CardLayout(int horz, int vert)

The following example creates a two-level card deck that allows the user to select an operating system. Windows-based operating systems are displayed in one card. Macintosh and Solaris are displayed in the other card.

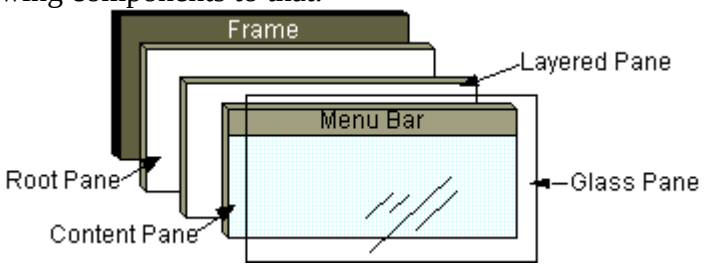
```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*<applet code="CardLayoutDemo" width=300 height=100> </applet>*/
public class CardLayoutDemo extends Applet implements ActionListener, MouseListener
{
Checkbox Win98, winNT, solaris, mac;
Panel osCards;
CardLayout cardLO;
Button Win, Other;
public void init() {
Win = new Button("Windows");
Other = new Button("Other");
add(Win);
add(Other);
cardLO = new CardLayout();
osCards = new Panel();
osCards.setLayout(cardLO);
Win98 = new Checkbox("Windows 98", null, true);
winNT = new Checkbox("Windows NT");
solaris = new Checkbox("Solaris");
mac = new Checkbox("MacOS");
Panel winPan = new Panel();
winPan.add(Win98);
winPan.add(winNT);
Panel otherPan = new Panel();
otherPan.add(solaris);
otherPan.add(mac);
osCards.add(winPan, "Windows");
osCards.add(otherPan, "Other");
add(osCards);
Win.addActionListener(this);
Other.addActionListener(this);
addMouseListener(this);
}
public void mousePressed(MouseEvent me) {
cardLO.next(osCards);
}
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() == Win) {
cardLO.show(osCards, "Windows");
}
else {
cardLO.show(osCards, "Other"); } } }
```

Output expected as per the program.

2 a **Explain Root Pane, Glass Pane, Layered Pane, Content Pane and Desktop Pane**

Swing offers some top-level containers such as - JApplet, JDialog, and JFrame. There are some problems for mixing lightweight and heavyweight components together in Swing, we can't just add anything but first, we must get something called a "content pane," and then we can add Swing components to that.



The Root Pane :

We don't directly create a JRootPane object. As an alternative, we get a JRootPane when we instantiate JInternalFrame or one of the top-level Swing containers, such as JApplet, JDialog, and JFrame. It's a lightweight container used behind the scenes by these top-level containers. As the preceding figure shows, a root pane has four parts:

I)The layered pane: It Serves to position its contents, which consist of the content pane and the optional menu bar. It can also hold other components in a specified order. **JLayeredPane** adds depth to a JFC/Swing container, allowing components to overlap each other when needed.It allows for the definition of a several layers within itself for the child components.

JLayeredPane manages its list of children like Container, but allows for the definition of a several layers within itself.

Ii) The content pane: The container of the root pane's visible components, excluding the menu bar.

iii) The optional menu bar: It is the home for the root pane's container's menus. If the container has a menu bar, we generally use the container's **setJMenuBar** method to put the menu bar in the appropriate place.

iv) The glass pane: It is hidden, by default. If we make the glass pane visible, then it's like a sheet of glass over all the other parts of the root pane. It's completely

transparent.The glass pane is useful when we want to be able to catch events or paint over an area that already contains one or more components. We can display an image over multiple components using the glass pane.

JdesktopPane:

The concept of showing multiple windows inside a large frame is implemented using Desktop pane. If we minimize the application frame, all of its windows are hidden at the same time. In

Windows environment, this is called as the **multiple document interface**or MDI.

Using it we can resize the internal frames inDesktop pane by dragging the resize corners.To achieve this we have follow these steps:

1. We can use a regular JFrame window for the program.
2. Set the content pane of the JFrame to a JDesktopPane.

2 b **Distinguish Between AWT & JFC**

JFC (Swing)is a huge set of components which includes labels, frames, tables, trees, and styled text documents. Almost all Swing components are derived from a single parent called **JComponent** which extends the **AWT** Container class. Swing is a layer on top of AWT rather than a substitution for it.

AWT	Swing
AWT stands for Abstract windows toolkit.	Swing is also called as JFC's (Java Foundation classes).
AWT components are called Heavyweight component.	Swings are called light weight component because swing components sits on the top of AWT components and do the work.
AWT components require java.awt package.	Swing components require javax.swing package.
AWT components are platform dependent.	Swing components are made in purely java and they are platform independent.
This feature is not supported in AWT.	We can have different look and feel in Swing.
These feature is not available in AWT.	Swing has many advanced features like JTabel, Jtabbed pane which is not available in AWT. Also. Swing components are called "lightweight" because they do not require a native OS object to implement their functionality. JDialog and JFrame are heavyweight, because they do have a peer. So components like JButton, JTextArea, etc., are lightweight because they do not have an OS peer.
With AWT, you have 21 "peers" (one for each control and one for the dialog itself). A "peer" is a widget provided by the operating system, such as a button object or an entry field object.	With Swing, you would have only one peer, the operating system's window object. All of the buttons, entry fields, etc. are drawn by the Swing package on the drawing surface provided by the window object. This is the reason that Swing has more code. It has to draw the button or other control and implement its behavior instead of relying on the host operating system to perform

		those functions.
	AWT is a thin layer of code on top of the OS. Using AWT, you have to implement a lot of things yourself.	Swing is much larger. Swing also has very much richer functionality. Swing has them built in.
2c	**Explain JpopupMenu Class with Example** This class provides an implementation of the Java Accessibility API appropriate to popup menu user-interface elements. Popup menu is a small window that pops up and displays a series of choices. A JPopupMenu is used for the menu that appears when the user selects an item on the menu bar. It is also used for "pull-right" menu that appears when the selects a menu item that activates it. A JPopupMenu can also be used anywhere else we want a menu to appear. For example, when the user right-clicks in a specified area. **Constructor:** ***JPopupMenu( )*** : It constructs a JPopupMenu without an "invoker". ***JPopupMenu(String)*** : It constructs a JPopupMenu with the specified title. **Method:** ***add(Action)*** : This method appends a new menu item to the end of the menu which dispatches the specified Action object. ***add(JMenuItem)***: This method appends the specified menu item to the end of this menu. ***addPopupMenuListener(PopupMenuListener)*** This method adds a PopupMenu listener. ***addSeparator( )*** : This method appends a new separator at the end of the menu. **Example:** ***import javax.swing.*;*** ***import java.awt.event.*;*** ***public class TPopUpMenu{*** ***JPopupMenu Pmenu;*** ***JMenuItem menuItem;*** ***public static void main(String[] args) {*** ***TPopUpMenu p = new TPopUpMenu();*** ***}*** ***public TPopUpMenu(){*** ***JFrame frame = new JFrame("Creating a Popup Menu");*** ***frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);*** ***Pmenu = new JPopupMenu();*** ***menuItem = new JMenuItem("Cut");*** ***Pmenu.add(menuItem);*** ***menuItem = new JMenuItem("Copy");*** ***Pmenu.add(menuItem);***	

```
menuItem = new JMenuItem("Paste");
Pmenu.add(menuItem);
menuItem.addActionListener(new ActionListener(){
public void actionPerformed(ActionEvent e){}
});
frame.addMouseListener(new MouseAdapter(){
public void mouseReleased(MouseEvent Me){
if(Me.isPopupTrigger()){
Pmenu.show(Me.getComponent(), Me.getX(), Me.getY());
}
}
});
frame.setSize(400,400);
frame.setVisible(true);
}
}
```

