

UNIT-IV

OVERVIEW

□ Networking Basics:

- Socket
- Factory Methods
- Inet Address

java.net.InetAddress

□ JDBC:

- JDBC Architecture
- JDBC Drivers
- Connecting to Database

java.sql

□ Introduction to Java Servlets:

- Life cycle 6 parts.
- Interfaces and classes in javax.servlet package
- Creating a simple servlet

(servlet)

javax.servlet.http

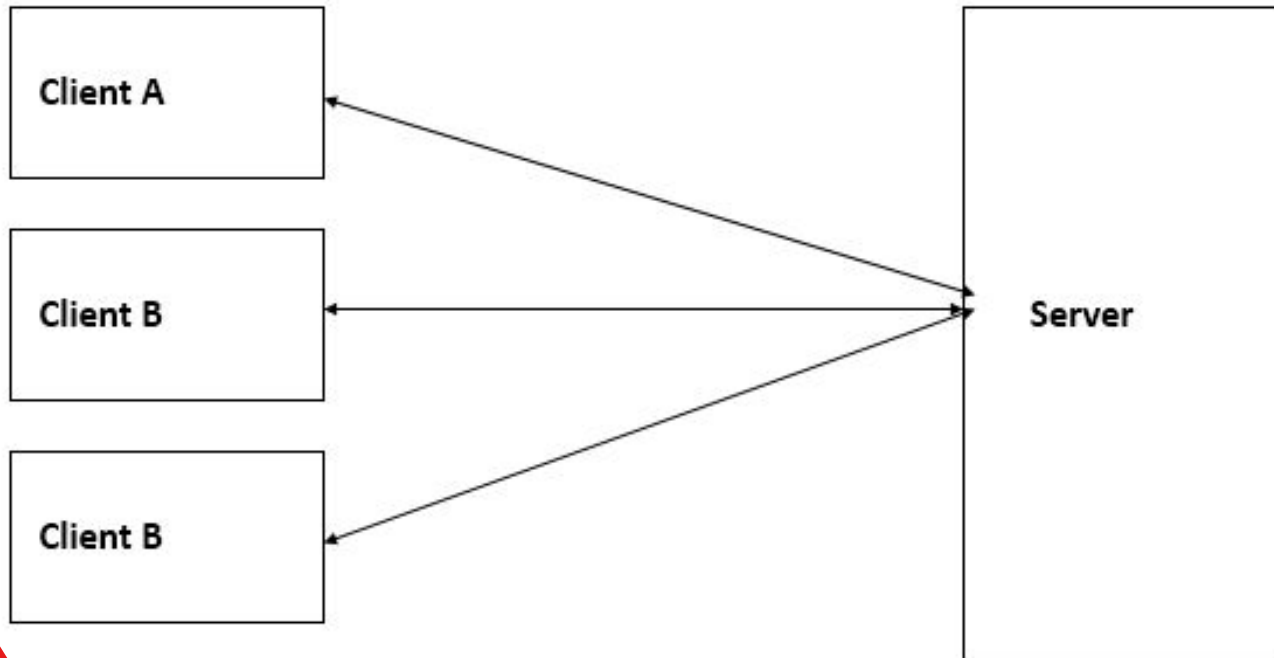


★ WHAT IS CLIENT/SERVER MODEL?

- ✓ ☐ The client server model is an application development architecture designed to separate the presentation of data from its internal processing and storage.
- ✓ ☐ A server is anything that has some resource that can be shared.
- ✓ ☐ A client is simply any entity that wants to gain access to a particular server.
 - ☐ The client request for service and the server services these requests.
 - ☐ The server and the client are not necessary hardware components. They can be programs working on the same machine or on different machines.
 - ☐ The server portion of the client/server application manages the resource shared among multiple users who access the server through multiple clients.
 - ☐ The requests are transferred from the client to the server over the network. The processing that is done by the server is hidden from the client.
 - ☐ In this relation server is permanently available to the client and client can connect when required and can disconnect when served.
- ✓ ☐ One server can service multiple clients.
 - ☐ The communication between the client and server is an important constituent in client/server models, and is usually through the network.

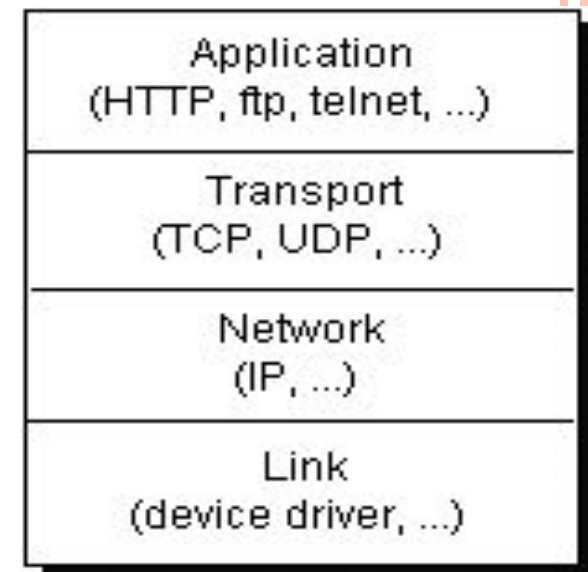


MULTIPLE CLIENT ACCESSING A SERVER



JAVA NETWORKING

- ❑ Java Networking is a concept of connecting two or more computing devices together so that we can share resources.
- ❑ Computers running on the Internet communicate to each other using either the Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP).
- ❑ User don't have to bother about TCP and UDP connectivity Instead, can use the classes in the java.net package
- ❑ These classes provide system-independent network communication



TCP(TRANSMISSION CONTROL PROTOCOL)

- When two applications want to communicate to each other reliably, they establish a connection and send data back and forth over that connection.
- Like telephone line, TCP guarantees that data sent from one end of the connection actually gets to the other end and in the same order it was sent.
- Otherwise, an error is reported. TCP provides a point-to-point channel for applications that require reliable communications.
- TCP (*Transmission Control Protocol*) is a connection-based protocol that provides a reliable flow of data between two computers.



JAVA SOCKET

- A socket is an endpoint between two way communication.
- A *network socket* is a lot like an electrical socket. Various plugs around the network have a standard way of delivering their payload.
- Anything that understands the standard protocol can “plug in” to the socket and communicate.
- A server application normally listens to a specific port waiting for connection requests from a client.
- When a connection request arrives, the client and the server establish a dedicated connection over which they can communicate. During the connection process, the client is assigned a local port number, and binds a *socket* to it.
- The client talks to the server by writing to the socket and gets information from the server by reading from it.
- The server also binds a socket to its local port and communicates with the client by reading from and writing to it.
- The client and the server must agree on a protocol--that is, they must agree on the language of the information transferred back and forth through the socket.



CONTD..

- ✓ ☐ **Definition:** A socket is one end-point of a two-way communication link between two programs running on the network.
- ✓ ☐ The java.net package in the Java development environment provides a class--**Socket**--that represents one end of a two-way connection between your Java program and another program on the network.
- ✓ ☐ The **Socket class** implements the client side of the two-way link.
- ☐ If you are writing server software, you will also be interested in the **ServerSocket** class which implements the server side of the two-way link.



SOCKET & SERVERSOCKET CLASS

- ❑ **Socket**: is the basic, which supports the TCP protocol . TCP is a reliable stream network connection protocol. The socket class provides methods for stream I/O, which makes reading from or writing to a socket easy.
- ❑ **ServerSocket**: is a class used by internet server programs for listening to client requests. ServerSocket does not actually perform the service; instead it creates a Socket object on behalf of the client. The communication is performed through the object created.



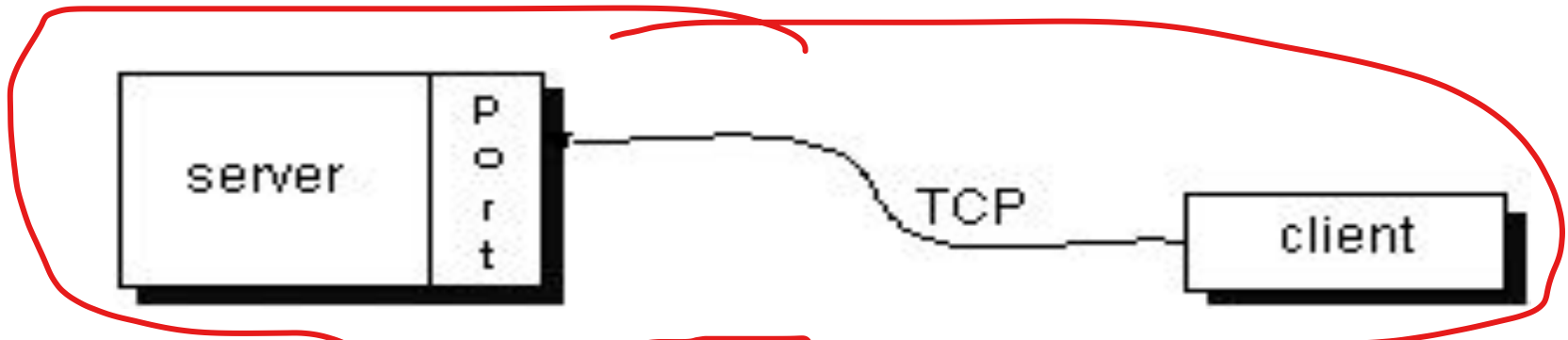
PORTS

- A single socket can serve many different clients at once, as well as serving many different types of information.
- All data destined for a particular computer arrives through that connection.
- ✓ □ However, ~~the data may be intended for different applications~~ running on the computer.
- This feat is managed by the introduction of ports, which is a numbered socket on a particular machine.
- ✓ □ ~~The computer is identified by its 32-bit IP address, which IP uses to deliver data to the right computer on the network.~~
- ✓ □ Ports are identified by a 16-bit number, which TCP and UDP use to deliver the data to the right application.



CONTD..

- A socket address is the combination of an IP address and a port number, much like one end of a telephone connection is the combination of a phone number and a particular extension.
- ✓ □ Based on this address, internet sockets deliver incoming data packets to the appropriate application process or thread.



