Dr. Ambedkar Institute of Technology, Bengaluru
Department of Computer Science & Engg.
UG Programme

Subject Name: JAVA Programming
Subject Code: 18CS52

PRESENTATION BY

Dr. SMITHA SHEKAR B

ASSOCIATE PROFESSOR

DEPT. OF C.S.E

DR. AMBEDKAR INSTITUTE OF TECHNOLOGY

BANGALORE-560056

Agenda

- Networking Basics
- Networking Classes and Interfaces
- TCP/IP Client Sockets
- TCP/IP Server Sockets

Dr. Smitha Shekar B

Java Networking

- Java Networking is a concept of connecting two or more computing devices together so that we can share resources.

- Java socket programming provides facility to share data between different computing devices.

- Advantage of Java Networking
  - sharing resources
  - centralize software management

Dr. Smitha Shekar B

Java Networking Terminology

- IP Address
- Protocol
- Port Number
- MAC Address
- Connection-oriented and Connection-less protocol
- Socket

Dr. Smitha Shekar B

IP Address

- A key component of the Internet is the *address*. Every computer on the Internet has one. An Internet address is a number that uniquely identifies each computer on the Net.
- IP address is a unique number assigned to a node of a network e.g. 192.168.0.1 . It is composed of octets that range from 0 to 255.
- It is a logical address that can be changed.
- Originally, all Internet addresses consisted of 32-bit values, organized as four 8-bit values.
- This address type was specified by IPv4 (Internet Protocol, version 4). However, a new addressing scheme, called IPv6 (Internet Protocol, version 6) has come into play.

- Just as the numbers of an IP address describe a network hierarchy, the name of an Internet address, called its *domain name,* describes a machine's location in a name space.

- For example, **www.HerbSchildt.com** is in the *COM* top-level domain (reserved for U.S. commercial sites); it is called *HerbSchildt*, and *www* identifies the server for web requests.

- An Internet domain name is mapped to an IP address by the *Domain Naming Service (DNS).*

- This enables users to work with domain names, but the Internet operates on IP addresses.

Protocol

- A protocol is a set of rules basically that is followed for communication.
- HTTP, TCP, FTP, Telnet, SMTP, POP etc.

- For example, HTTP is the protocol that web browsers and servers use to transfer hypertext pages and images. It is a quite simple protocol for a basic page-browsing web server.

- When a client requests a file from an HTTP server, an action known as a *hit*, it simply sends the name of the file in a special format to a predefined port and reads back the contents of the file.

- The server also responds with a status code to tell the client whether or not the request can be fulfilled and why.

Port Number

- The port number is used to uniquely identify different applications. It acts as a communication endpoint between applications.
- The port number is associated with the IP address for communication between two applications.
- Once a connection has been established, a higher-level protocol ensues, which is dependent on which port you are using.
- TCP/IP reserves the lower 1,024 ports for specific protocols. Many of these will seem familiar to you if you have spent any time surfing the Internet.
- Port number 21 is for FTP; 23 is for Telnet; 25 is for e-mail; 43 is for whois; 80 is for HTTP; 119 is for netnews—and the list goes on. It is up to each protocol to determine how a client should interact with the port.

Dr. Smitha Shekar B

MAC Address

- MAC (Media Access Control) Address is a unique identifier of NIC (Network Interface Controller).
- A network node can have multiple NIC but each with unique MAC.

Connection-oriented and connection-less protocol

- In connection-oriented protocol, acknowledgement is sent by the receiver.
- So it is reliable but slow. The example of connection-oriented protocol is **TCP**.
- But, in connection-less protocol, acknowledgement is not sent by the receiver.
- So it is not reliable but fast. The example of connection-less protocol is **UDP**.

