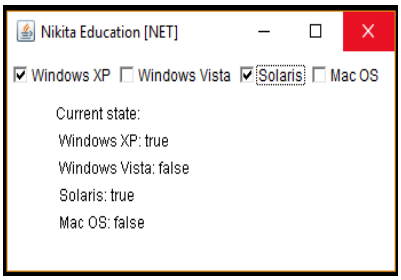
F.1. Important Constructors

Checkbox() Creates a check box with an empty string for its label.
Checkbox(String label) Creates a check box with the specified label.
Checkbox(String label, boolean state) Creates a check box with the specified label and sets the specified state.
Checkbox(String label, boolean state, CheckboxGroup group) Constructs a Checkbox with the specified label, set to the specified state, and in the specified check box group.

F.2. Important Methods

void addItemListener(ItemListener l) Adds the specified item listener to receive item events from this check box.
CheckboxGroup getCheckboxGroup() Determines this check box's group.
String getLabel() Gets the label of this check box.
boolean getState() Determines whether this check box is in the **on** or **off** state.
void setCheckboxGroup(CheckboxGroup g) Sets this check box's group to the specified check box group.
void setLabel(String label) Sets this check box's label to be the string argument.
void setState(boolean state) Sets the state of this check box to the specified state.

```
import java.awt.*;
import java.awt.event.*;
public class CheckboxDemo extends Frame implements ItemListener{
        String msg = "";
        Checkbox winXP, winVista, solaris, mac;
        public CheckboxDemo(){
                setTitle("Nikita Education [NET]");
                setSize(500,500);
                setVisible(true);
                setLayout(new FlowLayout(FlowLayout.LEFT));

                winXP = new Checkbox("Windows XP", null, true);
                winVista = new Checkbox("Windows Vista");
                solaris = new Checkbox("Solaris");
                mac = new Checkbox("Mac OS");
                add(winXP);
                add(winVista);
```



```
                add(solaris);
                add(mac);
                winXP.addItemListener(this);
                winVista.addItemListener(this);
                solaris.addItemListener(this);
                mac.addItemListener(this);
        }
        public void itemStateChanged(ItemEvent ie){
                repaint();
        }
        public void paint(Graphics g) {
                msg = "Current state: ";
                g.drawString(msg, 50, 80);
                msg = " Windows XP: " + winXP.getState();
                g.drawString(msg, 50, 100);
                msg = " Windows Vista: " + winVista.getState();
                g.drawString(msg, 50, 120);
                msg = " Solaris: " + solaris.getState();
                g.drawString(msg, 50, 140);
                msg = " Mac OS: " + mac.getState();
                g.drawString(msg, 50, 160);
        }
        public static void main(String [] args){
                CheckboxDemo obj1=new CheckboxDemo();
        }
}
```

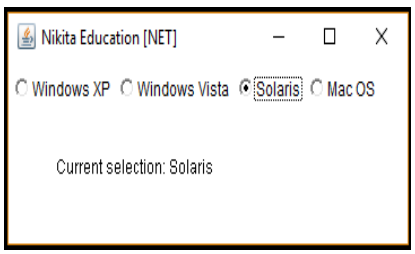
G. CheckboxGroup

It is possible to create a set of **mutually exclusive** check boxes in which one and only one check box in **the group** can be checked at any one time. These check boxes are often called ***radio buttons.***

G.1. Important Constructors & Methods

CheckboxGroup() () Creates a new instance of CheckboxGroup.
Checkbox getSelectedCheckbox() Gets the current choice from this check box group.
void setSelectedCheckbox(Checkbox box) Sets the currently selected check box in this group to be the specified check box.

```
import java.awt.*;
import java.awt.event.*;
public class CheckboxGroupDemo extends Frame
implements ItemListener{
        String msg = "";
        Checkbox winXP, winVista, solaris, mac;
        CheckboxGroup cbg;
        public CheckboxGroupDemo(){
                setTitle("Nikita Education [NET]");
                setSize(500,500);
                setVisible(true);
                setLayout(new FlowLayout(FlowLayout.LEFT));
```



```
                cbg = new CheckboxGroup();
                winXP = new Checkbox("Windows XP", cbg, true);
                winVista = new Checkbox("Windows Vista", cbg, false);
                solaris = new Checkbox("Solaris", cbg, false);
                mac = new Checkbox("Mac OS", cbg, false);
                add(winXP);
                add(winVista);
                add(solaris);
                add(mac);
                winXP.addItemListener(this);
                winVista.addItemListener(this);
                solaris.addItemListener(this);
                mac.addItemListener(this);
        }
        public void itemStateChanged(ItemEvent ie){
                repaint();
        }
        public void paint(Graphics g) {
                msg = "Current selection: ";
                msg += cbg.getSelectedCheckbox().getLabel();
                g.drawString(msg, 50, 100);
        }
        public static void main(String [] args){
                CheckboxGroupDemo obj1=new CheckboxGroupDemo();
        }
}
```

H. Choice

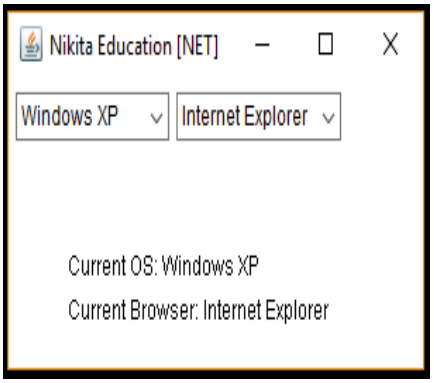
H.1. Important Methods

void add(String item): Adds an item to this Choice menu.
void addItemListener(ItemListener l) Adds the specified item listener to receive item events from this Choice menu.
String getItem(int index) Gets the string at the specified index in this Choice menu.
int getItemCount() Returns the number of items in this Choice menu.
int getSelectedIndex() Returns the index of the currently selected item.
String getSelectedItem() Gets a representation of the current choice as a string.
void insert(String item, int index) Inserts the item into this choice at the specified position.
void remove(int position) Removes an item from the choice menu at the specified position.
void remove(String item) Removes the first occurrence of item from the Choice menu.
void removeAll(): Removes all items from the choice menu.

void select(int pos)
Sets the selected item in this Choice menu to be the item at the specified position.

void select(String str)
Sets the selected item in this Choice menu to be the item whose name is equal to the specified string.

```
import java.awt.*;
import java.awt.event.*;
public class ChoiceDemo extends Frame implements ItemListener{
	Choice os, browser;
	String msg = "";
	public ChoiceDemo(){
		setTitle("Nikita Education [NET]");
		setSize(500,500);
		setVisible(true);
		setLayout(new FlowLayout(FlowLayout.LEFT));

		os = new Choice();
		browser = new Choice();

		// add items to os list
		os.add("Windows XP");
		os.add("Windows Vista");
		os.add("Solaris");
		os.add("Mac OS");

		// add items to browser list
		browser.add("Internet Explorer");
		browser.add("Firefox");
		browser.add("Opera");
		browser.add("Google Chrome");

		// add choice lists to window
		add(os);
		add(browser);

		// register to receive item events
		os.addItemListener(this);
		browser.addItemListener(this);
	}
	public void itemStateChanged(ItemEvent ie){
		repaint();
	}
	public void paint(Graphics g) {
		msg = "Current OS: ";
		msg += os.getSelectedItem();
		g.drawString(msg, 50, 120);

		msg = "Current Browser: ";
		msg += browser.getSelectedItem();
		g.drawString(msg, 50, 140);
	}
	public static void main(String [] args)
	{
		ChoiceDemo obj1=new ChoiceDemo();
	}
}
```



I. List

I.1. Important Constructors

List() Creates a new scrolling list.
List(int rows) Creates a new scrolling list initialized with the specified number of visible lines.
List(int rows, boolean multipleMode) Creates a new scrolling list initialized to display the specified number of rows.

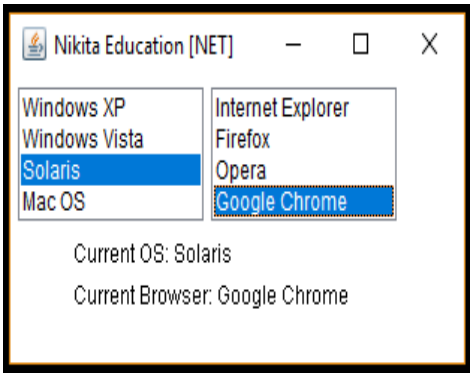
I.2. Important Methods

void add(String item) Adds the specified item to the end of scrolling list.
void add(String item, int index) Adds the specified item to the the scrolling list at the position indicated by the index.
void addActionListener(ActionListener l) Adds the specified action listener to receive action events from this list.
void addItemListener(ItemListener l) Adds the specified item listener to receive item events from this list.
void deselect(int index) Deselects the item at the specified index.
String getItem(int index) Gets the item associated with the specified index.
int getItemCount() Gets the number of items in the list.
String[] getItems() Gets the items in the list.
int getSelectedIndex() Gets the index of the selected item on the list,
int[] getSelectedIndexes() Gets the selected indexes on the list.
String getSelectedItem() Gets the selected item on this scrolling list.
String[] getSelectedItems() Gets the selected items on this scrolling list.
void remove(int position) Removes the item at the specified position from this scrolling list.
void remove(String item) Removes the first occurrence of an item from the list.
void removeAll(): Removes all items from this list.

void replaceItem(String newValue, int index)
Replaces the item at the specified index in the scrolling list with the new string.

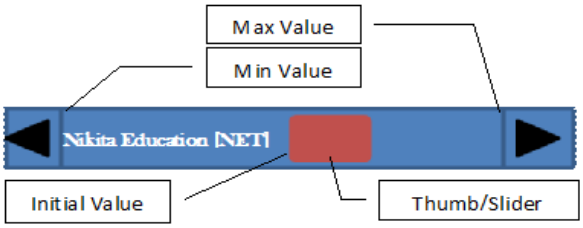
void select(int index)
Selects the item at the specified index in the scrolling list.

```
import java.awt.*;
import java.awt.event.*;
public class ListDemo extends Frame implements ActionListener{
	List os, browser;
	String msg = "";
	public ListDemo(){
		setTitle("Nikita Education [NET]");
		setSize(500,500);
		setVisible(true);
		setLayout(new FlowLayout(FlowLayout.LEFT));
		os = new List(4, true);
		browser = new List(4, false);

		// add items to os list
		os.add("Windows XP");
		os.add("Windows Vista");
		os.add("Solaris");
		os.add("Mac OS");

		// add items to browser list
		browser.add("Internet Explorer");
		browser.add("Firefox");
		browser.add("Opera");
		browser.add("Google Chrome");
		browser.select(1);

		// add lists to window
		add(os);
		add(browser);
		// register to receive action events
		os.addActionListener(this);
		browser.addActionListener(this);
	}
	public void actionPerformed(ActionEvent ae) {
		repaint();
	}
	public void paint(Graphics g) {
		int idx[];
		msg = "Current OS: ";
		idx = os.getSelectedIndexes();
		for(int i=0; i<idx.length; i++)
			msg += os.getItem(idx[i]) + " ";
		g.drawString(msg, 50, 120);
		msg = "Current Browser: ";
		msg += browser.getSelectedItem();
		g.drawString(msg, 50, 140);
	}
	public static void main(String [] args){
		ListDemo obj1=new ListDemo();
	}
}
```



J. Scrollbar

Scroll bars are used to select **continuous values** between a specified **minimum** and **maximum**. Scroll bars may be oriented **horizontally** or **vertically**. A scroll bar is actually a composite of several individual parts such as **min & max value, slider box, arrow keys, initial value** etc. Scrollbar control represents a scroll bar component in order to enable user to select from range of values.



Horizontal Scrollbar

J.1. Important Constants

- **static int HORIZONTAL** --A constant that indicates a horizontal scroll bar.
- **static int VERTICAL** --A constant that indicates a vertical scroll bar.

J.2. Important Constructors

Scrollbar() Constructs a new vertical scroll bar.
Scrollbar(int orientation) Constructs a new scroll bar with the specified orientation.
Scrollbar(int orientation, int value, int visible, int minimum, int maximum) Constructs a new scroll bar with the specified orientation, initial value, visible amount, and minimum and maximum values.

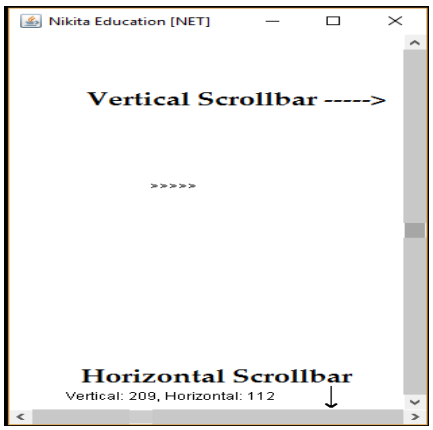
J.3. Important Methods

void addAdjustmentListener(AdjustmentListener l) Adds the specified adjustment listener to receive instances of AdjustmentEvent from this scroll bar.
int getBlockIncrement() Gets the block increment of this scroll bar.
int getMaximum() Gets the maximum value of this scroll bar.
int getMinimum() Gets the minimum value of this scroll bar.
int getUnitIncrement() Gets the unit increment for this scrollbar.
int getValue() Gets the current value of this scroll bar.
void setBlockIncrement(int v) Sets the block increment for this scroll bar.
void setMaximum(int newMaximum) Sets the maximum value of this scroll bar.
void setMinimum(int newMinimum): Sets the minimum value of this scroll bar.

void setUnitIncrement(int v)
Sets the unit increment for this scroll bar.

void setValue(int newValue)
Sets the value of this scroll bar to the specified value.

```
import java.awt.*;
import java.awt.event.*;
public class ScrollbarDemo extends Frame
implements AdjustmentListener, MouseMotionListener{
        Scrollbar vertSB, horzSB;
        String msg = "";
        public ScrollbarDemo(){
                setTitle("Nikita Education [NET]");
                setSize(500,500);
                setVisible(true);
                setLayout(new BorderLayout());

                int width = 400;
                int height = 400;

vertSB = new Scrollbar(Scrollbar.VERTICAL, 0, 1, 0, height);
horzSB = new Scrollbar(Scrollbar.HORIZONTAL, 0, 1, 0, width);

                add(vertSB,BorderLayout.EAST);
                add(horzSB,BorderLayout.SOUTH);
                // register to receive adjustment events
                vertSB.addAdjustmentListener(this);
                horzSB.addAdjustmentListener(this);
                addMouseMotionListener(this);
        }
        public void adjustmentValueChanged(AdjustmentEvent ae) {
                repaint();
        }
        // Update scroll bars to reflect mouse dragging.
        public void mouseDragged(MouseEvent me) {
                int x = me.getX();
                int y = me.getY();
                vertSB.setValue(y);
                horzSB.setValue(x);
                repaint();
        }
        // Necessary for MouseMotionListener
        public void mouseMoved(MouseEvent me){}
        public void paint(Graphics g)
        {
                msg = "Vertical: " + vertSB.getValue();
                msg += ", Horizontal: " + horzSB.getValue();
                g.drawString(msg, 50, 450);
                // show current mouse drag position
                g.drawString(">>>>>", horzSB.getValue(),
                vertSB.getValue());
        }
        public static void main(String [] args){
                ScrollbarDemo obj1=new ScrollbarDemo();
        }
}
```



K. TextField

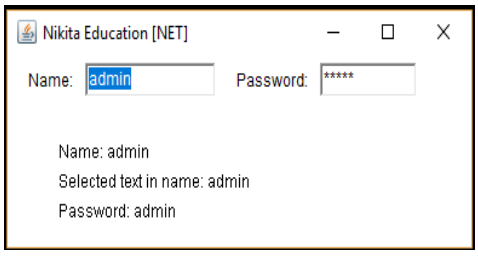
K.1. Important Constructors

TextField() Constructs a new text field.
TextField(int columns) Constructs a new empty text field with the specified number of columns.
TextField(String text) Constructs a new text field initialized with the specified text.
TextField(String text, int columns) Constructs a new text field initialized with the specified text to be displayed, and wide enough to hold the specified number of columns.

K.2. Important Methods

void addActionListener(ActionListener l) Adds the specified action listener to receive action events from this text field.
int getColumns() Gets the number of columns in this text field.
char getEchoChar() Gets the character that is to be used for echoing.
void setColumns(int columns) Sets the number of columns in this text field.
void setEchoChar(char c) Sets the echo character for this text field.
void setText(String t) Sets the text that is presented by this text component to be the specified text.
String getSelectedText( ) **void select(int *startIndex*, int *endIndex*)**

```
import java.awt.*;
import java.awt.event.*;
public class TextFieldDemo extends Frame implements ActionListener{
        TextField name, pass;
        public TextFieldDemo(){
                setTitle("Nikita Education [NET]");
                setSize(500,500);
                setVisible(true);
                setLayout(new FlowLayout(FlowLayout.LEFT));
        Label namep = new Label("Name: ", Label.RIGHT);
        Label passp = new Label("Password: ", Label.RIGHT);
                name = new TextField(12);
                pass = new TextField(8);
                pass.setEchoChar('*');
                add(namep);
                add(name);
```



```
                add(passp);
                add(pass);
                // register to receive action events
                name.addActionListener(this);
                pass.addActionListener(this);
        }
        public void actionPerformed(ActionEvent ae) {
                repaint();
        }
        public void paint(Graphics g) {
                g.drawString("Name: " + name.getText(), 50, 100);
                g.drawString("Selected text in name: "+
                                name.getSelectedText(), 50, 120);
                g.drawString("Password:"+pass.getText(), 50, 140);
        }
        public static void main(String [] args){
                TextFieldDemo obj1=new TextFieldDemo();
        }
}
```

L. TextArea

L.1. Important Constants

- **static int SCROLLBARS_BOTH** -- Create and display both vertical and horizontal scrollbars.
- **static int SCROLLBARS_HORIZONTAL_ONLY** -- Create and display horizontal scrollbar only.
- **static int SCROLLBARS_NONE** -- Do not create or display any scrollbars for the text area.
- **static int SCROLLBARS_VERTICAL_ONLY** -- Create and display vertical scrollbar only.

L.2. Important Constructors

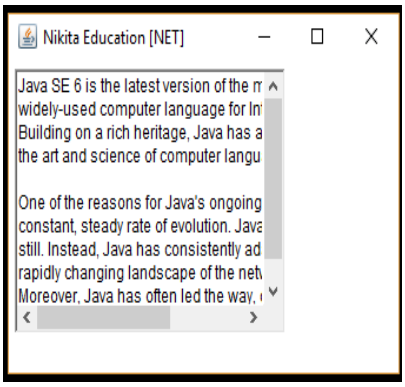
TextArea() Constructs a new text area with the empty string as text.
TextArea(int rows, int columns) Constructs a new text area with the specified number of rows and columns and the empty string as text.
TextArea(String text) Constructs a new text area with the specified text.
TextArea(String text, int rows, int columns) Constructs a new text area with the specified text, and with the specified number of rows and columns.
TextArea(String text, int rows, int columns, int scrollbars) Constructs a new text area with the specified text, and with the rows, columns, and scroll bar visibility as specified.

L.3. Important Methods

void append(String str) Appends the given text to the text area's current text.
int getColumns() Returns the number of columns in this text area.
int getRows() Returns the number of rows in the text area.

void insert(String str, int pos) Inserts the specified text at the specified position in this text area.
void replaceRange(String str, int start, int end) Replaces text between the indicated start and end positions with the specified replacement text.
void setColumns(int columns) Sets the number of columns for this text area.
void setRows(int rows) Sets the number of rows for this text area.

```
import java.awt.*;
import java.awt.event.*;
public class TextAreaDemo extends Frame{
        TextField name, pass;
        public TextAreaDemo(){
                setTitle("Nikita Education [NET]");
                setSize(500,500);
                setVisible(true);
                setLayout(new FlowLayout(FlowLayout.LEFT));

                String val =
"Java SE 6 is the latest version of the most\n" +
"widely-used computer language for Internet programming.\n" +
"Building on a rich heritage, Java has advanced both\n" +
"the art and science of computer language design.\n\n" +
"One of the reasons for Java's ongoing success is its\n" +
"constant, steady rate of evolution. Java has never stood\n" +
"still. Instead, Java has consistently adapted to the\n" +
"rapidly changing landscape of the networked world.\n" +
"Moreover, Java has often led the way, charting the\n" +
"course for others to follow.";

                TextArea text = new TextArea(val, 10, 30);
                add(text);
        }
        public static void main(String [] args)
        {
                TextAreaDemo obj1=new TextAreaDemo();
        }
}
```



M. Layout Manager

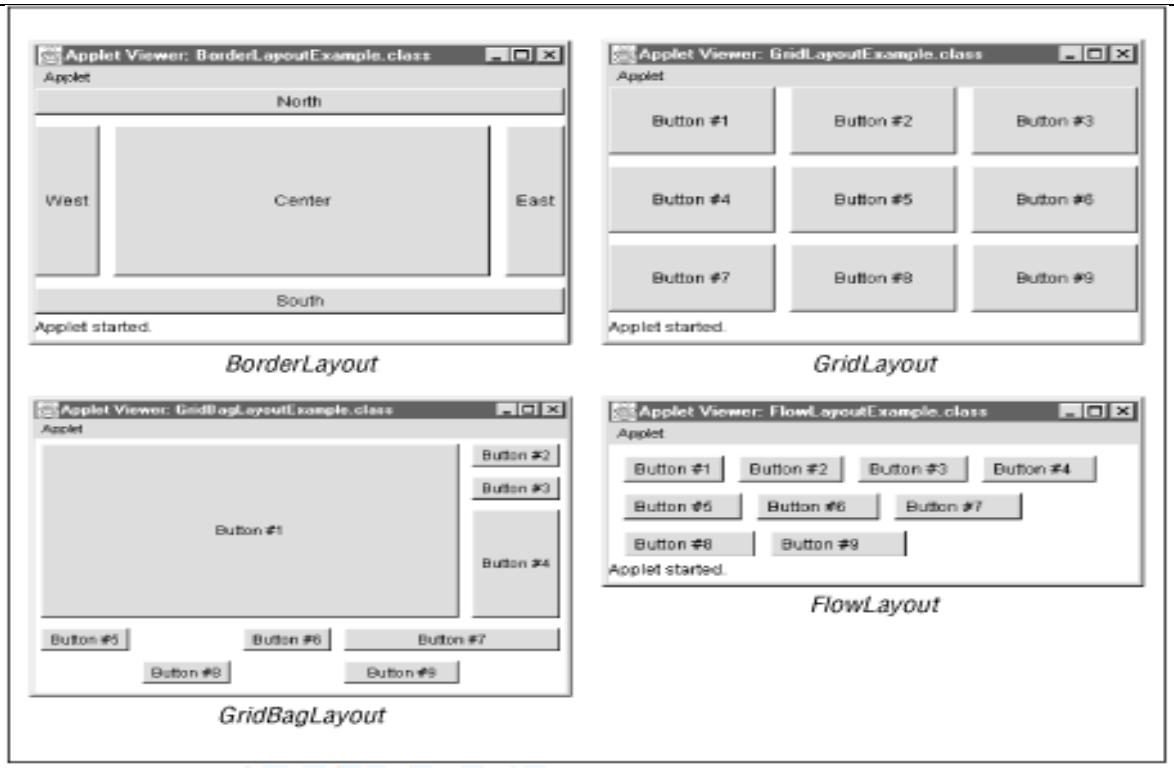
A layout manager **automatically arranges your controls** within a window by using some type of algorithm Sometimes device to device size information gets changed then manual placement of components become tedious. A layout manager is **an instance** of any class that implements the **LayoutManager** interface. The layout manager is set by the **setLayout( )** method.

void setLayout(LayoutManager *layoutObj*)

Layout Manager classes of java:

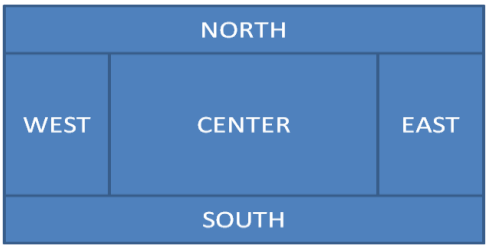
BorderLayout The borderlayout arranges the components to fit in the five regions: east, west, north, south and center.

FlowLayout The FlowLayout is the default layout.It layouts the components in a directional flow.
GridLayout The GridLayout manages the components in form of a rectangular grid.
CardLayout The CardLayout object treats each component in the container as a card. Only one card is visible at a time.
GridBagLayout This is the most flexible layout manager class. The object of GridBagLayout aligns the component vertically,horizontally or along their baseline without requiring the components of same size.



N. BorderLayout

BorderLayout is the default layout manager for Window means for **Frame.** The **BorderLayout** class implements a common layout style for top-level windows. It has four **narrow, fixed-width** components at the **edges** and one **large area** in the **center.** The four sides are referred to as **north, south, east, and west**. The middle area is called the **center**.



N.1. Important Constants

- **static String CENTER --** The center layout constraint (middle of container).
- **static String EAST --** The east layout constraint (right side of container).
- **static String NORTH --** The north layout constraint (top of container).
- **static String SOUTH** -- The south layout constraint (bottom of container).
- **static String WEST** -- The west layout constraint (left side of container).

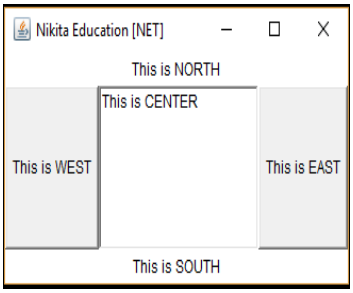
N.2. Important Constructors

BorderLayout(): Constructs a new border layout with no gaps between components.
BorderLayout(int hgap, int vgap): Constructs a border layout with the specified gaps between components.

N.2. Special form of add() method for BorderLayout

void add(Component *compObj*, Object *region*)

```
import java.awt.*;
import java.awt.event.*;
public class BorderLayoutDemo extends Frame implements ActionListener
{
	Label l1,l2;
	TextField t1;
	Button yes, no;
	String msg="";
	public BorderLayoutDemo(){
		setTitle("Nikita Education [NET]");
		setSize(500,500);
		setVisible(true);
		setLayout(new BorderLayout());
		l1=new Label("This is NORTH");
		l1.setAlignment(Label.CENTER);
		t1=new TextField("This is CENTER");
		l2=new Label("This is SOUTH");
		l2.setAlignment(Label.CENTER);
		yes = new Button("This is EAST");
		no = new Button("This is WEST");
		add(l1,BorderLayout.NORTH);
		add(t1,BorderLayout.CENTER);
		add(l2,BorderLayout.SOUTH);
		add(yes,BorderLayout.EAST);
		add(no,BorderLayout.WEST);
		yes.addActionListener(this);
		no.addActionListener(this);
	}
	public void actionPerformed(ActionEvent ae) {
		String str = ae.getActionCommand();
		if(str.equals("Yes")) {
			msg = "You pressed Yes.";
		}
		else if(str.equals("No")) {
			msg = "You pressed No.";
		}
		else {
			msg = "You pressed Undecided.";
		}
		repaint();
	}
	public void paint(Graphics g) {}
	public static void main(String [] args){
		BorderLayoutDemo obj1=new BorderLayoutDemo();
	}
}
```



O. FlowLayout

FlowLayout is the default layout manager for Panel means for **Applet. FlowLayout** implements a simple layout style, which is similar to how **words flow in a text editor.** Adds components from **left to right, top to bottom.** components are laid out **line-by-line** beginning at the upper-left corner. In all cases, when a line is filled, layout advances to the **next line.**

O.1. Important Constants

- **static int CENTER** -- This value indicates that each row of components should be centered.
- **static int LEADING** -- This value indicates that each row of components should be justified to the leading edge of the container's orientation, for example, to the left in left-to-right orientations.
- **static int LEFT** -- This value indicates that each row of components should be left-justified.
- **static int RIGHT** -- This value indicates that each row of components should be right-justified.
- **static int TRAILING** -- This value indicates that each row of components should be justified to the trailing edge of the container's orientation, for example, to the right in left-to-right orientations.

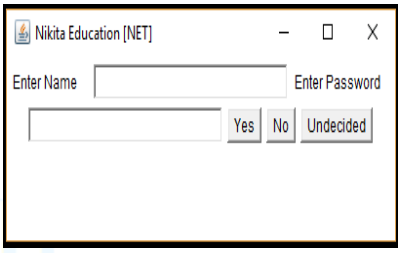
O.2. Important Constructors

FlowLayout()
Constructs a new FlowLayout with a centered alignment and a default 5-unit horizontal and vertical gap.

FlowLayout(int align)
Constructs a new FlowLayout with the specified alignment and a default 5-unit horizontal and vertical gap.

FlowLayout(int align, int hgap, int vgap)
Creates a new flow layout manager with the indicated alignment and the indicated horizontal and vertical gaps.

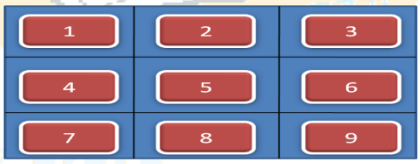
```
import java.awt.*;
import java.awt.event.*;
public class FlowLayoutDemo extends Frame implements ActionListener{
        Label l1,l2;
        TextField t1,t2;
        Button yes, no, maybe;
        String msg="";
        public FlowLayoutDemo(){
                setTitle("Nikita Education [NET]");
                setSize(500,500);
                setVisible(true);
                FlowLayout obj=new FlowLayout(FlowLayout.CENTER);
                setLayout(obj);
                l1=new Label("Enter Name");
                t1=new TextField(25);
                l2=new Label("Enter Password");
                t2=new TextField(25);
                yes = new Button("Yes");
                no = new Button("No");
                maybe = new Button("Undecided");
                add(l1);
                add(t1);
                add(l2);
                add(t2);
                add(yes);
                add(no);
                add(maybe);
                yes.addActionListener(this);
```



```
                no.addActionListener(this);
                maybe.addActionListener(this);
        }
        public void actionPerformed(ActionEvent ae) {
                String str = ae.getActionCommand();
                if(str.equals("Yes")){
                        msg = "You pressed Yes.";
                }
                else if(str.equals("No")) {
                        msg = "You pressed No.";
                }
                else {
                        msg = "You pressed Undecided.";
                }
                repaint();
        }
        public void paint(Graphics g) {
                g.drawString(msg, 100, 250);
        }
        public static void main(String [] args){
                FlowLayoutDemo obj1=new FlowLayoutDemo();
        }
}
```

P. GridLayout

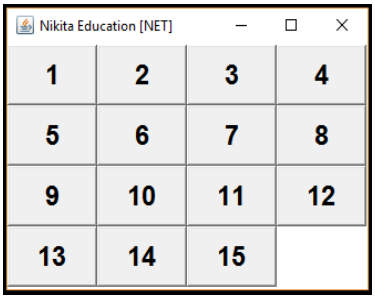
GridLayout lays out components in a two-dimensional grid. When you instantiate a **GridLayout**, you define the number of rows and columns. The class **GridLayout** arranges components in a rectangular grid.



P.1. Important Constructors

GridLayout() Creates a grid layout with a default of one column per component, in a single row.
GridLayout(int rows, int cols) Creates a grid layout with the specified number of rows and columns.
GridLayout(int rows, int cols, int hgap, int vgap) Creates a grid layout with the specified number of rows and columns.

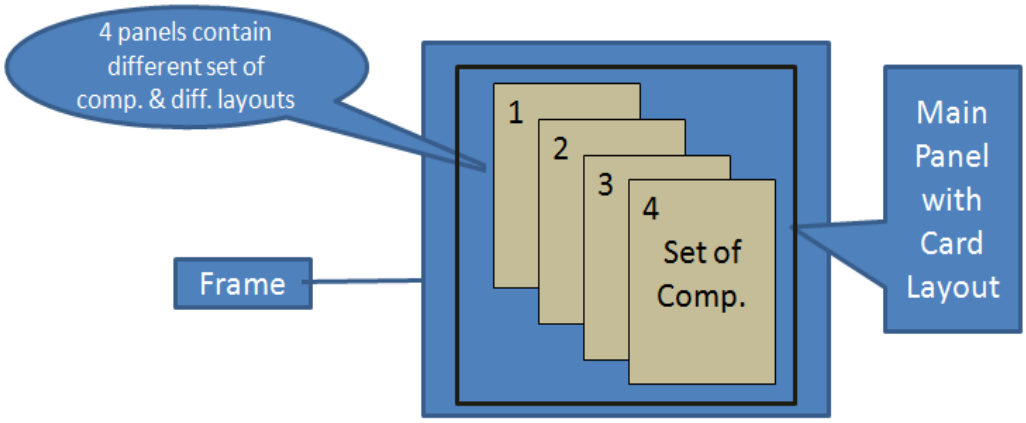
```
import java.awt.*;
import java.awt.event.*;
public class GridLayoutDemo extends Frame{
        public GridLayoutDemo(){
                setTitle("Nikita Education [NET]");
                setSize(500,500);
                setVisible(true);
                setLayout(new GridLayout(4,4));
                setFont(new Font("SansSerif", Font.BOLD, 24));
                for(int i = 0; i < 4; i++) {
                        for(int j = 0; j < 4; j++) {
                                int k = (i * 4) + j;
```



```
                        if(k > 0)
                                add(new Button("" + k));
                    }
                }
        }
        public static void main(String [] args){
                GridLayoutDemo obj1=new GridLayoutDemo();
        }
}
```

Q. CardLayout

The **CardLayout** class is unique among the other layout managers in that it **stores several different layouts.** Each layout can be thought of as being on **a separate index card** in a deck that can be shuffled so that any card is on top at a given time. This can be useful for user interfaces with optional components that can be dynamically enabled and disabled upon user input. You can prepare the other layouts and have them hidden, ready to be activated when needed.



Q.1. Important Constructors

CardLayout(): Creates a new card layout with gaps of size zero.
CardLayout(int hgap, int vgap) Creates a new card layout with the specified horizontal and vertical gaps.

Q.2. Important Methods

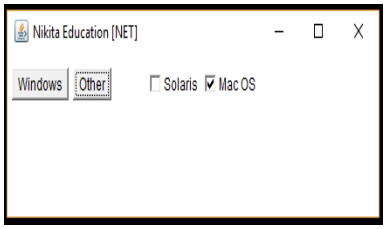
void first(Container parent): Flips to the first card of the container.
void last(Container parent) Flips to the last card of the container.
void next(Container parent) Flips to the next card of the specified container.
void previous(Container parent) Flips to the previous card of the specified container.
void show(Container parent, String name) Flips to the component that was added to this layout with the specified name, using addLayoutComponent.

Q.3. Special form of add() method for CardLayout

void add(Component *panelObj*, Object *name*)

```
import java.awt.*;
import java.awt.event.*;
public class CardLayoutDemo extends Frame implements ActionListener,
MouseListener{
        Checkbox winXP, winVista, solaris, mac;
        Panel osCards;
        CardLayout cardLO;
        Button Win, Other;
        public CardLayoutDemo(){
                setTitle("Nikita Education [NET]");
                setSize(500,500);
                setVisible(true);
                setLayout(new FlowLayout(FlowLayout.LEFT));
                Win = new Button("Windows");
                Other = new Button("Other");
                add(Win);
                add(Other);
                cardLO = new CardLayout();
                osCards = new Panel();
                osCards.setLayout(cardLO);
                // set panel layout to card layout
                winXP = new Checkbox("Windows XP", null, true);
                winVista = new Checkbox("Windows Vista");
                solaris = new Checkbox("Solaris");
                mac = new Checkbox("Mac OS");
                // add Windows check boxes to a panel
                Panel winPan = new Panel();
                winPan.add(winXP);
                winPan.add(winVista);
                // add other OS check boxes to a panel
                Panel otherPan = new Panel();
                otherPan.add(solaris);
                otherPan.add(mac);
                // add panels to card deck panel
                osCards.add(winPan, "Windows");
                osCards.add(otherPan, "Other");
                // add cards to main applet panel
                add(osCards);
                // register to receive action events
                Win.addActionListener(this);
                Other.addActionListener(this);
                // register mouse events
                addMouseListener(this);
        }
        public void mousePressed(MouseEvent me) {
                cardLO.next(osCards);
        }
// Provide empty implementations for the other MouseListener methods.
        public void mouseClicked(MouseEvent me) {}
        public void mouseEntered(MouseEvent me) {}
        public void mouseExited(MouseEvent me) {}
        public void mouseReleased(MouseEvent me) {}
        public void actionPerformed(ActionEvent ae) {
                if(ae.getSource().equals(Win)) {
                        cardLO.show(osCards, "Windows");
                }
                else {
                        cardLO.show(osCards, "Other");
```

Clicked on windows button



Clicked on other button

```
        }
    }
    public static void main(String [] args){
        CardLayoutDemo obj1=new CardLayoutDemo();
    }
}
```

R. GridBagLayout

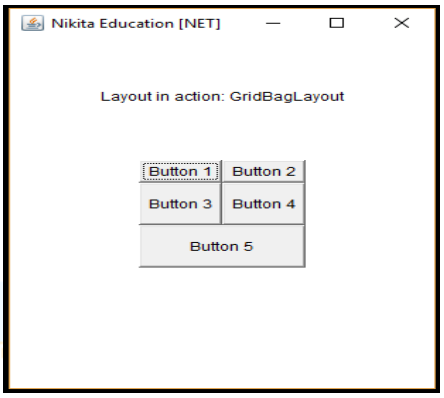
The class **GridBagLayout** arranges components in a horizontal and vertical manner.

GridBagLayout

R.1. Important Constructors

GridBagLayout()
Creates a grid bag layout manager.

```
import java.awt.*;
import java.awt.event.*;
public class AwtLayoutDemo {
   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;
   private Label msglabel;
   public AwtLayoutDemo(){
      prepareGUI();
   }
   public static void main(String[] args){
      AwtLayoutDemo  awtLayoutDemo = new AwtLayoutDemo();
      awtLayoutDemo.showGridBagLayoutDemo();
   }
   private void prepareGUI(){
      mainFrame = new Frame("Nikita Education [NET]");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));
      headerLabel = new Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);
      msglabel = new Label();
      msglabel.setAlignment(Label.CENTER);
      msglabel.setText("Welcome to TutorialsPoint AWT Tutorial.");
      controlPanel = new Panel();
      controlPanel.setLayout(new FlowLayout());
      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
```



```
    mainFrame.add(statusLabel);
    mainFrame.setVisible(true);
  }
  private void showGridBagLayoutDemo(){
    headerLabel.setText("Layout in action: GridBagLayout");
    Panel panel = new Panel();
    panel.setBackground(Color.darkGray);
    panel.setSize(300,300);
    GridBagLayout layout = new GridBagLayout();
    panel.setLayout(layout);
    GridBagConstraints gbc = new GridBagConstraints();
    gbc.fill = GridBagConstraints.HORIZONTAL;
    gbc.gridx = 0;
    gbc.gridy = 0;
    panel.add(new Button("Button 1"),gbc);
    gbc.gridx = 1;
    gbc.gridy = 0;
    panel.add(new Button("Button 2"),gbc);
    gbc.fill = GridBagConstraints.HORIZONTAL;
    gbc.ipady = 20;
    gbc.gridx = 0;
    gbc.gridy = 1;
    panel.add(new Button("Button 3"),gbc);
    gbc.gridx = 1;
    gbc.gridy = 1;
    panel.add(new Button("Button 4"),gbc);
    gbc.gridx = 0;
    gbc.gridy = 2;
    gbc.fill = GridBagConstraints.HORIZONTAL;
    gbc.gridwidth = 2;
    panel.add(new Button("Button 5"),gbc);
    controlPanel.add(panel);
    mainFrame.setVisible(true);
  }
}
```

S. Menu based application

A top-level window can have **a menu bar** associated with it. A menu bar displays a list of **top-level menu choices**. Each choice is associated with a drop-down menu. This concept is implemented in the AWT by the following classes: **MenuBar**, **Menu**, and **MenuItem**. In general, a menu bar contains one or more **Menu** objects. Each **Menu** object contains a list of **MenuItem** objects. Each **MenuItem** object represents something that can be **selected by the user.** To create menu based application using above classes perform following steps:

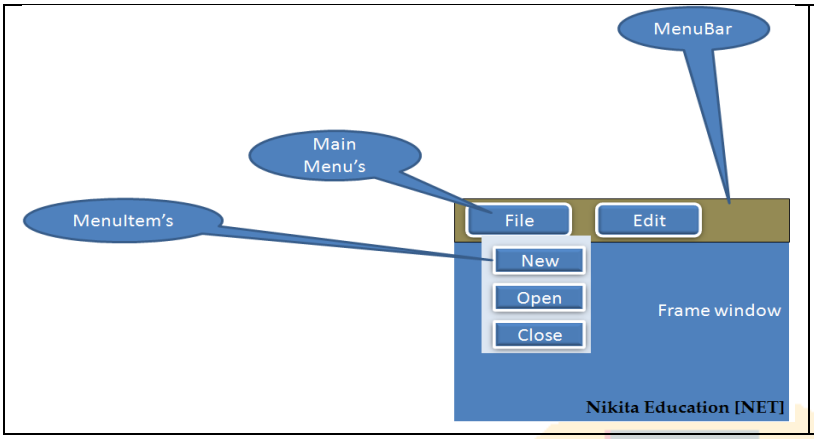
- Create instance of **MenuBar**
- Set menubar on container i.e. Frame using **setMenuBar(MenuBar obj)**
- Create instances of top level menus using **Menu** class
- Create instances of menu items using **MenuItem** class
- Add MenuItem into the Menu using **add(MenuItem obj)**
- Add Menu into MenuBar using **add(MenuItem obj)**

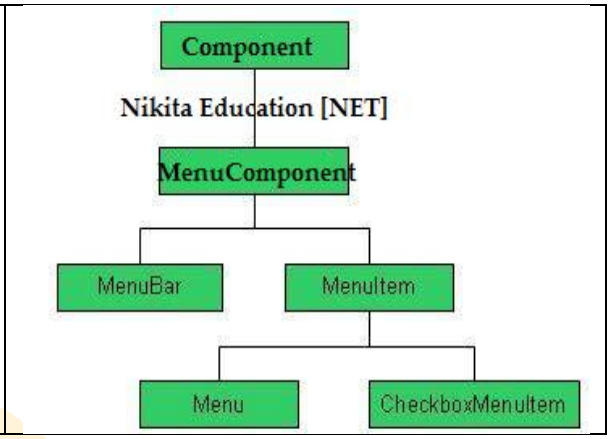
MenuComponent It is the top level class for all menu related controls.
MenuBar: The MenuBar object is associated with the top-level window.

MenuItem
The items in the menu must belong to the MenuItem or any of its subclass.

Menu
The Menu object is a pull-down menu component which is displayed from the menu bar.

CheckboxMenuItem
CheckboxMenuItem is subclass of MenuItem.





S.1. MenuBar

The MenuBar class provides menu bar bound to a frame and is platform specific.

S.1.1. Constructors

MenuBar()
Creates a new menu bar.

S.1.2. Methods

Menu add(Menu m)
Adds the specified menu to the menu bar.

Menu getMenu(int i)
Gets the specified menu.

int getMenuCount()
Gets the number of menus on the menu bar.

void remove(int index)
Removes the menu located at the specified index from this menu bar.

void remove(MenuComponent m)
Removes the specified menu component from this menu bar.

void setHelpMenu(Menu m)
Sets the specified menu to be this menu bar's help menu.

S.2. Menu

The Menu class represents pull-down menu component which is deployed from a menu bar.

S.2.1. Constructors

Menu(): Constructs a new menu with an empty label.

Menu(String label) Constructs a new menu with the specified label.
Menu(String label, boolean tearOff) Constructs a new menu with the specified label, indicating whether the menu can be torn off.

S.2.2. Methods

MenuItem add(MenuItem mi) Adds the specified menu item to this menu.
void add(String label) Adds an item with the specified label to this menu.
void addSeparator() Adds a separator line, or a hypen, to the menu at the current position.
MenuItem getItem(int index) Gets the item located at the specified index of this menu.
int getItemCount() Get the number of items in this menu.
void insert(MenuItem menuitem, int index) Inserts a menu item into this menu at the specified position.
void insert(String label, int index) Inserts a menu item with the specified label into this menu at the specified position.
void insertSeparator(int index) Inserts a separator at the specified position.
void remove(int index) Removes the menu item at the specified index from this menu.
void remove(MenuComponent item) Removes the specified menu item from this menu.
void removeAll() Removes all items from this menu.

S.3. MenuItem

The MenuBar class represents the actual item in a menu. All items in a menu should derive from class MenuItem, or one of its subclasses. By default, it embodies a simple labeled menu item.

S.3.1. Constructors

MenuItem() Constructs a new MenuItem with an empty label and no keyboard shortcut.
MenuItem(String label) Constructs a new MenuItem with the specified label and no keyboard shortcut.
MenuItem(String label, MenuShortcut s) Create a menu item with an associated keyboard shortcut.

S.3.2. Methods

void addActionListener(ActionListener l) Adds the specified action listener to receive action events from this menu item.
void deleteShortcut() Delete any MenuShortcut object associated with this menu item.
String getActionCommand() Gets the command name of the action event that is fired by this menu item.
String getLabel() Gets the label for this menu item.
boolean isEnabled() Checks whether this menu item is enabled.
void setActionCommand(String command) Sets the command name of the action event that is fired by this menu item.
void setEnabled(boolean b) Sets whether or not this menu item can be chosen.
void setLabel(String label) Sets the label for this menu item to the specified label.
void setShortcut(MenuShortcut s) Set the MenuShortcut object associated with this menu item.

S.4. CheckboxMenuItem

The CheckboxMenuItem class represents a check box which can be included in a menu. Selecting the check box in the menu changes control's state from **on** to **off** or from **off** to **on**.

S.4.1. Constructors

CheckboxMenuItem(label) Create a check box menu item with the specified label.
CheckboxMenuItem(label, boolean state) Create a check box menu item with the specified label and state.

S.4.2. Methods

void addItemListener(ItemListener l) Adds the specified item listener to receive item events from this check box menu item.
boolean getState(): Determines whether the state of this check box menu item is "on" or "off."
void setState(boolean b) Sets this check box menu item to the specifed state.

```
import java.awt.*;
import java.awt.event.*;
class MyMenuHandler implements ActionListener, ItemListener{
        MenuDemo menuFrame;
        public MyMenuHandler(MenuDemo menuFrame){
                this.menuFrame = menuFrame;
        }
```

```
        // Handle action events.
        public void actionPerformed(ActionEvent ae){
                String msg = "You selected ";
                String arg = ae.getActionCommand();
                if(arg.equals("New....."))
                        msg += "New.";
                else if(arg.equals("Open....."))
                        msg += "Open.";
                else if(arg.equals("Close....."))
                        msg += "Close.";
                else if(arg.equals("Quit....."))
                        msg += "Quit.";
                else if(arg.equals("Edit"))
                        msg += "Edit.";
                else if(arg.equals("Cut"))
                        msg += "Cut.";
                else if(arg.equals("Copy"))
                        msg += "Copy.";
                else if(arg.equals("Paste"))
                        msg += "Paste.";
                else if(arg.equals("First"))
                        msg += "First.";
                else if(arg.equals("Second"))
                        msg += "Second.";
                else if(arg.equals("Third"))
                        msg += "Third.";
                else if(arg.equals("Debug"))
                        msg += "Debug.";
                else if(arg.equals("Testing"))
                        msg += "Testing.";
                menuFrame.msg = msg;
                menuFrame.repaint();
        }
        // Handle item events.
        public void itemStateChanged(ItemEvent ie){
                menuFrame.repaint();
        }
}
```

```
import java.awt.*;
import java.awt.event.*;
public class MenuDemo extends Frame{
        String msg = "";
        MenuBar mbar;
        Menu file, edit, sub;
        MenuItem item1, item2, item3, item4, item5;
        MenuItem item6, item7, item8, item9;
        MenuItem item10, item11, item12;
        CheckboxMenuItem debug, test;
        public MenuDemo(){
                int n=4;
                setTitle("Nikita Education [NET]");
                setSize(500,500);
                setVisible(true);
                setLayout(new FlowLayout(FlowLayout.LEFT));

                mbar=new MenuBar();
                setMenuBar(mbar);
```

```
        file=new Menu("File");
        item1=new MenuItem("New.....");
        file.add(item1);
        item2=new MenuItem("Open.....");
        file.add(item2);
        item3=new MenuItem("Close.....");
        file.add(item3);
        item4=new MenuItem("-");
        file.add(item4);
        item5=new MenuItem("Quit.....");
        file.add(item5);
        mbar.add(file);

        edit=new Menu("Edit");
        item6=new MenuItem("Cut");
        edit.add(item6);
        item7=new MenuItem("Copy");
        edit.add(item7);
        item8=new MenuItem("Paste");
        edit.add(item8);
        item9=new MenuItem("-");
        edit.add(item9);
        sub=new Menu("Special");
        item10=new MenuItem("First");
        sub.add(item10);
        item11=new MenuItem("Second");
        sub.add(item11);
        item12=new MenuItem("Third");
        sub.add(item12);
        edit.add(sub);

        debug = new CheckboxMenuItem("Debug");
        edit.add(debug);
        test = new CheckboxMenuItem("Testing");
        edit.add(test);
        mbar.add(edit);

        // create an object to handle action and item events
        MyMenuHandler handler = new MyMenuHandler(this);
        // register it to receive those events
        item1.addActionListener(handler);
        item2.addActionListener(handler);
        item3.addActionListener(handler);
        item4.addActionListener(handler);
        item5.addActionListener(handler);
        item6.addActionListener(handler);
        item7.addActionListener(handler);
        item8.addActionListener(handler);
        item9.addActionListener(handler);
        item10.addActionListener(handler);
        item11.addActionListener(handler);
        item12.addActionListener(handler);
        debug.addItemListener(handler);
        test.addItemListener(handler);
    }
    public void paint(Graphics g) {
        g.drawString(msg, 50, 200);
```

```
                if(debug.getState())
                        g.drawString("Debug is on.", 50, 220);
                else
                        g.drawString("Debug is off.", 50, 220);
                if(test.getState())
                        g.drawString("Testing is on.", 50, 240);
                else
                        g.drawString("Testing is off.", 50, 240);
        }
        public static void main(String [] args){
                MenuDemo obj1=new MenuDemo();
        }
}
```

T. Dialog and FileDialog

T.1. Dialog

Dialog boxes are primarily used to obtain user input and are often child windows of a top-level window. Dialog boxes don't have menu bars. **Dialog boxes may be modal or modeless**. When a ***modal*** dialog box is active, all input is directed to it until it is closed. This means that you cannot access other parts of your program until you have closed the dialog box. When a ***modeless*** dialog box is active, input focus can be directed to another window in your program. Thus, other parts of your program remain active and accessible. Dialog boxes are of type **Dialog**. It is used to take some form of input from the user. It inherits the Window class. Unlike Frame, it doesn't have maximize and minimize buttons.

T.1.1. Constructors

Dialog(Frame owner) Constructs an initially invisible, modeless Dialog with the specified owner Frame and an empty title.
Dialog(Frame owner, boolean modal) Constructs an initially invisible Dialog with the specified owner Frame and modality and an empty title.
Dialog(Frame owner, String title) Constructs an initially invisible, modeless Dialog with the specified owner Frame and title.
Dialog(Frame owner, String title, boolean modal) Constructs an initially invisible Dialog with the specified owner Frame, title and modality.

T.1.2. Methods

String getTitle() Gets the title of the dialog.
boolean isModal() Indicates whether the dialog is modal.
void setModal(boolean modal): Specifies whether this dialog should be modal.
void setTitle(String title): Sets the title of the Dialog.
void setVisible(boolean b): Shows or hides this Dialog depending on the value of parameter b.

T.2. FileDialog

Java provides a built-in dialog box that lets the user specify a file. To create a file dialog box, instantiate an object of type **FileDialog**. This causes a file dialog box to be displayed. Usually, this is the standard file dialog box provided by the operating system. FileDialog control represents a dialog window from which the user can select a file.

T.2.1. Constants

- **static int LOAD --** This constant value indicates that the purpose of the file dialog window is to locate a file from which to read.
- **static int SAVE --** This constant value indicates that the purpose of the file dialog window is to locate a file to which to write.

T.2.2. Constructors

FileDialog(Frame parent) Creates a file dialog for loading a file.
FileDialog(Frame parent, String title) Creates a file dialog window with the specified title for loading a file.
FileDialog(Frame parent, String title, int mode) Creates a file dialog window with the specified title for loading or saving a file.

T.2.2. Methods

String getDirectory() Gets the directory of this file dialog.
String getFile() Gets the selected file of this file dialog.
int getMode() Indicates whether this file dialog box is for loading from a file or for saving to a file.
void setDirectory(String dir) Sets the directory of this file dialog window to be the specified directory.
void setFile(String file) Sets the selected file for this file dialog window to be the specified file.
void setFilenameFilter(FilenameFilter filter) Sets the filename filter for this file dialog window to the specified filter.
void setMode(int mode) Sets the mode of the file dialog.

```
//Program 1: MenuDemo.java
import java.awt.*;
import java.awt.event.*;
public class MenuDemo extends Frame{
        String msg = "";
        MenuBar mbar;
        Menu file, edit, sub;
        MenuItem item1, item2, item3, item4, item5;
        MenuItem item6, item7, item8, item9;
```

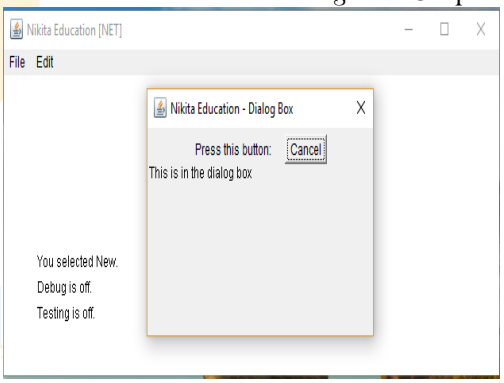
```
MenuItem item10, item11, item12;
CheckboxMenuItem debug, test;
public MenuDemo(){
        int n=4;
        setTitle("Nikita Education [NET]");
        setSize(500,500);
        setVisible(true);
        setLayout(new FlowLayout(FlowLayout.LEFT));

        mbar=new MenuBar();
        setMenuBar(mbar);

        file=new Menu("File");
        item1=new MenuItem("New.....");
        file.add(item1);
        item2=new MenuItem("Open.....");
        file.add(item2);
        item3=new MenuItem("Close.....");
        file.add(item3);
        item4=new MenuItem("-");
        file.add(item4);
        item5=new MenuItem("Quit.....");
        file.add(item5);
        mbar.add(file);

        edit=new Menu("Edit");
        item6=new MenuItem("Cut");
        edit.add(item6);
        item7=new MenuItem("Copy");
        edit.add(item7);
        item8=new MenuItem("Paste");
        edit.add(item8);
        item9=new MenuItem("-");
        edit.add(item9);

        sub=new Menu("Special");
        item10=new MenuItem("First");
        sub.add(item10);
        item11=new MenuItem("Second");
        sub.add(item11);
        item12=new MenuItem("Third");
        sub.add(item12);
        edit.add(sub);

        debug = new CheckboxMenuItem("Debug");
        edit.add(debug);
        test = new CheckboxMenuItem("Testing");
        edit.add(test);
        mbar.add(edit);

        // create an object to handle action and item events
        MyMenuHandler handler = new MyMenuHandler(this);

        // register it to receive those events
        item1.addActionListener(handler);
        item2.addActionListener(handler);
        item3.addActionListener(handler);
        item4.addActionListener(handler);
```

```
                item5.addActionListener(handler);
                item6.addActionListener(handler);
                item7.addActionListener(handler);
                item8.addActionListener(handler);
                item9.addActionListener(handler);
                item10.addActionListener(handler);
                item11.addActionListener(handler);
                item12.addActionListener(handler);
                debug.addItemListener(handler);
                test.addItemListener(handler);
        }
        public void paint(Graphics g) {
                g.drawString(msg, 50, 200);
                if(debug.getState())
                        g.drawString("Debug is on.", 50, 220);
                else
                        g.drawString("Debug is off.", 50, 220);
                if(test.getState())
                        g.drawString("Testing is on.", 50, 240);
                else
                        g.drawString("Testing is off.", 50, 240);
        }
        public static void main(String [] args){
                MenuDemo obj1=new MenuDemo();
        }
}
```

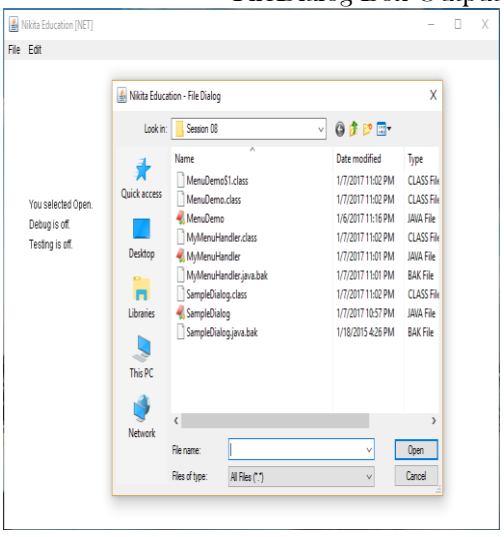
//Program 2: MyMenuHandler.java

```
import java.awt.*;
import java.awt.event.*;
class MyMenuHandler implements ActionListener, ItemListener
{
        MenuDemo menuFrame;
        public MyMenuHandler(MenuDemo menuFrame)
        {
                this.menuFrame = menuFrame;
        }
        public void actionPerformed(ActionEvent ae)
        {
                String msg = "You selected ";
                String arg = ae.getActionCommand();
                if(arg.equals("New....."))
                {
                        msg += "New.";
SampleDialog d = new SampleDialog(menuFrame, "Nikita Edu");
                        d.setVisible(true);
                }
                else if(arg.equals("Open....."))
                {
                        msg += "Open.";
FileDialog fd = new FileDialog(menuFrame, "Nikita Edu");
                        fd.setVisible(true);
                }
                else if(arg.equals("Close....."))
                        msg += "Close.";
                else if(arg.equals("Quit....."))
                        msg += "Quit.";
                else if(arg.equals("Edit"))
```

Dialog Box Output:



FileDialog Box Output



```
                msg += "Edit.";
            else if(arg.equals("Cut"))
                msg += "Cut.";

            else if(arg.equals("Copy"))
                msg += "Copy.";
            else if(arg.equals("Paste"))
                msg += "Paste.";
            else if(arg.equals("First"))
                msg += "First.";
            else if(arg.equals("Second"))
                msg += "Second.";
            else if(arg.equals("Third"))
                msg += "Third.";
            else if(arg.equals("Debug"))
                msg += "Debug.";
            else if(arg.equals("Testing"))
                msg += "Testing.";
                menuFrame.msg = msg;
                menuFrame.repaint();
        }
        public void itemStateChanged(ItemEvent ie)
        {
            menuFrame.repaint();
        }
}
```

//Program 3: SampleDialog.java

```
import java.awt.*;
import java.awt.event.*;
class SampleDialog extends Dialog implements ActionListener
{
        SampleDialog(Frame parent, String title)
        {
            super(parent, title, false);
            setLayout(new FlowLayout());
            setSize(300, 200);
            add(new Label("Press this button:"));
            Button b;
            add(b = new Button("Cancel"));
            b.addActionListener(this);
        }
        public void actionPerformed(ActionEvent ae)
        {
            dispose();
        }
        public void paint(Graphics g)
        {
            g.drawString("This is in the dialog box", 10, 70);
        }
}
```

SWING

1. Set of **API** (API – Set of Classes and Interfaces)
2. Provided to design a **Graphical User Interface**
3. An **extension library** to the AWT
4. Includes new and improved components that enhance the **look and functionality** of GUIs
5. Swing can be used to build **Standalone** swing GUI Apps as well as **Servlets and Applets.**
6. It employs **model/view** design architecture.
7. Swing is more **portable** and more **flexible** than AWT, Swing is **built on top of AWT**
8. **Entirely written in Java** (means do not use OS resources)

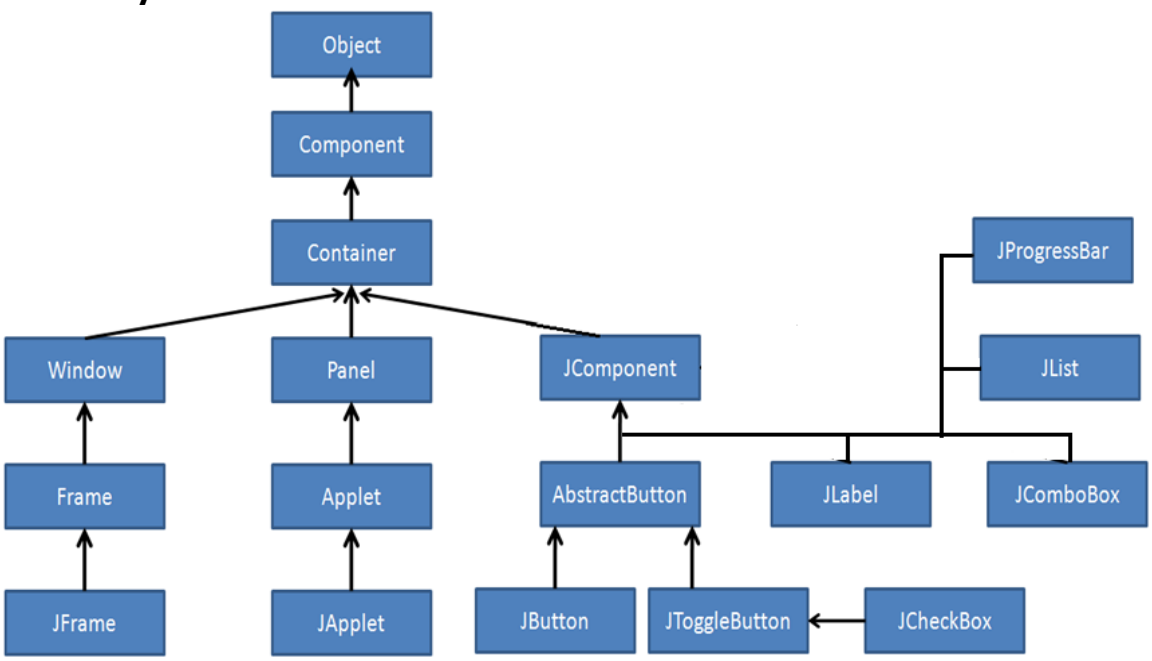
Swing features:

- Pluggable look & feel
- Uses MVC architecture
- Lightweight components
- Platform Independent
- Advance features such as JTable, JTabbedPane, JScrollPane etc

Difference between AWT and Swing

No.	Java AWT	Java Swing
1)	AWT components are **platform-dependent**.	Java swing components are **platform-independent**.
2)	AWT components are **heavyweight**.	Swing components are **lightweight**.
3)	AWT **doesn't support pluggable look and feel**.	Swing **supports pluggable look and feel**.
4)	AWT provides **less components** than Swing.	Swing provides **more powerful components** such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5)	AWT **doesn't follows MVC**(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing **follows MVC**.

