Q.1 a. What is DHTML? What are the features of DHTML? (2+2 Marks)

Answer:

DHTML

- DHTML stands for Dynamic HTML.
- DHTML is that it is neither a language like HTML, JavaScript etc. nor a web standard.
- It is just a combination of HTML, JavaScript and CSS.
- It just uses these languages features to build dynamic web pages.
- DHTML is a feature of Netscape Communicator 4.0, and Microsoft Internet Explorer 4.0 and 5.0 and is entirely a "client-side" technology.

Features of DHTML:

- Simplest feature is making the page dynamic.
- Can be used to create animations, games, applications, provide new ways of navigating through web sites.
- DHTML use low-bandwidth effect which enhance web page functionality.
- Dynamic building of web pages is simple as no plug-in is required.
- Facilitates the usage of events, methods and properties and code reuse.

b. What is a Socket? How to create a ServerSocket object? (2+2 Marks)

Answer:

Definition : Socket

A socket is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to.

Creation of ServerSocket object:

To create a server listening for requests, all need to do is create a ServerSocket object attached to a port number and call method accept().

For example, port 8080:

```
ServerSocket sSocket = new ServerSocket(8080);
Socket channel = sSocket.accept();
```

Method accept() returns when a client has connected to server. The channel socket has a **different** port number than 8080.

The server socket is 8080 so to get more than one person talking to the server at once, the server needs to hand off socket connections to a different port.

c. Write the rules of XML declaration and List out the three parts of XSL.

(2+2 Marks)

Answer:

- The XML declaration is case sensitive: it may not begin with "<?XML" or any other variant;
- If the XML declaration appears at all, it must be the very first thing in the XML document: not even white space or comments may appear before it; and
- It is legal for a transfer protocol like HTTP to override the encoding value that put in the XML declaration, so cannot guarantee that the document will actually use the encoding provided in the XML declaration.

Three part of XSL

- XSLT - a language for transforming XML documents
- XPath - a language for navigating in XML documents
- XSL-FO - a language for formatting XML documents

d. What is AJAX? Write the disadvantages of AJAX. (2+2 Marks)

Answer:

- AJAX = Asynchronous JavaScript and XML
- JAX is a technique for creating fast and dynamic web pages.
- AJAX allows web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes.
- Classic web pages, (which do not use AJAX) must reload the entire page if the content should change.
- Examples of applications using AJAX: Google Maps, Gmail, YouTube, and Facebook.

Disadvantages of AJAX :

- Search engines would not be able to index an AJAX application.
- The server information can not be accessed within AJAX.
- AJAX is not well integrated with any browser.
- ActiveX requests are enabled only in IE 5 and IE6
- Data of all requests is URL-encoded, which increases the size of the request.

e. Distinguish between Session and Entity Beans. (4 Marks)

Answer:

- A session bean is an enterprise bean that handles the logic or business rules, for example a stock transaction.
- It's called a session bean because it lasts as long as the session. Maybe session would be to log on, sell a stock and put that money in an account.
- This might be a stateful session bean, because it depends on the state.

- The state is retained. A stateless session bean might be just one transaction - something where don't need to know the state - like maybe just verifying a credit card number.
- An entity bean represents data - they don't do logic. So a session bean might use an entity bean to access data.

f. What is the JMS API? When can you use the JMS API? (2+2 Marks)

Answer:

JMS API

The JMS API defines a common set of interfaces and associated semantics that allow programs written in the Java programming language to communicate with other messaging implementations.

Use the JMS API

An enterprise application provider is likely to choose a messaging API over a tightly coupled API, such as a remote procedure call (RPC), under the following circumstances.
■ The provider wants the components not to depend on information about other components' interfaces, so components can be easily replaced.
■ The provider wants the application to run whether or not all components are up and running simultaneously.
■ The application business model allows a component to send information to another and to continue to operate without receiving an immediate response.

g. Give the important terminologies in HTTP. (4 Marks)

Answer:

Connection : A transport layer virtual circuit established between two application programs for the purpose of communication.
Message : The basic unit of HTTP communication, c consisting of a structured sequence of octets matching the syntax defined in and transmitted via the connection.
Request : An HTTP request message.
Response : An HTTP response message.

