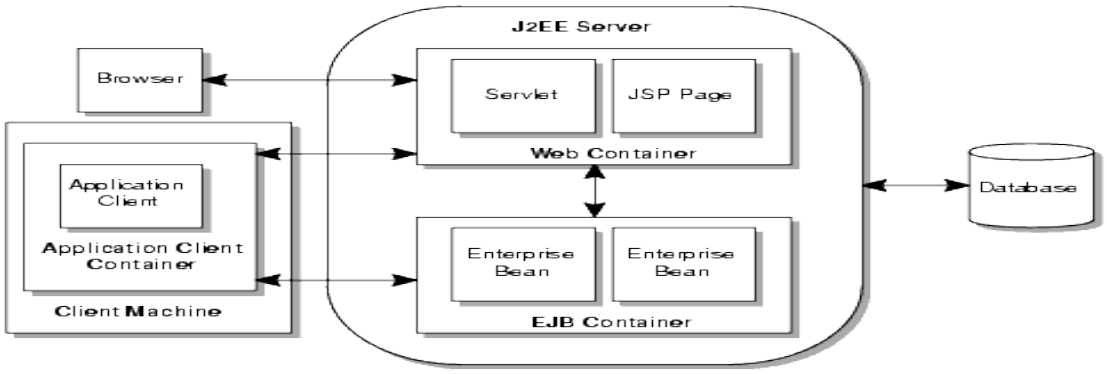
Q-1 Explain J2ee architecture in detail.

Ans-



- The JEE platform provides the environment to develop enterprise applications / services using multitier architecture.

- The highly intensified technology made the need for scalable, efficient, faster solutions for information management.

- The JEE technology is rightly apt for meeting these requirements.

- JEE provides a programming development environment which improves the productivity of development, standards for hosting / deploying enterprise applications.

- The following are the tiers in JEE application

Client Tier :

- The client tier includes the web components such as Servlets, JSP or standalone Java Desktop applications.

- This tier provides dynamic interfaces to the middle tier.

Middle Tier:

- This is also called as the server tier.

- In the middle tier enterprise beans and web services encapsulate distributable business logic for the applications which are reusable.

- The JEE application server contains the server-tier components which provides the platform for these web components for actions to be performed and data to be stored / persisted.

Enterprise data tier :

- The enterprise level data is stored / persisted preferably or typically in a relational database.

- In this tier, the JEE applications comprises of components, containers and services.

- All the web components (Servlets, JSP) provide dynamic requests and responses from a web page.

- The EJB components contain the server-side business logic for enterprise applications.

Explain J2EE is a container centric architecture.
- A container acts as an interface between a platform-specific functionality and a component.

- The component must be assembled before a web or enterprise bean or application client component execution, into a JEE application and deployed into its container.

- The settings of a container can be customized for underlying support provided by the JEE server.

- These include security, transaction management, and Java Naming and Directory Interface lookups.

- The management of non configurable services such as servlet life cycle, enterprise bean life cycle, database connection, data persistence, database connection pooling also can be done by the container.

1. Web Container :

- A web container is a part of web server.

- It provides the run time environment to execute a web application such as a servlet, JSP.

- A servlet container translates the URL requests into servlet requests.

- The JSP implicit objects such as request, response, out, page, pageContext etc., are exposed by JSP container.

2. EJB Container :

- The EJB container, like other containers provides run-time environment to execute EJB components such as enterprise beans.

- An EJB container manages transactions, state management details, multi threading, connection pooling.

- The applications are provided with security using EJB container.

- All database access required by the entity bean will be handled by the EJB container.
Explain the J2EE container architecture.
J2EE Container Architecture :

- The J2EE application components needs support at runtime.

- This support is provided by J2EE containers.

- They use protocols, methods of the containers to access other application components.

- The containers of J2EE are:
1. Web container
2. EJB container

1. The web container :

- It is used to host web applications. It provides the run time environment to execute Servlet and JSP component types.

2. EJB container :

- The business logic is dealt by the server components called EJB components.

- The access to local and remote enterprise beans is provided by the EJB container.

- The operations of the three beans namely Entity Bean, Session Bean and Message-driven bean are handled by the EJB Container.

Q-2 What is JDBC? Explain types of drivers in JDBC.

JDBC Driveris an acronym for Java Database Connectivity. It's an advancement for ODBC (Open Database Connectivity). JDBC is a standard API specification developed in order to move data from frontend to backend. This API consists of classes and interfaces written in Java.

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 JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver.