Classes Hierarchy



The MVC Connection

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

By separating a component into a model, a view, and a controller, the specific implementation of each can be changed without affecting the other two.

Class & Description
Component A Component is the abstract base class for the non menu user-interface controls of SWING. Component represents an object with graphical representation
Container A Container is a component that can contain other SWING components.
JComponent A JComponent is a base class for all swing UI components. In order to use a swing component that inherits from JComponent, component must be in a containment hierarchy whose root is a top-level Swing container.
JLabel A JLabel object is a component for placing text in a container.
JButton This class creates a labeled button.
JColorChooser A JColorChooser provides a pane of controls designed to allow a user to manipulate and select a color.
JCheck Box A JCheckBox is a graphical component that can be in either an **on** (true) or **off** (false) state.
JRadioButton The JRadioButton class is a graphical component that can be in either an **on** (true) or **off** (false) state. in a group.
JList A JList component presents the user with a scrolling list of text items.
JComboBox A JComboBox component presents the user with a to show up menu of choices.
JTextField A JTextField object is a text component that allows for the editing of a single line of text.
JPasswordField: A JPasswordField object is a text component specialized for password entry.

JTextArea A JTextArea object is a text component that allows for the editing of a multiple lines of text.
ImageIcon A ImageIcon control is an implementation of the Icon interface that paints Icons from Images
JScrollbar A Scrollbar control represents a scroll bar component in order to enable user to select from range of values.
JOptionPane JOptionPane provides set of standard dialog boxes that prompt users for a value or informs them of something.
JFileChooser A JFileChooser control represents a dialog window from which the user can select a file.
JProgressBar As the task progresses towards completion, the progress bar displays the task's percentage of completion.
JSlider A JSlider lets the user graphically select a value by sliding a knob within a bounded interval.
JSpinner A JSpinner is a single line input field that lets the user select a number or an object value from an ordered sequence.

A. JFrame

Important Constants

static int EXIT_ON_CLOSE -- The exit application default window close operation.
static int DISPOSE_ON_CLOSE -- The dispose-window default window close operation.
static int DO_NOTHING_ON_CLOSE -- The do-nothing default window close operation.
static int HIDE_ON_CLOSE -- The hide-window default window close operation

Important Constructors

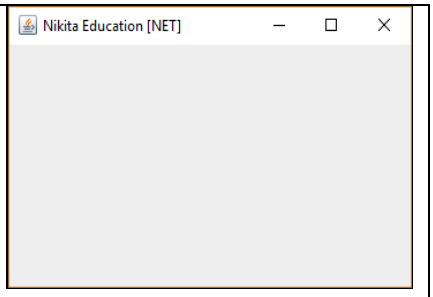
JFrame() Constructs a new frame that is initially invisible.
JFrame(GraphicsConfiguration gc) Creates a Frame in the specified GraphicsConfiguration of a screen device and a blank title**.**
JFrame(String title) Creates a new, initially invisible Frame with the specified title.

Important Methods

void setDefaultCloseOperation(int operation) Sets the operation that will happen by default when the user initiates a "close" on this frame.
void setJMenuBar(JMenuBar menubar): Sets the menubar for this frame.
void setLayout(LayoutManager manager): Sets the LayoutManager.

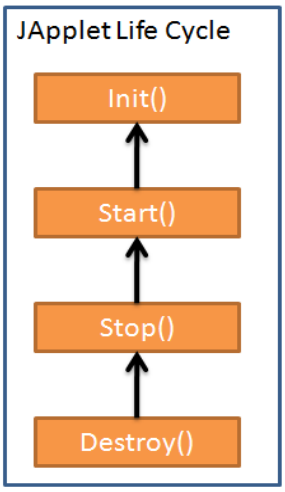
public void add(Component c) add a component on another component.
public void setSize(int width,int height) sets size of the container.
public void setVisible(boolean b) sets the visibility of the component. It is by default false.
public void setTitle(String title) Set the title for frame container.

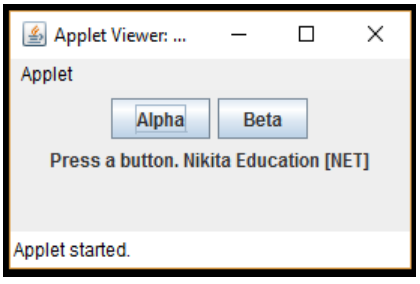
```
import java.awt.*;
import javax.swing.*;
public class MySwingFrame extends JFrame
{
	public MySwingFrame()
	{
		setTitle("Nikita Education [NET]");
		setSize(800,600);
		setVisible(true);
		setLayout(new FlowLayout(FlowLayout.LEFT));
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	}
	public static void main(String[] args) {
		MySwingFrame newFrame=new MySwingFrame();
	}
}
```



B. JApplet

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





```
                add(jbtnAlpha);
                add(jbtnBeta);
                jlab = new JLabel("Press a button. Nikita Education");
                add(jlab);
        }
}
```

Swing Controls:

JButton	JCheckBox	JComboBox	JLabel
JList	JRadioButton	JScrollPane	JTabbedPane
JTable	JTextField	JToggleButton	JTree

Important constants of SwingConstants interface for all swing components

static int	**BOTTOM** -- Box-orientation constant used to specify the bottom of a box.
static int	**CENTER** -- The central position in an area.
static int	**HORIZONTAL** -- Horizontal orientation.
static int	**LEADING** -- Identifies the leading edge of text for use with left-to-right and right-to-left languages.
static int	**LEFT** -- Box-orientation constant used to specify the left side of a box.
static int	**RIGHT** -- Box-orientation constant used to specify the right side of a box.
static int	**TRAILING** -- Identifies the trailing edge of text for use with left-to-right and right-to-left languages.
static int	**VERTICAL** -- Vertical orientation.

C. JLabel and ImageIcon

Important Constructors of JLabel

JLabel() Creates a JLabel instance with no image and with an empty string for the title.
JLabel(Icon image) Creates a JLabel instance with the specified image.
JLabel(String text) Creates a JLabel instance with the specified text.
JLabel(String text, Icon icon, int horizontalAlignment) Creates a JLabel instance with the specified text, image, and horizontal alignment.

Important Methods of JLabel

Icon getDisabledIcon() Returns the icon used by the label when it's disabled.
int getHorizontalAlignment() Returns the alignment of the label's contents along the X axis.
Icon getIcon() Returns the graphic image (glyph, icon) that the label displays.
String getText() Returns the text string that the label displays.

int getVerticalAlignment() Returns the alignment of the label's contents along the Y axis.
void setHorizontalAlignment(int alignment) Sets the alignment of the label's contents along the X axis.
void setIcon(Icon icon) Defines the icon this component will display.
void setText(String text) Defines the single line of text this component will display.
void setVerticalAlignment(int alignment) Sets the alignment of the label's contents along the Y axis.

Important Constructors of ImageIcon

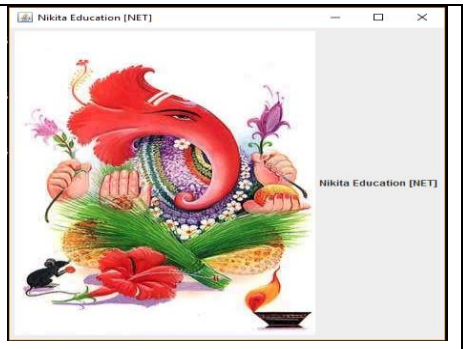
ImageIcon() Creates an uninitialized image icon.
ImageIcon(Image image) Creates an ImageIcon from an image object.
ImageIcon(Image image, String description) Creates an ImageIcon from the image.
ImageIcon(String filename) Creates an ImageIcon from the specified file.
ImageIcon(String filename, String description) Creates an ImageIcon from the specified file.

Important Methods of ImageIcon

String getDescription() Gets the description of the image.
int getIconHeight() Gets the height of the icon.
int getIconWidth() Gets the width of the icon.
Image getImage() Returns this icon's Image.
void setDescription(String description) Sets the description of the image.
void setImage(Image image) Sets the image displayed by this icon.

import java.awt.*; import java.awt.event.*; import javax.swing.*; import javax.swing.event.*; public class JLabelDemo extends JFrame{	

```
        public JLabelDemo(){
                setTitle("Nikita Education [NET]");
                setSize(800,600);
                setVisible(true);
                setLayout(new FlowLayout(FlowLayout.LEFT));
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

                ImageIcon ii = new ImageIcon("ganesh.jpg");
JLabel jl = new JLabel("Nikita Education [NET]", ii, JLabel.CENTER);
                add(jl);
        }
        public static void main(String[] args) {
                JLabelDemo newFrame=new JLabelDemo();
        }
}
```



D. JTextField

Important Constructors

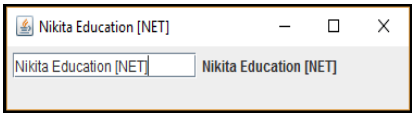
JTextField() Constructs a new TextField.
JTextField(int columns) Constructs a new empty TextField with the specified number of columns.
JTextField(String text) Constructs a new TextField initialized with the specified text.
JTextField(String text, int columns) Constructs a new TextField initialized with the specified text and columns.

Important Methods

void addActionListener(ActionListener l) Adds the specified action listener to receive action events from this textfield.
int getColumns() Returns the number of columns in this TextField.
int getHorizontalAlignment() Returns the horizontal alignment of the text.
void setActionCommand(String command) Sets the command string used for action events.
void setColumns(int columns) Sets the number of columns in this TextField, and then invalidate the layout.
void setFont(Font f): Sets the current font.
void setHorizontalAlignment(int alignment) Sets the horizontal alignment of the text.
void setScrollOffset(int scrollOffset) Sets the scroll offset, in pixels.
String getSelectedText(): Returns the selected text contained in this TextComponent.

String getText() Returns the text contained in this TextComponent.
void select(int selectionStart, int selectionEnd) Selects the text between the specified start and end positions.
void selectAll() Selects all the text in the TextComponent
void setText(String t) Sets the text of this TextComponent to the specified text.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
public class JTextFieldDemo extends JFrame{
	JTextField jtf;
	JLabel lb;
	String msg="";
	public JTextFieldDemo(){
		setTitle("Nikita Education [NET]");
		setSize(800,600);
		setVisible(true);
		setLayout(new FlowLayout(FlowLayout.LEFT));
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		jtf = new JTextField("Nikita Education [NET]",15);
		add(jtf);
		jtf.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent ae) {
				lb.setText(jtf.getText());
			}
		});
		lb=new JLabel();
		add(lb);
	}
	public static void main(String[] args) {
		JTextFieldDemo newFrame=new JTextFieldDemo();
	}
}
```



E. JTextArea

The object of a JTextArea class is a multi line region that displays text. It allows the editing of multiple line text. It inherits JTextComponent class.

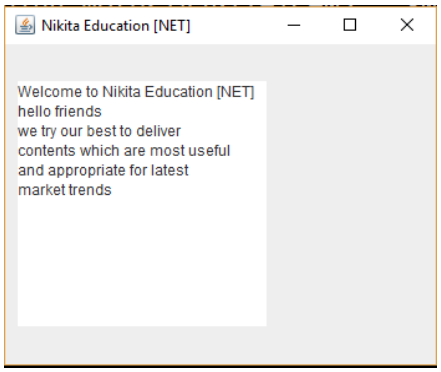
Commonly used Constructors:

Constructor	Description
JTextArea()	Creates a text area that displays no text initially.
JTextArea(String s)	Creates a text area that displays specified text initially.
JTextArea(int row, int column)	Creates a text area with the specified number of rows and columns that displays no text initially.
JTextArea(String s, int row, int column)	Creates a text area with the specified number of rows and columns that displays specified text.

Commonly used Methods:

Methods	Description
void setRows(int rows)	It is used to set specified number of rows.
void setColumns(int cols)	It is used to set specified number of columns.
void setFont(Font f)	It is used to set the specified font.
void insert(String s, int position)	It is used to insert the specified text on the specified position.
void append(String s)	It is used to append the given text to the end of the document.

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

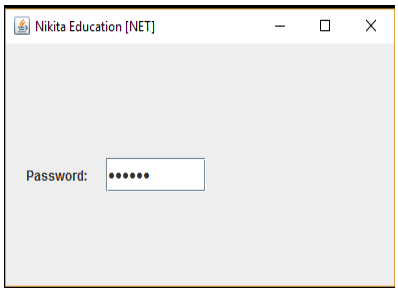


F. JPasswordField

Commonly used Constructors:

Constructor	Description
JPasswordField()	Constructs a new JPasswordField, with a default document, null starting text string, and 0 column width.
JPasswordField(int columns)	Constructs a new empty JPasswordField with the specified number of columns.
JPasswordField(String text)	Constructs a new JPasswordField initialized with the specified text.
JPasswordField(String text, int columns)	Construct a new JPasswordField initialized with the specified text and columns.

```
import javax.swing.*;
public class PasswordFieldExample{
        public static void main(String[] args) {
            JFrame f=new JFrame("Nikita Education [NET]");
            JPasswordField value = new JPasswordField();
            JLabel l1=new JLabel("Password:");
            l1.setBounds(20,100, 80,30);
            value.setBounds(100,100,100,30);
            f.add(value);  f.add(l1);
            f.setSize(300,300);
            f.setLayout(null);
            f.setVisible(true);
        }
}
```



G. Swing Buttons

Swing defines four types of buttons: JButton, JToggleButton, JCheckBox, and JRadioButton. **All are subclasses of the AbstractButton class**, which extends JComponent. Thus, all buttons share a set of common traits. AbstractButton contains many methods that allow you to control the behavior of buttons.

AbstractButton

Important Methods useful for all swing buttons i.e. JButton, JToggleButton, JCheckBox, JRadioButton

Modifier and Type	Method and Description
void	**addActionListener**(**ActionListener** l) Adds an ActionListener to the button.
void	**addItemListener**(**ItemListener** l) Adds an ItemListener to the checkbox.
String	**getActionCommand**() Returns the action command for this button.
Icon	**getIcon**() : Returns the default icon.
Icon	**getSelectedIcon**() Returns the selected icon for the button.
String	**getText**() : Returns the button's text.
void	**setActionCommand**(**String** actionCommand) Sets the action command for this button.
void	**setDisabledIcon**(**Icon** disabledIcon) Sets the disabled icon for the button.
void	**setEnabled**(boolean b) Enables (or disables) the button.
void	**setHorizontalAlignment**(int alignment) Sets the horizontal alignment of the icon and text.
void	**setIcon**(**Icon** defaultIcon) Sets the button's default icon.
void	**setPressedIcon**(**Icon** pressedIcon) Sets the pressed icon for the button.
void	**setSelected**(boolean b) Sets the state of the button.
void	**setSelectedIcon**(**Icon** selectedIcon) Sets the selected icon for the button.
void	**setText**(**String** text) Sets the button's text.

H. JButton

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:

Methods	Description
void setText(String s)	It is used to set specified text on button
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 addActionListener(ActionListener a)	It is used to add the action listener to this object.
void **setActionCommand**(**String** actionCommand)	Sets the action command for this button
String **getActionCommand**()	gets the action command for this button

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
public class JButtonDemo extends JFrame implements ActionListener{
	JLabel jlab;
	public JButtonDemo() {
		setTitle("Nikita Education [NET]");
		setSize(800,600);
		setVisible(true);
		setLayout(new FlowLayout(FlowLayout.LEFT));
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

		ImageIcon a = new ImageIcon("a.gif");
		JButton jb = new JButton(a);
		jb.setActionCommand("a");
		jb.addActionListener(this);
		add(jb);

		ImageIcon b = new ImageIcon("b.gif");
		jb = new JButton(b);
		jb.setActionCommand("b");
		jb.addActionListener(this);
		add(jb);

		ImageIcon c = new ImageIcon("c.gif");
		jb = new JButton(c);
		jb.setActionCommand("c");
		jb.addActionListener(this);
		add(jb);

		ImageIcon d = new ImageIcon("d.gif");
		jb = new JButton(d);
		jb.setActionCommand("d");
		jb.addActionListener(this);
		add(jb);

		jlab = new JLabel("Choose a Flag");
		add(jlab);
	}
	public void actionPerformed(ActionEvent ae){
		jlab.setText("You selected " + ae.getActionCommand());
	}
	public static void main(String[] args) {
		JButtonDemo newFrame=new JButtonDemo();
	}
}
```

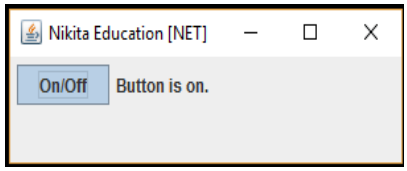
I. JToggleButton

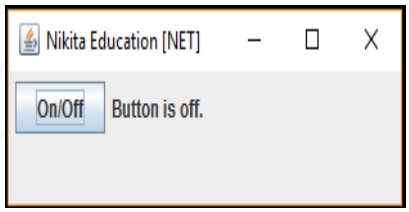
A useful variation on the push button is called a ***toggle button***. A toggle button looks just like a push button, but it acts differently because it has two states: **pushed and released.** Each time a toggle button is pushed, **it toggles between its two states.** Toggle buttons are objects of the **ToggleButton** class. JToggleButton is a **superclass** for **JCheckBox and JRadioButton.** JToggleButton defines the basic functionality of all **two-state components.**

Important Constructors

JToggleButton() Creates an initially unselected toggle button without setting the text or image.
JToggleButton(**Icon** icon) Creates an initially unselected toggle button with the specified image but no text.
JToggleButton(**Icon** icon, boolean selected) Creates a toggle button with the specified image and selection state, but no text.
JToggleButton(**String** text) Creates an unselected toggle button with the specified text.
JToggleButton(**String** text, boolean selected) Creates a toggle button with the specified text and selection state.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
public class JToggleButtonDemo extends JFrame{
        JLabel jlab;
        JToggleButton jtbn;
        public JToggleButtonDemo() {
                setTitle("Nikita Education [NET]");
                setSize(800,600);
                setVisible(true);
                setLayout(new FlowLayout(FlowLayout.LEFT));
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                jlab = new JLabel("Button is off.");
                jtbn = new JToggleButton("On/Off");
                jtbn.addItemListener(new ItemListener() {
                        public void itemStateChanged(ItemEvent ie) {
                                if(jtbn.isSelected())
                                        jlab.setText("Button is on.");
                                else
                                        jlab.setText("Button is off.");
                        }
                });
                add(jtbn);
                add(jlab);
        }
        public static void main(String[] args)
        {
        JToggleButtonDemo newFrame=new JToggleButtonDemo();
        }
}
```



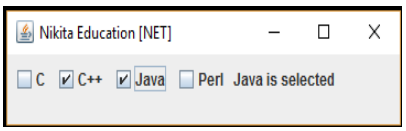


J. JCheckBox

Commonly used Constructors:

Constructor	Description
JCheckBox()	Creates an initially unselected check box button with no text, no icon.
JChechBox(String s)	Creates an initially unselected check box with text.
JCheckBox(String text, boolean selected)	Creates a check box with text and specifies whether or not it is initially selected.
JCheckBox(String text, Icon icon, boolean selected)	Creates a check box with text and icon, and specifies whether or not it is initially selected.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
public class JCheckBoxDemo extends JFrame implements ItemListener
{
	JLabel jlab;
	public JCheckBoxDemo(){
		setTitle("Nikita Education [NET]");
		setSize(800,600);
		setVisible(true);
		setLayout(new FlowLayout(FlowLayout.LEFT));
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

		JCheckBox cb = new JCheckBox("C");
		cb.addItemListener(this);
		add(cb);
		cb = new JCheckBox("C++");
		cb.addItemListener(this);
		add(cb);
		cb = new JCheckBox("Java");
		cb.addItemListener(this);
		add(cb);
		cb = new JCheckBox("Perl");
		cb.addItemListener(this);
		add(cb);

		jlab = new JLabel("Select languages");
		add(jlab);
	}
	public void itemStateChanged(ItemEvent ie) {
		JCheckBox cb = (JCheckBox)ie.getItem();
		if(cb.isSelected())
			jlab.setText(cb.getText() + " is selected");
		else
			jlab.setText(cb.getText() + " is cleared");
	}
	public static void main(String[] args) {
		JCheckBoxDemo newFrame=new JCheckBoxDemo();
	}
}
```



K. JRadioButton

Radio buttons are a group of **mutually exclusive buttons**, in which only one button can be selected at any one time. They are supported by the **JRadioButton** class, which extends JToggleButton. The JRadioButton class is used to create a radio button. Radio buttons **must be configured into a group**. Only one of the buttons in the group can be selected at any time. A button group is created by the **ButtonGroup** class.

ButtonGroup
Creates group of mutually exclusive buttons.
Constructors: public **ButtonGroup()**
Method: void **add(AbstractButton *ab*)**

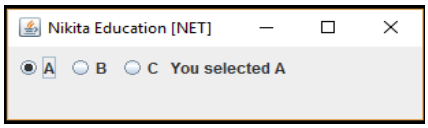
Commonly used Constructors:

Constructor	Description
JRadioButton()	Creates an unselected radio button with no text.
JRadioButton(String s)	Creates an unselected radio button with specified text.
JRadioButton(String s, boolean selected)	Creates a radio button with the specified text and selected status.
JRadioButton(String text, Icon icon, boolean selected)	Creates a radio button that has the specified text, image, and selection state.

Commonly used Methods:

Methods	Description
void setText(String s)	It is used to set specified text on button.
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 addActionListener(ActionListener a)	It is used to add the action listener to this object.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
public class JRadioButtonDemo extends JFrame implements ActionListener{
        JLabel jlab;
        public JRadioButtonDemo(){
                setTitle("Nikita Education [NET]");
                setSize(800,600);
                setVisible(true);
                setLayout(new FlowLayout(FlowLayout.LEFT));
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

                JRadioButton b1 = new JRadioButton("A");
                b1.addActionListener(this);
                add(b1);
```



```
                JRadioButton b2 = new JRadioButton("B");
                b2.addActionListener(this);
                add(b2);

                JRadioButton b3 = new JRadioButton("C");
                b3.addActionListener(this);
                add(b3);

                ButtonGroup bg = new ButtonGroup();
                bg.add(b1);
                bg.add(b2);
                bg.add(b3);
                jlab = new JLabel("Select One");
                add(jlab);
        }
        public void actionPerformed(ActionEvent ae){
        jlab.setText("You selected " + ae.getActionCommand());
        }
        public static void main(String[] args){
        JRadioButtonDemo newFrame=new JRadioButtonDemo();
        }
}
```

L. JComboBox

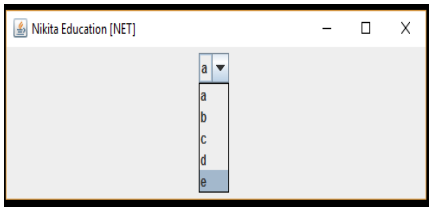
Commonly used Constructors:

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.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
public class JComboBoxDemo extends JFrame{
        JLabel jlab;
        ImageIcon a,b,c,d,e;
        JComboBox jcb;
        String flags[] = { "a", "b", "c", "d", "e" };
```



```
        public JComboBoxDemo(){
                setTitle("Nikita Education [NET]");
                setSize(500,500);
                setVisible(true);
                setLayout(new FlowLayout());
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                jcb = new JComboBox(flags);
                add(jcb);
                jcb.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent ae) {
                        String s = (String) jcb.getSelectedItem();
                        jlab.setIcon(new ImageIcon(s + ".gif"));
                }});
                jlab = new JLabel(new ImageIcon("a.gif"));
                add(jlab);
        }
        public static void main(String[] args){
                JComboBoxDemo newFrame=new JComboBoxDemo();
        }
}
```

M. JList

1. In Swing, the basic list class is called **JList**. It supports the **selection of one or more items** from a list. Although the list often consists of strings, it is possible to create a list of just about any object that can be displayed. **JList** is so widely used in Java
2. **JList is based on two models.**
3. The first is **ListModel**. This interface defines how access to the list data is achieved.
4. The second model is the **ListSelectionModel** interface, which defines methods that determine what list item or items are selected.
5. Although a JList will work properly by itself, most of the time you will wrap a **JList inside a JScrollPane**. This way, long lists will **automatically be scrollable**, which simplifies GUI design

Commonly used Constructors:

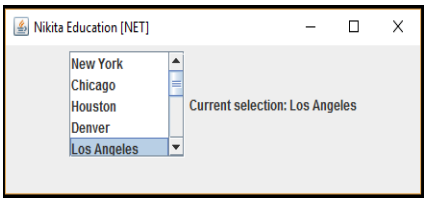
Constructor	Description
JList()	Creates a JList with an empty, read-only, model.
JList(ary[] listData)	Creates a JList that displays the elements in the specified array.
JList(ListModel<ary> dataModel)	Creates a JList that displays elements from the specified, non-null, model.

Commonly used Methods:

Methods	Description
Void addListSelectionListener(ListSelectionListener listener)	It is used to add a listener to the list, to be notified each time a change to the selection occurs.
int getSelectedIndex()	It is used to return the smallest selected cell index.
ListModel getModel()	It is used to return the data model that holds a list of items displayed by the JList component.
void setListData(Object[] listData)	It is used to create a read-only ListModel from an array of objects.

void **setSelectionMode(int *mode*)**	mode constants: 1. SINGLE_SELECTION 2. SINGLE_INTERVAL_SELECTION 3. MULTIPLE_INTERVAL_SELECTION

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
public class JListDemo extends JFrame{
        JList jlst;
        JLabel jlab;
        JScrollPane jscrlp;
        String Cities[] = { "New York", "Chicago", "Houston",
                            "Denver", "Los Angeles", "Seattle",
                            "London", "Paris", "New Delhi",
                            "Hong Kong", "Tokyo", "Sydney" };
        public JListDemo(){
                setTitle("Nikita Education [NET]");
                setSize(500,500);
                setVisible(true);
                setLayout(new FlowLayout());
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                jlst = new JList(Cities);
        jlst.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
                jscrlp = new JScrollPane(jlst);
                jscrlp.setPreferredSize(new Dimension(120, 90));
                jlab = new JLabel("Choose a City");
                jlst.addListSelectionListener(new  ListSelectionListener()
                {
                public void valueChanged(ListSelectionEvent le) {
                        int idx = jlst.getSelectedIndex();
                        if(idx != -1)
                        jlab.setText("Current selection: " + Cities[idx]);
                        else
                        jlab.setText("Choose a City");
                }});
                add(jscrlp);
                add(jlab);
        }
        public static void main(String[] args) {
                JListDemo newFrame=new JListDemo();
        }
}
```

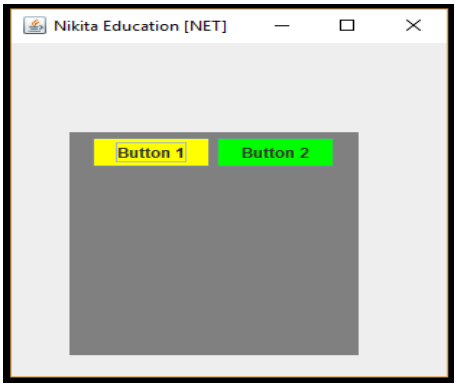


N. JPanel

Commonly used Constructors:

Constructor	Description
JPanel()	It is used to create a new JPanel with a double buffer and a flow layout.
JPanel(boolean isDoubleBuffered)	It is used to create a new JPanel with FlowLayout and the specified buffering strategy.
JPanel(LayoutManager layout)	It is used to create a new JPanel with the specified layout manager.

```
import java.awt.*;
import javax.swing.*;
public class PanelExample{
        PanelExample(){
                JFrame f= new JFrame("Nikita Education [NET]");
                JPanel panel=new JPanel();
                panel.setBounds(40,80,200,200);
                panel.setBackground(Color.gray);
                JButton b1=new JButton("Button 1");
                b1.setBounds(50,100,80,30);
                b1.setBackground(Color.yellow);
                JButton b2=new JButton("Button 2");
                b2.setBounds(100,100,80,30);
                b2.setBackground(Color.green);
                panel.add(b1); panel.add(b2);
                f.add(panel);
                f.setSize(400,400);
                f.setLayout(null);
                f.setVisible(true);
        }
        public static void main(String args[]){
                new PanelExample();
        }
}
```



O. JTabbedPane

1. JTabbedPane encapsulates **a *tabbed pane***.
2. It manages **a set of components** by linking them with tabs.
3. Selecting a tab causes the component associated with that tab to come to the forefront.
4. Tabbed panes are very common in the modern GUI
5. JTabbedPane uses the **SingleSelectionModel**

Steps:

1. Create an instance of JTabbedPane.
2. Add each tab by calling addTab( ).
3. Add the tabbed pane to the content pane.

Commonly used Constructors:

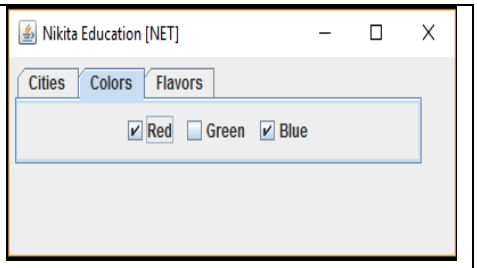
Constructor	Description
JTabbedPane()	Creates an empty TabbedPane with a default tab placement of JTabbedPane.Top.
JTabbedPane(int tabPlacement)	Creates an empty TabbedPane with a specified tab placement.
JTabbedPane(int tabPlacement, int tabLayoutPolicy)	Creates an empty TabbedPane with a specified tab placement and tab layout policy.

Methods:

void **addTab(String *name*, Component *comp*) --** add instance of new tab to the tabbed pane.

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

```
import javax.swing.event.*;
class CitiesPanel extends JPanel {
        public CitiesPanel() {
                JButton b1 = new JButton("New York");
                add(b1);
                JButton b2 = new JButton("London");
                add(b2);
                JButton b3 = new JButton("Hong Kong");
                add(b3);
                JButton b4 = new JButton("Tokyo");
                add(b4);
        }
}
class ColorsPanel extends JPanel {
        public ColorsPanel() {
                JCheckBox cb1 = new JCheckBox("Red");
                add(cb1);
                JCheckBox cb2 = new JCheckBox("Green");
                add(cb2);
                JCheckBox cb3 = new JCheckBox("Blue");
                add(cb3);
        }
}
class FlavorsPanel extends JPanel{
        public FlavorsPanel()  {
                JComboBox jcb = new JComboBox();
                jcb.addItem("Vanilla");
                jcb.addItem("Chocolate");
                jcb.addItem("Strawberry");
                add(jcb);
        }
}
public class JTabbedPaneDemo extends JFrame {
        public JTabbedPaneDemo()   {
                setTitle("Nikita Education [NET]");
                setSize(800,600);
                setVisible(true);
                setLayout(new FlowLayout(FlowLayout.LEFT));
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

                JTabbedPane jtp = new JTabbedPane();
                jtp.addTab("Cities", new CitiesPanel());
                jtp.addTab("Colors", new ColorsPanel());
                jtp.addTab("Flavors", new FlavorsPanel());
                add(jtp);
        }
        public static void main(String[] args) {
        JTabbedPaneDemo newFrame=new JTabbedPaneDemo();
        }
}
```



P. JScrollPane

1. JScrollPane is **a lightweight container** that automatically handles the **scrolling** of another component.
2. The component being scrolled can either be an individual component, such as a **table**, or a group of components contained within another lightweight container, such as a **Jpanel**

3. **JScrollPane automates scrolling**, it usually eliminates the need to manage individual scroll bars
4. The **viewable** area of a scroll pane is called the ***viewport***.
5. It is a window in which the component being scrolled is displayed.
6. Thus, the viewport displays the **visible portion of the component** being scrolled.
7. The scroll bars scroll the component through the viewport.
8. In its default behavior, a **JScrollPane will dynamically add or remove** a scroll bar as needed

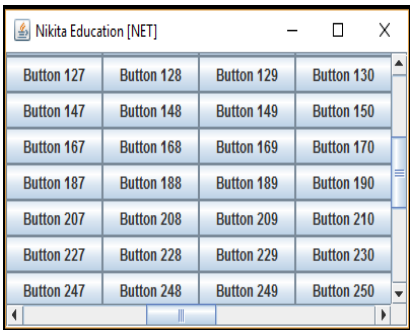
Steps:

1. Create the component to be scrolled.
2. Create an instance of JScrollPane, passing to it the object to scroll.
3. Add the scroll pane to the content pane

Constructors:

JScrollPane(Component *comp*)

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
public class JScrollPaneDemo extends JFrame
{
	public JScrollPaneDemo()
	{
		setTitle("Nikita Education [NET]");
		setSize(500,500);
		setVisible(true);
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		JPanel jp = new JPanel();
		jp.setLayout(new GridLayout(20, 20));
		int b = 0;
		for(int i = 0; i < 20; i++)
		{
			for(int j = 0; j < 20; j++)
			{
				jp.add(new JButton("Button " + b));
				++b;
			}
		}
		JScrollPane jsp = new JScrollPane(jp);
		add(jsp);
	}
	public static void main(String[] args)
	{
		JScrollPaneDemo newFrame=new JScrollPaneDemo();
	}
}
```



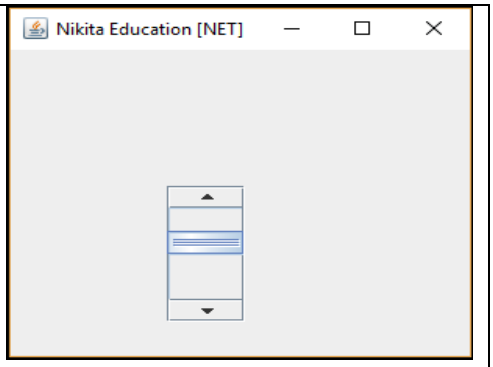
Q. JScrollBar

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.

```
import javax.swing.*;
class ScrollBarExample
{
        ScrollBarExample(){
                JFrame f= new JFrame("Nikita Education [NET]");
                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();
        }
}
```



R. 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.

1. A *tree* is a component that presents **a hierarchical view of data**.
2. The user has the ability **to expand or collapse individual subtrees** in this display.
3. Trees are implemented in Swing by the **JTree** class.
4. JTree relies on **two** models: **TreeModel and TreeSelectionModel.**
5. A JTree generates a variety of events, but three relate specifically to trees: **TreeExpansionEvent, TreeSelectionEvent, and TreeModelEvent**

Steps:

1. Create an instance of **JTree**.
2. Create a **JScrollPane** and specify the tree as the object to be scrolled.
3. Add the tree to the scroll pane.
4. Add the scroll pane to the content pane

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.

DefaultMutableTreeNode

The **DefaultMutableTreeNode** class implements the MutableTreeNode interface. It represents **a node** in a tree.

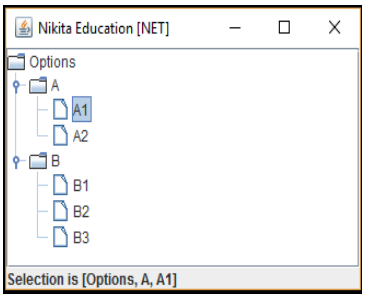
Constructors:

DefaultMutableTreeNode(Object *obj*)

Methods:

void **add(MutableTreeNode *child*)**

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.tree.*;
public class JTreeDemo extends JFrame{
        JTree tree;
        JLabel jlab;
        public JTreeDemo(){
                setTitle("Nikita Education [NET]");
                setSize(500,500);
                setVisible(true);
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

// Create top node of tree.
DefaultMutableTreeNode top = new DefaultMutableTreeNode("Options");
// Create subtree of "A".
DefaultMutableTreeNode a = new DefaultMutableTreeNode("A");
top.add(a);
DefaultMutableTreeNode a1 = new DefaultMutableTreeNode("A1");
a.add(a1);
DefaultMutableTreeNode a2 = new DefaultMutableTreeNode("A2");
a.add(a2);
// Create subtree of "B".
DefaultMutableTreeNode b = new DefaultMutableTreeNode("B");
top.add(b);
DefaultMutableTreeNode b1 = new DefaultMutableTreeNode("B1");
b.add(b1);
DefaultMutableTreeNode b2 = new DefaultMutableTreeNode("B2");
b.add(b2);
DefaultMutableTreeNode b3 = new DefaultMutableTreeNode("B3");
b.add(b3);
                // Create the tree.
                tree = new JTree(top);
                // Add the tree to a scroll pane.
                JScrollPane jsp = new JScrollPane(tree);
                // Add the scroll pane to the content pane.
                add(jsp);
                // Add the label to the content pane.
                jlab = new JLabel();
                add(jlab, BorderLayout.SOUTH);
                // Handle tree selection events.
                tree.addTreeSelectionListener(new TreeSelectionListener() {
                        public void valueChanged(TreeSelectionEvent tse) {
                                jlab.setText("Selection is " + tse.getPath());
                        }
                });
        }
        public static void main(String[] args){
                JTreeDemo newFrame=new JTreeDemo();
        }
}
```



S. JTable

JTable is a component that displays **rows and columns** of data. You can drag the cursor on column boundaries to resize columns. You can also drag a column to a new position. Depending on its configuration, it is also possible **to select a row, column, or cell** within the table, and to change the data within a cell. **JTable** is a sophisticated component that offers many more options and features. **JTable** relies **on three models**. The first is **the table model**, which is defined by the **TableModel** interface. This model defines those things related **to displaying data in a two-dimensional** format. The second is **the table column model**, which is represented by **TableColumnModel**. JTable is defined in terms of columns, and it is **TableColumnModel** that specifies the characteristics of a column The third model determines how **items are selected**, and it is specified by the **ListSelectionModel.**

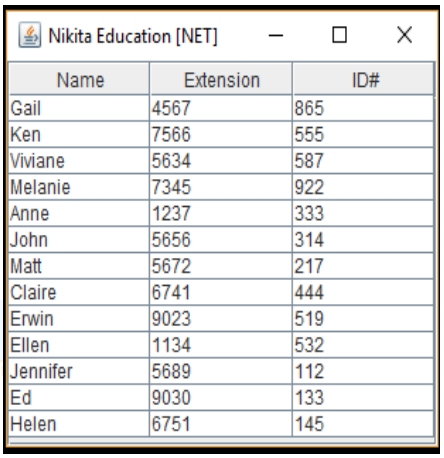
Steps:

1. Create an instance of JTable.
2. Create a JScrollPane object, specifying the table as the object to scroll.
3. Add the table to the scroll pane. Add the scroll pane to the content pane

Commonly used Constructors:

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

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
public class JTableDemo extends JFrame
{
        JTree tree;
        JLabel jlab;
        public JTableDemo()  {
                setTitle("Nikita Education [NET]");
                setSize(500,500);
                setVisible(true);
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                // Initialize column headings.
                String[] colHeads = { "Name", "Extension", "ID#" };
                // Initialize data.
                Object[][] data = {
                        { "Gail", "4567", "865" },
                        { "Ken", "7566", "555" },
                        { "Viviane", "5634", "587" },
                        { "Melanie", "7345", "922" },
                        { "Anne", "1237", "333" },
                        { "John", "5656", "314" },
                        { "Matt", "5672", "217" },
                        { "Claire", "6741", "444" },
                        { "Erwin", "9023", "519" },
                        { "Ellen", "1134", "532" },
                        { "Jennifer", "5689", "112" },
                        { "Ed", "9030", "133" },
                        { "Helen", "6751", "145" }
                };
```



```
                // Create the table.
                JTable table = new JTable(data, colHeads);
                // Add the table to a scroll pane.
                JScrollPane jsp = new JScrollPane(table);
                // Add the scroll pane to the content pane.
                add(jsp);
        }
        public static void main(String[] args) {
                JTableDemo newFrame=new JTableDemo();
        }
}
```

T. JProgressBar

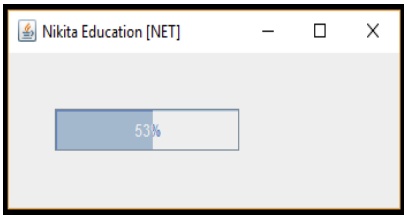
Commonly used Constructors:

Constructor	Description
JProgressBar()	It is used to create a horizontal progress bar but no string text.
JProgressBar(int min, int max)	It is used to create a horizontal progress bar with the specified minimum and maximum value.
JProgressBar(int orient)	It is used to create a progress bar with the specified orientation, it can be either Vertical or Horizontal by using SwingConstants.VERTICAL and SwingConstants.HORIZONTAL constants.
JProgressBar(int orient, int min, int max)	It is used to create a progress bar with the specified orientation, minimum and maximum value.

Commonly used Methods:

Method	Description
void setStringPainted(boolean b)	It is used to determine whether string should be displayed.
void setString(String s)	It is used to set value to the progress string.
void setOrientation(int orientation)	It is used to set the orientation, it may be either vertical or horizontal by using SwingConstants.VERTICAL and SwingConstants.HORIZONTAL constants.
void setValue(int value)	It is used to set the current value on the progress bar.

```
import javax.swing.*;
public class ProgressBarExample extends JFrame
{
        JProgressBar jb;
        int i=0,num=0;
        ProgressBarExample(){
                jb=new JProgressBar(0,2000);
                jb.setBounds(40,40,160,30);
                jb.setValue(0);
                jb.setStringPainted(true);
                add(jb);
                setSize(250,150);
                setTitle("Nikita Education [NET]");
                setLayout(null);
        }
```



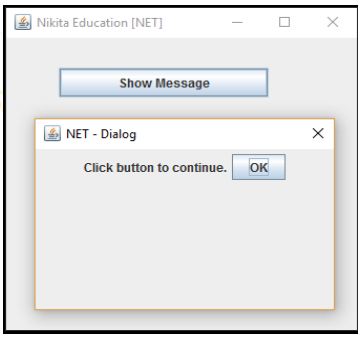
```
        public void iterate(){
                while(i<=2000)
                {
                        jb.setValue(i);
                        i=i+20;
                        try{
                                Thread.sleep(150);
                        }
                        catch(Exception e)
                        {
                        }
                }
        }
        public static void main(String[] args) {
                ProgressBarExample m=new ProgressBarExample();
                m.setVisible(true);
                m.iterate();
        }
}
```

U. JDialog

Commonly used Constructors:

Constructor	Description
JDialog()	It is used to create a modeless dialog without a title and without a specified Frame owner.
JDialog(Frame owner)	It is used to create a modeless dialog with specified Frame as its owner and an empty title.
JDialog(Frame owner, String title, boolean modal)	It is used to create a dialog with the specified title, owner Frame and modality.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class DialogExample{
        private static JDialog d;
        DialogExample(){
                JFrame f= new JFrame();
                JButton a=new JButton("Show Message");
                a.addActionListener ( new ActionListener(){
                   public void actionPerformed( ActionEvent e ){
                        d = new JDialog(f , "NET - Dialog", true);
                        d.setLayout( new FlowLayout() );
                        JButton b = new JButton ("OK");
                        b.addActionListener ( new ActionListener(){
                           public void actionPerformed( ActionEvent e ){
                                DialogExample.d.setVisible(false);
                        }});
                        d.add( new JLabel ("Click button to continue."));
                        d.add(b);
                        d.setSize(300,200);
                        d.setVisible(true);
                }});
                a.setBounds(50,30,200,30);
```



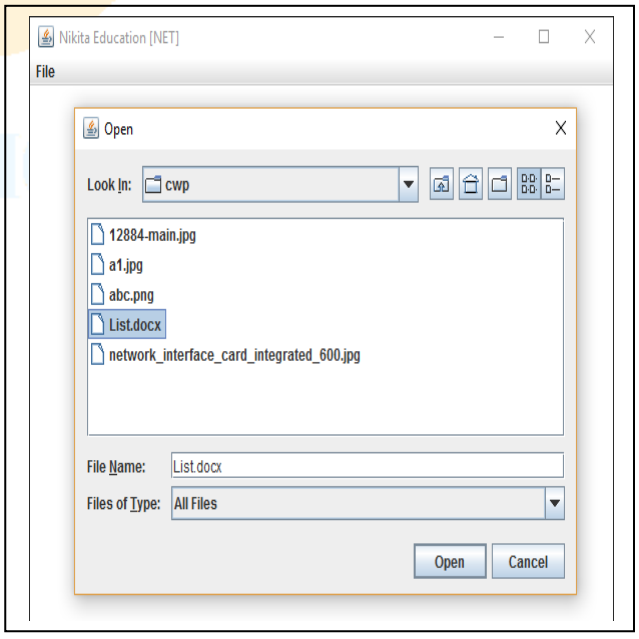
```
                f.add(a);
                f.setSize(500,500);
                f.setTitle("Nikita Education [NET]");
                f.setLayout(null);
                f.setVisible(true);
        }
        public static void main(String args[]) {
                new DialogExample();
        }
}
```

V. JFileChooser

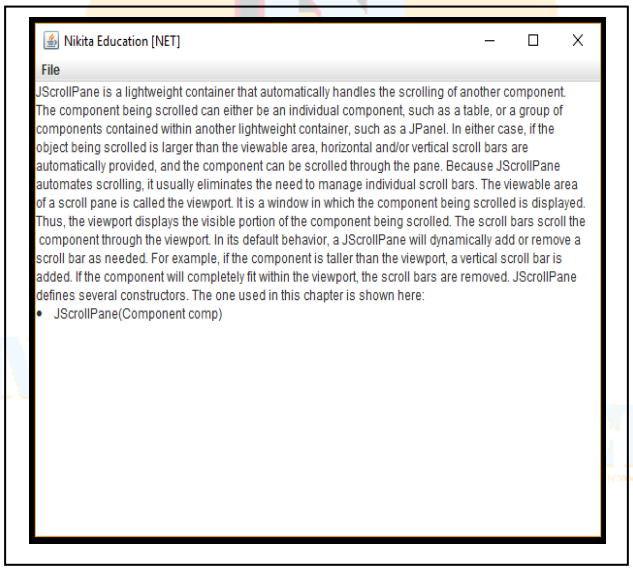
Commonly used Constructors:

Constructor	Description
JFileChooser()	Constructs a JFileChooser pointing to the user's default directory.
JFileChooser(File currentDirectory)	Constructs a JFileChooser using the given File as the path.
JFileChooser(String currentDirectoryPath)	Constructs a JFileChooser using the given path.