Q.2 a. What is the difference between javascript and HTML? (3 Marks)

Answer:

- HTML stands for Hypertext Mark-up Language and is an Internet World Wide Web description language that tells your web browser how a page is going to look.
- HTML is the language in which web pages are written to be interpreted by the server as a graphic interface.
- Javascipt is similar, but is actually a web-adapted version of actual programming code, intended to write applets (small, web-based applications) for use on websites.
- They are both a type of code, but html is NOT a programming code.
- The HTML defines what the webpage looks like, the javascript handles things like counters, games, etc.

- Other web-based scripting languages, such as php and asp can be used to create entire webpages, but, like Javascript, are generally confined to handling what goes beyond the cosmetics of a website. The cosmetics are what HTML is for.
- Javascript allows you to do things on a webpage that HTML.
- HTML is the base, with Javascript sitting on top of it.

b. How to create links to sections on the same page in HTML? (6 Marks)

To create links to a link within the page, two HTML tags need to be used.

<A HREF="#top">Top</A>

This first tag is almost the same as any other HTML tag would use to create a link to another page. However, to create a bookmark, must start the link with a # (pound symbol,) which represents a name statement, used in the next tag. When the user clicks on Top, the computer would then go to the name tag, if found on the same page of the link.

<A NAME="top"></A>

This next tag is where the first tag will go when the link is clicked. These are commonly referred to as bookmark and anchor.

Today, all browsers recognize top as being the top of the page. If you want the visitor to go to the very top of the page, the above tag can be left out and the A HREF link will still work.

<A HREF="http://www.computerhope.com/issues/chsafe.htm#02">Windows 2000 Safe Mode

Finally, a link can contain a web page with a bookmark that is on that page. In the above example, this code would create a link to our Safe Mode page and automatically move down to the Windows 2000 Safe Mode section.

c. Explain the different types of CSS in HTML. (4 Marks)

Answer:

There are three ways to include the CSS with HTML:

- **Inline CSS:** it is used when only small context is to be styled.
  o To use inline styles add the style attribute in the relevant tag.
- **External Style Sheet:** is used when the style is applied to many pages.
  o Each page must link to the style sheet using the <link> tag. The <link> tag goes inside the head section:
  <head>
  <link rel="stylesheet" type="text/css" href="mystyle.css" />
  </head>
- **Internal Style Sheet:** is used when a single document has a unique style.
  o Internal styles sheet needs to put in the head section of an HTML page, by using the <style> tag, like this:

```
<head>
<style type="text/css">
hr {color:sienna}
p {margin-left:20px}
body {background-image:url("images/back40.gif")}
</style>
</head>
```

d. Explain the event handlers in JavaScript. (5 Marks)

Answer:

JavaScript code that are not added inside the <script> tags, but rather, inside the html tags, that execute JavaScript when something happens, such as pressing a button, moving your mouse over a link, submitting a form etc. The basic syntax of these event handlers is:

name_of_handler="JavaScript code here"

For Example :

Google

This is certainly unlike a regular JavaScript code in that here we're inserting it directly inside a HTML tag, via the onClick event handler. When the above link is clicked, the user will first see an alert message before being taken to Google.

Different event handlers with with different HTML tags. For example, while "onclick" can be inserted into most HTML tags to respond to that tag's onclick action, something like "onload" (see below) only works inside the <body> and <img> tags. Below are some of the most commonly used event handlers supported by JavaScript:

Event Handlers:

onclick: Use this to invoke JavaScript upon clicking (a link, or form boxes)

onload: Use this to invoke JavaScript after the page or an image has finished

onmouseover: Use this to invoke JavaScript if the mouse passes by some link

onmouseout: Use this to invoke JavaScript if the mouse goes pass some link

onunload: Use this to invoke JavaScript right after someone leaves this page**.**

Q.3 a. Write java socket server program. (9 Marks)

Answer:

- Every server is a program that runs on a specific system and listens on specific port. Sockets are bound to the port numbers and when we run any server it just listens on the socket and wait for client requests.
- For example, tomcat server running on port 8080 waits for client requests and once it get any client request, it respond to them.