Socket

- At the core of Java's networking support is the concept of a *socket*.
- A socket identifies an endpoint in a network. The socket paradigm was part of the 4.2BSD Berkeley UNIX release in the early 1980s. Because of this, the term *Berkeley socket* is also used.
- Sockets are at the foundation of modern networking because a socket allows a single computer to serve many different clients at once, as well as to serve many different types of information.
- This is accomplished through the use of a *port,* which is a numbered socket on a particular machine. A server process is said to "listen" to a port until a client connects to it.
- A server is allowed to accept multiple clients connected to the same port number, although each session is unique.
- To manage multiple client connections, a server process must be multithreaded or have some other means of multiplexing the simultaneous I/O.

- Socket communication takes place via a protocol. *Internet Protocol (IP)* is a low-level routing protocol that breaks data into small packets and sends them to an address across a network, which does not guarantee to deliver said packets to the destination.
- *Transmission Control Protocol* (TCP) is a higher-level protocol that manages to robustly string together these packets, sorting and retransmitting them as necessary to reliably transmit data.
- A third protocol, *User Datagram Protocol (UDP),* sits next to TCP and can be used directly to support fast, connectionless, unreliable transport of packets.

- The term *network programming* refers to writing programs that execute across multiple devices (computers), in which the devices are all connected to each other using a network.
- The java.net package provides support for the two common network protocols −
  - **TCP** − TCP stands for Transmission Control Protocol, which allows for reliable communication between two applications. TCP is typically used over the Internet Protocol, which is referred to as TCP/IP.
  - **UDP** − UDP stands for User Datagram Protocol, a connection-less protocol that allows for packets of data to be transmitted between applications.

java.net package

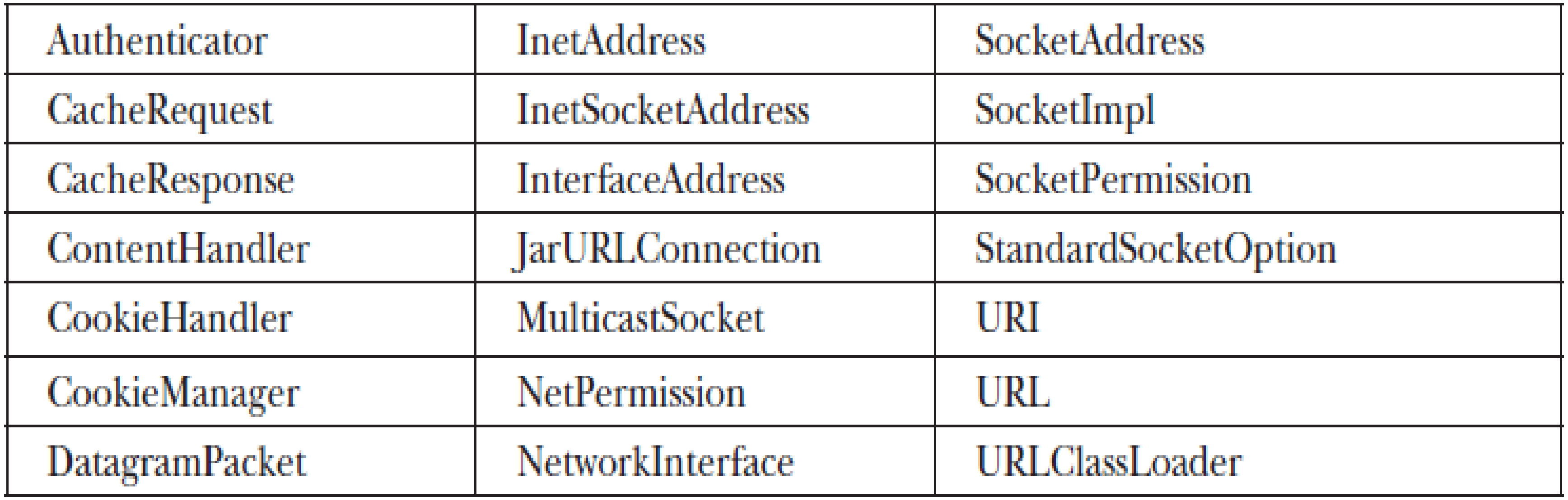
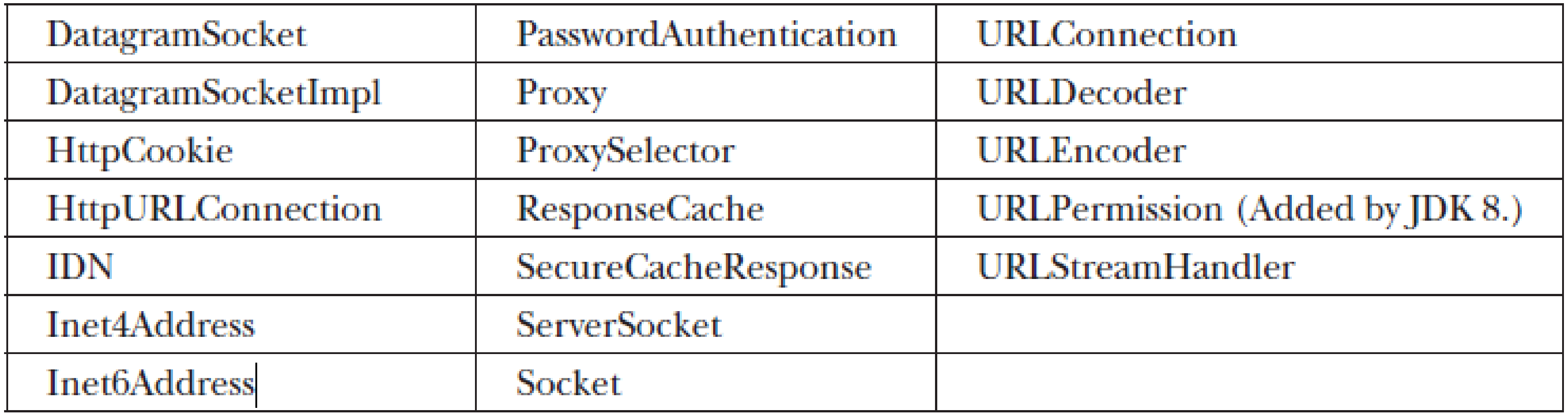
The java.net package provides many classes to deal with networking applications in Java.
A list of these classes is given below