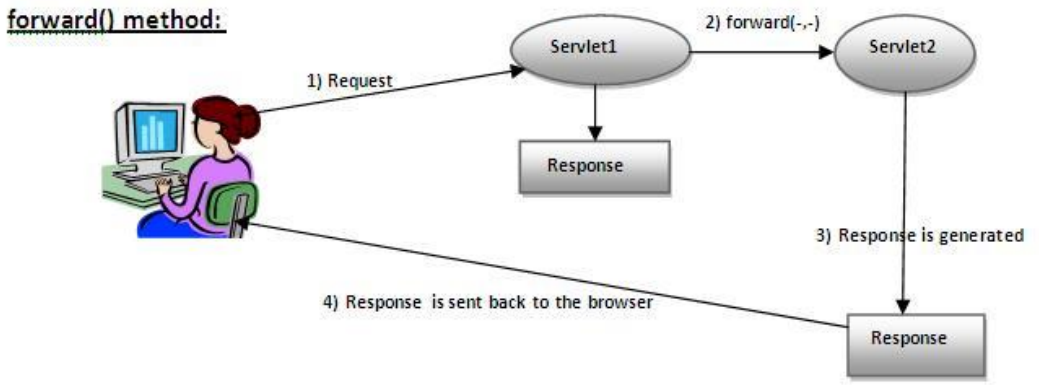
2 d **Write a program using JTree to generate the following hierarchical structure.**

```
import java.awt.BorderLayout;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JScrollPane;
import javax.swing.JTree;
import javax.swing.SwingUtilities;
import javax.swing.event.TreeSelectionEvent;
import javax.swing.event.TreeSelectionListener;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeCellRenderer;


public class TreeExample extends JFrame
{
   private JTree tree;
   private JLabel selectedLabel;

   public TreeExample()
   {
      //create the root node
      DefaultMutableTreeNode root = new DefaultMutableTreeNode("Root");
      //create the child nodes
      DefaultMutableTreeNode sem5Node = new DefaultMutableTreeNode("Semester
5");
      sem5Node.add(new DefaultMutableTreeNode("Network Security"));
```

```
        sem5Node.add(new DefaultMutableTreeNode("ASP.NET with C#"));
        sem5Node.add(new DefaultMutableTreeNode("Software Testing"));
        sem5Node.add(new DefaultMutableTreeNode("Advanced Java"));
        sem5Node.add(new DefaultMutableTreeNode("Linux Administration"));

        DefaultMutableTreeNode sem6Node = new DefaultMutableTreeNode("Semester
6");
        sem6Node.add(new DefaultMutableTreeNode("Internet Technologies"));
        sem6Node.add(new DefaultMutableTreeNode("Project Management"));
        sem6Node.add(new DefaultMutableTreeNode("Data Warehousing"));
        sem6Node.add(new DefaultMutableTreeNode("Elective"));

        //add the child nodes to the root node
        root.add(sem5Node);
        root.add(sem6Node);

        //create the tree by passing in the root node
        tree = new JTree(root);

        ImageIcon imageIcon = new
ImageIcon(TreeExample.class.getResource("/leaf.jpg"));
        DefaultTreeCellRenderer renderer = new DefaultTreeCellRenderer();
        renderer.setLeafIcon(imageIcon);

        tree.setCellRenderer(renderer);
        tree.setShowsRootHandles(true);
        tree.setRootVisible(false);
        add(new JScrollPane(tree));

        selectedLabel = new JLabel();
        add(selectedLabel, BorderLayout.SOUTH);

        tree.getSelectionModel().addTreeSelectionListener(new TreeSelectionListener() {
            @Override
            public void valueChanged(TreeSelectionEvent e) {
                System.out.println(e.getPath().toString());
                DefaultMutableTreeNode selectedNode = (DefaultMutableTreeNode)
tree.getLastSelectedPathComponent();
                selectedLabel.setText(selectedNode.getUserObject().toString());
            }
        });

        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setTitle("JTree Example");
        this.setSize(200, 200);
        this.setVisible(true);
```

	}
	
	public static void main(String[] args)
	{
	SwingUtilities.invokeLater(new Runnable() {
	@Override
	public void run() {
	new TreeExample();
	}
	});
	}
	}

3 a	**What are servlets? What are the advantages of servlet over CGI?** **What are Servlet?** A Java servlet is a server side program that services HTTP requests and return the results as HTTP responses. A good analogy for a servlet is a non visual applet and runs on a webserver.it has a lifecycle similar to that of an applet and runs inside a JVM at the web server.The **javax.Servlet** and **javax.Servlet.http** packages provide interfaces and classes for writing servlets. All servlets must implement the Servlet interface, which defines lifecycle methods. When implementing a generic service, we can use or extend the **GenericServlet** class provided with the Java Servlet API. The **HttpServlet** class provides methods, such as **doGet()** and **doPost()**, for handling HTTP-specific services. **What are the advantages of servlet over CGI?** • **Servlets are loaded into memory once and run from memory thereafter.** • **Servlets are swapped as a thread, not as a process.** • **Servlets are powerful object oriented abstraction of http.** • **Servlets are portable across multiple web servers and platforms.** • **Servlets are tightly integrated with web server.** • **Servlets run within the secure and reliable scope of JVM** • **Servlets provide direct database access using native and ODBC based Db drivers.** • **Being on the server side provide code protection.** • **Servlets are robust, scalable, secure CGI replacement.**
3 b	**What is request Dispatcher? What are its two methods?** **RequestDispatcher in Servlet** 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 antoher resource also. It is one of the way of servlet collaboration. There are two methods defined in the RequestDispatcher interface.

	Methods of RequestDispatcher interface The RequestDispatcher interface provides two methods. They are: 1. **public void forward(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException**:Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server. 2. **public void include(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException**:Includes the content of a resource (servlet, JSP page, or HTML file) in the response. 

3 c	**Explain Generic Servlets with its constructors and methods.**
	Generic Servlets
	• It extends **javax.Servlet.GenericServlet**.
	• They are protocol independent. They contain no inherent HTTP support or any other transport protocol.
	The class *GenericServlet* provides a fundamental implementation of the Servlet interface for protocol independent Servlets. It also implements the *ServletConfig* interface. *GenericServlet* includes **init()** and **destroy()** methods. The **service()** method is declared as abstract and must be overridden.
	Class Structure:
	public abstract class GenericServlet
	implements Servlet, ServletConfig, java.io.Serializable {
	// Constructors
	public GenericServlet();
	// We can add here instance Methods
	…….
	}
	Constructor:
	1) public GenericServlet(): The default GenericServlet constructor does not do any work. Any Servlet initialization tasks should be performed in **init()** method, rather than in the constructor.