- Port numbers range from 0 to 65,535 because ports are represented by 16-bit numbers. The port numbers ranging from 0 - 1023 are restricted. ✗
- They are reserved for use by well-known services such as HTTP and FTP and other system services. These ports are called well-known ports. Your applications ~~should not attempt to bind to them.~~



RESERVED SOCKETS

- ✓ ☐ TCP/IP reserves the lower 1,024 ports for specific protocols.
- ✓ ☐ Port number 21 is for FTP, 23 is for Telnet, 25 is for e-mail, 79 is for finger, 80 is for HTTP, 119 is for netnews etc.
- ☐ It is up to each protocol to determine how a client should interact with the port.
- ☐ For example, HTTP is the protocol that web browsers and servers use to transfer hypertext pages and images.



CLIENT SERVER CONNECTIVITY USING TCP

- A server runs on a specific computer and has a socket that is bound to a specific port number. The server just waits, listening to the socket for a client to make a connection request.
- The client knows the hostname of the machine on which the server is running and the port number on which the server is listening.
- To make a connection request, the client request to server on the server's machine and port.
- The client also needs to identify itself to the server so it binds to a local port number that it will use during this connection.



CONTD..

- If everything goes well, the server accepts the connection.
- Upon acceptance, the server gets a new socket bound to the same local port and also has its remote endpoint set to the address and port of the client.
- It needs a new socket so that it can continue to listen to the original socket for connection requests while tending to the needs of the connected client.
- On the client side, if the connection is accepted, a socket is successfully created and the client can use the socket to communicate with the server.
- The client and server can now communicate by writing to or reading from their sockets.
- A server process is listen to a port until a client connects to it.
- To listen multiple clients connections, a server process must be multithreaded.



PROXY SERVERS

- A *proxy server* speaks the client side of a protocol to another server.
- This is often required when clients have certain restrictions on which servers they can connect to.
- Thus, a client would connect to a proxy server, which did not have such restrictions, and the proxy server would in turn communicate for the client.
- A proxy server has the additional ability to filter certain requests or cache the results of those requests for future use.
- A caching proxy HTTP server can help reduce the bandwidth demands on a local network's connection to the Internet.
- When a popular web site is being hit by hundreds of users, a proxy server can get the contents of the web server's popular pages once, saving expensive internetwork transfers while providing faster access to those pages to the clients.



INTERNET ADDRESSING

- Every computer on internet has an address.
- An internet address is a number that uniquely identifies each computer on the Net.
- Internet address that are constituted of 32 bits are called IPV4 (Internet Protocol, version 4), however a new addressing scheme 128 bits , IPV6 (Internet Protocol, version 6) has come into play.
- The main advantage of IPV6 is that it supports a much larger address space than does IPv4. Fortunately, IPv6 is downwardly compatible with IPv4.
- There are 32 bits in an IPv4 IP address, and we often refer to them as a sequence of four numbers between 0 and 255 separated by dots (.) like 123.45.67.244





DOMAIN NAME SYSTEM

- The Internet wouldn't be a very friendly place to navigate if everyone had to refer to their addresses as numbers.
- For example, it is difficult to imagine seeing “http://192.9.9.1/” at the bottom of an advertisement.
- Just as the four numbers of an IP address describe a network hierarchy from left to right, the name of an Internet address, called its *domainname*, describes a machine's location in a name space, from right to left.
- For example, www.osborne.com is in the COM domain (reserved for U.S. commercial sites), it is called osborne (after the company name), and www is the name of the specific computer that is Osborne's web server. www corresponds to the rightmost number in the equivalent IP address.



INETADDRESS CLASS

- ❑ The InetAddress class is used to encapsulate both the numerical IP address and the domain name for that address.
- ❑ You interact with this class by using the name of an IP host, which is more convenient and understandable than its IP address.
- ❑ The InetAddress class hides the number inside.
- ❑ As of Java 2, version 1.4, InetAddress can handle both IPv4 and IPv6 addresses.
- ❑ The java.net.InetAddress class provides methods to get the IP of any host name *for example* www.javatpoint.com, www.google.com, www.facebook.com etc.



COMMONLY USED METHODS OF INETADDRESS CLASS

Method	Description
<code>public static InetAddress <u>getByName</u>(String host)</code> throws <code>UnknownHostException</code>	it returns the instance of <code>InetAddress</code> containing <u>LocalHost</u> IP and name. <i>(Remote Host A)</i>
<code>public static InetAddress <u>getLocalHost</u>()</code> throws <code>UnknownHostException</code>	it returns the instance of <code>InetAddress</code> containing <u>local</u> host name and address.
<code>public String <u>getHostName</u>()</code>	it returns the <u>host name</u> of the IP address.
<code>public String <u>getHostAddress</u>()</code>	it returns the <u>IP address</u> in string format.


EXAMPLE-

```
import java.io.*;
import java.net.*;
public class InetDemo{
public static void main(String[] args){
try{
InetAddress ip=InetAddress.getByName("www.javatpoint.com");

System.out.println("Host Name: "+ip.getHostName());
System.out.println("IP Address: "+ip.getHostAddress());
}catch(Exception e){System.out.println(e);}
}
}
```

Output:

```
Host Name: www.javatpoint.com
IP Address: 206.51.231.148
```



FACTORY METHODS

- A factory method is one whose return type is similar to the class name in which class is present.
- **Rules for writing factory method**
 - ✓ The return type of the factory method must be similar to class name in which class it presents.
 - ✓ Every factory method in java is static.
 - ✓ The access specifier of the factory method must be public.
- The **InetAddress** class has no visible constructors.
- To create an **InetAddress** object, you have to use one of the available factory methods.
- Three commonly used InetAddress factory methods are:

1. static *InetAddress* **getLocalHost()** throws **UnknownHostException**
2. static *InetAddress* **getByName** (*String* *hostname*) throws **UnknownHostException**
3. static *InetAddress*[] **getAllByName** (*String* *hostname*) throws **UnknownHostException**

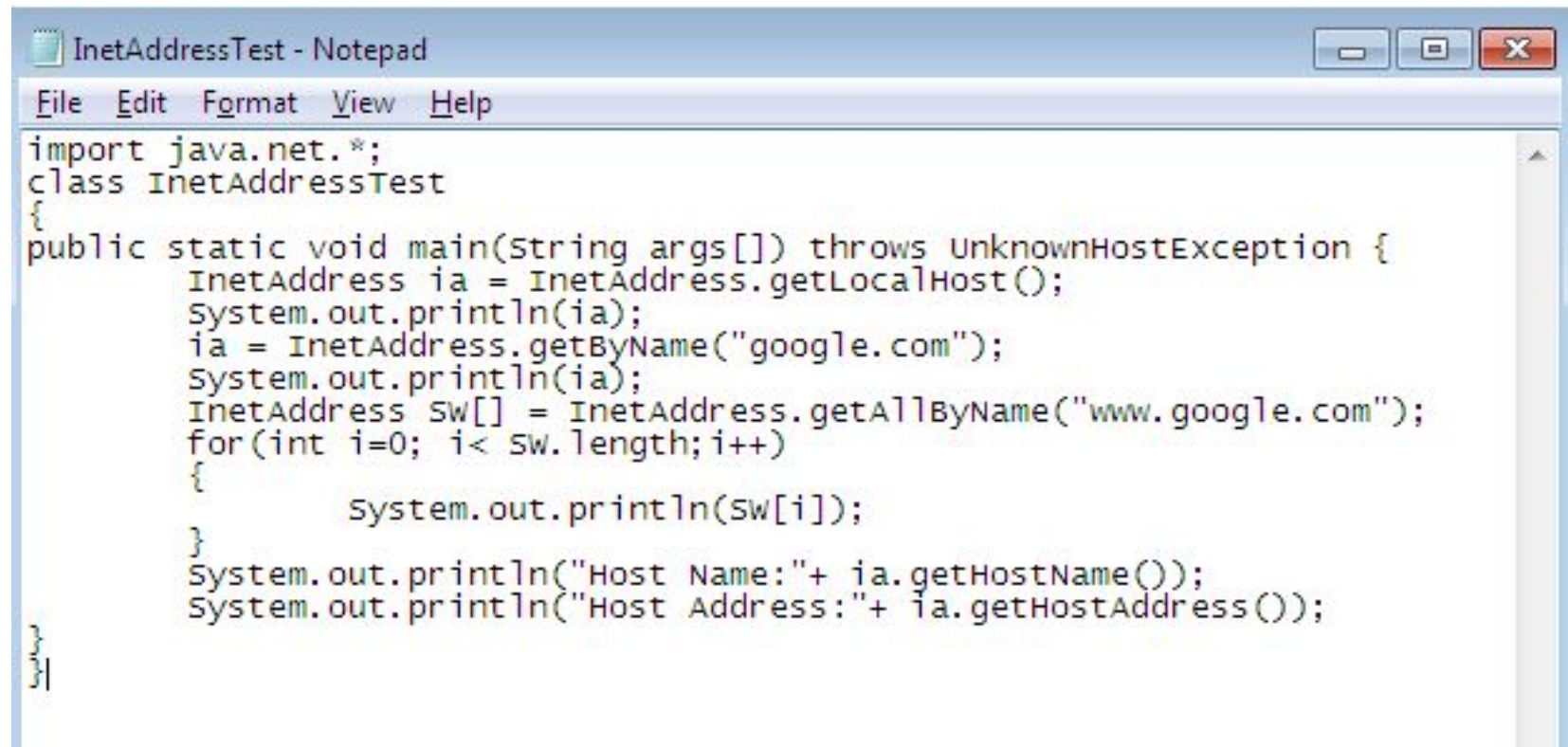


CONTD...

- ❑ The **getLocalHost()** method simply returns the **InetAddress** object that represents the local host.
- ❑ The **getByName()** method returns an **InetAddress** for a host name passed to it.
- ❑ If these methods are unable to resolve the host name, they throw an **UnknownHostException**.
- ❑ On the Internet, it is common for a single name to be used to represent several machines.
- ❑ The **getAllByName()** factory method returns an array of **InetAddresses** that represent all of the addresses that a particular name resolves to.
- ❑ It will also throw an **UnknownHostException** if it can't resolve the name to at least one address.