- **Authenticator**
- **CacheRequest**
- **CacheResponse**
- **ContentHandler**
- **CookieHandler**
- **CookieManager**
- **DatagramPacket**
- **DatagramSocket**
- **DatagramSocketImpl**
- **InterfaceAddress**
- **JarURLConnection**
- **MulticastSocket**
- **InetSocketAddress**
- **InetAddress**
- **Inet4Address**
- **Inet6Address**
- **IDN**
- **HttpURLConnection**
- **HttpCookie**
- **NetPermission**
- **NetworkInterface**
- **PasswordAuthentication**
- **Proxy**
- **ProxySelector**
- **ResponseCache**
- **SecureCacheResponse**
- **ServerSocket**
- **Socket**
- **SocketAddress**
- **SocketImpl**
- **SocketPermission**
- **StandardSocketOptions**
- **URI**
- **URL**
- **URLClassLoader**
- **URLConnection**
- **URLDecoder**
- **URLEncoder**
- **URLStreamHandler**

The Networking Classes and Interfaces

- Java supports TCP/IP both by extending the already established stream I/O interface and by adding the features required to build I/O objects across the network.
- Java supports both the TCP and UDP protocol families.
- TCP is used for reliable stream-based I/O across the network. UDP supports a simpler, hence faster, point-to-point datagram-oriented model.
- The classes contained in the **java.net** package

