UNIT 4
Networking Basics

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.
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

1. Sharing resources
2. Centralize software management

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 Networking Terminology

In Java Networking, many terminologies are used frequently. These widely used Java Networking Terminologies are given as follows:

1. **IP Address –** An IP address is a **unique address** that **distinguishes** a **device** on the **internet** or a **local network.**
   IP stands for "Internet Protocol."
   It comprises a **set of rules** governing the **format of data sent** via the **internet** or **local network**.
   IP Address is referred to as **a logical address** that can be **modified**.
   It is composed of **octets**. The range of each octet varies from 0 to 255.
- Range of the IP Address – 0.0.0.0 to 255.255.255.255
- For Example – 192.168.0.1

2. **Port Number –** A port number is a method to recognize a particular process connecting internet or other network information when it reaches a server. The port number is used to **identify different applications uniquely**. The port number behaves as a **communication endpoint** among **applications**. The port number is correlated with the IP address for transmission and communication among two applications. There are 65,535 port numbers, but not all are used every day.

Some reserved Ports for Specific Services:

20 – FTP (File Transfer Protocol)
22 – Secure Shell (SSH)
25 – Simple Mail Transfer Protocol (SMTP)
53 – Domain Name System (DNS)
80 – Hypertext Transfer Protocol (HTTP)
110 – Post Office Protocol (POP3)
143 – Internet Message Access Protocol (IMAP)
443– HTTP Secure (HTTPS)

3. Protocol –

A network protocol is an organized set of commands (Rules) that define how data is transmitted between different devices in the same network.
Eg:-TCP, FTP, Telnet, SMTP, POP (Post Office Protocol) etc.

4. MAC address

A MAC address (media access control address) is **a 12-digit hexadecimal number (48 Bit) assigned to each device connected to the network**. It identifies a device to other devices on the same local network.
It is also called physical address. It is Primarily **specified** as a **unique identifier** during **device manufacturing**. It is often **found** on a device's **network interface card (NIC).**

5. 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**.

6. Socket

A socket is an endpoint between two way communications.
Sockets allow communication between two different processes on the same or different machines.
A server is a process that **performs some functions** on request from a client. Most of the application-level protocols like **FTP**, **SMTP**, and **POP3** make **use of sockets** to **establish connection** between **client and server** and **then for exchanging data**.

Client Server:

- A computer,which requests for some service from another computer is called a **client**.
- The one that processes the request is called as **server.**
- A server waits till one of its **clients** makes a **request**.
- Socket allows a single computer to serve different types of information to different clients at the same time.
- This is managed by the port,which is a numbered socket on a particular machine.
- A server process can listen to a port number,although each session is unique.