EXAMPLE-



```
import java.net.*;
class InetAddressTest
{
    public static void main(String args[]) throws UnknownHostException {
        InetAddress ia = InetAddress.getLocalHost();
        System.out.println(ia);
        ia = InetAddress.getByName("google.com");
        System.out.println(ia);
        InetAddress sw[] = InetAddress.getAllByName("www.google.com");
        for(int i=0; i< sw.length; i++)
        {
            System.out.println(sw[i]);
        }
        System.out.println("Host Name:" + ia.getHostName());
        System.out.println("Host Address:" + ia.getHostAddress());
    }
}
```

Output:

NehaG-PC-103|192.168.0.107

google.com/216.58.220.206

www.google.com/216.58.220.196

Host Name:google.com

Host Address:216.58.220.206



TCP/IP CLIENT SOCKETS

- Socket class can be used to connect java's I/O system to other program that may reside either on the local machine or on any other machine on the internet.
- There are two types of TCP socket in java:
 - One for server (**ServerSocket** class is designed to be a “listener “ , which waits for the client to connect before doing anything)
 - Other is for client. (**Socket** class is designed to connect to server sockets and initiate protocol exchange.)
- The creation of Socket object implicitly establishes a connection between the client and server.
- There are no methods and constructor that explicitly expose the details of establishing that connection.
- Here are two constructors used to create client sockets:
 - **Socket(String *hostName*, int *port*)**
 - **Socket(InetAddress *ipAddress*, int *port*)**



CONTD..

- ❑ **Socket(String *hostName*, int *port*):** Creates a socket connecting the local host to the named host and port; can throw an **UnknownHostException** or an **IOException**.
- ❑
- ❑ **Socket(InetAddress *ipAddress*, int *port*):** Creates a socket using a pre-existing **InetAddress** object and a port; can throw an **IOException**.
- ❑ A socket can be examined at any time for the address and port information associated with it, by use of the following methods:
 - **InetAddress getAddress():** Returns the **InetAddress** associated with the **Socket** object.
 - **int getPort():** Returns the remote port to which this **Socket** object is connected.
 - **int getLocalPort():** Returns the local port to which this **Socket** object is connected.



CONTD...

- Once the **Socket** object has been created, it can also be examined to gain access to the input and output streams associated with it.
- Each of these methods can throw an **IOException** if the sockets have been invalidated by a loss of connection on the Net.
- These streams are used exactly like the I/O streams to send and receive data.
 - **InputStream getInputStream()**: Returns the **InputStream** associated with the invoking socket.
 - **OutputStream getOutputStream()**: Returns the **OutputStream** associated.



SERVERSOCKET CLASS

- ❑ The ServerSocket class can be used to create a server socket.
- ❑ This object is used to establish communication with the clients.

Method	Description
1) <code>public Socket accept()</code>	returns the socket and establish a connection between server and client.
2) <code>public synchronized void close()</code>	closes the server socket.



TOTAL 4 PROGRAMS ARE GIVEN IN **TCP/IP** PROTOCOL BASED COMMUNICATION.

APPLICATION NUMBER	FUNCTIONALITY
1st application	Client to server communication (one-way)
2nd application	Server to client communication (one-way)
3rd application	Server sends file contents to client (two-way, non-continuous)
4th application	Chat program (two-way, continuous)

In our next slide we will see the 4th application of TCP/IP



EXPLANATION-

- ❑ **Chat communication** is the process of exchanging messages between two systems continuously.
- ❑ Anyone can break the communication. Both systems come with the following same responsibilities:
 - **Reading from keyboard.** Uses an input stream like **BufferedReader** connected to `System.in`.
 - **Sending data(assume client)** to the other system what is read from keyboard. Uses an output stream like **PrintWriter** connected to **getOutputStream()** method of `Socket`.
 - **Receiving data(assume server)** from the other system. Uses an input stream like **BufferedReader** connected to **getInputStream()** method of `Socket`.



CONNECTING CLIENT TO THE SERVER(ONE WAY COMMUNICATION)

```
import java.net.*;
import java.io.*;
public class client{
    public static void
    main(String[]args)throws
    IOException{

    System.out.println("client
    started...");
    Socket socket=new
    Socket("localhost",1234);

    }
}
```

```
import java.io.*;
import java.net.*;
public class server{
    public static void
    main(String[]args)throws
    IOException{

    System.out.println("waiting for the
    client...");
    ServerSocket serverSocket=new
    ServerSocket(1234);
    Socket
    socket=serverSocket.accept();//estab
    lishes connection and waits for the
    client
    System.out.println("Connection
    established");

    }

}
//establishes connection and waits
for the client
```



```
C:\Users\hp\Documents\DSEU 4th Sem\JAVA\Solved Practical Java>javac client.java
```

```
C:\Users\hp\Documents\DSEU 4th Sem\JAVA\Solved Practical Java>java client  
client started...
```

```
C:\Users\hp\Documents\DSEU 4th Sem\JAVA\Solved Practical Java>javac server.java
```

```
C:\Users\hp\Documents\DSEU 4th Sem\JAVA\Solved Practical Java>java server  
waiting for the client...
```

```
Connection established client can send messages now
```

```
C:\Users\hp\Documents\DSEU 4th Sem\JAVA\Solved Practical Java>
```



SERVER SIDE CODING FOR MESSAGE PASSING(TWO WAY COMMUNICATION)

```
TestServer - Notepad
File Edit Format View Help
import java.io.*;
import java.net.*;
public class TestServer
{
    public static void main(String[] args) throws Exception
    {
        serversocket sersock = new Serversocket(3000);
        System.out.println("server ready for chatting");
        Socket sock = sersock.accept();
        // reading from keyboard (keyRead object)
        BufferedReader keyRead = new BufferedReader(new InputStreamReader(System.in));
        // sending to client (pwrite object)
        OutputStream ostream = sock.getOutputStream();
        PrintWriter pwrite = new PrintWriter(ostream, true);

        // receiving from server (receiveRead object)
        InputStream istream = sock.getInputStream();
        BufferedReader receiveRead = new BufferedReader(new InputStreamReader(istream));

        String receiveMessage, sendMessage;
        while(true)
        {
            if((receiveMessage = receiveRead.readLine()) != null)
            {
                System.out.println(receiveMessage);
            }
            sendMessage = keyRead.readLine();
            pwrite.println(sendMessage);
            pwrite.Flush();
        }
    }
}
```



CLIENT SIDE CODING FOR MESSAGE PASSING(TWO WAY COMMUNICATION)

```
TestClient - Notepad
File Edit Format View Help
import java.net.*;
import java.io.*;
public class TestClient
{
    public static void main(String[] args) throws Exception
    {
        Socket sock = new Socket("127.0.0.1", 3000);
        // reading from keyboard (keyRead object)
        BufferedReader keyRead = new BufferedReader(new InputStreamReader(System.in));
        // sending to client (pwrite object)
        OutputStream ostream = sock.getOutputStream();
        PrintWriter pwrite = new PrintWriter(ostream, true);

        // receiving from server (receiveRead object)
        InputStream istream = sock.getInputStream();
        BufferedReader receiveRead = new BufferedReader(new InputStreamReader(istream));

        System.out.println("Start the chitchat, type and press Enter key");

        String receiveMessage, sendMessage;
        while(true)
        {
            sendMessage = keyRead.readLine(); // keyboard reading
            pwrite.println(sendMessage);      // sending to server
            pwrite.flush();                   // flush the data
            if((receiveMessage = receiveRead.readLine()) != null) //receive from server
            {
                System.out.println(receiveMessage); // displaying at DOS prompt
            }
        }
    }
}
```

STEPS TO RUN-

- ❑ Open two windows of cmd prompt.
- ❑ In first window compile server program.
- ❑ In second window, compile client program.
- ❑ Go back to first window to run server program (we are running server program first because server should be ready for accepting the request from client).
- ❑ In second window, run client program.



URL

- The *URL* provides a reasonably intelligible form to uniquely identify or address information on the Internet.
- URLs are universal, every browser uses them to identify information on the Web.
- Two examples of URLs are **http://www.osborne.com/** and **http://www.osborne.com:80/index.htm.**
- A URL specification is based on four components:-
 - The first is the protocol to use, separated from the rest of the locator by a colon (:). Protocol may be HTTP, SMTP, NNTP(Network News Transfer Protocol), FTP or gopher.
 - The second component is the host name or IP address of the host to use; this is delimited on the left by double slashes (//) and on the right by a slash(/).
 - The third component, the port number, is an optional parameter, delimited on the left from the host name by a colon (:) and on the right by a slash (/). (It defaults to port 80)
 - The fourth part is the actual file path.



CONTD...

- Following are some of the constructors of URL class:
 - URL(String): Creates a URL object from the String representation
 - URL(String, String, int, String): Creates a URL object from the specified protocol , host, port number, and file.
 - URL(String, String, String): Creates an absolute URL from the specified protocol name, host name, and filename
 - URL(URL, String): Creates a URL by parsing the specification spec within a specified context



URL CODE-

```
URLDemo - Notepad
File Edit Format View Help

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

```
D:\Jan2019\JavaCourseMaterial\Downloaded\Networks\Networks>javac URLDemo.java
```