The Networking Classes

Authenticator	InetAddress	SocketAddress
CacheRequest	InetSocketAddress	SocketImpl
CacheResponse	InterfaceAddress	SocketPermission
ContentHandler	JarURLConnection	StandardSocketOption
CookieHandler	MulticastSocket	URI
CookieManager	NetPermission	URL
DatagramPacket	NetworkInterface	URLClassLoader
DatagramSocket	PasswordAuthentication	URLConnection
DatagramSocketImpl	Proxy	URLDecoder
HttpCookie	ProxySelector	URLEncoder
HttpURLConnection	ResponseCache	URLPermission (Added by JDK 8.)
IDN	SecureCacheResponse	URLStreamHandler
Inet4Address	ServerSocket	
Inet6Address	Socket	

Dr. Smitha Shekar B

Interfaces

ContentHandlerFactory	FileNameMap	SocketOptions
CookiePolicy	ProtocolFamily	URLStreamHandlerFactory
CookieStore	SocketImplFactory	
DatagramSocketImplFactory	SocketOption	

NOTE

- **Socket Programming** – This is the most widely used concept in Networking
- **URL Processing** -

Dr. Smitha Shekar B

Socket Programming

- Sockets provide the communication mechanism between two computers using TCP. A client program creates a socket on its end of the communication and attempts to connect that socket to a server.

- When the connection is made, the server creates a socket object on its end of the communication. The client and the server can now communicate by writing to and reading from the socket.

- The java.net.Socket class represents a socket, and the java.net.ServerSocket class provides a mechanism for the server program to listen for clients and establish connections with them.

Dr. Smitha Shekar B

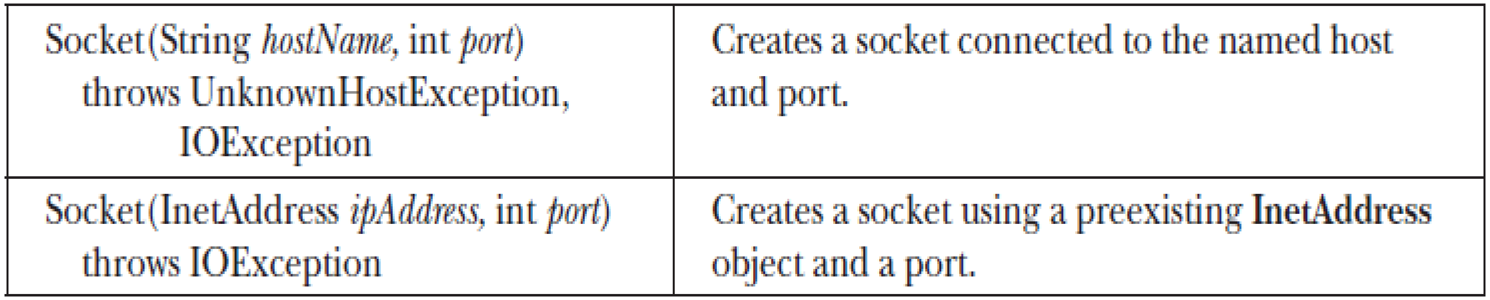
TCP/IP Client Sockets

- TCP/IP sockets are used to implement reliable, bidirectional, persistent, point-to-point, stream-based connections between hosts on the Internet.
- A socket can be used to connect Java's I/O system to other programs that may reside either on the local machine or on any other machine on the Internet.

- There are two kinds of TCP sockets in Java. One is for servers, and the other is for clients.
- The **ServerSocket** class is designed to be a "listener," which waits for clients to connect before doing anything.
- Thus, **ServerSocket** is for servers. The **Socket** class is for clients. It is designed to connect to server sockets and initiate protocol exchanges.
- Because client sockets are the most commonly used by Java applications, they are examined here.