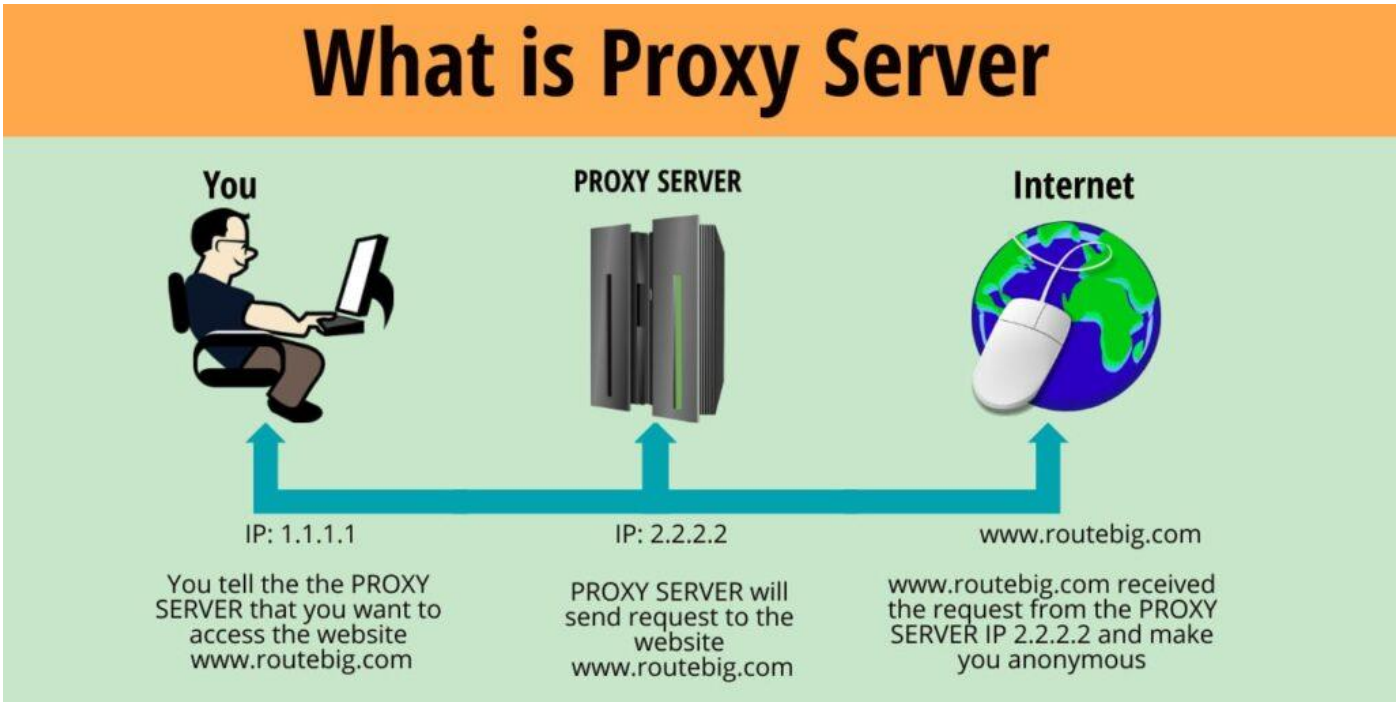
Proxy Server

What is Proxy Server and how it work

Now think, if you are using the internet directly and the website you are accessing is continuously pulling your personal information like your IP address, location, hostname, type and version of your operating system, browser history and chache etc. On the basis of the these information the website will provide you the requested data.

The **proxy server** stands between the you and the online web resource that you are accessing. It acts as a gateway between the content requested by you and the content delivered by the online resource.

It can filter or hide the content you are requesting to the online resources to prevent online resource to access your personal information and make you hide or anonymous.



How the Proxy server works

When you are trying to access the **online web resource** using **your web browser the request goes to the proxy server firstly**. The proxy server **modifies** or **encreypted** the **system** or **personal related infromation** along with the **IP address and forward it to the web resource**.

Now the web resource responded back with the required web content to the proxy server. The proxy server filters out web information and provides to you.
In this way the proxy server works as a filter and you get the data only you want.

Internet Addressing

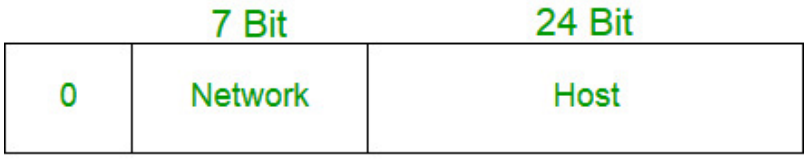
The 32 bit IP address is divided into five sub-classes. These are:

- Class A
- Class B
- Class C
- Class D
- Class E

Each of these classes has a valid range of IP addresses.
Classes **D** and **E** are reserved for **multicast** and **experimental** purposes respectively. The order of bits in the **first octet** determine the **classes of IP address**.
IPv4 address is divided into two parts:

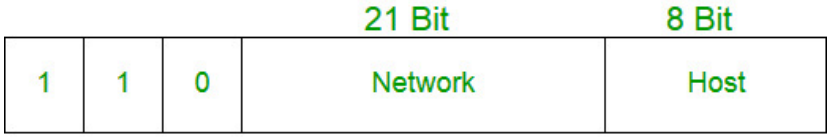
- **Network ID**
- **Host ID**

The class of IP address is used to **determine** the **bits** used for **network ID** and **host ID** and the **number of total networks** and **hosts** possible in that particular class. Each ISP or network administrator assigns IP address to each device that is connected to its network.