	Methods: **1) public void destroy():** This method is called by the server before unloading the Servlet. A Servlet can override this method to save its state, free its resources, etc. **2) public String getInitParameter(String name):** This method returns the value of the named Servlet initialization parameter or null if no matching parameter is found. **3) getInitParameterNames():**This method returns all the Servlets init parameter names as an enumeration of String objects or an empty Enumeration if no parameters exist. **4) public ServletConfig getServletConfig():**This method returns the Servlets *ServletConfig* object. This method is not often called by a *GenericServlet* because all of the *ServletConfig* methods are duplicated internally. **5) public ServletContext getServletContext():**This method returns the Servlets *ServletContext* object. **6) public String getServletInfo():**This method returns a developer defined String that describes the Servlet. A Servlet should override this method and provide a personalized identity string. **7) public void init(ServletConfig config) throws ServletException:** This method is called by the server after the Servlet is first loaded and before the Servlets **service()** method is called. It logs the Servlets initialization and stores the *ServletConfig* object for use by the methods in the *ServletConfig* interface. A Servlet should always call the super class implementation of *init*() using **super.init(config)** before executing any custom initialization code. **8)public void log(String msg):** This method writes the specified message to a Servlet log in an event log file. The output format and location are server-specific with this method. **9) public abstract void service(ServletRequest req, ServletResponse res) throws ServletException, IOException :** This method is called to handle a single client request. A Servlet gets request information through a *ServletRequest* object and sends data back to the client through a *ServletResponse* object. This is the only method that must be overridden when extending *GenericServlet*.
3 d	**Write a servlet application to find the sum of digits of the number entered by the user through HTML form.** Step-1. Create a HTML file that displays a single text box and a Submit button. Step-2. Create a new servlet. Step-3. Run the servlet and get its URL, put the URL of the servlet in the forms action attribute. Step-4. In the processRequest method of servlet get the value entered in the HTML form by the user using getParameter() method. Step-5. Convert the value into integer value and write the logic for finding the sum of the digits. Steps-6. Compile the servlet. Step-7. Run the HTM file enter any value and submit, sum of the digits for the value will be printed on browser. **(i) Servlet Code:-** import java.io.*; import java.net.*; import javax.servlet.*;

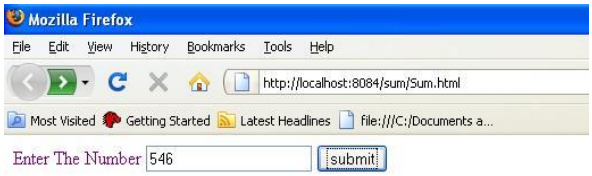
```
import javax.servlet.http.*;
public class Sum extends HttpServlet {
   protected void processRequest(HttpServletRequest request, HttpServletResponse
response)
   throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      Print Writer out = response.getWriter();
      String n=request.getParameter("number");
      out.println("<html>");
      out.println("<head>");
      out.println("<title>Servlet Sum</title>");
      out.println("</head>");
      out.println("<body>");
      int j=1,i,m=0;
      int n1=Integer.parseInt(n)*j;
      do
      {
         i=n1%10;
         m=m+i;
         n1=n1/10;
      }
      while(n1>0);
      out.println("The Sum Of Digit Is"+m);
      out.println("</body>");
      out.println("</html>");
      out.close();
   }
}
```

(ii) HTML Code:-

```
<html>
  <head>
    <title></title>
  </head>
  <body>
  <form method="get" action="http://localhost:8084/sum/Sum">
     Enter The Number
     <Input type="text" name = "number">
     <input type ="submit" name="button" value="submit">
  </form>
  </body>
</html>
```

OUTPUT:-

(i) Sum Of Digits:-



(ii) <u>Sum Of Digits</u>:-

4 a **What is JDBC driver? Explain the types of JDBC Drivers.**

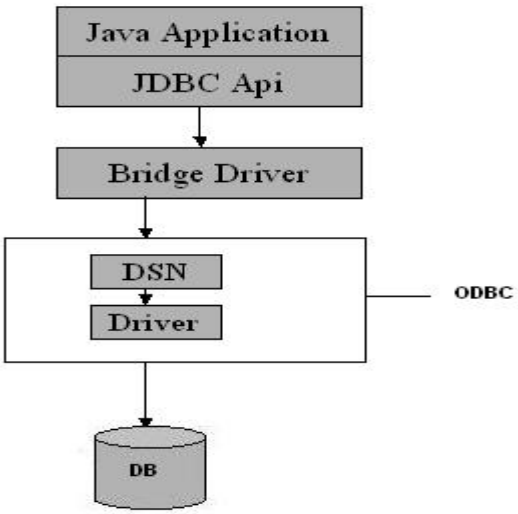
JDBC driver

The JDBC API defines the Java interfaces and classes that programmers use to connect to databases and send queries. A JDBC driver implements these interfaces and classes for a particular DBMS vendor. The JDBC driver converts JDBC calls into a network or database protocol or into a database library API call that makes communication with the database. This translation layer provides JDBC applications with database autonomy. In the case of any back-end database change, only we need to just replace the JDBC driver &some code modifications are required. "The Java program that uses the JDBC API loads the specified driver for a particular DBMS before it actually connects to a database. After thatthe JDBC DriverManager class then sends all JDBC API calls to the loaded driver".

There are four types of JDBC drivers as follows:

1) Type 1- JDBC-ODBC Bridge plus ODBC driver:

This driver converts JDBC API calls into Microsoft Open Database Connectivity (ODBC) calls that are then passed to the ODBC driver. The ODBC binary code must be loaded on every client computer that uses this type of driver.



Type 1: JDBC-ODBC Bridge

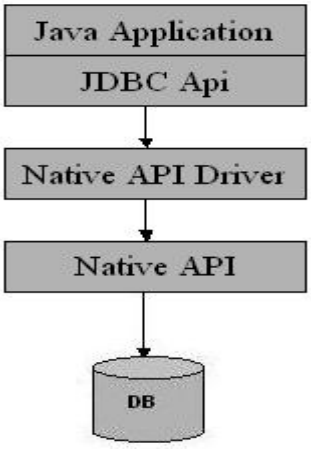
Advantage: The JDBC-ODBC Bridge allows access to almost any database, since the database's ODBC drivers are already available.

Disadvantages:

- The Bridge driver is not coded completely in Java; So Type 1 drivers are not portable.
- It is not good for the Web Application because it is not portable.
- It is comparatively slowest thanthe other driver types.
- The client system requires the ODBC Installation to use the driver.

2) Type 2 - Native-API/ Partly Java driver:

This driver converts JDBC API calls into DBMSprecise client API calls. Similar to the bridge driver, this type of driver requires that some binary code be loaded on each client computer.



Native API/Partly Java Driver

Advantage: This type of divers are normally offer better performance than the JDBC-ODBC Bridge as the layers of communication are less than that it and also it uses Resident/NativeAPI which is Database specific.