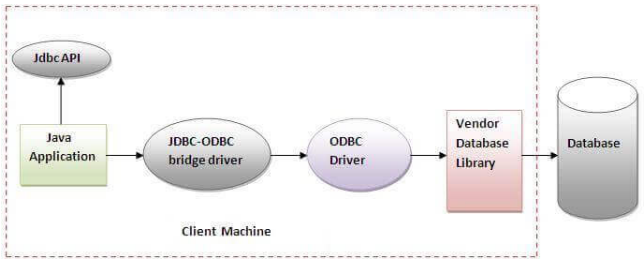


Figure- JDBC-ODBC Bridge Driver

Oracle does not support the JDBC-ODBC Bridge from Java 8. Oracle recommends that you use JDBC drivers provided by the vendor of your database instead of the JDBC-ODBC Bridge.

Advantages:

- easy to use.
- can be easily connected to any database.

Disadvantages:

- Performance degraded because JDBC method call is converted into the ODBC function calls.
- The ODBC driver needs to be installed on the client machine.

2) Native-API driver

The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.

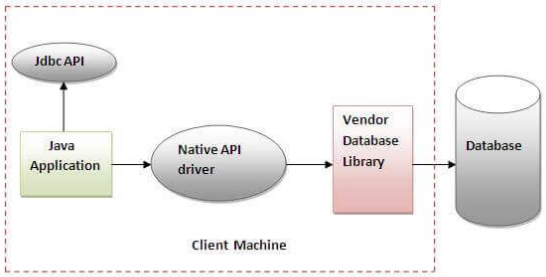


Figure- Native API Driver

Advantage:

- performance upgraded than JDBC-ODBC bridge driver.

Disadvantage:

- The Native driver needs to be installed on the each client machine.
- The Vendor client library needs to be installed on client machine.

3) Network Protocol driver

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.

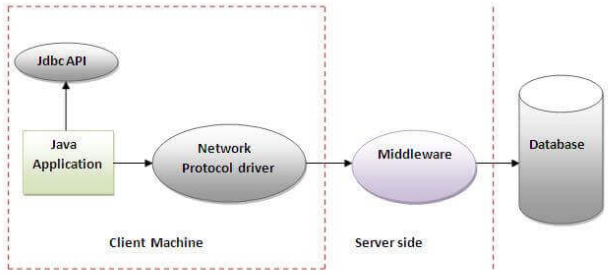


Figure- Network Protocol Driver

Advantage:

- No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

Disadvantages:

- Network support is required on client machine.
- Requires database-specific coding to be done in the middle tier.

- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

---

4) Thin driver

The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.

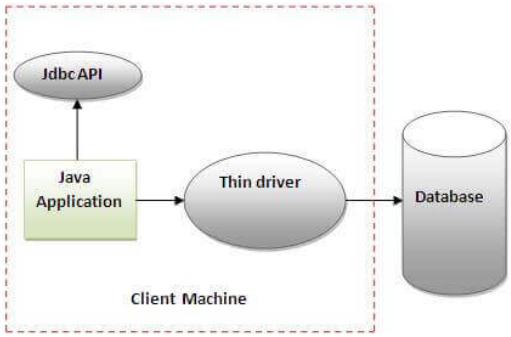


Figure- Thin Driver

Advantage:

- Better performance than all other drivers.
- No software is required at client side or server side.

Disadvantage:

- Drivers depend on the Database.

Q-3 What are the steps to establish connection with database to java web application.

There are 5 steps to connect any java application with the database using JDBC. These steps are as follows:

- Register the Driver class
- Create connection
- Create statement
- Execute queries
- Close connection

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 of forName() method

```
1. public static void forName(String className)throws ClassNotFoundException
```

Ex. 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 of getConnection() method

```
1. 1) public static Connection getConnection(String url)throws SQLException
2. 2) public static Connection getConnection(String url,String name,String password)
3. throws SQLException
```

Example to establish connection with the Oracle database

```
1. Connection con=DriverManager.getConnection(
2. "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 of createStatement() method

```
1. public Statement createStatement()throws SQLException
```

Example to create the statement object

```
1. 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 of executeQuery() method

```
1. public ResultSet executeQuery(String sql)throws SQLException
```

Example to execute query

```
1. ResultSet rs=stmt.executeQuery("select * from emp");
2.
```

```
3. while(rs.next()){
4. System.out.println(rs.getInt(1)+" "+rs.getString(2));
5. }
```

---

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 of close() method

```
1. public void close()throws SQLException
```

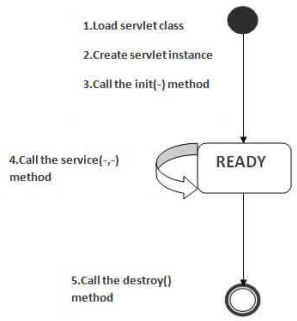
Example to close connection

```
1. con.close();
```

Q-4 explain servlet life cycle

The web container maintains the life cycle of a servlet instance. Let's see the life cycle of the servlet:

1. Servlet class is loaded.
2. Servlet instance is created.
3. init method is invoked.
4. service method is invoked.
5. destroy method is invoked.



