Unicode Transformation Format - UTF → 8/16 bit

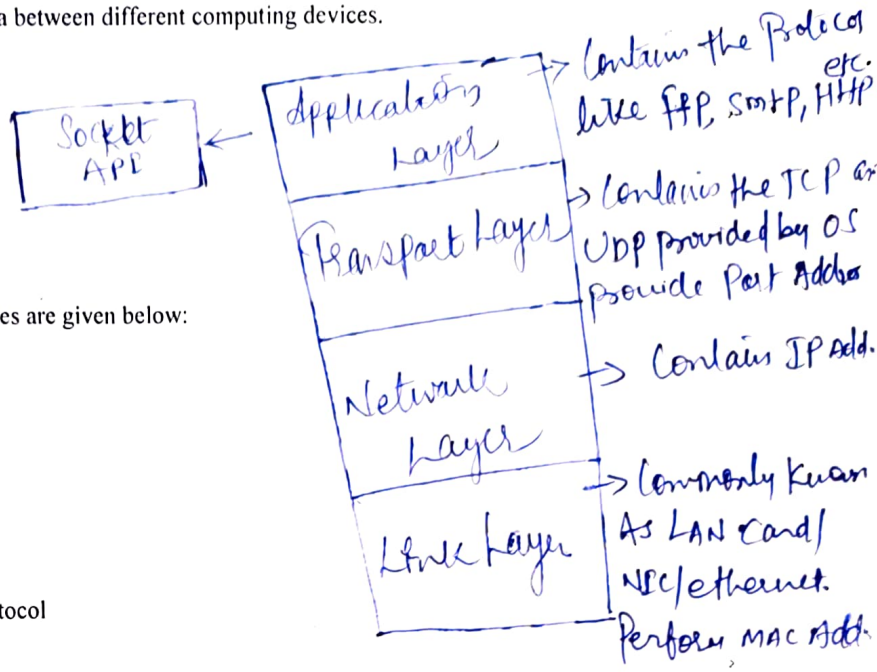
Java Networking AND Core-Java-II Book (Gay. S. Horstmann)

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



Java Networking Terminology

The widely used java networking terminologies are given below:

1. IP Address
2. Protocol
3. Port Number
4. MAC Address
5. Connection-oriented and connection-less protocol
6. Socket

1) IP Address

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.

2) Protocol

A protocol is a set of rules basically that is followed for communication. For example:

- TCP
- FTP
- Telnet
- SMTP
- POP etc.

3) 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.

4) 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.

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.

- **Networking package is java.net**
  - Stream-based communications
    - Applications view networking as streams of data
    - Connection-based protocol
    - Uses TCP (Transmission Control Protocol
  - Packet-based communications
    - Individual packets transmitted
    - Connectionless service
    - Uses UDP (User Datagram Protocol)
- **Client-server relationship**
  - Client requests some action be performed
  - Server performs the action and responds to client
  - Request-response model
    - Common implementation: Web browsers and Web servers

- **Java Socket Programming**

  Java Socket programming is used for communication between the applications running on different JRE.

  Java Socket programming can be connection-oriented or connection-less.

  Socket and ServerSocket classes are used for connection-oriented socket programming and Datagram Socket and Datagram Packet classes are used for connection-less socket programming.

  The client in socket programming must know two information:

1. IP Address of Server, and
2. Port number.

Socket class

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

Important methods

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

ServerSocket class

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

Important methods

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.

Example of Java Socket Programming

Let's see a simple of java socket programming in which client sends a text and server receives it.

File: MyServer.java

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

```
13. }
14. }
```

File: MyClient.java

```
1. import java.io.*;
2. import java.net.*;
3. public class MyClient {
4. public static void main(String[] args) {
5. try{
6. Socket s=new Socket("localhost",6666);
7. DataOutputStream dout=new DataOutputStream(s.getOutputStream());
8. dout.writeUTF("Hello Server");
9. dout.flush();
10. dout.close();
11. s.close();
12. }catch(Exception e){System.out.println(e);}
13. }
14. }
```

To execute this program open two command prompts and execute each program at each command prompt as displayed in the below figure.

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.

File: MyServer.java

```
1. import java.net.*;
2. import java.io.*;
3. class MyServer{
4. public static void main(String args[])throws Exception{
5. ServerSocket ss=new ServerSocket(3333);
6. Socket s=ss.accept();
7. DataInputStream din=new DataInputStream(s.getInputStream());
8. DataOutputStream dout=new DataOutputStream(s.getOutputStream());
```

```
9.  BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
10.
11. String str="",str2="";
12. while(!str.equals("stop")){
13. str=din.readUTF();
14. System.out.println("client says: "+str);
15. str2=br.readLine();
16. dout.writeUTF(str2);
17. dout.flush();
18. }
19. din.close();
20. s.close();
21. ss.close();
22. }}
```

File: MyClient.java

```
1.  import java.net.*;
2.  import java.io.*;
3.  class MyClient{
4.  public static void main(String args[])throws Exception{
5.  Socket s=new Socket("localhost",3333);
6.  DataInputStream din=new DataInputStream(s.getInputStream());
7.  DataOutputStream dout=new DataOutputStream(s.getOutputStream());
8.  BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
9.
10. String str="",str2="";
11. while(!str.equals("stop")){
12. str=br.readLine();
13. dout.writeUTF(str);
14. dout.flush();
15. str2=din.readUTF();
16. System.out.println("Server says: "+str2);
17. }
```

```
18.
19. dout.close();
20. s.close();
21. }}
```

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.google.com/

A URL contains many information:

1. **Protocol:** In this case, http or https is the protocol.
2. **Server name or IP Address:** In this case, www.example.com is the server name.
3. **Port Number:** It is an optional attribute. If we write http//www.example.com:80/nitinsir/, 80 is the port number. If port number is not mentioned in the URL, it returns -1.
4. **File Name or directory name:** In this case, index.jsp is the file name.

Commonly used methods of Java URL class

The java.net.URL class provides many methods. The important methods of URL class are given below.

Method	Description
public String getProtocol()	it returns the protocol of the URL.
public String getHost()	it returns the host name of the URL.
public String getPort()	it returns the Port Number of the URL.
public String getFile()	it returns the file name of the URL.
public URLConnection openConnection()	it returns the instance of URLConnection i.e., associated with this URL.

Example of Java URL class

```
1. //URLDemo.java
2. import java.io.*;
3. Import java.net.*;
4. public class URLDemo{
5. public static void main(String[] args){
6. try{
```

```
7.  URL url=new URL("http://www.google.com/ ");
8.
9.  System.out.println("Protocol: "+url.getProtocol());
10. System.out.println("Host Name: "+url.getHost());
11. System.out.println("Port Number: "+url.getPort());
12. System.out.println("File Name: "+url.getFile());
13.
14. }catch(Exception e){System.out.println(e);}
15. }
16. }
```