Disadvantage:

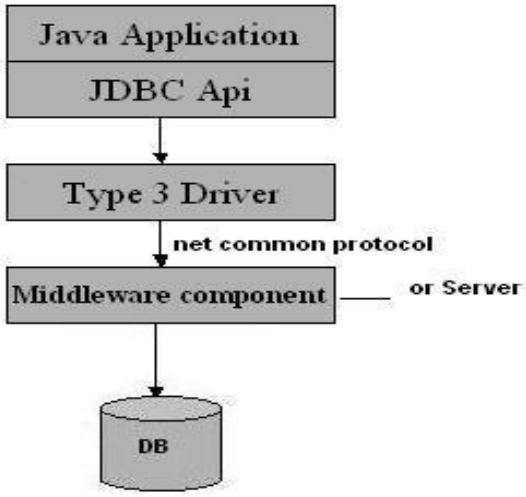
- Mostly out of date now.

- It is usually not thread safe.
- This API must be installed in the Client System; therefore this type of drivers cannot be used for the Internet Applications.
- Like JDBC-ODBC Bridgedrivers, it is not coded in Java which cause to portability issue.
- If we modify the Database then we also have to change the Native API as it is specific to a database.

3) Type 3 - JDBC-Net/Pure Java Driver:
This driver sends JDBC API calls to a middletier Net Server that translates the calls into the DBMSprecise Network protocol. The translated calls are then sent to a particular DBMS.

Advantage:

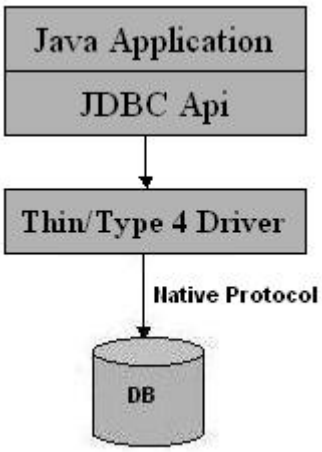
- This type of drivers is the most efficient amongst all driver types.
- This driver is totally coded in Java and hence Portable. It is suitable for the web Applications.
- This driver is serverbased, so there is no need for any vendor database library to be present on client machines.
- With this type of driver 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.
- This normally provides support for features such as caching, load balancing etc.
- Provides facilities for System administration such as logging and auditing.
- This driver is very flexible allows access to multiple databases using one driver.

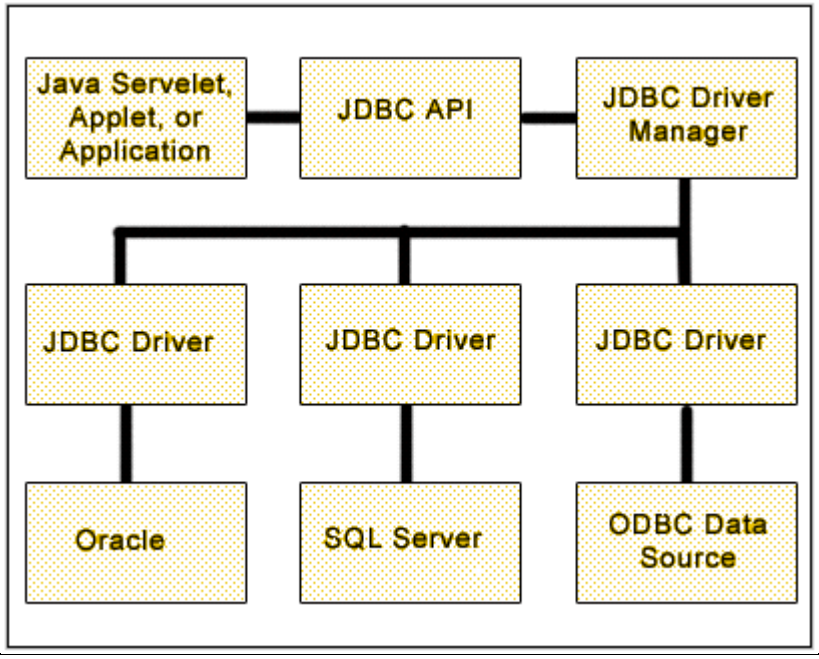


JDBC-Net/Pure Java Driver

Disadvantage : This driver It requires another server application to install and maintain. Traversing the recordset may take longer, since the data comes through the back-end server.

4)Type 4 - Native-protocol/Pure Java driver:
This driver converts JDBC API calls directly into the DBMSprecise network protocol

	without a middle tier. This allows the client applications to connect directly to the database server.  **Native-protocol/Pure Java driver** **Advantage:** • The Performance of this type of driver is normally quite good. • This driver is completely written in Java to achieve platform independence and eliminate deployment administration issues. It is most suitable for the web. • In this driver number of translation layers are very less i.e. type 4 JDBC drivers don't need to translate database requests to ODBC or a native connectivity interface or to pass the request on to another server. • We don't need to install special software on the client or server. • These drivers can be downloaded dynamically. **Disadvantage:** With this type of drivers, the user needs a different driver for each database.
4 b	**Explain JDBC architecture.** JDBC is a Java API for executing SQL statements and supports basic SQL functionality. The JDBC (Java Database Connectivity) is an API that defines interfaces and classes for writing database applications in Java by making database connections. It is a program designed to access many popular database products on a number of operating system platforms.Using JDBC we can send SQL, PL/SQL statements to almost any relational database. It provides RDBMS access by allowing us to embed SQL inside Java code. **JDBC Architecture** The main function of the JDBC API is to provide a means for the developer to issue SQL statements and process the results in a consistent, databaseindependently. JDBC provides wealthy object-oriented access to databases by defining classes and interfaces that represent objects such as: • Database connections : **(Short explanation Expected)** • SQL statements : **(Short explanation Expected)** • Result Set: **(Short explanation Expected)** • Database metadata: **(Short explanation Expected)**

	• Prepared statements: **(Short explanation Expected)** • Binary Large Objects (BLOBs): **(Short explanation Expected)** • Character Large Objects (CLOBs): **(Short explanation Expected)** • Callable statements : **(Short explanation Expected)** • Database drivers: **(Short explanation Expected)** • Driver manager: **(Short explanation Expected)** The JDBC API uses a Driver Manager and databaseprecise drivers to provide clear 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. **Layers of the JDBC Architecture** 
4 c	**What are the advantages and disadvantages of Java Server pages?** JavaServer Pages (JSP) is a serverside development technology that is used to create dynamic web pages and applications. It is introduced after Java Servlets. With Servlets Java became a full fledged application server programming language. This is achieved by embedding Java code into HTML, XML, DHTML, or other document types. When a client such as a web browser makes a request to the Java application container, which is typically a web server, the static page is converted behind the scenes, and displayed as dynamic content to the viewer. **1 Advantage of JSP:** • The JSP serves all facilities of Java i.e. write once run anywhere. • JSP is ideal for Web based Technology. • The JSP pages are translated and compiled into JAVA Servlet but are easier to develop than JAVA Servlet. • The JSP uses simplified scripting language based syntax for embedding HTML into JSP.