1) Servlet class is loaded

The classloader is responsible to load the servlet class. The servlet class is loaded when the first request for the servlet is received by the web container.

2) Servlet instance is created

The web container creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.

3) init method is invoked

The web container calls the init method only once after creating the servlet instance. The init method is used to initialize the servlet. It is the life cycle method of the javax.servlet.Servlet interface. Syntax of the init method is given below:

```
public void init(ServletConfig config) throws ServletException
```

4) service method is invoked

The web container calls the service method each time when request for the servlet is received. If servlet is not initialized, it follows the first three steps as described above then calls the service method. If servlet is initialized, it calls the service method. Notice that servlet is initialized only once. The syntax of the service method of the Servlet interface is given below:

```
public void service(ServletRequest request, ServletResponse response)

 throws ServletException, IOException
```

5) destroy method is invoked

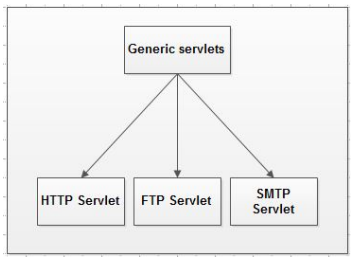
The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc. The syntax of the destroy method of the Servlet interface is given below:

```
public void destroy()
```

Q-5 Explain different types of servlets.

here is a possibility of developing 'n' types of servlets, like httpservlet, ftpservlet, smtpservlet etc. for all these protocol specific servlet classes GenericServlet is the common super class containing common properties and logics. So, GenericServlet is not a separate type of servlet.

As of now Servlet API is giving only one subclass to GenericServlet i.e HttpServlet class because all web servers are designed based on the protocol http.



- **Generic servlets extend javax.servlet.GenericServlet** – It is protocol independent servlet. Generic Servlet is a base class servlet from which all other Servlets are derived. Generic Servlet supports for HTTP, FTP and SMTP protocols. It implements the Servlet and ServletConfig interface. It has only init() and destroy() method of ServletConfig interface in its life cycle. It also implements the log method of ServletContext interface. Javax Servlet package is used and helps in extending java.lang.object to implement Generic Servlets along with Servlet Config which makes initialization parameters to be passed through XML language in the servlet.

- **HTTP servlets extend javax.servlet.HttpServlet** – HTTPServlet is HTTP dependent servlet. The HTTP protocol is a set of rules that allows Web browsers and servers to communicate. When Web browsers and servers support the HTTP protocol, Java-based web applications are dependent on HTTP Servlets.HttpServlet is Extended by Generic Servlet. It provides an abstract class for the developers for extend to create there own HTTP specific servlets. Javax servlet Http package is used where Generic Servlet is extended to implement java.io.serializable along with the public class of HTTP servlet class. HTTP servlet has the properties of both Generic and HTTP servlet so that developers can make use of all the functionalities in the servlet.

Q-6 What is Session?Explain Session management technique in detail.

A session is a period of time wherein a user interacts with an app. Usually triggered by the opening of an app, a session records the length and frequency of app use to show developers, marketers and product managers how much time users spend within an app.

For example, session data can be used to determine the average length of time users spend on an app, as well as the time of day users are most likely to engage with a particular app.
1) URL rewriting
2) Cookies
3) Hidden Form fields
4) HTTPS and SSL
1. URL rewriting
URL rewriting is a method of session tracking in which some extra data (session ID) is appended at the end of each URL. This extra data identifies the session. The server can associate this session identifier with the data it has stored about that session. This method is used with browsers that do not support cookies or where the user has disabled the cookies. If you need to track Session from JSP pages, then you can use the <c:out> tag for URL-rewriting. It automatically encodes session identifiers in URL.