```
D:\Jan2019\JavaCourseMaterial\Downloaded\Networks\Networks>java URLDemo
```

```
Protocol: http
```

```
Port: -1
```

```
Host: www.osborne.com
```

```
File: /downloads
```

```
Ext:http://www.osborne.com/downloads
```

```
D:\Jan2019\JavaCourseMaterial\Downloaded\Networks\Networks>
```

PARSING A URL

- URL (Uniform Resource Locator) is a reference (an address) to a resource on the Internet
- Java programs can use a class called URL in the java.net package to represent a URL address.
- The URL class provides several methods that let you query URL objects:

- getPath() - Returns the path component of this URL.
- getQuery() - Returns the query component of this URL.
- getFile() - Returns the filename component of the URL.

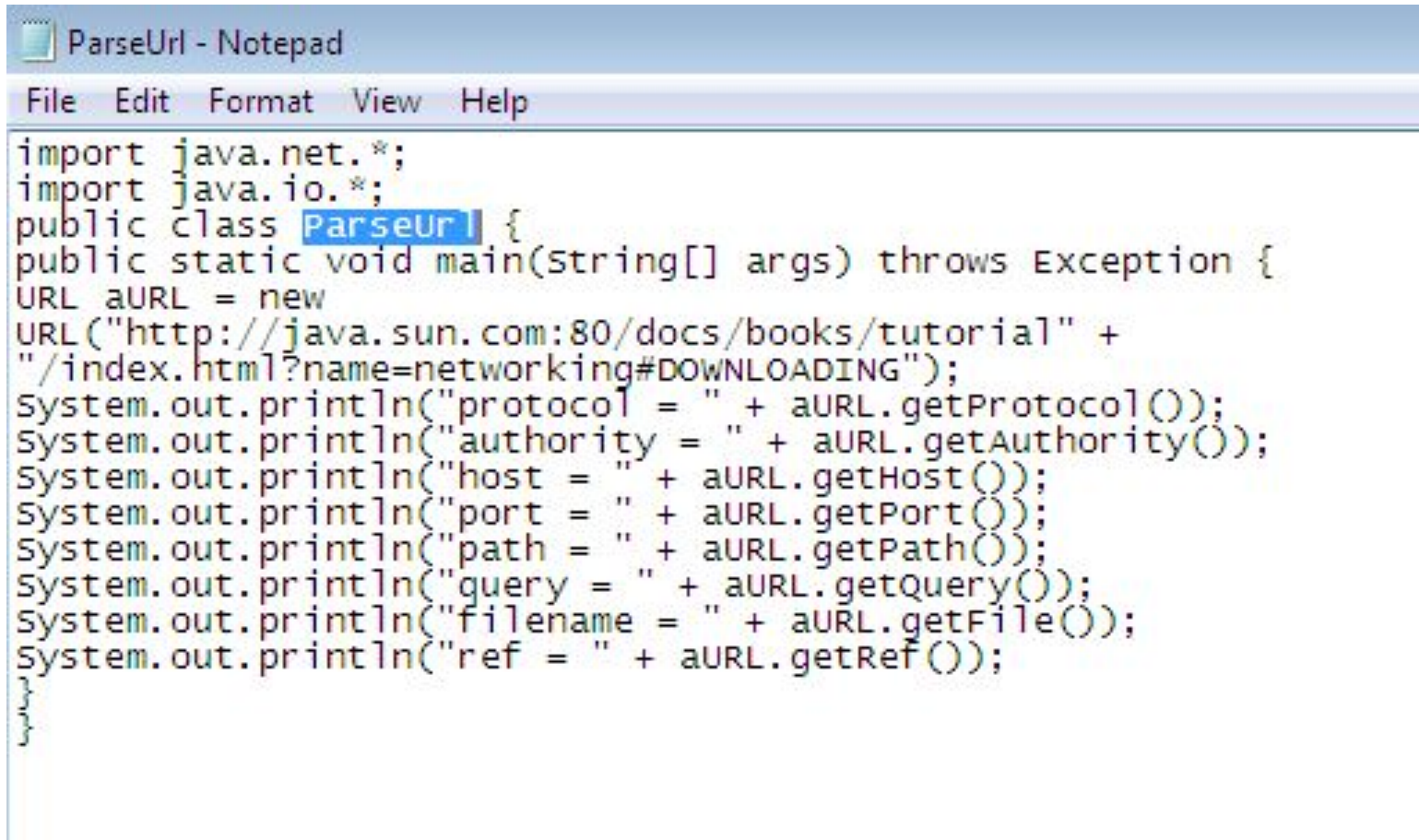
The getFile method returns the same as getPath, plus the concatenation of the value of getQuery, if any.

- getRef()- Returns the reference component of the URL.
- getProtocol() - Returns the protocol identifier component of the URL.
- getHost() - Returns the host name component of the URL.
- getPort() - Returns the port number component of the URL.

The getPort method returns an integer that is the port number. If the port is not set, getPort returns -1.



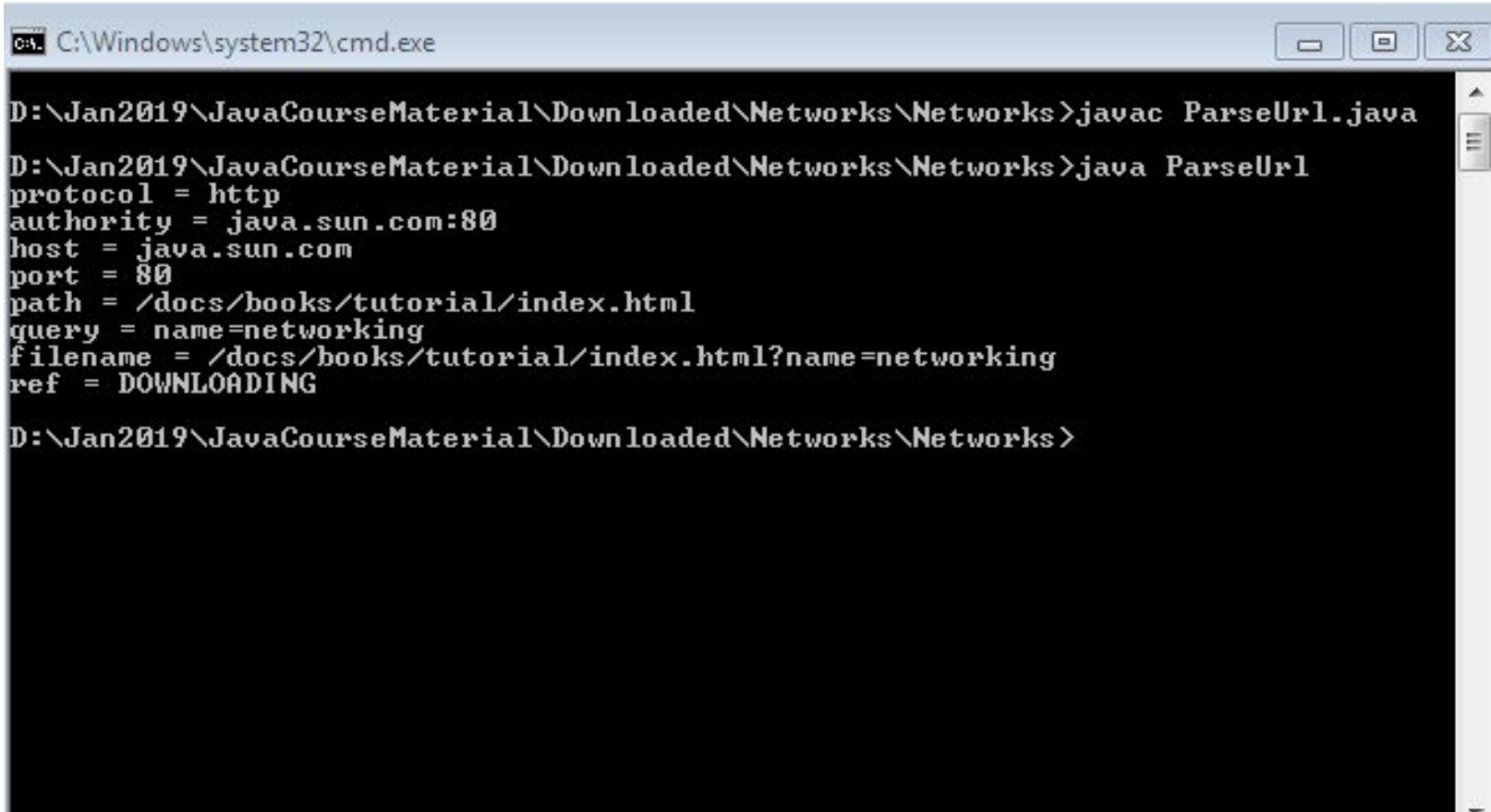
EXAMPLE-



```
import java.net.*;
import java.io.*;
public class ParseUrl {
public static void main(String[] args) throws Exception {
URL aURL = new
URL("http://java.sun.com:80/docs/books/tutorial" +
"/index.html?name=networking#DOWNLOADING");
System.out.println("protocol = " + aURL.getProtocol());
System.out.println("authority = " + aURL.getAuthority());
System.out.println("host = " + aURL.getHost());
System.out.println("port = " + aURL.getPort());
System.out.println("path = " + aURL.getPath());
System.out.println("query = " + aURL.getQuery());
System.out.println("filename = " + aURL.getFile());
System.out.println("ref = " + aURL.getRef());
}
```



OUTPUT-

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Windows\system32\cmd.exe'. The command prompt is at the directory 'D:\Jan2019\JavaCourseMaterial\Downloaded\Networks\Networks'. The user has executed 'javac ParseUrl.java' and then 'java ParseUrl'. The program outputs several variables: protocol = http, authority = java.sun.com:80, host = java.sun.com, port = 80, path = /docs/books/tutorial/index.html, query = name=networking, filename = /docs/books/tutorial/index.html?name=networking, and ref = DOWNLOADING.