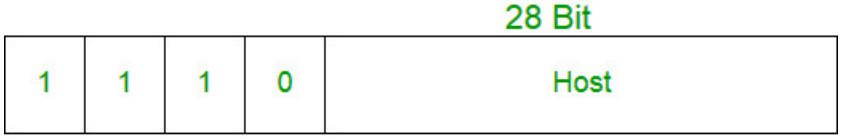


Class A

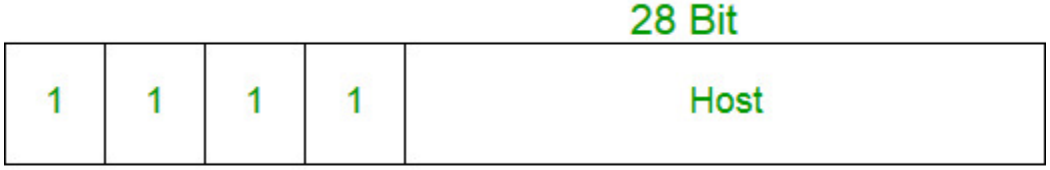
Class B



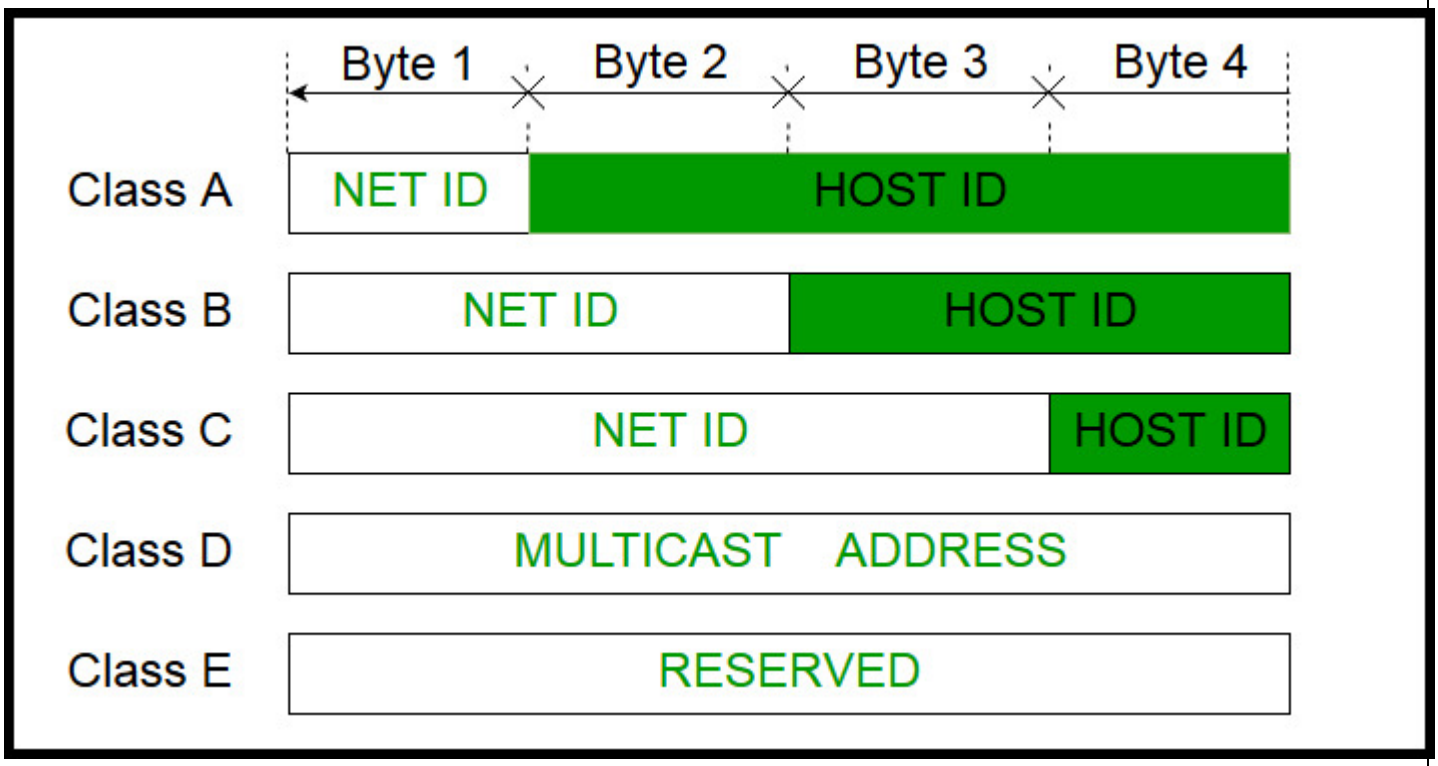
Class C



Class D



Class E



Summry

CLASS	LEADING BITS	NET ID BITS	HOST ID BITS	NO. OF NETWORKS	ADDRESSES PER NETWORK	START ADDRESS	END ADDRESS
CLASS A	0	8	24	2^7 (128)	2^{24} (16,777,216)	0.0.0.0	127.255.255.255
CLASS B	10	16	16	2^{14} (16,384)	2^{16} (65,536)	128.0.0.0	191.255.255.255
CLASS C	110	24	8	2^{21} (2,097,152)	2^8 (256)	192.0.0.0	223.255.255.255
CLASS D	1110	NOT DEFINED	NOT DEFINED	NOT DEFINED	NOT DEFINED	224.0.0.0	239.255.255.255
CLASS E	1111	NOT DEFINED	NOT DEFINED	NOT DEFINED	NOT DEFINED	240.0.0.0	255.255.255.255

Java Networking classes

The **java.net** package of the Java programming language includes various classes that provide an easy-to-use means to access network resources. The classes covered in the **java.net** package are given as follows –

1. **CacheRequest –** The CacheRequest class is used in java whenever there is a need to store resources in ResponseCache.

2. **CookieHandler –** The CookieHandler class is used in Java to implement a callback mechanism for securing up an HTTP state management policy implementation inside the HTTP protocol handler. The HTTP state management mechanism specifies the mechanism of how to make HTTP requests and responses.

3. **CookieManager –** The CookieManager class is used to provide a precise implementation of CookieHandler. This class separates the storage of cookies from the policy surrounding accepting and rejecting cookies. A CookieManager comprises a CookieStore and a CookiePolicy.

4. **DatagramPacket –** The DatagramPacket class is used to provide a facility for the connectionless transfer of messages from one system to another. This class provides tools for the production of datagram packets for connectionless transmission applying the datagram socket class.

5. **InetAddress –** The InetAddress class is used to provide methods to get the IP address of any hostname. An IP address is expressed by a 32-bit or 128-bit unsigned number. InetAddress can handle both IPv4 and IPv6 addresses.

6. **ServerSocket –** The ServerSocket class is used for implementing system-independent implementation of the server-side of a client/server Socket Connection. The constructor for ServerSocket class throws an exception if it can't listen on the specified port. For example – it will throw an exception if the port is already being used.