	• JSP containers provide easy coding for accessing standard objects and actions. • JSP acquire all the benefits provided by JAVA Servlets and web container environment. • The JSP use HTTP as default request /response communication model. **2: Disadvantage:** • The JSP implementation is normally causes for poor diagnostics. • Difficult looping in jsp. • The space used to store JSP page is comparatively more. • The first time loading of JSP is little bit time consuming because JSP pages must be compiled on the server when first accessed.
4 d	**Write a JSP application that computes the cubes of the numbers from 1 to 10. Simple code is expected with the output.**
5 a	**Explain Model-view-controller architecture.** **MVC Architecture** The MVC stands for Model, View, and Controller architecture. The MVC architecture separates the business logic and application data from the presentation data to the user.MVC is design pattern which allows a developer to write their applications in a specific format, following the same directory structure, using the same configuration, allowing making unique chain between the components & documents of the application. Core parts of MVC architecture. 1) **Model:** The model object only represents the data of an application. The model object knows about all the data that need to be displayed. The model is aware about all the operations that can be applied to transform that object. The model represents enterprise data and the business rules that govern access to and updates of this data. Model is not concern about the presentation data and how that data will be displayed to the browser. 2) **View:** The view represents the presentation of the application. The view object refers to the model. It uses the query methods of the model to obtain the contents and renders it. The view is not dependent on the application logic. It remains same if there is any modification in the business logic. It is the responsibility of the view's to maintain the consistency in its presentation when the model (data or logic) changes. 3) **Controller:** Whenever the user sends a request for something then it always go through the controller. The controller is responsible for intercepting the requests from view and passes it to the model for the appropriate action. After the action has been taken on the data, the controller is responsible for directing the appropriate view to the user. In GUI applications the views and the controllers often work very closely together.
5 b	**Explain in detail phases of JSF life cycle.** **JSF lifecycle** To understand how the framework treats the underlying request & Servlet API also how Faces processes each request, we'll go through the JSF request processing lifecycle. A Java Server Faces page is represented by a tree of UI components, called a view. During the lifecycle, the Java Server Faces implementation must build the view while

considering state saved from a previous submission of the page. When the client submits a page, the Java Server Faces implementation performs several tasks, such as validating the data input of components in the view and converting input data to types specified on the server side. The Java Server Faces implementation performs all these tasks as a series of steps in the Java Server Faces request response life cycle.

JSF Life cycle handles two kinds of requests:
Initial request: A user requests the page for the first time.
Postback: A user submits the form contained on a page that was previously loaded into the browser as a result of executing an initial request.

The phases of the JSF application lifecycle are as follows:

Phase 1: Restore view
In this phase, JSF classes build the tree of UI components for the incoming request. When a request for a JavaServer Faces page is made, such as when a link or a button is clicked, the JavaServer Faces implementation begins the restore view phase.
The JSF framework controller uses the view ID means a name of JSP to look up the components for the current view. If the view isn't available, the JSF controller creates a new one. If the view already exists, the JSF controller uses it. The view contains all the GUI components and there is a great deal of state management by JSF to track the status of the view – typically using HTML hidden fields.

Phase 2: ApplyRequest values
In this phase, the request parameters are examined and their values are used to set the values of the corresponding UI components. This process is called decoding. If the conversion of the value fails, an error message associated with the component is generated. This message will be displayed during the render response phase, along with any validation errors resulting from the process validations phase.

Phase 3: Process validations
In this phase triggers calls to all registered Validators.The components validate the new values coming from the request against the application's validation rules. Any input can be scanned by any number of Validators.These Validators can be predefined or defined by the developer. Any validation errors will abort the requesthandling process and skip to rendering the response with validation and conversion error messages.

Phase 4: Update Model Values
The Update Model phase brings a transfer of state from the UI component tree to any and all backing beans, according to the value expressions defined for the components themselves. In this phase converters are invoked to parse string representations of various values to their proper primitive or object types. If the data cannot be converted to the types specified by the bean properties, the life cycle calls directly to the render response phase so that the page is re-rendered with errors displayed.
In Apply Request Values phase, it moves values from client side HTML form controls to server side UI components; while Update model values phase the information moves from

	the UI components to the backing beans. **Phase 5: Invoke Application** This phase handles any application level events. Normally this takes the form of a call to process the action event generated by the submit button that the user clicked. In this phase… • Application level events are handled • Application methods are invoked • Navigation outcome are calculated **Phase 6: Render Response** The Render Response Phase brings several contrary behaviors together in one process like values are transferred back to the UI components from the bean including any modifications that may have been made by the bean itself or by the controller; The UI components save their state, not just their values, but other attributes having to do with the presentation itself. This can happen at serverside, but by default state is written into the HTML as hidden input fields and thus returns to the JSF implementation with the next request. If the request is a Postback and errors were encountered during the apply request values phase, process validations phase, or update model values phase, the original page is rendered during this phase. If the pages contain message or messages tags, any queued error messages are displayed on the page.
5 c	**Enumerate benefits of Enterprise beans.** **Benefits of EJB**: 1. Complete focus only on business logic. 2. Reusable components 3. Portable 4. Fast building of application 5. One business logic having many presentation logic. 6. Distributed deployment. 7. Application interoperability.
5 d	**Write a session bean code specification that.....** *SICalculator.java File* This java file comes under ejbModule under EJB project 1. 2. import javax.annotation.PostConstruct; 3. import javax.annotation.PreDestroy; 4. import javax.ejb.Remote; 5. import javax.ejb.Stateless; 6. 7. /** 8. * Session Bean implementation class SICalculator 9. */ 10. @Stateless 11. // This annotation create stateless session bean 12. public class SICalculator {

```
    @PostConstruct
    // This annotation invoked method before
    // the first business method is invoked
    // on the enterprise bean and after all
    // dependency injection has completed.
       void show() {
           System.out.println
           ("SICalculator started...");
       }
       @PreDestroy
       // This annotation invoke method, when
       // the bean is about to be destoryed by
       // EJB container
       void remove(){
           System.out.println
           ("SessionEjbBean ended...");
       }
    public double caculateSI
    (double priciple, double rate, int time ) {
       return ((priciple * rate * time)/100) ;
    }

}
```

SessionServlet.java File

This java(servlet) file comes under web project src folder

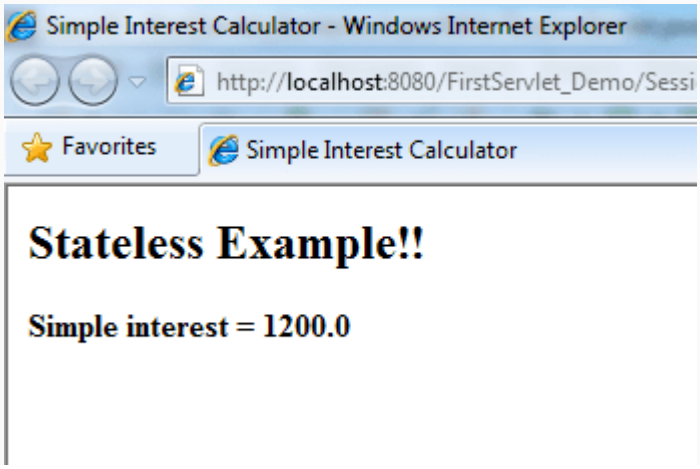
view plaincopy to clipboardprint?

```
package com.tkhts;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.List;

import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class SessionServlet
 */
@WebServlet("/SessionServlet")
public class SessionServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    @EJB // @EJB is used to inject EJB's
    SICalculator calculator;
    protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws
        ServletException, IOException {
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println(">title<Login Page</title>");
        out.println("<body>");
        out.println("<h2>Stateless Example!!</h2>");
        try {
            out.println("<h4> Simple interest =
```