2. Hidden Form Fields
This is one of the oldest ways to do session tracking in the Servlet application. Similar to URL rewriting. The server embeds new hidden fields in every dynamically generated form page for the client. When the client submits the form to the server the hidden fields identify the client. You can further see Head First Servlet and JSP for more details on how to use the hidden form field to manage sessions in Servlet JSP.

3. Cookies
A cookie is a small amount of information sent by a servlet to a Web browser. A cookie is saved by the browser and later sent back to the server in subsequent requests. A cookie has a name, a single value, expiration date, and optional attributes.

A cookie's value can uniquely identify a client. Since a client can disable cookies, this is not the most secure and fool-proof way to manage the session. If Cookies are disabled then you can fall back to URL rewriting to encode Session id e.g. JSESSIOINID into the URL itself.

Q-7 Explain J2EE technologies-book

Q-8 What is Enterprise?Explain different styles of enterprise architecture.

Ans- **Tier Architecture**

1. One-tier Architecture:

One-tier architecture has Presentation layer, Business layer and Data layers at the same tier i.e. at Client Tier. As the name suggested, all the layers and components are available on the same machine. MP3 player, MS Office etc. are some of the examples of one-tier architecture. To store the data (as a function of Data Layer) local system or a shared drive is used.

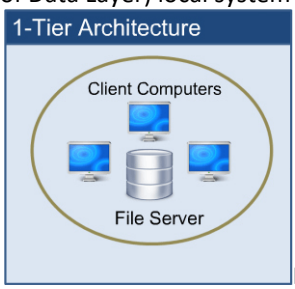


Figure 01: Software Architecture Type – 1 tier

2. Two-tier Architecture:

In such type of architecture, the client tier handles both Presentation and Application layers and the server handles the Database layer. The two-tier architecture is also known as a 'Client-Server Application'. In two-tier architecture, communication takes place between the Client and the Server. Client system sends the request to the server system and the server system processes the request and sends the response back to the client system.

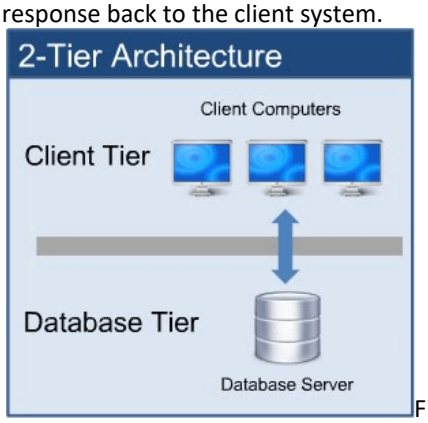


Figure 02: Software Architecture Type – 2 tier

3. Three-tier Architecture:

All three major layers are separated from each other. Presentation layer resides at Client Tier, Application layer acts as middle-ware and lies at Business Tier and Data layer is available at Data Tier. This is a very common architecture.

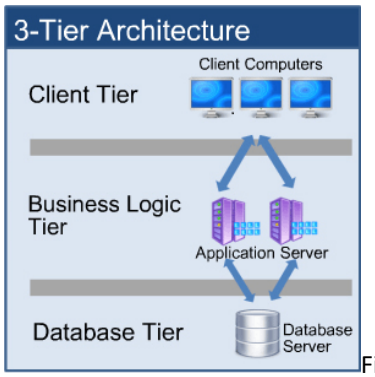


Figure 03: Software Architecture Type – 3 tier

4. N-tier Architecture:

N-tier architecture is also called a Distributed Architecture or Multi-tier Architecture. It is similar to three-tier architecture but the number of the application server is increased and represented in individual tiers in order to distribute the business logic so that the logic can be distributed.

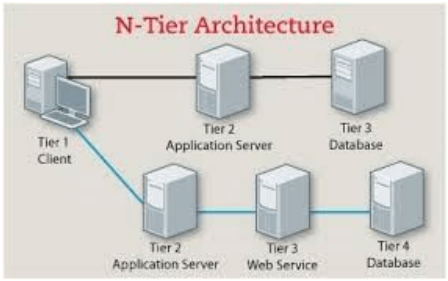


Figure 04: Software Architecture Type – N tier

Q-9 What is a use of JSP Directives?List out type of JSP Directives and Explain any one.