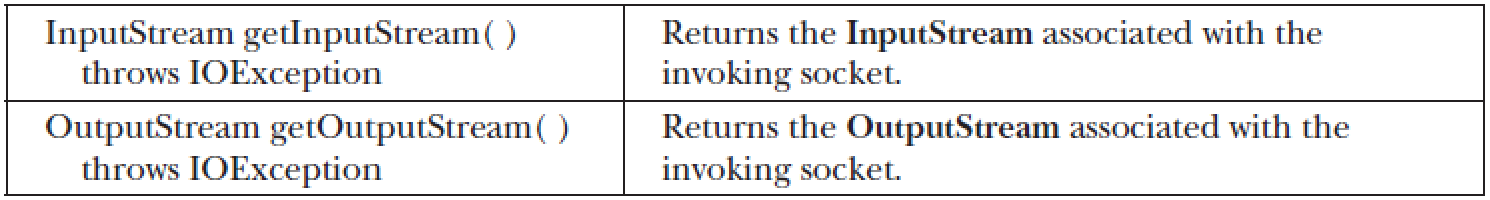
- The creation of a **Socket** object implicitly establishes a connection between the client and server.
- There are no methods or constructors that explicitly expose the details of establishing that connection.

Two constructors used to create client sockets



Socket(String *hostName*, int *port*) throws UnknownHostException, IOException	Creates a socket connected to the named host and port.
Socket(InetAddress *ipAddress*, int *port*) throws IOException	Creates a socket using a preexisting **InetAddress** object and a port.

You can gain access to the input and output streams associated with a **Socket** by use of the **getInputStream( )** and **getOuptutStream( )** methods, as shown here.
Each can throw an **IOException** if the socket has been invalidated by a loss of connection.
These streams are used exactly like the I/O streams described in Chapter 20 to send and receive data.



InputStream getInputStream( ) throws IOException	Returns the **InputStream** associated with the invoking socket.
OutputStream getOutputStream( ) throws IOException	Returns the **OutputStream** associated with the invoking socket.