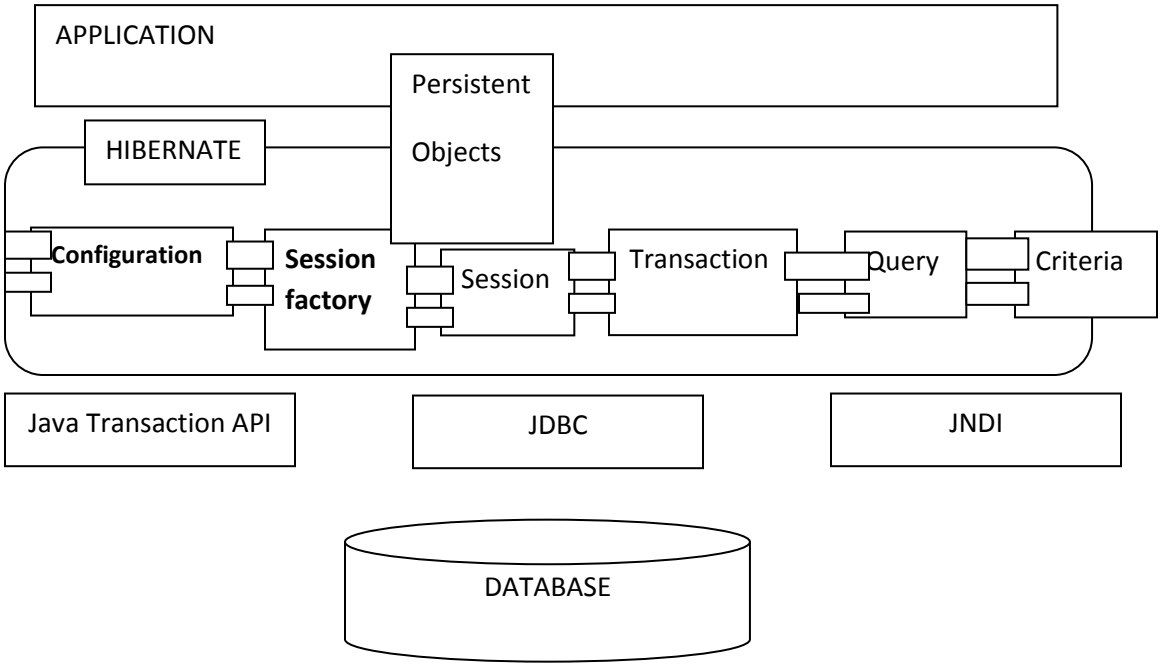
```
            "+calculator.caculateSI(20000, 3, 2)+"</h4>");
        }
        catch(Exception e){
            e.printStackTrace();
        }
        out.println("</body>");
        out.println("</html>");
    }
}
```

Note :Now make new Enterprise Application project and add both EJB and Web project and run the Enterprise Application project for output.

Output



6 a **Explain Architecture of Hibernate framework in detail.**



	Hibernate architecture has three main components as follows: **1 Connection Management:** Hibernate Connection management service provide well-organized management of the database connections. Database connection is the most expensive part of interacting with the database as it requires a lot of resources of open and close the database connection. **2: Transaction management:** Transaction management service of hibernate provides the ability to the user to execute more than one database statements at a time. **3 Object relational mapping:** Object relational mapping is technique of mapping the data representation from an object model to a relational data model. This part of hibernate is used to select, insert, update and delete the records form the underlying table. When we pass an object to a **Session.save()** method, Hibernate reads the state of the variables of that object and executes the necessary query. **4 configuration object** **The configuration object represents a configuration or properties file for Hibernate. It is usually created once during application initialization.** **5.Session factory** **The seession factory is created with the help of a configuration object during the application start up.it serves as as factory for spawning session objects when required.** **6. session** **Session objects are light weight and inexpensive to create.** **They provide the main interface to perform actual database operation.** **7.Transaction** **Transaction represents a unit of work with the database.** **8.Query** **Persistent objects are retrieved using a query object.** **9.Criteria** **Persistent object can also be retrieved using a criteria object.**
6 b	**Explain struture of hibernate.cfg.xml file** Hibernate uses the "**hibernate.cfg.xml**" file to create the connection & setup the required environment.

	This file contains information such as….. 1) Database Connection 2) Resource mapping **hibernate.cfg.xml** <?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd"> <hibernate-configuration> <session-factory> <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property> <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property> <property name="hibernate.connection.url"> jdbc:mysql://localhost:3306/Feedback</property> <property name="hibernate.connection.username">root</property> <property name="hibernate.connection.password">root</property> </session-factory> </hibernate-configuration> **Explaination:** 1. **hibernate.connection.driver_class:** It is the JDBC connection class for the specific database. 2. **hibernate.connection.url:** It is the full JDBC URL to the database. 3. **hibernate.connection.username:** It is the username used to connection the database. 4. **hibernate.connection.password:** It is the password used to authenticate the username. 5. **hibernate.dialect:** It is the name of SQL dialect for the database.
6 c	**What is value stack in struts? State and exaplin the execution flow of value stack.** **Value Stack:**Value Stack is nothing but stack of objects. The Value Stack is a storage area that holds all of the data associated with the processing of a Request. **Execution Flow of Value Stack:** 1. The framework receives a request and decides on the action the URL maps to 2. The framework moves the data to the Value Stack whilst preparing for request processing 3. The framework manipulates the data as required during the action execution 4. The framework reads the data from there and renders the results i.e. Response page struts 2 maintains a stack of the following objects in the **Value Stack:** **1) Temporary Objects:** Theseare generated and placed in the **Value Stack** during execution. These objects provide temporary storage and are normallygenerated while processing a request.For example, the current iteration value for a collection being looped over in JSP tag.

	2) The Model Object: If the application user Model objects, the current model object is placed in the Value Stack before the Action executes. **3) The Action Object:** It is the Action that is currently being executed. **4) Named Objects:** Any object assigned to an identifier called as a **Named Object.**These objects can either be created by the developer of pre-defined such as #application **Accessing Value Stack :**The **Value Stack** can be accessed by simply using the **tags** provided for JSP. • When the Value Stack Is queried for an **attribute value,** each stack element, in the provided order, is asked whether it holds the queried property. • If it holds the queried property, then the value is returned. • If it does not hold the queried property, then the next element down is queried. This continues until the last element in the stack is scanned.
6 d	**Explain interceptors in struts.** **Interceptors** Interceptors allow developing code that can be run before and/or after the execution of an action. A request is usually processed as follows: • A user requests a resource that maps to an action • The Struts 2 framework invokes the appropriate action to serve the request If interceptors are written, available and configured, then: • **Before the action is executed** the invocation could be intercepted by another object • **After the action executes**, the invocation could be intercepted again by another object • Such objects who intercept the invocation are called **Interceptors.**Conceptually, interceptors are very similar to **Servlet Filters** or the JDKs **Proxy class**. **Interceptor Configuration:** Interceptors configured in the struts.xml file appear as: <interceptors> <interceptor name="test1" class="..."/> <interceptor name=”test2" class="..."/> </interceptors> <action name="WelcomeSturts">

```
        <interceptor-ref name=" test 1"/>
        <interceptor-ref name=" test 2"/>
            <result name="SUCCESS">/ Welcome.jsp</result>
            <result name="ERROR">/ Error.jsp</result>
    </action>
```

In above code two interceptors named test1 and test2 are defined. Both of these are them mapped to the action named WelcomeSturts.

