UNIT II JAVA NETWORKING FUNDAMENTALS

- Overview of Java Networking
  - TCP
  - UDP
  - InetAddress and Ports
- Socket Programming
- Working with URLs
- Internet Protocols simulation
  - HTTP
  - SMTP
  - POP
  - FTP
- Remote Method Invocation
- Multithreading Concepts.

Presented by,
B.Vijayalakshmi
Computer Centre
MIT Campus
Anna University

Socket

- Normally, **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.
- **<u>On the client-side:</u>**
  - 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 tries to make contact with the 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.
  - This is usually assigned by the system.

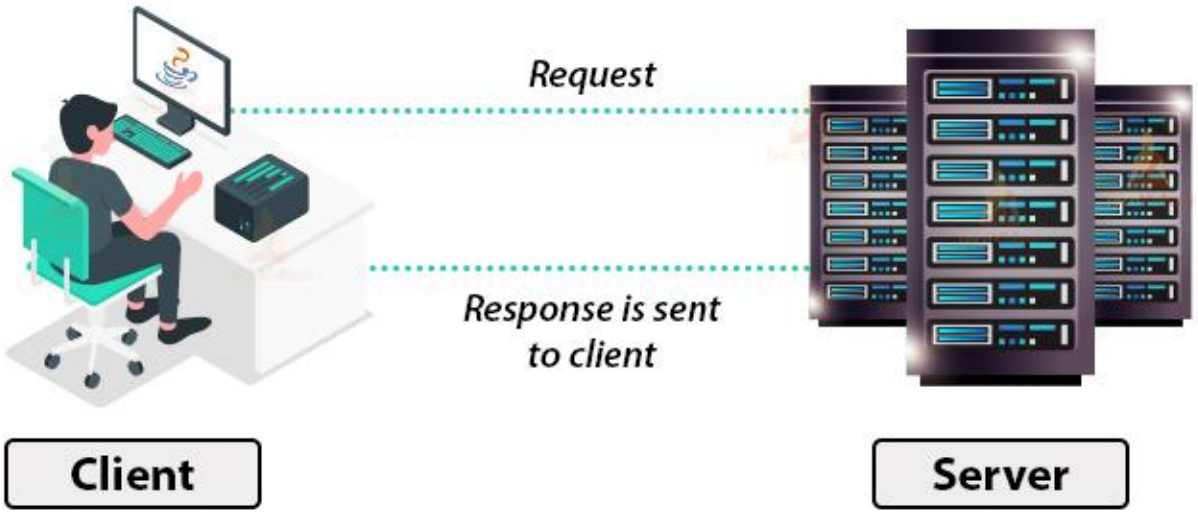
Socket Cont'd

- 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.