```
import javax.swing.*;
import java.awt.event.*;
import java.io.*;
public class FileChooserExample extends JFrame implements ActionListener{
        JMenuBar mb;
        JMenu file;
        JMenuItem open;
        JTextArea ta;
        FileChooserExample() {
                open=new JMenuItem("Open File");
                open.addActionListener(this);
                file=new JMenu("File");
                file.add(open);
                mb=new JMenuBar();
                mb.setBounds(0,0,800,20);
                mb.add(file);
                ta=new JTextArea(800,800);
                ta.setBounds(0,20,800,800);
                add(mb);
                add(ta);
        }
        public void actionPerformed(ActionEvent e)  {
                if(e.getSource()==open){
                    JFileChooser fc=new JFileChooser();
                    int i=fc.showOpenDialog(this);
                    if(i==JFileChooser.APPROVE_OPTION){
                                File f=fc.getSelectedFile();
                                String filepath=f.getPath();
                                try{
```



```
                        BufferedReader br=new BufferedReader(new FileReader(filepath));
                        String s1="",s2="";
                    while((s1=br.readLine())!=null){
                        s2+=s1+"\n";
                    }
                    ta.setText(s2);
                    br.close();
                }catch (Exception ex) {
                        ex.printStackTrace();
                }
            }
        }
    }
    public static void main(String[] args){
        FileChooserExample om=new FileChooserExample();
        om.setSize(500,500);
        om.setTitle("Nikita Education [NET]");
        om.setLayout(null);
        om.setVisible(true);
        om.setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
}
```



Unit-II
Event Handling

Unit	Topic Name	Topics	Marks
2	**Event Handling**	Delegation Event Model, Event Classes: WindowEvent, ActionEvent, MouseEvent, KeyEvent, ItemEvent, FocusEvent, TextEvent etc., Event Listeners: WindowListener, ActionListener, MouseListener, KeyListener, ItemListener, FocusListener, TextListener etc., Adapter Classes.	**10**

A. Introduction

Applets are event-driven programs that use a graphical user interface to interact with the user. Furthermore, any program that uses a graphical user interface, such as a Java application written for Windows, is event driven. Thus, you cannot write these types of programs without a solid command of event handling. Events are supported by a number of packages, including **java.util**, **java.awt**, and **java.awt.event**. Most events to which your program will respond are generated when the user interacts with a GUI-based program. There are several types of events, including those generated by the mouse, the keyboard, and various GUI controls, such as a push button, scroll bar, or check box etc.

B. The Delegation Event Model

The delegation event model is based on the Event Sources and Event Listeners. Event Listener is handler that receives the messages / events. The Event Source is any object which creates the message / event. The Event Delegation model is based on – The Event Classes, The Event Listeners, and Event Objects.

Events: Event is an object that describes a state of change in a source. Event may be generated as a consequence of a person interacting with the GUI elements such as pressing a button, entering a character through keyboard, selecting an item from a list.

Event Sources: "Source" is an object that generates an event. listeners must register with the source in order to receive event notification.

Event Listeners: A listener is an object that is notified when an event occurs. To do so 2 things should be satisfied. 1) It must have been registered with one or more sources to receive notification. 2) It must implement methods to receive and process these notifications.

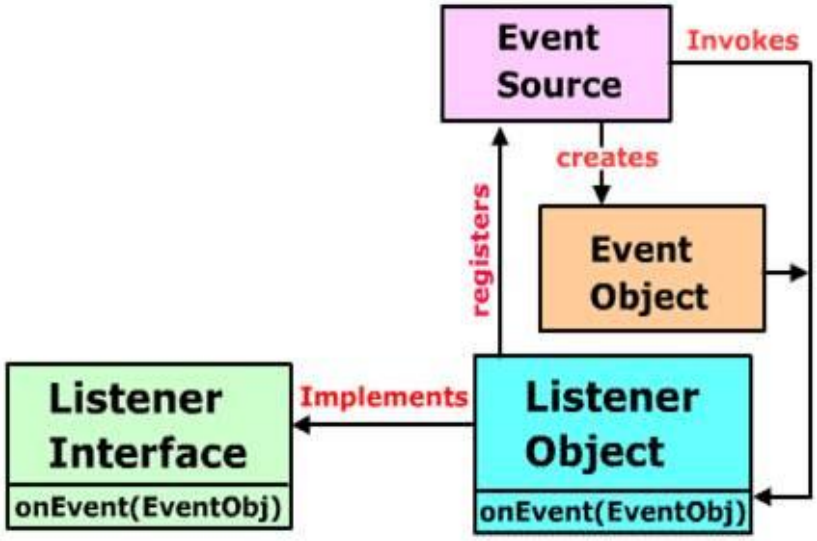
B.1. Process of Event Handling:

- The modern approach to handling events is based on the delegation event model.
- In this model source generates an event and sends it to one or more listeners.
- The listener simply waits until it receives an event.
- Once received, the listener processes the event and then returns.
- The advantage of this design is that the application logic that processes events is clearly separated from the user interface logic that generates those events.
- User interface logic is able to delegate the processing of an event to a separate piece of code.
- In the delegation model, listeners must register with the source in order to receive event notification. An each type of event has its own registration method and following is the general form of it.

public void addTypeListener(TypeListener el)

For Example:

addActionListener(ActionListener obj);
addKeyListener(KeyListener obj);
addMouseListener(MouseListener obj);
addItemListener(ItemListener obj);



The Delegation Event Model

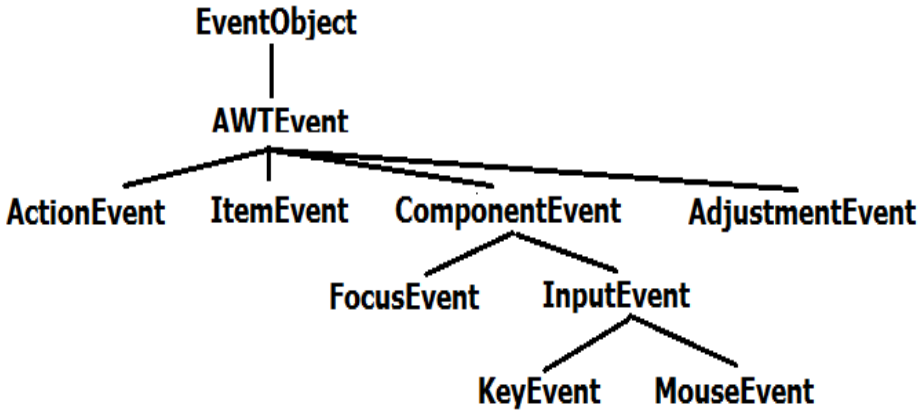
Steps to Handle Events

- Declare and instantiate event sources (or components) such as buttons, menus, choices etc.
- Implement an interface (listener) to provide the event handler that responds to event source activity.
- Register this event handler with the event source.
- Add the event source to the container like applet, frame, panel etc.
- Implement the methods of listener interface and provide actions for respective type of events.

C. Event Classes

Event Class	Description
EventObject	It is the super class for all events which is in **java.util** package. Important method of EventObject class: **Object getSource( )**
AWTEvent	It is a super class of all AWT events that are handled by the delegation event model.
ActionEvent	An ActionEvent is generated when a button is pressed, a list item is double-clicked, or a menu item is selected. Important method: **String getActionCommand( )**
FocusEvent	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.**
ItemEvent	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** and **SELECTED**
KeyEvent	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.
MouseEvent	There are eight types of mouse events. MouseEvent is a subclass of InputEvent. The MouseEvent class defines the different integer constants that can be used to identify mouse event.

TextEvent	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.
WindowEvent	There are ten types of window events. **WindowEvent** is a subclass of **ComponentEvent.** The **WindowEvent** class defines integer constants that can be used to identify them.



Hierarchy of event classes

D. Listener Interfaces

Interface	Description	Methods
ActionListener	Defines one method to receive action events.	void actionPerformed(ActionEvent *ae*)
AdjustmentListener	Defines one method to receive adjustment events.	void adjustmentValueChanged (AdjustmentEvent *ae*)
FocusListener	Defines two methods to recognize when a component gains or loses keyboard focus	void focusGained(FocusEvent *fe*) void focusLost(FocusEvent *fe*)
ItemListener	Defines one method to recognize when the state of an item changes.	void itemStateChanged(ItemEvent *ie*)
KeyListener	Defines three methods to recognize when a key is pressed, released, or typed.	void keyPressed(KeyEvent *ke*) void keyReleased(KeyEvent *ke*) void keyTyped(KeyEvent *ke*)
MouseListener	Defines five methods to recognize when the mouse is clicked, enters a component, exits a component, is pressed, or is released.	void mouseClicked(MouseEvent *me*) void mouseEntered(MouseEvent *me*) void mouseExited(MouseEvent *me*) void mousePressed(MouseEvent *me*) void mouseReleased(MouseEvent *me*)
MouseMotionListener	Defines two methods to recognize when the mouse is dragged or moved.	void mouseDragged(MouseEvent *me*) void mouseMoved(MouseEvent *me*)
TextListener	Defines one method to recognize when a text value changes.	void textChanged(TextEvent *te*)
WindowListener	Defines seven methods to recognize when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.	void windowActivated(WindowEvent *we*) void windowClosed(WindowEvent *we*) void windowClosing(WindowEvent *we*) void windowDeactivated(WindowEvent *we*) void windowDeiconified(WindowEvent *we*) void windowIconified(WindowEvent *we*) void windowOpened(WindowEvent *we*)

E. Event Handling Example

```
//AEvent2.java

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

class AEvent2 extends Frame
{
	TextField tf;
	Button b;

	AEvent2()
	{
		//create components
		tf=new TextField();
		tf.setBounds(60,50,170,20);

		b=new Button("click me");
		b.setBounds(100,120,80,30);

		//register listener by passing outer class instance
		Outer o=new Outer(this);
		b.addActionListener(o);

		//add components and set size, layout and visibility
		add(b);
		add(tf);
		setSize(300,300);
		setLayout(null);
		setVisible(true);
	}
	public static void main(String args[])
	{
		new AEvent2();
	}
}

//Outer.java

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");
	}
}
```

Registration Methods

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

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

F. Event Classes

EventObject Class

The classes that represent events are at the core of Java's event handling mechanism. Thus, a discussion of event handling must begin with the event classes. It is important to understand, however, that Java defines several types of event classes. The most widely used events are those defined by the AWT and those defined by Swing. This chapter focuses on the AWT events. (Most of these events also apply to Swing.) At the root of the Java event class hierarchy is EventObject, which is in java.util. It is the super class for all events. It's one constructor is shown here:

EventObject(Object *src*)

Here, *src* is the object that generates this event. EventObject contains two methods: getSource() and toString(). The getSource( ) method returns the source of the event. Its general form is shown here:

Object getSource( )

- EventObject is a super class of all events.
- AWTEvent is a super class of all AWT events that are handled by the delegation event model.

The package java.awt.event defines many types of events that are generated by various user interface elements. Commonly used constructors and methods in each class are described in the following sections.

1. The ActionEvent Class

An ActionEvent is generated when a button is pressed, a list item is double-clicked, or a menu item is selected. You can obtain the command name for the invoking ActionEvent object by using the getActionCommand( ) method, shown here:

String getActionCommand( )

For example, when a button is pressed, an action event is generated that has a command name equal to the label on that button.

2. The AdjustmentEvent Class

An AdjustmentEvent 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.
UNIT_DECREMENT	The button at the end of the scroll bar was clicked to decrease its value.
UNIT_INCREMENT	The button at the end of the scroll bar was clicked to increase its value.

3. The 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 is a subclass of ComponentEvent.

4. The 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. The getItem( ) method can be used to obtain a reference to the item that generated an event. Its signature is shown here:

Object getItem( )

5. The 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. Remember, not all keypresses result in characters. For example, pressing SHIFT does not generate a character. 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_DOWN	VK_LEFT	VK_RIGHT
VK_CANCEL	VK_ENTER	VK_PAGE_DOWN	VK_SHIFT
VK_CONTROL	VK_ESCAPE	VK_PAGE_UP	VK_UP

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

6. The MouseEvent Class

There are eight types of mouse events. MouseEvent is a subclass of InputEvent. 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.

7. The 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.

8. The WindowEvent Class

There are ten types of window events. **WindowEvent** is a subclass of **ComponentEvent.** 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.

G. Listener Interfaces

As explained, the delegation event model has two parts: sources and listeners. Listeners are created by implementing one or more of the interfaces defined by the **java.awt.event** package. When an event occurs, the event source invokes the appropriate method defined by the listener and provides an event object as its argument. The following sections examine the specific methods that are contained in each interface.

1. The ActionListener Interface

This interface defines the actionPerformed( ) method that is invoked when an action event occurs. Its general form is shown here:

void actionPerformed(ActionEvent ae)

Interface	Description
ActionListener	Defines one method to receive action events.
AdjustmentListener	Defines one method to receive adjustment events.
ComponentListener	Defines four methods to recognize when a component is hidden, moved, resized, or shown.
ContainerListener	Defines two methods to recognize when a component is added to or removed from a container.
FocusListener	Defines two methods to recognize when a component gains or loses keyboard focus.
ItemListener	Defines one method to recognize when the state of an item changes.
KeyListener	Defines three methods to recognize when a key is pressed, released, or typed.
MouseListener	Defines five methods to recognize when the mouse is clicked, enters a component, exits a component, is pressed, or is released.
MouseMotionListener	Defines two methods to recognize when the mouse is dragged or moved.
MouseWheelListener	Defines one method to recognize when the mouse wheel is moved.
TextListener	Defines one method to recognize when a text value changes.
WindowFocusListener	Defines two methods to recognize when a window gains or loses input focus.
WindowListener	Defines seven methods to recognize when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

2. The AdjustmentListener Interface

This interface defines the adjustmentValueChanged( ) method that is invoked when an adjustment event occurs. Its general form is shown here:

void adjustmentValueChanged(AdjustmentEvent ae)

3. The ComponentListener Interface

This interface defines four methods that are invoked when a component is resized, moved, shown, or hidden. Their general forms are shown here:

void componentResized(ComponentEvent ce)
void componentMoved(ComponentEvent ce)
void componentShown(ComponentEvent ce)
void componentHidden(ComponentEvent ce)

4. The ContainerListener Interface

This interface contains two methods. When a component is added to a container, componentAdded( ) is invoked. When a component is removed from a container, componentRemoved( ) is invoked. Their general forms are shown here:

void componentAdded(ContainerEvent ce)
void componentRemoved(ContainerEvent ce)

5. The FocusListener Interface

This interface defines two methods. When a component obtains keyboard focus, focusGained() is invoked. When a component loses keyboard focus, focusLost( ) is called. Their general forms are shown here:

void focusGained(FocusEvent fe)
void focusLost(FocusEvent fe)

6. The ItemListener Interface

This interface defines the itemStateChanged( ) method that is invoked when the state of an item changes. Its general form is shown here:

void itemStateChanged(ItemEvent ie)

7. The KeyListener Interface

This interface defines three methods. The keyPressed( ) and keyReleased( ) methods are invoked when a key is pressed and released, respectively. The keyTyped( ) method is invoked when a character has been entered. For example, if a user presses and releases the A key, three events are generated in sequence: key pressed, typed, and released. If a user presses and releases the HOME key, two key events are generated in sequence: key pressed and released. The general forms of these methods are shown here:

void keyPressed(KeyEvent ke)
void keyReleased(KeyEvent ke)
void keyTyped(KeyEvent ke)

8. The MouseListener Interface

This interface defines five methods. If the mouse is pressed and released at the same point, mouseClicked( ) is invoked. When the mouse enters a component, the mouseEntered( ) method is called. When it leaves, mouseExited( ) is called. The mousePressed( ) and mouseReleased( ) methods are invoked when the mouse is pressed and released, respectively. The general forms of these methods are shown here:

void mouseClicked(MouseEvent me)
void mouseEntered(MouseEvent me)
void mouseExited(MouseEvent me)
void mousePressed(MouseEvent me)
void mouseReleased(MouseEvent me)

9. The MouseMotionListener Interface

This interface defines two methods. The mouseDragged( ) method is called multiple times as the mouse is dragged. The mouseMoved( ) method is called multiple times as the mouse is moved. Their general forms are shown here:

void mouseDragged(MouseEvent *me*)
void mouseMoved(MouseEvent *me*)

10. The MouseWheelListener Interface

This interface defines the mouseWheelMoved( ) method that is invoked when the mouse wheel is moved. Its general form is shown here:

void mouseWheelMoved(MouseWheelEvent mwe)

11. The TextListener Interface

This interface defines the textChanged( ) method that is invoked when a change occurs in a text area or text field. Its general form is shown here:

void textChanged(TextEvent te)

12.The WindowFocusListener Interface

This interface defines two methods: windowGainedFocus( ) and windowLostFocus( ). These are called when a window gains or loses input focus. Their general forms are shown here:

void windowGainedFocus(WindowEvent we)
void windowLostFocus(WindowEvent we)

13.The WindowListener Interface

This interface defines seven methods. The windowActivated( ) and windowDeactivated( ) methods are invoked when a window is activated or deactivated, respectively. If a window is iconified, the windowIconified( ) method is called. When a window is deiconified, the windowDeiconified( ) method is called. When a window is opened or closed, the windowOpened( ) or windowClosed( ) methods are called, respectively. The windowClosing() method is called when a window is being closed. The general forms of these methods are

void windowActivated(WindowEvent we)
void windowClosed(WindowEvent we)
void windowClosing(WindowEvent we)
void windowDeactivated(WindowEvent we)
void windowDeiconified(WindowEvent we)
void windowIconified(WindowEvent we)
void windowOpened(WindowEvent we)

H. Adapter Classes

Java provides a special feature, called an *adapter class that* can simplify the creation of event handlers in certain situations. An adapter class provides an empty implementation of all methods in an event listener interface. Adapter classes are useful when you want to receive and process only some of the events that are handled by a particular event listener interface. You can define a new class to act as an event listener by extending one of the adapter classes and implementing only those events in which you are interested.

Adapter classes of java.awt.event:

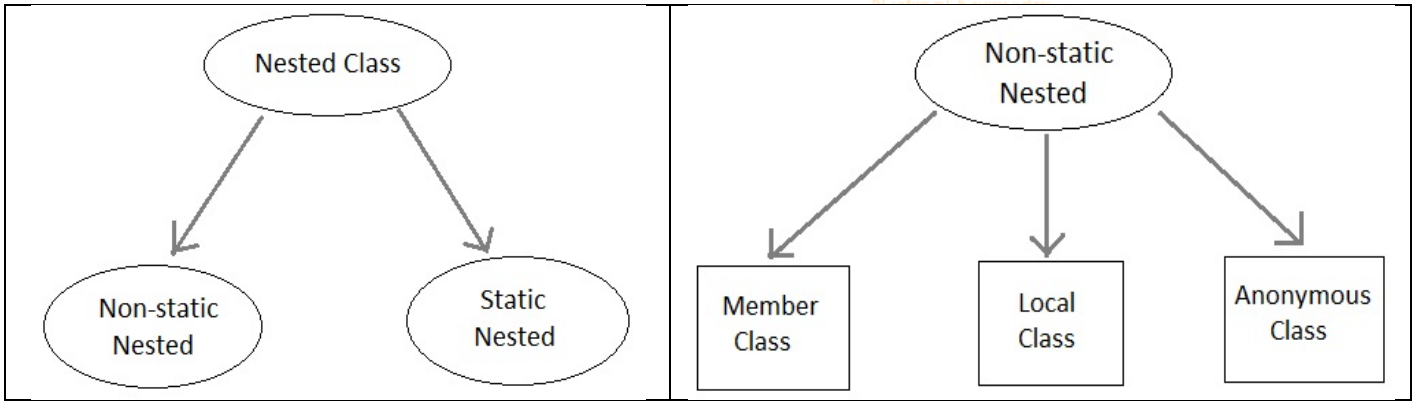
1. **ComponentAdapter**
2. **KeyAdapter**
3. **WindowAdapter**
4. **FocusAdapter**
5. **MouseAdapter**

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

/*<applet code="AdapterDemo" width=300 height=100></applet>*/

public class AdapterDemo extends Applet
{
        public void init()
        {
                addMouseListener(new MyMouseAdapter(this));
                addMouseMotionListener(new MyMouseMotionAdapter(this));
        }
}
```

```
class MyMouseAdapter extends MouseAdapter
{
        AdapterDemo adapterDemo;

        public MyMouseAdapter(AdapterDemo adapterDemo)
        {
                this.adapterDemo = adapterDemo;
        }

        // Handle only mouse clicked.
        public void mouseClicked(MouseEvent me)
        {
                adapterDemo.showStatus("Mouse clicked");
        }
}

class MyMouseMotionAdapter extends MouseMotionAdapter
{
        AdapterDemo adapterDemo;

        public MyMouseMotionAdapter(AdapterDemo adapterDemo)
        {
                this.adapterDemo = adapterDemo;
        }

        // Handle only mouse dragged.
        public void mouseDragged(MouseEvent me)
        {
                adapterDemo.showStatus("Mouse dragged");
        }
}
```

I. Inner Classes

A class within another class is known as Inner class or Nested class. The scope of the inner is bounded by the scope of its enclosing class.



a. Static Nested Class

A static nested class is the one that has **static** modifier applied. Because it is static it cannot refer to non-static members of its enclosing class directly.

b. Non-static Nested class (Inner Classes)

Non-static Nested class is most important type of nested class. It is also known as **Inner** class. It has access to all variables and methods of **Outer** class and may refer to them directly. But the reverse is not true, that is, **Outer** class cannot directly access members of **Inner** class. One more important thing to notice about an **Inner** class is that it can be created only within the scope of **Outer** class. Java compiler generates an error if any code outside **Outer** class attempts to instantiate **Inner** class.

i. Example of Inner class:

```
class Outer{
    public void display(){
        Inner in=new Inner();
        in.show();
    }
    class Inner{
        public void show(){
            System.out.println("Inside inner");
        }
    }
    public static void main(String[] args){
        Outer ot=new Outer();
        ot.display();
    }
}
```

ii. Example of Inner class inside a method:

```
class Outer{
    int count;
    public void display(){
        for(int i=0;i<5;i++){
            class Inner{
                //Inner class defined inside for loop
                public void show(){
                    System.out.println("Inside inner "+(count++));
                }
            }
            Inner in=new Inner();
            in.show();
        }
    }
    public static void main(String[] args){
        Outer ot=new Outer();
        ot.display();
    }
}
```

Output :
Inside inner 0
Inside inner 1
Inside inner 2
Inside inner 3
Inside inner 4

iii. Example of Inner class instantiated outside Outer class

```
class Outer{
        int count;
        public void display(){
                Inner in=new Inner();
                in.show();
        }
        class Inner{
                public void show(){
                        System.out.println("Inside inner "+(++count));
                }
        }
}

class Test{
        public static void main(String[] args){
                Outer ot=new Outer();
                Outer.Inner in= ot.new Inner();
                in.show();
        }
}
```

Output :
Inside inner 1

iv. Anonymous class: A class without any name is called Anonymous class.

```
interface Animal{
        void type();
}

public class ATest {
        public static void main(String args[]){
                Animal an = new Animal(){
                        //Annonymous class created
                        public void type(){
                                System.out.println("Annonymous animal");
                        }
                };
                an.type();
        }
}
```

Output :
Annonymous animal

Unit-III
Networking & Security

Unit	Topic Name	Topics	Marks
3	**Networking**	Basic Concepts of Networking, Protocols: IP, TCP, UDP, IP Addresses, Proxy Servers, Sockets etc., InetAddress Class: Factory & Instance Methods, URL, URLConnection, HttpURLConnection, Socket & ServerSocket Class, DatagramPacket & DatagramSocket Class.	**14**
	Security	Introduction to Java Security, Java Security Model, Secure Programming Guidelines.	

A. Basic Concepts of Network

Computer Network: A computer network is a group of computer systems and other computing hardware devices that are linked together through communication channels to facilitate communication and resource-sharing among a wide range of users. Networks are commonly categorized based on their characteristics. Eg. LAN, MAN, WAN, Internet, Intranet, WiFi.

IP Address: IP address is short for Internet Protocol (IP) address. An IP address is an identifier for a computer or device on a TCP/IP network. Networks using the TCP/IP protocol route messages based on the IP address of the destination.

i. **IPv4:** Internet Protocol Version 4 (IPv4) is the fourth revision of the IP and a widely used protocol in data communication over different kinds of networks. It uses 32 bit IP address.
ii. **IPv6:** IPv6) is the successor to Internet Protocol Version 4 (IPv4). IPv6 was designed as an evolutionary upgrade to the Internet Protocol. It uses 128 bit IP address.

Subnet Mask: An IP address has two components, the network address and the host address. A subnet mask separates the IP address into the network and host addresses (<network><host>). A Subnet mask is a 32-bit number that masks an IP address, and divides the IP address into network address and host address. Subnet Mask is made by setting network bits to all "1"s and setting host bits to all "0"s.

Subnetting: Divides the host part of an IP address into a subnet and host address (<network><subnet><host>) if additional sub network is needed.

IP Classes: The IPv4 address space can be subdivided into 5 classes - Class A (1 - 126), B (128 - 191), C (192 - 223), D (224 - 239) and E (240 - 254). Each class consists of a contiguous subset of the overall IPv4 address range.

Class	First Octet or Series	Octets as Network vs. Host	Netmask Binary
A	1 – 126	Network.Host.Host.Host	1111 1111 0000 0000 0000 0000 0000 0000 or 255.0.0.0
B	128 – 191	Network.Network.Host.Host	1111 1111 1111 1111 0000 0000 0000 0000 or 255.255.0.0
C	192 – 223	Network.Network.Network.Host	1111 1111 1111 1111 1111 1111 0000 0000 or 255.255.255.0
D	*Defined for multicast operation and not used for normal operation*		
E	*Defined for experimental use and not used for normal operation*		

Client / Server: Client/server architecture is a computing model, in which the server hosts, delivers and manages most of the resources and services to be consumed by the client. This type of architecture has one or more client computers connected to a central server over a network or Internet connection. This system shares computing resources.

Client: A client is a piece of computer hardware or software that accesses a service made available by a server. Two types: Thin Client and Thick Client.

Server: A computer or device on a network that manages network resources. There are many different types of servers. For example: **File server, Print server, Network server, Database server.**

Proxy Server: A server that sits between a client application, such as a Web browser, and a real server. It intercepts all requests to the real server to see if it can fulfill the requests itself. If not, it forwards the request to the real server.

TCP (*Transmission Control Protocol)*: TCP is one of the main protocol in TCP/IP networks. TCP enables two hosts to establish a connection and exchange streams of data. TCP guarantees delivery of data and also guarantees that packets will be delivered in the same order in which they were sent.

UDP (*User Datagram Protocol*)**:** *UDP* is a connectionless protocol runs on top of IP networks. UDP/IP provides very few error recovery services, offering instead a direct way to send and receive datagram's over an IP network. It's used primarily for broadcasting messages over a network.

IP: The Internet Protocol (IP) is the method or protocol by which data is sent from one computer to another on the Internet. Each computer (known as a host) on the Internet has at least one IP address that uniquely identifies it from all other computers on the Internet.

TCP/IP: Transmission Control Protocol/Internet Protocol, TCP/IP is the suite of communications protocols used to connect hosts on the Internet. TCP/IP uses several protocols, the two main ones being TCP and IP.

URL (Uniform Resource Locator) it is the global address of documents and other resources on the World Wide Web. The first part of the URL is called a protocol identifier and it indicates what protocol to use and the second part is called a resource name and it specifies the IP address or the domain name where the resource is located. The protocol identifier and the resource name are separated by a colon and two forward slashes.

WWW: The World Wide Web (abbreviated as **WWW** or W3, commonly known as the Web) **is** a system of interlinked hypertext documents that are accessed via the Internet. With a web browser, one can view web pages that may contain text, images, videos, and other multimedia and navigate between them via hyperlinks. The documents are formatted in a markup language called HTML (Hypertext Markup Language).

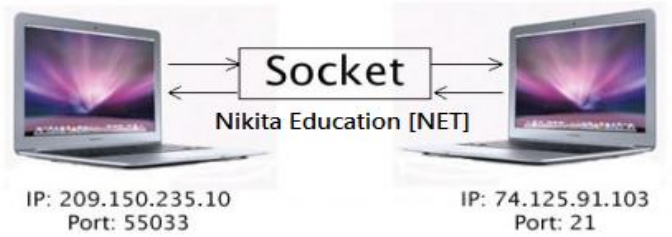
TCP vs. UDP

1. TCP stands for "**Transmission Control Protocol**" while UDP stands for "**User datagram Protocol**".
2. TCP is **connection oriented** protocol while UDP is **connectionless** protocol.
3. TCP is more **reliable** , UDP is **not reliable**.
4. UDP is **more faster** for data sending than TCP.
5. UDP makes **error checking but no reporting** but TCP makes **checks for errors and reporting**.

6. TCP gives **guarantee** that the **order of data** at receiving end is same as on sending end while UDP has **no such guarantee**.
7. Header size of TCP is **20 bytes** while that of UDP is **8 bytes**.
8. TCP is **heavy weight** as it needs three packets to setup a connection while UDP is **light weight**.
9. TCP has **acknowledgement** segments but UDP has **no acknowledgement**.
10. TCP is used for **application that require high reliability but less time critical** whereas UDP is used for application that are **time sensitive but require less reliability.**

B. Concept of Socket

Socket is a logical entity which provides end points to establish connection between processes or computers. Socket is a logical entity which consist IP address and port number to uniquely identify systems and processes over a network.



Types of sockets:

- **Stream Sockets**: It is a connection oriented socket provides reliable and sequenced delivery of information over a network using TCP protocol. Stream socket establishes a connection between client and server using TCP protocol.

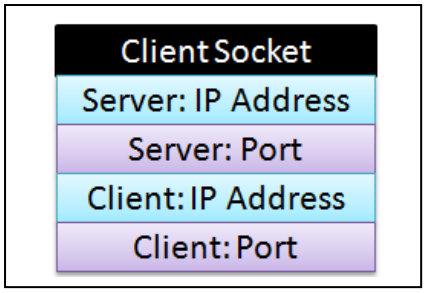
- **Datagram Sockets**: It is a connectionless socket provides faster but unreliable and non-sequenced delivery of information over a network using UDP protocol. Datagram socket sends different packets of information over a network through all available paths.

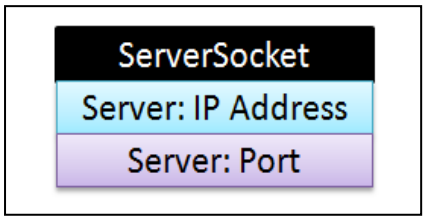
- **Raw Sockets**: These sockets provide users access to the underlying communication protocols, which support socket abstractions. These sockets are normally datagram oriented, though their exact characteristics are dependent on the interface provided by the protocol. Raw sockets are not intended for the general user; they have been provided mainly for those interested in developing new communication protocols, or for gaining access to some of the more cryptic facilities of an existing protocol.

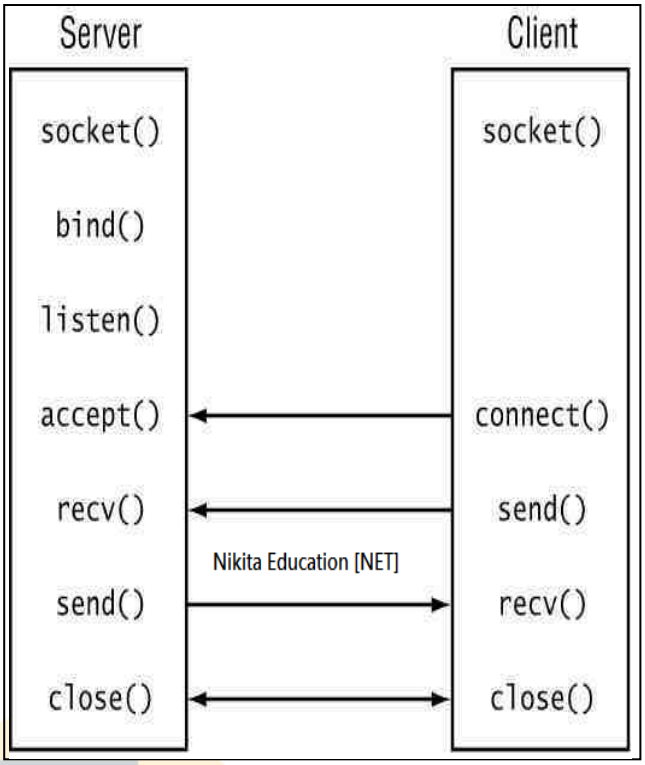
Socket Primitives:

Socket primitives defines different stages or methods through which socket performs different operations during communication between hosts. Following are different socket primitives.

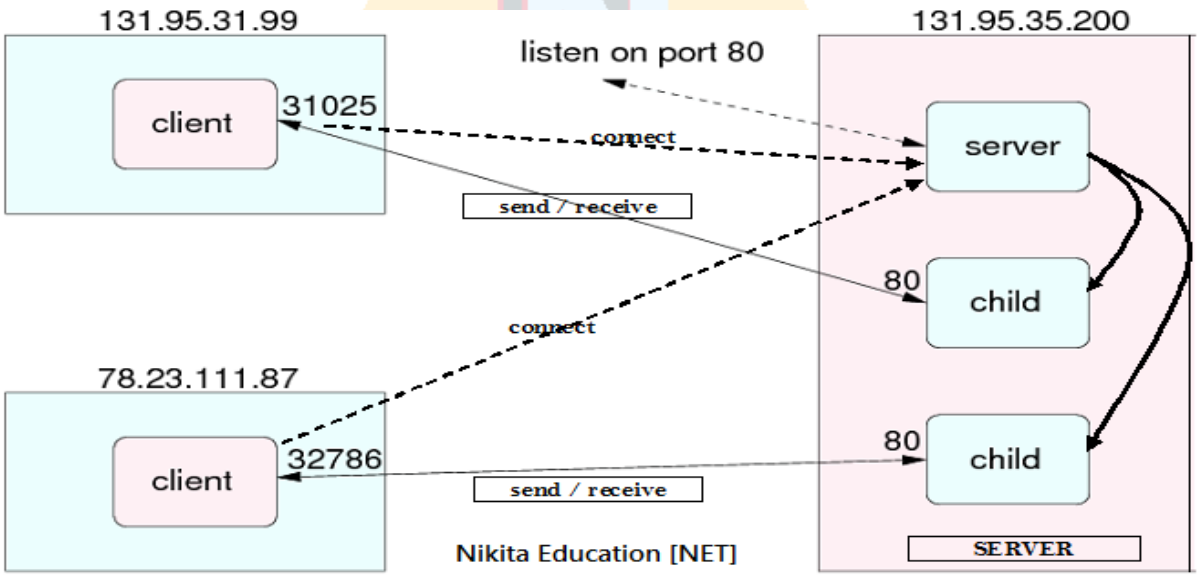
Primitive	Meaning
SOCKET	Create a new communication end point
BIND	Attach a local address to a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Block the caller until a connection attempt arrives
CONNECT	Actively attempt to establish a connection
SEND	Send some data over the connection
RECEIVE	Receive some data from the connection
CLOSE	Release the connection







How socket works:



C. Port

It is a sixteen bit unique number used to uniquely identify processes over a network. It is also called as a numbered socket. A port number is the logical address of each application or process that uses a network or the Internet to communicate. A port number uniquely identifies a network-based application on a computer. Each application/program is allocated a 16-bit integer port number. This number is assigned automatically by the OS, manually by the user or is set as a default for some popular applications.

Maximum Port Numbers: **2^{16} = 65536 i.e. 0 to 65535**

Types of Ports:

- **Well known/reserved ports:** Some port numbers are reserved for standard protocols and applications by IANA are called as well-known or reserved ports. These numbers are pre-assigned to standard protocols and no one can use these for general purposes. Total reserved ports are **1024** i.e. **0 to 1023.**

Port Number	Protocol/Process	Port Number	Protocol/Process
20	FTP Data	79	Finger
21	FTP Control	80	HTTP
22	SSH Remote Login Protocol	109, 110	POP2, POP3
23	Telnet	137	NetBIOS
25	SMTP	161	SNMP
43	WhoIs	179	BGP
53	Domain Name System (DNS)	389	LDAP
69	TFTP	443	HTTPS
70	Gopher	546, 547	DHCP Client, DHCP Server

- **Registered ports:** Some port numbers can be registered by software companies for their network applications or specific protocols are called as Registered Ports. Range of ports available for registration is : **1024 to 49151.**
- **Dynamic or private ports:** Port numbers **49152 to 65535** are available for general purposes or research purposes are called as dynamic or private ports. These port numbers can be used by anybody for their private use.

D. Network Classes and Interfaces

Java is a premier language for network programming. **java.net** package encapsulate large number of classes and interface that provides an easy-to use means to access network resources. Java supports TCP/IP both by extending the already established stream I/O interface introduced in Chapter I/O and by adding the features required to build I/O objects across the network. Java supports both the TCP and UDP protocol families. TCP is used for reliable stream-based I/O across the network. UDP supports a simpler, hence faster, point-to-point datagram-oriented model. Here are some important classes and interfaces of **java.net** package.

Classes:

InetAddress	This class represents an Internet Protocol (IP) address.
Inet4Address	This class represents an Internet Protocol version 4 (IPv4) address.
Inet6Address	This class represents an Internet Protocol version 6 (IPv6) address.
URL	Class URL represents a Uniform Resource Locator, a pointer to a "resource" on the World Wide Web.
URLConnection	The abstract class URLConnection is the superclass of all classes that represent a communications link between the application and a URL.
ServerSocket	This class implements server sockets.
Socket	This class implements client sockets (also called just "sockets").
DatagramPacket	This class represents a datagram packet.
DatagramSocket	This class represents a socket for sending and receiving datagram packets.
ContentHandler	The abstract class ContentHandler is the superclass of all classes that read an Object from a URLConnection.
CookieHandler	A CookieHandler object provides a callback mechanism to hook up a HTTP state management policy implementation into the HTTP protocol handler.
CookieManager	CookieManager provides a concrete implementation of **CookieHandler**, which separates the storage of cookies from the policy surrounding accepting and rejecting cookies.

HttpCookie	An HttpCookie object represents an http cookie, which carries state information between server and user agent.
HttpURLConnection	A URLConnection with support for HTTP-specific features.
NetPermission	This class is for various network permissions.

Interfaces:

CookiePolicy	CookiePolicy implementations decide which cookies should be accepted and which should be rejected.
CookieStore	A CookieStore object represents a storage for cookie.
ProtocolFamily	Represents a family of communication protocols.
SocketImplFactory	This interface defines a factory for socket implementations.
SocketOptions	Interface of methods to get/set socket options.
URLStreamHandlerFactory	This interface defines a factory for URL stream protocol handlers.

E. InetAddress

Inet Address encapsulates both numerical IP address and the domain name for that address. Inet address can handle both IPv4 and Ipv6 addresses. Inet Address class has no visible constructor. To create an inet Address object, you have to use **Factory methods**.

Factory methods

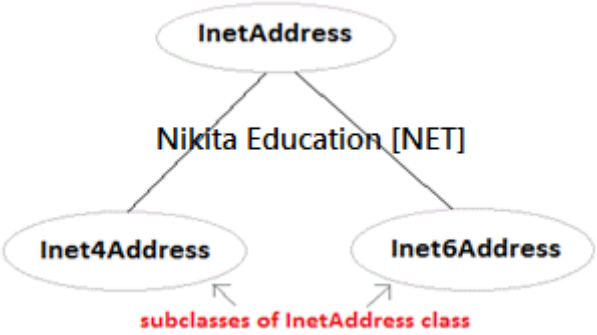
Factory methods are the static methods of any class that returns the object of same class where these methods are defined. Factory methods of InetAddress class returns an instance of InetAddress class because it does not provides any visible constructor.

- `static InetAddress` **`getLocalHost( )`** `throws UnknownHostException`
- `static InetAddress` **`getByName(String hostName)`** `throws UnknownHostException`
- `static InetAddress[]` **`getAllByName(String hostName)`** `throws UnknownHostExcep`

Instance Methods:

Instance methods are the non-static methods of any class which are defined in that class and accessed only by object of that class. Instance method means member methods of object of a class. Instance methods of InetAddress returns IP Address information of any host stored by its object.

boolean equals(Object *other*)	Returns **true** if this object has the same Internet address as *other*.
byte[ ] getAddress( )	Returns a byte array that represents the object's IP address in network byte order.
String getHostAddress( )	Returns a string that represents the host address associated with the **InetAddress** object.
String getHostName( )	Returns a string that represents the host name associated with the **InetAddress** object.
boolean isMulticastAddress( )	Returns **true** if this address is a multicast address. Otherwise, it returns **false**.
String toString( )	Returns a string that lists the host name and the IP address for convenience.

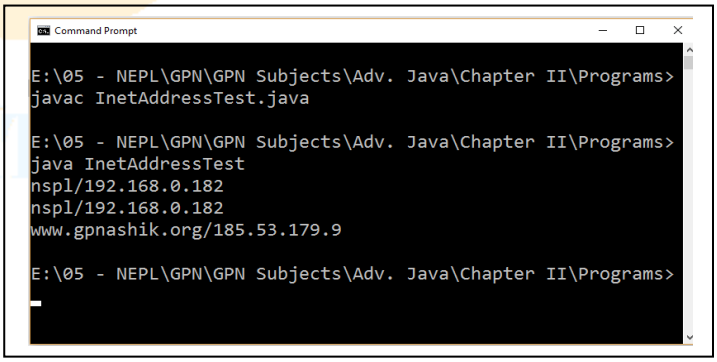


Inet4Address / Inet6Address

Inet4Address represents a traditional-style IPv4 address. **Inet6Address** encapsulates a new-style IPv6 address. Because they are subclasses of **InetAddress**, an **InetAddress** reference can refer to either. This is one way that Java was able to add IPv6 functionality without breaking existing code or adding many more classes. For the most part, you can simply use **InetAddress** when working with IP addresses because it can accommodate both styles.

Example of Factory Methods:

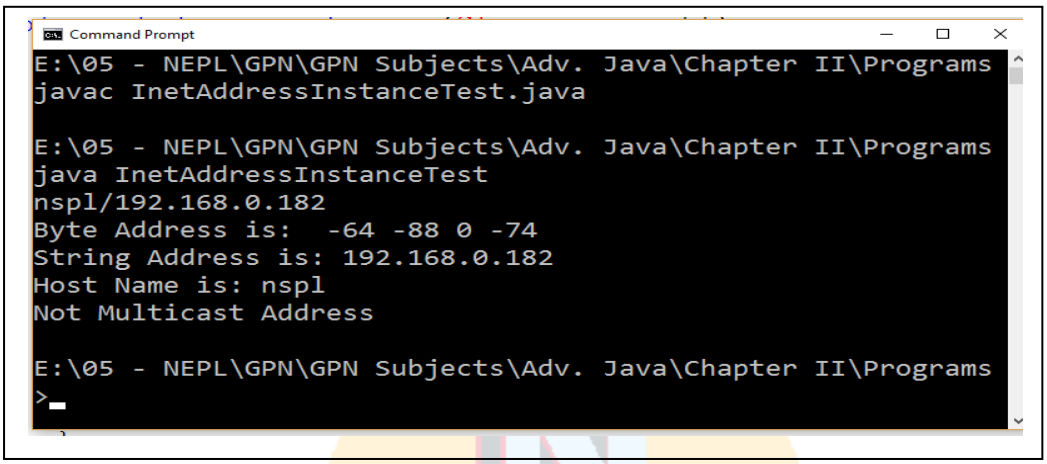
```
import java.net.*;

class InetAddressTest
{
	public static void main(String args[]) throws UnknownHostException
	{
		InetAddress Address = InetAddress.getLocalHost();
		System.out.println(Address);

		Address = InetAddress.getByName("nspl");
		System.out.println(Address);

		InetAddress SW[] = InetAddress.getAllByName("www.gpnashik.org");
		for (int i=0; i<SW.length; i++)
		{
			System.out.println(SW[i]);
		}
	}
}
```



Example of Instance Methods:

```
import java.net.*;
class InetAddressInstanceTest
{
	public static void main(String args[]) throws UnknownHostException
	{
		InetAddress Address = InetAddress.getLocalHost();
		System.out.println(Address);

		byte [] adr=Address.getAddress();
		String adrStr=Address.getHostAddress();
		String hostName=Address.getHostName();
		boolean mltcst=Address.isMulticastAddress();
		System.out.print("Byte Address is: ");
		for (int i=0;i<adr.length ;i++)
		{
			System.out.print(" "+adr[i]);
		}
```

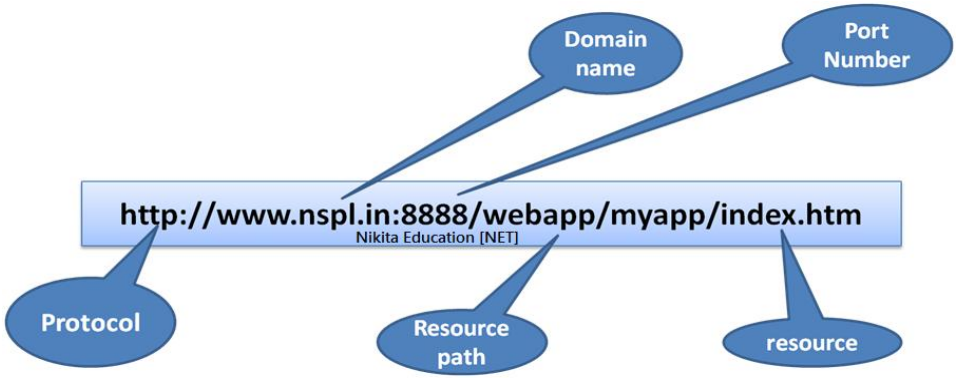
```
            System.out.println("\nString Address is: "+adrStr);
            System.out.println("Host Name is: "+hostName);
            if (mltcst==false)
            {
                  System.out.println("Not Multicast Address");
            }
            else
            {
                  System.out.println("Multicast Address");
            }

      }
}
```



F. URL

The URL (Uniform Resource Locator) provides a reasonably intelligible form to uniquely identify or address information on the Internet. URLs are ubiquitous; every browser uses them to identify information on the Web. Within Java's network class library, the **URL** class provides a simple, concise API to access information across the Internet using URLs. All URLs share the same basic format, although some variation is allowed. Here are two examples: **http://www.nspl.in/** and **http://www.nspl.in:8486/index.htm**.

A URL specification is based on **four components**: **(1)** The **protocol** to use, separated from the rest of the locator by a colon (:). **(2)** The **host name or IP address** of the host to use; this is delimited on the left by double slashes (//) and on the right by a slash (/) or optionally a colon (:). **(3)** The **port** number, is an optional parameter, delimited on the left from the host name by a colon (:) and on the right by a slash (/). (It defaults to port 80, the predefined HTTP port; thus, ":80" is redundant.) **(4)** The actual **file path.** Most HTTP servers will append a file named **index.html** or **index.htm** to URLs that refer directly to a directory resource. Thus, **http://www.nspl.in** is the same as **http://www.nspl.in/index.htm**.



To deal with above concept of URL, java.net package provides a URL class which has following constructors and methods:

Constructor and Description
URL(**String** spec) Creates a URL object from the String representation.
URL(**String** protocol, **String** host, int port, **String** file) Creates a URL object from the specified protocol, host, port number, and file.
URL(**String** protocol, **String** host, **String** file) Creates a URL from the specified protocol name, host name, and file name.
URL(**URL** context, **String** spec) Creates a URL by parsing the given spec within a specified context.

Method Summary

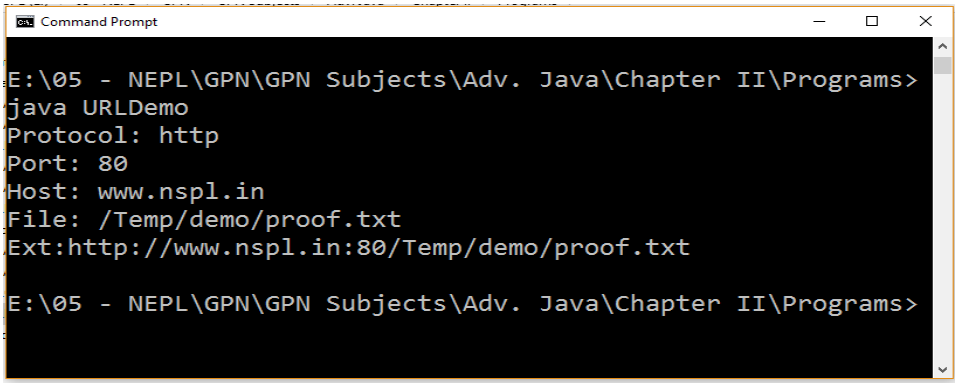
String	**getFile**() Gets the file name of this URL.
String	**getHost**() Gets the host name of this URL, if applicable.
String	**getPath**() Gets the path part of this URL.
int	**getPort**() Gets the port number of this URL.
String	**getProtocol**() Gets the protocol name of this URL.

To access the actual bits or content information of a URL, create a URLConnection object from it, using its openConnection( ) method.

- **Syntax:** URLConnection openConnection( ) throws IOException
- **Example:** URLConnection urlc = urlObject.openConnection()

Example:

```
import java.net.*;
class URLDemo
{
      public static void main(String args[])throws MalformedURLException
      {
            URL hp = new URL("http://www.nspl.in:80
                              /Temp/demo/proof.txt");
            System.out.println("Protocol: " + hp.getProtocol());
            System.out.println("Port: " + hp.getPort());
            System.out.println("Host: " + hp.getHost());
            System.out.println("File: " + hp.getFile());
            System.out.println("Ext:" + hp.toExternalForm());
      }
}
```



G. URLConnection

URLConnection is a general-purpose class for accessing the attributes of a remote resource. Once you make a connection to a remote server, you can use **URLConnection** to inspect the properties of the remote object before actually transporting it locally. To get object of URLConnection class use its openConnection() method on object of URL class.

Method Summary:

int	**getContentLength**() Returns the value of the content-length header field.
String	**getContentType()** **Returns the value of the content-type header field.**
long	**getDate()** **Returns the value of the date header field.**
long	**getExpiration()** **Returns the value of the expires header field.**
String	**getHeaderField(int n)** **Returns the value for the nth header field.**
String	**getHeaderField(String name)** **Returns the value of the named header field.**
InputStream	**getInputStream()** **Returns an input stream that reads from this open connection.**
OutputStream	**getOutputStream()** **Returns an output stream that writes to this connection.**
URL	**getURL()** **Returns the value of this URLConnection's URL field.**

Example:

```
import java.net.*;
import java.io.*;
import java.util.Date;
class URLConnectionDemo{
     public static void main(String args[]) throws Exception {
          int c;
          URL hp = new URL("http://www.nspl.in:80/Temp/index.html");
          URLConnection hpCon = hp.openConnection();

          // get date
          long d = hpCon.getDate();
          if(d==0)
               System.out.println("No date information.");
          else
               System.out.println("Date: " + new Date(d));

          // get content type
          System.out.println("Content-Type: "+hpCon.getContentType());

          // get expiration date
          d = hpCon.getExpiration();
          if(d==0)
               System.out.println("No expiration information.");
          else
               System.out.println("Expires: " + new Date(d));
```

```
            // get last-modified date
            d = hpCon.getLastModified();
            if(d==0)
                 System.out.println("No last-modified information.");
            else
                 System.out.println("Last-Modified: " + new Date(d));

            // get content length
            int len = hpCon.getContentLength();
            if(len == -1)
                 System.out.println("Content length unavailable.");
            else
                 System.out.println("Content-Length: " + len);
            if(len != 0){
                 System.out.println("=== Content ===");
                 InputStream input = hpCon.getInputStream();
                 int i = len;
                 while (((c = input.read()) != -1))
                 {
                      System.out.print((char) c);
                 }
                 input.close();
            }
            else
            {
                 System.out.println("No content available.");
            }
       }
}
```

H. HttpURLConnection

A URLConnection with support for HTTP-specific features. It is a subclass of URLConnection. Each HttpURLConnection instance is used to make a single request but the underlying network connection to the HTTP server may be transparently shared by other instances. You can obtain an HttpURLConnection in the same way just shown, by calling openConnection( ) on a URL object, but you must cast the result to HttpURLConnection. It is a subclass of URLConnection hence you can call all the methods of URLConnection class on this object.

Method Summary:

String	**getHeaderField**(int n) Returns the value for the nth header field.
String	**getRequestMethod()** Get the request method.
int	**getResponseCode()** Gets the status code from an HTTP response message.
String	**getResponseMessage()** Gets the HTTP response message, if any, returned along with the response code from a server.
void	**setRequestMethod(String method)** Set the method for the URL request, one of: GET POST HEAD OPTIONS PUT DELETE TRACE are legal, subject to protocol restrictions.

Example:

```
import java.net.*;
import java.io.*;
import java.util.*;
```

```
class HttpURLDemo{
  public static void main(String args[]) throws Exception{
     URL hp = new URL("http://www.nspl.in");
     HttpURLConnection hpCon = (HttpURLConnection) hp.openConnection();
     // Display request method.
     System.out.println("Request method is " +
     hpCon.getRequestMethod());
     // Display response code.
     System.out.println("Response code is " +
     hpCon.getResponseCode());
     // Display response message.
     System.out.println("Response Message is " +
     hpCon.getResponseMessage());
     // Get a list of the header fields and a set
     // of the header keys.
     Map<String, List<String>> hdrMap = hpCon.getHeaderFields();
     Set<String> hdrField = hdrMap.keySet();
     System.out.println("\nHere is the header:");
     // Display all header keys and values.
     for(String k : hdrField) {
             System.out.println("Key: "+k+"Value: "+ hdrMap.get(k));
         }
     }
}
```

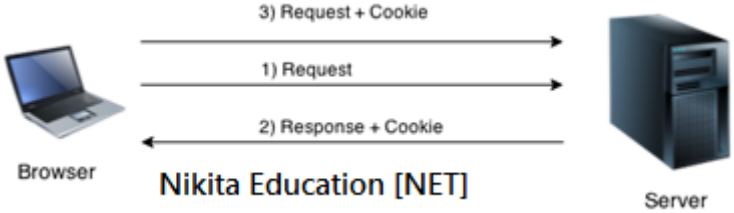
I. URI

A relatively recent addition to Java is the URI class, which encapsulates a *Uniform Resource Identifier (URI)*. URI's are similar to URL's. In fact, URLs constitute a subset of URIs. A URI represents a standard way to identify a resource. URL also describes how to access the resource.

J. Cookies

Cookies are small files which are stored on a user's computer. They are designed to hold a modest amount of data specific to a particular client and website, and can be accessed either by the web server or the client computer. A **cookie** is a small piece of information that is persisted between the multiple client requests. A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

The **java.net** package includes classes and interfaces that help manage cookies and can be used to create a stateful HTTP session. The classes are **CookieHandler, CookieManager**, and **HttpCookie**. The interfaces are **CookiePolicy** and **CookieStore**.



1. CookieManager

Constructor and Description
CookieManager() Create a new cookie manager.
CookieManager(CookieStore store, CookiePolicy cookiePolicy) Create a new cookie manager with specified cookie store and cookie policy.

Methods of CookieManager

Map<String,List<String>>	get(URI uri, Map<String,List<String>> requestHeaders) Gets all the applicable cookies from a cookie cache for the specified uri in the request header.
CookieStore	getCookieStore() To retrieve current cookie store.
void	put(URI uri, Map<String,List<String>> responseHeaders) Sets all the applicable cookies, examples are response header fields that are named Set-Cookie2, present in the response headers into a cookie cache.
void	setCookiePolicy(CookiePolicy cookiePolicy) To set the cookie policy of this cookie manager.

K. Stream Socket

TCP/IP sockets are used to implement reliable, bidirectional, persistent, point-to-point, and stream-based connections between hosts on the Internet. A socket can be used to connect Java's I/O system to other programs that may reside either on the local machine or on any other machine on the Internet. There are two kinds of TCP sockets in Java. **One is for servers, and the other is for clients.**

The **ServerSocket** class is designed to be a "listener," which waits for clients to connect before doing anything. Thus, **ServerSocket** is for servers.

The **Socket** class is for clients. It is designed to connect to server sockets and initiate protocol exchanges. Because client sockets are the most commonly used by Java applications, they are examined here. The creation of a **Socket** object implicitly establishes a connection between the client and server.

- **ServerSocket** is for servers.
- **Socket** class is for client.

K.1. ServerSocket

The **ServerSocket** class is used to create servers that listen for either local or remote client programs to connect to them on published ports. **ServerSocket**s are quite different from normal **Socket**s. When you create a **ServerSocket**, it will register itself with the system as having an interest in client connections. The constructors for **ServerSocket** reflect the port number that you want to accept connections on and, optionally, how long you want the queue for said port to be. The queue length tells the system how many client connections it can leave pending before it should simply refuse connections. **The default is 50**. The constructors might throw an **IOException** under adverse conditions.

Commonly used constructors:

public ServerSocket(int port) throws IOException Attempts to create a server socket bound to the specified port. An exception occurs if the port is already bound by another application.
public ServerSocket(int port, int backlog) throws IOException Similar to the previous constructor, the backlog parameter specifies how many incoming clients to store in a wait queue.
public ServerSocket(int port, int backlog, InetAddress address) throws IOException Similar to the previous constructor, the InetAddress parameter specifies the local IP address to bind to. The InetAddress is used for servers that may have multiple IP addresses, allowing the server to specify which of its IP addresses to accept client requests on.
public ServerSocket() throws IOException Creates an unbound server socket. When using this constructor, use the bind() method when you are ready to bind the server socket.

Commonly used methods:

public int getLocalPort() Returns the port that the server socket is listening on. This method is useful if you passed in 0 as the port number in a constructor and let the server find a port for you.
public Socket accept() throws IOException Waits for an incoming client. This method blocks until either a client connects to the server on the specified port or the socket times out, assuming that the time-out value has been set using the setSoTimeout() method. Otherwise, this method blocks indefinitely.
public void bind(SocketAddress host, int backlog) Binds the socket to the specified server and port in the SocketAddress object. Use this method if you have instantiated the ServerSocket using the no-argument constructor.

K.2. Socket

A socket is simply an endpoint for communications between the machines. The Socket class can be used to create a socket. The **java.net.Socket** class represents the socket that both the client and the server use to communicate with each other. The client obtains a Socket object by instantiating one, whereas the server obtains a Socket object from the return value of the accept() method.

Commonly used constructors:

public Socket(String host, int port) throws UnknownHostException, IOException. This method attempts to connect to the specified server at the specified port. If this constructor does not throw an exception, the connection is successful and the client is connected to the server.
public Socket(InetAddress host, int port) throws IOException This method is identical to the previous constructor, except that the host is denoted by an InetAddress object.
public Socket(String host, int port, InetAddress localAddress, int localPort) throws IOException. Connects to the specified host and port, creating a socket on the local host at the specified address and port.
public Socket(InetAddress host, int port, InetAddress localAddress, int localPort) throws IOException. This method is identical to the previous constructor, except that the host is denoted by an InetAddress object instead of a String.
public Socket() Creates an unconnected socket. Use the connect() method to connect this socket to a server.

Commonly used methods:

public void connect(SocketAddress host, int timeout) throws IOException This method connects the socket to the specified host. This method is needed only when you instantiate the Socket using the no-argument constructor.
public InetAddress getInetAddress() This method returns the address of the other computer that this socket is connected to.
public int getPort() Returns the port the socket is bound to on the remote machine.
public int getLocalPort() Returns the port the socket is bound to on the local machine.

public SocketAddress getRemoteSocketAddress() Returns the address of the remote socket.
public InputStream getInputStream() throws IOException Returns the input stream of the socket. The input stream is connected to the output stream of the remote socket.
public OutputStream getOutputStream() throws IOException Returns the output stream of the socket. The output stream is connected to the input stream of the remote socket.
public void close() throws IOException Closes the socket, which makes this Socket object no longer capable of connecting again to any server.

Example: Chat Application between client and server

Program 1: MyServer.java

```
import java.net.*;
import java.io.*;

class MyServer
{
     public static void main(String args[])throws Exception
     {
          ServerSocket ss=new ServerSocket(3333);
          Socket s=ss.accept();
          DataInputStream din=new DataInputStream(s.getInputStream());
          DataOutputStream dout=
                         new DataOutputStream(s.getOutputStream());
          BufferedReader br=new BufferedReader(
                                   new InputStreamReader(System.in));
          String str="",str2="";
          while(!str.equals("NET"))
          {
               str=din.readUTF();
               System.out.println("client says: "+str);
               str2=br.readLine();
               dout.writeUTF(str2);
               dout.flush();
          }
          din.close();
          s.close();
          ss.close();
     }
}
```

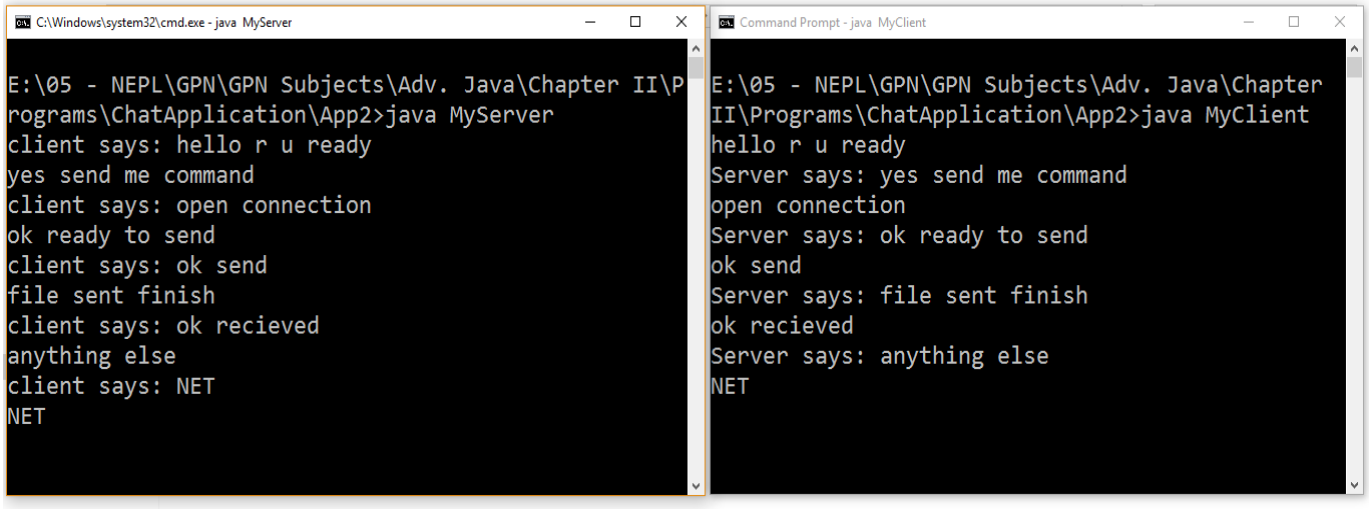
Program 2: MyClient.java

```
import java.net.*;
import java.io.*;
class MyClient
{
     public static void main(String args[])throws Exception
     {
          Socket s=new Socket("localhost",3333);
          DataInputStream din=new DataInputStream(s.getInputStream());
          DataOutputStream dout=
                         new DataOutputStream(s.getOutputStream());
```

```
            BufferedReader br=new BufferedReader(
                                    new InputStreamReader(System.in));
            String str="",str2="";
            while(!str.equals("NET"))
            {
                  str=br.readLine();
                  dout.writeUTF(str);
                  dout.flush();
                  str2=din.readUTF();
                  System.out.println("Server says: "+str2);
            }
            dout.close();
            s.close();
      }
}
```

How to run above chat application

- **Step 1:** compile both files using javac i.e. MyServer and MyClient
- **Step 2:** open two saperate command prompts one for server and another for client
- **Step 3:** run MyServer in one cmd and MyClient in another cmd using java command
- **Step 4:** first type message from MyClient cmd then press enter
- **Step 5:** now type message from MyServer cmd and go on between client and server
- **Step 6:** finally type NET in both cmd to finish communication



L. Datagram Socket

Datagram socket is a type of interprocess communications socket or network socket which provides a connectionless point for sending or receiving data packets. Each packet sent or received on a datagram socket is individually addressed and routed. Order and reliability are not guaranteed with datagram sockets, so multiple packets sent from one machine or process to another may arrive in any order or might not arrive at all.

A ***datagram*** is an independent, self-contained message sent over the network whose arrival, arrival time, and content are not guaranteed. It is a small piece of information transmitted over a connectionless protocol. *Datagrams* are bundles of information passed between machines.

Java **DatagramSocket** and **DatagramPacket** classes are used for connection-less socket programming.

L.1. DatagramSocket

Java DatagramSocket class represents a connection-less socket for sending and receiving datagram packets. A datagram is basically an information but there is no guarantee of its content, arrival or arrival time.

Commonly used Constructors

- **DatagramSocket() throws SocketEeption:** it creates a datagram socket and binds it with the available Port Number on the localhost machine.
- **DatagramSocket(int port) throws SocketEeption:** it creates a datagram socket and binds it with the given Port Number.
- **DatagramSocket(int port, InetAddress address) throws SocketEeption:** it creates a datagram socket and binds it with the specified port number and host address.

Commonly used methods

void	**bind**(**SocketAddress** addr) Binds this DatagramSocket to a specific address and port.
InetAddress	**getInetAddress**() Returns the address to which this socket is connected.
InetAddress	**getLocalAddress**() Gets the local address to which the socket is bound.
int	**getLocalPort**() Returns the port number on the local host to which this socket is bound.
int	**getPort**() Returns the port number to which this socket is connected.
int	**getReceiveBufferSize**() Get value of the SO_RCVBUF option for this DatagramSocket, that is the buffer size used by the platform for input on this DatagramSocket.
boolean	**isConnected**() Returns the connection state of the socket.
void	**receive**(**DatagramPacket** p) Receives a datagram packet from this socket.
void	**send**(**DatagramPacket** p) Sends a datagram packet from this socket.