7. **Socket –** The Socket class is used to create socket objects that help the users in implementing all fundamental socket operations. The users can implement various networking actions such as sending, reading data, and closing connections. Each Socket object built using **java.net.Socket** class has been connected exactly with 1 remote host; for connecting to another host, a user must create a new socket object.

8. **DatagramSocket –** The DatagramSocket class is a network socket that provides a connection-less point for sending and receiving packets. Every packet sent from a datagram socket is individually routed and delivered. It can further be practiced for transmitting and accepting broadcast information. Datagram Sockets is Java's mechanism for providing network communication via UDP instead of TCP.

9. **Proxy –** A proxy is a changeless object and a kind of tool or method or program or system, which serves to preserve the data of its users and computers. It behaves like a wall between computers and internet users. A Proxy Object represents the Proxy settings to be applied with a connection.

10. **URL –** The URL class in Java is the entry point to any available sources on the internet. A Class URL describes a Uniform Resource Locator, which is a signal to a "resource" on the World Wide Web. A source can denote a simple file or directory, or it can indicate a more difficult object, such as a query to a database or a search engine.

11. **URLConnection –** The URLConnection class in Java is an abstract class describing a connection of a resource as defined by a similar URL. The URLConnection class is used for assisting two distinct yet interrelated purposes. Firstly it provides control on interaction with a server(especially an HTTP server) than a URL class. Furthermore, with a URLConnection, a user can verify the header transferred by the server and can react consequently. A user can also configure header fields used in client requests using URLConnection.

Java Networking Interfaces

The **java.net** package of the Java programming language includes various interfaces also that provide an easy-to-use means to access network resources. The interfaces included in the **java.net** package are as follows:

1. **CookiePolicy –** The CookiePolicy interface in the **java.net** package provides the classes for implementing various networking applications. It decides which cookies should be accepted and which should be rejected. In CookiePolicy, there are three pre-defined policy implementations, namely ACCEPT_ALL, ACCEPT_NONE, and ACCEPT_ORIGINAL_SERVER.

2. **CookieStore –** A CookieStore is an interface that describes a storage space for cookies. CookieManager combines the cookies to the CookieStore for each HTTP response and recovers cookies from the CookieStore for each HTTP request.

3. **FileNameMap –** The FileNameMap interface is an uncomplicated interface that implements a tool to outline a file name and a MIME type string. FileNameMap charges a filename map (known as a mimetable) from a data file.

4. **SocketOption –** The SocketOption interface helps the users to control the behavior of sockets. Often, it is essential to develop necessary features in Sockets. SocketOptions allows the user to set various standard options.

5. **SocketImplFactory –** The SocketImplFactory interface defines a factory for SocketImpl instances. It is used by the socket class to create socket implementations that implement various policies.

6. **ProtocolFamily –** This interface represents a family of communication protocols. The ProtocolFamily interface contains a method known as name(), which returns the name of the protocol family.

InetAddress

Inet Address encapsulates both numerical IP address and the domain name for that address. **Inet** address can handle both **IPv4** and **Ipv6** addresses.

There are no public **InetAddress()** constructors. To create an **InetAddress** object, you have to use one of the available factory methods.

- **Factory methods** are merely a **convention** whereby **static methods** in a **class return** an **instance of that class**.
- The Factory Method defines a method, **which should be used** for **creating objects instead** of using a **direct constructor** call **(new operator**).

Factory methods.

Three commonly used Inet Address factory methods are.

a)static InetAddress getLocalHost( )throws UnknownHostException

Returns the InetAddress object that represents the localhost.

b)static InetAddress getByName(String hostName) throws UnknownHostException

Returns the InetAddress for a host name passed to it.

c)static InetAddress[ ] getAllByName(String hostName) throws UnknownHostException

Returns an array of InetAddress that represent all of the addresses that a particular name resolves to.

If these methods are unable to resolve the host name,they throw an UnknownHostException.

- **Instance Methods:-**

The InetAddress class also has several other methods.

- **Boolean equals(Object other)**
  Returns **true** if this object has the same Internet address as *other*.
- **byte[ ] getAddress( )**
  Returns a four-element byte array that represents the object's Internet address in network byte order.
- **String getHostAddress( )**
  Returns a string that represents the host address associated with the **InetAddress** object.
- **String getHostName( )**
  Returns a string that represents the host name associated with the **InetAddress** object
- **int hashCode( )**
  Returns the hash code of the invoking object.

- **boolean isMulticastAddress( )**
  Returns **true** if this Internet address is a multicast address. Otherwise, it returns **false**.
- **String toString( )**
  Returns a string that lists the host name and the IP address for convenience; for example, "starwave.com/192.147.170.6".