Socket programming

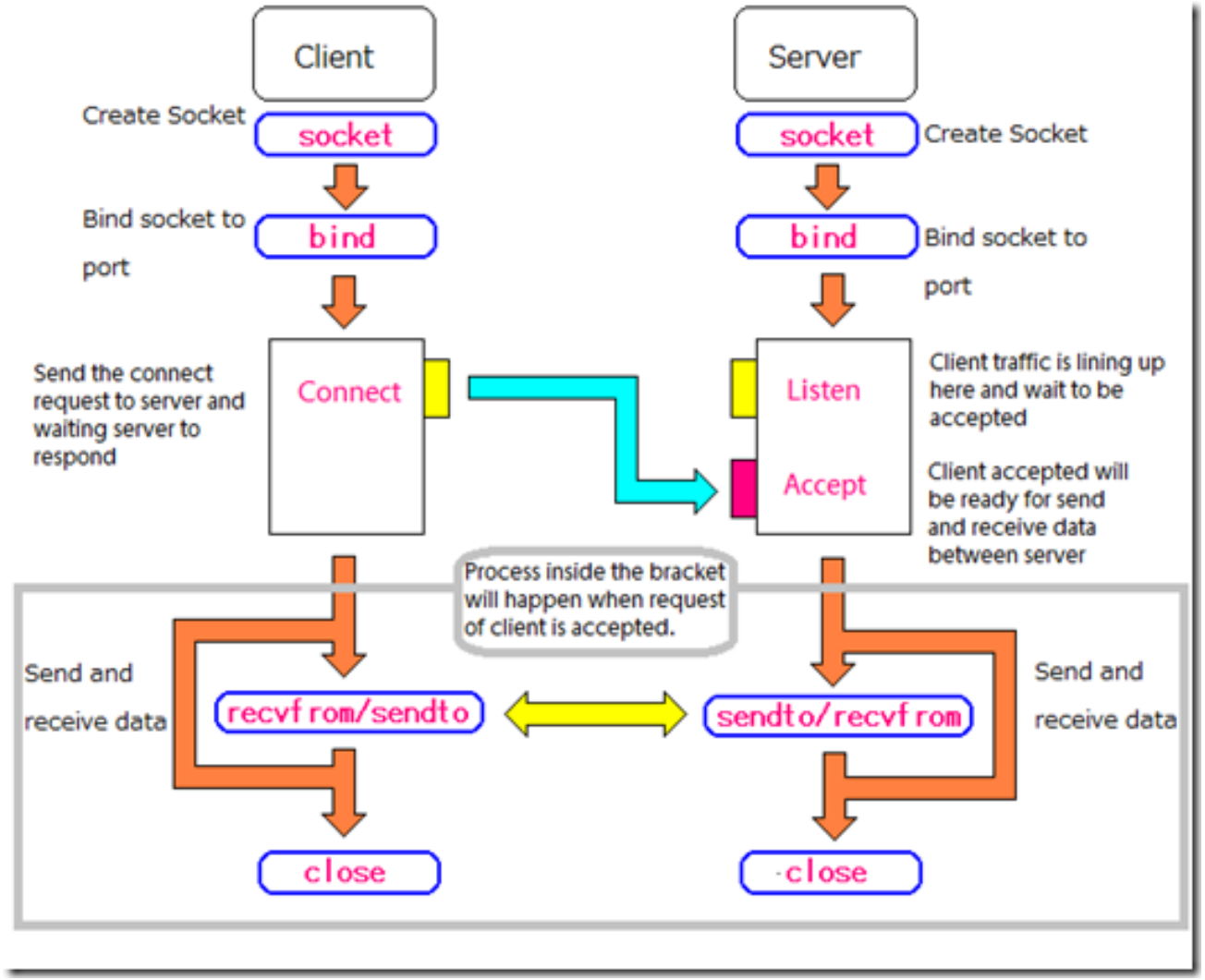


- Socket programming is a **way of connecting two nodes on a network to communicate with each other**.
- One socket(node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection.
- Server forms the listener socket while client reaches out to the server.
- For example, tomcat server running on port 8080 waits for client requests and once it gets any client request, it responds to them.

Java Socket Programming

- **Java Socket programming is used for communication between the applications running on different JRE**.
- Java Socket programming can be,
  - connection-oriented (TCP) or
  - connection-less (UDP)
- The java.net package provides support for the two common network protocols TCP and UDP
- TCP is the protocol of choice for HTTP, FTP, SMTP, POP3 for its guaranteed delivery.
- Protocols like NTP (Network Time Protocol),DNS (Domain Name Service),BOOTP, DHCP,NNP (Network News Protocol),TFTP, RIP, OSPF uses UDP.
- **Socket** and **ServerSocket** classes are used for connection-oriented socket programming
- **DatagramSocket** and **DatagramPacket** classes are used for connection-less socket programming.

Overview of Socket Programming



TCP/IP SOCKETS

- These sockets are used to implement reliable, bidirectional, persistent, point-to-point stream based connections between hosts on the internet.
- There are 2 types of sockets available,
  - ServerSocket
  - Socket

ServerSocket

- **This class is used by server applications** to obtain a port and listen for client requests
- **constructors:**
  - public ServerSocket(int port) throws IOException
  - public ServerSocket(int port, int backlog) throws IOException
  - public ServerSocket(int port, int backlog, InetAddress address) throws IOException
  - public ServerSocket() throws IOException

Methods of ServerSocket

public int getLocalPort() →Returns the port that the server socket is listening on.

public Socket accept() throws IOException

→Waits for an incoming client.

→This method blocks until either a client connects to the server on the specified port or the socket times out, assuming that the time-out value has been set using the setSoTimeout() method. Otherwise, this method blocks indefinitely.

public void setSoTimeout(int timeout)

→Sets the time-out value for how long the server socket waits for a client during the accept().

public void bind(SocketAddress host, int backlog)

→Binds the socket to the specified server and port in the SocketAddress object.

→Use this method if you have instantiated the ServerSocket using the no-argument constructor.

Socket Class

- This class **represents the socket that both the client and the server use** to communicate with each other.
- The client obtains a Socket object by instantiating one whereas the server obtains a Socket object from the return value of the accept() method.

Socket Class Constructors

- **public Socket(String host, int port) throws UnknownHostException, IOException**

  →This method attempts to connect to the specified server at the specified port. If this constructor does not throw an exception, the connection is successful and the client is connected to the server.
- **public Socket(InetAddress host, int port) throws IOException**

  →This method is identical to the previous constructor, except that the host is denoted by an InetAddress object.
- **public Socket(String host, int port, InetAddress localAddress, int localPort) throws IOException.**

  →Connects to the specified host and port, creating a socket on the local host at the specified address and port.
- **public Socket(InetAddress host, int port, InetAddress localAddress, int localPort) throws IOException.**

  →This method is identical to the previous constructor, except that the host is denoted by an InetAddress object instead of a String.
- **public Socket()**

  →Creates an unconnected socket. Use the connect() method to connect this socket to a server.