SocketServerExample.java

```
package com.journaldev.socket;


import java.io.IOException;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.lang.ClassNotFoundException;

import java.net.ServerSocket;

import java.net.Socket;

 /**

 * This class implements java Socket server

  */

public class SocketServerExample {


   //static ServerSocket variable

   private static ServerSocket server;

   //socket server port on which it will listen

   private static int port = 9876;


   public static void main(String args[]) throws IOException, ClassNotFoundException{

      //create the socket server object

      server = new ServerSocket(port);

      //keep listens indefinitely until receives 'exit' call or program terminates
```

```
        while(true){
            System.out.println("Waiting for client request");
            //creating socket and waiting for client connection
            Socket socket = server.accept();
            //read from socket to ObjectInputStream object
            ObjectInputStream ois = new ObjectInputStream(socket.getInputStream());
            //convert ObjectInputStream object to String
            String message = (String) ois.readObject();
            System.out.println("Message Received: " + message);
            //create ObjectOutputStream object
            ObjectOutputStream oos = new
ObjectOutputStream(socket.getOutputStream());
            //write object to Socket
            oos.writeObject("Hi Client "+message);
            //close resources
            ois.close();
            oos.close();
            socket.close();
            //terminate the server if client sends exit request
            if(message.equalsIgnoreCase("exit")) break;
        }
        System.out.println("Shutting down Socket server!!");
        //close the ServerSocket object
        server.close();
    }

}
```

b. Write a program to Create SMTP Client and SMTP Server. Verify with SSL Certificates. (9 Marks)

Answer:

// Client_Sock.java code

```
package client_sock;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.net.Socket;

package client_sock;

public class Client_Sock {

public static void main(String[] args) {
    try
{
   Socket sock =new Socket ("locakhost",9999);
PrintStream pr = new PrintStream(sock.getOutputStream());
system.out.print("Ergasia 01 pes 620 20013:");
InputStreamReader rd = new InputStreamReader (System.in);
BufferedReader ed = new Bufferedreader(rd);

String temp = ed.readLine();

pr.println(temp);

BufferedReader gt = new BufferedReader(new InputStreamReader(sock.getInputStream()));
String tm = gt.readLine();
System.out.print(tm);


}
  catch (Exception ex)
{

        }
    }
}
```

//Server_Sock.java code

```
package server_sock;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.net.ServerSocket;
import java.net.Socket;

public class Server_Sock {
```

```
   public static void main(String[] args) {
try
{
   ServerSocket ser = new ServerSocket (9999);
   Socket sock = ser.accept();

   BufferedReader ed =new BufferedReader (new
InputStreamReader(sock.getInputsStream()));
   String tmp = ed.readLine();
   System.out.print("Irecieved:"+tmp);

   PrintStream pr = new PrintStream(sock.getOutputStream());
   String str = "The message delivered succesfully !!"
   pr.println(str);
   }
   catch(Exception ex){}

   }
}
```

Q.4 a. What is XPath? Discuss different types of nodes in the XPath data model.
(9 Marks)

Answer:

- XPath is a language for addressing an XML document's elements and attributes.
- As an example, receive an XML document that contains the details of a shipment and you want to retrieve the element/attribute values from the XML document. Don't just want to list the values of all the nodes, but also want to output the values of specificc elements or attributes.
- In such a case, would use XPath to retrieve the values of those elements and attributes. XPath constructs a hierarchical structure of an XML document, a tree of nodes, which is the XPath data model

The XPath data model consists of seven node types. The different types of nodes in the XPath data model as follows:

Node Type
Description

Root Node
The root node is the root of the DOM tree. The document element (the root element) is a child of the root node. The root node also has the processing instructions and comments as child nodes.

Element Node
This represents an element in an XML document. The character data, elements, processing instructions, and comments within an element are the child nodes of the element node.

Attribute Node
This represents an attribute other than the xmlns-prefixed attribute, which declares a namespace.

Text Node
The character data within an element is a text node. A text node has at least one character of data. A whitespace is also considered as a character of data. By default, the ignorable

whitespace after the end of an element and before the start of the following element is also a text node.

Comment Node

This represents a comment in an XML document, except the comments within the DOCTYPE declaration.

Processing

Instruction Node

This represents a processing instruction in an XML document except the processing instruction within the DOCTYPE declaration. The XML declaration is not considered as a processing instruction node.

Namespace Node

This represents a namespace mapping, which consists of a xmlns:-prefixed attribute such as xmlns:xsd="http://www. w3.org/2001/XMLSchema". A namespace node consists of a namespace prefix (xsd in the example) and a namespace URI (http://www.w3.org/2001/XMLSchema in the example).