Program to use above Methods

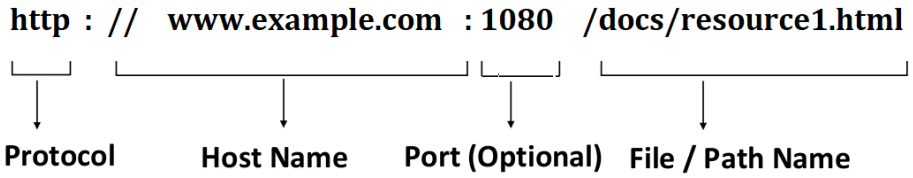
```
import java.net.*;
class InetAddressDemo
{
public static void main(String args[])throws UnknownHostException
{
InetAddress addr=InetAddress.getLocalHost();
System.out.println(addr);
addr=InetAddress.getByName("www.msbte.com");
System.out.println(addr);
System.out.println("Host Name: "+addr.getHostName());
System.out.println("IP Address: "+addr.getHostAddress());
System.out.println("Whether address of www.msbte.com is Multicast :- "+ addr.isMulticastAddress());
InetAddress a[]=InetAddress.getAllByName("www.google.com");
for(int i=0;i<a.length;i++)
{
System.out.println(a[i]);
}
}
}
```

URL

URL known as **Uniform Resource Locator** is simply a **string** of **text** that **identifies** all the **resources** on the **Internet**, telling us the **address** of **the resource**, how to communicate with it, and retrieve something from it.

Class **URL** represents a **Uniform Resource Locator**, a **pointer** to a "**resource**" on the **World Wide Web**. A resource can be something as simple as a **file** or a **directory**, or it can be a **reference** to a more complicated object, such as a **query** to a database or to a search engine.

Format of URL path
In general, a URL can be broken into several parts.
Consider the following example:



The URL above indicates that the protocol to use is http (HyperText Transfer Protocol) and that the information resides on a host machine named www.example.com. The information on that host machine is named /docs/resource1.html

A URL can optionally specify a "port", which is the port number to which the TCP connection is made on the remote host machine. If the port is not specified, the default port for the protocol is used instead. For example, the default port for http is 80.

URL Class

- The **URL class** is a **wrapper** of **standard URL string**. It is a **final class** and is directly inherited from the **Object class.**
- This class provides a simple way of accessing information across the Internet using URLs.
- URL class has following constructors,and each can throw a **MalformedURLException.**

Constructor and Description

URL(String protocol, String host, int port, String file) throws MalformedURLException Creates a URL object from the specified protocol, host, port number, and file. Host can be expressed as a host name or a literal IP address. Specifying a port number of -1 indicates that the URL should use the default port for the protocol. If this is the first URL object being created with the specified protocol, a stream protocol handler object, an instance of class URLStreamHandler, is created for that protocol**.**
URL(**String** spec) Creates a URL object from the String representation.
URL(String protocol, String host, int port, String file, URLStreamHandler handler) Creates a URL object from the specified protocol, host, port number, file, and handler.

URL(**String** protocol, **String** host, **String** file) Creates a URL from the specified protocol name, host name, and file name.
URL(**URL** context, **String** spec) Creates a URL by parsing the given spec within a specified context.
URL(**URL** context, **String** spec, **URLStreamHandler** handler) Creates a URL by parsing the given spec with the specified handler within a specified context.

Methods of URL Class

1. **public String getPath()**

Returns:
the path part of this URL, or an empty string if one does not exist

2. **public String getUserInfo()**

Returns:
the userInfo part of this URL, or null if one does not exist

3. **public String getAuthority()**

Returns:
the authority part of this URL

4. **public int getPort()**

Gets the port number of this URL.
Returns:
the port number, or -1 if the port is not set

5. **public int getDefaultPort()**

Gets the default port number of the protocol associated with this URL. If the URL scheme or the URLStreamHandler for the URL do not define a default port number, then -1 is returned.

6. **public String getProtocol()**

Returns:
the protocol of this URL.

7. **public String getHost()**

Returns:
the host name of this URL.

8. **public String getFile()**

Returns:
the file name of this URL, or an empty string if one does not exist

Example 1:

```
import java.net.*;
public class URLDemo{
public static void main(String[] args){
try
{
URL url=new URL("https://msbte.org.in/file/ALearningManualBAsicScienceM_090220210025.pdf");

System.out.println("Protocol: "+url.getProtocol());
System.out.println("Host Name: "+url.getHost());
System.out.println("Port Number: "+url.getPort());
System.out.println("Default Port Number: "+url.getDefaultPort());
System.out.println("File: "+url.getFile());
}
catch(Exception e){System.out.println(e);}
}
}
```

Output:

```
Protocol: https
Host Name: msbte.org.in
Port Number: -1
Default Port Number: 443
File: /file/ALearningManualBAsicScienceM_090220210025.pdf
```

URLConnection

1. URLConnection is an abstract class. The two subclasses **HttpURLConnection** and **JarURLConnection** makes the **connetion** between the **client Java program** and **URL resource** on the internet.
2. With the help of **URLConnection class**, a user can **read** and **write** to and from any **resource referenced** by an **URL** object.
3. Once a connection is established and the Java program has an URLConnection object, we can use it to read or write or get further information like content length, etc.

Constructor	Description
1) protected URLConnection(URL url)	It constructs a URL connection to the specified URL.

The openConnection() method returns a **java.net.URLConnection**, an abstract class whose subclasses represent the various types of URL connections.