L.2. DatagramPacket

DatagramPacket is a message that can be sent or received. If you send multiple packet, it may arrive in any order. Additionally, packet delivery is not guaranteed.

Constructor and Description
DatagramPacket(byte[] buf, int length) Constructs a DatagramPacket for receiving packets of length length.
DatagramPacket(byte[] buf, int length, InetAddress address, int port) Constructs a datagram packet for sending packets of length length to the specified port number on the specified host.
DatagramPacket(byte[] buf, int offset, int length, InetAddress address, int port) Constructs a datagram packet for sending packets of length length with offset ioffsetto the specified port number on the specified host.

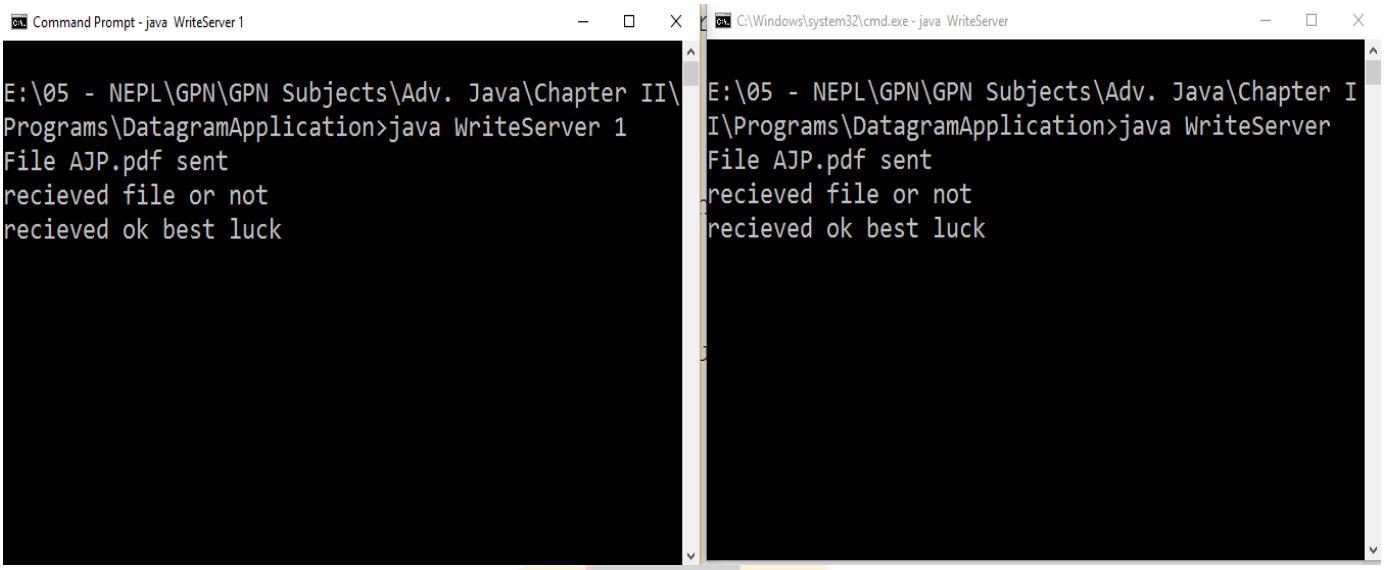
Modifier	**Method and Description**
InetAddress	getAddress() Returns the IP address of the machine to which this datagram is being sent or from which the datagram was received.
byte[]	getData() Returns the data buffer.

int	**getLength**() Returns the length of the data to be sent or the length of the data received.
int	**getPort**() Returns the port number on the remote host to which this datagram is being sent or from which the datagram was received.
void	**setPort**(int iport) Sets the port number on the remote host to which this datagram is being sent.

Example:

```
import java.net.*;
class WriteServer{
      public static int serverPort = 998;
      public static int clientPort = 999;
      public static int buffer_size = 1024;
      public static DatagramSocket ds;
      public static byte buffer[] = new byte[buffer_size];

      public static void TheServer() throws Exception
      {
            int pos=0;
            while (true)
            {
                  int c = System.in.read();
                  switch (c)
                  {
                        case -1:
                              System.out.println("Server Quits.");
                              return;
                        case '\r':
                              break;
                        case '\n':
                              ds.send(new DatagramPacket(buffer,pos,
                              InetAddress.getLocalHost(),clientPort));
                              pos=0;
                          break;
                          default:
                              buffer[pos++] = (byte) c;
                  }
            }
      }

      public static void TheClient() throws Exception
      {
            while(true)
            {
            DatagramPacket p = new DatagramPacket(buffer, buffer.length);
            ds.receive(p);
            System.out.println(new String(p.getData(), 0, p.getLength()));
            }
      }

      public static void main(String args[]) throws Exception
      {
            if(args.length == 1)
            {
                  ds = new DatagramSocket(serverPort);
                  TheServer();
            }
            else
```

```
            {
                  ds = new DatagramSocket(clientPort);
                  TheClient();
            }
      }
}
```



M. Java Security

The Java platform provides a number of features designed for improving the **security** of Java applications. This includes enforcing runtime constraints through the use of the Java Virtual Machine (JVM), a security manager that sandboxes un-trusted code from the rest of the operating system, and a suite of security APIs that Java developers can utilize.

1. The JVM:

The binary form of programs running on the Java platform is not native machine code but an intermediate bytecode. The JVM performs verification on this bytecode before running it to prevent the program from performing unsafe operations such as branching to incorrect locations, which may contain data rather than instructions. It also allows the JVM to enforce runtime constraints such as array bounds checking. This means that Java programs are significantly less likely to suffer from memory safety flaws such as buffer overflow than programs written in languages such as C which do not provide such memory safety guarantees. The platform does not allow programs to perform certain potentially unsafe operations such as pointer arithmetic or unchecked type casts. It also does not allow manual control over memory allocation and de-allocation; users are required to rely on the automatic garbage collection provided by the platform. This also contributes to type safety and memory safety.

2. Security manager

The platform provides a security manager which allows users to run un-trusted bytecode in a "sandboxed" environment designed to protect them from malicious or poorly written software by preventing the un-trusted code from accessing certain platform features and APIs. For example, un-trusted code might be prevented from reading or writing files on the local file system, running arbitrary commands with the current user's privileges, accessing communication networks, accessing the internal private state of objects using reflection, or causing the JVM to exit.
The security manager also allows Java programs to be cryptographically signed; users can choose to allow code with a valid digital signature from a trusted entity to run with full privileges in circumstances where it would otherwise be un-trusted.
Users can also set fine-grained access control policies for programs from different sources. For example, a user may decide that only system classes should be fully trusted, that code from certain trusted entities may be allowed to read certain specific files, and that all other code should be fully sandboxed.

3. Security APIs

The Java Class Library provides a number of APIs related to security, such as standard cryptographic algorithms, authentication, and secure communication protocols. Java provides three main packages for the security:

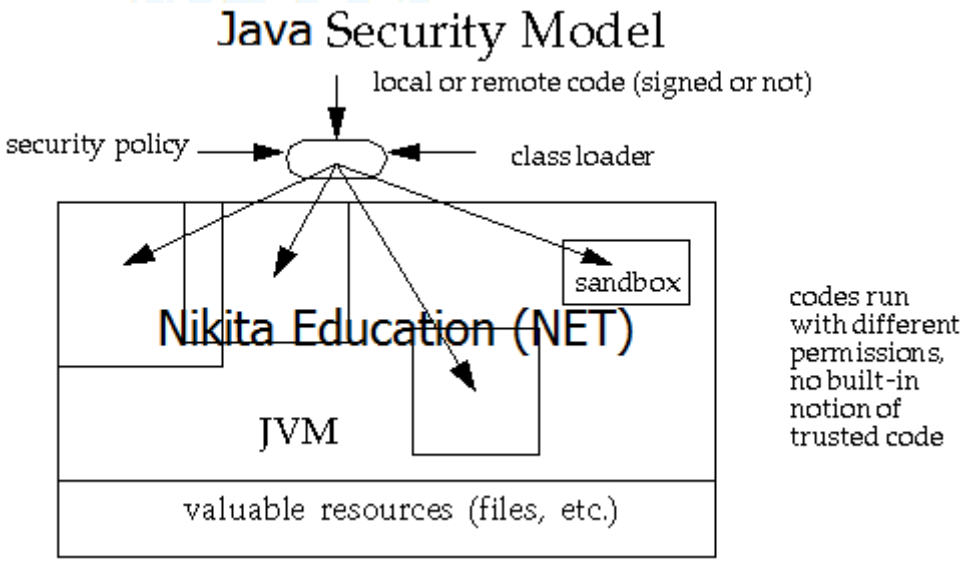
1. Java Cryptography Extension (JCE) provides
   - Encryption
   - Secure keys exchange
   - Secure message digest
   - An alternate key management system
2. Java Secure Socket Layer (JSSL) provides
   - Tools for secure communications
3. Java Authentication and Authorization System (JAAS) provides
   - Tools for authenticating the user of a program and authorize or deny the access to sensitive data.
   - Protect end user from the influence of developer
   - End user gives permissions to the developer to access resources of the user's end machine
   - JAAS allows developer to grant or deny access to their programs based on the credentials provided by the users

M.1. Java Security Model (The Sandbox)

The "Sandbox"

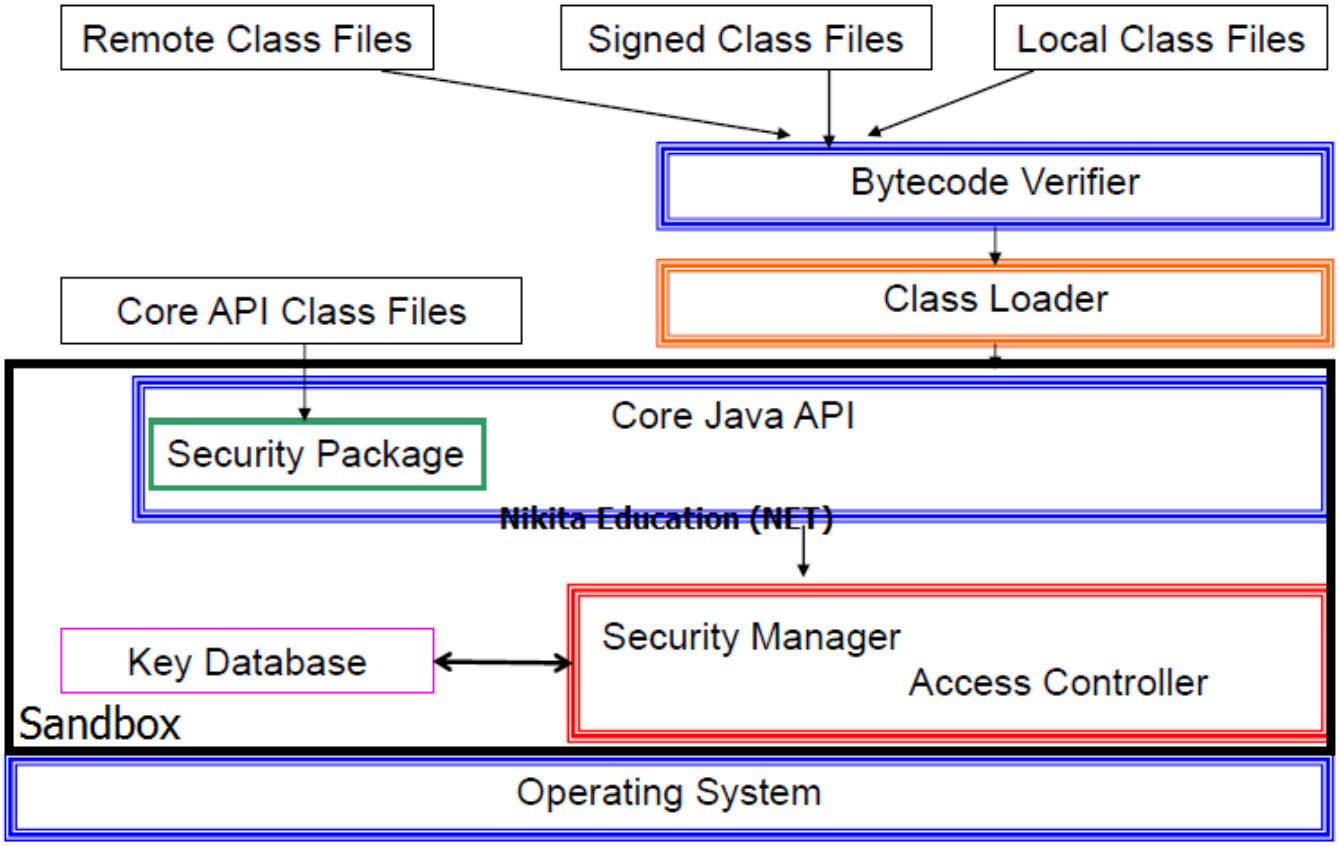
Java security relies entirely on software technology. Java accepts all downloaded programs and runs them within a security "sandbox". This can be looked at as a security fence that surrounds the program and keeps it away from your private data. As long as that fence has no holes, data is safe. Java security relies on the software implementing the sandbox to work correctly. A Java program must play only inside its sandbox. It can do anything within the boundaries of its sandbox, but it can't take any action outside of those boundaries. For example, the sandbox for un-trusted Java applets prohibits many activities including:

- Reading or writing to the local disk
- Making a network connection to any host, except the host from which the applet came
- Creating a new process
- Loading a new dynamic library and directly calling a native method



Features of Sandbox:

- Fine-grained access control.
- Easily configurable security policy.
- Easily extensible access control structure.
- Extension of security checks to all Java programs, including applications as well as applets.



Byte-code verifier
Since Java code can be imported from anywhere in the network, it is critical to screen the code to be sure that it was produced by a trustworthy compiler. The *byte-code verifier,* sometimes referred to as a mini-theorem prover, tries to prove that a given series of Java byte codes are legal.

ClassLoader
The ClassLoader, which loads Java byte codes into the JVM, is an important link in the security chain. It works in conjunction with the SecurityManager and the access controller to enforce security rules.

CodeSource
The CodeSource encapsulates the code's origin, which is specified as an URL, and the set of digital certificates containing public keys corresponding to the set of private keys used to sign the code.

Permissions
Permission classes are at the very core of Java security and represent access to various system resources such as files, sockets, and so on. A collection of permissions can be construed as a customizable security policy for an installation.

Protection domains
It's possible to associate permissions with classes; however, it's more flexible to group classes into protection domains and associate permissions with those domains.

Policy
The numerous mappings of permissions to classes are collectively referred to as *policy*. A policy file is used to configure the policy for a particular implementation. It can be composed by a simple text editor or using policy-tool.

SecurityManager
The class java.lang.SecurityManager is at the focal point of authorization in the implementation of the sandbox model. SecurityManager is concrete, with a public constructor and appropriate checks in place to ensure that it can be invoked in an authorized manner. SecurityManager consists of a number of *check* methods.

AccessController

The java.security.AccessController class is used for three purposes:

- To decide whether access to a critical system resource should be allowed or denied, based on the security policy currently in effect
- To mark code as privileged, thus affecting subsequent access determinations
- To obtain a snapshot of the current calling context, so access-control decisions from a different context can be made with respect to the saved context

The Keystore

The Keystore is a password-protected database that holds private keys and certificates. The password is selected at the time of creation. Each database entry can be guarded by its own password for extra security. Certificates accepted into the Keystore are considered to be trusted. Keystore information can be used and updated by the security tools provided with the SDK.

Aspects of Java 2 Security

Aspects of Java 2 Security can be broadly classified as:

- **Core security:** the core classes that deal with security
- **Security extensions:** the optional packages that supplement the platform security
- **Security tools:** the Java 2 Software Development Kit (SDK) tools pertaining to security
- **Application, applet, and plugin security:** security deployment

Core security

Java 2's security pieces reside primarily in:

- java.lang
- java.security
- java.security.cert
- java.security.interfaces
- java.security.spec

N. Secure Coding Guidelines

Producing secure programs requires secure designs. However, even the best designs can lead to insecure programs if developers are unaware of the many security pitfalls inherent in Java programming. Secure coding guidelines are important to achieve following objectives:

- Improve the overall security of any Java application
- Avoid injection attacks, such as SQL injection and XSS
- Understand Java's memory model, with a thorough grounding of concurrency, and learn how to prevent race conditions while avoiding deadlock
- Learn when to throw and catch exceptions
- Avoid I/O vulnerabilities, including file-based race conditions
- Learn how historical exploits on Java were executed and later disabled

Following are the secure coding guidelines:

N.1. Fundamental

- Prefer to have obviously no flaws rather than no obvious flaws
- Design APIs to avoid security concerns
- Avoid duplication
- Restrict privileges
- Establish trust boundaries
- Minimize the number of permission checks
- Encapsulate
- Document security-related information

N.2. Denial of Service

- Beware of activities that may use disproportionate resources
- Release resources in all cases
- Resource limit checks should not suffer from integer overflow

N.3. Confidential Information

- Purge sensitive information from exceptions
- Do not log highly sensitive information
- Consider purging highly sensitive from memory after use

N.4. Accessibility and Extensibility

- Limit the accessibility of classes, interfaces, methods, and fields
- Limit the accessibility of packages
- Isolate unrelated code
- Limit exposure of ClassLoader instances
- Limit the extensibility of classes and methods
- Understand how a superclass can affect subclass behavior

N.5. Input Validation

- Validate inputs
- Validate output from untrusted objects as input
- Define wrappers around native methods

Unit-IV
Java Database Connectivity

Unit	Topic Name	Topics	Marks
4	**JDBC**	Basic Database Concepts, Introduction to JDBC, Two-Tier & Three-Tier Database Architectures, JDBC Features, Types of JDBC Drivers, JDBC Classes & Interfaces, CRUD Operations (Create, Read, Update, & Delete).	**16**

A. Database Terminologies

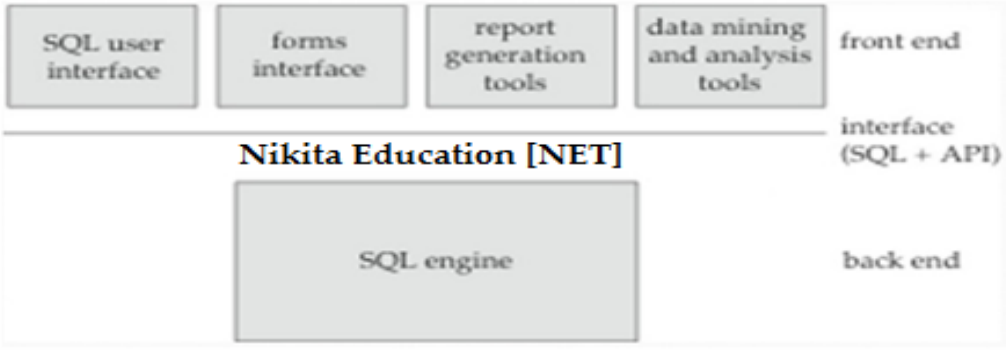
Front end
It is a set of programs usually a graphical user interface (GUI) through which end user can interact with the application or system. Java programming API's such as AWT, Swing, Applets, Servlets, or JSP's are used to design GUI's or Web applications through which end user can interact with the application or system. So the programs designed using these API's provides different services and features to end user.

Back end
It is a background or behind the scene service or application usually a database that stores information passed by front end or responds to queries asked by front end through some intermediate. Software's like Oracle, Sybase, MS SQL Server, MySQL, MS Access are known as a back end.

Java acts as a database front end
Java programming API's such as AWT, Swing, Applets, Servlets, or JSP's are used to design GUI's or Web applications through which end user can interact with the application or system. These programs collect some useful information of user interactions and generally store it into the databases. **Hence java acts as a database front end**.

JDBC acts as a mediator
It performs different operations between **Front end** and **Back end. JDBC** passes front end method calls to back end in back end understandable formats and provides results back to front end in programming formats.

Database Client/Server Methodology:
The client–server model of computing is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. Client-Server model splits the processing of an application between front end client and back end processor. In DBMS system, database functionality can be divided into**: (1) Back end**: manages access structures, query evaluation and optimization, concurrency control and recovery. **(2)Front end:** consist of tools such as forms, report writers, and graphical user interface facilities. The interface between the front end and back end is through the SQL or through an application program interface. Client and server may reside on same computer.



Structure of typical DBMS systems

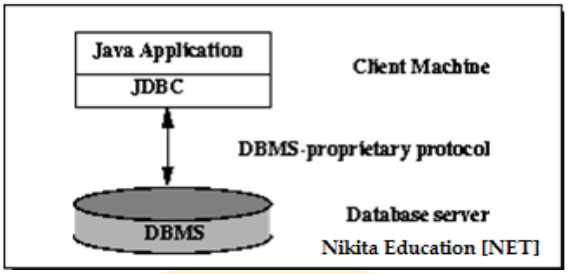
Advantage of this design is Flexibility in locating resources and expanding facilities, Better functionality for the cost, better user interface, and easier maintenance. **Hence DBMS system is implemented on Client / Server methodology.**

1. Two Tier Database Design

The two-tier is based on Client Server architecture. The two-tier architecture is like client server application. The direct communication takes place between client and server. There is no intermediate between client and server. Because of tight coupling a 2 tiered application will run faster. In two tier database design user interface programs and application programs for database access run on client machine. The interface program called JDBC provides an API that allows application programs to call DBMS through JDBC drivers. The Two-tier architecture is divided into two parts:

1) **Client Application (Client Tier)**
2) **Database (Data Tier)**

On client application side the code is written for saving the data in the SQL server database. Client sends the request to server and it process the request & send back with data. The main problem of two tier architecture is the server cannot respond multiple request same time, as a result it cause a data integrity issue.



Advantages:

1. Easy to maintain and modification is bit easy
2. Communication is faster

Disadvantages:

1. In two tier architecture application performance will be degrade upon increasing the users.
2. Cost-ineffective

2. Three Tier Database Design:

Three-tier architecture typically comprises a presentation tier, a business or data access tier, and a data tier. Three layers in the three tier architecture are as follows:

1) **Client layer**
2) **Business layer**
3) **Data layer**

1) Client layer:

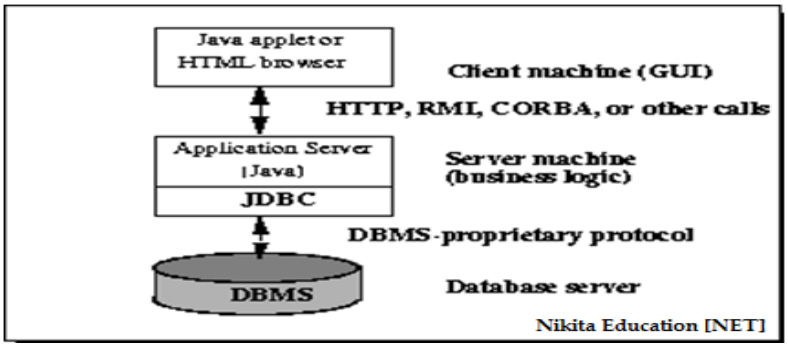
It is also called as *Presentation layer* which contains UI part of our application. This layer is used for the design purpose where data is presented to the user or input is taken from the user. For example designing registration form which contains text box, label, button etc.

2) Business layer:

In this layer all business logic written likes validation of data, calculations, data insertion etc. This acts as a interface between Client layer and Data Access Layer. This layer is also called the intermediary layer helps to make communication faster between client and data layer.

3) Data layer:

In this layer actual database is comes in the picture. Data Access Layer contains methods to connect with database and to perform insert, update, delete, get data from database based on our input data.



Advantages

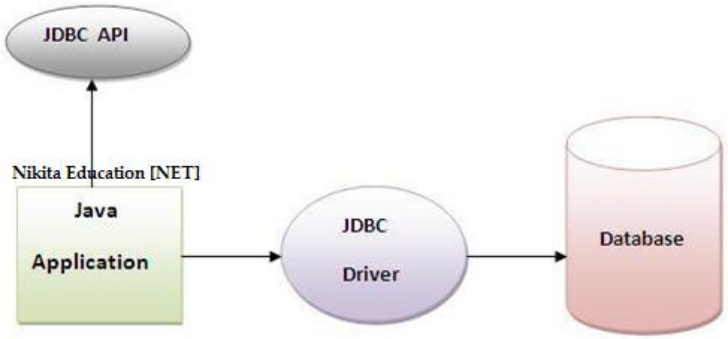
1. High performance, lightweight persistent objects
2. Scalability – Each tier can scale horizontally
3. Performance – Because the Presentation tier can cache requests, network utilization is minimized, and the load is reduced on the Application and Data tiers.
4. High degree of flexibility in deployment platform and configuration
5. Better Re-use
6. Improve Data Integrity
7. Improved Security – Client is not direct access to database.
8. Easy to maintain and modification is bit easy, won't affect other modules
9. In three tier architecture application performance is good.

Disadvantages

1. Increase Complexity/Effort

B. JDBC

Java Database Connectivity (**JDBC**) is an application program interface (**API**) specification for connecting programs written in Java to the data in popular databases. The application program interface lets you encode access request statements in Structured Query Language (SQL) that are then passed to the program that manages the database. It returns the results through a similar interface. Java JDBC is a java API to connect and execute query with the database. JDBC API uses jdbc drivers to connect with the database.

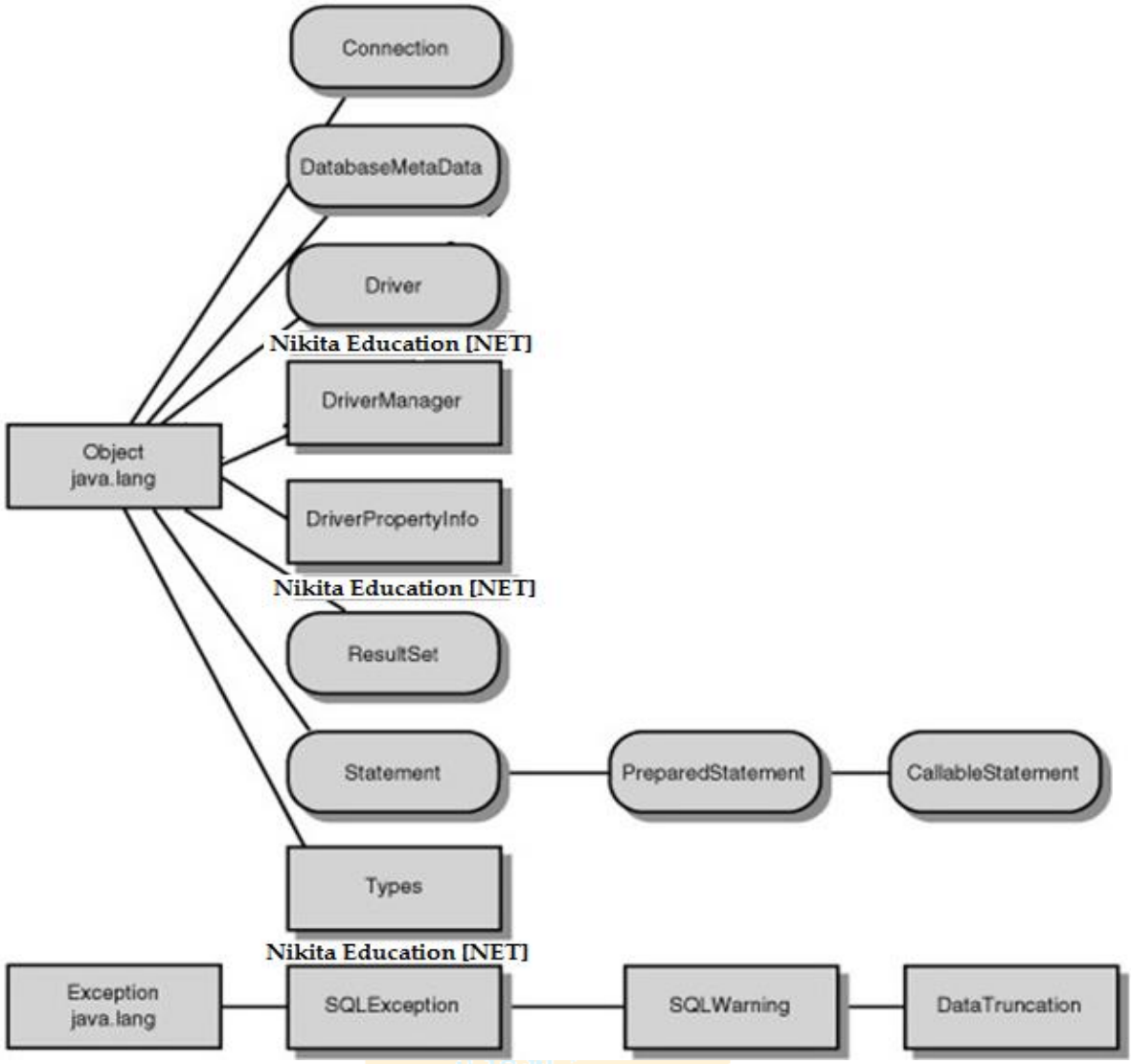


JDBC Features:

- **Get a connection**: A java application can be connected to a database either using DriverManager or DataSource object.
- **Connection Pooling**: It allows the java application to reuse database connection that has been created already instead of creating a new connection every time.
- **Rowsets:** The rowsets object contains the tabular data. It makes the possible to pass the rows data to the network. Therefore they are widely used in distributed application.
- **New data type supports:** This is the ability of JDBC to manipulate large object such as BLOB and CLOB without bringing them to the java programmer from the database server.
- **Batch Updating:** This feature provides the ability to send multiple updates to the database to be executed as batch rather than sending each update separately.
- **Result set enhancement:**
  - *Scrollable Result set:* It provides the ability to move the cursor backward and forward to a specific position.
  - *Updateable Result set*: It allows the modification of data in a database table using result set.
- **Savepoints:** JDBC contains a Savepoint interface which contains a new method to set a savepoints, to release a save point and to rollback a transaction to desired savepoints

C. JDBC API

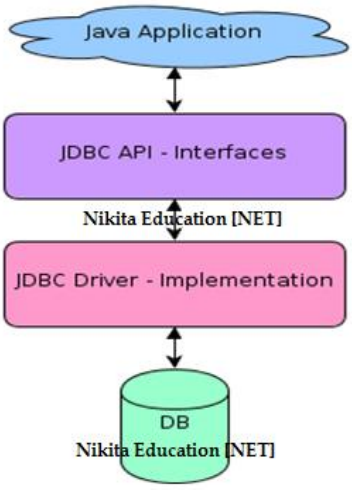
JDBC API components are nothing but the collection of classes and interfaces which are necessary for establishing communication with databases and java applications. Following are the important JDBC API:



The JDBC API is defined by two packages in Java:

- The core API **java.sql** provides the API for accessing and processing the data stored in a database (usually a relational database) using Java. This package provides the foundation and most commonly used objects such as Connection, ResultSet, Statement, and PreparedStatement. The DriverManager class is the main component here that loads drivers and retrieves a connection.

- The extension API **javax.sql** provides the API for server-side data source access and processing from Java. This package provides services for Java EE such as DataSource and RowSet. The DataSource interface is an alternative to DriverManager for establishing a connection with a data source. JDBC extension API also provides extension services such as connection pooling

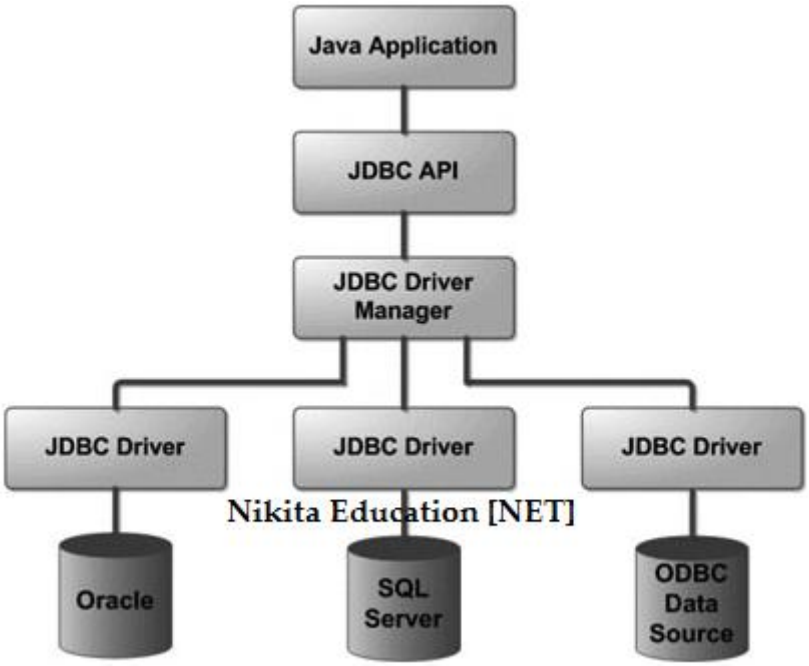


D. JDBC Architecture

The JDBC API supports both two-tier and three-tier processing models for database access but in general JDBC Architecture consists of two layers:

- **JDBC API:** This provides the application-to-JDBC Manager connection.
- **JDBC Driver API:** This supports the JDBC Manager-to-Driver Connection.

The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases. The JDBC driver manager ensures that the correct driver is used to access each data source. The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases.



JDBC Architecture

E. JDBC Components

The JDBC API provides the following interfaces and classes:

- **DriverManager:** This class manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication sub protocol. The first driver that recognizes a certain sub protocol under JDBC will be used to establish a database Connection.
- **Driver:** This interface handles the communications with the database server. You will interact directly with Driver objects very rarely. Instead, you use DriverManager objects, which manage objects of this type. It also abstracts the details associated with working with Driver objects
- **Connection:** This interface with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only.
- **Statement:** You use objects created from this interface to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.
- **ResultSet:** These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.
- **SQLException:** This class handles any errors that occur in a database application.

F. DriverManager [Class]

The DriverManager class acts as an interface between user and drivers. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver. The DriverManager class maintains a list of Driver classes that have registered themselves by calling the method DriverManager.registerDriver().

Commonly used methods of DriverManager class:

`public static void registerDriver(Driver driver):`	is used to register the given driver with DriverManager.
`public static void deregisterDriver(Driver driver):`	is used to deregister the given driver (drop the driver from the list) with DriverManager.
`public static Connection getConnection(String url):`	is used to establish the connection with the specified url.
`public static Connection getConnection(String url,String userName,String password):`	is used to establish the connection with the specified url, username and password.

G. Interfaces

G.1. Driver

Driver is the interface that every driver class must implement. The Java SQL framework allows for multiple database drivers. Each driver should supply a class that implements the Driver interface. The DriverManager will try to load as many drivers as it can find and then for any given connection request, it will ask each driver in turn to try to connect to the target URL.

G.2. Connection

A Connection is the session between java application and database. The Connection interface is a factory of Statement, PreparedStatement, and DatabaseMetaData i.e. object of Connection can be used to get the object of Statement and DatabaseMetaData. The Connection interface provide many methods for transaction management like commit(),rollback() etc. By default, connection commits the changes after executing queries.

Commonly used methods of Connection interface:

`public Statement createStatement()`	creates a statement object that can be used to execute SQL queries.
`public Statement createStatement(int resultSetType,int resultSetConcurrency)`	Creates a Statement object that will generate ResultSet objects with the given type and concurrency.
`public void setAutoCommit(boolean status)`	is used to set the commit status.By default it is true.
`public void commit()`	saves the changes made since the previous commit/rollback permanent.
`public void rollback()`	Drops all changes made since the previous commit/rollback.
`public void close()`	closes the connection and Releases a JDBC resources immediately.

G.3. Statement

The **Statement interface** provides methods to execute queries with the database. The statement interface is a factory of ResultSet i.e. it provides factory method to get the object of ResultSet.

Commonly used methods of Statement interface:

`public ResultSet executeQuery(String sql)`	is used to execute SELECT query. It returns the object of ResultSet.
`public int executeUpdate(String sql)`	is used to execute specified query, it may be create, drop, insert, update, delete etc.
`public boolean execute(String sql)`	is used to execute queries that may return multiple results.
`public int[] executeBatch()`	is used to execute batch of commands.

G.4. PreparedStatement

The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized query. Let's see the example of parameterized query:

- String sql="insert into emp values(?,?,?)";

As you can see, we are passing parameter (?) for the values. Its value will be set by calling the setter methods of PreparedStatement. **Improves performance**: The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.

Commonly used methods of PreparedStatement interface:

`public void setInt(int paramIndex, int value)`	sets the integer value to the given parameter index.
`public void setString(int paramIndex, String value)`	sets the String value to the given parameter index.
`public void setFloat(int paramIndex, float value)`	sets the float value to the given parameter index.
`public void setDouble(int paramIndex, double value)`	sets the double value to the given parameter index.
`public int executeUpdate()`	executes the query. It is used for create, drop, insert, update, delete etc.
`public ResultSet executeQuery()`	executes the select query. It returns an instance of ResultSet.

G.5. CallableStatement

To call the stored procedures and functions, CallableStatement interface is used. We can have business logic on the database by the use of stored procedures and functions that will make the performance better because these are precompiled. Suppose you need the get the age of the employee based on the date of birth, you may create a function that receives date as the input and returns age of the employee as the output.

Stored Procedure	Function
is used to perform business logic.	is used to perform calculation.
must not have the return type.	must have the return type.
may return 0 or more values.	may return only one values.
We can call functions from the procedure.	Procedure cannot be called from function.
Procedure supports input and output parameters.	Function supports only input parameter.
Exception handling using try/catch block can be used in stored procedures.	Exception handling using try/catch can't be used in user defined functions.

G.6. ResultSet

The object of ResultSet maintains a cursor pointing to a particular row of data. Initially, cursor points to before the first row. By default, ResultSet object can be moved forward only and it is not updatable. But we can make this object to move forward and backward direction by passing either TYPE_SCROLL_INSENSITIVE or TYPE_SCROLL_SENSITIVE in createStatement(int,int) method as well as we can make this object as updatable by:

- **Statement stmt = con.createStatement**(ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);

Commonly used methods of ResultSet interface

`public boolean next()`	is used to move the cursor to the one row next from the current position.
`public boolean previous()`	is used to move the cursor to the one row previous from the current position.
`public boolean first()`	is used to move the cursor to the first row in result set object.
`public boolean last()`	is used to move the cursor to the last row in result set object.

`public boolean absolute(int row)`	is used to move the cursor to the specified row number in the ResultSet object.
`public boolean relative(int row)`	is used to move the cursor to the relative row number in the ResultSet object, it may be positive or negative.
`public int getInt(int columnIndex)`	is used to return the data of specified column index of the current row as int.
`public int getInt(String columnName)`	is used to return the data of specified column name of the current row as int.
`public String getString(int columnIndex)`	is used to return the data of specified column index of the current row as String.
`public String getString(String columnName)`	is used to return the data of specified column name of the current row as String.

G.7. ResultSetMetaData

The metadata means data about data i.e. we can get further information from the data. If you have to get metadata of a table like total number of column, column name, column type etc., ResultSetMetaData interface is useful because it provides methods to get metadata from the ResultSet object. Commonly used methods of ResultSetMetaData interface

Commonly used methods of ResultSetMetaData interface

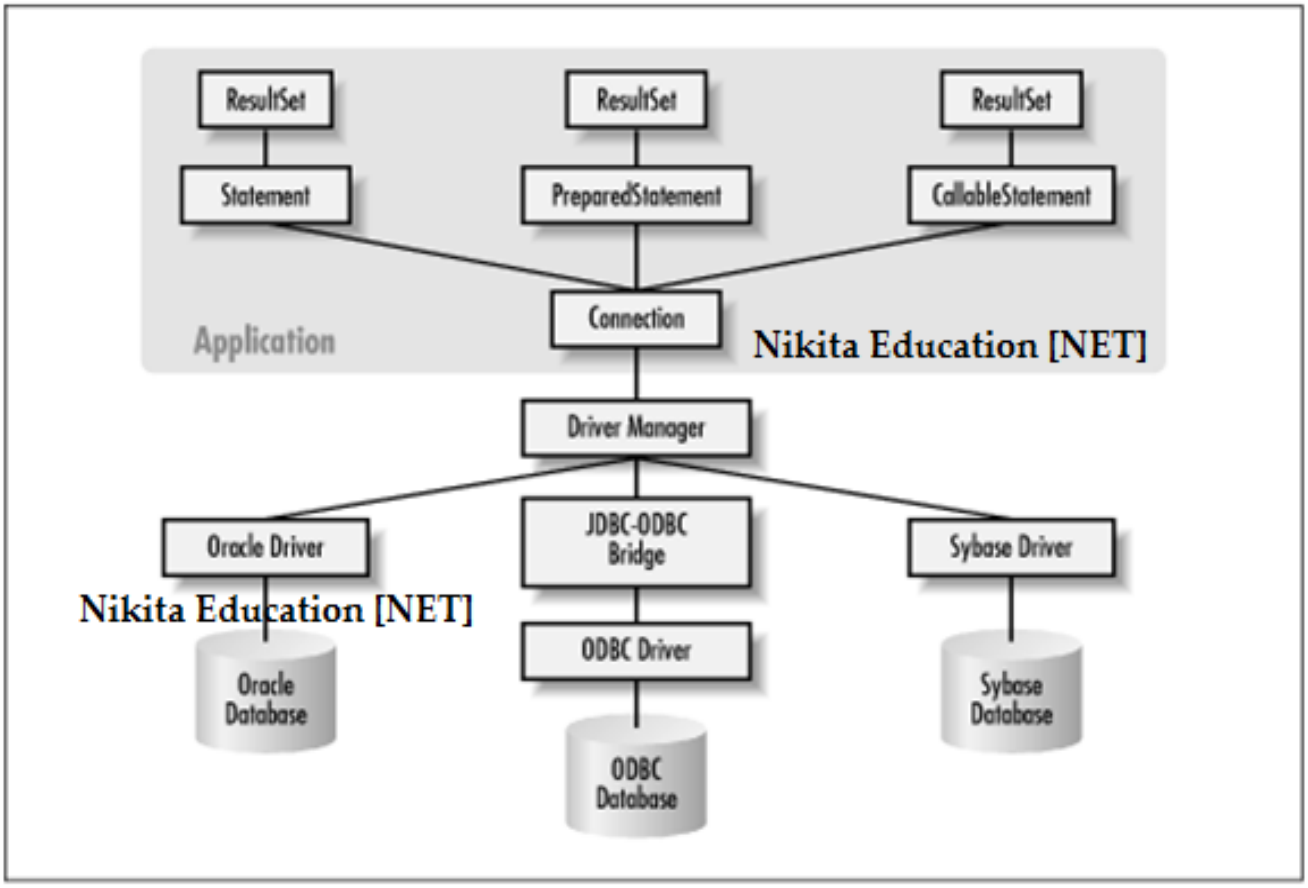
`public int getColumnCount()throws SQLException`	it returns the total number of columns in the ResultSet object.
`public String getColumnName(int index)throws SQLException`	it returns the column name of the specified column index.
`public String getColumnTypeName(int index)throws SQLException`	it returns the column type name for the specified index.
`public String getTableName(int index)throws SQLException`	it returns the table name for the specified column index.

G.8. DatabaseMetaData

DatabaseMetaData interface provides methods to get meta data of a database such as database product name, database product version, driver name, name of total number of tables, name of total number of views etc.

Commonly used methods of DatabaseMetaData interface

`public String getDriverName()throws SQLException`	it returns the name of the JDBC driver.
`public String getDriverVersion()throws SQLException`	it returns the version number of the JDBC driver.
`public String getUserName()throws SQLException`	it returns the username of the database.
`public String getDatabaseProductName()throws SQLException`	it returns the product name of the database.
`public String getDatabaseProductVersion()throws SQLException`	it returns the product version of the database.
`public ResultSet getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)throws SQLException`	it returns the description of the tables of the specified catalog. The table type can be TABLE, VIEW, ALIAS, SYSTEM TABLE, SYNONYM etc.



H. JDBC Drivers

A **JDBC driver** is a software component enabling a Java application to interact with a database. **JDBC drivers** are analogous to ODBC **drivers**, ADO.NET data providers, and OLE DB providers. To connect with individual databases, **JDBC** (the Java Database Connectivity API) requires **drivers** for each database. The JDBC driver gives out the connection to the database and implements the protocol for transferring the query and result between client and database. JDBC Driver is a software component that enables java application to interact with the database. There are 4 types of JDBC drivers:

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)
4. Thin driver (fully java driver)

1. JDBC-ODBC bridge driver:

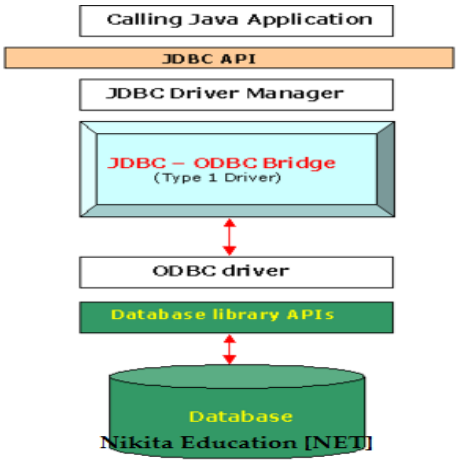
The Type 1 driver translates all JDBC calls into ODBC calls and sends them to the ODBC driver. ODBC is a generic API. The JDBC-ODBC Bridge driver is recommended only for experimental use or when no other alternative is available. In a Type 1 driver, a JDBC bridge is used to access ODBC drivers installed on each client machine. Using ODBC requires configuring on your system a Data Source Name (DSN) that represents the target database. When Java first came out, this was a useful driver because most databases only supported ODBC access but now this type of driver is recommended only for experimental use or when no other alternative is available.

Advantage

The JDBC-ODBC Bridge allows access to almost any database, since the database's ODBC drivers are already available.

Disadvantages

- Since the Bridge driver is not written fully in Java, Type 1 drivers are not portable.
- A performance issue is seen as a JDBC call goes through the bridge to the ODBC driver, then to the database, and this applies even in the reverse process. They are the slowest of all driver types.
- The client system requires the ODBC Installation to use the driver.
- Not good for the Web.



2. Native-API driver (partially java driver)

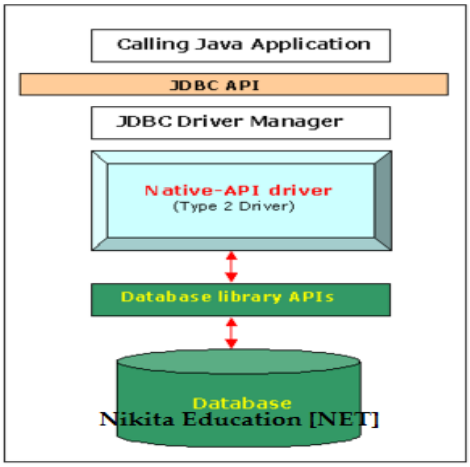
The distinctive characteristic of type 2 jdbc drivers is that Type 2 drivers convert JDBC calls into database-specific calls i.e. this driver is specific to a particular database. Some distinctive characteristic of type 2 jdbc drivers are shown below. Example: Oracle will have oracle native API. In a Type 2 driver, JDBC API calls are converted into native C/C++ API calls which are unique to the database. These drivers typically provided by the database vendors and used in the same manner as the JDBC-ODBC Bridge, the vendor-specific driver must be installed on each client machine. If we change the Database we have to change the native API as it is specific to a database and they are mostly obsolete now but you may realize some speed increase with a Type 2 driver, because it eliminates ODBC's overhead.

Advantage

The distinctive characteristic of type 2 jdbc drivers are that they are typically offer better performance than the JDBC-ODBC Bridge as the layers of communication (tiers) are less than that of Type 1 and also it uses Native api which is Database specific.

Disadvantage

- Native API must be installed in the Client System and hence type 2 drivers cannot be used for the Internet.
- Like Type 1 drivers, it's not written in Java Language which forms a portability issue.
- If we change the Database we have to change the native api as it is specific to a database
- Mostly obsolete now
- Usually not thread safe.



3. Network Protocol driver (fully java driver)

Type 3 database requests are passed through the network to the middle-tier server. The middle-tier then translates the request to the database. If the middle-tier server can in turn use Type1, Type 2 or Type 4 drivers. In a Type 3 driver, a three-tier approach is used to accessing databases. The JDBC clients use standard network sockets to communicate with an middleware application server. The socket information is then translated by the middleware application server into the call format required by the DBMS, and forwarded to the database server. This kind of driver is extremely flexible, since it requires no code installed on the client and a single driver can actually provide access to multiple databases.

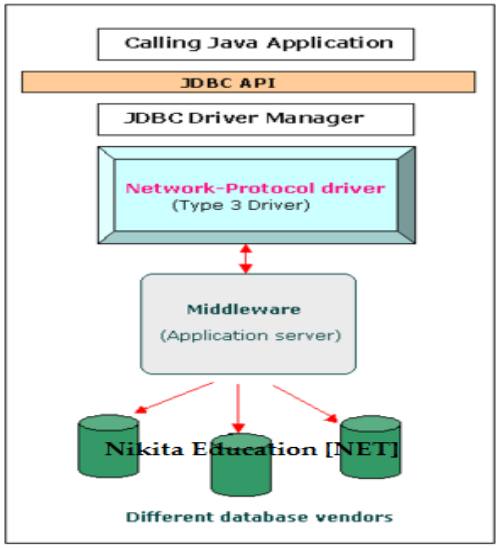
Advantage

- This driver is server-based, so there is no need for any vendor database library to be present on client machines.

- This driver is fully written in Java and hence Portable. It is suitable for the web.
- There are many opportunities to optimize portability, performance, and scalability.
- The net protocol can be designed to make the client JDBC driver very small and fast to load.
- The type 3 driver typically provides support for features such as caching (connections, query results, and so on), load balancing, and advanced system administration such as logging and auditing.
- This driver is very flexible allows access to multiple databases using one driver.
- They are the most efficient amongst all driver types.

Disadvantage

It requires another server application to install and maintain. Traversing the record set may take longer, since the data comes through the backend server.



4. Thin driver (fully java driver)

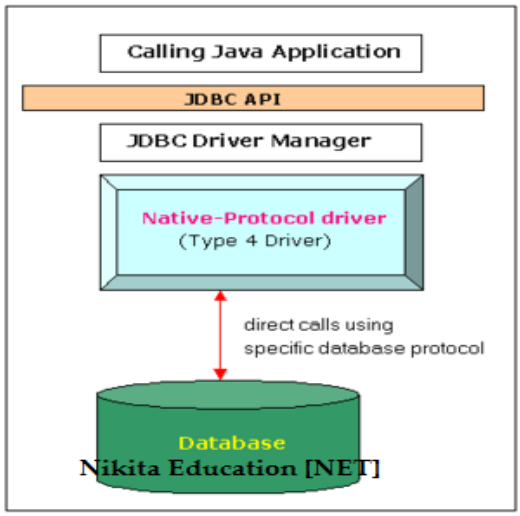
The Type 4 uses java networking libraries to communicate directly with the database server. In a Type 4 driver, a pure Java-based driver that communicates directly with vendor's database through socket connection. This is the highest performance driver available for the database and is usually provided by the vendor itself. This kind of driver is extremely flexible; you don't need to install special software on the client or server. Further, these drivers can be downloaded dynamically.

Advantage

- The major benefit of using type 4 jdbc drivers are that they are completely written in Java to achieve platform independence and eliminate deployment administration issues. It is most suitable for the web.
- Number of translation layers is very less i.e. type 4 JDBC drivers don't have to translate database requests to ODBC or a native connectivity interface or to pass the request on to another server, performance is typically quite good.
- You don't need to install special software on the client or server. Further, these drivers can be downloaded dynamically.

Disadvantage

With type 4 drivers, the user needs a different driver for each database.