```
C:\Windows\system32\cmd.exe

D:\Jan2019\JavaCourseMaterial\Downloaded\Networks\Networks>javac ParseUrl.java

D:\Jan2019\JavaCourseMaterial\Downloaded\Networks\Networks>java ParseUrl
protocol = http
authority = java.sun.com:80
host = java.sun.com
port = 80
path = /docs/books/tutorial/index.html
query = name=networking
filename = /docs/books/tutorial/index.html?name=networking
ref = DOWNLOADING

D:\Jan2019\JavaCourseMaterial\Downloaded\Networks\Networks>
```



★ JDBC

- ✓ ☐ JDBC stands for Java Database Connectivity.
- ✓ ☐ JDBC is a Java API to connect and execute the query with the database.
- ✓ ☐ JDBC works with Java on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX.
- ☐ JDBC stands for **Java Database Connectivity**, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.
- ☐ The JDBC library includes APIs for each of the tasks commonly associated with database usage:
 - Making a connection to a database
 - Creating SQL or MySQL statements
 - Executing that SQL or MySQL queries in the database
 - Viewing & Modifying the resulting records



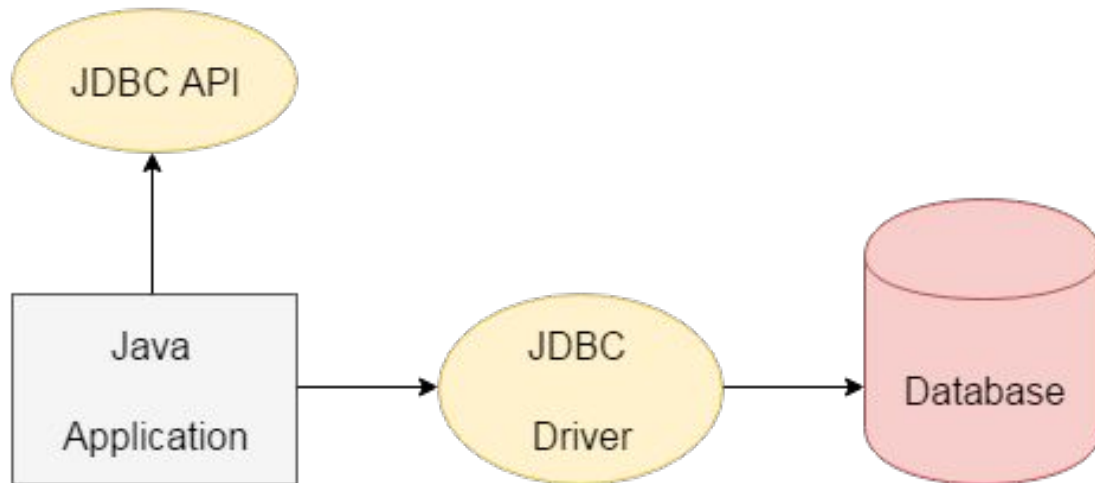
CONTD...

- We can use JDBC API to access tabular data stored in any relational database. By the help of JDBC API, we can save, update, delete and fetch data from the database.
- It is like Open Database Connectivity (ODBC) provided by Microsoft.



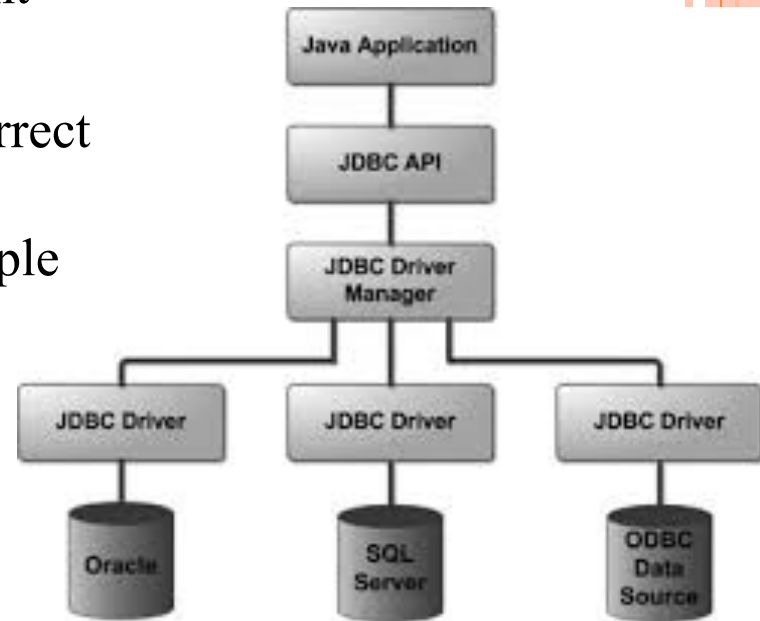
JDBC ARCHITECTURE

- The JDBC API supports both two-tier and three-tier processing models for database access but in general JDBC Architecture consists of two layers:
 - **JDBC API:** This provides the application-to-JDBC Manager connection.
 - **JDBC Driver API:** This supports the JDBC Manager-to-Driver Connection.



CONTD..

- ❑ The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases.
- ❑ The JDBC driver manager ensures that the correct driver is used to access each data source. The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases.
- ❑ Following is the architectural diagram, which shows the location of the driver manager with respect to the JDBC drivers and the Java application:



COMMON JDBC COMPONENTS:

- ❑ The JDBC API provides the following interfaces and classes:
- ❑ **Driver Manager:** This class manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication sub protocol. The first driver that recognizes a certain sub protocol under JDBC will be used to establish a database Connection.
- ❑ **Driver:** This interface handles the communications with the database server. You will interact directly with Driver objects very rarely. Instead, you use Driver Manager objects, which manages objects of this type. It also abstracts the details associated with working with Driver objects



CONTD...

- ❑ **Connection** : This interface with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only.
- ❑ **Statement** : You use objects created from this interface to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.
- ❑ **ResultSet**: These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.
- ❑ **SQLException**: This class handles any errors that occur in a database application.



WHAT IS JDBC DRIVER ?

- JDBC drivers implement the defined interfaces in the JDBC API for interacting with your database server.
- For example, using JDBC drivers enable you to open database connections and to interact with it by sending SQL or database commands then receiving results with Java.
- The Java.sql package that ships with JDK contains various classes with their behaviors defined and their actual implementations are done in third-party drivers. Third party vendors implements the java.sql.Driver interface in their database driver.

□ There are 4 types of JDBC drivers:

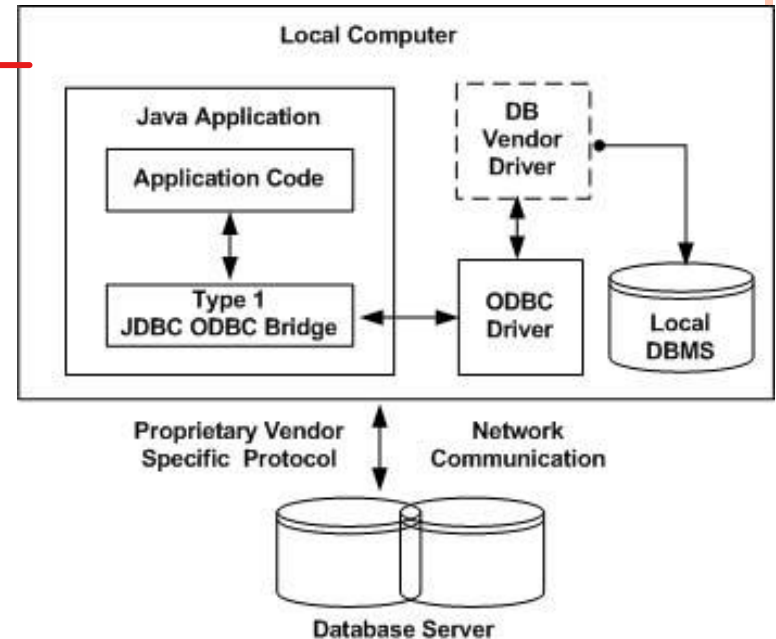
- JDBC-ODBC bridge driver ⁺¹
- Native-API driver (partially java driver) ²
- Network Protocol driver (fully java driver) ^{x3}
- Thin driver (fully java driver) ⁴

JDBC NCP (Pure Java)
100% Pure Java



JDBC-ODBC BRIDGE DRIVER

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



CONTD...

Advantage

- Easy to use
- Allow easy connectivity to all database supported by the ODBC Driver.

Disadvantage

- Slow execution time
- Dependent on ODBC Driver.
- Uses Java Native Interface(JNI) to make ODBC call.

NATIVE-API DRIVER

- In a Type 2 driver, JDBC API calls are converted into native C/C++ API calls which are unique to the database.
- These drivers typically provided by the database vendors and used in the same manner as the JDBC-ODBC Bridge, the vendor-specific driver must be installed on each client machine.
- If we change the Database we have to change the native API as it is specific to a database and they are mostly obsolete now but you may realize some speed increase with a Type 2 driver, because it eliminates ODBC's overhead.



CONTD...

Advantage

- faster as compared to **Type-1 Driver**
- Contains additional features.

Disadvantage

- Requires native library
- Increased cost of Application

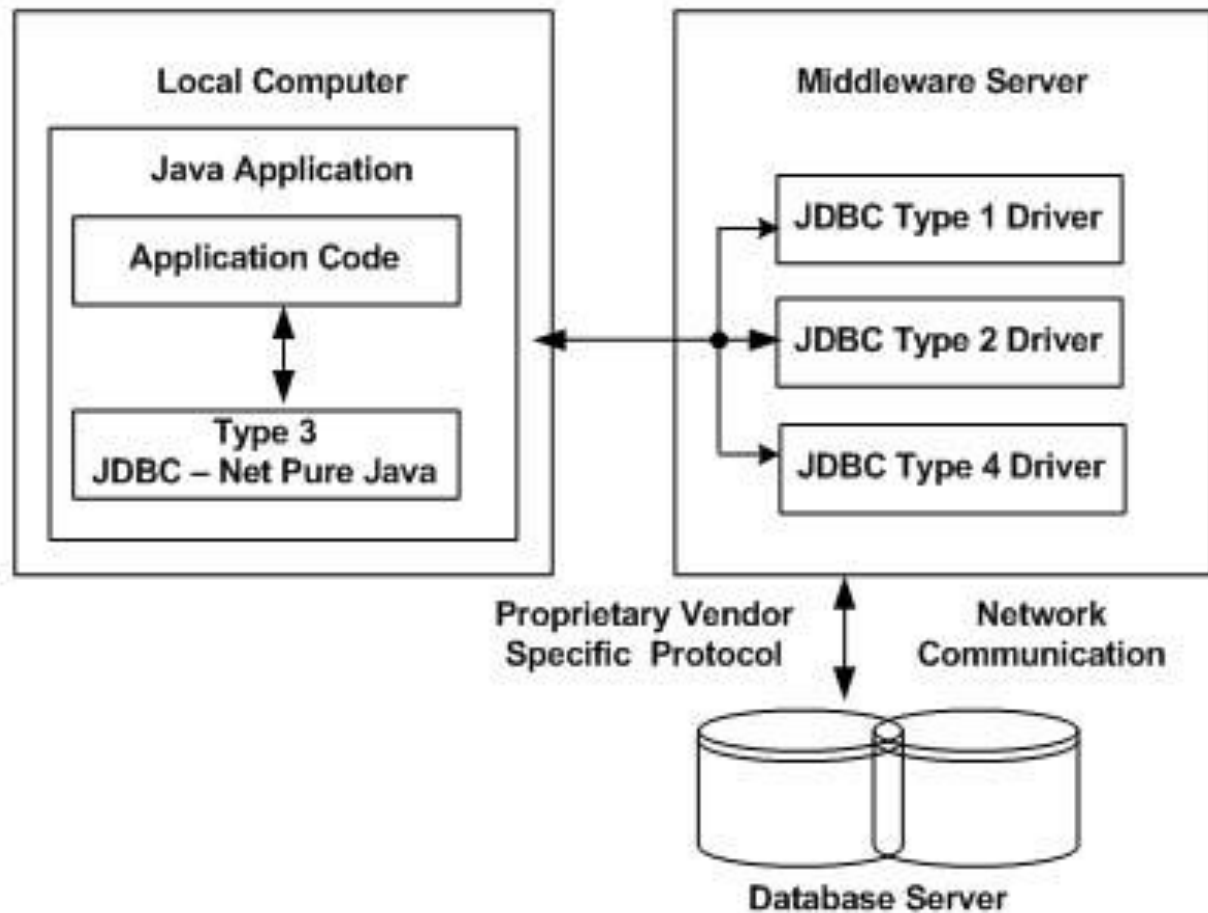


JDBC-NET PURE JAVA:

- In a Type 3 driver, a three-tier approach is used to accessing databases.
- The JDBC clients use standard network sockets to communicate with middleware application server.
- The socket information is then translated by the middleware application server into the call format required by the DBMS, and forwarded to the database server.
- This kind of driver is extremely flexible, since it requires no code installed on the client and a single driver can actually provide access to multiple databases.



CONTD....



CONTD...

- You can think of the application server as a JDBC "proxy," meaning that it makes calls for the client application.
- As a result, you need some knowledge of the application server's configuration in order to effectively use this driver type.
- Your application server might use a Type 1, 2, or 4 driver to communicate with the database, understanding the nuances will prove helpful.

Advantage

- Does not require any native library to be installed.
- Database Independency.
- Provide facility to switch over from one database to another database.

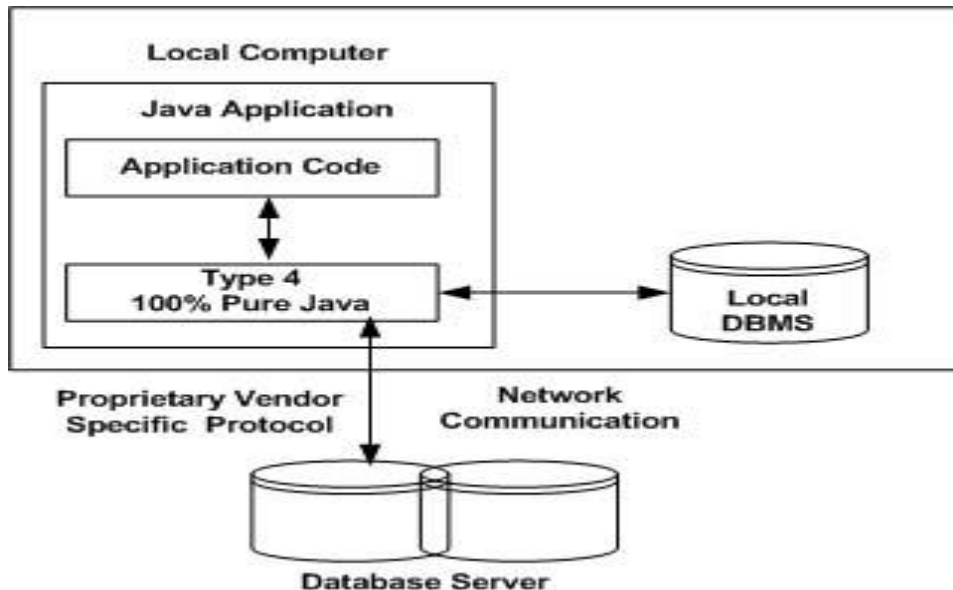
Disadvantage

- Slow due to increase number of network call.



100% PURE JAVA:

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



CONTD...

- MySQL's Connector/J driver is a Type 4 driver. Because of the proprietary nature of their network protocols, database vendors usually supply type 4 drivers.

Advantage

- Does not require any native library.
- Does not require any Middleware server.
- Better Performance than other driver.

Disadvantage

- Slow due to increase number of network call.



WHICH DRIVER SHOULD BE USED?

- If you are accessing one type of database, such as Oracle, SQLbase, or IBM, the preferred driver type is 4.
- If your Java application is accessing multiple types of databases at the same time, type 3 is the preferred driver.
- Type 2 drivers are useful in situations, where a type 3 or type 4 driver is not available yet for your database.
- The type 1 driver is not considered a deployment-level driver, and is typically used for development and testing purposes only.



HOW TO LOAD DRIVERS

Two ways to load the jar file:

1. Paste the mysqlconnector.jar file in c:/programfiles/java/ jre/lib/ext folder,


Download the mysqlconnector.jar file. Go to jre/lib/ext folder and paste the jar file here.






2. Set classpath.

Go to environment variable then click on new tab. In variable name write **classpath** and in variable value paste the path to the mysqlconnector.jar file by appending mysqlconnector.jar;; as C:\folder\mysql-connector-java-5.0.8-bin.jar;;



DATABASE CONNECTIVITY

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

-  Register the driver class
-  Creating connection
-  Creating statement
-  Executing queries
-  Closing connection



1) REGISTER THE DRIVER CLASS

- The forName() method of Class class is used to register the driver class.
- This method is used to dynamically load the driver class.
- ~~□~~ Syntax of forName() method
public static void forName(String className) throws ClassNotFoundException
- Example to register the OracleDriver
class-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
 - ❑ *public static Connection getConnection(String url)throws SQLException*
 - ❑ *public static Connection getConnection(String url,String name,String password) throws SQLException*
 - ❑ Example to establish connection with the Oracle database:
 - ❑
 - ❑ Connection con= DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","password");
-



3) *CREATE THE STATEMENT OBJECT*

- ❑ The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.
- ❑ Syntax of createStatement() method
- ❑ *public Statement createStatement()throws SQLException*
- ❑ Example to create the statement object
- ❑ *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
- *public ResultSet executeQuery(String sql) throws SQLException*
- Example to execute query
- *ResultSet rs=stmt.executeQuery("select * from emp");*
- *while(rs.next()){*
- *System.out.println(rs.getInt(1)+" "+rs.getString(2));*
- *}*



5) *CLOSE THE CONNECTION OBJECT*

- By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.
- Syntax of close() method
- *public void close()throws SQLException*
- Example to close connection
- con.close();



JAVA DATABASE CONNECTIVITY WITH MYSQL

- To connect Java application with the MySQL database, we need to follow 5 following steps.
- example we are using MySql as the database. So we need to know following informations for the mysql database:
 1. **Driver class:** The driver class for the mysql database is **com.mysql.jdbc.Driver**.
 2. **Connection URL:** The connection URL for the mysql database is **jdbc:mysql://localhost:3306/mydb** where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address, 3306 is the port number and sonoo is the database name. We may use any database, in such case, we need to replace the sonoo with our database name.
 3. **Username:** The default username for the mysql database is **root**.
 4. **Password:** It is the password given by the user at the time of installing the mysql database. In this example, we are going leave empty Double Quotes""as the password.



PROGRAM FOR JDBC-