For example –

- If you connect to a URL whose protocol is **HTTP**, the openConnection() method returns an **HttpURLConnection object.**
- If you connect to a **URL** that represents a **JAR file**, the **openConnection**() method returns a **JarURLConnection object, etc.**

URLConnection Class Methods

Method	Description
Object getContent()	It retrieves the contents of the URL connection.
int getContentLength()	It returns the value of the content-length header field.
String getContentType()	It returns the value of the date header field.
long getDate()	It returns the value of the date header field.
long getExpiration()	It returns the value of the expires header fileD.
long getLastModified()	It returns the value of the last-modified header field.
URL getURL()	It returns the value of the URLConnection's URL field.
InputStream getInputStream()	It returns an input stream that reads from the open condition.
OutputStream getOutputStream()	It returns an output stream that writes to the connection.

Program to display Date, Content Type, Content Length and Contents of Web Page.

```
import java.io.*;
import java.net.*;
import java.util.Date;
public class URLDemo
{
public static void main(String[] args)
{
try
{
URL url=new URL("https://msbte.org.in/");
URLConnection con=url.openConnection();
System.out.println("Date="+new Date(con.getDate()));
System.out.println("Type of contents="+(con.getContentType()));
System.out.println("length of contents="+(con.getContentLength()));
System.out.println("Expiration Date="+new Date(con.getExpiration()));
System.out.println("Last Modified Date="+new Date(con.getLastModified()));
/* This Block prints the contents of Web Page
InputStream stream=con.getInputStream();
```

```
int i;
while((i=stream.read())!=-1)
{
System.out.print((char)i);
}
*/
}
catch(Exception e)
{
}
}
}
```

```
Date=Tue Jan 17 13:09:25 IST 2023
Type of contents=text/html
length of contents=1121949
Expiration Date=Thu Jan 01 05:30:00 IST 1970
Last Modified Date=Tue Jan 17 11:12:56 IST 2023
```

TCP/IP Sockets

TCP/IP sockets are used to implement reliable two-way, persistent, point-to-point streaming connections between hosts on the Internet..

Client sends a message to the server, server reads the message and prints it.
A socket can be used to connect java's Input/Output system to other programs,that may reside either on the local machine or any other machine or any other machine on the Internet.
In java,TCP sockets are classified as:

- TCP Server Socket
- TCP Client Socket

The **java.net** package provides two classes that allow you to create the two kinds of sockets.
The classes are **Socket** and **ServerSocket** respectively.
The **Socket class**, basically **used** to create **client sockets**, is designed **to connect to server sockets** and initiate protocol exchanges.
The **ServerSocket class**,basically used to **create server sockets**, is designed to be a **listener**, which waits **for clients** to **connect** before **doing anything**.

1. Socket class

The **Socket** class encapsulates a **two-way communication mechanism** that is **connected** at **both ends** to an **IP address** and **port number**.
It implements a **stream-based**, **connection-oriented** data communication.
When you create a **Socket** object, it implicitly establishes the connection between the client and the server.

Constructors to create client sockets:-

1. **Socket(String hostName, int port) throws UnknownHostException, IOException:** Creates a socket connected to the specified host and port.
2. **Socket(InetAddress ipAddress, int port) throws IOException:** Creates a socket using a pre-existing InetAddress object and a port.

These constructors are used to create a socket by specifying hostname and port or by using an existing InetAddress object and a port.
After you have created a socket, you can get the address and port details associated with the socket by the use of the following methods:

1. **InetAddress getInetAddress( ):** It returns the InetAddress associated with the Socket object. It returns null if the socket is not connected.
2. **int getPort( ):** It returns the **remote port** to which the invoking Socket object is connected. It returns 0 if the socket is not connected.
3. **int getLocalPort( ):** Returns the **local port** to which the invoking Socket object is bound. It returns **-1** if the socket is not bound.

4. **InputStream getInputStream( ) throws IOException:** Returns the InputStream associated with the invoking socket.
5. **OutputStream getOutputStream( ) throws IOException:** Returns the OutputStream associated with the invoking socket.
6. **connect( ):** Allows you to specify a new connection.
7. **isConnected( ):** Returns true if the socket is connected to a server
8. **isBound( ):** Returns true if the socket is bound to an address
9. **isClosed( ):** Returns true if the socket is closed.

2. ServerSocket class:-

The **ServerSocket** class is used to **create servers** that **listen** for **either local or remote client programs** to connect to them on standard ports.
The **ServerSocket** object can create a **Socket object** to **receive** and **transmit data** through the **socket**.

Constructor:-

ServerSocket(int port) throws IOException
This constructor is used to create a server socket on the specified port. It throws an IOException.

Methods:

1. **Socket accept()**
   This method listens for a connection request and accepts the connection. When the connection is accepted, a Socket object is returned, that can be used to send and receive data through the connection.
2. **InetAddress getInetAddress()**
   These methods returns an IP address used by the invoking ServerSocket object.
3. **Int getLocalPort()**
   These methods returns port number used by the invoking ServerSocket object.
4. **void close()**
   Last method closes the socket and releases any system resources associated with it.