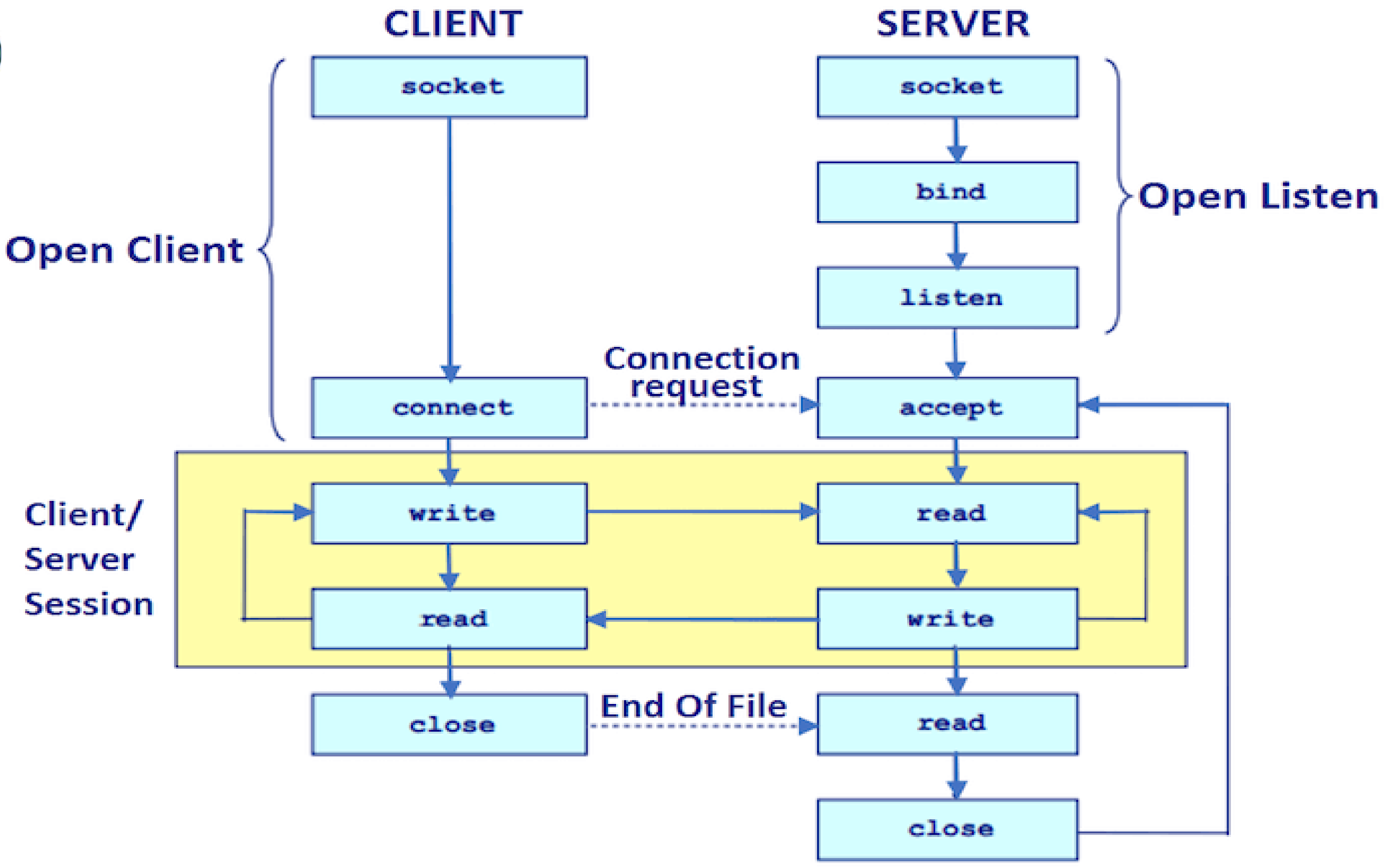
Dr. Smitha Shekar B

NOTE

- Several other methods are available, including **connect( ),** which allows you to specify a
- new connection; **isConnected( ),** which returns true if the socket is connected to a server;
- **isBound( ),** which returns true if the socket is bound to an address; and **isClosed( ),** which
- returns true if the socket is closed. To close a socket, call **close( )**. Closing a socket also
- closes the I/O streams associated with the socket.

The following steps occur when establishing a TCP connection between two computers using sockets

- The server instantiates a ServerSocket object, denoting which port number communication is to occur on.
- The server invokes the accept() method of the ServerSocket class. This method waits until a client connects to the server on the given port.
- After the server is waiting, a client instantiates a Socket object, specifying the server name and the port number to connect to.
- The constructor of the Socket class attempts to connect the client to the specified server and the port number. If communication is established, the client now has a Socket object capable of communicating with the server.
- On the server side, the accept() method returns a reference to a new socket on the server that is connected to the client's socket.
- After the connections are established, communication can occur using I/O streams. Each socket has both an OutputStream and an InputStream. The client's OutputStream is connected to the server's InputStream, and the client's InputStream is connected to the server's OutputStream.
- TCP is a two-way communication protocol, hence data can be sent across both streams at the same time. Following are the useful classes providing complete set of methods to implement sockets.



SOCKET API

Dr. Smitha Shekar B

Socket class

- A socket is simply an endpoint for communications between the machines. The Socket class can be used to create a socket.

Method	Description
1) public InputStream getInputStream()	returns the InputStream attached with this socket.
2) public OutputStream getOutputStream()	returns the OutputStream attached with this socket.
3) public synchronized void close()	closes this socket

Dr. Smitha Shekar B

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) public Socket accept()	returns the socket and establish a connection between server and client.
2) public synchronized void close()	closes the server socket.