I. Steps to connect to the database in java

1. Register the driver class

The forName() method of Class class is used to register the driver class. This method is used to dynamically load the driver class.

```
Syntax:
public static void forName(String className)throws ClassNotFoundException

Example:
Class.forName("oracle.jdbc.driver.OracleDriver");
```

2. Create the connection object

The getConnection() method of DriverManager class is used to establish connection with the database.

```
Syntax:
public static Connection getConnection(String url)throws SQLException
public static Connection getConnection(String url,String name,String password)

Example:
Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe",
"system","password");
```

3. Create the Statement object

The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.

```
Syntax:
public Statement createStatement()throws SQLException

Example:
Statement stmt=con.createStatement();
```

4. Execute the query

The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

```
Syntax:
public ResultSet executeQuery(String sql)throws SQLException
Example:
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next()){
      System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
```

5. Close the connection object

By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

```
Syntax:    public void close()throws SQLException

Example:   con.close();
```

J. Simple JDBC Example

First crate table 'LOGIN' in MS Access database with two fields name (varchar) and password (varchar) respectively and insert some records into table. Create dsn in ODBC tools named as 'mydsn':

```
import java.sql.*;

class Test
{
     public static void main(String ar[])
     {
          try{
               String url="jdbc:odbc:mydsn";
               Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
               Connection c=DriverManager.getConnection(url);
               Statement st=c.createStatement();
               ResultSet rs=st.executeQuery("select * from login");
               while(rs.next()){
                    System.out.println(rs.getString(1));
                    System.out.println(rs.getString(2));
               }
          }catch(Exception ee){
               System.out.println(ee);
          }
     }
}
```

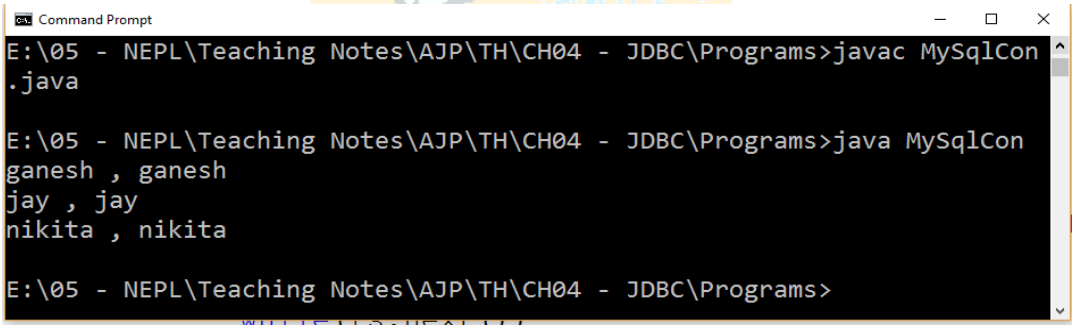
K. JDBC Examples

1. Simple jdbc-odbc program

```
import java.sql.*;
class SimpleJdbcDemoTwo
{
     public static void main(String ar[])
     {
          try
          {
               String database="MyDemoDB.accdb";
               String url="jdbc:odbc:Driver={Microsoft Access Driver
                              (*.accdb)};DBQ=" + database +
                              ";DriverID=22;READONLY=true";
               Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
               Connection c=DriverManager.getConnection(url);
               Statement st=c.createStatement();
               ResultSet rs=st.executeQuery("select * from login");
               while(rs.next()){
                    System.out.println(rs.getString(1));
                    System.out.println(rs.getString(2));
               }
          }
          catch(Exception ee)
          {
               System.out.println(ee);
          }
     }
}
```

2. MySql Connection

```
import java.sql.*;

class MySqlCon
{
     public static void main(String args[])
     {
          try
          {
               Class.forName("com.mysql.jdbc.Driver");
               Connection con=
                    DriverManager.getConnection("jdbc:mysql:
                         //localhost:3306
                              /mydemodb","root","root");
               Statement stmt=con.createStatement();
               ResultSet rs=stmt.executeQuery("select * from login");
               while(rs.next())
               {
                    System.out.print(rs.getString(1));
                    System.out.println(" , "+rs.getString(2));
               }
               con.close();
          }
          catch(Exception e)
          {
               System.out.println(e);
          }
     }
}
```



3. SQL Server Connection

```
import java.sql.Connection;
import java.sql.DatabaseMetaData;
import java.sql.DriverManager;
import java.sql.SQLException;

public class JdbcSQLServerConnection
{
   public static void main(String[] args)throws SQLException,
                                                 ClassNotFoundException
   {
      Connection conn = null;
      try
      {
```

```
            Class.forName("com.microsoft.sqlserver.
                                jdbc.SQLServerDriver");
            String dbURL = "jdbc:sqlserver://localhost\\MSSQLSERVER";
            String user = "sa";
            String pass = "nspl08";
            conn = DriverManager.getConnection(dbURL, user, pass);
            if (conn != null)
            {
                  DatabaseMetaData dm =
                              (DatabaseMetaData) conn.getMetaData();
                  System.out.println("Driver name: " +
                                    dm.getDriverName());
                  System.out.println("Driver version: " +
                                    dm.getDriverVersion());
                  System.out.println("Product name: " +
                                    dm.getDatabaseProductName());
                  System.out.println("Product version: " +
                                    dm.getDatabaseProductVersion());
            }
      }
      catch (SQLException ex)
      {
            ex.printStackTrace();
      }
      finally
      {
            try
            {
                  if (conn != null && !conn.isClosed())
                  {
                        conn.close();
                  }
            }
            catch (SQLException ex)
            {
                  ex.printStackTrace();
            }
      }
   }
}
```

```
Command Prompt
E:\05 - NEPL\Teaching Notes\AJP\TH\CH04 - JDBC\Programs>
javac JdbcSQLServerConnection.java

E:\05 - NEPL\Teaching Notes\AJP\TH\CH04 - JDBC\Programs>
java JdbcSQLServerConnection
Driver name: Microsoft JDBC Driver 4.1 for SQL Server
Driver version: 4.1.5605.100
Product name: Microsoft SQL Server
Product version: 10.50.1600

E:\05 - NEPL\Teaching Notes\AJP\TH\CH04 - JDBC\Programs>
```

4. Oracle Connection

```
import java.sql.*;
class OracleCon
{
   public static void main(String args[])
   {
      try
      {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection con=DriverManager.getConnection(
            "jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
            Statement stmt=con.createStatement();
            ResultSet rs=stmt.executeQuery("select * from emp");
            while(rs.next())
            {
                  System.out.println( rs.getInt(1)+"  "+
                                      rs.getString(2)+"  "+
                                        rs.getString(3));
            }
            con.close();
      }
      catch(Exception e)
      {
            System.out.println(e);
      }
   }
}
```

5. MySQL Connection

```
import java.sql.*;

class MySqlConTwo
{
   public static void main(String args[])
   {
      try
      {
            Class.forName("com.mysql.jdbc.Driver");
            Connection con=
                  DriverManager.getConnection("jdbc:mysql://
                        localhost:3306/MyDemoDB","root","root123");
            Statement stmt=con.createStatement();
            ResultSet rs=stmt.executeQuery("select * from
                                    StudBasic order by RollNo");
            System.out.println("\nRollNo \t Name \t Branch
                                              \t Mobile \t Avg \n");
            while(rs.next())
```

```
            {
                  System.out.println( rs.getInt(1)+" \t "+
                                          rs.getString(2)+" \t "+
                                          rs.getString(3)+" \t "+
                                          rs.getLong(4)+" \t "+
                                          rs.getFloat(5)+"\n");
            }
            con.close();
      }
      catch(Exception e)
      {
            System.out.println(e);
      }
   }
}
```

6. PreparedStatement Example

```
import java.sql.*;

public class MySqlPrepStmt
{
   static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
   static final String DB_URL = "jdbc:mysql://localhost/MyDemoDB";

   static final String USER = "root";
   static final String PASS = "root123";

   public static void main(String[] args)
   {
      Connection conn = null;
      PreparedStatement stmt = null;
      try
      {
            Class.forName(JDBC_DRIVER);

            System.out.println("Connecting to database...");
            conn = DriverManager.getConnection(DB_URL,USER,PASS);

            System.out.println("Creating statement...");
            String sql = "UPDATE StudBasic set StudBranch=?
                        WHERE RollNo=?";
            stmt = conn.prepareStatement(sql);

            stmt.setString(1, "Computer");
            stmt.setInt(2, 126130);

            int rows = stmt.executeUpdate();
            System.out.println("Rows impacted : " + rows );
```

```
        sql = "SELECT * FROM StudBasic";
        ResultSet rs = stmt.executeQuery(sql);

        while(rs.next())
        {
             int roll  = rs.getInt("RollNo");
             String sname = rs.getString("StudName");
             String  branch= rs.getString("StudBranch");
             long mobile=rs.getLong(4);
             float avg=rs.getFloat(5);

             //Display values
             System.out.print("RollNo: " + roll);
             System.out.print(", Name: " + sname);
             System.out.print(", Branch: " + branch);
             System.out.print(", Mobile: " + mobile);
             System.out.println(", Avg: " + avg);
        }
        //STEP 6: Clean-up environment
        rs.close();
        stmt.close();
        conn.close();
    }
    catch(SQLException se)
    {
        //Handle errors for JDBC
        se.printStackTrace();
    }
    catch(Exception e)
    {
        //Handle errors for Class.forName
        e.printStackTrace();
    }
    finally
    {
        //finally block used to close resources
        try
        {
             if(stmt!=null)
                  stmt.close();
        }
        catch(SQLException se2)
        {
        }
        try
        {
             if(conn!=null)
                  conn.close();
```

```
            }
            catch(SQLException se)
            {
                  se.printStackTrace();
            }
      }
      System.out.println("Goodbye!");
   }
}
```

Example 2:

```
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.text.DateFormat;
import java.text.SimpleDateFormat;

public class JDBCPreparedStatementInsertExample
{
   private static final String DB_DRIVER = "com.mysql.jdbc.Driver";
   private static final String DB_CONNECTION =
                                    "jdbc:mysql://localhost/MyDemoDB";
   private static final String DB_USER = "root";
   private static final String DB_PASSWORD = "root123";

   public static void main(String[] argv)
   {
      try
      {
            insertRecordIntoTable();
      }
      catch (SQLException e)
      {
            System.out.println(e.getMessage());
      }
   }
   private static void insertRecordIntoTable() throws SQLException
   {
      Connection dbConnection = null;
      PreparedStatement preparedStatement = null;
      String sql=null;

      String insertTableSQL = "INSERT INTO StudBasic" +
                              "(RollNo, StudName, StudBranch," +
                               "StudMobile, StudAvg)VALUES" +
                                "(?,?,?,?,?)";
```

```
try
{
     dbConnection = getDBConnection();
     preparedStatement =
               dbConnection.prepareStatement(insertTableSQL);

     preparedStatement.setInt(1, 126162);
     preparedStatement.setString(2, "Sapana");
     preparedStatement.setString(3, "Mechanical");
     preparedStatement.setLong(4, 94234745);
     preparedStatement.setFloat(5,92.57f);

     // execute insert SQL stetement
     int rows = preparedStatement.executeUpdate();

     System.out.println(rows + " Record inserted........");

     System.out.println("\n\nPrinting All Table Records");

     sql = "SELECT * FROM StudBasic";
     ResultSet rs = preparedStatement.executeQuery(sql);

     while(rs.next())
     {
          int roll  = rs.getInt("RollNo");
          String sname = rs.getString("StudName");
          String  branch= rs.getString("StudBranch");
          long mobile=rs.getLong(4);
          float avg=rs.getFloat(5);

          //Display values
          System.out.print("RollNo: " + roll);
          System.out.print(", Name: " + sname);
          System.out.print(", Branch: " + branch);
          System.out.print(", Mobile: " + mobile);
          System.out.println(", Avg: " + avg);
     }
     rs.close();

} catch (SQLException e) {

     System.out.println(e.getMessage());

} finally {

     if (preparedStatement != null) {
          preparedStatement.close();
     }
```

```
            if (dbConnection != null) {
                 dbConnection.close();
            }

      }

   }

   private static Connection getDBConnection() {

      Connection dbConnection = null;

      try {

            Class.forName(DB_DRIVER);

      } catch (ClassNotFoundException e) {

            System.out.println(e.getMessage());

      }

      try {

            dbConnection = DriverManager.getConnection(
                              DB_CONNECTION, DB_USER,DB_PASSWORD);
            return dbConnection;

      } catch (SQLException e) {

            System.out.println(e.getMessage());

      }

      return dbConnection;

   }
}
```

7. ResultSet Metadata

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Statement;
```

```
public class MyResultSetMetadata
{
   public static void main(String a[])
   {
      Connection con = null;
         Statement st = null;
         ResultSet rs = null;
         try
      {
              Class.forName("com.mysql.jdbc.Driver");
              con = DriverManager.getConnection("jdbc:mysql://
                           localhost:3306/MyDemoDB","root","root123");
              st = con.createStatement();
              rs = st.executeQuery("select * from StudBasic");
              ResultSetMetaData rsmd = rs.getMetaData();
              int columnCount = rsmd.getColumnCount();
              for(int i=1;i<=columnCount;i++){
                  System.out.println(rsmd.getColumnName(i));
                  System.out.println(rsmd.getColumnType(i));
              }
         } catch (ClassNotFoundException e) {
              // TODO Auto-generated catch block
              e.printStackTrace();
         } catch (SQLException e) {
              // TODO Auto-generated catch block
              e.printStackTrace();
         } finally{
              try{
                  if(rs != null) rs.close();
                  if(st != null) st.close();
                  if(con != null) con.close();
              } catch(Exception ex){}
         }
     }
}
```

8. Database Metadata

```
import java.sql.Connection;
import java.sql.DatabaseMetaData;
import java.sql.DriverManager;
import java.sql.SQLException;

public class MyDatabaseMetadata
{
   public static void main(String a[])
   {
         Connection con = null;
         try {
```

```
            Class.forName("com.mysql.jdbc.Driver");
            con = DriverManager.getConnection("jdbc:mysql://
                        localhost:3306/MyDemoDB","root","root123");
            DatabaseMetaData dm = con.getMetaData();
            System.out.println(dm.getDriverVersion());
            System.out.println(dm.getDriverName());
            System.out.println(dm.getDatabaseProductName());
            System.out.println(dm.getDatabaseProductVersion());
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (ClassNotFoundException cnfe){
          cnfe.printStackTrace();
     }finally{
            if(con != null){
                try {
                    con.close();
                } catch (SQLException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        }
    }
}
```

Unit-V
Servlet & JSP

Unit	Topic Name	Topics	Marks
5	**Servlet**	Basic Concepts of Web & Web Servers, Introduction to Servlet, Servlet Life Cycle, Types of Servlet, Servlet API, Sessions, Cookies, Session Management.	**24**
	JSP	Introduction to JSP, JSP life cycle, JSP tags: Declaration, Expression, Scriptlet, JSP Directives, Implicit Objects, Action Elements, EL, JSTL, Custom Tags.	
	RMI & EJB	Introduction to RMI, Introduction to EJB.	

A. Basic Concepts

- **World Wide Web:** The Web is a network of computers all over the world. All the computers in the Web can communicate with each other. All the computers use a communication protocol called HTTP. Web information is stored in documents called web pages. Web pages are files stored on computers called web servers. Computers reading the web pages are called web clients. Web clients view the pages with a program called a web browser. Popular browsers are Google Chrome, Firefox, and Internet Explorer.

- **Browser:** A browser fetches a page from a web server by a request. A request is a standard HTTP request containing a page address. An address may look like this: http://www.example.com/a.html. All web pages contain instructions for display. The browser displays the page by reading these instructions. The most common display instructions are called HTML tags HTML tags look like this This is a paragraph.

- **Website:** Website is a collection of related web pages that may contain text, images, audio and video. The first page of a website is called home page. Each website has specific internet address (URL) that you need to enter in your browser to access a website. Website is hosted on one or more servers and can be accessed by visiting its homepage using a computer network.

- **Static Web Page:** "Static" means unchanged or constant, static web pages contain the same prebuilt content each time the page is loaded. Standard HTML pages are static web pages. They contain HTML code, which defines the structure and content of the Web page. Each time an HTML page is loaded, it looks the same. The only way the content of an HTML page will change is if the Web developer updates and publishes the file.

- **Dynamic Web Page:** "dynamic" means changing or lively, the content of dynamic Web pages can be generated on-the-fly. PHP, ASP, Servlets and JSP pages are dynamic Web pages. These pages contain "server-side" code, which allows the server to generate unique content each time the page is loaded. Technologies used to create dynamic web pages are PHP, ASP, JSP, Servlets, CGI etc.

- **Web Application:** A web application is an application accessible from the web used to generate dynamic responses. A web application is composed of web components like Servlet, JSP, Filter etc. and other components such as HTML. The web components typically execute in Web Server and respond to HTTP request.

B. HTTP

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. This is the foundation for data communication for the World Wide Web (i.e. internet) since 1990. HTTP is a generic and stateless protocol which can be used for other purposes as well using extensions of its request methods, error codes, and headers. Basically, HTTP is a TCP/IP based communication protocol, that is used to deliver data (HTML files, image files, query results, etc.) on the World Wide Web. The default port is TCP 80, but other ports can be used as well. It provides a

standardized way for computers to communicate with each other. HTTP specification specifies how clients' request data will be constructed and sent to the server, and how the servers respond to these requests.

HTTP Features:

There are three basic features that make HTTP a simple but powerful protocol:

- **HTTP is connectionless**: The HTTP client, i.e., a browser initiates an HTTP request and after a request is made, the client disconnects from the server and waits for a response. The server processes the request and re-establishes the connection with the client to send a response back.
- **HTTP is media independent**: It means, any type of data can be sent by HTTP as long as both the client and the server know how to handle the data content. It is required for the client as well as the server to specify the content type using appropriate MIME-type.
- **HTTP is stateless**: As mentioned above, HTTP is connectionless and it is a direct result of HTTP being a stateless protocol. The server and client are aware of each other only during a current request. Afterwards, both of them forget about each other. Due to this nature of the protocol, neither the client nor the browser can retain information between different requests across the web pages.

HTTP Request

An HTTP client sends an HTTP request to a server in the form of a request message which includes following format:

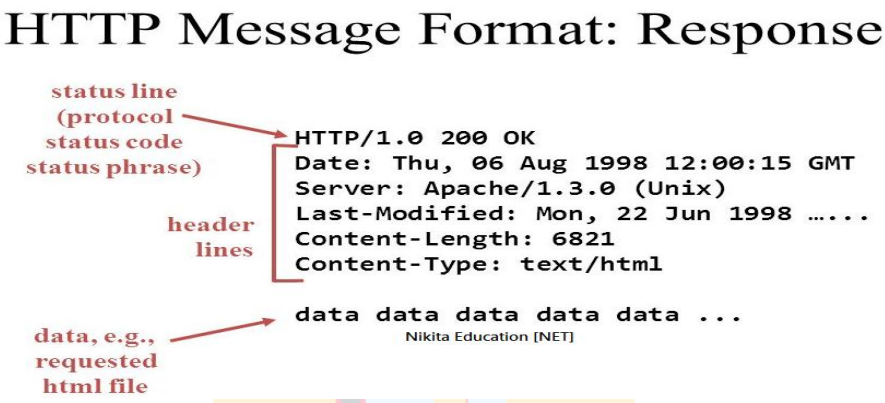
Request Methods:
GET: The GET method is used to retrieve information from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data.
HEAD: Same as GET, but it transfers the status line and the header section only.
POST: A POST request is used to send data to the server, for example, customer information, file upload, etc. using HTML forms.
PUT: Replaces all the current representations of the target resource with the uploaded content.
DELETE: Removes all the current representations of the target resource given by URI.
CONNECT: Establishes a tunnel to the server identified by a given URI.
OPTIONS: Describe the communication options for the target resource.
TRACE: Performs a message loop back test along with the path to the target resource.

GET	POST
In case of Get request, only limited amount of data can be sent because data is sent in header.	In case of post request, large amount of data can be sent because data is sent in body.
Get request is not secured because data is exposed in URL bar.	Post request is secured because data is not exposed in URL bar.
Get request can be bookmarked	Post request cannot be bookmarked

Get request is idempotent. It means second request will be ignored until response of first request is delivered.	Post request is non-idempotent
Get request is more efficient and used more than Post	Post request is less efficient and used less than get.

HTTP Response

After receiving and interpreting a request message, a server responds with an HTTP response message which includes following format:



Status codes used in response: The Status-Code element is a 3-digit integer where first digit of the Status-Code defines the class of response and the last two digits do not have any categorization role. There are 5 values for the first digit.
1xx: Informational It means the request was received and the process is continuing.
2xx: Success It means the action was successfully received, understood, and accepted.
3xx: Redirection It means further action must be taken in order to complete the request.
4xx: Client Error It means the request contains incorrect syntax or cannot be fulfilled.
5xx: Server Error It means the server failed to fulfill an apparently valid request.

Content Type: Content Type is also known as MIME (Multipurpose internet Mail Extension) Type. It is a **HTTP header** that provides the description about what are you sending to the browser. There are many content types:	
• **text/html** • **text/plain** • **application/msword** • **application/vnd.ms-excel** • **application/jar**	• **application/pdf** • **application/octet-stream** • **application/x-zip** • **images/jpeg** • **video/quicktime etc.**

C. Server Software's

Web Server:
A Web server is a program that, using the client/server model and the World Wide Web's Hypertext Transfer Protocol (HTTP), serves the files that form Web pages to Web users. Two leading Web servers are Apache , the most widely-installed Web server, and Microsoft's Internet Information Server (IIS).
Any computer can be turned into a Web server by installing server software and connecting the machine to the Internet. A Web server can be either a computer program or a computer running a program that is responsible for accepting HTTP requests from clients, serving back HTTP responses along with optional data contents, which usually are web pages such as HTML documents and linked objects on it.

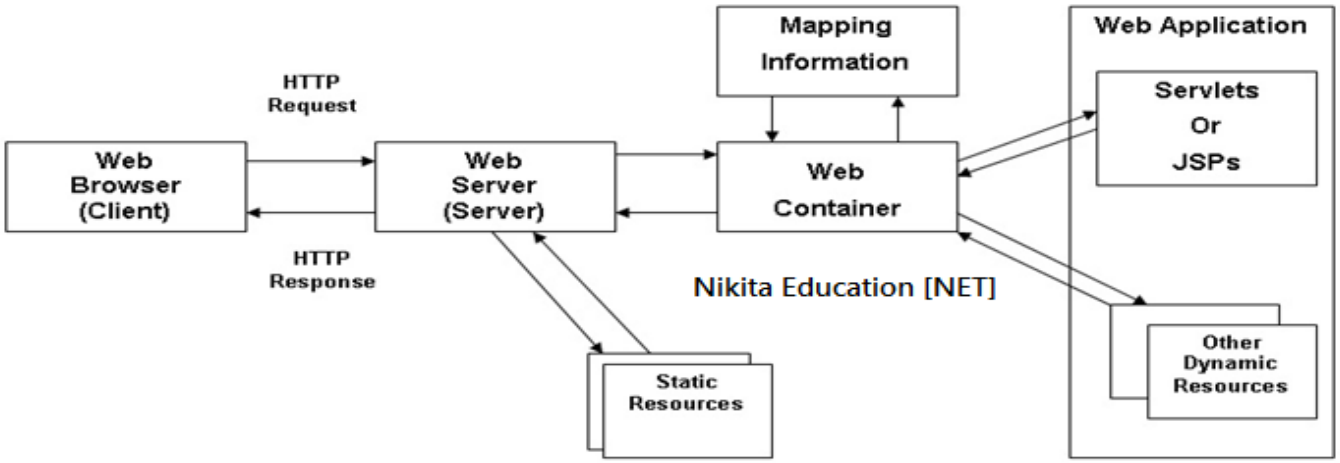
Application Server:

An application server is a program that handles all application operations between users and an organization's backend business applications or databases. An application server is typically used for complex transaction-based applications. To support high-end needs, an application server has to have built-in redundancy, monitor for high-availability, high-performance distributed application services and support for complex database access. An application server is the kind of software engine that will deliver various applications to another device.

	Application Server	**Web Server**
What is it?	A server that exposes business logic to client applications through various protocols including HTTP.	A server that handles HTTP protocol.
Job	Application server is used to serve web based applications and enterprise based applications (i.e servlets, jsps and ejbs...). Application servers may contain a web server internally.	Web server is used to serve web based applications.(i.e servlets and jsps)
Functions	To deliver various applications to another device, it allows everyone in the network to run software off of the same machine.	Keeping HTML, PHP, ASP etc files available for the web browsers to view when a user accesses the site on the web, handles HTTP requests from clients.
Supports	distributed transaction and EJB's	Servlets and JSP
Resource utilization	High	Low
Examples	**JBoss** Open-source server. **Glassfish** provided by Oracle. **Weblogic** provided by Oracle. more secured. **Websphere** provided by IBM.	**Apache, Apache Tomcat**, **Resin**.

D. Working of Web Server

It's a server used to communicate with Web Browsers as its clients and the communication protocol used in this case is HTTP (Hyper Text Transfer Protocol). This is why a Web Server is also called an HTTP Server. The client (i.e., the Web Browser) and the server (i.e., HTTP/Web Server) should be able to communicate with each other in a defined way. These pre-defined sets of rules which form the basis of the communication are normally termed as a protocol and in this case the underlying protocol will be HTTP. Irrespective of how the client or the server has been implemented, there will always be a way to form a valid HTTP Request for the client to work and similarly the server needs to be capable of understanding the HTTP Requests sent to it and form valid HTTP Responses to all the arrived HTTP Requests. Both the client and the server machines should also be equipped with the capability of establishing the connection to each other (in this case it'll be a TCP reliable connection) to be able to transfer the HTTP Requests (client -> server) and HTTP Responses (server -> client).

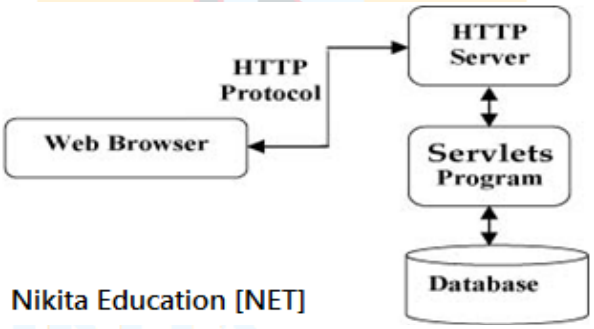


A **Web Container** is a J2EE compliant implementation which provides an environment for the Servlets and JSPs to run. Putting it differently we can say that a Web Container is combination of a Servlet Engine and a JSP Engine. If an HTTP Request refers to a Web Component (typically a Servlet or a JSP) then the request is forwarded to the Web Container and the result of the request is sent back to Web Server, which uses that result to prepare the HTTP Response for the particular HTTP Request.

E. Servlet

Servlet technology is used to create web application (resides at server side and generates dynamic web page). Servet technology is robust and scalable as it uses the java language. Before Servlet, CGI (Common Gateway Interface) scripting language was used as a server-side programming language. But there were many disadvantages of this technology. Java Servlets are programs that run on a Web or Application server and act as a middle layer between a requests coming from a Web browser or other HTTP client and databases or applications on the HTTP server. Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically. Java Servlets often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But Servlets offer several advantages in comparison with the CGI. There are many interfaces and classes in the servlet API such as Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse etc.

- Servlet is a technology i.e. used to create web application.
- Servlet is an API that provides many interfaces and classes including documentations.
- Servlet is an interface that must be implemented for creating any servlet.
- Servlet is a class that extends the capabilities of the servers and responds to the incoming request. It can respond to any type of requests.
- Servlet is a web component that is deployed on the server to create dynamic web page.

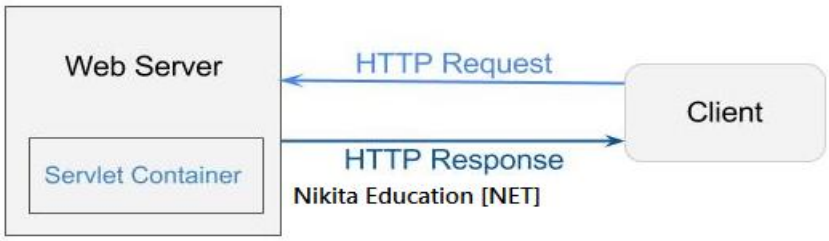


Servlets Tasks:

- Read the explicit data sent by the clients (browsers). This includes an HTML form on a Web page or it could also come from an applet or a custom HTTP client program.
- Read the implicit HTTP request data sent by the clients (browsers). This includes cookies, media types and compression schemes the browser understands, and so forth.
- Process the data and generate the results. This process may require talking to a database, executing an RMI or CORBA call, invoking a Web service, or computing the response directly.
- Send the explicit data (i.e., the document) to the clients (browsers). This document can be sent in a variety of formats, including text (HTML or XML), binary (GIF images), Excel, etc.
- Send the implicit HTTP response to the clients (browsers). This includes telling the browsers or other clients what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such tasks.

Servlet Container (Web Container)

It provides the runtime environment for JavaEE (j2ee) applications. The client/user can request only a static WebPages from the server. If the user wants to read the web pages as per input then the servlet container is used in java. The servlet container is used in java for dynamically generate the web pages on the server side. Therefore the servlet container is the part of a web server that interacts with the servlet for handling the dynamic web pages from the client.

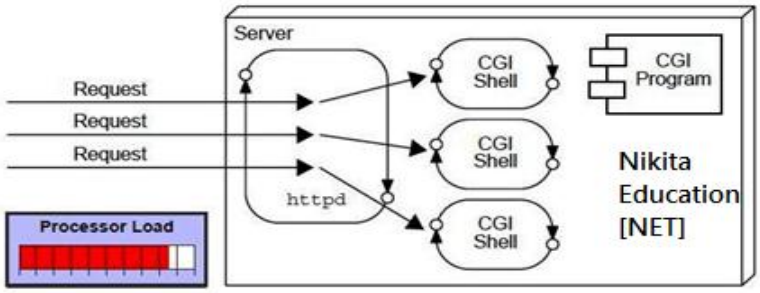


The Servlet Container performs:

- Life Cycle Management
- Multithreaded support
- Object Pooling
- Security etc.

F. CGI(Common Gateway Interface)

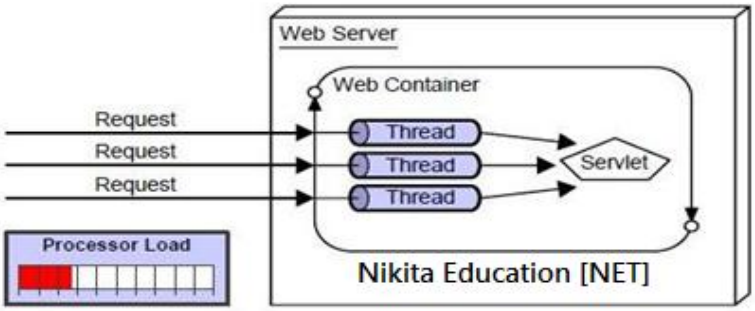
CGI technology enables the web server to call an external program and pass HTTP request information to the external program to process the request. For each request, it starts a new process. It is a older technology used to generate dynamic responses for client requests.



Disadvantages of CGI

1. If number of client's increases, it takes more time for sending response.
2. For each request, it starts a process and Web server is limited to start processes.
3. It uses platform dependent language e.g. C, C++, perl.

Advantage of Servlet:



1. **Better performance:** because it creates a thread for each request not process.
2. **Portability:** Servlets are platform-independent because they are written in Java.
3. **Robust:** Servlets are managed by JVM so no need to worry about memory leak, garbage collection etc.
4. **Secure:** because it uses java language...
5. Servlets execute within the address space of a Web server. It is not necessary to create a separate process to handle each client request.
6. Java security manager on the server enforces a set of restrictions to protect the resources on a server machine. So servlets are trusted.
7. The full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases, or other software via the sockets and RMI mechanisms that you have seen already.

G. Servlets - Environment Setup

A development environment is where you would develop your Servlet, test them and finally run them. Like any other Java program, you need to compile a servlet by using the Java compiler javac and after compilation the servlet application, it would be deployed in a configured environment to test and run. This development environment setup involves following steps:

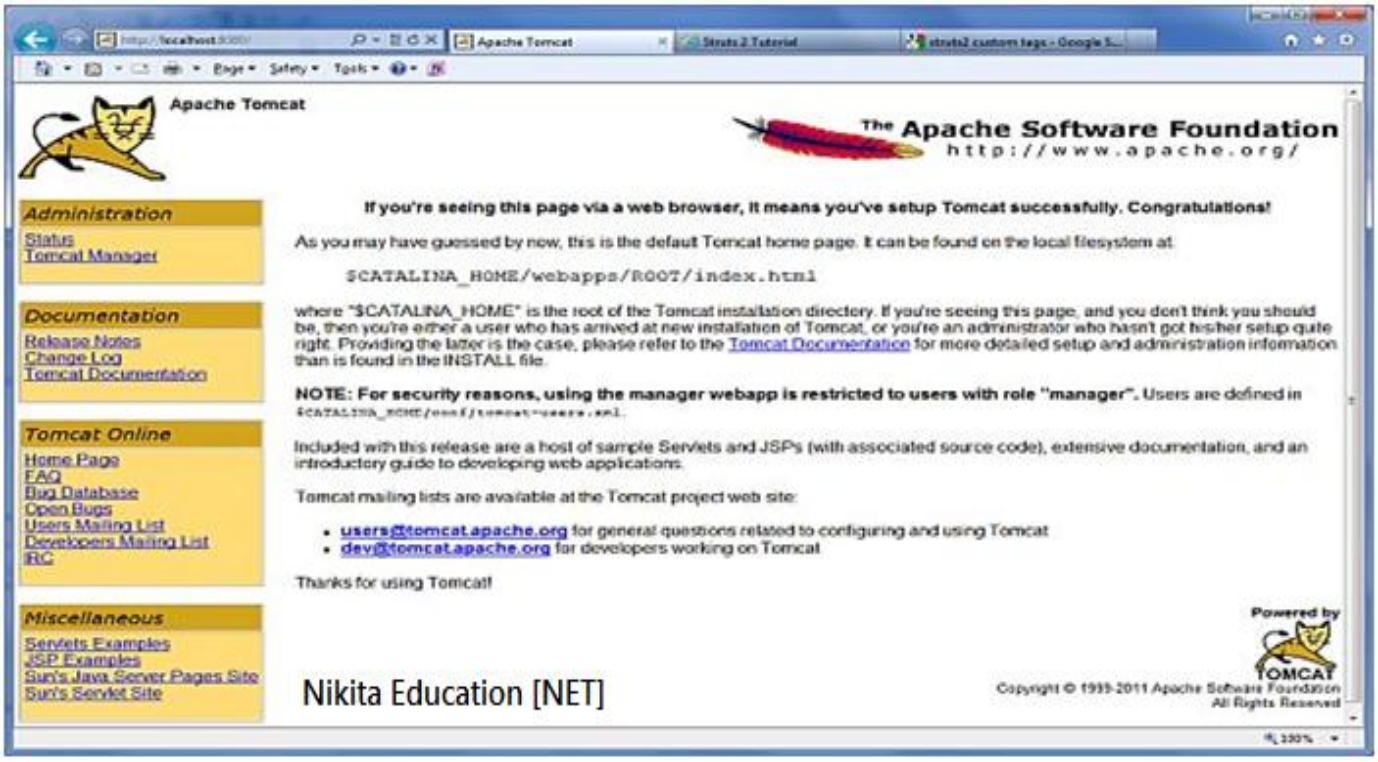
Setting up Web Server: Tomcat

A number of Web Servers that support servlets are available in the market. Some web servers are freely downloadable and Tomcat is one of them. Tomcat is an open source software implementation of the Java Servlet and Java Server Pages technologies and can act as a standalone server for testing servlets and can be integrated with the Apache Web Server. Here are the steps to setup Tomcat on your machine:

- Download latest version of Tomcat from http://tomcat.apache.org/.
- Once you downloaded the installation, unpack the binary distribution into a convenient location. For example in C:\apache-tomcat-5.5.29 on windows and create CATALINA_HOME environment variable pointing to these locations.
- Tomcat can be started by executing the following commands on windows machine:

%CATALINA_HOME%\bin\startup.bat or C:\apache-tomcat-5.5.29\bin\startup.bat

After startup, the default web applications included with Tomcat will be available by visiting **http://localhost:8080/.** If everything is fine then it should display following result:



Further information about configuring and running Tomcat can be found on the Tomcat web site: http://tomcat.apache.org. Tomcat can be stopped by executing the following commands on windows machine: C:\apache-tomcat-5.5.29\bin\shutdown

Setting up CLASSPATH

Since servlets are not part of the Java Platform, Standard Edition, you must identify the servlet classes to the compiler. If you are running Windows, you need to set Environment Variables, right-click on My Computer, select Properties, then Advanced, then Environment Variables.

CLASSPATH: C:\apache-tomcat-5.5.29\common\lib\servlet-api.jar;%CLASSPATH%

Or

Paste the servlet-api.jar file in JRE/lib/ext folder

H. Servlet Life Cycle

A servlet life cycle can be defined as the entire process from its creation till the destruction. The following are the paths followed by a servlet

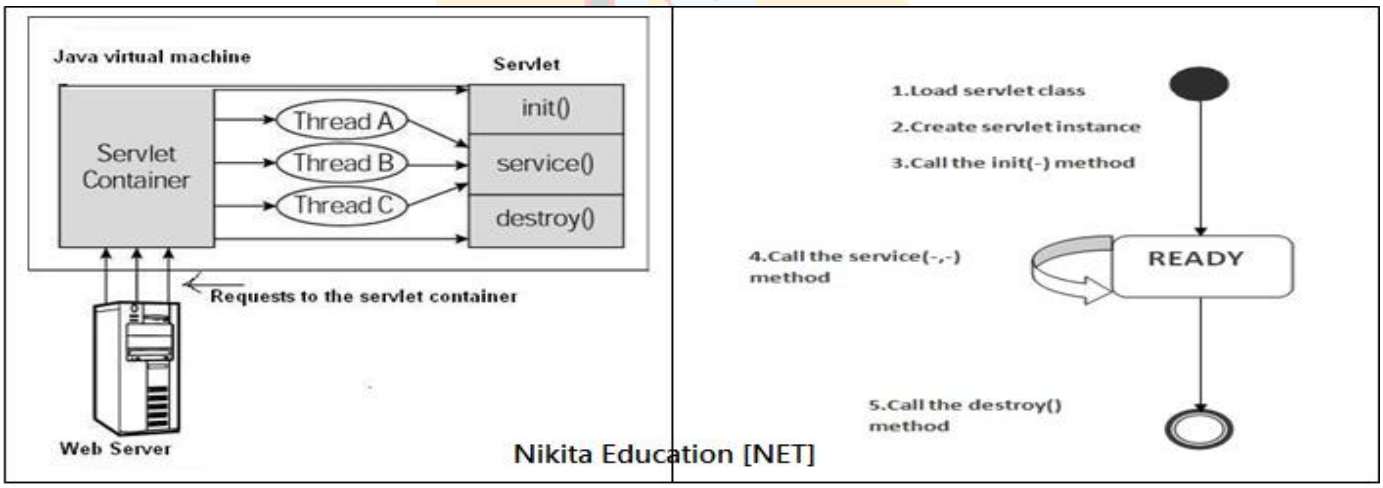
- The servlet is initialized by calling the **init ()** method.
- The servlet calls **service()** method to process a client's request.
- The servlet is terminated by calling the **destroy()** method.
- Finally, servlet is garbage collected by the garbage collector of the JVM.

Now let us discuss the life cycle methods in details.

The init() method :

The init method is designed to be called only once. It is called when the servlet is first created, and not called again for each user request. So, it is used for one-time initializations, just as with the init method of applets. The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started. When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread. The init() method simply creates or loads some data that will be used throughout the life of the servlet. The init method definition looks like this:

```
public void init() throws ServletException
{
        // Initialization code...
}
```



The service() method :

The service() method is the main method to perform the actual task. The servlet container (i.e. web server) calls the service() method to handle requests coming from the client(browsers) and to write the formatted response back to the client. Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate. Here is the signature of this method:

```
public void service(ServletRequest request, ServletResponse response)
throws ServletException, IOException
{
        //generate response to client
}
```

The service () method is called by the container and service method invokes doGet, doPost, doPut, doDelete, etc. methods as appropriate. The doGet() and doPost() are most frequently used methods within each service request.

The destroy() method :

The destroy() method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities. After the destroy() method is called, the servlet object is marked for garbage collection. The destroy method definition looks like this:

```
public void destroy() {
   // Finalization code...
}
```

I. Servlet API

The **javax.servlet** and **javax.servlet.http** packages represent interfaces and classes for servlet api. The javax.servlet package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol. The javax.servlet.http package contains interfaces and classes that are responsible for hyper text transfer protocol only.

The javax.servlet package

Interfaces:		
Servlet	ServletContext	ServletRequestListener
ServletRequest	SingleThreadModel	ServletRequestAttributeListener
ServletResponse	Filter	ServletContextListener
RequestDispatcher	FilterConfig	ServletContextAttributeListener
ServletConfig	FilterChain	

Classes:		
GenericServlet	ServletResponseWrapper	ServletContextAttributeEvent
ServletInputStream	ServletRequestEvent	ServletException
ServletOutputStream	ServletContextEvent	UnavailableException
ServletRequestWrapper	ServletRequestAttributeEvent	

I.1. Interfaces

1. Servlet

Servlet interface provides common behaviour to all the servlets. Servlet interface needs to be implemented for creating any servlet (either directly or indirectly). It provides 3 life cycle methods that are used to initialize the servlet, to service the requests, and to destroy the servlet and 2 non-life cycle methods.

Method	**Description**
public void init(ServletConfig config)	initializes the servlet. It is the life cycle method of servlet and invoked by the web container only once.
public void service(ServletRequest request, ServletResponse response)	provides response for the incoming request. It is invoked at each request by the web container.
public void destroy()	is invoked only once and indicates that servlet is being destroyed.
public ServletConfig getServletConfig()	returns the object of ServletConfig.
public String getServletInfo()	returns information about servlet such as writer, copyright, version etc.

2. ServletConfig

An object of ServletConfig is created by the web container for each servlet. This object can be used to get configuration information from web.xml file. If the configuration information is modified from the web.xml file, we don't need to change the servlet. So it is easier to manage the web application if any specific content is modified from time to time.

Methods:
public String getInitParameter(String name): Returns the parameter value for the specified parameter name.
public Enumeration getInitParameterNames(): Returns an enumeration of all the initialization parameter names.
public String getServletName(): Returns the name of the servlet.
public ServletContext getServletContext(): Returns an object of ServletContext.

Example: ServletConfig config=getServletConfig();

3. ServletContext

An object of ServletContext is created by the web container at time of deploying the project. This object can be used to get configuration information from web.xml file. There is only one ServletContext object per web application. If any information is shared to many servlet, it is better to provide it from the web.xml file using the **<context-param>** element.

Usage of ServletContext Interface

1. The object of ServletContext provides an interface between the container and servlet.
2. The ServletContext object can be used to get configuration information from the web.xml file.
3. The ServletContext object can be used to set, get or remove attribute from the web.xml file.
4. The ServletContext object can be used to provide inter-application communication.

Methods:
public String getInitParameter(String name): Returns the parameter value for the specified parameter name.
public Enumeration getInitParameterNames(): Returns the names of the context's initialization parameters.
public void setAttribute(String name,Object object):sets the given object in the application scope.
public Object getAttribute(String name): Returns the attribute for the specified name.
public Enumeration getInitParameterNames(): Returns the names of the context's initialization parameters as an Enumeration of String objects.
public void removeAttribute(String name): Removes the attribute with the given name from the servlet context.

Example: ServletContext application=getServletContext();

4. ServletRequest

An object of ServletRequest is used to provide the client request information to a servlet such as content type, content length, parameter names and values, header informations, attributes etc.

Method	Description
public String getParameter(String name)	is used to obtain the value of a parameter by name.
public String[] getParameterValues(String name)	returns an array of String containing all values of given parameter name. It is mainly used to obtain values of a Multi select list box.
java.util.Enumeration getParameterNames()	returns an enumeration of all of the request parameter names.
public int getContentLength()	Returns the size of the request entity data, or -1 if not known.
public String getCharacterEncoding()	Returns the character set encoding for the input of this request.
public String getContentType()	Returns the Internet Media Type of the request entity data, or null if not known.
public ServletInputStream getInputStream() throws IOException	Returns an input stream for reading binary data in the request body.
public abstract String getServerName()	Returns the host name of the server that received the request.
public int getServerPort()	Returns the port number on which this request was received.

5. ServletResponse

The ServletResponse interface contains various methods that enable a servlet to respond to the client requests. A servlet can send the response either as character or binary data. The PrintWriter stream can be used to send character data as servlet response, and ServletOutputStream stream to send binary data as servlet response.

Methods	Description
PrintWriter getWriter()	returns a PrintWriter object that can send character text to the client.
void setBufferSize(int size)	Sets the preferred buffer size for the body of the response
void setContentLength(int len)	Sets the length of the content body in the response In HTTP servlets, this method sets the HTTP Content-Length header
void setContentType(String type)	sets the content type of the response being sent to the client before sending the respond.
void setBufferSize(int size)	sets the preferred buffer size for the body of the response.
boolean isCommitted()	returns a boolean indicating if the response has been committed
void setLocale(Locale loc)	sets the locale of the response, if the response has not been committed yet.

I.2. Classes

1. GenericServlet

GenericServlet class implements Servlet, ServletConfig and Serializable interfaces. It provides the implementaion of all the methods of these interfaces except the service method. GenericServlet class can handle any type of request so it is protocol-independent. You may create a generic servlet by inheriting the GenericServlet class and providing the implementation of the service method.

Methods
public void init(ServletConfig config) is used to initialize the servlet.
public abstract void service(ServletRequest request, ServletResponse response) provides service for the incoming request. It is invoked at each time when user requests for a servlet.
public void destroy() is invoked only once throughout the life cycle and indicates that servlet is being destroyed.
public ServletConfig getServletConfig() returns the object of ServletConfig.
public String getServletInfo() returns information about servlet such as writer, copyright, version etc.
public void init() it is a convenient method for the servlet programmers, now there is no need to call super.init(config)
public ServletContext getServletContext() returns the object of ServletContext.
public String getInitParameter(String name) returns the parameter value for the given parameter name.
public Enumeration getInitParameterNames() returns all the parameters defined in the web.xml file.
public String getServletName() returns the name of the servlet object.
public void log(String msg) writes the given message in the servlet log file.
public void log(String msg,Throwable t) writes the explanatory message in the servlet log file and a stack trace.

2. ServletInputStream

ServletInputStream class provides stream to read binary data such as image etc. from the request object. It is an abstract class. The **getInputStream()** method of **ServletRequest** interface returns the instance of ServletInputStream class. So can be get as:

ServletInputStream sin=request.getInputStream();

Method: **int readLine(byte[] b, int off, int len) :** it reads the input stream.

3. ServletOutputStream

ServletOutputStream class provides a stream to write binary data into the response. It is an abstract class. The **getOutputStream()** method of **ServletResponse** interface returns the instance of ServletOutputStream class. It may be get as:

ServletOutputStream out=response.getOutputStream();

Methods:

void print(boolean b){}	void print(double d){}	void println(int i){}
void print(char c){}	void print(String s){}	void println(long l){}
void print(int i){}	void println{}	void println(float f){}
void print(long l){}	void println(boolean b){}	void println(double d){}
void print(float f){}	void println(char c){}	void println(String s){}

4. ServletException

javax.servlet defines two exceptions. The first is **ServletException**, which indicates that a servlet problem has occurred. The second is **UnavailableException**, which extends ServletException. It indicates that a servlet is unavailable.

The javax.servlet.http package

Interfaces:	
HttpServletRequest HttpServletResponse HttpSession HttpSessionListener	HttpSessionAttributeListener HttpSessionBindingListener HttpSessionActivationListener HttpSessionContext (deprecated now)

Classes:	
HttpServlet Cookie HttpServletRequestWrapper HttpServletResponseWrapper	HttpSessionEvent HttpSessionBindingEvent HttpUtils (deprecated now)

I.3. Interfaces

1. HttpServletRequest

The HttpServletRequest interface enables a servlet to obtain information about a client.

Method Summary	
String	**getContextPath**() Returns the portion of the request URI that indicates the context of the request.
Cookie[]	**getCookies**() Returns an array containing all of the Cookie objects the client sent with this request.
String	**getHeader**(java.lang.String name) Returns the value of the specified request header as a String.
String	**getMethod**() Returns the name of the HTTP method with which this request was made, for example, GET, POST, or PUT.
String	**getPathInfo**() Returns any extra path information associated with the URL the client sent when it made this request.
StringBuffer	**getRequestURL**() Reconstructs the URL the client used to make the request.
String	**getServletPath**() Returns the part of this request's URL that calls the servlet.
HttpSession	**getSession**() Returns the current session associated with this request, or if the request does not have a session, creates one.
void	**login**(java.lang.String username, java.lang.String password) Validate the provided username and password in the password validation realm used by the web container login mechanism configured for the ServletContext.
void	**logout**() Establish null as the value returned when getUserPrincipal, getRemoteUser, and getAuthType is called on the request.

2. HttpServletResponse

The HttpServletResponse interface enables a servlet to formulate an HTTP response to a client. Several constants are defined. These correspond to the different status codes that can be assigned to an HTTP response. For example, SC_OK indicates that the HTTP request succeeded, and SC_NOT_FOUND indicates that the requested resource is not available.

Method Summary	
void	**addCookie**(Cookie cookie) Adds the specified cookie to the response.

void	**addHeader**(java.lang.String name, java.lang.String value) Adds a response header with the given name and value.
String	**encodeRedirectURL**(java.lang.String url) Encodes the specified URL for use in the sendRedirect method or, if encoding is not needed, returns the URL unchanged.
String	**encodeURL**(java.lang.String url) Encodes the specified URL by including the session ID in it, or, if encoding is not needed, returns the URL unchanged.
void	**sendError**(int sc, java.lang.String msg) Sends an error response to the client using the specified status and clears the buffer.
void	**sendRedirect**(java.lang.String location) Sends a temporary redirect response to the client using the specified redirect location URL and clears the buffer.
void	**setHeader**(java.lang.String name, java.lang.String value) Sets a response header with the given name and value.
void	**setStatus**(int sc) Sets the status code for this response.

3. HttpSession

The HttpSession interface enables a servlet to read and write the state information that is associated with an HTTP session. **HttpSession** object is used to store entire session with a specific client. We can store, retrieve and remove attribute from **HttpSession** object. Any servlet can have access to **HttpSession** object throughout the getSession() method of the **HttpServletRequest** object.

Methods	**Description**
long getCreationTime()	returns the time when the session was created, measured in milliseconds since midnight January 1, 1970 GMT.
String getId()	returns a string containing the unique identifier assigned to the session.
long getLastAccessedTime()	returns the last time the client sent a request associated with the session
int getMaxInactiveInterval()	returns the maximum time interval, in seconds.
void invalidate()	destroy the session
boolean isNew()	returns true if the session is new else false
void setMaxInactiveInterval(int interval)	Specifies the time, in seconds,after servlet container will invalidate the session.

I.4. Classes

1. HttpServlet

The HttpServlet class extends the GenericServlet class and implements Serializable interface. HttpServlet is also an abstract class. This class gives implementation of various service() methods of **Servlet** interface. To create a servlet, we should create a class that extends **HttpServlet** abstract class. The Servlet class that we will create, must not override service() method. Our servlet class will override only the doGet() and/or doPost() methods. The service() method of **HttpServlet** class listens to the Http methods (GET, POST etc) from request stream and invokes doGet() or doPost() methods based on Http Method type.

Methods
public void service(ServletRequest req,ServletResponse res) dispatches the request to the protected service method by converting the request and response object into http type. **protected void service(HttpServletRequest req, HttpServletResponse res)** receives the request from the service method, and dispatches the request to the doXXX() method depending on the incoming

http request type.
protected void doGet(HttpServletRequest req, HttpServletResponse res) handles the GET request. It is invoked by the web container.
protected void doPost(HttpServletRequest req, HttpServletResponse res) handles the POST request. It is invoked by the web container.
protected void doHead(HttpServletRequest req, HttpServletResponse res) handles the HEAD request. It is invoked by the web container.
protected void doOptions(HttpServletRequest req, HttpServletResponse res) handles the OPTIONS request. It is invoked by the web container.
protected void doPut(HttpServletRequest req, HttpServletResponse res) handles the PUT request. It is invoked by the web container.
protected void doTrace(HttpServletRequest req, HttpServletResponse res) handles the TRACE request. It is invoked by the web container.
protected void doDelete(HttpServletRequest req, HttpServletResponse res) handles the DELETE request. It is invoked by the web container.
protected long getLastModified(HttpServletRequest req) returns the time when HttpServletRequest was last modified since midnight January 1, 1970 GMT.

2. Cookies

Cookies are small pieces of information that are sent in response from the web server to the client. **Cookies** are the simplest technique used for storing client state. **Cookies** are stored on client's computer. They have a lifespan and are destroyed by the client browser at the end of that lifespan. It is information for future use that is stored by the server on the client side of a client/server communication. Cookies are usually small text files, given ID tags that are stored on your computer's browser directory or program data subfolders. Cookies are created using **Cookie** class present in Servlet API. Cookies are added to **response** object using the addCookie() method. This method sends cookie information over the HTTP response stream. getCookies() method is used to access the cookies that are added to response object.

Advantage of Cookies
1. Simplest technique of maintaining the state.
2. Cookies are maintained at client side.

Disadvantage of Cookies
1. It will not work if cookie is disabled from the browser.
2. Only textual information can be set in Cookie object.

Constructor	Description
Cookie()	constructs a cookie.
Cookie(String name, String value)	constructs a cookie with a specified name and value.

Method	Description
public void setMaxAge(int expiry)	Sets the maximum age of the cookie in seconds.
public String getName()	Returns the name of the cookie. The name cannot be changed after creation.
public String getValue()	Returns the value of the cookie.
public void setName(String name)	changes the name of the cookie.
public void setValue(String value)	changes the value of the cookie.

Other methods required for using Cookies

public void addCookie(Cookie ck):method of HttpServletResponse interface is used to add cookie in response object.
public Cookie[] getCookies():method of HttpServletRequest interface is used to return all the cookies from the browser.

Creating a new Cookie

```
Cookie ck = new Cookie("username",name);
```

creating a new cookie object

Setting up lifespan for a cookie

```
ck.setMaxAge(30*60);
```

setting maximum age of cookie

Sending the cookie to the client

```
response.addCookie(ck);
```

adding cookie to **response** object

Getting cookies from client request


```
Cookie[] cks = request.getCookies();
```

getting the cookie for **request** object

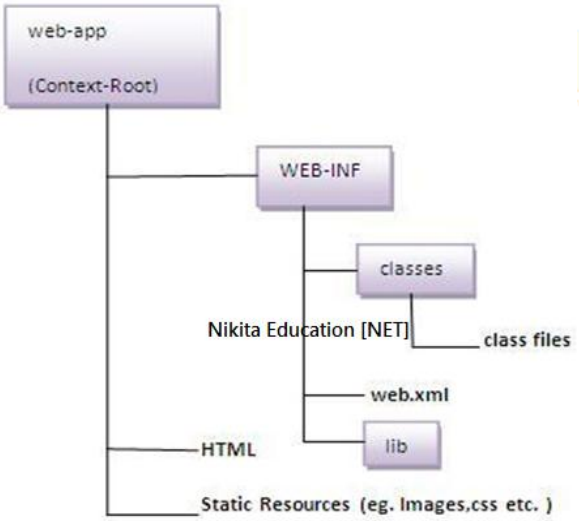
J. Servlet Development

Steps to create a servlet example

1. Create a directory structure
2. Create a Servlet
3. Compile the Servlet
4. Create a deployment descriptor
5. Start the server and deploy the project

Create a directory structures

The **directory structure** is a hierarchy of folders that defines where to put the different types of files so that web container may get the information and respond to the client. The Sun Microsystem defines a unique standard to be followed by all the server vendors. Let's see the directory structure that must be followed to create the servlet.



As you can see that the servlet class file must be in the classes folder. The web.xml file must be under the WEB-INF folder.

Create a Servlet: There are three ways to create the servlet.

1. By implementing the Servlet interface
2. By inheriting the GenericServlet class
3. By inheriting the HttpServlet class

The HttpServlet class is widely used to create the servlet because it provides methods to handle http requests such as doGet(), doPost, doHead() etc.

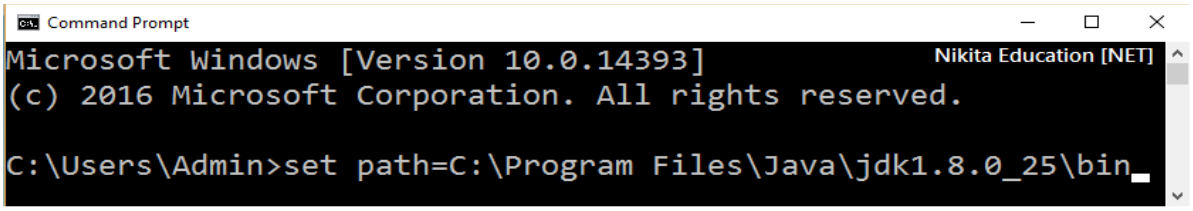
```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;

public class DemoServlet extends HttpServlet
{
     public void doGet(HttpServletRequest req,
                       HttpServletResponse res) throws
                                 ServletException, IOException
     {
          // setting the content type
          res.setContentType("text/html");

          //get the stream to write the data
          PrintWriter pw=res.getWriter();
          //writing html in the stream
          pw.println("<html><head><title>");
          pw.println("Nikita Education [NET]");
          pw.println("</title></head>");
          pw.println("<body>");
          pw.println("Welcome to Nikita Education [NET]");
          pw.println("</body></html>");
          pw.close();//closing the stream
     }
}
```

Note: Save above file as a DemoServlet.java

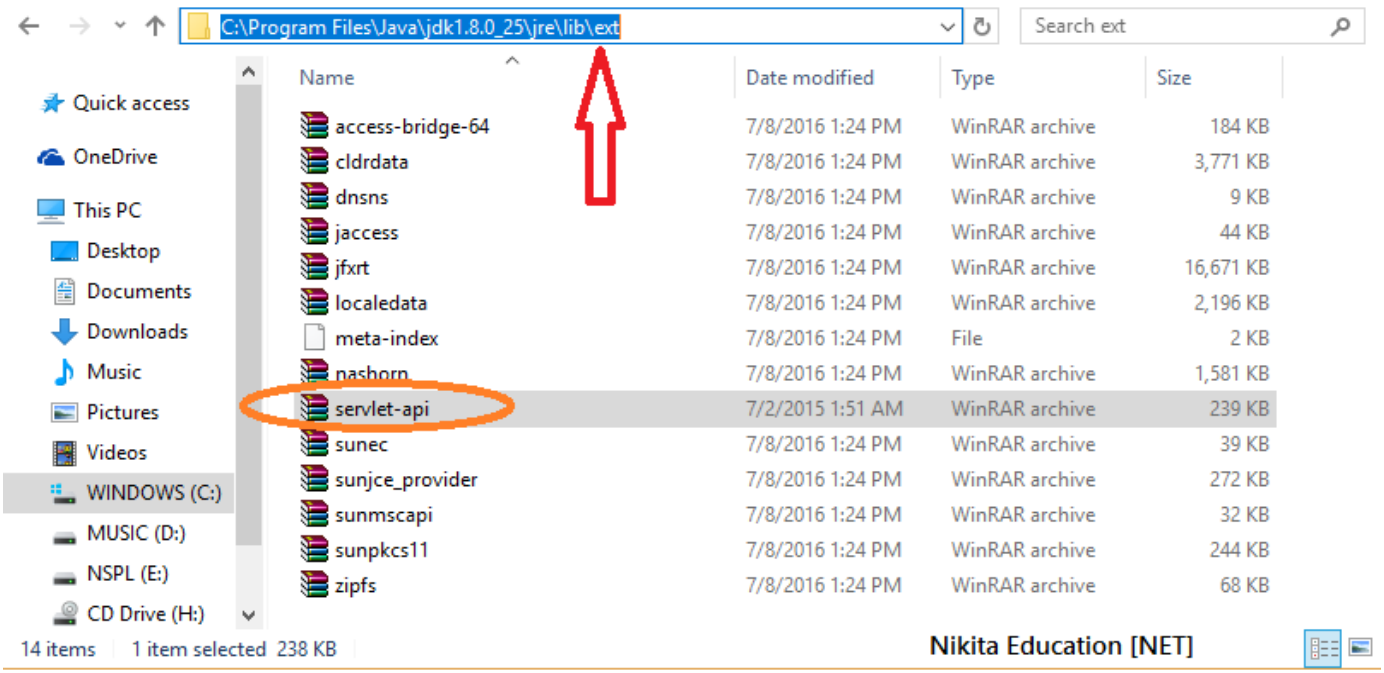
Compile the servlet

To compile a Servlet a JAR file is required. Different servers require different JAR files. In Apache Tomcat server servlet-api.jar file is required to compile a servlet class.

- Set the Path for Java.



- Download **servlet-api.jar** file.
- Paste the **servlet-api.jar** file inside **Java\jdk\jre\lib\ext** directory.



- Compile the Servlet class.

NOTE: After compiling your Servlet class you will have to paste the class file into WEB-INF/classes/ directory.

Create the deployment descriptor (web.xml file)

The deployment descriptor is an xml file, from which Web Container gets the information about the servlet to be invoked. The web container uses the Parser to get the information from the web.xml file. There are many xml parsers such as SAX, DOM and Pull. There are many elements in the web.xml file. Here is given some necessary elements to run the simple servlet program.

web.xml file

```
<web-app>

        <servlet>
                <servlet-name>MyFirstHttpServlet</servlet-name>
                <servlet-class>DemoServlet</servlet-class>
        </servlet>

        <servlet-mapping>
                <servlet-name> MyFirstHttpServlet </servlet-name>
                <url-pattern>/welcome</url-pattern>
        </servlet-mapping>