- JSP directives are the messages to JSP container. They provide global information about an entire JSP page.
- JSP directives are used to give special instruction to a container for translation of JSP to servlet code.
- In JSP life cycle phase, JSP has to be converted to a servlet which is the translation phase.
- They give instructions to the container on how to handle certain aspects of JSP processing
- A JSP directive affects the overall structure of the servlet class. It usually has the following form –
- <%@ directive attribute = "value" %>
- Directives can have a number of attributes which you can list down as key-value pairs and separated by commas.
- The blanks between the @ symbol and the directive name, and between the last attribute and the closing %>, are optional.
- There are three types of directive tag –

S.No.	Directive & Description
1	**<%@ page ... %>** Defines page-dependent attributes, such as scripting language, error page, and buffering requirements.
2	**<%@ include ... %>** Includes a file during the translation phase.
3	**<%@ taglib ... %>** Declares a tag library, containing custom actions, used in the page

- JSP - The page Directive
- The **page** directive is used to provide instructions to the container. These instructions pertain to the current JSP page. You may code page directives anywhere in your JSP page. By convention, page directives are coded at the top of the JSP page.

- Following is the basic syntax of the page directive −
- <%@ page attribute = "value" %>
- You can write the XML equivalent of the above syntax as follows −
- <jsp:directive.page attribute = "value" />
- Attributes
- Following table lists out the attributes associated with the page directive −

S.No.	Attribute & Purpose
1	**buffer** Specifies a buffering model for the output stream.
2	**autoFlush** Controls the behavior of the servlet output buffer.
3	**contentType** Defines the character encoding scheme.
4	**errorPage** Defines the URL of another JSP that reports on Java unchecked runtime exceptions.
5	**isErrorPage** Indicates if this JSP page is a URL specified by another JSP page's errorPage attribute.
6	**extends** Specifies a superclass that the generated servlet must extend.
7	**import** Specifies a list of packages or classes for use in the JSP as the Java import statement does for Java classes.
8	**info** Defines a string that can be accessed with the servlet's **getServletInfo()** method.

9	**isThreadSafe** Defines the threading model for the generated servlet.
10	**language** Defines the programming language used in the JSP page.
11	**session** Specifies whether or not the JSP page participates in HTTP sessions
12	**isELIgnored** Specifies whether or not the EL expression within the JSP page will be ignored.
13	**isScriptingEnabled** Determines if the scripting elements are allowed for use.

- Check for more details related to all the above attributes at Page Directive.
- The include Directive
- The **include** directive is used to include a file during the translation phase. This directive tells the container to merge the content of other external files with the current JSP during the translation phase. You may code the ***include*** directives anywhere in your JSP page.
- The general usage form of this directive is as follows −
- <%@ include file = "relative url" >
- The filename in the include directive is actually a relative URL. If you just specify a filename with no associated path, the JSP compiler assumes that the file is in the same directory as your JSP.
- You can write the XML equivalent of the above syntax as follows −
- <jsp:directive.include file = "relative url" />
- For more details related to include directive, check the Include Directive.
- The taglib Directive
- The JavaServer Pages API allow you to define custom JSP tags that look like HTML or XML tags and a tag library is a set of user-defined tags that implement custom behavior.
- The **taglib** directive declares that your JSP page uses a set of custom tags, identifies the location of the library, and provides means for identifying the custom tags in your JSP page.
- The taglib directive follows the syntax given below −
- <%@ taglib uri="uri" prefix = "prefixOfTag" >
- Here, the **uri** attribute value resolves to a location the container understands and the **prefix** attribute informs a container what bits of markup are custom actions.

- You can write the XML equivalent of the above syntax as follows −
- <jsp:directive.taglib uri = "uri" prefix = "prefixOfTag" />

Q-10 Write a short note on JSP Action

JSP actions are special XML tags that control the behavior of the servlet engine. JSP actions allow you to insert a file dynamically, reuse external JavaBean components, forward the request to the other page and generate HTML for Java Applet Plugin.

jsp:include action

JSP include action allows you to include a file at runtime. The syntax of JSP include action is as follows:

```
<jsp:include page="Relative URL" flush="true" />
```
Code language: HTML, XML (xml)

In the page attribute, you insert a relative URL of a file which could be an HTML file or another JSP page. Unlike the include directive, the jsp include action insert a file at the time page is being requested.

jsp:useBean action

JSP useBean action lets you load a JavaBean component into the page and use it later. JSP useBean action allows you to reuse other Java classes. The syntax of JSP useBean action is as follows:

```
<jsp:useBean id="objectName" class="package.class" />
```
Code language: JavaScript (javascript)

By using the jsp:useBean action, you create a new object with the object name*objectName* of the class *package.class.* Later on, you can access the properties of this object by using either *jsp:setProperty* or *jsp:getProperty*.

jsp:forward Action

jsp:forward action allows you to forward a request to the other page. The syntax of the jsp:forward action is listed as below. There is one attribute called page which value is a page you want to forward the request to. You can specify the page statically or dynamically by using the expression.

```
<jsp:forward page="error.jsp" />
<jsp:forward page="<%= java-expression %>" />
```
Code language: HTML, XML (xml)

jsp:plugin Action

jsp:plugin action allows you to embedded Java Applet into a page. Suppose you have an applet which demonstrates the JSP page life cycle called *com.jsp.jspapplet.*

Q-11 write a short note on JSP implicit object.