Example of Java Socket Programming

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
```

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);
```

Let's see a simple of Java socket programming where client sends a text and server receives and prints it.

// 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=dis.readLine();
System.out.println("message= "+str);
ss.close();
}
catch(Exception e)
{
System.out.println(e);
}
}
}
```

```
// 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.writeBytes("Hello Server");
dout.flush();
dout.close();
s.close();
}
catch(Exception e)
{
        System.out.println(e);
}
}
}
```

To execute this program open two command prompts and execute each program at each command prompt.
After running the client application, a message will be displayed on the server console.

Example of Java Socket Programming (Read-Write both side)

In this example, client will write first to the server then server will receive and print the text. Then server will write to the client and client will receive and print the text. The step goes on.

MyServer.java

```
import java.net.*;
import java.io.*;
class MyServer{
public static void main(String args[])throws Exception
{
ServerSocket ss=new ServerSocket(3333);
Socket s=ss.accept();
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
```

```
String str="",str2="";
while(!str.equals("stop"))
{
str=din.readUTF();
System.out.println("client says: "+str);
str2=br.readLine();
dout.writeUTF(str2);
dout.flush();
}
din.close();
s.close();
ss.close();
}}
```

//MyClient.java

```
import java.net.*;
import java.io.*;
class MyClient{
public static void main(String args[])throws Exception
{
Socket s=new Socket("localhost",3333);
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

String str="",str2="";
while(!str.equals("stop"))
{
str=br.readLine();     // Write something to stream
dout.writeUTF(str);
dout.flush();

str2=din.readUTF();                    //Read from server
System.out.println("Server says: "+str2);
}