- Specifi c nodes including element, attribute, and text nodes may be accessed with XPath.
- XPath supports nodes in a namespace.
- Nodes in XPath are selected with an XPath expression.
- An expression is evaluated to yield an object of one of the following four types: node set, Boolean, number, or string.

An example of a location step is:

- child::journal[position()=2]
- In the example, the child axis contains the child nodes of the context node. Node test is the journal node set, and predicate is the second node in the journal node set. An absolute location path is defined with respect to the root node, and starts with "/".
- The difference between a relative location path and an absolute location path is that a relative location path starts with a location step, and an absolute location path starts with "/".

b. How to transform an XML Document into another XML Document?

(9 Marks)

Answer:

- The following example transforms an XML document into another XML document.
- This is the input XML document named XMLInput.xml, which contains data about vehicles.
- Each second-level repeating element describes a particular car, with the nested elements that contain information about the model and year.
- The make information is an attribute on the second-level repeating element.

```
<?xml version="1.0" ?>
<vehicles>
  <car make="Ford">
    <model>Mustang</model>
    <year>1965</year>
  </car>
  <car make="Chevrolet">
    <model>Nova</model>
```

```
   <year>1967</year>
  </car>
</vehicles>
```

This is the XSL style sheet named XSLTransform.xsl that describes how to transform the XML. The conversion creates <root> as the root-enclosing element and <model> as the second-level repeating element. Each <model> element in the output XML document will include the values from the <car> element and the make= attribute from the input XML document.

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="xml" indent="yes"/>

<xsl:template match="/vehicles">
      <root> <xsl:apply-templates select="car"/> </root>
</xsl:template>

<xsl:template match="car">
          <model make="{@make}">
            <xsl:value-of select="model" />
          </model>
</xsl:template>

</xsl:stylesheet>
```

The following SAS program transforms the XML document. The procedure specifies the input XML document, the XSL style sheet, and the output XML document.