Socket Class Methods

- **public void connect(SocketAddress host, int timeout) throws IOException**

  →This method connects the socket to the specified host. This method is needed only when you instantiate the Socket using the no-argument constructor.
- **public InetAddress getInetAddress()**

  →This method returns the address of the other computer that this socket is connected to.
- **public int getPort()**

  →Returns the port the socket is bound to on the remote machine.
- **public int getLocalPort()**

  →Returns the port the socket is bound to on the local machine.

Socket Class Methods Cont'd

- **public SocketAddress getRemoteSocketAddress()**

  Returns the address of the remote socket.
- **public InputStream getInputStream() throws IOException**

  Returns the input stream of the socket.

  The input stream is connected to the output stream of the remote socket.
- **public OutputStream getOutputStream() throws IOException**

  Returns the output stream of the socket.

  The output stream is connected to the input stream of the remote socket.
- **public void close() throws IOException**

  Closes the socket, which makes this Socket object no longer capable of connecting again to any server.

Steps occur when establishing a TCP connection between two computers

1. The server instantiates a ServerSocket object, denoting which port number communication is to occur on.
2. The server invokes the accept() method of the ServerSocket class [This method waits until a client connects to the server on the given port]
3. After the server is waiting, a client instantiates a Socket object, specifying the server name and the port number to connect to.
4. 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.
5. 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.

Simple TCP/IP Communication (Server send message to client)

- Server
- Client

TCP Server

```
import java.io.*;
import java.net.*;
import java.util.Scanner;
public class TCPServer
{
        public static void main(String[] args)
        {
                try
                {
                        System.out.println("Server binding to a port...");
                        ServerSocket ss=new ServerSocket(5555);

                        System.out.println("Server waiting for client");
                        Socket s=ss.accept();//establishes connection
                        InetAddress Add=s.getInetAddress();
                        System.out.println("IP address:"+Add);
                        System.out.println("Remote port:"+s.getPort());
                        System.out.println("Local port:"+s.getLocalPort());
```

```
 System.out.println("Received request from client");
 Scanner sc=new Scanner(System.in);
 System.out.println("Enter text...");
 String str=sc.nextLine();
 DataOutputStream dout=new DataOutputStream(s.getOutputStream());
 dout.writeBytes(str);
 dout.close();
 s.close();
 ss.close();
 }catch(Exception e){System.out.println(e);}
 }
}
```

TCP Client

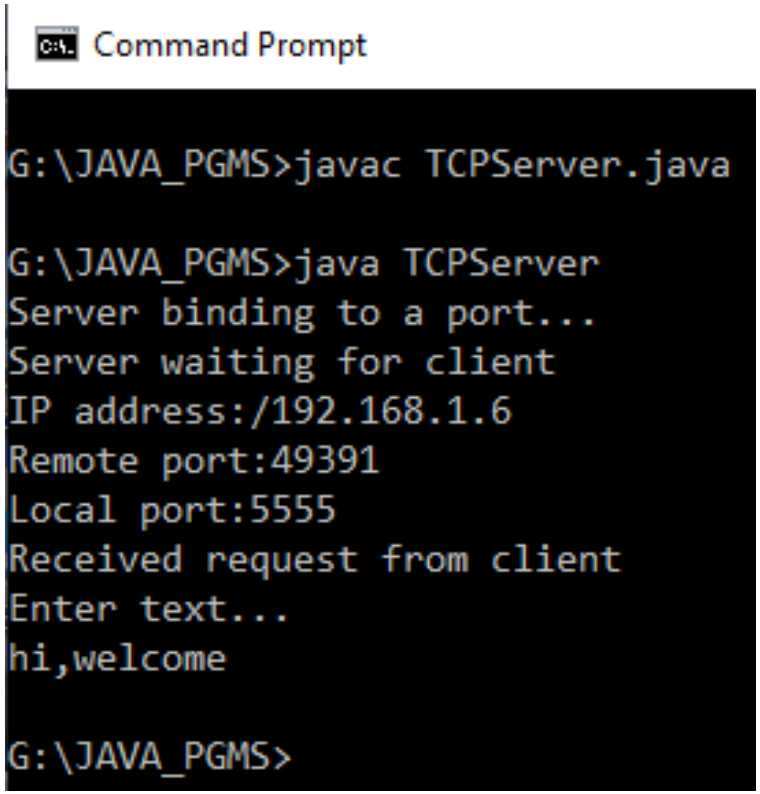
```
import java.net.*;
import java.io.*;
class TCPClient
{
        public static void main(String args[])throws Exception
        {
                try
                {
                        System.out.println("Sending request to server...");
                        //Socket s=new Socket("localhost",5555);
                        Socket s=new Socket("192.168.1.5",5555);
                        InetAddress Add=s.getInetAddress();

                        System.out.println("IP address:"+Add);
                        System.out.println("Remote port:"+s.getPort());
                        System.out.println("Local port:"+s.getLocalPort());
```

```
   System.out.println("waiting for server message...");
   BufferedInputStream bin=new BufferedInputStream(s.getInputStream());
   int i;
   System.out.println("Server says: ");
   while((i=bin.read())!=-1)
   {
            System.out.print((char)i);
   }
   s.close();
  }catch(Exception e){System.out.println(e);}
 }
}
```

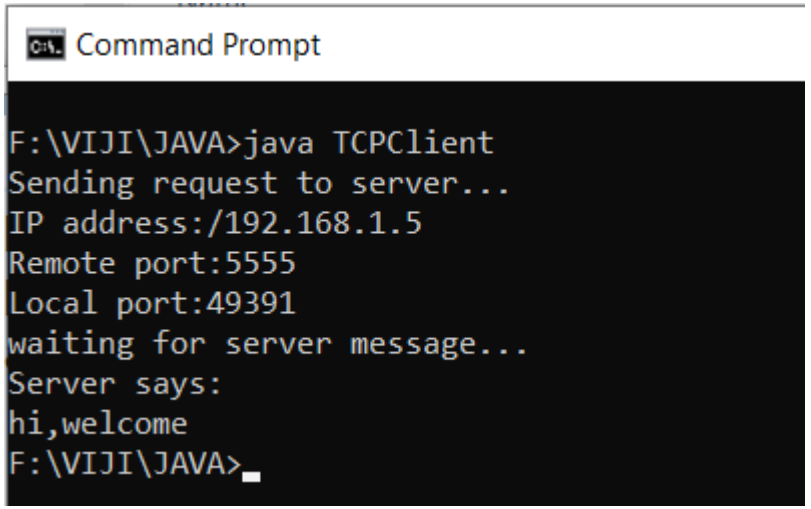
Get IP Address of a System - ipconfig

```
G:\JAVA_PGMS>ipconfig

Windows IP Configuration


Ethernet adapter Ethernet:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 3:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 4:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Ethernet adapter Ethernet 2:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Ethernet adapter Ethernet 3:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::f9af:388e:4e11:495e%6
   IPv4 Address. . . . . . . . . . . : 192.168.1.5
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : fe80::1%6
                                       192.168.1.1
```

TCP SERVER



TCP CLIENT

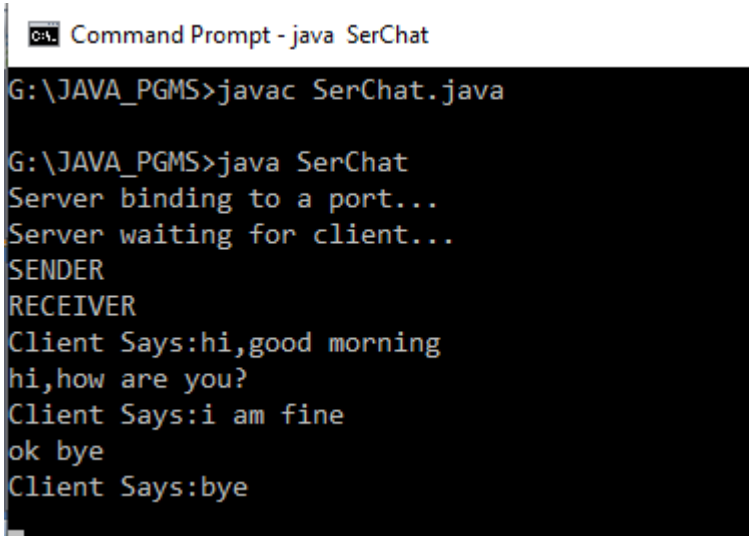


Server & Client simultaneously send and receive message to each other (chatting – Peer to Peer Communication)

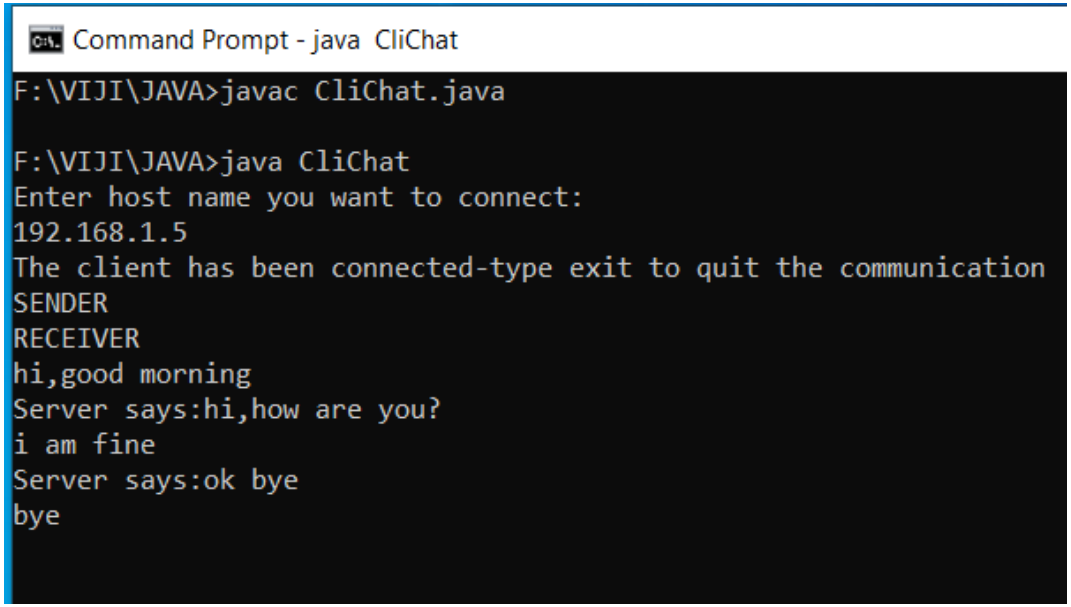
Chat Server

Chat Client

CHAT -SERVER



CHAT CLIENT

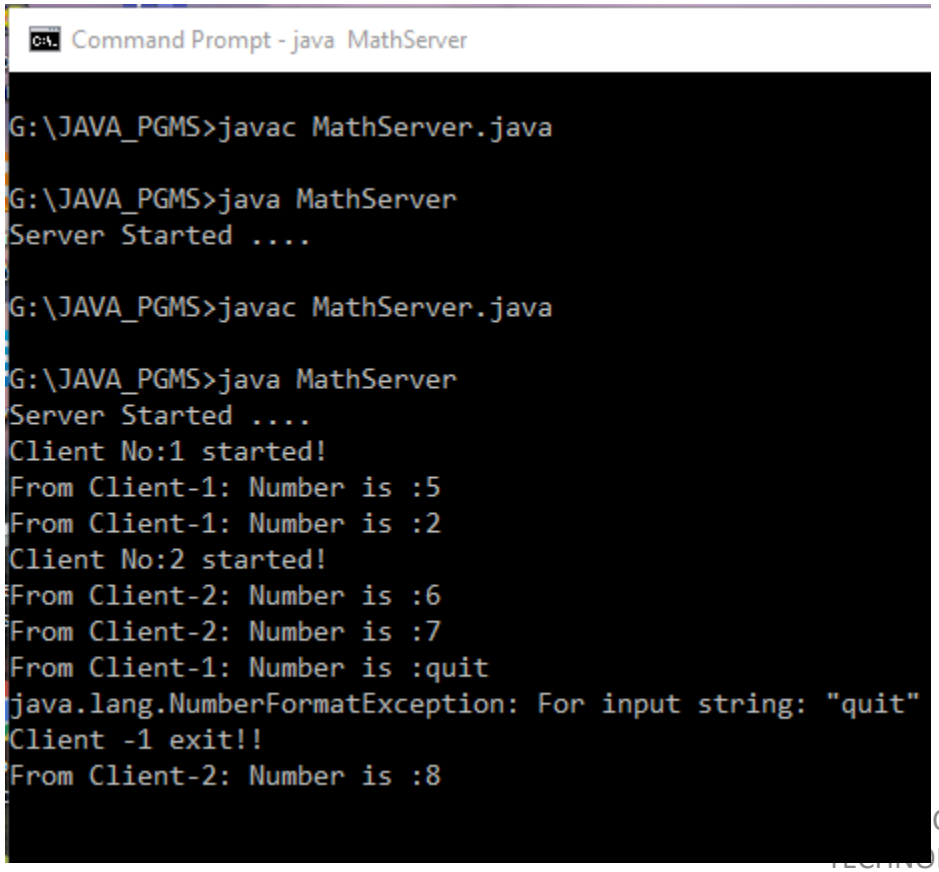


Multiple clients served by a single Server (A mathematical server find the square and cube of a number given by the clients)

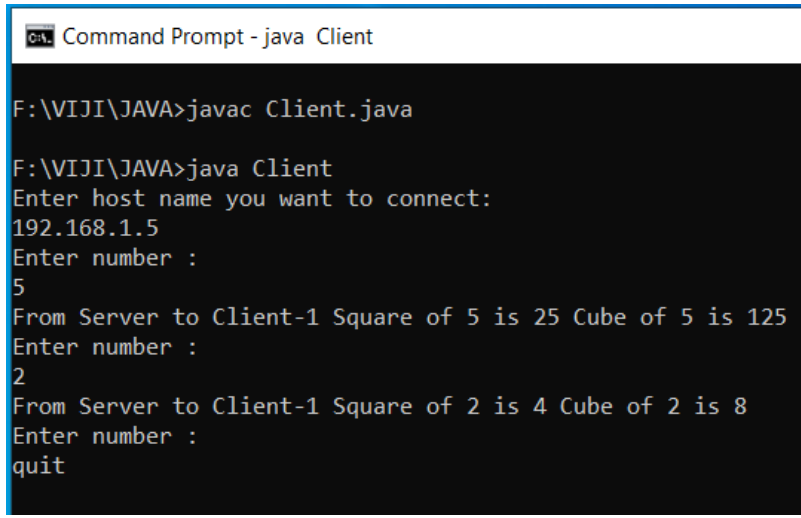
Server

Client

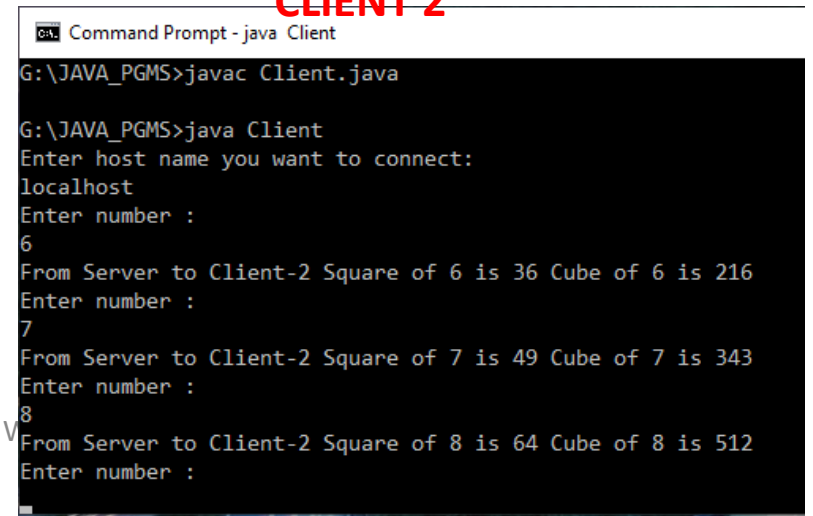
SERVER



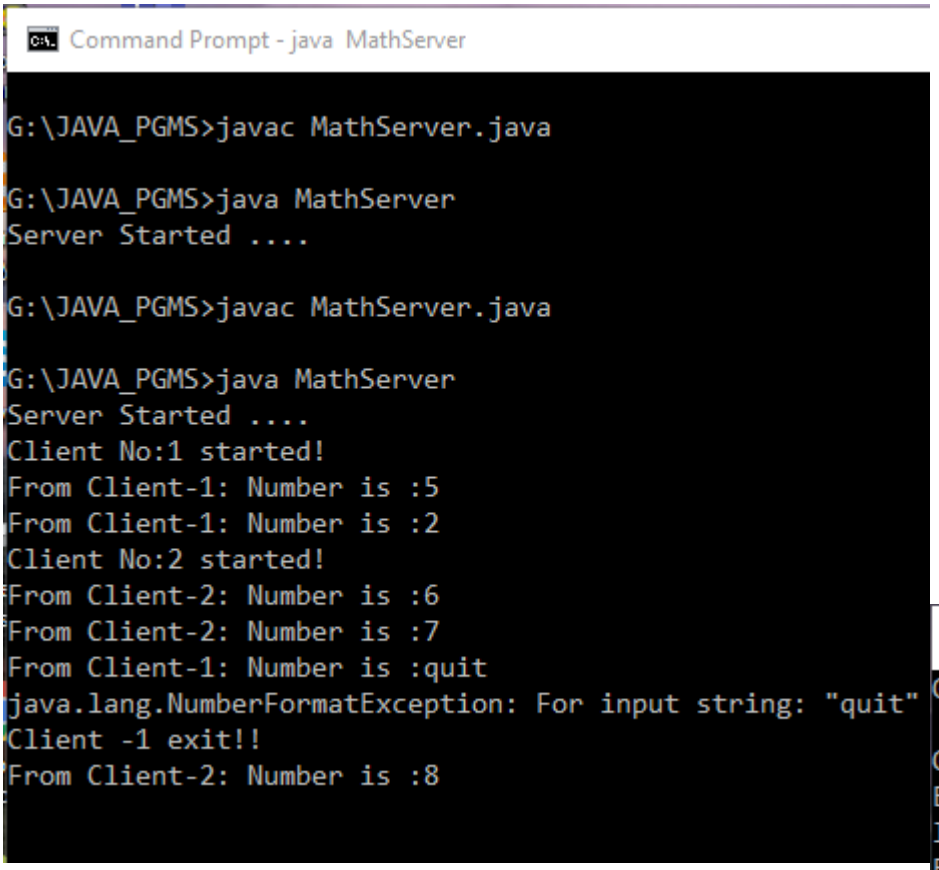
CLIENT 1



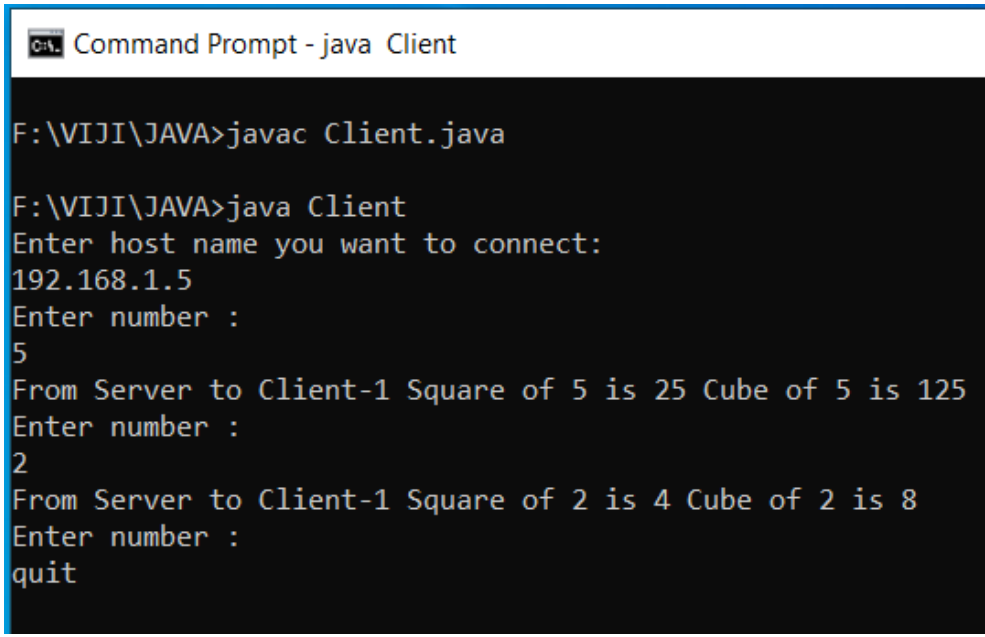
CLIENT 2



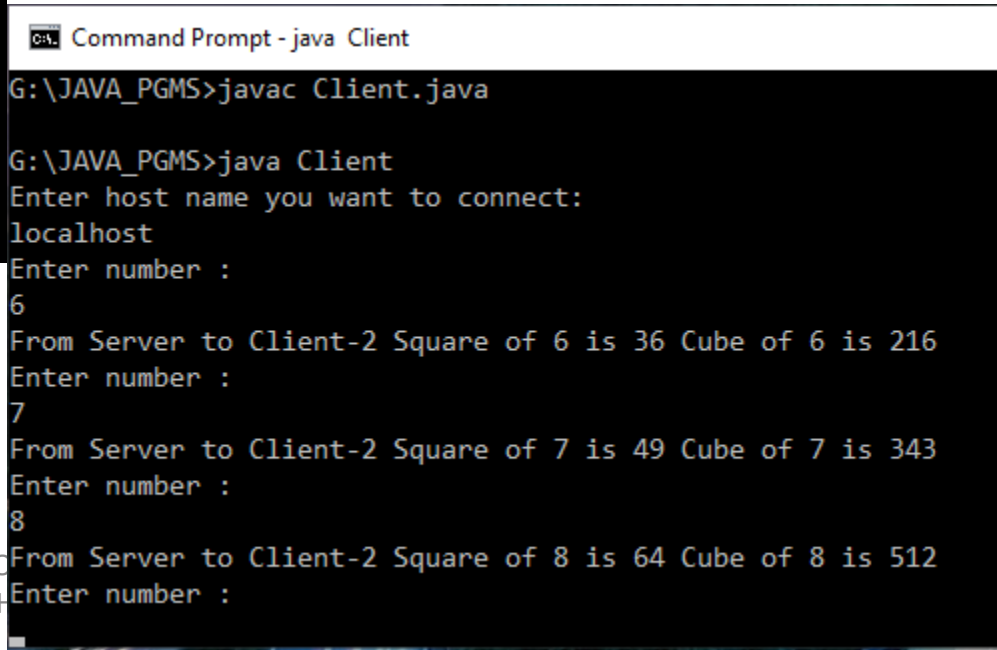
SERVER



CLIENT 1



CLIENT 2



UDP
DatagramSocket

- This class represents a connection-less socket for sending and receiving datagram packets.
- **Constructors**
  - **DatagramSocket() throws SocketException:** it creates a datagram socket and binds it with the available Port Number on the localhost machine.
  - **DatagramSocket(int port) throws SocketException:** it creates a datagram socket and binds it with the given Port Number.
  - **DatagramSocket(int port, InetAddress address) throws SocketException:** it creates a datagram socket and binds it with the specified port number and host address.

DatagramPacket

- **Java DatagramPacket** is a message that can be sent or received.
- If we send multiple packet, it may arrive in any order.
- Additionally, packet delivery is not guaranteed.
- **Constructors**
  - **DatagramPacket(byte[] barr, int length):** it creates a datagram packet. This constructor is used to receive the packets.
  - **DatagramPacket(byte[] barr, int length, InetAddress address, int port):** it creates a datagram packet. This constructor is used to send the packets.

Destination address

Destination port

DatagramPacket Methods

- **send(DatagramPacket d)** throws IOException

  → dispatches the given DatagramPacket object
- **receive(DatagramPacket p)** throws IOException

  → receives the given DatagramPacket
- **close()**

  → close the socket connection

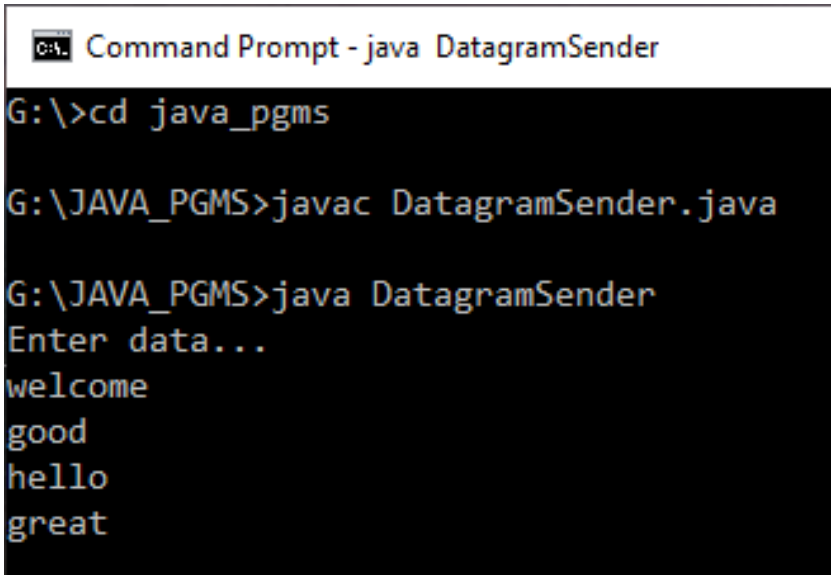
DatagramSender

```
import java.io.*;
import java.net.*;
class DatagramSender
{
  public static void main(String a[]) throws Exception
  {
     int serport=2000,cliport=3000;
     DatagramSocket ds=new DatagramSocket(serport);
     byte buff[]=new byte[1024];
     BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
     System.out.println("Enter data...");
     InetAddress ia=InetAddress.getByName("localhost");
    //InetAddress ia=InetAddress.getByName("192.168.115.65");//Receiver IP address
     while(true)
     {
              String str=br.readLine();
              if((str==null || str.equals("end")))
                       break;
              buff=str.getBytes();
              ds.send(new DatagramPacket(buff,buff.length,ia,cliport));
      }
  }
}
```

DatagramReceiver

```
import java.io.*;
import java.net.*;
class DatagramReceiver
{
        public static void main(String a[]) throws Exception
        {
                int cliport=3000;
                byte buff[]=new byte[1024];
                DatagramSocket ds=new DatagramSocket(cliport);

                System.out.println("Client is waiting for message...");
                System.out.println("Press ctrl+c to come out...");
                while(true)
                {
                        DatagramPacket dp=new DatagramPacket(buff,buff.length);
                        ds.receive(dp);
                        String str=new String(dp.getData(),0,dp.getLength());
                        System.out.println(str);
                }
        }
}
```

Datagram Sender



Datagram Receiver