There are **9 jsp implicit objects**. These objects are *created by the web container* that are available to all the jsp pages.

The available implicit objects are out, request, config, session, application etc.

A list of the 9 implicit objects is given below:

Object	Type
out	JspWriter
request	HttpServletRequest
response	HttpServletResponse
config	ServletConfig
application	ServletContext
session	HttpSession
pageContext	PageContext
page	Object
exception	Throwable

JSP out implicit object

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:

PrintWriter out=response.getWriter();

But in JSP, you don't need to write this code.

Example of out implicit object

In this example we are simply displaying date and time.

index.jsp

```
1. <html>
2. <body>
3. <% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
4. </body>
5. </html>
```

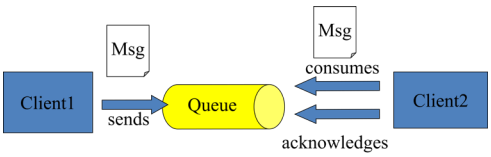
Q-12 Explain JMS with Example

- A **specification** that describes a common way for Java programs to create, send, receive and read distributed enterprise messages

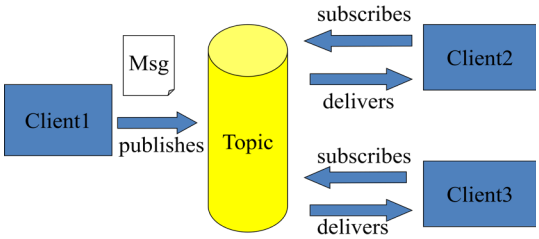
- *Asynchronous* messaging
- *Reliable* delivery
  - A message is guaranteed to be delivered once and only once.
- Outside the specification
  - Security services
  - Management services

JMS Messaging Domains(Modes)

- Point-to-Point (PTP) (One-to-one)
  - built around the concept of message queues
  - each message has only one consumer
  - Send a message to a JMS Queue



- Publish-Subscribe (One-to-Many)
  - Send (publish) message to a JMS Topic
  - Enables many readers (subscribers)
  - uses a “topic” to send and receive messages
  - each message has multiple consumers



JMS Client Example

- Setting up a connection and creating a session

```
InitialContext jndiContext=new InitialContext();

//look up for the connection factory

ConnectionFactory cf=jndiContext.lookup(connectionfactoryname);

//create a connection

Connection connection=cf.createConnection();

//create a session
```

Session session=connection.createSession(false,Session.AUTO_ACKNOWLEDGE);

//create a destination object

Destination dest1=(Queue) jndiContext.lookup("/jms/myQueue"); //for PointToPoint

Destination dest2=(Topic)jndiContext.lookup("/jms/myTopic"); //for publish-subscribe

Producer Sample

- Setup connection and create a session
- Creating producer

  MessageProducer producer=session.createProducer(dest1);

- Send a message

  Message m=session.createTextMessage();

  m.setText("just another message");

  producer.send(m);

- Closing the connection

  connection.close();

Consumer Sample (Synchronous)

- Setup connection and create a session
- Creating consumer

  MessageConsumer consumer=session.createConsumer(dest1);

- Start receiving messages

  connection.start();

  Message m=consumer.receive();

Consumer Sample (Asynchronous)

- Setup the connection, create a session
- Create consumer
- Registering the listener
  - MessageListener listener=new myListener();
  - consumer.setMessageListener(listener);
- myListener should have onMessage()

```
public void onMessage(Message msg){

        //read the massage and do computation

}
```

Q-13 What is Java Mail?Explain Java Mail API in detail.

The **JavaMail** is an API that is used to compose, write and read electronic messages (emails).