```
proc xsl
  in='C:\XMLInput.xml'
  xsl='C:\XSLTransform.xsl'
  out='C:\XMLOutput.xml';
run;
```

Here is the resulting output XML document named XMLOutput.xml.

Output XML Document

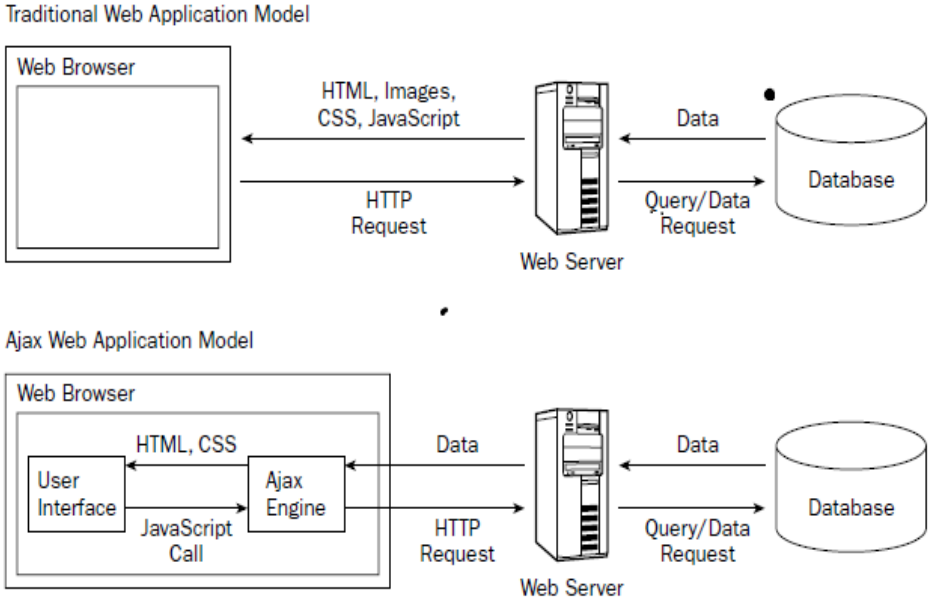
```
<?xml version="1.0" encoding="UTF-8"?>
<root>
<model make="Ford">Mustang</model>
<model make="Chevrolet">Nova</model>
</root>
```

Q.5 a. Explain how traditional web application model in different from AJAX web application model. List any three Ajax priciples. (9 Marks)

Answer:

- Ajax is nothing more than an approach to web interaction.
- This approach involves transmitting only a small amount of information to and from the server in order to give the user the most responsive experience possible.
- Instead of the traditional web application model where the browser itself is responsible for initiating requests to, and processing requests from, the web server, the Ajax model provides an intermediate layer— what Garrett calls an Ajax engine— to handle this communication.
- An Ajax engine is really just a JavaScript object or function that is called whenever information needs to be requested from the server.

- Instead of the traditional model of providing a link to another resource (such as another web page), each link makes a call to the Ajax engine, which schedules and executes the request.
- The request is done asynchronously, meaning that code execution doesn't wait for a response before continuing.
- The server—which traditionally would serve up HTML, images, CSS, or JavaScript—is configured to return data that the Ajax engine can use.
- This data can be plain text, XML, or any other data format that may need.
- The only requirement is that the Ajax engine can understand and interpret the data When the Ajax engine receives the server response, it goes into action, often parsing the data and making several changes to the user interface based on the information it was provided.
- Because this process involves transferring less information than the traditional web application model, user interface updates are faster, and the user is able to do his or her work more quickly. Figure displaying the difference between the traditional and Ajax web application models.



Ajax Principles

Software developer and usability expert, identified several key principles of good Ajax applications that are worth repeating:

i. **Minimal traffic:** Ajax applications should send and receive as little information as possible to and from the server. In short, Ajax can minimize the amount of traffic between the client and the server.

ii. **No surprises:** Ajax applications typically introduce different user interaction models than traditional web applications. As opposed to the web standard of click- and-wait, some Ajax applications use other user interface paradigms such as drag-and-drop or double-clicking.

iii. **Established conventions:** Don't waste time inventing new user interaction models that users will be unfamiliar with. Borrow heavily from traditional web applications and desktop applications, so there is a minimal learning curve.

iv. **No distractions:** Avoid unnecessary and distracting page elements such as looping animations and blinking page sections. Such gimmicks distract the user from what he or she is trying to accomplish.

v. **Accessibility:** Consider who primary and secondary users will be and how they most likely will access Ajax application.

vi. **Avoid entire page downloads:** All server communication after the initial page download should be managed by the Ajax engine. Don't ruin the user experience by downloading small amounts of data in one place but reloading the entire page in others.

vii. **User first:** Design the Ajax application with the users in mind before anything else.

b. How does a servlet communicate with a JSP page? (9 Marks)

Answer:

The following code snippet shows how a servlet instantiates a bean and initializes it with FORM data posted by a browser. The bean is then placed into the request, and the call is then forwarded to the JSP page, Bean1.jsp, by means of a request dispatcher for downstream processing.