```
import java.sql.*;
public class MysqlCon{
public static void main(String args[]){
try{
Class.forName("com.mysql.jdbc.Driver");
Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb","root","");
//here sonoo is database name, root is username and password
Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery("select * from myguests");
while(rs.next())
System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
con.close();
}catch(Exception e){ System.out.println(e);}
}
}
```



```
C:\Users\hp\Documents\DSEU 4th Sem\JAVA\Solved Practical Java>javac MysqlCon.java
```

```
C:\Users\hp\Documents\DSEU 4th Sem\JAVA\Solved Practical Java>java MysqlCon
```

```
1 John Doe  
2 Aman Verma  
3 Pankaj Kapoor  
4 Dev Arora
```

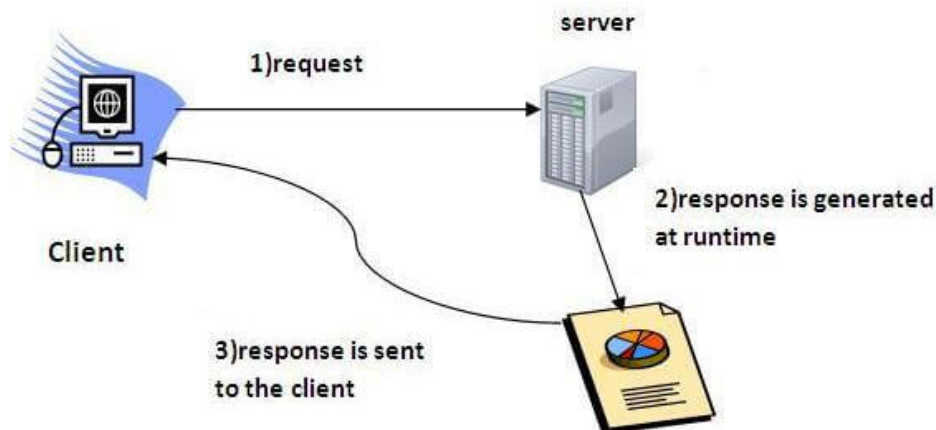
```
C:\Users\hp\Documents\DSEU 4th Sem\JAVA\Solved Practical Java>java MysqlCon
```

```
1 John Doe  
2 Aman Verma  
3 Pankaj Kapoor  
4 Dev Arora  
5 Ankit Kapoor
```