Creating Server

- To create the server application, we need to create the instance of ServerSocket class. Here, we are using 6666 port number for the communication between the client and server.
- You may also choose any other port number.
- The accept() method waits for the client. If clients connects with the given port number, it returns an instance of Socket.
- ServerSocket ss=**new** ServerSocket(6666);
- Socket s=ss.accept();//establishes connection and waits for the client

Dr. Smitha Shekar B

MyServer.java

```
import java.io.*;
import java.net.*;
public class MyServer {
public static void main(String[] args){
try{
ServerSocket ss=new ServerSocket(6666);
Socket s=ss.accept();//establishes connection
DataInputStream dis=new DataInputStream(s.getInputStream());
String  str=(String)dis.readUTF();
System.out.println("message= "+str);
ss.close();
}catch(Exception e){System.out.println(e);}
}
}
```

Creating Client

- To create the client application, we need to create the instance of Socket class. Here, we need to pass the IP address or hostname of the Server and a port number. Here, we are using "localhost" because our server is running on same system.
- Socket s=**new** Socket("localhost",6666);

MyClient.java

```
import java.io.*;
import java.net.*;
public class MyClient {
public static void main(String[] args) {
try{
Socket s=new Socket("localhost",6666);
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
dout.writeUTF("Hello Server");
dout.flush();
dout.close();
s.close();
}catch(Exception e){System.out.println(e);}
}
}
```

Lab Program –For Output

- Create a file called testfile.txt in the folder where Client.java and Server.java is located. Add some content.
- Open two terminals
- Navigate to the src folder of your project

First terminal

```
javac Server.java
java Server 4000
```

Second terminal

```
javac Client.java
java localhost 4000
```

```
test.txt
```

Dr. Smitha Shekar B

Java URL

- The **Java URL** class represents an URL. URL is an acronym for Uniform Resource Locator. It points to a resource on the World Wide Web.
- For example: http://www.yahoo.com

- A URL contains many information:
- **Protocol:** In this case, http is the protocol.
- **Server name or IP Address:** In this case, www.yahoo.com is the server name.
- **Port Number:** It is an optional attribute. If port number is not mentioned in the URL, it returns -1.
- **File Name or directory name:** In this case, index.jsp is the file name.

- The URL provides a reasonably intelligible form to uniquely identify or address
- information on the Internet. URLs are ubiquitous; every browser uses them to identify information on the Web.
- Within Java's network class library, the **URL** class provides a simple, concise API to access information across the Internet using URLs.

REMOTE METHOD INVOCATION (RMI)

Introduction

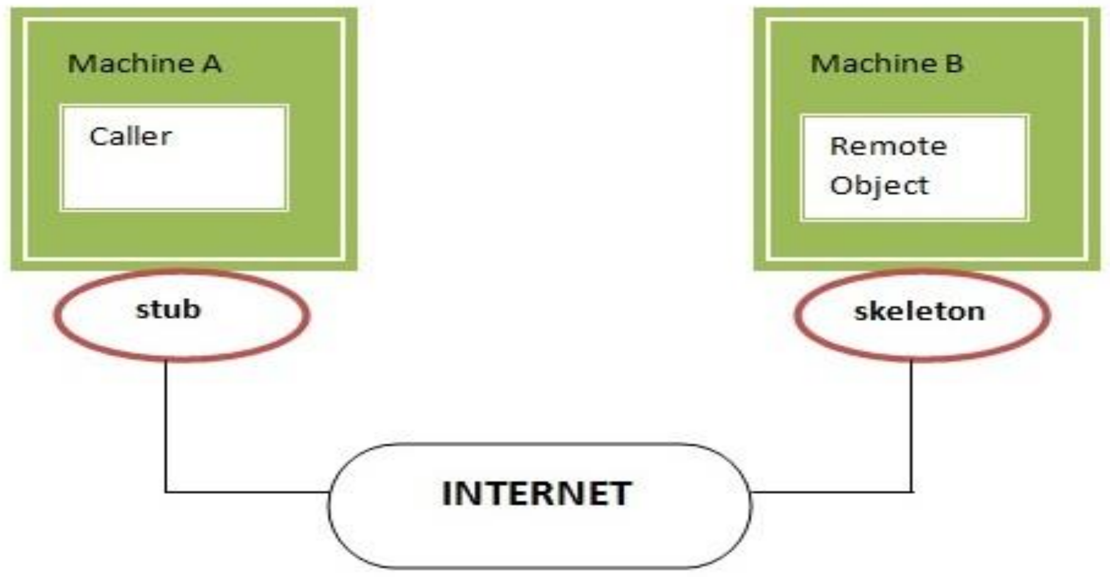
- The **RMI** (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM.