dout.close();
s.close();
}
}
```

UDP

UDP is the simplest transport layer communication protocol.
It is considered an **unreliable** protocol.
The UDP is a **connectionless** protocol as it **does not create a virtual path** to transfer the data.
Ordered delivery of data is not guaranteed:-
In the case of UDP, the datagrams are sent in some order will be received in the same order is not guaranteed as the datagrams are not numbered.
The UDP protocol uses different **port numbers** so that the data can be sent to the correct destination.
UDP enables **faster transmission** as it is a **connectionless** protocol,
UDP provides **no acknowledgment mechanism**, which means that the receiver does not send the acknowledgment for the received packet, and the sender also does not wait for the acknowledgment for the packet that it has sent.

Datagram

Datagrams are collection of information sent from one device to another device via the established network. When the datagram is sent to the targeted device, there is no assurance that it will reach to the target device safely and completely. It may get damaged or lost in between. Likewise, the receiving device also never know if the datagram received is damaged or not.
The UDP protocol is used to implement the datagrams in Java.

Java DatagramSocket and DatagramPacket

Java DatagramSocket and DatagramPacket classes are used for connection-less socket programming using the UDP instead of TCP.

1. Java DatagramSocket class

Java **DatagramSocket** class represents a **connection-less socket** for **sending and receiving datagram packets**. It is a mechanism used for transmitting datagram packets over network.`
A datagram is basically an information but there is no guarantee of its content, arrival or arrival time.

➢ Constructors of DatagramSocket class

1. **DatagramSocket() throws SocketEeption**: it creates a datagram socket and binds it with the available Port Number on the localhost machine.
2. **DatagramSocket(int port) throws SocketEeption**: it creates a datagram socket and binds it with the given Port Number.
3. **DatagramSocket(int port, InetAddress address) throws SocketEeption**: it creates a datagram socket and binds it with the specified port number and host address.

➢ Methods of DatagramSocket

1. **void send(DatagramPacket packet)throws IOException**
   This method sends the datagram's data to the previously defined host and port by taking a DatagramPacket object as a parameter.

2. **void receive(DatagramPacket packet)throws IOException**
   This method receives a datagram by taking a DatagramPacket object as a parameter.
   To extract data from the received datagram use the **getData()** method defined in the **DatagramPacket** class.

3. **void close() :**This method closes the opened datagram socket.

2. Java DatagramPacket Class

Java DatagramPacket is a message that can be sent or received. It is a data container. If you send multiple packet, it may arrive in any order. Additionally, packet delivery is not guaranteed.

➢ Constructors of DatagramPacket class

1. **DatagramPacket(byte[] barr, int length):** it creates a datagram packet. This constructor is used to receive the packets.
2. **DatagramPacket(byte[] barr, int length, InetAddress address, int port)**: it creates a datagram packet. This constructor is used to send the packets.

➢ Methods of DatagramPacket class

Method	Description
1) InetAddress getAddress()	It returns the IP address of the machine to which the datagram is being sent or from which the datagram was received.
2) byte[] getData()	It returns the data contained in Datagram. This is stored in array of bytes
3) int getLength()	It returns the length of the data to be sent or the length of the data received.
4) int getPort()	It returns the port number on the remote host to which the datagram is being sent or from which the datagram was received.
5) SocketAddress getSocketAddress()	It gets the SocketAddress (IP address + port number) of the remote host that the packet is being sent to or is coming from.
6) void setAddress (InetAddress iaddr)	It sets the IP address of the machine to which the datagram is being sent.

Example of Sending DatagramPacket by DatagramSocket

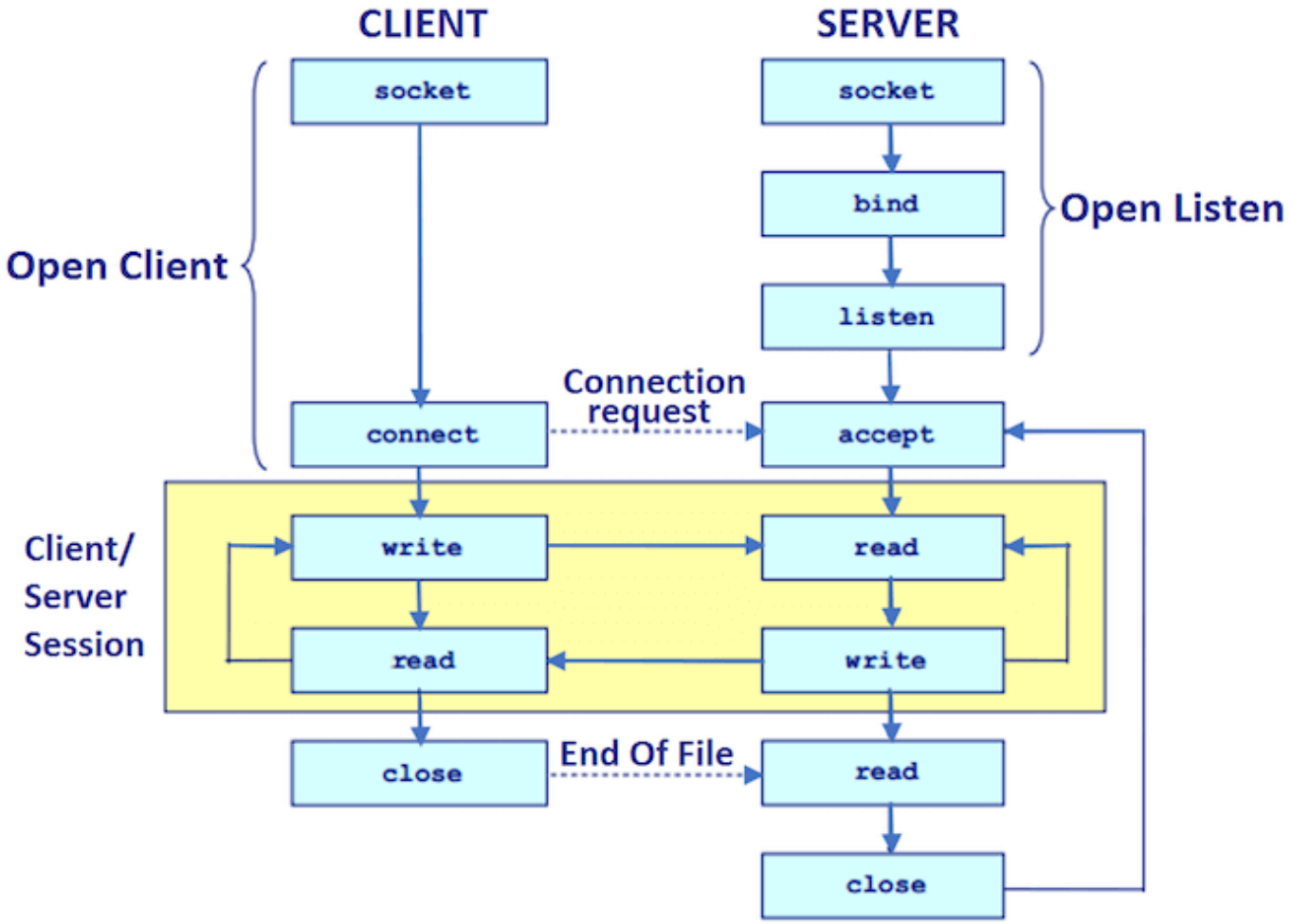
// Dsender.java

```
import java.net.*;
public class Dsender
{
  public static void main(String[] args) throws Exception
{
    DatagramSocket ds = new DatagramSocket();
    String str = "Welcome java";
    InetAddress ip = InetAddress.getByName("127.0.0.1");
    DatagramPacket dp = new DatagramPacket(str.getBytes(), str.length(), ip, 3000);
    ds.send(dp);
    ds.close();
  }
}
```

Example of Receiving DatagramPacket by DatagramSocket

//DReceiver.java

```
import java.net.*;
public class Dreceiver
{
  public static void main(String[] args) throws Exception
{
    DatagramSocket ds = new DatagramSocket(3000);
    byte[] buf = new byte[1024];
    DatagramPacket dp = new DatagramPacket(buf, 1024);
    ds.receive(dp);
    String str = new String(dp.getData(), 0, dp.getLength());
    System.out.println(str);
    ds.close();
  }
}
```


CLIENT
SERVER
socket
socket
bind
listen
Open Client
Open Listen
Connection request
connect
accept
Client/ Server Session
write
read
read
write
close
End Of File
read
close

SOCKET API