```
C:\Users\hp\Documents\DSEU 4th Sem\JAVA\Solved Practical Java>
```

SERVLETS

- ❑ **Servlet** technology is used to create a web application (resides at server side and generates a dynamic web page).
- ❑ Servlets have access to the entire family of Java APIs, including the JDBC API to access enterprise databases.
- ❑ **Servlet** technology is robust and scalable because of java language.
- ❑ Before Servlet, CGI (Common Gateway Interface) scripting language was common as a server-side programming language.
- ❑ Just as applets dynamically extend the functionality of a Web browser, servlets dynamically extend the functionality of a Web server.



WHAT IS A SERVLET?

- Servlet can be described in many ways, depending on the context.
 - **Servlet** is a technology which is used to create a web application.
 - **Servlet** is an API that provides many interfaces and classes including documentation.
 - **Servlet** is an interface that must be implemented for creating any Servlet.
 - **Servlet** is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests.
 - **Servlet** is a web component that is deployed on the server to create a dynamic web page.



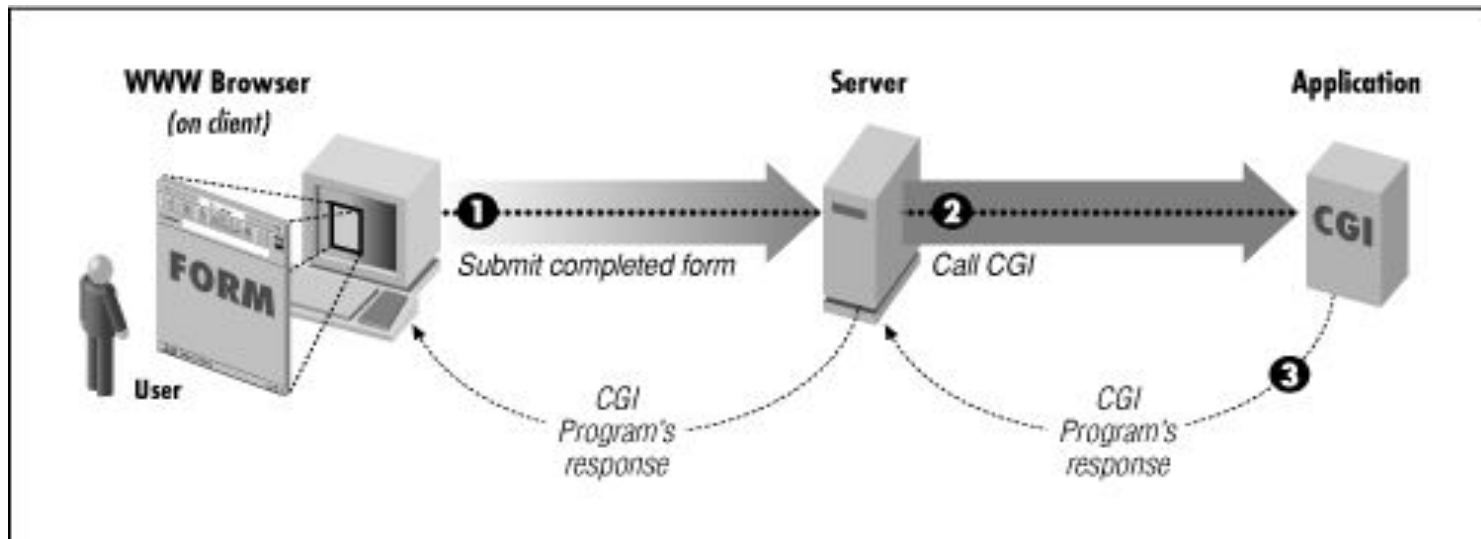
WHAT IS A WEB APPLICATION?

- A web application is an application accessible from the web.
- A web application is composed of web components like Servlet, JSP, Filter, etc. and other elements such as HTML, CSS, and JavaScript.
- The web components typically execute in Web Server and respond to the HTTP request.



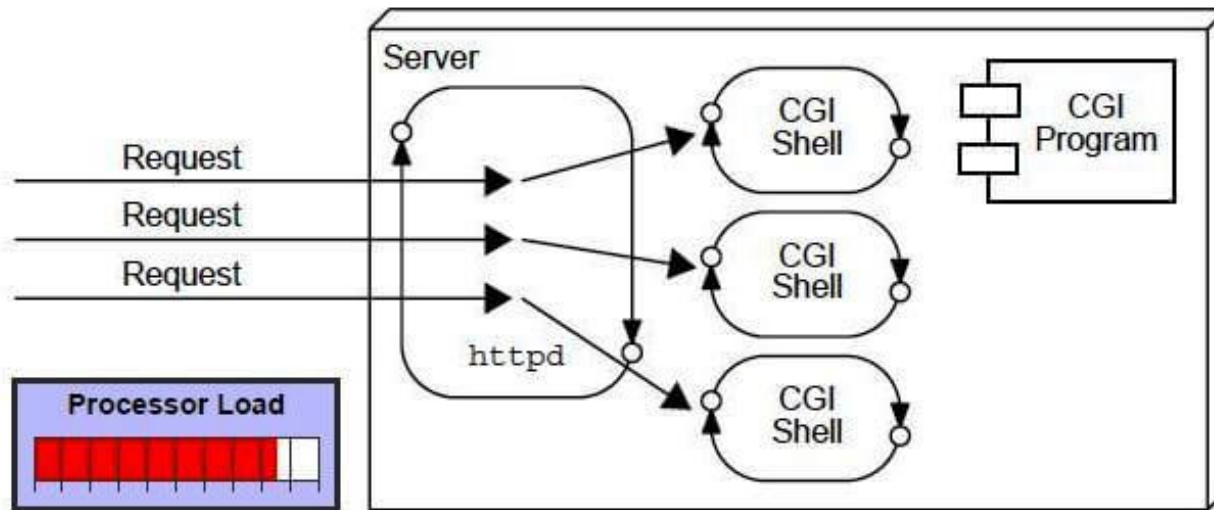
CGI (COMMON GATEWAY INTERFACE)

- ✓ CGI is an interface which tells the webserver how to pass data to and from an application.
- CGI is the part of the Web server that can communicate with other programs running on the server.
- With CGI, the Web server can call up a program, while passing user-specific data to the program
- The program then processes that data and the server passes the program's response back to the Web browser.



DISADVANTAGES OF CGI

- ❑ There are many problems in CGI technology:
 - If the number of clients increases, it takes more time for sending the response.
 - For each request, it starts a process, and the web server is limited to start processes.
 - It uses platform dependent language e.g. C, C++, perl.

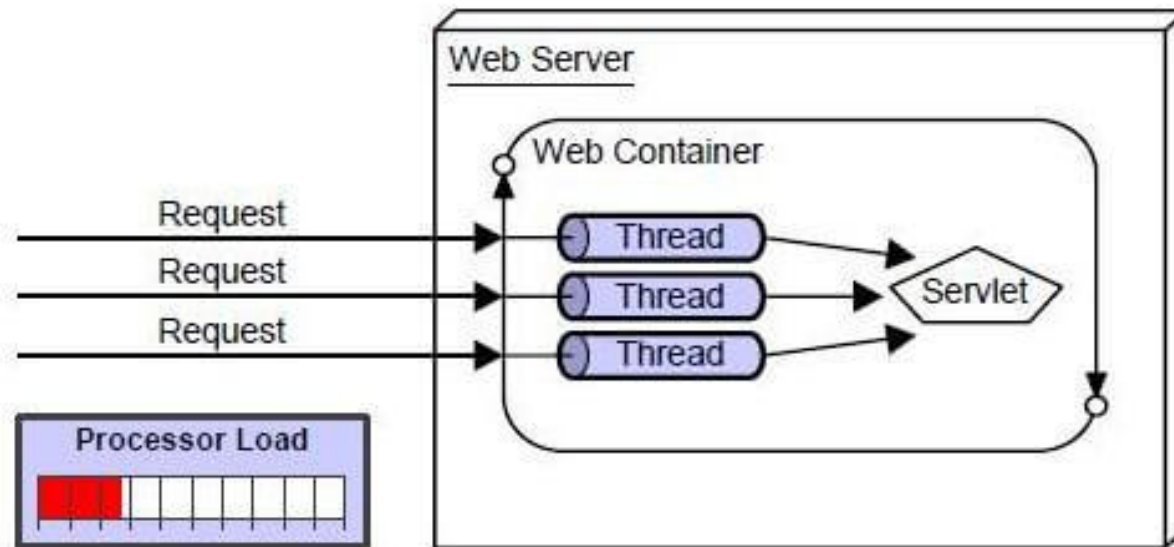


ADVANTAGES OF SERVLET

- There are many advantages of Servlet over CGI.
- The web container creates threads for handling the multiple requests to the Servlet.
- Threads have many benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low. The advantages of Servlet are as follows:
 - **Better performance**: because it creates a thread for each request, not process.
 - **Portability**: because it uses Java language.
 - **Robust**: JVM manages Servlets, so we don't need to worry about the memory leak, garbage collection, etc.
 - **Secure**: because it uses java language.



CONTD...



LIFE CYCLE OF A SERVLET

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

- Servlet class is loaded.
- Servlet instance is created.
- init method is invoked.
- service method is invoked.]
- destroy method is invoked.

□ There are three states of a servlet:

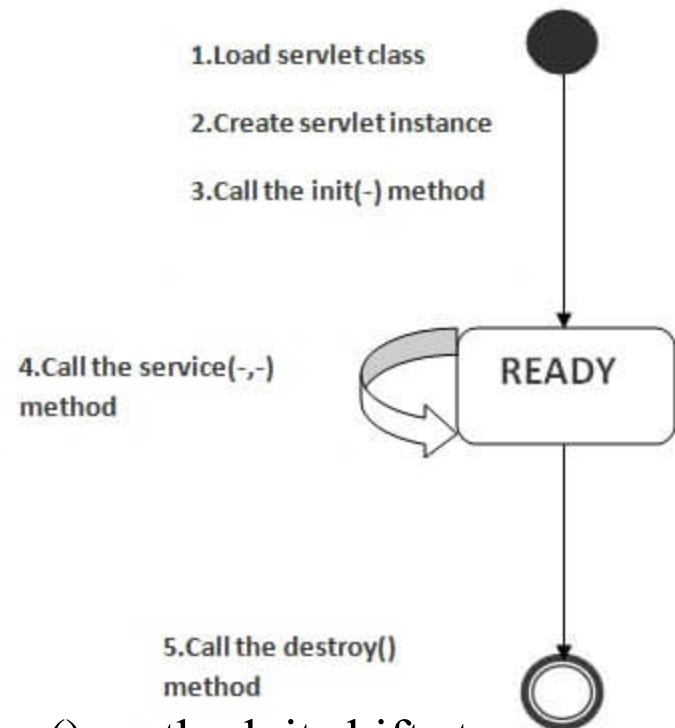
□ new, ready and end

□ The servlet is in new state if servlet instance is created.

□ After invoking the init() method, Servlet comes in the ready state.

□ In the ready state, servlet performs all the tasks.

□ When the web container invokes the destroy() method, it shifts to the end state.



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()
```