```
public void doPost (HttpServletRequest request, HttpServletResponse response)
{
try {
govi.FormBean f = new govi.FormBean();
String id = request.getParameter("id");
f.setName(request.getParameter("name"));
f.setAddr(request.getParameter("addr"));
f.setAge(request.getParameter("age"));
//use the id to compute
//additional bean properties like info
//maybe perform a db query, etc.
// . . .
f.setPersonalizationInfo(info);
request.setAttribute("fBean",f);
getServletConfig().getServletContext().getRequestDispatcher
```

("/jsp/Bean1.jsp").forward(request, response);

} catch (Exception ex) {

. . .

}

}

The JSP page Bean1.jsp can then process fBean, after first extracting it from the default request scope via the useBean action.

jsp:useBean id="fBean" class="govi.FormBean" scope="request"/ jsp:getProperty

name="fBean" property="name" / jsp:getProperty name="fBean" property="addr"

/ jsp:getProperty name="fBean" property="age" / jsp:getProperty name="fBean"

property="personalizationInfo" /

Q.6 a. How can you hold and pass references to enterprise beans? (9 Marks)

Answer:

The EJB 1.1 specification does not prescribe a way for entity beans to store references to other entity beans. Application developers may choose from among several ways that entity beans can reference one another. These strategies include:

Store the remote reference in a transient, protected field of an entity bean. Entity beans can retain references to other entity beans in instance fields. The field should be protected, as all internal state should be; and it should be transient, because remote references aren't Serializable, and nontransient, nonserializable fields cause default serialization to fail. This particular method of dealing with enterprise bean references applies only to entity beans.

- **Use the referenced bean's primary key.** If the referenced bean's home interface is known and fixed, the referencing bean can use get the home interface of the referenced bean, and then use findByPrimaryKey() to find the bean's remote interface.
- **Use the bean's handle.** Handle . This handle is a direct reference to the referenced bean. The referencing bean can instantiate the Handle, and call the resulting object's method getEJBObject() to retrieve the referenced bean's remote interface.

- Every remote reference (that is, every instance of a subclass of EJBObject) has a method getHandle() that returns a serializable Handle .
- A handle is a serializable object, created by the application server, that specifies how to find a particular enterprise bean. The Handle object has a method called getEJBObject() which will return the referenced object if that object exists and can be found.
- Handles are typically used as robust references to server-side objects. Since they are implemented by the server vendor, their implementations are opaque; that is, there's no specified format for what should be in a Handle object.

- But since handles are Serializable, they can be turned into a byte string and stored or transmitted through a network, by way of Java RMI or any other protocol. Cooperating clients or server-side objects can pass handles to one another as byte strings.
- The receiving program can instantiate a received Handle, and then request its EJBObject.There is no guarantee that the bean will in fact exist when its handle is referenced. Session beans aren't guaranteed to survive container crashes, and the data representing an entity bean can be removed while serialized handles are outstanding.

The Handle does guarantee either to return a reference to the remote interface for a bean, or to throw an exception if the bean couldn't be located.

b. Explain the different types of EJB. (9 Marks)

Answer: Types of EJB:

Mainly three types of EJB :

i. Entity Bean,
ii. Session Bean and
iii. Message Driven Bean(MDB).

i. **Entity Bean**: it represents an entity which is mapped with database or we can say it makes OR object Relational mapping with Database. Entity bean typically represent table in RDBMS and each instance represent row in the table.

Two types of entity bean:

· **CMP Entity bean**: Container managed entity bean its responsibility of container to manage the bean persistence behavior.

· **BMP Entity bean**: Programmer manage the bean persistence behavior.

ii. **Session bean:** Session bean is responsible for developing business logic it makes the client server relationship so session beans exist till there is a session exist between client and server, it doesn't contain persistent business concept.

Types of session bean

Stateless session bean: when there is not need to maintain state of a particular client stateless session bean is used .They alive for short period of time.

For example if we are validating the credit card we can use stateless session bean.

Stateful session bean: stateful session bean maintain the conversational state of client over the series of method call before the bean instance goes to passive state conversational state is saved to persistence area like Hard disk and again when same client send a request and bean instance come into the active state it will come out from hard disk to main memory.

For Example when we do online banking transaction ,online reservation we use stateful session bean

iii. **Message Driven Beans**: these beans are work as a listener for messaging services like JMS

Q.7 a. Explain the JMS API Architecture with neat diagram. (9 Marks)

Answer: **JMS application is composed of the following parts.**

■ A *JMS provider* is a messaging system that implements the JMS interfaces and provides administrative and control features. An implementation of the Java EE platform includes a JMS provider.