Interceptor Stack: We can bind Interceptor together using an Interceptor **Stack** which can be referenced together. We can use the same set of interceptors multiple times. So, instead of configuring a number of interceptors every time, an interceptor stack can be configured with all the required interceptors held within. To use Intercept Stack we need to modify the struts.xml file as follows:

```
<interceptors>
        <interceptor name="test1" class="..."/>
        <interceptor name= test2" class="..."/>
        <interceptor-stack name="MyStack">
            <interceptor-ref name=" test1"/>
            <interceptor-ref name=" test2"/>
        </interceptor-stack>
</interceptors>
<action name="WelcomeSturts " class="test.WelcomeSturts ">
        <result name="SUCCESS">/ Welcome.jsp</result>
        <result name="ERROR">/ Error.jsp</result>
</action>
```

In above code two interceptors named **test1**and **test2** are defined and a stack named **MyStack**group them both.The stack holding both these interceptors is then mapped to the Action named **WelcomeStruts.**

Execution Flow of Interceptors: Interceptors are executed as follows:

1. The framework receives a request and decides on the action the URL maps to
2. The framework consults the application's configuration file, to discover which interceptors should fire and in what sequence
3. The framework starts the invocation process by executing the first Interceptor in the **Stack**
4. After all the interceptors are invoked, the framework causes the action itself to be **executed.**
5.

7 a **What is CheckBoxGroup? Explain with Example.**

A *check box* is a control that is used when there are multiple options and multiple selections to turn an option on or off. It consists of a small box that can either contain a check mark or not.

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***. Check box groups are objects of type **CheckboxGroup**. Only the default constructor is defined, which creates an empty group.

Methods:

Checkbox getSelectedCheckbox( ) : Reads the selected radio button option from group

void setSelectedCheckbox(Checkbox which): Selects radio button option of a group.

Example:

```
import java.applet.*;
import java.awt.*;
/*
<applet code="TCBCBG" width=200 height=150>
</applet>
*/
public class TCBCBG extends Applet {
Checkbox morning,noon,evening;
   public void init() {
   this.add(new Label("How will you pay for your pizza?"));

 CheckboxGroup cbg = new CheckboxGroup();
   this.add(new Checkbox("Visa", cbg, false));
   this.add(new Checkbox("Cash", cbg, true)); // the default
   this.add(new Label("Select Time Slots for Delivery"));

  morning = new Checkbox("Morning", null, true);
   noon = new Checkbox("Afternoon");
   evening = new Checkbox("Evening");

   this.add(morning);
   this.add(noon);
   this.add(evening);
  }
}
```

7 b **Explain JScrollPane and JScrollBar with example.**

JScrollPane

Provides a scrollable view of a lightweight component. A JScrollPane manages a viewport, optional vertical and horizontal scroll bars, and optional row and column heading viewports. You can find task-oriented documentation ofJScrollPane in How to Use Scroll Panes, a section in *The Java Tutorial*. Note that JScrollPane does not support heavyweight components.

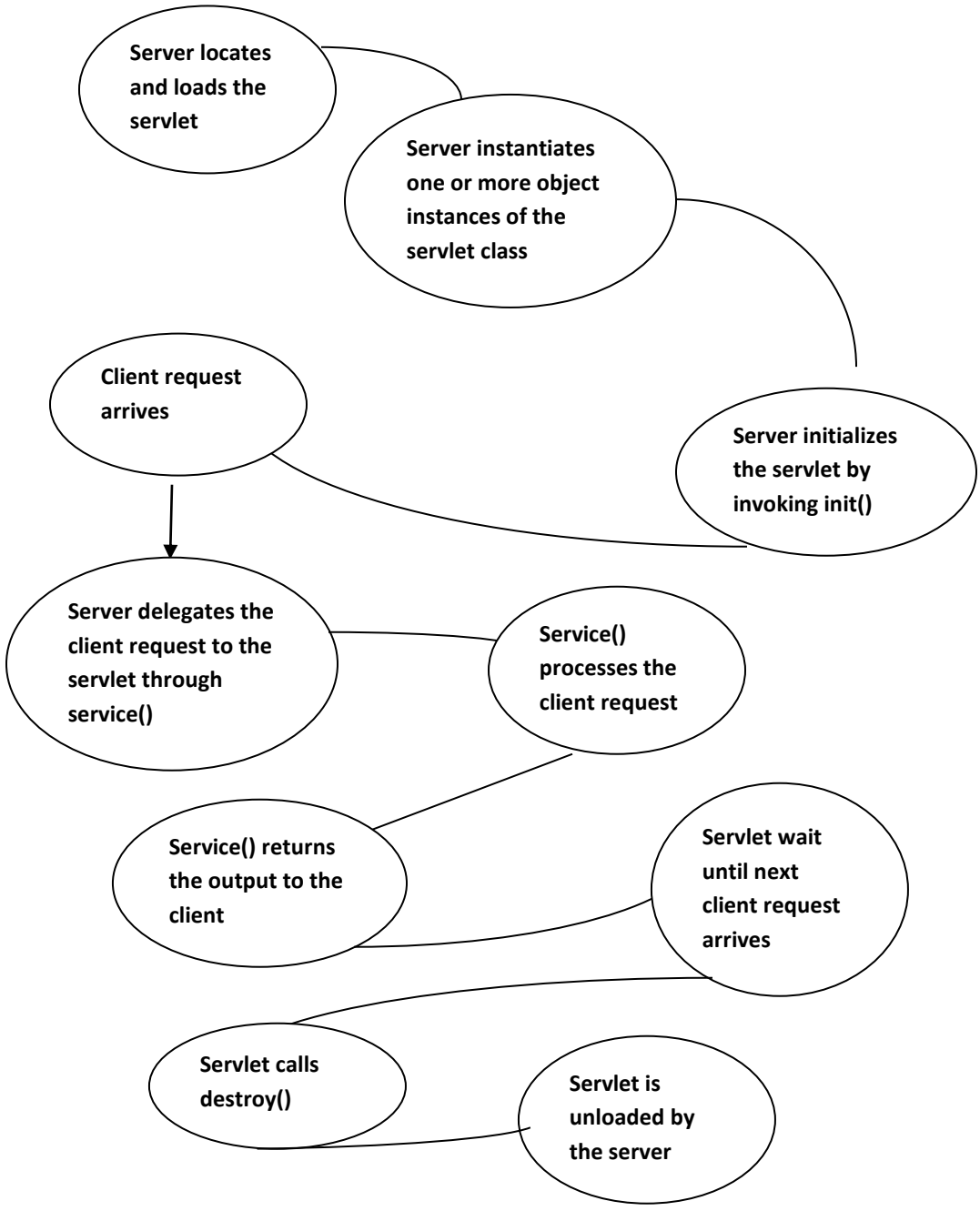
Example

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

class ScrolledPane extends JFrame
{
        Private JScrollPane scrollPane;

        public ScrolledPane()
        {
                setTitle( "Scrolling Pane Application" );
                setSize( 300, 200 );
                setBackground( Color.gray );

                JPanel topPanel = new JPanel();
                topPanel.setLayout( new BorderLayout() );
                getContentPane().add( topPanel );