</web-app>
```

Description of the elements of web.xml file

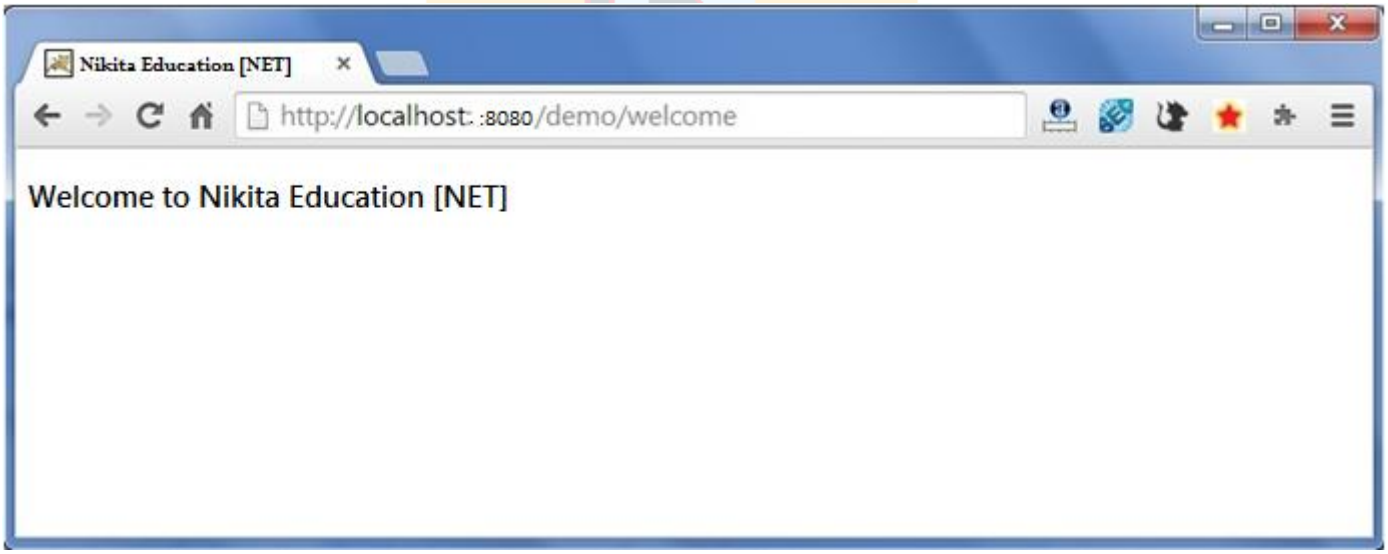
There are too many elements in the web.xml file. Here is the illustration of some elements that is used in the above web.xml file. The elements are as follows:

<web-app> represents the whole application.
<servlet> is sub element of <web-app> and represents the servlet.
<servlet-name> is sub element of <servlet> represents the name of the servlet.
<servlet-class> is sub element of <servlet> represents the class of the servlet.
<servlet-mapping> is sub element of <web-app>. It is used to map the servlet.
<url-pattern> is sub element of <servlet-mapping>. This pattern is used at client side to invoke the servlet.

Start the Server and deploy the project

To start Apache Tomcat server, double click on the startup.bat file under apache-tomcat/bin directory. Copy the project and paste it in the webapps folder under apache tomcat.

Run application in Web Browser



K. How Servlet works?

It is important to learn how servlet works for understanding the servlet well. Here, we are going to get the internal detail about the first servlet program. The server checks if the servlet is requested for the first time.

If yes, web container does the following tasks:

- loads the servlet class.
- instantiates the servlet class.
- calls the init method passing the ServletConfig object

else

- calls the service method passing request and response objects

The web container calls the destroy method when it needs to remove the servlet such as at time of stopping server or undeploying the project.

How web container handles the servlet request?

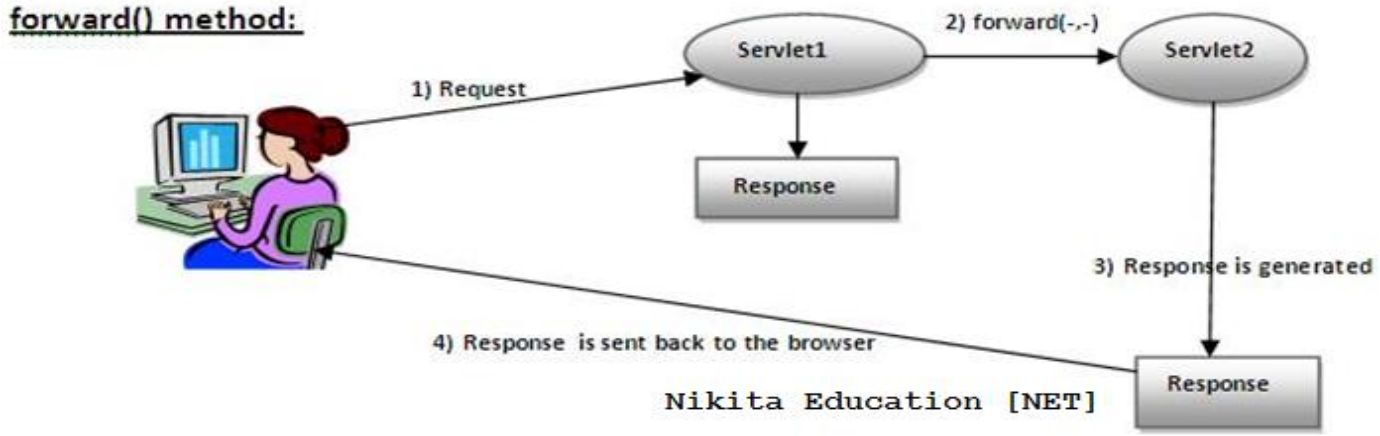
The web container is responsible to handle the request. Let's see how it handles the request.

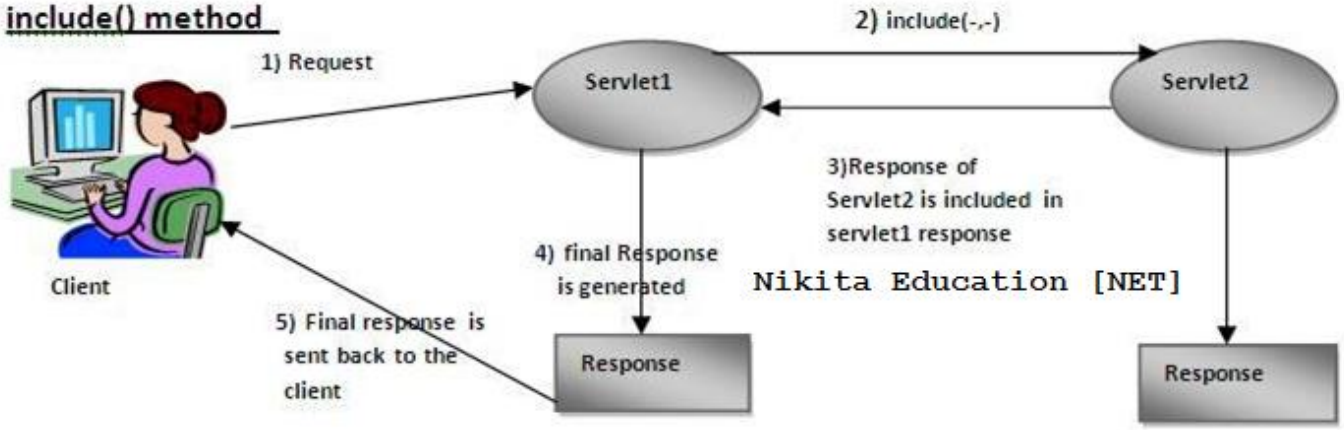
- maps the request with the servlet in the web.xml file.
- creates request and response objects for this request
- calls the service method on the thread
- The public service method internally calls the protected service method
- The protected service method calls the doGet method depending on the type of request.
- The doGet method generates the response and it is passed to the client.
- After sending the response, the web container deletes the request and response objects. The thread is contained in the thread pool or deleted depends on the server implementation.

L. RequestDispatcher

The RequestDispacher interface provides the facility of dispatching the request to another resource it may be html, servlet or jsp. This interface can also be used to include the content of another resource also. It is one of the ways of servlet collaboration. There are two methods defined in the RequestDispatcher interface.

Methods	Description
void forward(ServletRequest request, ServletResponse response)	forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server
void include(ServletRequest request, ServletResponse response)	includes the content of a resource (servlet, JSP page, HTML file) in the response

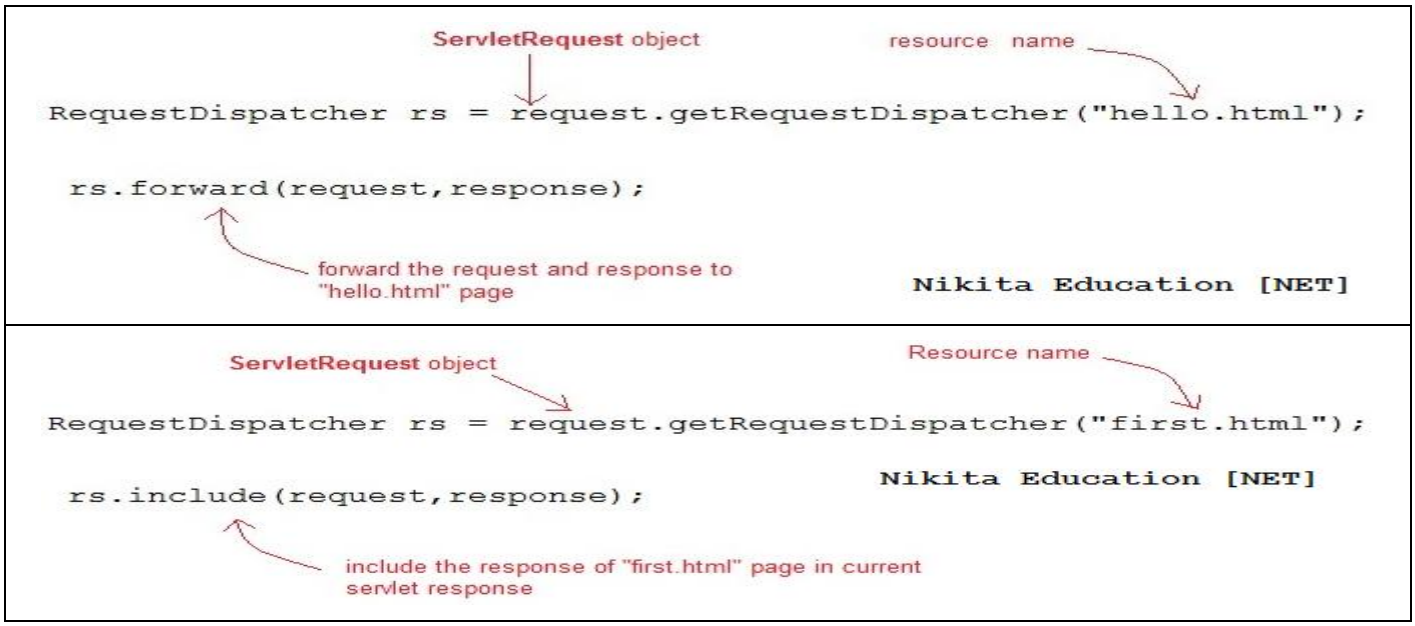




How to get an Object of RequestDispatcher

getRequestDispatcher() method of **ServletRequest** returns the object of **RequestDispatcher**.

Syntax: public RequestDispatcher getRequestDispatcher(String resource);



M. sendRedirect method of HttpServletResponse interface

The **sendRedirect()** method of **HttpServletResponse** interface can be used to redirect response to another resource, it may be servlet, jsp or html file. It accepts relative as well as absolute URL. It works at client side because it uses the url bar of the browser to make another request. So, it can work inside and outside the server.

forward() method	**sendRedirect() method**
The forward() method works at server side.	The sendRedirect() method works at client side.
It sends the same request and response objects to another servlet.	It always sends a new request.
It can work within the server only.	It can be used within and outside the server.
Example: request.getRequestDispacher("servlet2").forward(request,response);	Example: response.sendRedirect("servlet2");

Syntax: public void sendRedirect(String URL)throws IOException;

Example:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class MyServlet extends HttpServlet
{
     protected void doGet(HttpServletRequest request,
                          HttpServletResponse response)
                          throws ServletException, IOException
     {
          response.setContentType("text/html;charset=UTF-8");
          PrintWriter out = response.getWriter();
          try{
               response.sendRedirect("http://www.nspl.in");
          }finally {
               out.close();
          }
     }
}
```

N. Session Tracking or Session Management

Session: The session of activity that a user with a unique IP address spends on a Web site during a specified period of time. The number of user sessions on a site is used in measuring the amount of traffic a Web site gets. The site administrator determines what the time frame of a user session will be (e.g., 30 minutes). If the visitor comes back to the site within that time period, it is still considered one user session because any number of visits within that 30 minutes will only count as one session. If the visitor returns to the site after the allotted time period has expired, say an hour from the initial visit, then it is counted as a separate user session. **session**: an abstract concept to represent a series of HTTP requests and responses between a specific Web browser and server. HTTP doesn't support the notion of a session, but Java does.

sessions vs. cookies:

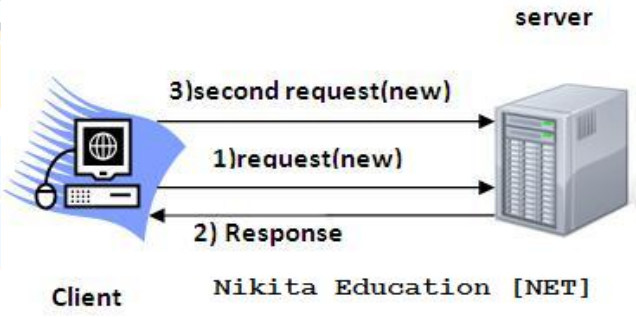
- a cookie is data stored on the client
- a session's data is stored on the server (only 1 session per client)

Session tracking is the capability of a server to maintain the current state of a single client's sequential requests. The HTTP protocol used by Web servers is stateless. This means that every transaction is autonomous. This type of stateless transaction is not a problem unless you need to know the sequence of actions a client has performed while at your site. **Session Management** is a mechanism used by the **Web container** to store session information for a particular user.

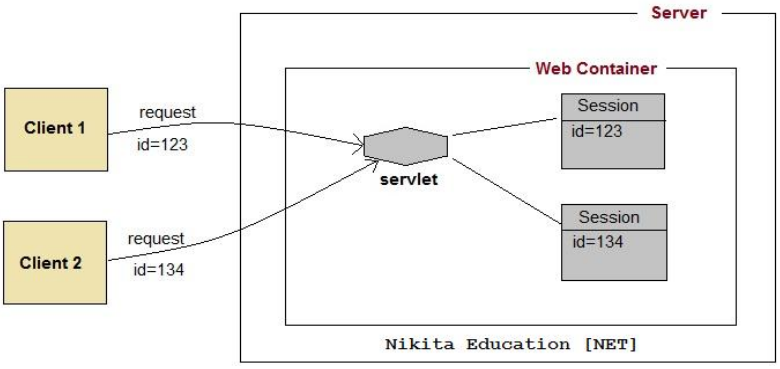
Session Tracking in Servlets

Session simply means a particular interval of time. **Session Tracking** is a way to maintain state (data) of an user. It is also known as **session management** in servlet. Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user. HTTP is stateless that means each request is considered as the new request. It is shown in the figure given below:

Why use Session Tracking?:
To recognize the user



How Session Works



The basic concept behind session is, whenever a user starts using our application, we can save a unique identification information about him, in an object which is available throughout the application, until its destroyed. So wherever the user goes, we will always have his information and we can always manage which user is doing what. Whenever a user wants to exit from your application, destroy the object with his information.

Session Tracking Techniques

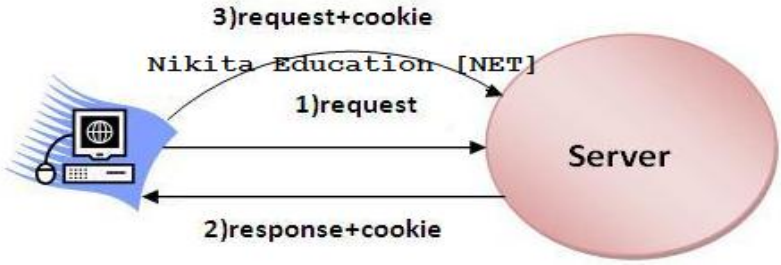
- **Cookies:** You can use HTTP cookies to store information. Cookies will be stored at browser side.
- **URL rewriting:** With this method, the information is carried through url as request parameters. In general added parameter will be sessionid, userid.
- **HttpSession:** Using HttpSession, we can store information at server side. Http Session provides methods to handle session related information.
- **Hidden form fields:** By using hidden form fields we can insert information in the webpages and these information will be sent to the server. These fields are not visible directly to the user, but can be viewed using view source option from the browsers. The hidden form fields are as given below: <input type='hidden' name='siteName' value='nspl'/>

1. Cookies

A **cookie** is a small piece of information that is persisted between the multiple client requests. A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

How Cookie works

By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser. After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.



Types of Cookie

1. **Non-persistent cookie**
   It is **valid for single session** only. It is removed each time when user closes the browser.
2. **Persistent cookie**
   It is **valid for multiple session** . It is not removed each time when user closes the browser. It is removed only if user logout or signout.

Advantage of Cookies

1. Simplest technique of maintaining the state.
2. Cookies are maintained at client side.

Disadvantage of Cookies

1. It will not work if cookie is disabled from the browser.
2. Only textual information can be set in Cookie object.

javax.servlet.http.Cookie class provides the functionality of using cookies. It provides a lot of useful methods for cookies.

2. URL Rewriting

In URL rewriting, we append a token or identifier to the URL of the next Servlet or the next resource. We can send parameter name/value pairs using the following format:

url?name1=value1&name2=value2&??

A name and a value is separated using an equal = sign, a parameter name/value pair is separated from another parameter using the ampersand(&). When the user clicks the hyperlink, the parameter name/value pairs will be passed to the server. From a Servlet, we can use getParameter() method to obtain a parameter value.

Advantage of URL Rewriting

1. It will always work whether cookie is disabled or not (browser independent).
2. Extra form submission is not required on each pages.

Disadvantage of URL Rewriting

1. It will work only with links.
2. It can send Only textual information.

3. HttpSession

In such case, container creates a session id for each user. The container uses this id to identify the particular user. An object of HttpSession can be used to perform two tasks:

1. bind objects
2. view and manipulate information about a session, such as the session identifier, creation time, and last accessed time.

How to get the HttpSession object ?

The HttpServletRequest interface provides two methods to get the object of HttpSession:

1. **public HttpSession getSession():**Returns the current session associated with this request, or if the request does not have a session, creates one.
2. **public HttpSession getSession(boolean create):**Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session.

Commonly used methods of HttpSession interface

1. **public String getId():**Returns a string containing the unique identifier value.
2. **public long getCreationTime():**Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
3. **public long getLastAccessedTime():**Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.
4. **public void invalidate():**Invalidates this session then unbinds any objects bound to it.

4. Hidden Form Fields

In case of Hidden Form Field **a hidden (invisible) textfield** is used for maintaining the state of an user. In such case, we store the information in the hidden field and get it from another servlet. This approach is better if we have to submit form in all the pages and we don't want to depend on the browser. Let's see the code to store value in hidden field.

```
<input type="hidden" name="uname" value="Nikita Education [NET]">
```

Here, uname is the hidden field name and Vimal Jaiswal is the hidden field value.

Real application of hidden form field

It is widely used in comment form of a website. In such case, we store page id or page name in the hidden field so that each page can be uniquely identified.

Advantage of Hidden Form Field

1. It will always work whether cookie is disabled or not.

Disadvantage of Hidden Form Field:

1. It is maintained at server side.
2. Extra form submission is required on each pages.
3. Only textual information can be used.

O. Examples

1. Generic Servlet Example

MyGenericServlet.java

```
import java.io.*;
import javax.servlet.*;

public class MyGenericServlet extends GenericServlet
{
        public void service(ServletRequest request, ServletResponse response)
                                                        throws ServletException, IOException
        {
                response.setContentType("text/html");
                PrintWriter pw=response.getWriter();

                pw.println("<html><head><title> My Simple Generic Servlet </title></head>");
                pw.println("<body><center> Example of generic servlet </center><br><br>");
                pw.println("<hr/><br><form name=form1 ");
                pw.println("action=\"http://localhost:8080/GenApp/welcome.html\">");
                pw.println("Enter Name : <input type=textbox name=text1 size=50 value=\"\"/><br>");
                pw.println("Enter Addr : <input type=textbox name=text2 size=50 value=\"\"/><br>");
                pw.println("<input type=submit name=submit1 value=\"submit\"/>");
                pw.rpintln("</form></body></html>");
                pw.close();
        }
}
```

web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app>
        <description>
                Generic Servlet Example.
        </description>

        <display-name>Generic Servlet Example</display-name>

        <servlet>
                <servlet-name>nsplservlet</servlet-name>
                <servlet-class>MyGenericServlet</servlet-class>
        </servlet>

         <servlet-mapping>
                <servlet-name>nsplservlet</servlet-name>
                <url-pattern>/nspl</url-pattern>
        </servlet-mapping>
</web-app>
```

welcome.html

```
<html>
        <head>
                <title> Nikita Education [NET] </title>
        </head>
        <body>
                <h1>Hello, you have successfully
                executed the GenericServlet</h1>
        </body>
</html>
```

2. HttpServlet Example

login.html

```
<html>
	<head>
		<title>NSPL HttpServletDemo - Login Page </title>
	</head>
	<body>
		<h1> Welcome to Nikita Education [NET] </h1>
		<hr><br>
		<form name=form1 method=get action="/nspl/nsplhttpservlet">
			<b> Enter User Name : </b>
			<input type=text name=text1 size=35 value=""/><br>
			<b> Enter User Pass : </b>
			<input type=text name=text2 size=35 value=""/>
			<br><br><br>
			<input type=submit name=submit1 value="Login"/>
			<input type=submit name=submit2 value="Cancel"/>
		</form>
	</body>
</html>
```

web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app>
	<description>
		NSPL - Http Servlet Examples.
	</description>

	<display-name>NSPL - Http Servlet Examples</display-name>

	<servlet>
		<servlet-name>nsplhttpserv</servlet-name>
		<servlet-class>nsplhttpservlet.MyHttpServletDemo</servlet-class>
	</servlet>

	<servlet-mapping>
		<servlet-name>nsplhttpserv</servlet-name>
		<url-pattern>/nsplhttpservlet</url-pattern>
	</servlet-mapping>

	<welcome-file-list>
		<welcome-file>login.html</welcome-file>
	</welcome-file-list>

</web-app>
```

MyHttpServletDemo.java

```
package nsplhttpservlet;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class MyHttpServletDemo extends HttpServlet
{
	String usr="nspl";
	String pwd="nspl58";
```

```
        public void doGet(HttpServletRequest request, HttpServletResponse response)
                                                        throws ServletException, IOException
        {
                String log=request.getParameter("submit1");
                String log1=request.getParameter("submit2");
                if (log.equals("Login"))
                {
                        String u=request.getParameter("text1");
                        String p=request.getParameter("text2");

                        if (usr.equals(u) && pwd.equals(p))
                        {
                                response.setContentType("text/html");
                                PrintWriter out=response.getWriter();
                                out.println("<html><head><title>my http servlet</title></head>");
                                out.println("<body><center><h1> Welcome to Nikita Education
                                                                [NET] </h1></center>");
                                out.println("<br><hr><center><h3><b> you have successfully executed
                                                        the http servlet</b></h3></center>");
                                out.println("</body></html>");
                                out.close();
                        }
                        else
                        {
                                response.sendRedirect(request.getContextPath()+"/login.html");
                        }
                }
                else if(log1.equals("Cancel"))
                {
                        response.sendRedirect(request.getContextPath()+"/login.html");
                }
        }
}
```

3. RequestDispatcher Example

login.html

```
<form action="go" method="get" action="/go">
        Name:<input type="text" name="userName"/><br/>
        Password:<input type="password" name="userPass"/><br/>
        <input type="submit" value="login"/>
</form>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
        <servlet>
                <servlet-name>Simple</servlet-name>
                <servlet-class>Simple</servlet-class>
        </servlet>
        <servlet>
                <servlet-name>WelcomeServlet</servlet-name>
                <servlet-class>WelcomeServlet</servlet-class>
        </servlet>
        <servlet-mapping>
                <servlet-name>Simple</servlet-name>
                <url-pattern>/go</url-pattern>
        </servlet-mapping>
```

```
        <servlet-mapping>
                <servlet-name>WelcomeServlet</servlet-name>
                <url-pattern>/welcome</url-pattern>
        </servlet-mapping>
        <welcome-file-list>
                <welcome-file>login.html</welcome-file>
        </welcome-file-list>
</web-app>
```

Simple.java

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.*;
import javax.servlet.http.*;

public class Simple extends HttpServlet
{
        public void doGet(HttpServletRequest request, HttpServletResponse response)
                                                        throws ServletException, IOException
        {
                response.setContentType("text/html");
                PrintWriter out = response.getWriter();

                String p=request.getParameter("userPass");
                if(p.equals("servlet")){
                        RequestDispatcher rd=request.getRequestDispatcher("welcome");
                        rd.forward(request, response);

                }
                else{
                        out.print("Sorry username or password error!");
                        RequestDispatcher rd=request.getRequestDispatcher("login.html");
                        rd.include(request, response);
                }
        }
}
```

WelcomeServlet.java

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class WelcomeServlet extends HttpServlet
{
        public void doGet(HttpServletRequest request, HttpServletResponse response)
                                                        throws ServletException, IOException
        {
                response.setContentType("text/html");
                PrintWriter out = response.getWriter();
                String n=request.getParameter("userName");
                out.print("Welcome "+n);
        }
}
```

4. sendRedirect() Example

index.html

```
<html>
	<head>
		<title>sendRedirect example</title>
	</head>
	<body>
		<form action="MySearcher">
			<input type="text" name="name">
			<input type="submit" value="Google Search">
		</form>
	</body>
</html>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
	<display-name>GoogleSearcher</display-name>
	<welcome-file-list>
		<welcome-file>index.html</welcome-file>
	</welcome-file-list>
	<servlet>
		<servlet-name>MySearcher</servlet-name>
		<servlet-class>MySearcher</servlet-class>
	</servlet>
	<servlet-mapping>
		<servlet-name>MySearcher</servlet-name>
		<url-pattern>/MySearcher</url-pattern>
	</servlet-mapping>
</web-app>
```

MySearcher.java

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class MySearcher extends HttpServlet
{
	protected void doGet(HttpServletRequest request, HttpServletResponse response)
                                                                throws ServletException, IOException
	{
		String name=request.getParameter("name");
		response.sendRedirect("https://www.google.co.in/#q="+name);
	}
}
```

P. Session Tracking Examples

1. Cookies

cookies.html

```
<html>
	<head>
		<title> NSPL - Cookies Demo </title>
```

```
	</head>
	<body>
		<h1> Welcome to Nikita Education [NET] </h1>
		<hr><br>
		<form name=form1 method=post action="/myCookieApps/addcookieserv">
			<b> Enter a value for cookie :  </b>
			<input type=text name=text1 size=25 value=""/>
			<br><br>
			<input type=submit name=submit1 value="AddCookie"/>
		</form>
	</body>
</html>
```

web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app>
	<servlet>
		<servlet-name>addcookieserv</servlet-name>
		<servlet-class>mycookies.AddCookieServlet</servlet-class>
	</servlet>
	<servlet-mapping>
		<servlet-name>addcookieserv</servlet-name>
		<url-pattern>/addcookieserv</url-pattern>
	</servlet-mapping>

	<servlet>
		<servlet-name>getcookieserv</servlet-name>
		<servlet-class>mycookies.GetCookiesServlet</servlet-class>
	</servlet>
	<servlet-mapping>
		<servlet-name>getcookieserv</servlet-name>
		<url-pattern>/getcookieserv</url-pattern>
	</servlet-mapping>

	<welcome-file-list>
		<welcome-file>cookies.html</welcome-file>
	</welcome-file-list>
</web-app>
```

AddCookieServlet.java

```
package mycookies;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class AddCookieServlet extends HttpServlet
{
	public void doPost(HttpServletRequest request, HttpServletResponse response)
                                                        throws ServletException, IOException
	{
		String data=request.getParameter("text1");

		Cookie cook=new Cookie("MyCookie",data);

		response.addCookie(cook);

		response.setContentType("text/html");
```

```
                PrintWriter out=response.getWriter();
                out.println("<h2> my cookie has been set to : "+data+"</h2>");
                out.close();
        }
}
```

GetCookiesServlet.java

```
package mycookies;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class GetCookiesServlet extends HttpServlet
{
        public void doGet(HttpServletRequest request, HttpServletResponse response)
                                                                throws ServletException, IOException
        {
                Cookie [] cookies=request.getCookies();

                response.setContentType("text/html");
                PrintWriter out=response.getWriter();

                out.println("<b>");
                for (int i=0;i<cookies.length ; i++)
                {
                        String name=cookies[i].getName();
                        String value=cookies[i].getValue();
                        out.println("name : "+name+"<br> value : "+value+"<br>");
                }
                out.println("</b>");
                out.close();
        }
}
```

2. URL Rewriting

index.html

```
<form action="servlet1">
        Name:<input type="text" name="userName"/><br/>
        <input type="submit" value="go"/>
</form>
```

web.xml

```
<web-app>

<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
```

```
</servlet>

<servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>

</web-app>
```

FirstServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class FirstServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response){
                try{

                response.setContentType("text/html");
                PrintWriter out = response.getWriter();

                String n=request.getParameter("userName");
                out.print("Welcome "+n);

                //appending the username in the query string
                out.print("<a href='servlet2?uname="+n+"'>visit</a>");

                out.close();

            }catch(Exception e){System.out.println(e);}
        }

}
```

SecondServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SecondServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response){
                try{

                response.setContentType("text/html");
                PrintWriter out = response.getWriter();

                //getting value from the query string
                String n=request.getParameter("uname");
                out.print("Hello "+n);

                out.close();

            }catch(Exception e){System.out.println(e);}
        }
}
```

3. HttpSession

index.html

```
<form action="servlet1">
        Name:<input type="text" name="userName"/><br/>
        <input type="submit" value="go"/>
</form>
```

web.xml

```
<web-app>

<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>

</web-app>
```

FirstServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class FirstServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response){
                try{

                response.setContentType("text/html");
                PrintWriter out = response.getWriter();

                String n=request.getParameter("userName");
                out.print("Welcome "+n);

                HttpSession session=request.getSession();
                session.setAttribute("uname",n);
                out.print("<a href='servlet2'>visit</a>");

                out.close();

            }catch(Exception e){System.out.println(e);}
        }
}
```

SecondServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SecondServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response){
                try{

                response.setContentType("text/html");
                PrintWriter out = response.getWriter();

                HttpSession session=request.getSession(false);
                String n=(String)session.getAttribute("uname");
                out.print("Hello "+n);

                out.close();

            }catch(Exception e){System.out.println(e);}
        }
}
```

4. Hidden Form Fields

index.html

```
<form action="go" method="get">
        Name:<input type="text" name="uname"/><br/>
        <input type="submit" value="go"/>
</form>
```

web.xml

```
<web-app>
<servlet>
        <servlet-name>Servlet1</servlet-name>
        <servlet-class>FirstServlet</servlet-class>
</servlet>
<servlet>
        <servlet-name>SecondServlet</servlet-name>
        <servlet-class>SecondServlet</servlet-class>
</servlet>

<servlet-mapping>
        <servlet-name>Servlet1</servlet-name>
        <url-pattern>/go</url-pattern>
</servlet-mapping>
<servlet-mapping>
        <servlet-name>SecondServlet</servlet-name>
        <url-pattern>/welcome</url-pattern>
</servlet-mapping>
<welcome-file-list>
        <welcome-file>index.html</welcome-file>
</welcome-file-list>
</web-app>
```

FirstServlet.java

```
import java.io.*;
import javax.servlet.*;
```

```
import javax.servlet.http.*;

public class FirstServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response){
                try{

                response.setContentType("text/html");
                PrintWriter out = response.getWriter();

                String n=request.getParameter("uname");
                out.print("Welcome "+n);

                //creating form that have invisible textfield
                out.print("<form action='welcome'>");
                out.print("<input type='hidden' name='uname' value='"+n+"'>");
                out.print("<input type='submit' value='go'>");
                out.print("</form>");

                out.close();

            }catch(Exception e){System.out.println(e);}
        }
}
```

SecondServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SecondServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response){
                try{

                response.setContentType("text/html");
                PrintWriter out = response.getWriter();

                //Getting the value from the hidden field
                String n=request.getParameter("uname");
                out.print("Hello "+n);

                out.close();

            }catch(Exception e){System.out.println(e);}
        }
}
```

Unit-V
Servlet & JSP

Unit	Topic Name	Topics	Marks
5	**Servlet**	Basic Concepts of Web & Web Servers, Introduction to Servlet, Servlet Life Cycle, Types of Servlet, Servlet API, Sessions, Cookies, Session Management.	**24**
	JSP	Introduction to JSP, JSP life cycle, JSP tags: Declaration, Expression, Scriptlet, JSP Directives, Implicit Objects, Action Elements, EL, JSTL, Custom Tags.	
	RMI & EJB	Introduction to RMI, Introduction to EJB.	

A. Introduction to JSP

JSP technology is used to create web application just like Servlet technology. It can be thought of as an extension to servlet because it provides more functionality than servlet such as expression language, jstl etc. A JSP page consists of HTML tags and JSP tags. The jsp pages are easier to maintain than servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tag etc. Java Server Pages (JSP) is a technology for developing web pages that support dynamic content which helps developers insert java code in HTML pages by making use of special JSP tags, most of which **start with <% and end with %>.** A Java Server Pages component is a type of Java servlet that is designed to fulfill the role of a user interface for a Java web application. Web developers write JSPs as text files that combine HTML or XHTML code, XML elements, and embedded JSP actions and commands.

Importance of JSP

- Performance is significantly better because JSP allows embedding Dynamic Elements in HTML Pages itself instead of having a separate CGI files.
- JSP are always compiled before it's processed by the server unlike CGI/Perl which requires the server to load an interpreter and the target script each time the page is requested.
- Java Server Pages are built on top of the Java Servlets API, so like Servlets; JSP also has access to all the powerful Enterprise Java APIs, including JDBC, JNDI, EJB, JAXP etc.
- JSP pages can be used in combination with servlets that handle the business logic, the model supported by Java servlet template engines.

Advantages of JSP

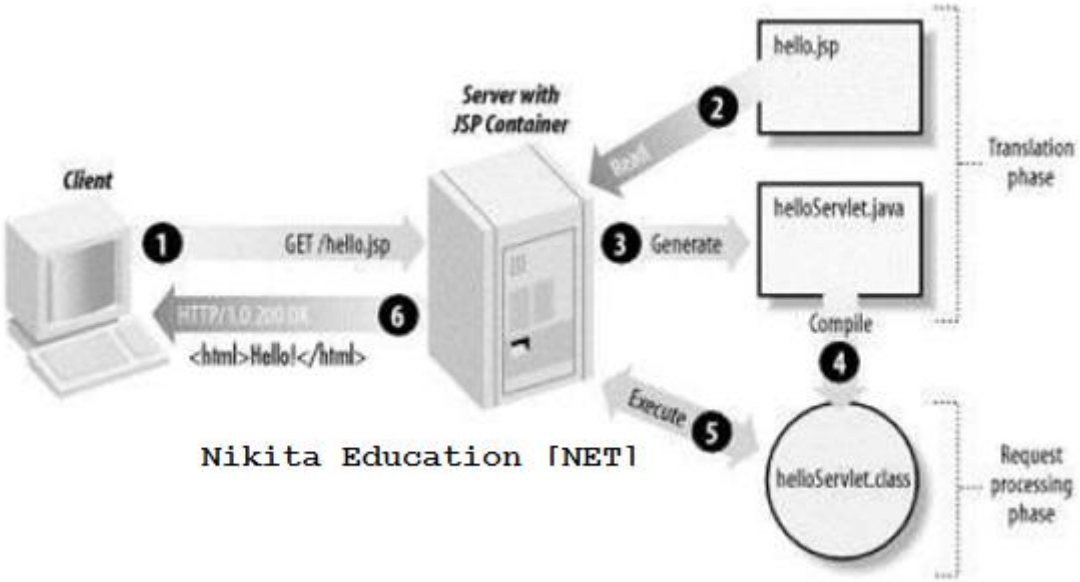
- **vs. Active Server Pages (ASP):** The advantages of JSP are twofold. First, the dynamic part is written in Java, not Visual Basic or other MS specific language, so it is more powerful and easier to use. Second, it is portable to other operating systems and non-Microsoft Web servers.
- **vs. Pure Servlets:** It is more convenient to write (and to modify!) regular HTML than to have plenty of println statements that generate the HTML.
- **vs. Server-Side Includes (SSI):** SSI is really only intended for simple inclusions, not for "real" programs that use form data, make database connections, and the like.
- **vs. JavaScript:** JavaScript can generate HTML dynamically on the client but can hardly interact with the web server to perform complex tasks like database access and image processing etc.
- **vs. Static HTML:** Regular HTML, of course, cannot contain dynamic information.
- **Extension to Servlet:** JSP technology is the extension to servlet technology. We can use all the features of servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.
- **Easy to maintain:** JSP can be easily managed because we can easily separate our business logic with presentation logic. In servlet technology, we mix our business logic with the presentation logic.
- **Fast Development (No need to recompile and redeploy):** If JSP page is modified, we don't need to recompile and redeploy the project. The servlet code needs to be updated and recompiled if we have to change the look and feel of the application.
- **Less code than Servlet:** In JSP, we can use a lot of tags such as action tags, jstl, custom tags etc. that reduces the code. Moreover, we can use EL, implicit objects etc.

B. JSP Architecture

The web server needs a JSP engine i.e. container to process JSP pages. The JSP container is responsible for intercepting requests for JSP pages. This tutorial makes use of Apache which has built-in JSP container to support JSP pages development. A JSP container works with the Web server to provide the runtime environment and other services a JSP needs. It knows how to understand the special elements that are part of JSPs. Following diagram shows the position of JSP container and JSP files in a Web Application.

JSP Processing:

- As with a normal page, your browser sends an HTTP request to the web server.
- The web server recognizes that the HTTP request is for a JSP page and forwards it to a JSP engine. This is done by using the URL or JSP page which ends with **.jsp** instead of .html.
- The JSP engine loads the JSP page from disk and converts it into servlet content. This conversion is very simple in which all template text is converted to println( ) statements and all JSP elements are converted to Java code that implements the corresponding dynamic behavior of the page.
- The JSP engine compiles the servlet into an executable class and forwards the original request to a servlet engine.
- A part of the web server called the servlet engine loads the Servlet class and executes it. During execution, the servlet produces an output in HTML format, which the servlet engine passes to the web server inside an HTTP response.
- The web server forwards the HTTP response to your browser in terms of static HTML content.
- Finally web browser handles the dynamically generated HTML page inside the HTTP response exactly as if it were a static page.



Typically the JSP engine checks to see whether a servlet for a JSP file already exists and modification date on the JSP is older than the servlet. If the JSP is older than its generated servlet, the JSP container assumes that the JSP hasn't changed and that the generated servlet still matches the JSP's contents. This makes the process more efficient than with other scripting languages (such as PHP) and therefore faster. So in a way, a JSP page is really just another way to write a servlet without having to be a Java programming wiz. Except for the translation phase, a JSP page is handled exactly like a regular servlet.

C. JSP Life Cycle

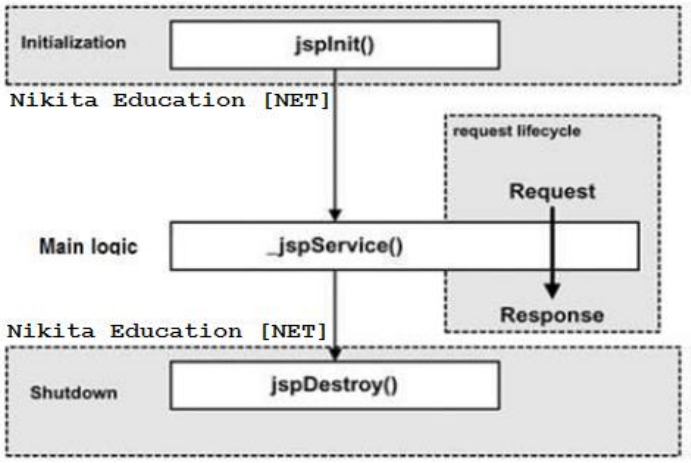
The key to understanding the low-level functionality of JSP is to understand the simple life cycle they follow. A JSP life cycle can be defined as the entire process from its creation till the destruction which is similar to a servlet life cycle with an additional step which is required to compile a JSP into servlet. The following are the paths followed by a JSP

- Compilation
- Initialization
- Execution
- Cleanup

JSP Compilation

When a browser asks for a JSP, the JSP engine first checks to see whether it needs to compile the page. If the page has never been compiled, or if the JSP has been modified since it was last compiled, the JSP engine compiles the page. The compilation process involves three steps:

- Parsing the JSP.
- Turning the JSP into a servlet.
- Compiling the servlet.



JSP Initialization

When a container loads a JSP it invokes the jspInit() method before servicing any requests. If you need to perform JSP-specific initialization, override the jspInit() method:

```
public void jspInit()
{
        // Initialization code...
}
```

Typically initialization is performed only once and as with the servlet init method, you generally initialize database connections, open files, and create lookup tables in the jspInit method.

JSP Execution

This phase of the JSP life cycle represents all interactions with requests until the JSP is destroyed. Whenever a browser requests a JSP and the page has been loaded and initialized, the JSP engine invokes the **_jspService()** method in the JSP. The _jspService() method takes an **HttpServletRequest** and an **HttpServletResponse** as its parameters as follows:

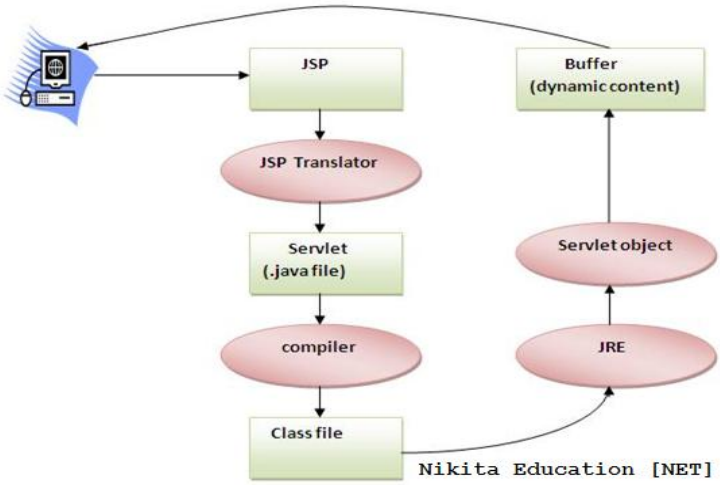
```
void _jspService(HttpServletRequest request, HttpServletResponse response)
{
        // Service handling code...
}
```

The _jspService() method of a JSP is invoked once per a request and is responsible for generating the response for that request and this method is also responsible for generating responses to all seven of the HTTP methods ie. GET, POST, DELETE etc.

JSP Cleanup

The destruction phase of the JSP life cycle represents when a JSP is being removed from use by a container. The **jspDestroy()** method is the JSP equivalent of the destroy method for servlets. Override jspDestroy when you need to perform any cleanup, such as releasing database connections or closing open files. The jspDestroy() method has the following form:

```
public void jspDestroy()
{
        // Your cleanup code goes here.
}
```



D. Request - Response Cycle

Web flow starts from a request being made by user's browser, the request is made as http request so that server can understand it. Based on request the server searches for an appropriate resource and sends it back to client in form of http response.

1. Http Request: A http request basically have three major components.

1. HTTP method, there are 7 methods defined in java servlets but most of the time you will see either a get or post method. We will get to know about these methods and their usage in later part of this blog.
2. The requested page URL, the page to access like www.google.com.
3. Parameters, parameters(as id, name, email.. etc.) are being send as part of request on which the response is being generated.

2. Http Response: A http request basically have three major components.

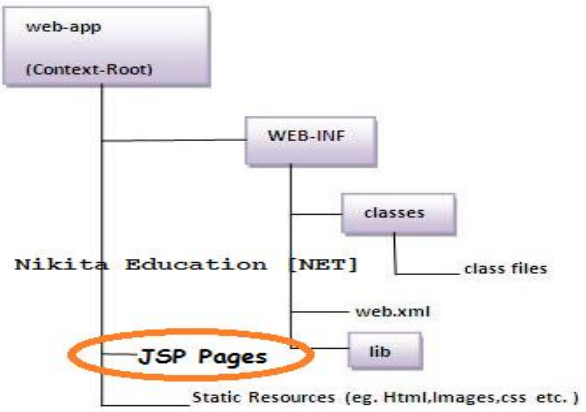
1. A status code, this code tells the browser whether the request is successful or not.
2. Content type, it tells the browser about the type of content that response web page contains in it (text, picture, html...etc).
3. The content, the important and last information that is the served resource that the user was requested.

E. Simple JSP Page

To create the first jsp page, write some html code as given below, and save it by .jsp extension. We have save this file as index.jsp. Put it in a folder and paste the folder in the web-apps directory in apache tomcat to run the jsp page.

index.jsp

<html> <body> <% out.print(2*5); %> </body> </html> **Output:** 10 [it will print on web page inside web browser]	**How to run a simple JSP Page?** • Start the apache tomcat server • put the jsp file in a folder and deploy on the server • visit the browser by the url : http://localhost:portno/contextRoot/jspfile • e.g. http://localhost:8080/myapplication/index.jsp



F. JSP API

The javax.servlet.jsp package

Interfaces	Classes	
1. JspPage 2. HttpJspPage	• JspWriter • PageContext • JspFactory	• JspEngineInfo • JspException • JspError

1. JspPage

According to the JSP specification, all the generated servlet classes must implement the JspPage interface. It extends the Servlet interface. It provides two life cycle methods.

Methods of JspPage interface

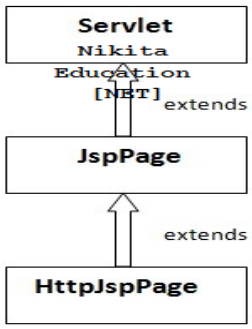
1. **public void jspInit():** It is invoked only once during the life cycle of the JSP when JSP page is requested firstly. It is used to perform initialization. It is same as the init() method of Servlet interface.
2. **public void jspDestroy():** It is invoked only once during the life cycle of the JSP before the JSP page is destroyed. It can be used to perform some clean up operation.

2. HttpJspPage

The HttpJspPage interface provides the one life cycle method of JSP. It extends the JspPage interface.

Method of HttpJspPage interface:

1. **public void _jspService():** It is invoked each time when request for the JSP page comes to the container. It is used to process the request. The underscore _ signifies that you cannot override this method.



G. JSP Elements & Syntax

In JSP, java code can be written inside the jsp page using different JSP elements called as **scripting elements**. Following are the three different scripting elements of jsp:

- scriptlet tag
- expression tag
- declaration tag

JSP scriptlet tag

A scriptlet tag is used to execute java source code in JSP. Syntax is as follows:

```
<% java source code %>
```

Example

```
<html>
        <body>
                <% out.print("welcome to jsp"); %>
        </body>
</html>
```

Example

In this example, we have created two files index.html and welcome.jsp. The index.html file gets the username from the user and the welcome.jsp file prints the username with the welcome message.

index.html

```
<html>
    <body>
        <form action="welcome.jsp">
            <input type="text" name="uname">
            <input type="submit" value="go"><br/>
        </form>
    </body>
</html>
```

welcome.jsp

```
<html>
    <body>
        <%
            String name=request.getParameter("uname");
            out.print("welcome "+name);
        %>
    </body>
</html>
```

JSP expression tag

The code placed within expression tag is written to the output stream of the response. So you need not write out.print() to write data. It is mainly used to print the values of variable or method.

```
<%= statement %>
```

Example

```
<html>
    <body>
        <%= "welcome to jsp" %>
    </body>
</html>
```

Note: Do not end your statement with semicolon in case of expression tag.

Example

To display the current time, we have used the getTime() method of Calendar class. The getTime() is an instance method of Calendar class, so we have called it after getting the instance of Calendar class by the getInstance() method.

index.jsp

```
<html>
    <body>
        Current Time: <%= java.util.Calendar.getInstance().getTime() %>
    </body>
</html>
```

In this example, we are printing the username using the expression tag. The index.html file gets the username and sends the request to the welcome.jsp file, which displays the username.

index.html

```
<html>
    <body>
        <form action="welcome.jsp">
            <input type="text" name="uname"><br/>
            <input type="submit" value="go">
        </form>
```

```
    </body>
</html>
```

welcome.jsp

```
<html>
    <body>
        <%= "Welcome "+request.getParameter("uname") %>
    </body>
</html>
```

JSP Declaration Tag

The JSP declaration tag is used to declare fields and methods. The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet. So it doesn't get memory at each request.

```
<%!  field or method declaration %>
```

Scriplet vs. Declaration tag

Jsp Scriptlet Tag	Jsp Declaration Tag
The jsp scriptlet tag can only declare variables not methods.	The jsp declaration tag can declare variables as well as methods.
The declaration of scriptlet tag is placed inside the _jspService() method.	The declaration of jsp declaration tag is placed outside the _jspService() method.

Example

In this example of JSP declaration tag, we are declaring the field and printing the value of the declared field using the jsp expression tag.

index.jsp

```
<html>
    <body>

        <%! int data=50; %>

        <%= "Value of the variable is:"+data %>

    </body>
</html>
```

Example

In this example of JSP declaration tag, we are defining the method which returns the cube of given number and calling this method from the jsp expression tag. But we can also use jsp scriptlet tag to call the declared method.

index.jsp

```
<html>
    <body>
        <%!
            int cube(int n){
                return n*n*n*;
            }
        %>

        <%= "Cube of 3 is:"+cube(3) %>
    </body>
</html>
```

H. JSP Implicit Objects

There are **9 jsp implicit objects**. These objects are *created by the web containers* that are available to all the jsp pages.

Object	Class
out	JspWriter
request	HttpServletRequest
response	HttpServletResponse
config	ServletConfig
application	ServletContext
session	HttpSession
pageContext	PageContext
page	Object
exception	Throwable

H.1. out

For writing any data to the buffer, JSP provides an implicit object named out. It is the object of JspWriter. In case of servlet you need to write:

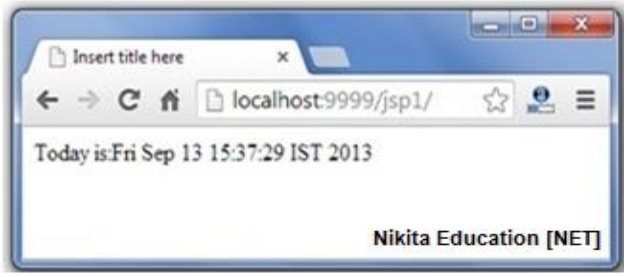
1. PrintWriter out=response.getWriter();

But in JSP, you don't need to write this code.

Example:

index.jsp

```
<html>
<body>
<% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
</body>
</html>
```



H.2. request

The **JSP request** is an implicit object of type HttpServletRequest i.e. created for each jsp request by the web container. It can be used to get request information such as parameter, header information, remote address, server name, server port, content type, character encoding etc. It can also be used to set, get and remove attributes from the jsp request scope. Let's see the simple example of request implicit object where we are printing the name of the user with welcome message.

Example

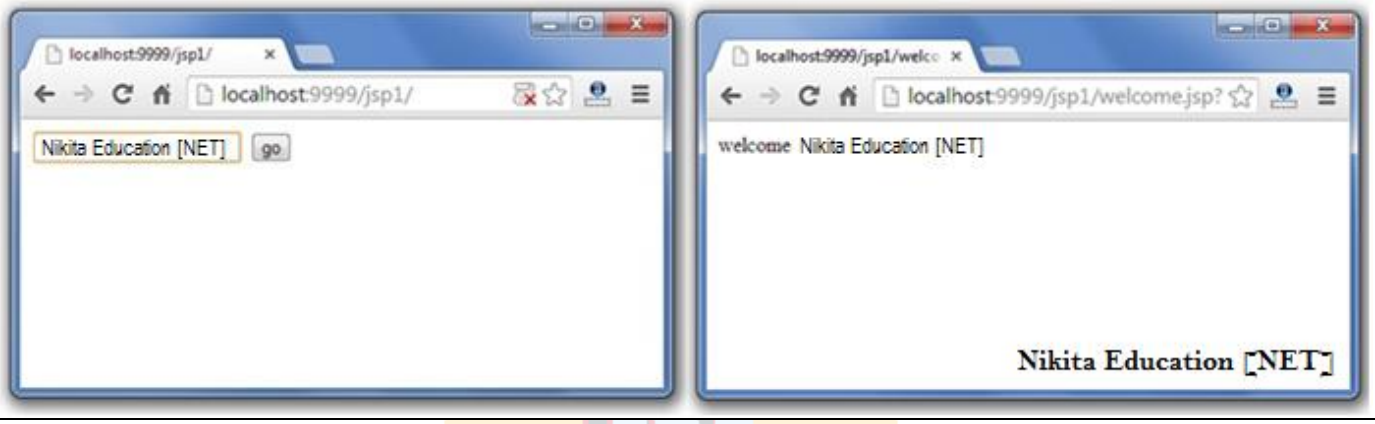
index.html

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

welcome.jsp

```
<%
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
```

Output



H.3. response

In JSP, response is an implicit object of type HttpServletResponse. The instance of HttpServletResponse is created by the web container for each jsp request. It can be used to add or manipulate response such as redirect response to another resource, send error etc. Let's see the example of response implicit object where we are redirecting the response to the Google.

Example

index.html

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

welcome.jsp

```
<%
response.sendRedirect("http://www.google.com");
%>
```

Output

H.4. config

In JSP, config is an implicit object of type *ServletConfig*. This object can be used to get initialization parameter for a particular JSP page. The config object is created by the web container for each jsp page. Generally, it is used to get initialization parameter from the web.xml file.

Example

index.html

```
<form action="welcome">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

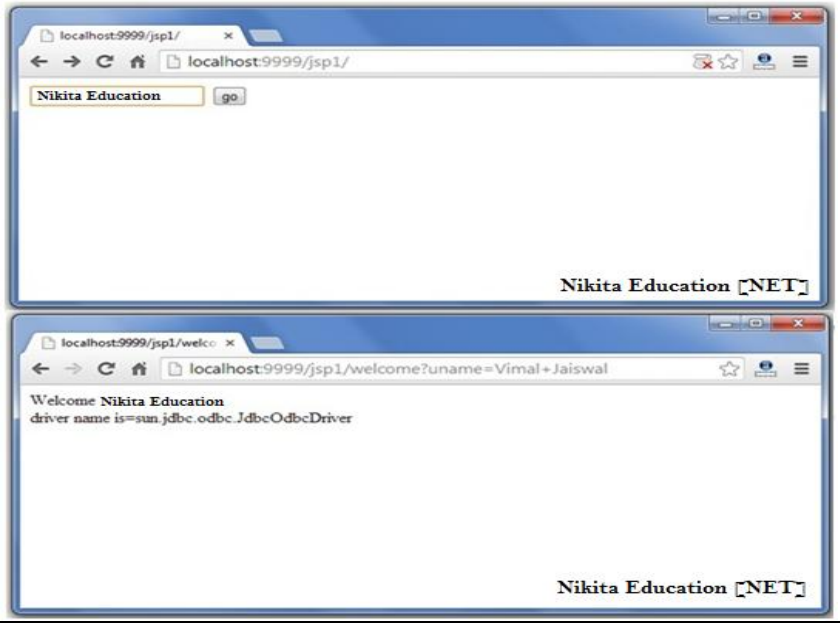
web.xml

```
<web-app>

<servlet>
<servlet-name>sonoojaiswal</servlet-name>
<jsp-file>/welcome.jsp</jsp-file>

<init-param>
<param-name>dname</param-name>
<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
</init-param>

</servlet>

<servlet-mapping>
<servlet-name>sonoojaiswal</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>

</web-app>
```

welcome.jsp

```
<%
out.print("Welcome "+request.getParameter("uname"));

String driver=config.getInitParameter("dname");
out.print("driver name is="+driver);
%>
```

Output



H.5. application

In JSP, application is an implicit object of type *ServletContext*. The instance of ServletContext is created only once by the web container when application or project is deployed on the server. This object can be used to get initialization parameter from configuration file (web.xml). It can also be used to get, set or remove attribute from the application scope. This initialization parameter can be used by all jsp pages.