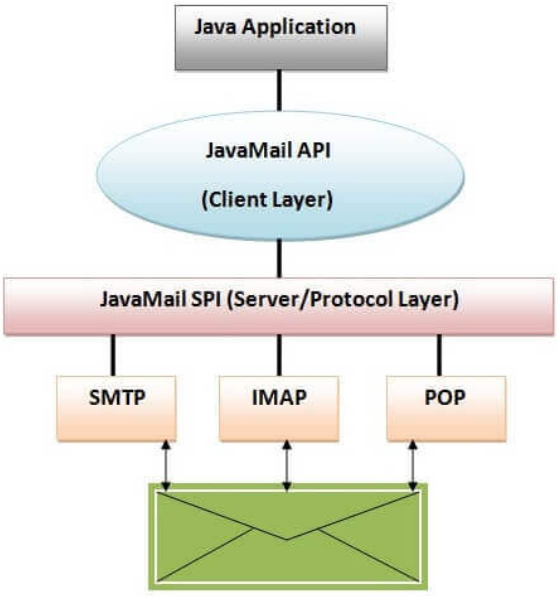
The JavaMail API provides protocol-independent and plateform-independent framework for sending and receiving mails.

The **javax.mail** and **javax.mail.activation** packages contains the core classes of JavaMail API.

The JavaMail facility can be applied to many events. It can be used at the time of registering the user (sending notification such as thanks for your interest to my site), forgot password (sending password to the users email id), sending notifications for important updates etc. So there can be various usage of java mail ap

JavaMail Architecture

The java application uses JavaMail API to compose, send and receive emails. The JavaMail API uses SPI (Service Provider Interfaces) that provides the intermediatory services to the java application to deal with the different protocols. Let's understand it with the figure given below:



Protocols used in JavaMail API

There are some protocols that are used in JavaMail API.

- SMTP
- POP
- IMAP
- MIME
- NNTP and others

SMTP

SMTP is an acronym for Simple Mail Transfer Protocol. It provides a mechanism to deliver the email. We can use Apache James server, Postcast server, cmail server etc. as an SMTP server. But if we purchase the host space, an SMTP server is bydefault provided by the host provider. For example, my smtp server is mail.javatpoint.com. If we use the SMTP server provided by the host provider, authentication is required for sending and receiving emails.

POP

POP is an acronym for Post Office Protocol, also known as POP3. It provides a mechanism to receive the email. It provides support for single mail box for each user. We can use Apache James server, cmail server etc. as an POP server. But if we purchase the host space, an POP server is bydefault provided by the host provider. For example, the pop server provided by the host provider for my site is mail.javatpoint.com. This protocol is defined in RFC 1939.

IMAP

IMAP is an acronym for Internet Message Access Protocol. IMAP is an advanced protocol for receiving messages. It provides support for multiple mail box for each user ,in addition to, mailbox can be shared by multiple users. It is defined in RFC 2060.

MIME

Multiple Internet Mail Extension (MIME) tells the browser what is being sent e.g. attachment, format of the messages etc. It is not known as mail transfer protocol but it is used by your mail program.

NNTP and Others

There are many protocols that are provided by third-party providers. Some of them are Network News Transfer Protocol (NNTP), Secure Multipurpose Internet Mail Extensions (S/MIME) etc.

Q-14 What is RMI?Write a Steps to create RMI application of addition of two numbers.

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*.

Understanding stub and skeleton

RMI uses stub and skeleton object for communication with the remote object.

A **remote object** is an object whose method can be invoked from another JVM. Let's understand the stub and skeleton objects:

stub

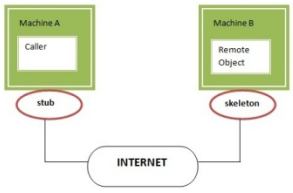
The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

1. It initiates a connection with remote Virtual Machine (JVM),
2. It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
3. It waits for the result
4. It reads (unmarshals) the return value or exception, and
5. It finally, returns the value to the caller.

skeleton

The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

1. It reads the parameter for the remote method
2. It invokes the method on the actual remote object, and
3. It writes and transmits (marshals) the result to the caller.



Addition of two numbers

1) create the remote interface
For creating the remote interface, extend the Remote interface and declare the RemoteException with all the methods of the remote interface. Here, we are creating a remote interface that extends the Remote interface. There is only one method named add() and it declares RemoteException.

```
1.  import java.rmi.*;
2.  public interface Adder extends Remote{
3.  public int add(int x,int y)throws RemoteException;
4.  }
```

---

2) Provide the implementation of the remote interface
Now provide the implementation of the remote interface. For providing the implementation of the Remote interface, we need to

- Either extend the UnicastRemoteObject class,
- or use the exportObject() method of the UnicastRemoteObject class

In case, you extend the UnicastRemoteObject class, you must define a constructor that declares RemoteException.

```
1.  import java.rmi.*;
2.  import java.rmi.server.*;
3.  public class AdderRemote extends UnicastRemoteObject implements Adder{
4.  AdderRemote()throws RemoteException{
5.  super();
6.  }
7.  public int add(int x,int y){return x+y;}
8.  }
```

---

3) create the stub and skeleton objects using the rmic tool.

