1. Introduction to Socket Programming

- 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.
  - Java Networking Terminology
    - 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.
    - Protocol: A protocol is a set of rules basically that is followed for communication. For example: TCP, FTP, Telnet, etc.
    - 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.
    - 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 address. For example, an ethernet card may have a MAC address of 00:0d:83::b1:c0:8e.
    - 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: A socket is an endpoint between two way communications. Visit next page for Java socket programming.
  - java.net package
    - The java.net package can be divided into two sections:
      - A Low-Level API: It deals with the abstractions of addresses i.e. networking identifiers, Sockets i.e. bidirectional data communication mechanism and Interfaces i.e. network interfaces.
      - A High Level API: It deals with the abstraction of URIs i.e. Universal Resource Identifier, URLs i.e. Universal Resource Locator, and Connections i.e. connections to the resource pointed by URLs.
    - The java.net package provides many classes to deal with networking applications in Java. A list of these classes is given below:
      Ex. Socket, ServerSocket, Proxy, URL,…etc.
    - The java.net package provides many interfaces: SocketOptions, URLStreamHandlerFactory, CookieStore, etc.

- 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 DatagramSocket and DatagramPacket classes are used for connection-less socket programming.

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

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

Ex.

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.

```
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.

```
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);}

}

}
```

4. URL class
   - 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.
   - A URL contains many information: http://adit.ac.in/cpfaculties.jsp
     - Protocol: In this case, http is the protocol.
     - Server name or IP Address: like adit.ac.in.
     - Port Number: It is an optional attribute.
     - File Name or directory name: Here, cpfaculties.jsp

     Ex.
     ```
     import java.net.*;
     public class URLDemo{
     public static void main(String[] args){
     try{
     URL url=new URL("http://adit.ac.in/cpfaculties.jsp");

     System.out.println("Protocol: "+url.getProtocol());
     System.out.println("Host Name: "+url.getHost());
     System.out.println("Port Number: "+url.getPort());
     System.out.println("File Name: "+url.getFile());
     ```

```
}catch(Exception e){System.out.println(e);}
}
}
```

5. InetAddress class
    - Java InetAddress class represents an IP address. The java.net.InetAddress class provides methods to get the IP of any host name for example www.google.com, www.facebook.com, etc.
    - An IP address is represented by 32-bit or 128-bit unsigned number. An instance of InetAddress represents the IP address with its corresponding host name. There are two types of addresses: Unicast and Multicast. The Unicast is an identifier for a single interface whereas Multicast is an identifier for a set of interfaces.
      Ex.
      ```
      import java.io.*;
      import java.net.*;
      public class InetDemo{
      public static void main(String[] args){
      try{
      InetAddress ip=InetAddress.getByName("www.adit.ac.in");

      System.out.println("Host Name: "+ip.getHostName());
      System.out.println("IP Address: "+ip.getHostAddress());
      }catch(Exception e){System.out.println(e);}
      }
      }
      ```
6. DatagramSocket and DatagramPacket
    - Java DatagramSocket and DatagramPacket classes are used for connection-less socket programming using the UDP instead of TCP.
    - 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 class represents a connection-less socket for sending and receiving datagram packets. It is a mechanism used for transmitting datagram packets over network.
    - 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.

Ex. Sending DatagramPacket by DatagramSocket

```
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();
  }
}
```

Ex. Receiving DatagramPacket by DatagramSocket

```
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();
  }
}
```