Example

index.html

```
<form action="welcome">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

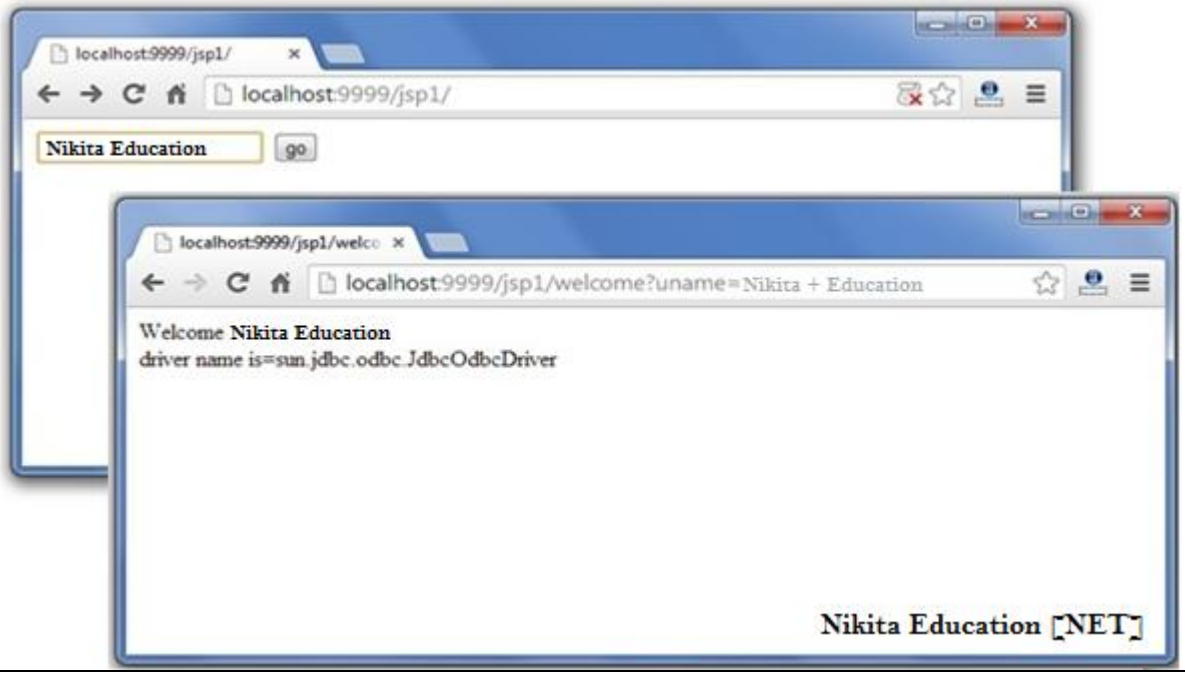
web.xml

```
<web-app>
<servlet>
<servlet-name>sonoojaiswal</servlet-name>
<jsp-file>/welcome.jsp</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>sonoojaiswal</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>
<context-param>
<param-name>dname</param-name>
<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
</context-param>
</web-app>
```

welcome.jsp

```
<%
out.print("Welcome "+request.getParameter("uname"));
String driver=application.getInitParameter("dname");
out.print("driver name is="+driver);
%>
```

Output



H.6. session

In JSP, session is an implicit object of type HttpSession. The Java developer can use this object to set, get or remove attribute or to get session information.

Example

index.html

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
</body>
</html>
```

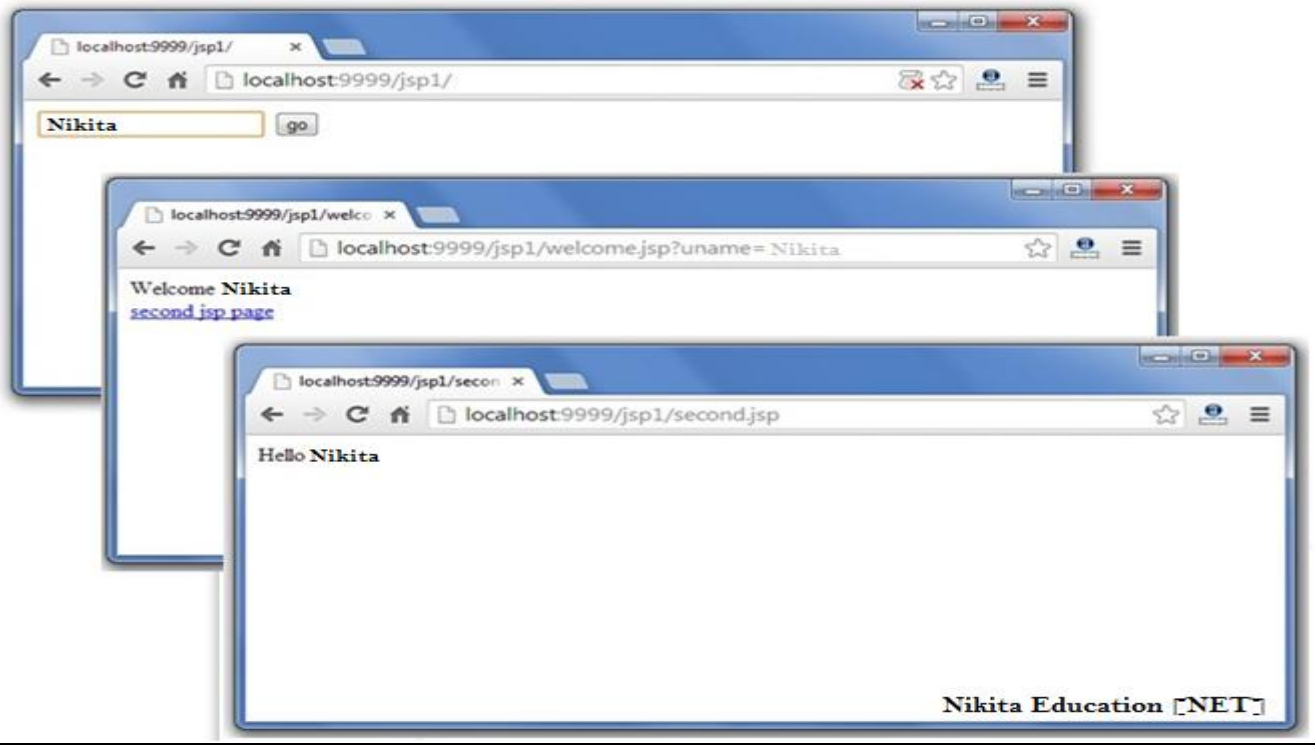
welcome.jsp

```
<html>
<body>
<%
String name=request.getParameter("uname");
out.print("Welcome "+name);
session.setAttribute("user",name);
<a href="second.jsp">second jsp page</a>
%>
</body>
</html>
```

second.jsp

```
<html>
<body>
<%
String name=(String)session.getAttribute("user");
out.print("Hello "+name);
%>
</body>
</html>
```

Output



H.7. pageContext

In JSP, pageContext is an implicit object of type PageContext class. The pageContext object can be used to set, get or remove attribute from one of the following scopes:

- page (default scope of jsp)
- request
- session
- application

Example

index.html

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
</body>
</html>
```

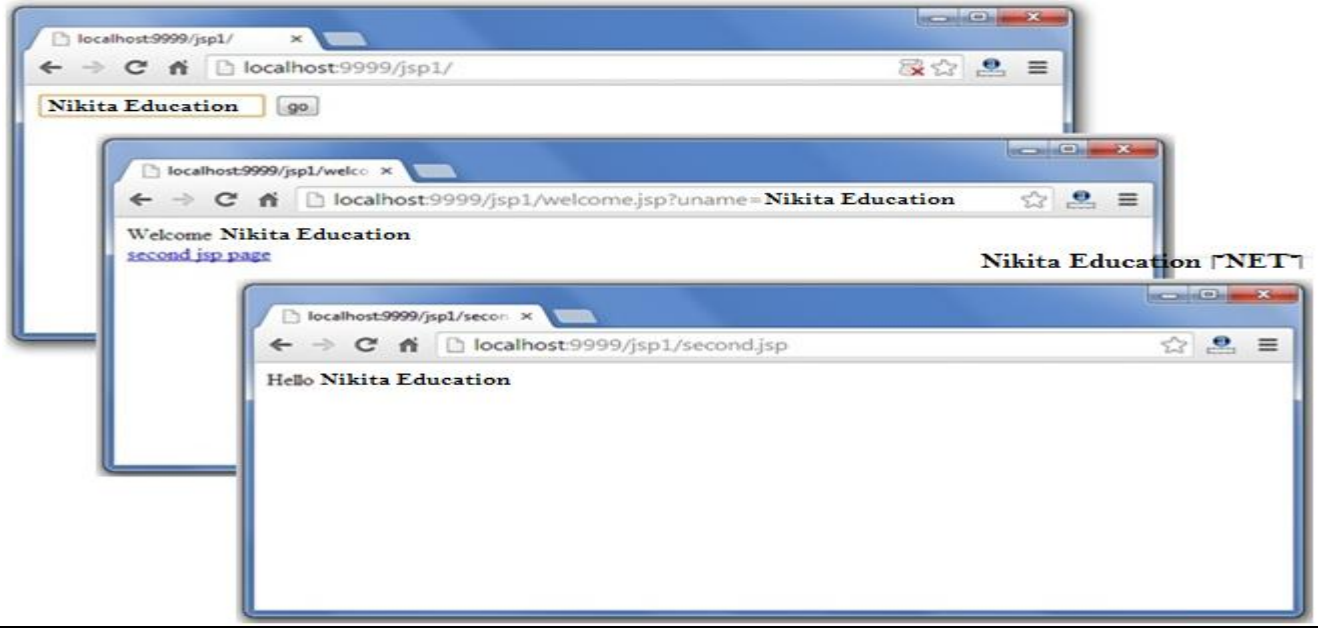
welcome.jsp

```
<html>
<body>
<%
String name=request.getParameter("uname");
out.print("Welcome "+name);
pageContext.setAttribute("user",name,PageContext.SESSION_SCOPE);
<a href="second.jsp">second jsp page</a>
%>
</body>
</html>
```

second.jsp

```
<html>
<body>
<%
String name=(String)pageContext.getAttribute("user",PageContext.SESSION_SCOPE);
out.print("Hello "+name);
%>
</body>
</html>
```

Output



H.8. page

In JSP, page is an implicit object of type Object class. This object is assigned to the reference of auto generated servlet class. It is written as:

Object page=this;

For using this object it must be cast to Servlet type. For example:

<% (HttpServlet)page.log("message"); %>

Since, it is of type Object it is less used because you can use this object directly in jsp.For example:

<% this.log("message"); %>

H.9. exception

In JSP, exception is an implicit object of type java.lang.Throwable class. This object can be used to print the exception. But it can only be used in error pages. It is better to learn it after page directive.

Example
error.jsp
<%@ page isErrorPage="true" %> <html> <body> Sorry following exception occured:<%= exception %> </body> </html>

I. JSP Directives

The **jsp directives** are messages that tell the web container how to translate a JSP page into the corresponding servlet. There are three types of directives:

- page directive
- include directive
- taglib directive

Syntax: **<%@ directive attribute="value" %>**

I.1. page

The page directive defines attributes that apply to an entire JSP page.

Syntax: **<%@ page attribute="value" %>**

Attributes:		
import	info	isELIgnored
contentType	buffer	isThreadSafe
extends	language	autoFlush
session	pageEncoding	errorPage
isErrorPage		

import

The import attribute is used to import class,interface or all the members of a package.It is similar to import keyword in java class or interface.

Example

```
<html>
<body>

<%@ page import="java.util.Date" %>
Today is: <%= new Date() %>

</body>
</html>
```

contentType

The contentType attribute defines the MIME(Multipurpose Internet Mail Extension) type of the HTTP response.The default value is "text/html;charset=ISO-8859-1".

Example

```
<html>
<body>

<%@ page contentType=application/msword %>
Today is: <%= new java.util.Date() %>

</body>
</html>
```

extends

The extends attribute defines the parent class that will be inherited by the generated servlet. It is rarely used.

info

This attribute simply sets the information of the JSP page which is retrieved later by using getServletInfo() method of Servlet interface.

Example

```
<html>
<body>

<%@ page info="composed by Sonoo Jaiswal" %>
Today is: <%= new java.util.Date() %>

</body>
</html>
```

The web container will create a method getServletInfo() in the resulting servlet.

```
public String getServletInfo() {
        return "composed by Sonoo Jaiswal";
}
```

buffer

The buffer attribute sets the buffer size in kilobytes to handle output generated by the JSP page.The default size of the buffer is 8Kb.

Example

```
<html>
<body>
```

```
<%@ page buffer="16kb" %>
Today is: <%= new java.util.Date() %>

</body>
</html>
```

language

The language attribute specifies the scripting language used in the JSP page. The default value is "java".

isELIgnored

We can ignore the Expression Language (EL) in jsp by the isELIgnored attribute. By default its value is false i.e. Expression Language is enabled by default. We see Expression Language later.

1. <%@ page isELIgnored="true" %>//Now EL will be ignored

isThreadSafe

Servlet and JSP both are multithreaded.If you want to control this behaviour of JSP page, you can use isThreadSafe attribute of page directive.The value of isThreadSafe value is true.If you make it false, the web container will serialize the multiple requests, i.e. it will wait until the JSP finishes responding to a request before passing another request to it.If you make the value of isThreadSafe attribute like:

<%@ page isThreadSafe="false" %>

The web container in such a case, will generate the servlet as:

```
1. public class SimplePage_jsp extends HttpJspBase
2.   implements SingleThreadModel{
3.         .......
4. }
```

errorPage

The errorPage attribute is used to define the error page, if exception occurs in the current page, it will be redirected to the error page.

Example: index.jsp

```
<body>

<%@ page errorPage="myerrorpage.jsp" %>
<%= 100/0 %>

</body>
</html>
```

isErrorPage

The isErrorPage attribute is used to declare that the current page is the error page. ***Note: The exception object can only be used in the error page.***

Example: myerrorpage.jsp

```
<html>
<body>
<%@ page isErrorPage="true" %>
Sorry an exception occured!<br/>
The exception is: <%= exception %>
</body>
</html>
```

I.2. include

The include directive is used to include the contents of any resource it may be jsp file, html file or text file. The include directive includes the original content of the included resource at page translation time (the jsp page is translated only once so it will be better to include static resource). **Advantage** is Code Reusability.

Syntax: **<%@ include file="resourceName" %>**

Example

In this example, we are including the content of the header.html file. To run this example you must create an header.html file.

```
<html>
<body>

<%@ include file="header.html" %>
Today is: <%= java.util.Calendar.getInstance().getTime() %>

</body>
</html>
```

I.3. taglib

The JSP taglib directive is used to define a tag library that defines many tags. We use the TLD (Tag Library Descriptor) file to define the tags. In the custom tag section we will use this tag so it will be better to learn it in custom tag.

Syntax: **<%@ taglib uri="uriofthetaglibrary" prefix="prefixoftaglibrary" %>**

Example

In this example, we are using our tag named currentDate. To use this tag we must specify the taglib directive so the container may get information about the tag.

```
<html>
<body>

<%@ taglib uri="http://www.javatpoint.com/tags" prefix="mytag" %>

<mytag:currentDate/>

</body>
</html>
```

J. Exception handling in JSP

The exception is normally an object that is thrown at runtime. Exception Handling is the process to handle the runtime errors. There may occur exception any time in your web application. So handling exceptions is a safer side for the web developer. In JSP, there are two ways to perform exception handling:

1. By **errorPage** and **isErrorPage** attributes of page directive
2. By **<error-page>** element in web.xml file

Example: Using errorPage

In this case, you must define and create a page to handle the exceptions, as in the error.jsp page. The pages where may occur exception, define the errorPage attribute of page directive, as in the process.jsp page.

There are 3 files:

- index.jsp for input values

- process.jsp for dividing the two numbers and displaying the result
- error.jsp for handling the exception

index.jsp

```
<form action="process.jsp">
No1:<input type="text" name="n1" /><br/><br/>
No1:<input type="text" name="n2" /><br/><br/>
<input type="submit" value="divide"/>
</form>
```

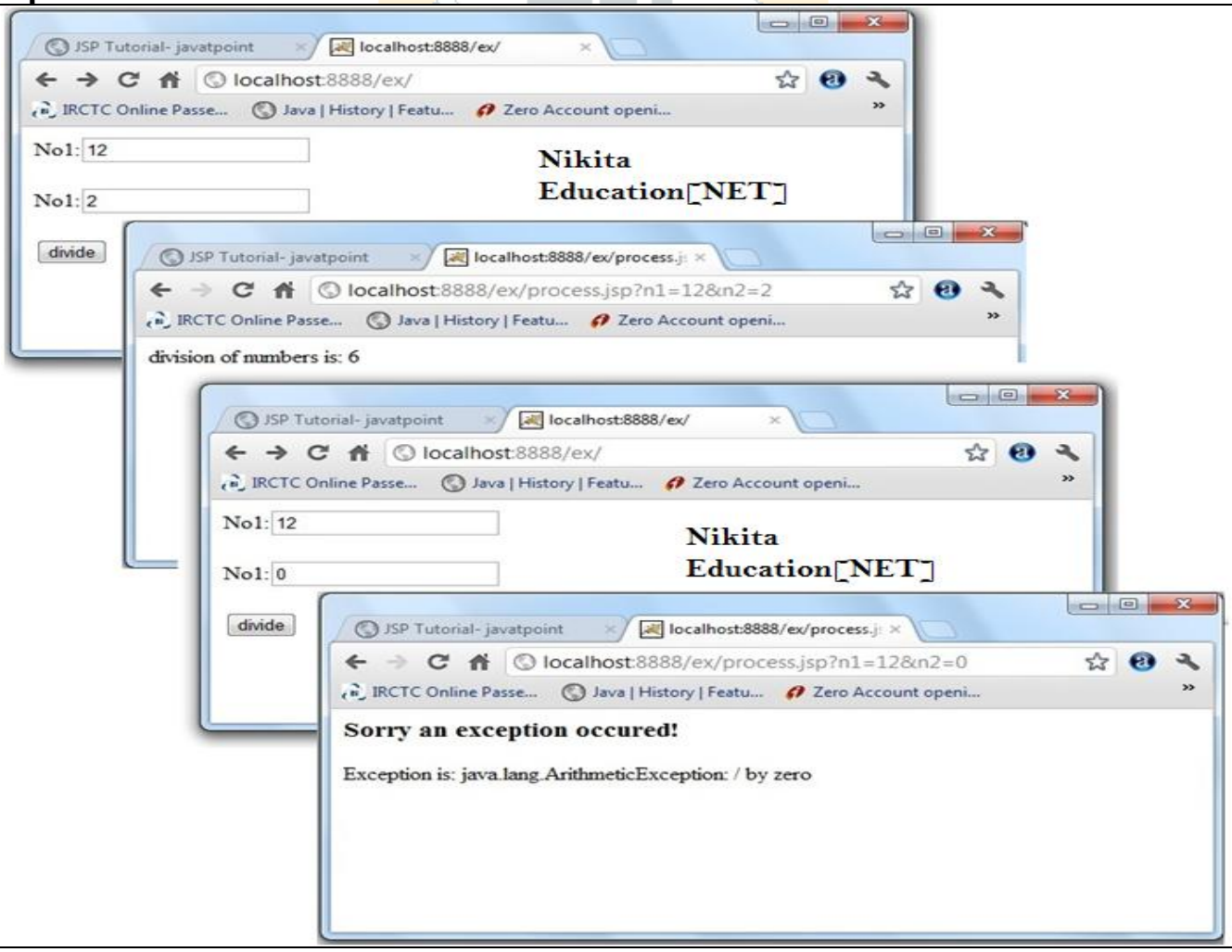
process.jsp

```
<%@ page errorPage="error.jsp" %>
<%

String num1=request.getParameter("n1");
String num2=request.getParameter("n2");

int a=Integer.parseInt(num1);
int b=Integer.parseInt(num2);
int c=a/b;
out.print("division of numbers is: "+c);

%>
```

error.jsp

```
<%@ page isErrorPage="true" %>

<h3>Sorry an exception occured!</h3>

Exception is: <%= exception %>
```

Output



Example: error-page element in web.xml

This approach is better because you don't need to specify the errorPage attribute in each jsp page. Specifying the single entry in the web.xml file will handle the exception. In this case, either specify exception-type or error-code with the location element. If you want to handle all the exception, you will have to specify the java.lang.Exception in the exception-type element.

- web.xml file for specifying the error-page element
- index.jsp for input values
- process.jsp for dividing the two numbers and displaying the result
- error.jsp for displaying the exception

web.xml

```
<web-app>

 <error-page>
  <exception-type>java.lang.Exception</exception-type>
  <location>/error.jsp</location>
  </error-page>

</web-app>
```

This approach is better if you want to handle any exception. If you know any specific error code and you want to handle that exception, specify the error-code element instead of exception-type as given below:

```
<web-app>

 <error-page>
  <error-code>500</error-code>
  <location>/error.jsp</location>
  </error-page>

</web-app>
```

index.jsp

```
<form action="process.jsp">
No1:<input type="text" name="n1" /><br/><br/>
No1:<input type="text" name="n2" /><br/><br/>
<input type="submit" value="divide"/>
</form>
```

process.jsp

```
<%@ page errorPage="error.jsp" %>
<%

String num1=request.getParameter("n1");
String num2=request.getParameter("n2");

int a=Integer.parseInt(num1);
int b=Integer.parseInt(num2);
int c=a/b;
out.print("division of numbers is: "+c);

%>
```

error.jsp

```
<%@ page isErrorPage="true" %>

<h3>Sorry an exception occured!</h3>

Exception is: <%= exception %>
```

K. Action Tags

There are many JSP action tags or elements. Each JSP action tag is used to perform some specific tasks. The action tags are used to control the flow between pages and to use Java Bean. The Jsp action tags are given below.

JSP Action Tags	Description
jsp:forward	forwards the request and response to another resource.
jsp:include	includes another resource.
jsp:useBean	creates or locates bean object.
jsp:setProperty	sets the value of property in bean object.
jsp:getProperty	prints the value of property of the bean.
jsp:plugin	embeds another components such as applet.
jsp:param	sets the parameter value. It is used in forward and include mostly.
jsp:fallback	can be used to print the message if plugin is working. It is used in jsp:plugin.

The jsp:useBean, jsp:setProperty and jsp:getProperty tags are used for bean development.

K.1. jsp:forward

The jsp:forward action tag is used to forward the request to another resource it may be jsp, html or another resource.

Syntax of jsp:forward action tag without parameter

```
<jsp:forward page="relativeURL | <%= expression %>" />
```

Syntax of jsp:forward action tag with parameter

```
<jsp:forward page="relativeURL | <%= expression %>">
<jsp:param name="parametername" value="parametervalue | <%=expression%>" />
</jsp:forward>
```

Example: without parameter

index.jsp

```
<html>
<body>
<h2>this is index page</h2>

<jsp:forward page="printdate.jsp" />
</body>
</html>
```

printdate.jsp

```
<html>
<body>
<% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
</body>
</html>
```

Example: with parameter

index.jsp

```
<html>
<body>
<h2>this is index page</h2>
```

```
<jsp:forward page="printdate.jsp" >
<jsp:param name="name" value="javatpoint.com" />
</jsp:forward>

</body>
</html>
```

printdate.jsp

```
<html>
<body>

<% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
<%= request.getParameter("name") %>

</body>
</html>
```

K.2. jsp:include

The **jsp:include action tag** is used to include the content of another resource it may be jsp, html or servlet. The jsp include action tag includes the resource at request time so it is **better for dynamic pages** because there might be changes in future. The jsp:include tag can be used to include static as well as dynamic pages. **Code reusability** : We can use a page many times such as including header and footer pages in all pages. So it saves a lot of time.

JSP include directive	JSP include action
includes resource at translation time.	includes resource at request time.
better for static pages.	better for dynamic pages.
includes the original content in the generated servlet.	calls the include method.

Syntax of jsp:include action tag without parameter

1. <jsp:include page="relativeURL | <%= expression %>" />

Syntax of jsp:include action tag with parameter

1. <jsp:include page="relativeURL | <%= expression %>">
2. <jsp:param name="parametername" value="parametervalue | <%=expression%>" />
3. </jsp:include>

Example: without parameter

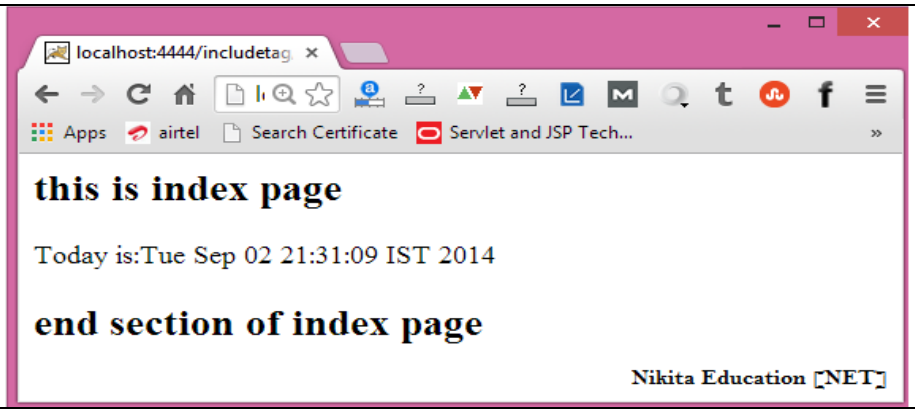
index.jsp

```
<h2>this is index page</h2>
<jsp:include page="printdate.jsp" />
<h2>end section of index page</h2>
```

printdate.jsp

```
<% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
```

Output



L. javaBean

A Java Bean is a java class that should follow following conventions:

- It should have a no-arg constructor.
- It should be Serializable.
- It should provide methods to set and get the values of the properties, known as getter and setter methods.

Why use Java Bean?

According to Java white paper, it is a reusable software component. A bean encapsulates many objects into one object, so we can access this object from multiple places. Moreover, it provides the easy maintenance.

Simple example of java bean class

```
//Employee.java

package mypack;
public class Employee implements java.io.Serializable{
private int id;
private String name;

public Employee(){}

public void setId(int id){this.id=id;}

public int getId(){return id;}

public void setName(String name){this.name=name;}

public String getName(){return name;}

}
```

How to access the java bean class?

To access the java bean class, we should use getter and setter methods.

```
package mypack;
public class Test{
public static void main(String args[]){

Employee e=new Employee();//object is created

e.setName("Arjun");//setting value to the object

System.out.println(e.getName());
}
}
```

Note: There are two ways to provide values to the object, one way is by constructor and second is by setter method.

jsp:useBean

The jsp:useBean action tag is used to locate or instantiate a bean class. If bean object of the Bean class is already created, it doesn't create the bean depending on the scope. But if object of bean is not created, it instantiates the bean.

Syntax of jsp:useBean action tag

1. <jsp:useBean id= "instanceName" scope= "page | request | session | application"
2. **class**= "packageName.className" type= "packageName.className"
3. beanName="packageName.className | <%= expression >" >
4. </jsp:useBean>

Attributes and Usage of jsp:useBean action tag

1. **id:** is used to identify the bean in the specified scope.
2. **scope:** represents the scope of the bean. It may be page, request, session or application. The default scope is page.
   - **page:** specifies that you can use this bean within the JSP page. The default scope is page.
   - **request:** specifies that you can use this bean from any JSP page that processes the same request. It has wider scope than page.
   - **session:** specifies that you can use this bean from any JSP page in the same session whether processes the same request or not. It has wider scope than request.
   - **application:** specifies that you can use this bean from any JSP page in the same application. It has wider scope than session.
3. **class:** instantiates the specified bean class (i.e. creates an object of the bean class) but it must have no-arg or no constructor and must not be abstract.
4. **type:** provides the bean a data type if the bean already exists in the scope. It is mainly used with class or beanName attribute. If you use it without class or beanName, no bean is instantiated.
5. **beanName:** instantiates the bean using the java.beans.Beans.instantiate() method.

Example

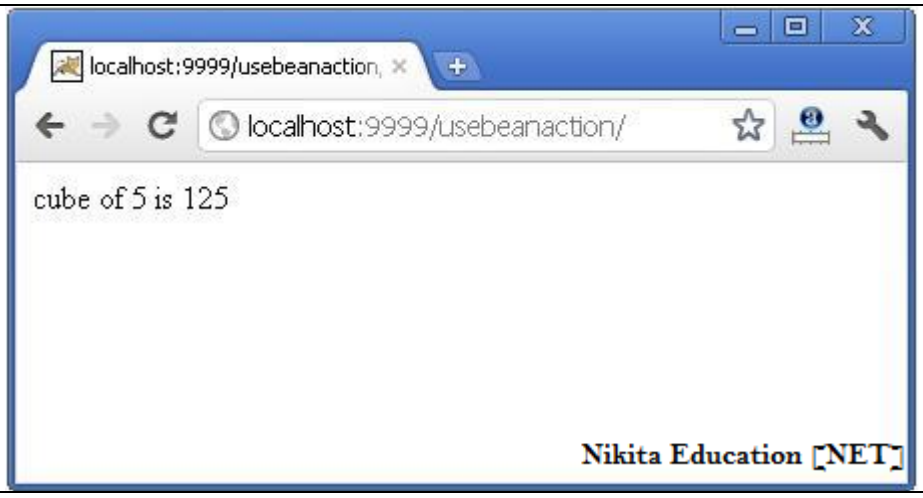
Calculator.java (a simple Bean class)

```
package com.javatpoint;
public class Calculator{

public int cube(int n){return n*n*n;}

}
```

index.jsp

```
<jsp:useBean id="obj" class="com.javatpoint.Calculator"/>

<%
int m=obj.cube(5);
out.print("cube of 5 is "+m);
%>
```

Output



jsp:setProperty and jsp:getProperty

The setProperty and getProperty action tags are used for developing web application with Java Bean. In web devlopment, bean class is mostly used because it is a reusable software component that represents data. **The jsp:setProperty action tag** sets a property value or values in a bean using the setter method.

Syntax of jsp:setProperty action tag

```
1. <jsp:setProperty name="instanceOfBean" property= "*"   |
2. property="propertyName" param="parameterName"  |
3. property="propertyName" value="{ string | <%= expression %>}"
4. />
```

Example of jsp:setProperty action tag if you have to set all the values of incoming request in the bean

```
1. <jsp:setProperty name="bean" property="*" />
```

Example of jsp:setProperty action tag if you have to set value of the incoming specific property

```
1. <jsp:setProperty name="bean" property="username" />
```

Example of jsp:setProperty action tag if you have to set a specific value in the property

```
1. <jsp:setProperty name="bean" property="username" value="Kumar" />
```

jsp:getProperty

The jsp:getProperty action tag returns the value of the property.

Syntax of jsp:getProperty action tag

```
1. <jsp:getProperty name="instanceOfBean" property="propertyName" />
```

Simple example of jsp:getProperty action tag

```
1. <jsp:getProperty name="obj" property="name" />
```

Example

- index.html for input of values
- welocme.jsp file that sets the incoming values to the bean object and prints the one value
- User.java bean class that have setter and getter methods

index.html

```
<form action="process.jsp" method="post">
Name:<input type="text" name="name"><br>
Password:<input type="password" name="password"><br>
Email:<input type="text" name="email"><br>
<input type="submit" value="register">
</form>
```

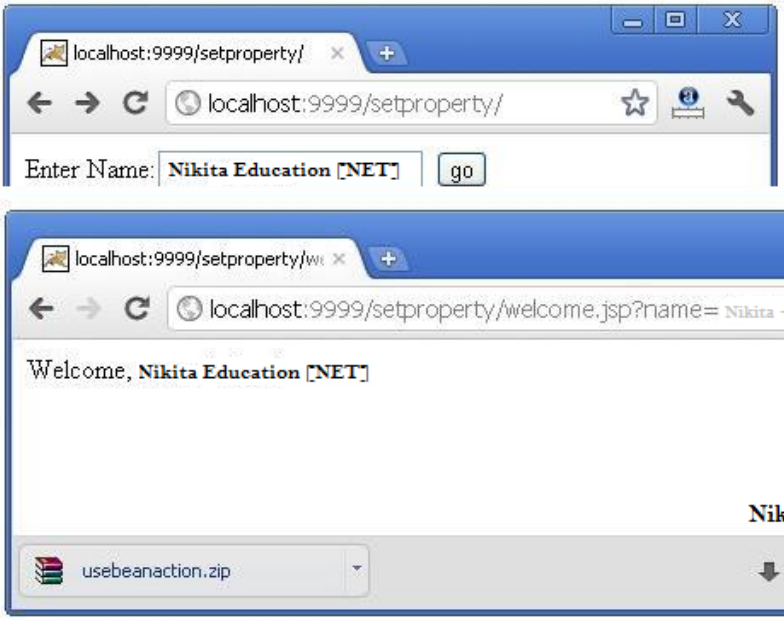
process.jsp

```
<jsp:useBean id="u" class="org.sssit.User"></jsp:useBean>
<jsp:setProperty property="*" name="u"/>

Record:<br>
<jsp:getProperty property="name" name="u"/><br>
<jsp:getProperty property="password" name="u"/><br>
<jsp:getProperty property="email" name="u" /><br>
```

User.java

```
package org.sssit;

public class User {
private String name,password,email;
//setters and getters
}
```

Output



Reusing Bean in Multiple Jsp Pages

Let's see the simple example, that prints the data of bean object in two jsp pages.

index.jsp
Same as above.

User.java
Same as above.

process.jsp

```
1. <jsp:useBean id="u" class="org.sssit.User" scope="session"></jsp:useBean>
2. <jsp:setProperty property="*" name="u"/>
3.
4. Record:<br>
5. <jsp:getProperty property="name" name="u"/><br>
6. <jsp:getProperty property="password" name="u"/><br>
7. <jsp:getProperty property="email" name="u" /><br>
8.
9. <a href="second.jsp">Visit Page</a>
```

second.jsp

```
1. <jsp:useBean id="u" class="org.sssit.User" scope="session"></jsp:useBean>
2. Record:<br>
3. <jsp:getProperty property="name" name="u"/><br>
4. <jsp:getProperty property="password" name="u"/><br>
5. <jsp:getProperty property="email" name="u" /><br>
```

Using variable value in setProperty tag

In some case, you may get some value from the database, that is to be set in the bean object, in such case, you need to use expression tag. For example:

process.jsp

```
<jsp:useBean id="u" class="org.sssit.User"></jsp:useBean>
<%
String name="arjun";
%>
<jsp:setProperty property="name" name="u" value="<%=name %>"/>

Record:<br>
<jsp:getProperty property="name" name="u"/><br>
```

M. Applet in JSP (jsp:plugin)

The jsp:plugin action tag is used to embed applet in the jsp file. The jsp:plugin action tag downloads plugin at client side to execute an applet or bean.

Syntax of jsp:plugin action tag

```
<jsp:plugin type= "applet | bean" code= "nameOfClassFile"
codebase= "directoryNameOfClassFile"
</jsp:plugin>
```

Example of displaying applet in JSP

In this example, we are simply displaying applet in jsp using the jsp:plugin tag. You must have MouseDrag.class file (an applet class file) in the current folder where jsp file resides. You may simply download this program that contains index.jsp, MouseDrag.java and MouseDrag.class files to run this application.

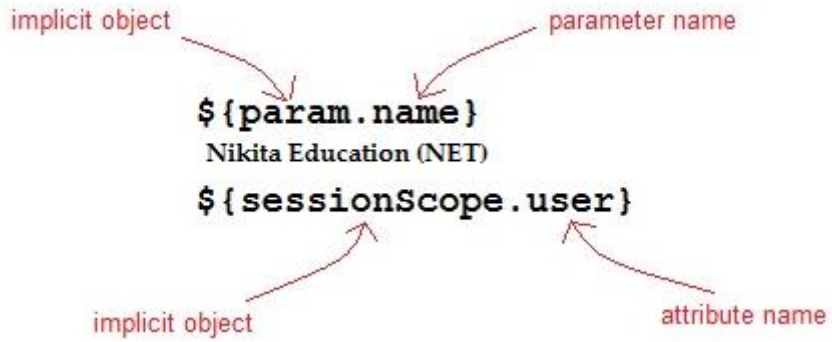
index.jsp

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Mouse Drag</title>
    </head>
    <body bgcolor="khaki">
<h1>Mouse Drag Example</h1>

 <jsp:plugin align="middle" height="500" width="500"
    type="applet"  code="MouseDrag.class" name="clock" codebase="."/>

    </body>
</html>
```

N. JSP Expression Language

The **Expression Language** (EL) simplifies the accessibility of data stored in the Java Bean component, and other objects like request, session, application etc. There are many implicit objects, operators and reserve words in EL. It is the newly added feature in JSP technology version 2.0.

Syntax for Expression Language (EL): **${ expression }**



How EL expression is used?

EL expression can be used in two ways in a JSP page:

1. *As attribute values in standard and custom tags.*

   Example:
   <jsp:include page="${location}">

Where **location** variable is separately defines in the jsp page. Expressions can also be used in jsp:setProperty to set a properties value, using other bean properties like : If we have a bean named **Square** with properties *length*, *breadth* and *area*.

<**jsp:setProperty** name="square" property="area" value="**${square.length*square.breadth}**" />

2. *To output in HTML tag :*

   <h1>Welcome ${name}</h1>

To deactivate the evaluation of EL expressions, we specify the isELIgnored attribute of the page directive as below:

<%@ page **isELIgnored** ="*true/false*" %>

Implicit Objects in Expression Language (EL)

Implicit Objects	**Usage**
pageScope	it maps the given attribute name with the value set in the page scope
requestScope	it maps the given attribute name with the value set in the request scope
sessionScope	it maps the given attribute name with the value set in the session scope
applicationScope	it maps the given attribute name with the value set in the application scope
param	it maps the request parameter to the single value
paramValues	it maps the request parameter to an array of values
header	it maps the request header name to the single value
headerValues	it maps the request header name to an array of values
cookie	it maps the given cookie name to the cookie value
initParam	it maps the initialization parameter
pageContext	it provides access to many objects request, session etc.

Example of EL

index.jsp

```
<form method="POST" action="welcome.jsp">
   Name <input type="text" name="user" >
   <input type="submit" value="Submit">
</form>
```

welcome.jsp

```
<html>
   <head>
      <title>Welcome Page</title>
   </head>
   <body>
      <h1>Welcome ${param.name}</h1>
   </body>
</html>
```

Arithmetic Operations available in EL

Arithmetic Operation	Operator
Addition	+
Subtraction	-
Multiplication	*
Division	/ and div
Remainder	% and mod

Logical and Relational Operators available in EL

Logical and Relational Operator	Operator
Equals	== and eq
Not equals	!= and ne
Less Than	< and lt
Greater Than	> and gt
Greater Than or Equal	>= and ge
Less Than or Equal	<= and le
and	&& and and
or	|| and or
not	! and not

Reserve words in EL

lt	le	gt	ge
eq	ne	true	false
and	or	not	instanceof
div	mod	empty	null

O. JSTL

JSP Standard Tag Library (JSTL) is a standard library of readymade tags. The JSTL contains several tags that can remove scriptlet code from a JSP page by providing some ready to use, already implemented common functionalities.

Advantage of JSTL

1. **Fast Development** JSTL provides many tags that simplify the JSP.
2. **Code Reusability** We can use the JSTL tags on various pages.
3. **No need to use scriptlet tag** It avoids the use of scriptlet tag.

JSTL is divided into 5 groups:

1. **JSTL Core**: JSTL Core provides several core tags such as **if**, **forEach**, **import**, **out** etc to support some basic scripting task. Url to include JSTL Core Tag inside JSP page is

   <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

2. **JSTL Formatting**: JSTL Formatting library provides tags to format text, date, number for Internationalised web sites. Url to include JSTL Formatting Tags inside JSP page is

   <%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>

3. **JSTL sql**: JSTL SQL library provides support for Relational Database Connection and tags to perform operations like insert, delete, update, select etc on SQL databases. Url to include JSTL SQL Tag inside JSP page is

   <%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>

4. **JSTL XML**: JSTL XML library provides support for XML processing. It provides flow control, transformation features etc. Url to include JSTL XML Tag inside JSP page is

   <%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>

5. **JSTL functions**: JSTL functions library provides support for string manipulation. Url to include JSTL Function Tag inside JSP page is

   <%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>

JSTL Core Library

The JSTL core library contains several tags that can be used to eliminate the basic scripting overhead such as for loop, if...else conditions etc from a JSP Page. Let's study some important tags of JSTL Core library.

- **JSTL if tag**: The if tag is a conditional tag used to evaluate conditional expressions. When a body is supplied with if tag, the body is evaluated only when the expression is true. For Example :

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
  <head>
    <title>Tag Example</title>
  </head>
  <body>
      <c:if test="${param.name == 'studytonight'}">
      <p>Welcome to ${param.name} </p>
    </c:if>
  </body>
</html>
```

- **JSTL out tag**: The out tag is used to evaluate an expression and write the result to **JspWriter**. For Example :

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
  <head>
    <title>Tag Example</title>
  </head>
  <body>
```

```
    <c:out value="${param.name}" default="StudyTonight" />
  </body>
</html>
```

The value attribute specifies the expression to be written to the JspWriter. The default attribute specifies the value to be written if the expression evaluates null.

- **JSTL forEach tag**: This tag provides a mechanism for iteration within a JSP page. JSTL forEachtag works similarly to **enhanced for** loop of Java Technology. You can use this tag to iterate over an existing collection of items. For Example :

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
  <head>
    <title>Tag Example</title>
  </head>
  <body>
     <c:forEach var="message" items="${errorMsgs}" >
       <li>${message}</li>
     </c:forEach>
  </body>
</html>
```

Here the attribute **items** has its value as an EL expression which is a collection of error messages. Each item in the iteration will be stored in a variable called **message** which will be available in the body of the **forEach** tag.

- **JSTL choose, when, otherwise tag**: These are conditional tags used to implement conditional operations. If the test condition of the when tag evaluates to true, then the content within when tag is evaluated, otherwise the content within the otherwise tag is evaluated.
  We can also implement if-else-if construct by using multiple **when** tag. The **when** tags are mutually exclusive, that means the first when tag which evaluates to true is evaluated and then, the control exits the choose block. If none of the when condition evaluates to true, then otherwise condition is evaluated. For Example

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
        <head>
                <title>Tag Example</title>
        </head>
        <body>
        <c:forEach var="tutorial" items="${MyTutorialMap}" begin="0" end="5"
        varStatus="status">
                <c:choose>
                        <c:when test="${status.count %2 == 0 }">
                                <p> Divisible by 2 : ${tutorial.key} </p><br/>
                        </c:when>

                        <c:when test="${status.count %5 == 0 }">
                                <p > Divisible by 5 : ${tutorial.key} </p><br/>
                        </c:when>

                        <c:otherwise>
                                <p> Neither divisible by 2 nor 5 : ${tutorial.key} </p><br/>
                        </c:otherwise>
                </c:choose>
        </c:forEach>
        </body>
</html>
```

- **JSTL import tag**: < c:import> tag is used to dynamically add the contents from the provided URL to the current page, at request time. The URL resource used in the < c:import> url attribute can be from outside the web Container. For Example :

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
  <head>
    <title>Tag Example</title>
  </head>
  <body>
    <c:import url="http://www.example.com/hello.html">>
    <c:param name="showproducts" value="true"/>
    </c:import>
  </body>
</html>
```

- **JSTL url tag**: The JSTL url tag is used to store a url in a variable and also perform url rewriting when necessary. For Example

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
  <head><title>Tag Example</title></head>
  <body>
    <a href='<c:url value="/home.jsp"/>' > Go Home </a>
  </body>
</html>
```

- **JSTL set tag**: The JSTL set tag is used to store a variable in specified scope or update the property of JavaBean instance. Following is the example of setting the **name** property of a Student bean :

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
  <head>
    <title>Tag Example</title>
  </head>
  <body>
    <c:set target="student" property="name" value="${param.name}" />
  </body>
</html>
```

- **JSTL catch tag**: The JSTL catch tag is used to handle exception and doesn't forward the page to the error page. For Example :

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
  <head>
    <title>Tag Example</title>
  </head>
  <body>
    <c:catch>
      <% int a = 0;
         int b = 10;
         int c = b/a;
      %>
     </c:catch>
  </body>
</html>
```

P. JSP Custom Tags

When EL and Standard Action elements aren't enough to remove scriptlet code from your JSP Page, you can use Custom Tags. Custom tags are nothing but user-defined tags. Custom tags are an excellent way to abstract the complexity of business logic from the presentation of Web pages in a way that is easy for the Web author to use and control. It also allows for reusability as custom tags can be used again and again.

Advantages of Custom Tags

1. **Eliminates the need of scriptlet tag** The custom tags eliminates the need of scriptlet tag which is considered bad programming approach in JSP.
2. **Separation of business logic from JSP** The custom tags separate the the business logic from the JSP page so that it may be easy to maintain.
3. **Re-usability** The custom tags makes the possibility to reuse the same business logic again and again.

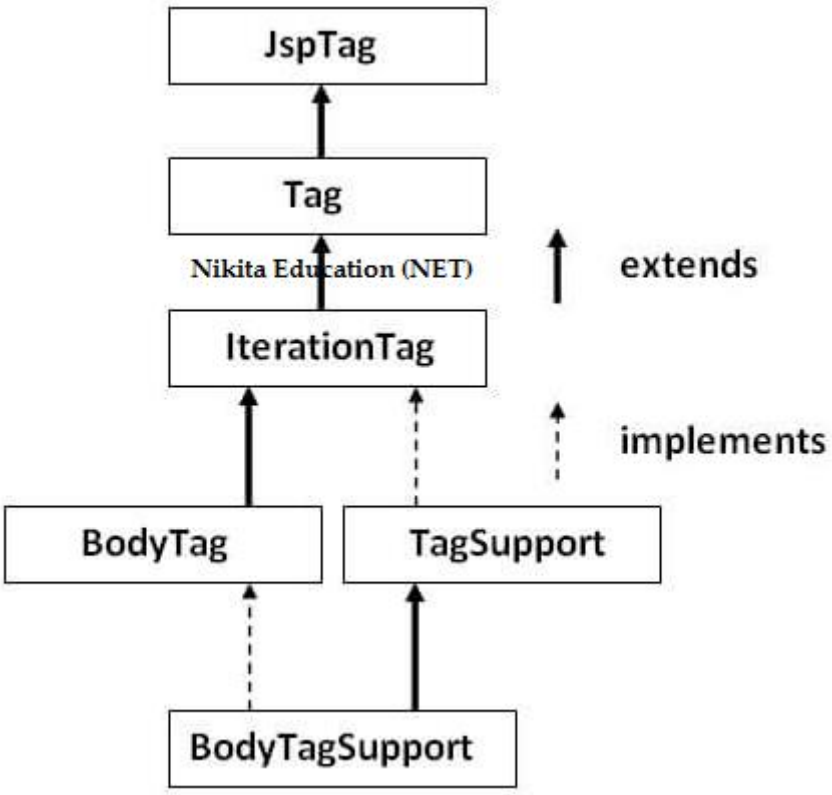
Syntax

The format of a custom tag can either be empty, called an **Empty tag**, or can contain a body, called a **Body tag**. The number of attributes that a tag will accept depends on the implementation of the Tag Handler class.

Empty Tag	**<prefix:tagname** attr1=value1....attrn=valuen **/>**
Body Tag	**<prefix:tagname** attr1=value1....attrn=valuen **>** body code **</prefix:tagname>**

JSP Custom Tag API

The javax.servlet.jsp.tagext package contains classes and interfaces for JSP custom tag API. The JspTag is the root interface in the Custom Tag hierarchy.



Creating a Custom Tag

1. The **Tag Handler** class which should extend SimpleTagSupport.
2. The **Tag Library Descriptor(TLD)** file
3. Use the Custom Tag in your JSP file

Tag Handler Class

You can create a Tag Handler class in two different ways:

1. By implementing one of three interfaces : SimpleTag, Tag or BodyTag, which define methods that are invoked during the life cycle of the tag.
2. By extending an abstract base class that implements the SimpleTag, Tag, or BodyTag interfaces. The **SimpleTagSupport**, **TagSupport**, and **BodyTagSupport** classes implement the SimpleTag, Tag andBodyTag interfaces . Extending these classes relieves the tag handler class from having to implement all methods in the interfaces and also provides other convenient functionality.

Tag Library Descriptor

A Tag Library Descriptor is an XML document that contains information about a library as a whole and about each tag contained in the library. TLDs are used by the web container to validate the tags and also by JSP page development tools.

Tag library descriptor file must have the extension .tld and must be packaged in the **/WEB-INF/**directory or subdirectory of the WAR file or in the **/META-INF/** directory or subdirectory of a tag library packaged in a JAR.

Example of Custom Tag

In our example, we will be creating a Tag Handler class that extends the **TagSupport** class. When we extend this class, we have to override the method doStartTag(). There are two other methods of this class namely doEndTag() and release(), that we can decide to override or not depending on our requirement.

CountMatches.java

```
package com.studytonight.taghandler;

import java.io.IOException;
import javax.servlet.jsp.*;
import org.apache.commons.lang.StringUtils;

public class CountMatches extends TagSupport {
  private String inputstring;
  private String lookupstring;

  public String getInputstring() {
    return inputstring;
  }

  public void setInputstring(String inputstring) {
    this.inputstring = inputstring;
  }

  public String getLookupstring() {
    return lookupstring;
  }

  public void setLookupstring(String lookupstring) {
    this.lookupstring = lookupstring;
  }

  @Override
  public int doStartTag() throws JspException {
```

```
    try {
      JspWriter out = pageContext.getOut();
      out.println(StringUtils.countMatches(inputstring, lookupstring));
    }
    catch (IOException e) {
      e.printStackTrace();
    }
    return SKIP_BODY;
  }
}
```

In the above code, we have an implementation of the doStartTag() method which is must if we are extending **TagSupport** class. We have declared two variables inputstring and lookupstring. These variables represents the **attributes** of the custom tag. We must provide getter and setter for these variables in order to set the values into these variables that will be provided at the time of using this custom tag. We can also specify whether these attributes are required or not.

CountMatchesDescriptor.tld

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib>
  <tlibversion>1.0</tlibversion>
  <jspversion>1.1</jspversion>
  <shortname>cntmtchs</shortname>
  <info>Sample taglib for Substr operation</info>
  <uri>http://studytonight.com/jsp/taglib/countmatches</uri>

  <tag>
    <name>countmatches</name>
    <tagclass>com.studytonight.taghandler.CountMatches</tagclass>
    <info>String Utility</info>
    <attribute>
      <name>inputstring</name>
      <required>true</required>
    </attribute>
    <attribute>
      <name>lookupstring</name>
      <required>true</required>
    </attribute>
  </tag>
</taglib>
```

The taglib element specifies the schema, required JSP version and the tags within this tag library. Each **tag** element within the TLD represents an individual custom tag that exist in the library. Each of these tag should have a tag handler class associated with them.
The **uri** element represents a Uniform Resource Identifier that uniquely identifies the tag library. The two **attribute** elements within the **tag** element represents that the tag has two attributes and the **true** value provided to the **required** element represents that both of these attributes are required for the tag to function properly.

test.jsp