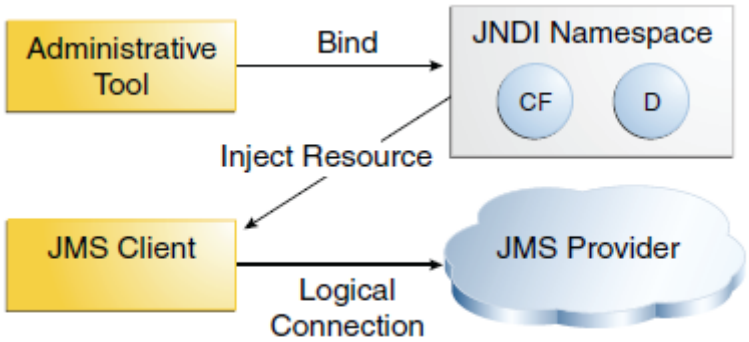
■ *JMS clients* are the programs or components, written in the Java programming language, that produce and consume messages. Any Java EE application component can act as a JMS client.

■ *Messages* are the objects that communicate information between JMS clients.

■ *Administered objects* are preconfigured JMS objects created by an administrator for the use of clients. The two kinds of JMS administered objects are destinations and connection factories

Figure. illustrates the way these parts interact. Administrative tools allow to bind destinations and connection factories into a JNDI namespace.

- A JMS client can then use resource injection to access the administered objects in the namespace and then establish a logical connection to the same objects through the JMS provider.



Messaging Domains

- Before the JMS API existed, most messaging products supported either the point-to-point or the publish/subscribe approach to messaging.

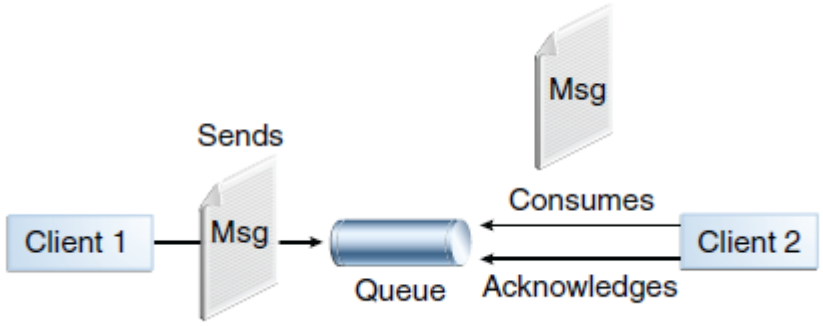
Point-to-Point Messaging Domain

- A *point-to-point* (PTP) product or application is built on the concept of message *queues*,senders, and receivers.
- Each message is addressed to a specific queue, and receiving clients extract messages from the queues established to hold their messages. Queues retain all messages sent to them until the messages are consumed or expire. PTP messaging, illustrated in Figure , has the following characteristics:

■ Each message has only one consumer.

■ A sender and a receiver of a message have no timing dependencies. The receiver can fetch the message whether or not it was running when the client sent the message.

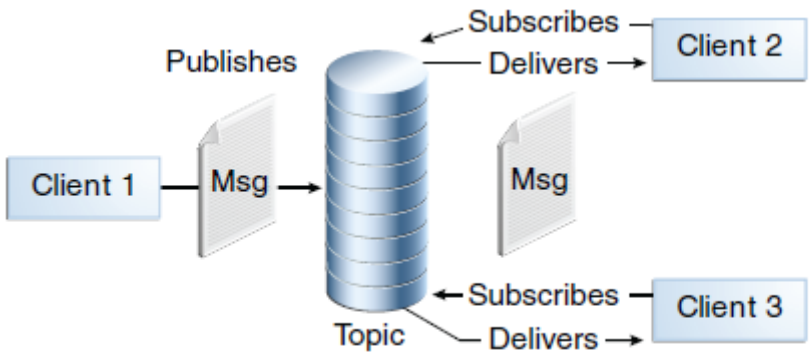
■ The receiver acknowledges the successful processing of a message.



Use PTP messaging when every message send must be processed successfully by one consumer.

Publish/Subscribe Messaging Domain

- In a *publish/subscribe* (pub/sub) product or application, clients address messages to a *topic*, which functions somewhat like a bulletin board. Publishers and subscribers are generally anonymous and can dynamically publish or subscribe to the content hierarchy.
- The system see fig.Point-to-PointMessaging takes care of distributing the messages arriving from a topic's multiple publishers to its multiple subscribers.



Message Consumption

- Messaging products are inherently asynchronous: There is no fundamental timing dependency between the production and the consumption of a message.
- However, the JMS specification uses this term in a more precise sense.Messages can be consumed in either of two ways:

■ **Synchronously**: A subscriber or a receiver explicitly fetches the message from the

destination by calling the receive method.

■ **Asynchronously**: A client can register a *message listener* with a consumer. A message listener is similar to an event listener.

b. Explain any three basic mechanisms for achieving reliable message delivery in JMS API. (3×3 Marks)

Answer:

There are five basic mechanisms for achieving or affecting reliable message delivery are as follows:

i. **Controlling message acknowledgment**: Specify various levels of control over message acknowledgment.
ii. **Specifying message persistence**: Specify that messages are persistent, meaning they must not be lost in the event of a provider failure.
iii. **Setting message priority levels**: Set various priority levels for messages, which can affect the order in which the messages are delivered.
iv. **Allowing messages to expire**: Specify an expiration time for messages so they will not be delivered if they are obsolete.
v. **Creating temporary destinations**: Create temporary destinations that last only for the duration of the connection in which they are created.

i. <u>Controlling Message Acknowledgment</u>

Until a JMS message has been acknowledged, it is not considered to be successfully consumed.

The successful consumption of a message ordinarily takes place in three stages.

1. The client receives the message.

2. The client processes the message.

3. The message is acknowledged. Acknowledgment is initiated either by the JMS provider or by the client, depending on the session acknowledgment mode.

■ Session.AUTO_ACKNOWLEDGE: The session automatically acknowledges a client's receipt of a message either when the client has successfully returned from a call to receive or when the MessageListener it has called to process the message returns successfully.

A synchronous receive in an AUTO_ACKNOWLEDGE session is the one exception to the rule that message consumption is a three-stage process.

■ Session.CLIENT_ACKNOWLEDGE: A client acknowledges a message by calling the message's acknowledge method.

ii. Specifying Message Persistence

The JMS API supports two delivery modes specifying whether messages are lost if the JMS provider fails. These delivery modes are fields of the DeliveryMode interface.

■ The PERSISTENT delivery mode, the default, instructs the JMS provider to take extra care to ensure that a message is not lost in transit in case of a JMS provider failure. A message sent with this delivery mode is logged to stable storage when it is sent.

■ The NON_PERSISTENT delivery mode does not require the JMS provider to store the message or otherwise guarantee that it is not lost if the provider fails.

Specify the delivery mode in either of two ways.

■ Use the setDeliveryMode method of the MessageProducer interface to set the delivery mode for all messages sent by that producer. For example, the following call sets the delivery mode to NON_PERSISTENT for a producer:

```
producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);
```

■ Use the long form of the send or the publish method to set the delivery mode for a specific message. The second argument sets the delivery mode. For example, the following send call sets the delivery mode for message to NON_PERSISTENT:

```
producer.send(message, DeliveryMode.NON_PERSISTENT, 3, 10000);
```

The third and fourth arguments set the priority level and expiration time.

If do not specify a delivery mode, the default is PERSISTENT.

iii. Setting Message Priority Levels

Use message priority levels to instruct the JMS provider to deliver urgent messages first.

Set the priority level in either of two ways.

■ Use the setPriority method of the MessageProducer interface to set the priority level for all messages sent by that producer. For example, the following call sets a priority level of 7 for a prducer:

```
producer.setPriority(7);
```

■ Use the long form of the send or the publish method to set the priority level for a specific message. The third argument sets the priority level. For example, the following send call sets the priority level for message to 3:

producer.send(message, DeliveryMode.NON_PERSISTENT, 3, 10000);

The ten levels of priority range from 0 (lowest) to 9 (highest). If do not specify a priority level, the default level is 4. A JMS provider tries to deliver higher-priority messages before lower-priority ones but does not have to deliver messages in exact order of priority.

iv. Allowing Messages to Expire

By default, a message never expires. If a message will become obsolete after a certain period, however, may want to set an expiration time. Do this in either of two ways.

■ Use the setTimeToLive method of the MessageProducer interface to set a default expiration time for all messages sent by that producer. For example, the following call sets a time to live of one minute for a producer:

producer.setTimeToLive(60000);

■ Use the long form of the send or the publish method to set an expiration time for a specific message. The fourth argument sets the expiration time in milliseconds. For example, the following send call sets a time to live of 10 seconds:

producer.send(message, DeliveryMode.NON_PERSISTENT, 3, 10000);

If the specified timeToLive value is 0, the message never expires.

v. CreatingTemporary Destinations

- Normally, create JMS destinations (queues and topics) administratively rather than programmatically. JMS provider includes a tool to create and remove destinations, and it is common for destinations to be long-lasting.
- The JMS API also enables to create destinations (TemporaryQueue and TemporaryTopicobjects) that last only for the duration of the connection in which they are created. Create these destinations dynamically using the Session.createTemporaryQueue and the Session.createTemporaryTopic methods.
- The only message consumers that can consume from a temporary destination are those created by the same connection that created the destination. Any message producer can send to the temporary destination..
- Use temporary destinations to implement a simple request/reply mechanism.

TEXT-BOOKS

I. Java Network Programming, Merlin Hughes, Michael Shoffner, Derek Hamner, 2nd Edition.

II. Professional AJAX by Nicholas Zakas et alia, Wrox Press.