- The RMI provides remote communication between the applications using two objects *stub* and *skeleton*.

Dr. Smitha Shekar B

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.



Dr. Smitha Shekar B

stub and skeleton

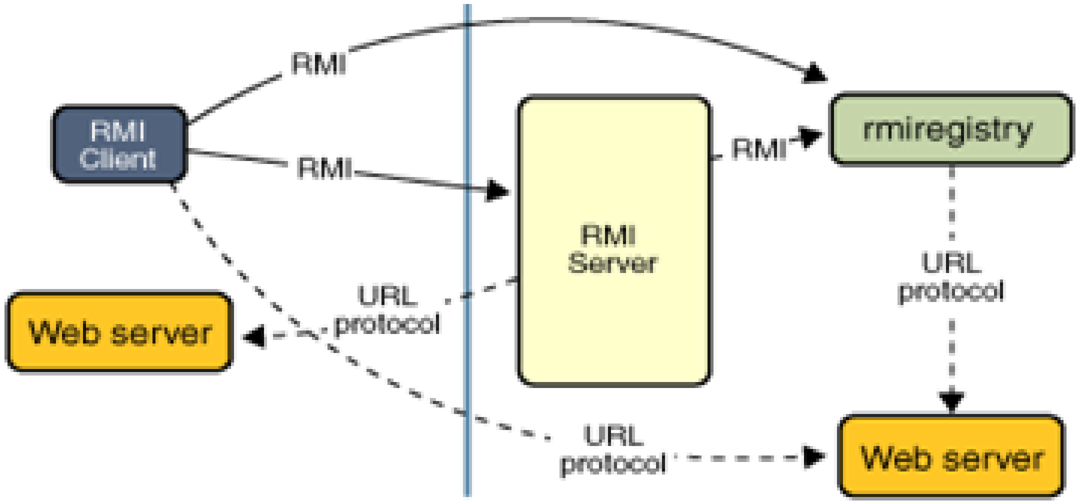
- The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:
- It initiates a connection with remote Virtual Machine (JVM),
- It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
- It waits for the result
- It reads the return value or exception, and
- It finally, returns the value to the caller.

****

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

Steps to write the RMI program

- Create the remote interface
- Provide the implementation of the remote interface
- Compile the implementation class and create the stub and skeleton objects using the rmic tool
- Start the registry service by rmiregistry tool
- Create and start the remote application
- Create and start the client application



Dr. Smitha Shekar B

Defining the Remote Interface

- A remote interface provides the description of all the methods of a particular remote object. The client communicates with this remote interface.
- **To create a remote interface** –
- Create an interface that extends the predefined interface **Remote** which belongs to the package.
- Declare all the business methods that can be invoked by the client in this interface.
- Since there is a chance of network issues during remote calls, an exception named **RemoteException** may occur; throw it.

Developing the Implementation Class (Remote Object)

- We need to implement the remote interface created in the earlier step. (We can write an implementation class separately or we can directly make the server program implement this interface.)
- **To develop an implementation class –**
- Implement the interface created in the previous step.
- Provide implementation to all the abstract methods of the remote interface.

For Output

Open a terminal

Navigate to the src folder of your project

- javac AddServerIntf.java
- javac AddServerImpl.java
- javac AddServer.java
- javac AddClient.java
- rmic AddServerImpl
- start rmiregistry

In another terminal (while previous one is still running)

Navigate to the src folder of your project

- java AddServer

In third terminal (while previous both are still open)

Navigate to the src folder of your project

- java AddClient 2 3

Dr. Smitha Shekar B

Thank You

Dr. Smitha Shekar B