```
<%@taglib prefix="mytag" uri="/WEB-INF/CountMatchesDescriptor.tld"%>
<html>
  <mytag:countmatches inputstring="Studytonight" lookupstring="t">
  </mytag:countmatches>
</html>
```

If this tag works fine it should print a value 3 in the browser as there 't' occurs 3 times in the word 'Studytonight'.

Q. Introduction to RMI

The RMI (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM. The RMI provides remote communication between the applications using two objects stub and skeleton. RMI (Remote Method Invocation) is a way that a programmer, using the Java programming language and development environment, can write object-oriented programming in which objects on different computers can interact in a distributed network. RMI is the Java version of what is generally known as a remote procedure call (RPC), but with the ability to pass one or more objects along with the request. The object can include information that will change the service that is performed in the remote computer. Remote Method Invocation (RMI) facilitates object function calls between Java Virtual Machines (JVMs).

RPC vs. RMI

RPC (Remote Procedure Call) and RMI (Remote Method Invocation) are two mechanisms that allow the user to invoke or call processes that will run on a different computer. RPC uses procedure oriented mechanism whereas RMI uses object oriented mechanism. RPC is a relatively old protocol that is based on the C language whereas RMI is new and it is based on Java language.

- RMI is object oriented while RPC isn't
- RPC is C bases while RMI is Java only
- RMI invokes methods while RPC invokes functions
- RPC is antiquated while RMI is the future

RMI Architecture

Application Layer

This layer is nothing but the actual systems (client and server) involved in communication. A client Java program communicates with the other Java program on the server side. RMI is nothing but a communication between two JVMs placed on different systems.

Operational Layers: The RMI system consists of three layers:

1. The Stub/Skeleton Layer or proxy layer

The proxy layer consists of **proxies** (named as **stub** and **skeleton**) for client and server. **Stub is client side proxy** and **Skeleton is server side proxy**. The stub/skeleton layer is the interface between the application layer and the rest of the RMI system. This layer does not deal with specifics of any transport, but transmits data to the remote reference layer via the abstraction of *marshal streams*. **Marshaling** is nothing but converting data into a special format suitable to pass through the distributed environment without losing **object persistence**. **Un-marshaling** is the process of extracting marshaled data into its original format. **Stub** performs the marshaling, whereas **Skeleton** performs the un-marshaling.

- **Stub**
  The stub is a client-side object that represents (or acts as a proxy for) the remote object. Sequence of events performed by the stub:
  - ✓ Initiates a connection with the remote VM containing the remote object
  - ✓ Marshals (writes and transmits) the parameters to the remote VM.
  - ✓ Waits for the result of the method invocation.
  - ✓ Unmarshals (reads) the return value or exception returned.
  - ✓ Return the value to the caller

- **Skeleton**
  On the server side, the skeleton object takes care of all the details of "remoteness" so that the actual remote object does not need to worry about them. Sequence of events performed by the skeleton:
  - ✓ Unmarshals (reads) the parameters for the remote method (remember that these were marshaled by the stub on the client side)
  - ✓ Invokes the method on the actual remote object implementation.
  - ✓ Marshals (writes and transmits) the result (return value or exception) to the caller (which is then unmarshalled by the stub)

2. The Remote Reference Layer

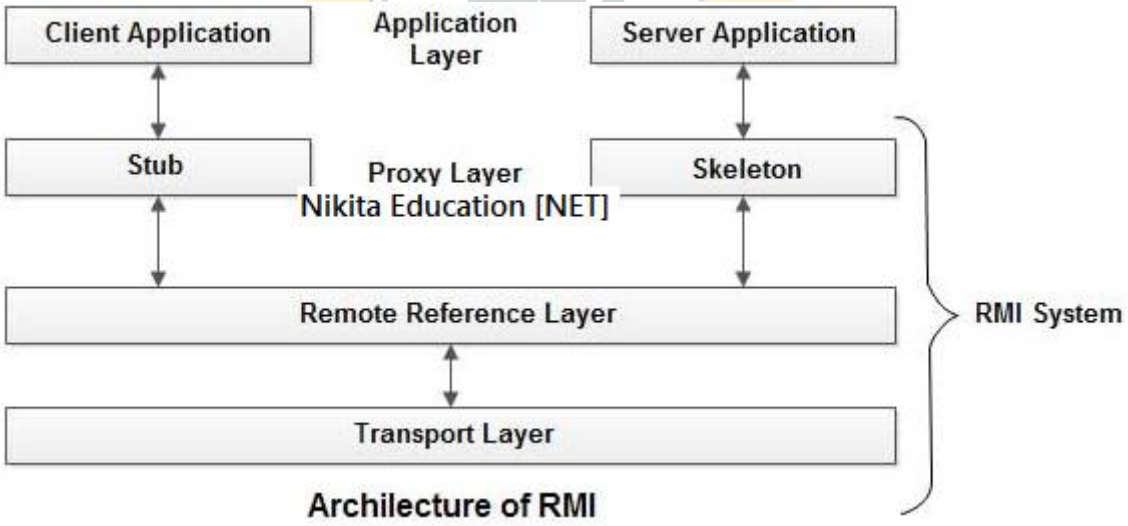
Proxies are implicitly connected to RMI mechanism through Remote reference layer, the layer responsible for object communication and transfer of objects between client and server. It is responsible for dealing with semantics of remote invocations and implementation – specific tasks with remote objects. In this layer, actual implementation of communication protocols is handled.

- Unicast point-to-point invocation.
- Invocation to replicated object groups.
- Support for a specific replication strategy.
- Support for a persistent reference to the remote object (enabling activation of the remote object).
- Reconnection strategies (if remote object becomes inaccessible).

3. The Transport Layer

Transport layer does not exist separately but is a part of Remote reference layer. Transport layer is responsible for actually setting up connections and handling the transport of data from one machine to another. It can be modified to handle encrypted streams, compression algorithms and a number of other security/performance related enhancements. In general, the transport layer of the RMI system is responsible for:

- Setting up connections to remote address spaces.
- Managing connections.
- Monitoring connection "liveness."
- Listening for incoming calls.
- Maintaining a table of remote objects that reside in the address space.
- Setting up a connection for an incoming call.
- Locating the dispatcher for the target of the remote call and passing the connection to this dispatcher.



Archilecture of RMI

Goals of RMI

A primary goal for the RMI designers was to allow programmers to develop distributed Java programs with the same syntax and semantics used for non-distributed programs.

- ✓ Minimize difference between working with local and remote objects
- ✓ Minimize complexity
- ✓ Preserve type safety
- ✓ Distributed garbage collection
- ✓ Invocation of object methods in another java virtual machines
- ✓ Distribution transference (java semantics)
- ✓ Easy to use

Advantages of RMI	Disadvantages of RMI
Handles threads and Sockets for underlying communication.	Overhead of marshaling and un-marshaling
Server-side implementation can be changed without the knowledge of the client side.	Overhead of object serialization
To extend an RMI solution, you can extend or add new classes, just like in a non-distributed application	Cannot use the code out of the scope of java.

Basic Terminologies of RMI

1) RMI Registry:
A remote object registry is a bootstrap naming service that is used by RMI servers on the same host to bind remote objects to names. Clients on local and remote hosts can then look up remote objects and make remote method invocations. **Default port for RMI Registry is: 1099**

2) rmic compiler:
The **rmic compiler generates stub and skeleton** class files (JRMP protocol) and stub and tie class files (IIOP protocol) for remote objects. These classes files are generated from compiled Java programming language classes that are remote object implementation classes.

Steps to design RMI application:

1. Create the remote interface
2. Provide the implementation of the remote interface
3. Create Server application and register service with registry
4. Create client application

Steps to execute RMI application:

1. compile all the java files
2. create stub and skeleton object by rmic tool
3. start rmi registry in one command prompt
4. start the server in another command prompt
5. start the client application in another command prompt

R. Introduction to EJB

EJB is an acronym for ***enterprise java bean***. It is a specification provided by Sun Microsystems to develop secured, robust and scalable distributed applications. Enterprise Java Beans ('EJB') is a managed, server-side component architecture for modular construction of enterprise applications. EJB is a server-side model that encapsulates the business logic of an application. The EJB specification intends to provide a standard way to implement the back-end 'business' code typically found in enterprise applications.

Features of EJB

The EJB specification details how an application server provides the following responsibilities:

- Transaction processing
- Integration with the persistence services offered by the Java Persistence API (JPA)
- Concurrency control
- Event-driven programming using Java Message Service and Java EE Connector Architecture
- Asynchronous method invocation
- Job scheduling
- Naming and directory services (JNDI)
- Inter-process Communication using RMI-IIOP and Web services

- Security (JCE and JAAS)
- Deployment of software components in an application server
- EJB components are server-side components written entirely in the Java programming language
- EJB architecture is inherently transactional, distributed, portable multi-tier, scalable and secure.
- EJB components are fully portable across any EJB server and any OS.
- EJB architecture is wire-protocol neutral--any protocol can be utilized like IIOP,JRMP, HTTP, DCOM etc.

Benefits of EJB

- Simplified development of large scale enterprise level application.
- Application Server/ EJB container provides most of the system level services like transaction handling, logging, load balancing, persistence mechanism, exception handling and so on. Developer has to focus only on business logic of the application.
- EJB container manages life cycle of ejb instances thus developer needs not to worry about when to create/delete ejb objects.

Types of EJB

Session Bean	Session bean stores data of a particular user for a single session. It can be stateful or stateless. It is less resource intensive as compared to entity beans. Session bean gets destroyed as soon as user session terminates.
Entity Bean	Entity bean represents persistent data storage. User data can be saved to database via entity beans and later on can be retrieved from the database in the entity bean.
Message Driven Bean	Message driven beans are used in context of JMS (Java Messaging Service). Message Driven Beans can consumes JMS messages from external entities and act accordingly.

RMI vs. EJB

Both RMI and EJB, provides services to access an object running in another JVM (known as remote object) from another JVM. The differences between RMI and EJB are given below:

RMI	**EJB**	**Web Service**
In RMI, middleware services such as security, transaction management, object pooling etc. need to be done by the java programmer.	In EJB, middleware services are provided by EJB Container automatically.	In EJB, bean component and bean client both must be written in java language. If bean client need to be written in other language such as **.net**, **php** etc, we need to go with **web services** (SOAP or REST). So EJB with web service will be better option.
RMI is not a server-side component. It is not required to be deployed on the server.	EJB is a server-side component; it is required to be deployed on the server.	
RMI is built on the top of socket programming.	EJB technology is built on the top of RMI.	

Limitations of EJB

1. Requires application server
2. Requires only java client. For other language client, you need to go for web service.
3. Complex to understand and develop EJB applications.

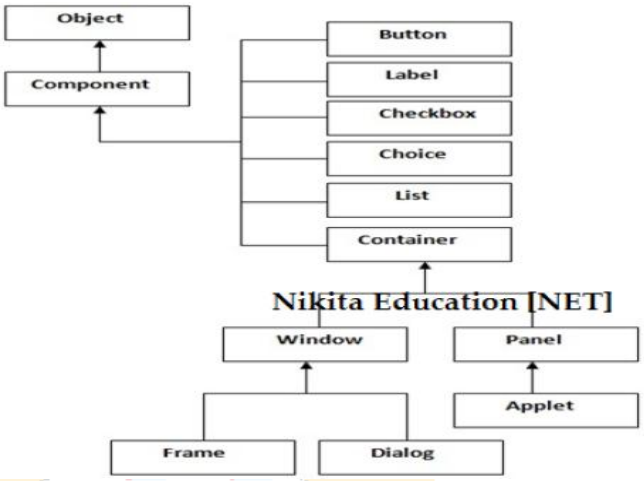
Appendices

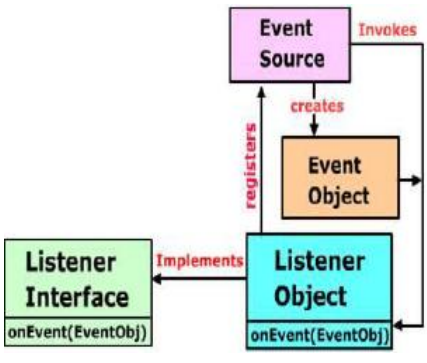
Appendix - A

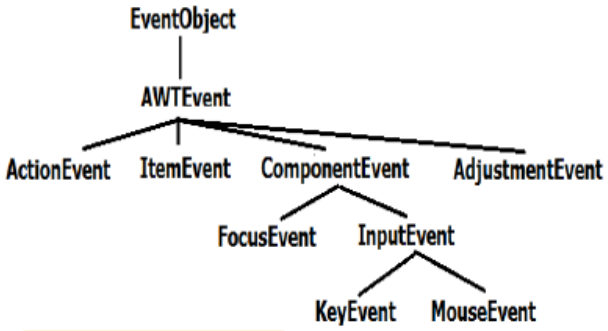
[Practice Set]

[Sample Oral Questions]

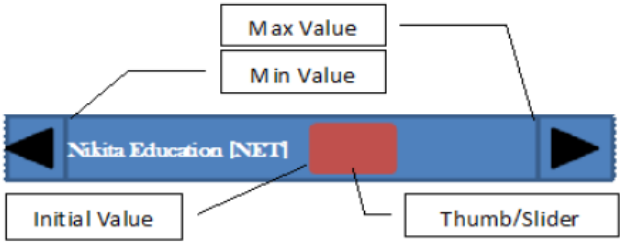
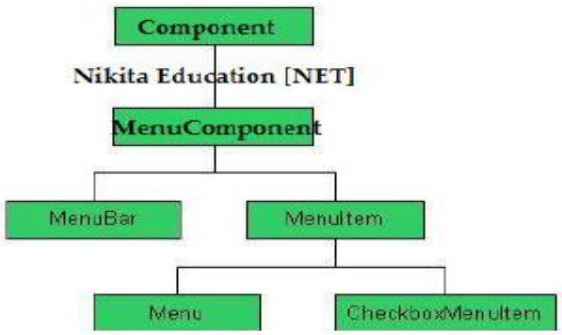
Practice Set - Sample Oral Questions and Answers

No.	Questions & Answers
01 - AWT	*Nikita Education [NET]*
1	**What is GUI?** Graphical User Interface (GUI) offers user interaction via some graphical components. *Nikita Education [NET]*
2	**What is AWT?** Abstract Window Toolkit (AWT) is a set of application program interfaces (API - collection of classes and interfaces) used by Java programmers to create graphical user interface (GUI). *Nikita Education [NET]*
3	**Draw AWT hierarchy.**  *Nikita Education [NET]*
4	**What is Component, Container, Window, Panel, Frame?** A **Component** is an abstract super class for GUI controls and it represents any object of its subclasses with graphical representation. The **Container** is a component in AWT that can contain another components like buttons, textfields, labels. The **Window** is the container that have no borders and menu bars. The **Panel** is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc. The **Frame** is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc. *Nikita Education [NET]*
5	**Enlist constructors and important methods for Frame.** public Frame() public Frame(String title) setSize(int width, int height) setVisible(boolean b) setTitle(String title) addWindowListener(WindowListener wl) setLayout(LayoutManager mgr) *Nikita Education [NET]*
6	**How many ways are available to create a frame window?** 1. Create object of frame class 2. Inherit properties of frame class *Nikita Education [NET]*
7	**Write any four methods of Graphics class.** void drawRect(int top, int left, int width, int height) void drawOval(int top, int left, int width, int height) void drawPolygon(int x[ ], int y[ ], int numPoints) void fillRoundRect(int top, int left, int width, int height, int xDiam, int yDiam) *Nikita Education [NET]*
8	**Enlist constants of Color class** Color.black, Color.red, Color.green, Color.blue, Color.pink, Color.orange, Color.yellow *Nikita Education [NET]*

9	**Enlist constructors of Color class** Color(int, int, int) // int value in between 0 – 255 Color(int rgbValue) Color(float, float, float) // float value in between 0.0 – 1.0 *Nikita Education [NET]*
10	**How do we can set color on container and components?** 1. Setting color on container: void setBackground(Color c) void setForeground(Color c) 2. Setting color on components or graphics: void setColor(Color newColor) Color getColor( ) *Nikita Education [NET]*
11	**What is paint mode?** The paint mode determines how objects are drawn in a window. By default, new output to a window overwrites any preexisting contents. *Nikita Education [NET]*
12	**What is family name, logical name, and face name in fonts?** 1. The family name is the general name of the font, such as Courier. 2. The logical name specifies a category of font, such as Monospaced. 3. The face name specifies a specific font, such as Courier Italic. *Nikita Education [NET]*
13	**Write constructor for creating Font object.** **public Font**(String name, int style, int size) *Nikita Education [NET]*
14	**Setting or Obtaining Font** **Font getFont**() - Gets the font of this component. **void setFont**(**Font** f) - Sets the font of this component. *Nikita Education [NET]*
15	**FontMetrics** class encapsulates **information about a font** **The common terminology used when describing fonts:** 1. **Height** The top-to-bottom size of a line of text 2. **Baseline** The line that the bottoms of characters are aligned to (not counting descent) 3. **Ascent** The distance from the baseline to the top of a character 4. **Descent** The distance from the baseline to the bottom of a character 5. **Leading** The distance between the bottom of one line of text and the top of the next *Nikita Education [NET]*
16	**Obtaining FontMetrics object** • FontMetrics FM=getFontMetrics(Font F); • FontMetrics FMS=g.getFontMetrics(); *Nikita Education [NET]*
17	**Events:** Event is an object that describes a state of change in a source. Event may be generated as a consequence of a person interacting with the GUI elements *Nikita Education [NET]*
18	**Event Sources:** "Source" is an object that generates an event. listeners must register with the source in order to receive event notification. *Nikita Education [NET]*
19	**Event Listeners:** A listener is an object that is notified when an event occurs.
20	**Delegation Event Model**  *Nikita Education [NET]*

<table>
<tr><td>21</td><td colspan="2">Enlist any five event classes and interfaces. Nikita Education [NET]</td></tr>
<tr><td></td><td>ActionEvent
MouseEvent
KeyEvent
ItemEvent
WindowEvent</td><td>ActionListener
MouseListener
KeyListener
ItemListener
WindowListener</td></tr>
<tr><td>22</td><td colspan="2">What is adapter class? Enlist any three. Nikita Education [NET]
An adapter class provides an empty implementation of all methods in an event listener interface.
Classes: WindowAdapter, KeyAdapter, MouseAdapter</td></tr>
<tr><td>23</td><td colspan="2">Event Hirarchy
 

Nikita Education [NET]</td></tr>
<tr><td>24</td><td colspan="2">Standard steps to create and add components in a container.
Create instance of control, set basic properties, register event, add components using add() method</td></tr>
<tr><td>25</td><td colspan="2">Which containers use a border layout as their default layout?
The Window, Frame and Dialog classes use a border layout as their default layout.
Nikita Education [NET]</td></tr>
<tr><td>26</td><td colspan="2">Which containers use a FlowLayout as their default layout?
The Panel and Applet classes use the FlowLayout as their default layout.</td></tr>
<tr><td>27</td><td colspan="2">What is a heavyweight component?
For every paint call, there will be a native call to get the graphical units</td></tr>
<tr><td>28</td><td colspan="2">What is an applet?
An applet is a small java program that runs inside the browser and generates dynamic contents.
Nikita Education [NET]</td></tr>
<tr><td>29</td><td colspan="2">What is a layout manager?
A layout manager dictates the style of arranging the components in a container.</td></tr>
<tr><td>30</td><td colspan="2">How many layout managers are available in Java?
There are 5 layout managers defined in java.awt package – FlowLayout, BorderLayout, GridLayout, CardLayout and GridBagLayout.</td></tr>
<tr><td>31</td><td colspan="2">What is the method used to place some text in the text field?
setText(String str) method of TextField class.
Nikita Education [NET]</td></tr>
<tr><td>32</td><td colspan="2">What is the method used to get the data entered by the user in the text field?
getText() method of TextField class.</td></tr>
<tr><td>33</td><td colspan="2">What is the difference between text field and text area?
TextField and TextArea are used to get or display messages from user. The difference is text field displays the message in one line of text only but of any length where as text area is used to display multiple lines of text.
Nikita Education [NET]</td></tr>
<tr><td>34</td><td colspan="2">What is the method used to change the characters entered by the user in the text field (used for password)?
setEchoChar(char ch) of TextField class.</td></tr>
<tr><td>35</td><td colspan="2">How to make the text field non-editable by the user (user cannot enter anything)?
setEditable(boolean state) of TextField class. This type of text field is used only to display text to the user.
Nikita Education [NET]</td></tr>
</table>

36	**What is the method used to know the label of the button clicked by the user?** getActionCommand() method of ActionEvent class. *Nikita Education [NET]*
37	**What is the super class of TextField and TextArea?** java.awt.TextComponent (a subclass of java.awt.Component).
38	**How many ways you can align the label in a container?** 3 ways. The variables defined in Label class LEFT, RIGHT and CENTER are used to change the alignment. Default alignment is left. *Nikita Education [NET]*
39	**What is HeadlessExceptin?** It is an unchecked exception. This runtime exception is thrown when the code that is dependent on the hardware like keyboard, display or mouse is called in an environment that does not support a keyboard, display or mouse. *Nikita Education [NET]*
40	**What is the listener used to handle the events of a text field?** java.awt.event.ActionListener interface
41	**Enlist any three constructors of Checkbox** Checkbox() Checkbox(String label) Checkbox(String label, CheckboxGroup group, boolean state) *Nikita Education [NET]*
42	**How do we convert checkbox into radio button in AWT?** Using object of CheckboxGroup class
43	**What is the difference between Choice and List control of AWT?** Choice is a combo box that provides drop down list. It takes less area to display component. List provides normal list of elements. It takes more space to display component on GUI *Nikita Education [NET]*
44	**What is thumb, minimum, maximum, and initial value in Scrollbar.**  *Nikita Education [NET]*
45	**Constants of Scrollbar:** Scrollbar.VERTICAL, Scrollbar.HORIZONTAL
46	**Important constants of BorderLayout:** BorderLayout.NORTH, BorderLayout.CENTER, BorderLayout.SOUTH, BorderLayout.EAST, BorderLayout.WEST
47	**Enlist methods of CardLayout:** void first(Container parent) void last(Container parent) void next(Container parent) void previous(Container parent) void show(Container parent, String name) *Nikita Education [NET]*
48	**Draw hierarchy of menu classes**  *Nikita Education [NET]*

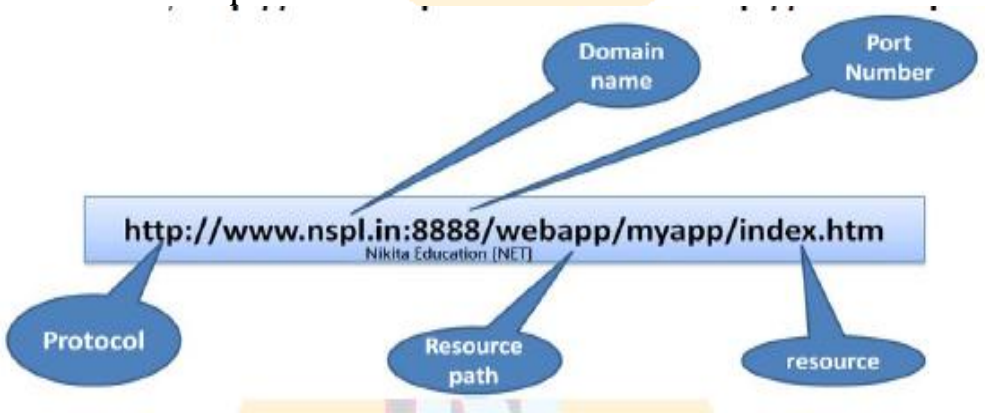
49	**What is Dialog and FileDialog? Enlist their constructors.** **Dialog** boxes are primarily used to obtain user input and are often child windows of a top-level window. **FileDialog** control represents a dialog window from which the user can select a file. **Dialog**(Frame owner, String title, boolean modal) **FileDialog**(Frame parent, String title, int mode) *Nikita Education [NET]*
50	**Dialog boxes may be modal or modeless**. When a **modal** dialog box is active, all input is directed to it until it is closed. This means that you cannot access other parts of your program until you have closed the dialog box. When a **modeless** dialog box is active, input focus can be directed to another window in your program. Thus, other parts of your program remain active and accessible.

02 - SWING *Nikita Education [NET]*

1	**What is Swing?** Set of classes and interfaces used to design flexible and attractive GUI.	
2	**Difference between AWT and Swing** *Nikita Education [NET]*	
	Heavyweight	Lightweight
	Java.awt package	Javax.swing package
	Platform dependant	Platform independent
	OS dependant look & feel	Pluggable look & feel
	Uses peer objects of OS for each components	Entirely written in java
	Uses simple architecture	Uses MVC architecture
	Simple and less portable	Portable and flexible
3	**What is JFC?** JFC stands for Java Foundation Classes. The Java Foundation Classes (JFC) are a set of Java class libraries provided as part of Java 2 Platform, Standard Edition (J2SE) to support building graphics user interface (GUI) and graphics functionality for client applications that will run on popular platforms such as Microsoft Windows, Linux, and Mac OSX. *Nikita Education [NET]*	
4	**Swing features:** Plugable look & feel, Uses MVC architecture , Lightweight components, Platform Independant, Advance features such as JTable, JTabbedPane, JScrollPane etc.	
5	**Swing is built on AWT** *Nikita Education [NET]*	
6	**MVC Architecture: Model - View - Controller** (model corresponds to the state information, view determines how the component is displayed on the screen, controller determines how the component reacts)	
7	A Swing GUI consists of two key items: **components** and **containers**. A **component** is an independent visual control, such as a push button A **container** holds a group of components. *Nikita Education [NET]*	
8	**JRootPane contains:** 1. **Glass Pane**: This enables you to manage mouse events that affect the entire container 2. **Layered Pane**: This allows components to be given a depth value. This value determines which component overlays another. It contains Content Pane and Menu Bar. 3. **Content Pane**: This pane holds the visual components such as buttons, labels, textfields	
9	**Important constants of Swing frame:** static int EXIT_ON_CLOSE, static int DISPOSE_ON_CLOSE, static int DO_NOTHING_ON_CLOSE, static int HIDE_ON_CLOSE *Nikita Education [NET]*	
10	**Enlist important constants of SwingConstants interface:** static int BOTTOM static int CENTER static int HORIZONTAL static int LEADING static int LEFT static int RIGHT static int TRAILING static int VERTICAL *Nikita Education [NET]*	

11	**Important methods of JLabel** *Nikita Education [NET]* Icon getIcon() String getText() void setIcon(Icon icon) void setText(String text)
12	**How do we create image icon in swing?** By creating object of ImageIcon class using following constructor. This icon can be set to any supported components by passing this object to constructor of that component. **Constructor:** ImageIcon(String filename)
13	**Difference between JTextField and JTextArea.** JTextField is a single line text control whereas JTextArea is a multi line control which allows us to append text, display file data etc. *Nikita Education [NET]*
14	**Which swing control is used for passwords?** JPasswordField
15	**Enlist constructors of JTextArea:** JTextArea(), JTextArea(String s), JTextArea(int row, int cols)
16	**Commonly used methods for text controls:** void addActionListener(ActionListener l) void setActionCommand(String command) String getSelectedText() String getText() void select(int selectionStart, int selectionEnd) void selectAll() void setText(String t) *Nikita Education [NET]*
17	**What are the type of Buttons in Swing:** JButton, JToggleButton, JCheckBox, JRadioButton. All are subclasses of AbstractButton. **Important methods of AbstractButton:** getSelectedIcon() setDisabledIcon(Icon disabledIcon) setIcon(Icon defaultIcon) setPressedIcon(Icon pressedIcon) setSelectedIcon(Icon selectedIcon)
18	**What is Toggle button?** Toggle button provides only two states i.e. on or off. When it is pressed it switches only between to states. Two important subclasses of JToggleButton are JCheckBox and JRadioButton. *Nikita Education [NET]*
19	**How do we create group of radio buttons in swing?** By adding radio buttons in object of ButtonGroup class. **Constructor** of ButtonGroup: public ButtonGroup()
20	**What is the difference between JComboBox and JList?** JComboBox displays one entry at atime whereas JList displays multiple entries at a time. JComboBox takes less space on GUI whereas JList takes more space. JComboBox provides drop down list, JList provides normal list of items. *Nikita Education [NET]*
21	**What is the method used in JComboBox or JList to add elements?** public addItem(Object obj)
22	**What is JTabbedPane?** It manages **a set of components** by linking them with tabs.
23	**Steps to create JScrollPane:** 1. Create the component to be scrolled. 2. Create an instance of JScrollPane, passing to it the object to scroll. 3. Add the scroll pane to the content pane
24	**What is JTree?** The JTree class is used to display the tree structured data or hierarchical data *Nikita Education [NET]*

25	**Enlist constructors of JTable.** *Nikita Education [NET]* JTable() JTable(Object[][] rows, Object[] columns)
03 - Java Networking	*Nikita Education [NET]*
1	**Computer Network:** A computer network is a group of computer systems and other computing hardware devices that are linked together through communication channels to facilitate communication and resource-sharing among a wide range of users
2	**IP Address:** IP address is short for Internet Protocol (IP) address. An IP address is an identifier for a computer or device on a TCP/IP network. *Nikita Education [NET]*
3	**Subnet Mask:** An IP address has two components, the network address and the host address. A subnet mask separates the IP address into the network and host addresses (<network><host>).
4	**IP Classes:** The IPv4 address space can be subdivided into 5 classes - Class A (1 - 126), B (128 - 191), C (192 - 223), D (224 - 239) and E (240 - 254). *Nikita Education [NET]*
5	**Proxy Server:** A server that sits between a client application and a real server. It intercepts all requests to the real server to see if it can fulfill the requests itself. If not, it forwards the request to the real server.
6	**TCP vs. UDP** 1. TCP stands for "**Transmission Control Protocol**" while UDP stands for "**User datagram Protocol**". 2. TCP is **connection oriented** protocol while UDP is **connectionless** protocol. 3. TCP is more **reliable** , UDP is **not reliable**. 4. UDP is **more faster** for data sending than TCP. 5. UDP makes **error checking but no reporting** but TCP makes **checks for errors and reporting**. 6. TCP gives **guarantee** that the **order of data** at receiving end is same as on sending end while UDP has **no such guarantee**. 7. Header size of TCP is **20 bytes** while that of UDP is **8 bytes**. 8. TCP is **heavy weight** as it needs three packets to setup a connection while UDP is **light weight**. 9. TCP has **acknowledgement** segments but UDP has **no acknowledgement**. 10. TCP is used for **application that require high reliability but less time critical** whereas UDP is used for application that are **time sensitive but require less reliability.** *Nikita Education [NET]*
7	**What is URL, WWW?** URL - Uniform Resource Locator, WWW - World Wide Web.
8	**What is Socket?** Socket is a logical entity which provides end points to establish connection between processes or computers. Socket is a logical entity which consist IP address and port number to uniquely identify systems and processes over a network. **Types of Socket:** stream socket, datagram socket, raw socket.
9	**Socket primitives:** Socket, Bind, Listen, Accept, Connect, Send, Receive, Close. *Nikita Education [NET]*
10	**What is port?** It is a sixteen bit unique number used to uniquely identify processes over a network. It is also called as a numbered socket. A port number is the logical address of each application or process. **Maximum possible port numbers:** $2^{16} = 65536$ i.e. 0 to 65535
11	**Well known ports: 0 to 1023 i.e. 1024** **Registered ports: 1024 to 49151 i.e. 48128** **Dynamic or private ports: 49152 to 65535 i.e. 16383** *Nikita Education [NET]*
12	**What is ServerSocket?** As the name indicates, the ServerSocket works on server-side and its job is to bind the connection on the specific port number requested by the client.
13	**What is URL class?** An URL represents the host name or address along with the protocol. *Nikita Education [NET]*

14	**What is URLConnection class?** *Nikita Education [NET]* URLConnection object operates on the communication link established by the URL (client) with an application running on the server. The address of the resource on the server is referred by the URL on the client-side.
15	**What is InetAddress class?** Used to convert numeric IP addresses (URLs) to host names (URNs) and vice versa.
16	**What is a Protocol?** Protocol is a set of rules to be followed while communicating with systems in network. Depending on the task, different protocols are available. *Nikita Education [NET]*
17	**Enlist different protocols:** IP, TCP, UDP, SMTP, POP, Telnet, ARP, RARP, RSTP, FTP, TFTP, HTTP, HTTPS, SNMP, DHCP, BGP, LDAP etc.
18	**Enlist some common well known ports:** 20 - FTP Data 21 - FTP Control 80 - HTTP 23 - Telnet 25 - SMTP 161 - SNMP 53 - Domain Name System (DNS) 443 - HTTPS 546, 547 - DHCP Client, DHCP Server *Nikita Education [NET]*
19	**What is Factory and Instance methods?** **Factory methods** are the static methods of any class that returns the object of same class where these methods are defined. **Instance methods** are the non-static methods of any class which are defined in that class and accessed only by object of that class. **Factory methods of InetAddress class:** static InetAddress getLocalHost( ) throws UnknownHostException static InetAddress getByName(String hostName) throws UnknownHostException static InetAddress[] getAllByName(String hostName) throws UnknownHostException
20	**What are the four components of URL?**  *Nikita Education [NET]*
21	**How do we retrieves object of URLConnection?** **Syntax:** URLConnection openConnection( ) //method of URL class **Example:** URLConnection urlc = urlObject.openConnection()
22	**What is cookie?** Cookies are small files which are stored on a user's computer. They are designed to hold a modest amount of data specific to a particular client and website, and can be accessed either by the web server or the client computer. *Nikita Education [NET]*
23	**Enlist any two constructors of ServerSocket class.** public ServerSocket(int port) throws IOException public ServerSocket(int port, int backlog) throws IOException *Nikita Education [NET]*

24	**Enlist any two constructors of Socket class.** *Nikita Education [NET]* public Socket(String host, int port) throws UnknownHostException, IOException public Socket(InetAddress host, int port) throws IOException
25	**What is datagram?** A **datagram** is an independent, self-contained message sent over the network whose arrival, arrival time, and content are not guaranteed. It is a small piece of information transmitted over a connectionless protocol. Datagrams are bundles of information passed between machines.

04 - JDBC *Nikita Education [NET]*

1	**What is JDBC?** JDBC is a Java API that is used to connect and execute query to the database. JDBC API uses jdbc drivers to connects to the database.
2	**What is JDBC Driver?** JDBC Driver is a software component that enables java application to interact with the database. There are 4 types of JDBC drivers: 1. JDBC-ODBC bridge driver 2. Native-API driver (partially java driver) 3. Network Protocol driver (fully java driver) 4. Thin driver (fully java driver) *Nikita Education [NET]*
3	**What are the steps to connect to the database in java?** ○ Registering the driver class ○ Creating connection ○ Creating statement ○ Executing queries ○ Closing connection
4	**What are the JDBC API components?** *Nikita Education [NET]* The java.sql package contains interfaces and classes for JDBC API. **Interfaces:** ○ **Connection** ○ **Statement** ○ **PreparedStatement** ○ **ResultSet** ○ **ResultSetMetaData** ○ **DatabaseMetaData** ○ **CallableStatement etc.** **Classes:** ○ **DriverManager** ○ **Blob** ○ **Clob** ○ **Types** ○ **SQLException etc.** *Nikita Education [NET]*
5	**What are the JDBC statements?** 1. Statement 2. PreparedStatement 3. CallableStatement *Nikita Education [NET]*
6	**What is the difference between Statement and PreparedStatement interface?** In case of Statement, query is complied each time whereas in case of PreparedStatement, query is complied only once. So performance of PreparedStatement is better than Statement.
7	**How can we execute stored procedures and functions?** By using **Callable statement** interface, we can execute procedures and functions. *Nikita Education [NET]*
8	**What is the role of JDBC DriverManager class?** The **DriverManager class** manages the registered drivers. It can be used to register and unregister drivers. It provides factory method that returns the instance of Connection.
9	**What does the JDBC Connection interface?** The **Connection interface** maintains a session with the database. It can be used for transaction management. It provides factory methods that returns the instance of Statement, PreparedStatement, CallableStatement and DatabaseMetaData.
10	**What does the JDBC ResultSet interface?** The ResultSet object represents a row of a table. It can be used to change the cursor pointer and get the information from the database. *Nikita Education [NET]*

11	**What does the JDBC ResultSetMetaData interface?** *Nikita Education [NET]* The ResultSetMetaData interface returns the information of table such as total number of columns, column name, column type etc.
12	**What does the JDBC DatabaseMetaData interface?** The DatabaseMetaData interface returns the information of the database such as username, driver name, driver version, number of tables, number of views etc.
13	**Which interface is responsible for transaction management in JDBC?** The **Connection interface** provides methods for transaction management such as commit(), rollback() etc. *Nikita Education [NET]*
14	**What is batch processing and how to perform batch processing in JDBC?** By using batch processing technique in JDBC, we can execute multiple queries. It makes the performance fast.
15	**How can we store and retrieve images from the database?** By using PreparedStatement interface, we can store and retrieve images. *Nikita Education [NET]*
16	**What is front end?** It is a set of programs usually a graphical user interface (GUI) through which end user can interact with the application or system
17	**What is back end?** It is a background or behind the scene service or application usually a database that stores information passed by front end or responds to queries asked by front end *Nikita Education [NET]*
18	**Why JDBC is called as a mediator?** It performs different operations between Front end and Back end. JDBC passes front end method calls to back end in back end understandable formats and provides results back to front end in programming formats.
19	**Two Tier Database Design** The two-tier is based on Client Server architecture. In two tier database design user interface programs and application programs for database access run on client machine. The interface program called JDBC provides an API that allows application programs to call DBMS through JDBC drivers. The Two-tier architecture is divided into two parts: 1) Client Application (Client Tier) 2) Database (Data Tier)
20	**Two Tier Database Design** Advantages: 1. Easy to maintain and modification is bit easy 2. Communication is faster Disadvantages: 1. In two tier architecture application performance will be degrade upon increasing the users. 2. Cost-ineffective *Nikita Education [NET]*
21	**Three Tier Database Design:** Three-tier architecture typically comprises a presentation tier, a business or data access tier, and a data tier. Three layers in the three tier architecture are as follows: 1) Client layer 2) Business layer 3) Data layer
22	**Three Tier Database Design:** Advantages 1. High performance, lightweight persistent objects 2. Scalability – Each tier can scale horizontally Disadvantages 1. Increase Complexity/Effort *Nikita Education [NET]*
23	**JDBC Features** Get a connection, Connection Pooling, Rowsets, New data type supports, Batch Updating, Result set enhancement, Scrollable Result set, Updateable Result set, Savepoints.
24	**Simple JDBC program** class SimpleJdbcDemo{ public static void main(String ar[]){ try{ String url="jdbc:odbc:mydsn"; Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); Connection c=DriverManager.getConnection(url); Statement st=c.createStatement(); *Nikita Education [NET]*

```
                        ResultSet rs=st.executeQuery("select * from login");
                        while(rs.next()) {
                                System.out.println(rs.getString(1));
                        }
                }catch(Exception ee){
                        System.out.println(ee);
                }
        }
}
```

Nikita Education [NET]

05 - RMI

Nikita Education [NET]

1 **What is RMI?**

The RMI (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM.

2 **RPC (Remote Procedure Call) vs. RMI**

- RMI is object oriented while RPC isn't
- RPC is C bases while RMI is Java only
- RMI invokes methods while RPC invokes functions
- RPC is antiquated while RMI is the future

Nikita Education [NET]

3 **What is stub and skeleton in RMI?**

Stub: The stub is a client-side object that represents (or acts as a proxy for) the remote object. Stub is client side proxy.

Skeleton: On the server side, the skeleton object takes care of all the details of "remoteness". Skeleton is server side proxy

Nikita Education [NET]

4 **Marshaling and Unmarshaling (Earth hauling / Un-earth hauling)**

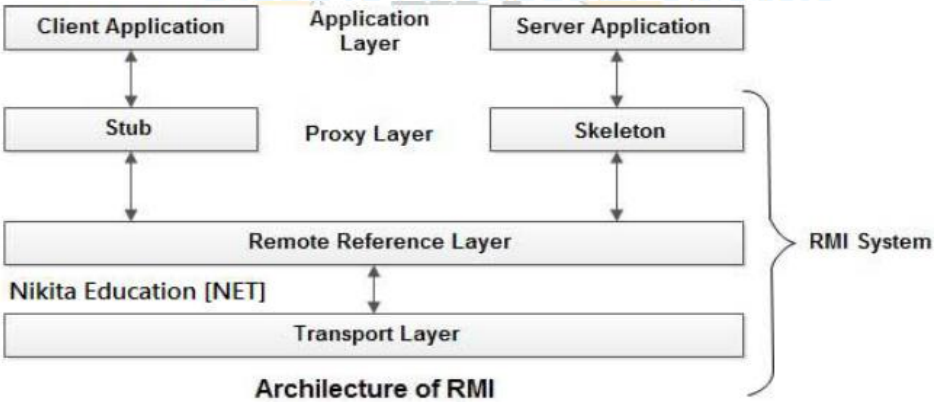
Marshaling is nothing but converting data into a special format suitable to pass through the distributed environment without losing object persistence. The stub marshals the data of client and then sends to the skeleton.

Unmarshaling is the process of extracting marshaled data into its original format. Skeleton performs the job of unmarshaling data received from stub.

5 **The Remote Reference Layer:** Proxies are implicitly connected to RMI mechanism through Remote reference layer, the layer responsible for object communication and transfer of objects between client and server.

Nikita Education [NET]

6 **Architecture of RMI**



Archilecture of RMI

Nikita Education [NET]

7 **Goals of RMI**

- Minimize difference between working with local and remote objects
- Minimize complexity
- Preserve type safety
- Distributed garbage collection
- Invocation of object methods in another java virtual machines
- Distribution transference (java semantics)
- Easy to use

Nikita Education [NET]

8	**Methods of Naming class** *Nikita Education [NET]* bind(String name, Remote obj) list(String name) lookup(String name) rebind(String name, Remote obj) unbind(String name)
9	**RMI Registry:** A remote object registry is a bootstrap naming service that is used byRMI servers on the same host to bind remote objects to names. Clients on local and remote hosts can then look up remote objects and make remote method invocations. **default port for RMI Registry is: 1099**
10	**rmic compiler:** The **rmic compiler generates stub and skeleton** class files (JRMP protocol) and stub and tie class files (IIOP protocol) for remote objects. These classes files are generated from compiled Java programming language classes *Nikita Education [NET]*
11	**Steps to develop RMI application** • Create the remote interface • Provide the implementation of the remote interface • Create Server application and register service with registry • Create client application
12	**Steps to execute RMI application** • compile all the java files : javac *.java • create stub and skeleton object by rmic tool : rmic AdderRemote • start rmi registry in one command prompt : rmiregistry 5000 • start the server in another command prompt : java -Djava.security.policy=sample.policy MyServer • start the client application in another command prompt : java -Djava.security.policy=sample.policy MyClient *Nikita Education [NET]*
13	**Write down simple remote interface for RMI application** import java.rmi.*; public interface Adder extends Remote { public int add(int x,int y)throws RemoteException; public int sub(int x,int y)throws RemoteException; }
14	**What is RMI security?** RMI security is a mechanism that provides security to the java applications as well as computer systems. This mechanism enforces different security policies in different execution environments using security policy files of java. *Nikita Education [NET]*
15	**The security manager** is the most important component in the sandbox; the role of the security manager is to perform run-time checks on potentially dangerous operations: □ File I/O □ Network access □ Creation on new class loaders etc. The manager reserves the right to verify any such operations. **System.setSecurityManager(new RMISecurityManager());**
16	**Security file** The java.security is master security file which enforces specified security parameters, properties, and policies for each java program running on JVM. *Nikita Education [NET]*
17	**The java.policy** file is a global default policy file shared by all of the Java programs running in the Java Virtual Machine (JVM) on the node.
18	**Specific Permission Policy File** grant { permission java.net.SocketPermission "*:1024-65535", "connect,accept,resolve"; permission java.net.SocketPermission "*:80", "connect"; *Nikita Education [NET]*

	permission java.awt.AWTPermission "accessEventQueue"; *Nikita Education [NET]* permission java.awt.AWTPermission "showWindowWithoutWarningBanner"; };
06 - Servlet	*Nikita Education [NET]*
1	**Static Web Page:** "Static" means unchanged or constant, static web pages contain the same prebuilt content each time the page is loaded.
2	**Dynamic Web Page:** "dynamic" means changing or lively, the content of dynamic Web pages can be generated on-the-fly. PHP, ASP, Servlets and JSP pages are dynamic Web pages.
3	**Web Application:** A web application is an application accessible from the web used to generate dynamic responses. A web application is composed of web components like Servlet, JSP, Filter etc. and other components such as HTML.
4	The **Hypertext Transfer Protocol (HTTP)** is an application-level protocol for distributed, collaborative, hypermedia information systems. *Nikita Education [NET]*
5	**HTTP Methods:** **GET**: The GET method is used to retrieve information from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data. **HEAD**: Same as GET, but it transfers the status line and the header section only. **POST**: A POST request is used to send data to the server, for example, customer information, file upload, etc. using HTML forms. **PUT:** Replaces all the current representations of the target resource with the uploaded content. **DELETE:** Removes all the current representations of the target resource given by URI. **CONNECT**: Establishes a tunnel to the server identified by a given URI. **OPTIONS**: Describe the communication options for the target resource. **TRACE**: Performs a message loop back test along with the path to the target resource. *Nikita Education [NET]*
6	**HTTP Response status codes** 1xx: Informational It means the request was received and the process is continuing. 2xx: Success It means the action was successfully received, understood, and accepted. 3xx: Redirection It means further action must be taken in order to complete the request. 4xx: Client Error It means the request contains incorrect syntax or cannot be fulfilled. 5xx: Server Error It means the server failed to fulfill an apparently valid request. *Nikita Education [NET]*
7	**HTTP content types** text/html text/plain application/msword application/vnd.ms-excel application/jar application/pdf application/octet-stream application/x-zip images/jpeg video/quicktime etc. *Nikita Education [NET]*
8	**What do you mean by HTTP is a stateless protocol?** The server and client are aware of each other only during a current request. Afterwards, both of them forget about each other. Due to this nature of the protocol, neither the client nor the browser can retain information between different requests across the web pages.
9	**Web Server:** A Web server is a program that, using the client/server model and the World Wide Web's Hypertext Transfer Protocol (HTTP), serves the files that form Web pages to Web users. **Application Server:** An application server is a program that handles all application operations between users and an organization's backend business applications or databases
10	Web servers: **Apache, Apache Tomcat**, **Resin**. *Nikita Education [NET]*

	Application servers: *Nikita Education [NET]* **JBoss** Open-source server. **Glassfish** provided by Oracle. **Weblogic** provided by Oracle. more secured. **Websphere** provided by IBM.
11	A **Web Container** is a J2EE compliant implementation which provides an environment for the Servlets and JSPs to run.
12	**What is Servlet?** Servlet technology is used to create web application (resides at server side and generates dynamic web page). Servet technology is robust and scalable as it uses the java language. *Nikita Education [NET]*
13	**CGI(Common Gateway Interface)** CGI technology enables the web server to call an external program and pass HTTP request information to the external program to process the request
14	**Servlet life cycle** • The servlet is initialized by calling the **init ()** method. • The servlet calls **service()** method to process a client's request. • The servlet is terminated by calling the **destroy()** method. • Finally, servlet is garbage collected by the garbage collector of the JVM. *Nikita Education [NET]*
15	**Methods of Servlet interface** public void init(ServletConfig config) public void service(ServletRequest request, ServletResponse response) public void destroy()
16	**Methods of ServletRequest interface** public String getParameter(String name) public int getContentLength() public String getContentType() public ServletInputStream getInputStream() throws IOException *Nikita Education [NET]*
17	**Methods of ServletResponse interface** PrintWriter getWriter() void setContentType(String type)
18	**Methods of GenericServlet class** public void init(ServletConfig config) public abstract void service(ServletRequest request, ServletResponse response) public void destroy() public ServletConfig getServletConfig() public ServletContext getServletContext() *Nikita Education [NET]*
19	**Enlist servlet exceptions:** ServletException, UnavailableException
20	**Methods of HttpServlet class** protected void doGet(HttpServletRequest req, HttpServletResponse res) protected void doPost(HttpServletRequest req, HttpServletResponse res) protected void doHead(HttpServletRequest req, HttpServletResponse res) protected void doOptions(HttpServletRequest req, HttpServletResponse res) protected void doPut(HttpServletRequest req, HttpServletResponse res) protected void doTrace(HttpServletRequest req, HttpServletResponse res) protected void doDelete(HttpServletRequest req, HttpServletResponse res) *Nikita Education [NET]*
21	**What is cookie?** Cookies are small pieces of information that are sent in response from the web server to the client. Cookies are the simplest technique used for storing client state. Cookies are stored on client's computer.
22	**Constructors of Cookie class** Cookie() Cookie(String name, String value) *Nikita Education [NET]*

23

Methods of Cookie class *Nikita Education [NET]*

public String getName()
public String getValue()
public void setName(String name)
public void setValue(String value)
public void addCookie(Cookie ck) {method of HttpServletResponse}
public Cookie[] getCookies() {method of HttpServletRequest}

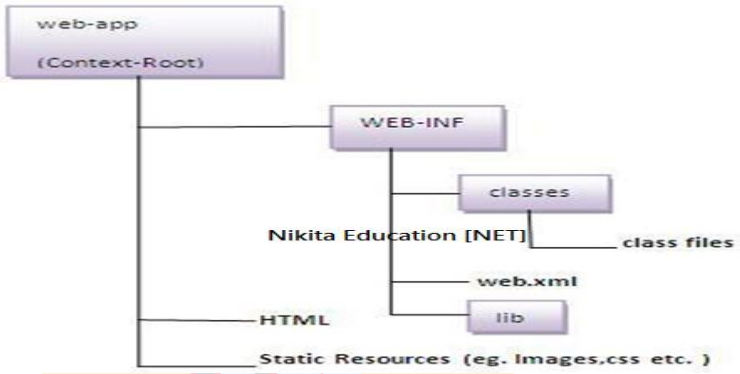
24

Steps to create a servlet

6. Create a directory structure
7. Create a Servlet
8. Compile the Servlet
9. Create a deployment descriptor
10. Start the server and deploy the project *Nikita Education [NET]*

25

Draw the directory structure for servlet



Nikita Education [NET]

26

How do we create HttpServlet? {by inheriting properties of HttpServlet class}

```
public class DemoServlet extends HttpServlet
{
        public void doGet(HttpServletRequest req, HttpServletResponse res) throws
                                                        ServletException, IOException
        {
                res.setContentType("text/html");
                PrintWriter pw=res.getWriter();
                pw.println("<html><head><title>");
                pw.println("Nikita Education [NET]");
                pw.println("</title></head>");
                pw.println("<body>");
                pw.println("Welcome to Nikita Education [NET]");
                pw.println("</body></html>");
                pw.close();
        }
}
```

Nikita Education [NET]

27

Which jar file is required for compiling servlet class in apache tomcat?

servlet-api.jar

28

What is deployment descriptor?

The deployment descriptor is an xml file, from which Web Container gets the information about the servlet to be invoked.

29

web.xml *Nikita Education [NET]*

```
<web-app>
        <servlet>
                <servlet-name> logical name for servlet  as it is given below</servlet-name>
                <servlet-class>name of servlet class file</servlet-class>
        </servlet>
        <servlet-mapping>
                <servlet-name> any logical name for servlet </servlet-name>
                <url-pattern>url pattern of request in browsers address bar</url-pattern>
        </servlet-mapping>
</web-app>
```

30	**What is RequestDispatcher?** *Nikita Education [NET]* The RequestDispacher interface provides the facility of dispatching the request to another resource it may be html, servlet or jsp. This interface can also be used to include the content of another resource also. **Methods of interface:** void forward(ServletRequest request, ServletResponse response) void include(ServletRequest request, ServletResponse response)
31	**How to get an Object of RequestDispatcher** getRequestDispatcher() method of **ServletRequest** returns the object of **RequestDispatcher**. **Syntax:**public RequestDispatcher getRequestDispatcher(String resource); *Nikita Education [NET]*
32	The **sendRedirect()** method of **HttpServletResponse** interface can be used to redirect response to another resource, it may be servlet, jsp or html file. It accepts relative as well as absolute URL.
33	**What is Session?** The session of activity that a user with a unique IP address spends on a Web site during a specified period of time. *Nikita Education [NET]*
34	**What is Session Management / Tracking?** Session tracking is the capability of a server to maintain the current state of a single client's sequential requests. The HTTP protocol used by Web servers is stateless. This means that every transaction is autonomous. This type of stateless transaction is not a problem unless you need to know the sequence of actions a client has performed while at your site. Session Management is a mechanism used by the Web container to store session information for a particular user. *Nikita Education [NET]*
35	**Session Tracking Techniques** ▪ **Cookies:** You can use HTTP cookies to store information. Cookies will be stored at browser side. ▪ **URL rewriting:** With this method, the information is carried through url as request parameters. In general added parameter will be sessionid, userid. ▪ **HttpSession:** Using HttpSession, we can store information at server side. Http Session provides methods to handle session related information. ▪ **Hidden form fields:** By using hidden form fields we can insert information in the webpages and these information will be sent to the server. These fields are not visible directly to the user, but can be viewed using view source option from the browsers. The hidden form fields are as given below: <input type='hidden' name='siteName' value='nspl'/> *Nikita Education [NET]*
36	**Types of Cookie** **1. Non-persistent cookie** It is **valid for single session** only. It is removed each time when user closes the browser. **2. Persistent cookie** It is **valid for multiple session** . It is not removed each time when user closes the browser. It is removed only if user logout or signout.
37	**sessions vs. cookies:** • a cookie is data stored on the client • a session's data is stored on the server (only 1 session per client) *Nikita Education [NET]*
38	**How to get the HttpSession object ?** The HttpServletRequest interface provides two methods to get the object of HttpSession: 3. **public HttpSession getSession():**Returns the current session associated with this request, or if the request does not have a session, creates one. 4. **public HttpSession getSession(boolean create):**Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session. *Nikita Education [NET]*
39	**Who is responsible to create the object of servlet?** The web container or servlet container. *Nikita Education [NET]*
40	**When servlet object is created?** At the time of first request. *Nikita Education [NET]*

41 **What is difference between Get and Post method?** *Nikita Education [NET]*

Get	Post
1) Limited amount of data can be sent because data is sent in header.	Large amount of data can be sent because data is sent in body.
2) Not Secured because data is exposed in URL bar.	Secured because data is not exposed in URL bar.
3) Can be bookmarked	Cannot be bookmarked
4) Idempotent	Non-Idempotent
5) It is more efficient and used than Post	It is less efficient and used

42 **What is difference between PrintWriter and ServletOutputStream?**
PrintWriter is a character-stream class where as ServletOutputStream is a byte-stream class. The PrintWriter class can be used to write only character-based information whereas ServletOutputStream class can be used to write primitive values as well as character-based information.
Nikita Education [NET]

43 **What is difference between GenericServlet and HttpServlet?**
The GenericServlet is protocol independent whereas HttpServlet is HTTP protocol specific. HttpServlet provides additional functionalities such as state management etc.

44 **Can you call a jsp from the servlet?**
Yes, one of the way is RequestDispatcher interface for example:

```
1. RequestDispatcher rd=request.getRequestDispatcher("/login.jsp");
2. rd.forward(request,response);
```

Nikita Education [NET]

45 **Difference between forward() method and sendRedirect() method ?**

forward() method	sendRedirect() method
1) forward() sends the same request to another resource.	1) sendRedirect() method sends new request always because it uses the URL bar of the browser.
2) forward() method works at server side.	2) sendRedirect() method works at client side.
3) forward() method works within the server only.	3) sendRedirect() method works within and outside the server.

46 **What is difference between ServletConfig and ServletContext?**
The container creates object of ServletConfig for each servlet whereas object of ServletContext is created for each web application.
Nikita Education [NET]

47 **What is difference between Cookies and HttpSession?**
Cookie works at client side whereas HttpSession works at server side.

48 **What is the disadvantage of cookies?**
It will not work if cookie is disabled from the browser.
Nikita Education [NET]

49 **What is war file?**
A war (web archive) file specifies the web elements. A servlet or jsp project can be converted into a war file. Moving one servlet project from one place to another will be fast as it is combined into a single file.

50 **How to create war file?**
The war file can be created using jar tool found in jdk/bin directory. If you are using Eclipse or Netbeans IDE, you can export your project as a war file.
To create war file from console, you can write following code.

```
1. jar -cvf abc.war *
```

Now all the files of current directory will be converted into abc.war file.
Nikita Education [NET]

07 - JSP *Nikita Education [NET]*

1 **What is JSP?**
Java Server Pages technology (JSP) is used to create dynamic web page. It is an extension to the servlet technology. A JSP page is internally converted into servlet.

2 **What are the life-cycle methods for a jsp?**
public void jspInit()
public void _jspService(ServletRequest request,ServletResponse)throws ServletException,IOException
public void jspDestroy()
Nikita Education [NET]

3

What are the JSP implicit objects ? *Nikita Education [NET]*

JSP provides 9 implicit objects by default. They are as follows:

Object	Type
1) out	JspWriter
2) request	HttpServletRequest
3) response	HttpServletResponse
4) config	ServletConfig
5) session	HttpSession
6) application	ServletContext
7) pageContext	PageContext
8) page	Object
9) exception	Throwable

Nikita Education [NET]

4

What is difference between include directive and include action?

include directive	include action
1) The include directive includes the content at page translation time.	1) The include action includes the content at request time.
2) The include directive includes the original content of the page so page size increases at runtime.	2) The include action doesn't include the original content rather invokes the include() method of Vendor provided class.
3) It's better for static pages.	3) It's better for dynamic pages.

5

How can we handle the exceptions in JSP ?

There are two ways to perform exception handling, one is by the errorPage element of page directive, and second is by the error-page element of web.xml file.

6

What are the two ways to include the result of another page. ?

There are two ways to include the result of another page:

- By include directive
- By include action

Nikita Education [NET]

7

How can we forward the request from jsp page to the servlet ?

Yes of course! With the help of forward action tag, but we need to give the url-pattern of the servlet.

8

Can we use the exception implicit object in any jsp page ?

No. The exception implicit object can only be used in the error page which defines it with the isErrorPage attribute of page directive.

9

What are the different scope values for the <jsp:useBean> tag?

1. page
2. request
3. session
4. application

Nikita Education [NET]

10

What is EL in JSP?

The Expression Language(EL) is used in JSP to simplify the accessibility of objects. It provides many objects that can be used directly like param, requestScope, sessionScope, applicationScope, request, session etc.

11

What is JSTL?

JSP Standard Tag Library is library of predefined tags that ease the development of JSP.

Nikita Education [NET]

12

What are the 3 tags used in JSP bean development?

1. jsp:useBean
2. jsp:setProperty
3. jsp:getProperty

13

How to disable session in JSP?

1. <%@ page session="false" %>

Nikita Education [NET]

14

Importance of JSP

- Performance is significantly better because JSP allows embedding Dynamic Elements in HTML Pages itself instead of having a separate CGI files.
- JSP are always compiled before it's processed by the server unlike CGI/Perl which requires the server to

load an interpreter and the target script each time the page is requested.

- Java Server Pages are built on top of the Java Servlets API, so like Servlets; JSP also has access to all the powerful Enterprise Java APIs, including JDBC, JNDI, EJB, JAXP etc.
- JSP pages can be used in combination with servlets that handle the business logic, the model supported by Java servlet template engines.

Nikita Education [NET]

15

Advantages of JSP

- **vs. Active Server Pages (ASP):** The advantages of JSP are twofold. First, the dynamic part is written in Java, not Visual Basic or other MS specific language, so it is more powerful and easier to use. Second, it is portable to other operating systems and non-Microsoft Web servers.
- **vs. Pure Servlets:** It is more convenient to write (and to modify!) regular HTML than to have plenty of println statements that generate the HTML.
- **vs. Server-Side Includes (SSI):** SSI is really only intended for simple inclusions, not for "real" programs that use form data, make database connections, and the like.
- **vs. JavaScript:** JavaScript can generate HTML dynamically on the client but can hardly interact with the web server to perform complex tasks like database access and image processing etc.
- **vs. Static HTML:** Regular HTML, of course, cannot contain dynamic information.
- **Extension to Servlet:** JSP technology is the extension to servlet technology. We can use all the features of servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.
- **Easy to maintain:** JSP can be easily managed because we can easily separate our business logic with presentation logic. In servlet technology, we mix our business logic with the presentation logic.
- **Fast Development: No need to recompile and redeploy:** If JSP page is modified, we don't need to recompile and redeploy the project. The servlet code needs to be updated and recompiled if we have to change the look and feel of the application.
- **Less code than Servlet:** In JSP, we can use a lot of tags such as action tags, jstl, custom tags etc. that reduces the code. Moreover, we can use EL, implicit objects etc.

Nikita Education [NET]

16

The javax.servlet.jsp package

Interfaces	Classes	
3. JspPage 4. HttpJspPage	• JspWriter • PageContext • JspFactory	• JspEngineInfo • JspException • JspError

17

In JSP, java code can be written inside the jsp page using different JSP elements called as **scripting elements**. Following are the three different scripting elements of jsp:

- scriptlet tag
- expression tag
- declaration tag

Nikita Education [NET]

18

JSP scriptlet tag

A scriptlet tag is used to execute java source code in JSP. Syntax is as follows:

```
<% java source code %>
```

Nikita Education [NET]

19

JSP expression tag

The code placed within expression tag is written to the output stream of the response. So you need not write out.print() to write data. It is mainly used to print the values of variable or method.

```
<%= statement %>
```

Nikita Education [NET]

20

JSP Declaration Tag

The JSP declaration tag is used to declare fields and methods. The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet. So it doesn't get memory at each request.

```
<%! field or method declaration %>
```

Nikita Education [NET]

21

Scriplet vs. Declaration tag *Nikita Education [NET]*

Jsp Scriptlet Tag	Jsp Declaration Tag
The jsp scriptlet tag can only declare variables not methods.	The jsp declaration tag can declare variables as well as methods.
The declaration of scriptlet tag is placed inside the _jspService() method.	The declaration of jsp declaration tag is placed outside the _jspService() method.

08 - EJB

Nikita Education [NET]

1

What is EJB?
EJB stands for Enterprise Java Bean. It is a server-side component to develop scalable, robust and secured enterprise applications in java.

2

What are the types of Enterprise Bean?
There are 3 types of enterprise bean in java.

1. Session Bean
2. Message Driven Bean
3. Entity Bean

Nikita Education [NET]

3

What is session bean?
Session Bean encapsulates business logic. It can be invoked by local, remote or web service client.
There are 3 types of session bean.

1. Stateless Session Bean
2. Stateful Session Bean
3. Singleton Session Bean

4

What is stateless session bean?
Stateless session bean is a business object that doesn't maintain conversational state with the client.

Nikita Education [NET]

5

What is stateful session bean?
Stateful session bean is a business object that maintains conversational state with the client.

6

What is singleton session bean?
Singleton session bean is instantiated only once for the application. It exists for the life cycle of the application.

Nikita Education [NET]

7

What is JMS?
Java Message Service is a messaging service to create, send and receive messages asynchronously.

8

What are the advantages of JMS?

- Asynchronous
- Reliable

Nikita Education [NET]

9

What is MDB?
Message Driven Bean (MDB) encapsulates business logic. It is invoked by passing message. It is like JMS receiver.

10

What is Entity Bean?
Entity Bean is a server side component that represents the persistent data. Since EJB 3.x, it is replaced by JPA.

Nikita Education [NET]

11

Features of EJB

- Transaction processing
- Integration with the persistence services offered by the Java Persistence API (JPA)
- Concurrency control
- Event-driven programming using Java Message Service and Java EE Connector Architecture
- Asynchronous method invocation
- Job scheduling
- Naming and directory services (JNDI)

Nikita Education [NET]

12

RMI	EJB	Web Service
In RMI, middleware services such as security, transaction management, object pooling etc. need to be done by the java programmer.	In EJB, middleware services are provided by EJB Container automatically.	In EJB, bean component and bean client both must be written in java language. If bean client need to be written in other language such as **.net**, **php** etc, we need to go with **webservices** (SOAP or REST). So EJB with web service will be better option.
RMI is not a server-side component. It is not required to be deployed on the server.	EJB is a server-side component; it is required to be deployed on the server.	
RMI is built on the top of socket programming.	EJB technology is built on the top of RMI.	

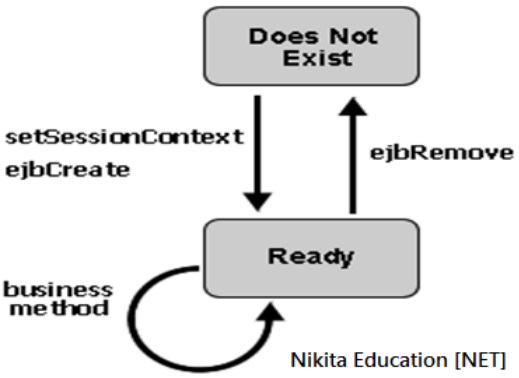
13

Limitations of EJB *Nikita Education [NET]*

4. Requires application server
5. Requires only java client. For other language client, you need to go for web service.
6. Complex to understand and develop ejb applications.

14

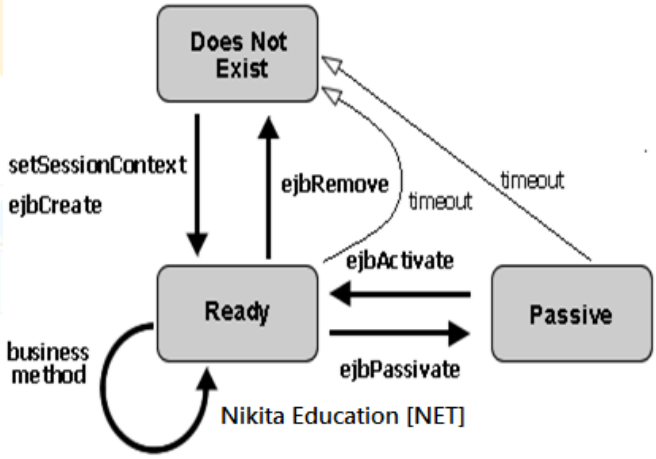
Life Cycle Stateless Session bean



Nikita Education [NET]

15

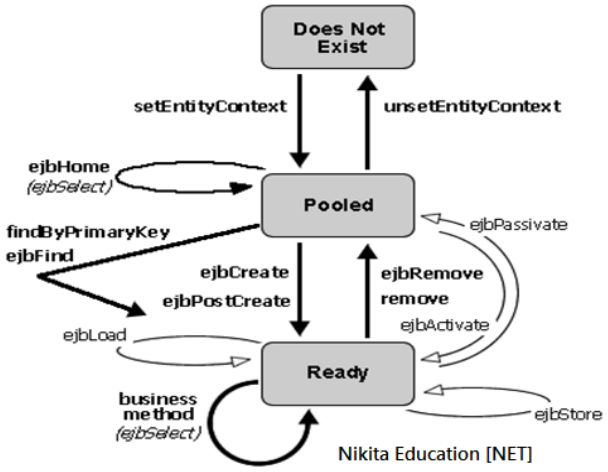
Life Cycle Stateful Session bean



Nikita Education [NET]

16

Stateless Session Beans	Stateful Sessions Beans
Are pooled in memory, to save the overhead of creating a bean every time one is needed. WebLogic Server uses a bean instance when needed and puts it back in the pool when the work is complete. Stateless sessions beans provide faster performance than stateful beans.	Each client creates a new instance of a bean, and eventually removes it. Instances may be passivated to disk if the cache fills up. An application issues an ejbRemove() to remove the bean from the cache. Stateful sessions beans do not perform as well as stateless sessions beans.
Have no identity and no client association; they are anonymous.	Are bound to particular client instances.Each bean has an implicit identity. Each time a client interacts with a stateful session bean during a session, it is the same object.
Do not persist. The bean has no state between calls.	Persist. A stateful session bean's state is preserved for the duration of a session.

17	**Life Cycle Entity Beans** *Nikita Education [NET]*  *Nikita Education [NET]*
18	**Features of Session Beans** *Nikita Education [NET]* ▪ Execute for a single client. ▪ Can be transaction aware. ▪ Do not represent directly shared data in an underlying database, although they may access and update this data. ▪ Are short lived. ▪ Are not persisted in a database. ▪ Are removed if the container crashes; the client has to establish a new session.
19	**Features of Entity Beans** *Nikita Education [NET]* ▪ Entity beans provide an object view of persistent data. ▪ Entity beans are transactional. ▪ Entity beans are multiuser. ▪ Entity beans are long-lived. ▪ Entity beans survive container crashes. Such crashes are typically transparent to the clients.
20	**Features of Message Driven Beans** ▪ They execute upon receipt of a single client message. ▪ They are invoked asynchronously. ▪ They are relatively short-lived. ▪ They do not represent directly shared data in the database, but they can access and update this data. ▪ They can be transaction-aware. ▪ They are stateless. *Nikita Education [NET]*
21	**Steps to create EJB application** 1. Component development a. Describe Remote interface b. Describe Home interface c. Implement the Bean class 2. Write deployment descriptor(s) 3. Package in an archive (jar file) all EJB files 4. Deployment into the container 5. Implement the client application *Nikita Education [NET]*

Best of Luck

Nikita Education [NET]