                Icon image = new ImageIcon( "main.gif" );
                JLabel label = new JLabel( image );

                // Create a tabbed pane
                scrollPane = new JScrollPane();
                scrollPane.getViewport().add( label );
                topPanel.add( scrollPane, BorderLayout.CENTER );
        }


        public static void main( String args[] )
        {
                // Create an instance of the test application
                ScrolledPane mainFrame     = new ScrolledPane();
                mainFrame.setVisible( true );
        }
}
```

	JScrollBar: The JScrollBar lets the user graphically select a value by sliding a knob within a bounded interval. It is us a very useful component when we want to display large amount of elements on the screen. **Constructors:** **JScrollBar ( ):** It Creates a JScrollBar instance with a range of 0-100, an initial value of 0, and vertical orientation. **JScrollBar (int Orientation) :** It creates a JScrollBar instance with a range of 0-100, an initial value of 0, and the specified orientation. **Methods :** ***addAdjustmentListener (AdjustmentListener Handler)*** : Use to handle the scroll bar events. ***getValue ( )*** : It gives the current value of scroll bar. ***setBackground (Color BackgroundColor)*** : Sets the color to scroll bar background. ***setMaximum (int Max)*** :The maximum value of the scrollbar is maximum – extent. ***setMinimum (int Min)*** : Returns the minimum value supported by the scrollbar (usually zero). ***setValue (int Value)*** :Sets the scrollbar's value.
7 C	**Explain the life cycle of servlet** The following are the life cycle methods of a Servlet instance:. **1: init()** This method is called once for a Servlet instance. When first time Servlet is called, Servlet container creates instance of that Servlet and loaded into the memory. Future requests will be served by the same instance without creating the new instance. **init()** method is used for inilializing Servlet variables which are required to be passed from the deployment descriptor web.xml. *ServletConfig* is passed as the parameter to **init()** method which stores all the values configured in the web.xml. **2: service()** This method is called for the each request. This is the entry point for the every Servlet request and here we have to write ourbusinesslogic or any other processes. This method takes *HttpServletRequest* and *HttpServletresponse* as the parameters.

3: destroy()

This method will be called once for an instance. It is used for releasing any resources used by the Servlet instance. Most of the times it could be database connections, file IO operations, etc. destroy () is called by the container when it is removing the instance from the Servlet container.



SERVLET LIFE CYCLE

7 d	**Write a JDBC program that inserts values in database. [table Name : Employee, Fileds : Empid, Name, Dept, Designation]**
7 e	**What are the difftrent typres of enterprise beans? Explain.** Enterprise java beans are reusable modules of code that combine related tasks into well defined interface. These enterprise beans EJB components contain the method that executes business logic and access data sources. **1: Session beans:** **A) Stateful Session Beans:** Stateful Session Beans are business objects having state, means they can keep track of which calling client they are dealing with throughout a session and thus access to the bean instance is strictly limited to "only one client at a time". In the case of concurrent access to a single bean is attempted anyway the container serializes those requests, but via the **@AccessTimeout** annotation the container can throw an exception instead. Stateful session beans' state may be persisted automatically by the container to free up memory after the client hasn't accessed the bean for some time. **Example:** The hotel check out may be handled by a stateful session bean that would use its state to keep track of where the customer is in the checkout process, possibly holding locks on the items the customer is charged for services. **B) Stateless Session Beans:** Stateless Session Beans are business objects that do not have state associated with them. Access to a single bean instance is limited to only one client at a time and thus concurrent access to the bean is banned. In case concurrent access to a single bean is attempted anyway the container simply routes each request to a different instance. Instances of Stateless Session beans are typically pooled. If a second client accesses a specific bean right after a method call on it made by a first client has finished, it might get the same instance. **Example:** Sending an Email to customer support may be handled by a stateless bean since this is a oneoff operation and not part of a multistep process. **C) Singleton Session Beans:** Singleton Session Beans are business objects having a global shared state in a JVM. Concurrent access to the one and only bean instance can be controlled by the container or by the bean itself. Container Managed concurrency can be tuned using the **@Lock** annotation, that designates whether a read lock or a write lock will be used for a method call. Also the Singleton Session Beans can explicitly request to be instantiated when the EJB container starts up, using the **@Startup** annotation. **Examples:** Loading a daily price list that will be the same for every user might be done with a singleton session bean, since this will prevent the application having to do the same query to a database over and over again. **14.4.2: Message Driven Beans:** Message Driven Beans are business objects whose execution is happened by the messages instead by method calls. Like session beans, a Message Driven Beans does not have any type of client view, i.e. clients cannot lookup a Message Driven Beans instance. It just listens for any incoming message on, for example, a JMS queue or topic and processes them automatically. Message Driven Beans can support many messaging protocols. The

	difference between session- and message driven beans is only in method calling and messaging. **Example:** Submitting a job to a work bunch might be done by sending a JMS message to a 'message queue' and could also be handled by a Message Driven Bean.
7 f	**What is OGNL? Explainthe execution flow of OGNL.** **OGNL [Object-Graph Navigation Language]** The Object-Graph Navigation Language is a fully featured expression language for **Retrieving** and **Setting** properties of the Java object s. It helps **data transfer** and **type conversion.** In the **Value Stack**,**Searching** or **evaluating**, a particular **expression** can be done using OGNL. OGNL provides a mechanism to navigate object graphs using a **dot notation** and **evaluate** expressions, including calling methods on the objects being retrieved. OGNL supportsType conversion, calling methods, Collectionmanipulation, Generation, Projection across collections, Expression evaluation, Lambda expressions etc **OGNL Examples** The following are a few examples where OGNL is used: • **emp.name** It returns the value that is actually returned when **getEmp().getName()** is Invoked • **emp.toSpring()** It returns the value that is actually returned when **getEmp().toString()** is invoked. • **@test.auth.name@fName()** It returns the value that is actual returned when the static method named fName() is invoked on the class name. • **firstName in {"Sharanam","vaishali"}** invokes getfName() and determines he the value returned is either Tushar or Sonali. If it is, then returns True. **Execution flow of OGNL :** 1. A user enters the data in and data entry form and submits the form 2. The Request enters the Struts 2 framework and is made available to the Java Language as an **HttpServletRequest** object. **3.** The request paramets are stored as name/value paris where the name are the Names of the data entry from's text fields and the Value are the Value entered by the user when the form is submitted **4.** Now the OGNL comes into picture , to handle the transfer and type conversion of the data from these request parameters **5.** Using the OGNL expression , the value stack is scanned to locate the destination property where the data has to be moved **6.** On locating the correct property in the Value Stack, the data is moved to the

	property by invoking the property's SETTER method with the appropriate value. The value stack acts as a place holder for viewing the data model throughout the execution **7.** Whilst moving such data, OGNL consults its set of available type converters to determine if any of them can handle this particular conversion, if a conversion is required. The value is converted and set on the object's property. This makes the data available them action begins its job, immediately after the available Interceptors ate fired. **8.** After Action completes to its job successfully, a Result fires that renders the result they to the user. **9.** Results have access to the Value Stack, via the OGNL expression language with the help of tags. These tags retrieve data from the Value Stack by referencing specific values using OGNL expressions. **10.** Whilst rendering the view , once again, the value that is accessed from the value Slack is converted from the java type to a String that is written on the HTML page