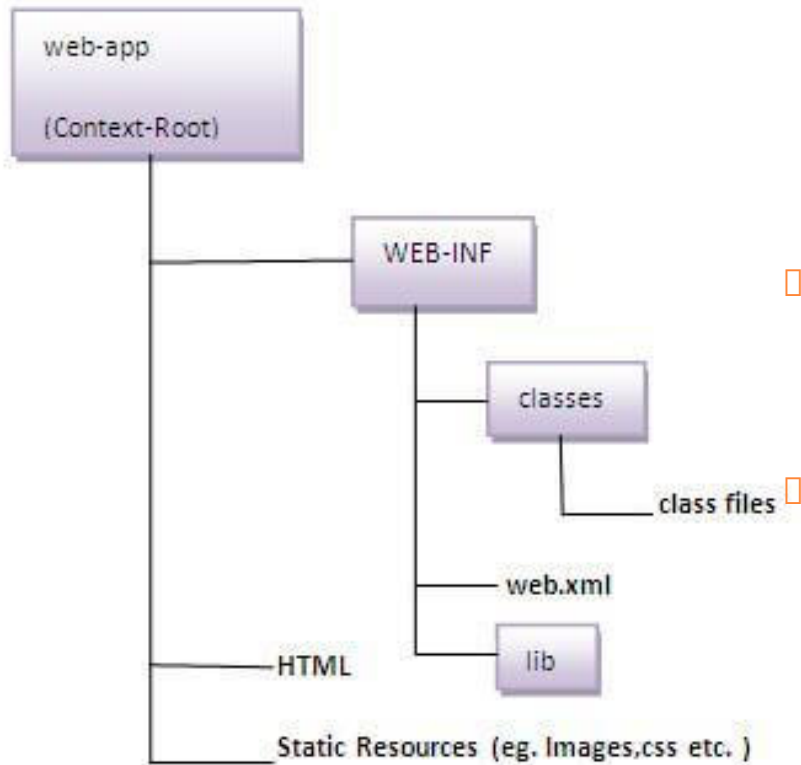


STEPS TO CREATE A SERVLET EXAMPLE

- There are given 6 steps to create a **servlet example**. These steps are required for all the servers.
- The servlet example can be created by three ways:
 - By implementing Servlet interface,
 - By inheriting GenericServlet class, (or)
 - By inheriting HttpServlet class
- The mostly used approach is by extending HttpServlet because it provides http request specific method such as doGet(), doPost(), doHead() etc.
- Here, we are going to use apache tomcat server in this example. The steps are as follows:
 1. Create a directory structure
 2. Create a Servlet
 3. Compile the Servlet
 4. Create a deployment descriptor
 5. Start the server and deploy the project
 6. Access the servlet



1) CREATE A DIRECTORY STRUCTURES



- The **directory structure** defines that where to put the different types of files so that web container may get the information and respond to the client.
- Let's see the directory structure that must be followed to create the servlet.
- As you can see that the servlet class file must be in the classes folder. The web.xml file must be under the WEB-INF folder.



2)CREATE A SERVLET

- There are three ways to create the servlet:
 - By implementing the Servlet interface
 - By inheriting the GenericServlet class
 - By inheriting the HttpServlet class
- In this example, we are inheriting the HttpServlet class and providing the implementation of the doGet() method. Notice that get request is the default request.

DemoServlet.java

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

public class DemoServlet extends HttpServlet{
    public void doGet(HttpServletRequest req,HttpServletResponse res)
        throws ServletException,IOException
    {
        res.setContentType("text/html");//setting the content type
        PrintWriter pw=res.getWriter();//get the stream to write the data

        //writing html in the stream
        pw.println("<html><body>");
        pw.println("Welcome to servlet");
        pw.println("</body></html>");

        pw.close();//closing the stream
    }
}
```

3) COMPILE THE SERVLET

- For compiling the Servlet, jar file is required to be loaded. Different Servers provide different jar files:

Jar file	Server
1) <u>servlet-api.jar</u>	<u>Apache Tomcat</u>
2) weblogic.jar	Weblogic
3) javaee.jar	Glassfish
4) javaee.jar	JBoss

- Two ways to load the jar file:
 - set classpath
 - paste the jar file in JRE/lib/ext folder
- Put the java file in any folder. After compiling the java file, paste the class file of servlet in **WEB-INF/classes** directory



4) CREATE THE DEPLOYMENT DESCRIPTOR (WEB.XML FILE)

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

web.xml file

```
<web-app>

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

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

</web-app>
```

DESCRIPTION OF THE ELEMENTS OF WEB.XML FILE

<web-app> represents the whole application.

<servlet> is sub element of <web-app> and represents the servlet.

<servlet-name> is sub element of <servlet> represents the name of the servlet.

<servlet-class> is sub element of <servlet> represents the class of the servlet.

<servlet-mapping> is sub element of <web-app>. It is used to map the servlet.

<url-pattern> is sub element of <servlet-mapping>. This pattern is used at client side to invoke the servlet



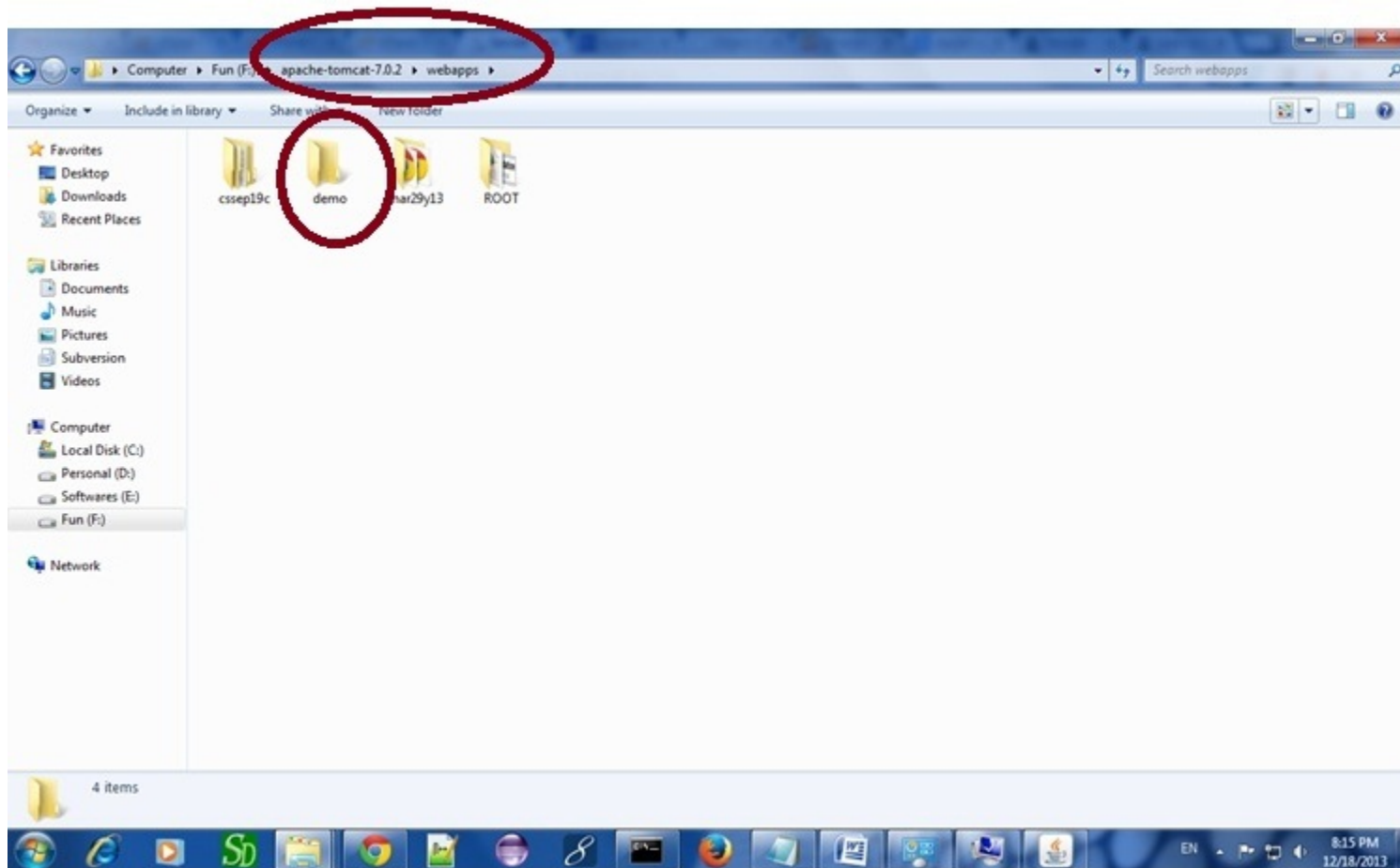
5) START THE SERVER AND DEPLOY THE PROJECT

- To start Apache Tomcat server, double click on the startup.bat file under apache-tomcat/bin directory.
- For make it working, set JAVA_HOME or JRE_HOME in environment variable.
- Note: Go to My Computer properties -> Click on advanced tab then environment variables -> Click on the new tab of user variable -> Write JAVA_HOME in variable name and paste the path of jdk folder in variable value -> ok -> ok -> ok.
- Now server is started successfully.



5) HOW TO DEPLOY THE SERVLET PROJECT

- Copy the project and paste it in the webapps folder under apache tomcat.



6) HOW TO ACCESS THE SERVLET

- ❑ Open browser and write `http://hostname:portno/contextroot/urlpatternofservlet`. For example:
- ❑ `http://localhost:9999/demo/welcome`