Next step is to create stub and skeleton objects using the rmi compiler. The rmic tool invokes the RMI compiler and creates stub and skeleton objects.

```
1.  rmic AdderRemote
```

---

4) Start the registry service by the rmiregistry tool

Now start the registry service by using the rmiregistry tool. If you don't specify the port number, it uses a default port number. In this example, we are using the port number 5000.

```
1.  rmiregistry 5000
```

---

5) Create and run the server application

Now rmi services need to be hosted in a server process. The Naming class provides methods to get and store the remote object. The Naming class provides 5 methods.

1. **public static java.rmi.Remote lookup(java.lang.String) throws java.rmi.NotBoundException, java.net.MalformedURLException, java.rmi.RemoteException;** it returns the reference of the remote object.
2. **public static void bind(java.lang.String, java.rmi.Remote) throws java.rmi.AlreadyBoundException, java.net.MalformedURLException, java.rmi.RemoteException;** it binds the remote object with the given name.
3. **public static void unbind(java.lang.String) throws java.rmi.RemoteException, java.rmi.NotBoundException, java.net.MalformedURLException;** it destroys the remote object which is bound with the given name.
4. **public static void rebind(java.lang.String, java.rmi.Remote) throws java.rmi.RemoteException, java.net.MalformedURLException;** it binds the remote object to the new name.
5. **public static java.lang.String[] list(java.lang.String) throws java.rmi.RemoteException, java.net.MalformedURLException;** it returns an array of the names of the remote objects bound in the registry.

In this example, we are binding the remote object by the name sonoo.

```
1.  import java.rmi.*;
2.  import java.rmi.registry.*;
3.  public class MyServer{
4.  public static void main(String args[]){
5.  try{
6.  Adder stub=new AdderRemote();
7.  Naming.rebind("rmi://localhost:5000/sonoo",stub);
8.  }catch(Exception e){System.out.println(e);}
9.  }
10. }
```

---

6) Create and run the client application

At the client we are getting the stub object by the lookup() method of the Naming class and invoking the method on this object. In this example, we are running the server and client applications, in the same machine so we are using localhost. If you want to access the remote object from another machine, change the localhost to the host name (or IP address) where the remote object is located.

```
1.  import java.rmi.*;
2.  public class MyClient{
3.  public static void main(String args[]){
4.  try{
5.  Adder stub=(Adder)Naming.lookup("rmi://localhost:5000/sonoo");
6.  System.out.println(stub.add(34,4));
7.  }catch(Exception e){}
8.  }
9.  }
```

Q-15 What is struts?Explain MVC Architecture in detail.

Struts in Java are used to develop web applications which are usually based on servlet and JSP. It simplifies the development and maintenance of web applications by providing predefined functionality. It is based on a front controller, which means it has a controller in front of it which decides on which model request has to come or go. Struts in Java are very helpful for us as MVC is a guideline followed by all technologies in today's world. There is no better option to simplify web applications other than MVC. Struts are not distributed.

- MVC is an abbreviation for Model-View-Controller.
- For ease of development and deployment of the project, the web-application is divided into basic 3 modules: model, view and controller.
- As the name suggests:
    - Model: responsible for handling database interaction with the controller (database layer)
    - View: responsible for displaying contents of user (presentation layer)
    - Controller: responsible for acting as per user action (business logic layer)
- Considering the example of AWT or Java Swing (Drop-down menu/Combobox menu), if the user wants to retrieve the states belonging to the selected country from drop-down menu,
    - Model: retrieves the selected names of states from the database
    - View: displays the Combobox menu to the user (coordinates with model)
    - Controller: sends message to model about the user's selection of country name.
- Considering Struts or Spring or Hibernate frameworks:
    - Model: Java Beans (database)
    - View: HTML or JSP page
    - Controller: back-end servlet (converted JSP's)

- The complete architecture is mainly controlled by the controller. The end-user (client) actions on view, based on which the controller delegates the requests to model. The model retrieves required data from beans or database (through database queries) and responses back to the controller. Based on the response received, the controller changes/modifies the data in view.

- The figure below represents MVC in client-server: