Chapter-4
Java Network Programming

Introduction to Network programming

- 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 contains a collection of classes and interfaces that provide the low-level communication details, allowing you to write programs that focus on solving the problem at hand.
- The java.net package provides support for the two common network protocols:
  1. **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.
  2. **UDP:** UDP stands for User Datagram Protocol, a connection-less protocol that allows for packets of data to be transmitted between applications.
- **Protocols** are set of rules required for exchanging of information between computers on the network. So network protocols are the set of rules and conventions followed by system that can communicate over a network.

TCP VS UDP

	TCP	UDP
Definition	TCP is connection based protocol that provides a reliable flow of data between two computers.	UDP is a connection less protocol that sends independent packets of data, called datagram, from one computer to another with no guarantee about arrival.
Acronym for	Transmission control protocol	User Datagram protocol
Connection	TCP is connection oriented protocol	UDP is connection less protocol
Usage	TCP is used for applications that require high reliability and transmission time is really less critical.	UDP is suitable for applications that need fast, efficient transmission such as game.
Ordering of Data packets	TCP rearrange data packets in the order specified hence provide delivery in sequenced order.	UDP doesn't provide ordering of packets as each packet is independent of each other hence datagram packets may arrive in different order.
Speed	The speed of TCP is slower than UDP	The speed of UDP is faster than TCP
Error Checking	TCP does error checking.	UDP also does error checking but no recovery option.
Acknowledge	Provides acknowledgement	No acknowledgement
Package	In java.net API package, URL, URLConnection, Socket and ServerSocket all class use TCP to communicate over the network.	In java.net API package, DatagramPacket, DatagramSocket and MulticastSocket all class uses UDP to communicate over the network.
Example	Application like webpage, File Transfer etc. uses TCP.	Example: Application like video conference, internet telephony etc. uses UDP.

IP Address

- IP address is a unique string of numbers separated by full stops that identifies each computer using the Internet Protocol to communicate over a network.
- It is 32 bit decimal or hexadecimal number represented as a series of four 8 bit numbers ranging in the value from 0 to 255.
- Example: 192. 168.32.192.
- The DNS (Domain Naming System) translates the domain name into the appropriate IP address thus allowing us to type domain name instead of remembering the IP address.

Example: The DNS translated www.google.com into the IP address 120.89.96.148.

- **To obtain the IP address, we can use InetAddress class provided by java.net package.**
- Following program demonstrates how to obtain the IP address.

```
import java.net.*;
public class InetAddressDemo
{
	public static void main(String [] args) throws Exception //for UnKnownHostException
	{
		InetAddress ia=InetAddress.getLocalHost();
		InetAddress ia1=InetAddress.getByName("www.google.com");
		System.out.println("The name and IP address of local host is:- "+ia);
		System.out.println("The name and IP address of GOOGLE is:- "+ia1);
	}
}
```

Output:

```
C:\Users\Hp-User\Desktop\Java programs\Network Programming>java InetAddressDemo
The name and IP address of local host is:- Hp/192.168.226.1
The name and IP address of GOOGLE is:- www.google.com/74.125.130.147
```

Port Number

- A port is a 16-bit number (0 to 65,535), which TCP and UDP use to deliver the data to the right application.
- The TCP and UDP protocols use Ports to map incoming data to a particular process running on a computer.
- The ports from 0 to 1023 are reserved for use by system services and hence can't be used. These ports are called **well-known port**. Some default port number 80 for HTTP, 21 for FTP etc.

URL and URL Classes

- URL stands for Uniform Resource Locator which is an address to resources on the internet.
- 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
- Some of the constructor provided by URL class are
    1. URL(String spec): Creates a URL object from the String representation.
    2. URL(String protocol, String host, String file): Creates a URL from the specified protocol name, host name, and file name.
    3. URL(String protocol, String host, int port, String file): Creates a URL object from the specified protocol, host, port number, and file.
- Example:

    https://www.facebook.com

    Where, https is protocol, facebook.com is Resource name.

- Some of the methods of URLclass are listed below.
    1. String getFile(): Gets the file name for this URL
    2. String getHost(): Gets the host name for this URL.
    3. getProtocol(): Gets the protocol name of this URL
    4. getQuery(): Gets the query part of this URL.
    5. URLConnection openConection(): Returns a URLConection objects that represents a connection to the remote object referred to by this URL.
    6. InputStream getInputStream(): returns InputStream for reading from that connection.
    7. String toExternalForm(): construct string representation of this URL.

8. int getPort(): Return the port number of this URL, -1 if not available

Example1: Java program to demonstrate URL class methods

```
import java.net.*;
public class URLClassDemo
{
	public static void main(String [] args) throws Exception //UnKnownHostException
	{
		URL x=new URL("https://www.facebook.com/groups/388363178235197/?ref=bookmarks");
		String f=x.getFile();
		System.out.println("File name="+f);
		String h=x.getHost();
		System.out.println("Host name="+h);
		String p=x.getPath();
		System.out.println("Path part="+p);
		String pr=x.getProtocol();
		System.out.println("Protocol="+pr);
		String q=x.getQuery();
		System.out.println("Query Part="+q);
	}
}
```

Output:

```
C:\Users\Bheeshma\Desktop>java URLClassDemo
File name=/groups/388363178235197/?ref=bookmarks
Host name=www.facebook.com
Path part=/groups/388363178235197/
Protocol=https
Query Part=ref=bookmarks
```

Example2: JAVA program to demonstrate how to read data from URL

```
import java.net.*;
import java.io.*;
public class URLDataDemo
{
	public static void main(String [] args) throws Exception //UnKnownHostException
	{
		URL fb=new URL("http://www.google.com/");
		URLConnection con=fb.openConnection();// openConnection() method opens a socket to the specified
                                                      url and returns a URLConnection object.
		InputStreamReader isr=new InputStreamReader(con.getInputStream());
		BufferedReader br=new BufferedReader(isr);
		String data;
		while((data=br.readLine())!=null)
		{
			System.out.println(data);
		}
	}
}
```

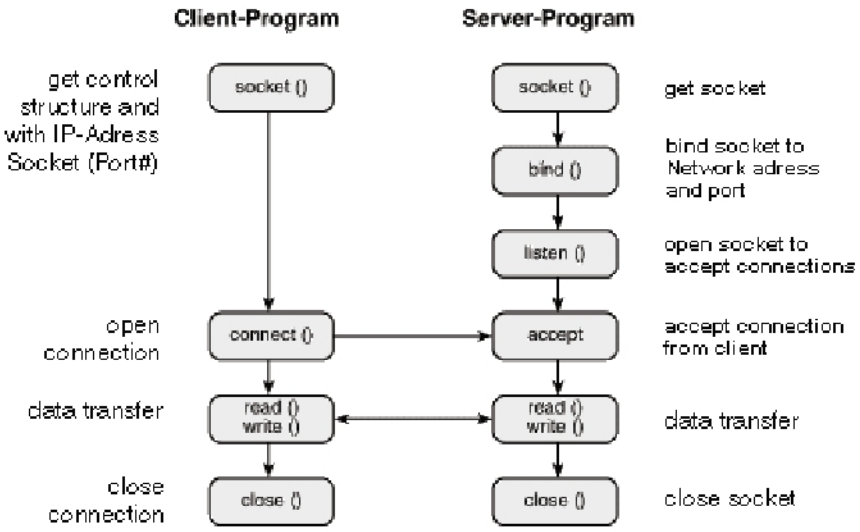
Ouput:

The above program will display the content of google.com in HTML form.

Creating a client/server Application using TCP Socket

- Networks in which certain computers have special dedicated tasks, providing services to other computers (in the network) are called client-server networks.

- Examples of computer applications that use the client–server model are Email, network printing, and the World Wide Web.
- In client-server architecture two types of process are involved in communication as below.
  a. **Client Process:**
  This is the process which typically makes a request for information. After getting the response this process may terminate or may do some other processing. For example: Internet Browser works as a client application which sends a request to Web Server to get one HTML web page.
  b. **Server Process:**
  This is the process which takes a request from the clients. After getting a request from the client, this process will do required processing and will gather requested information and will send it to the requestor client. Once done, it becomes ready to serve another client. Server process are always alert and ready to serve incoming requests. For example: Web Server keeps waiting for requests from Internet Browsers and as soon as it gets any request from a browser, it picks up a requested HTML page and sends it back to that Browser.
- In client-server communications the client needs to know of the existence and the address of the server, but the server does not need to know the address or even the existence of the client prior to the connection being established.
- Once a connection is established, both sides can send and receive information as shown in the figure below.



- A socket is a class for communication between a client program and a server program in a network.

- **The steps involved in creating client side application are as follows:**
  1. **Connect to the server using socket object.**
  Socket class is used to establish connection between client and server. Socket class takes two parameters, IP address and port number .

  Socket sc=new Socket("hostName",PortNumber);

For example: Socket sc=new Socket("192.168.0.1",6789); or Socket sc=new Socket("localhost",6789)

  2. **Read and write to the socket.**
  I/O stream classes are used to read and write socket. Two methods getInputStream() and getOutputStream() are used to read from and write to socket respectively.

  3. **Close the connection.**
  After performing the read and write operation, we use close() method to close the stream and connection.
  Example: sc.close(): this method will shutdown both input and output from the socket.

- The steps involved in creating server side application
  1. **Create a server by using SocketServer object.**
  The ServerSocket class is used to create a server socket. Constructor for creating server socket is

  ServerSocket(int port)

  This creates a server socket as specified port number on the local computer.
  2. **Listen for the client request.**
  The accept() method which return client socket is used to listen the client. The accept() method is invoked with server socket and it returns client socket.

3. **Read input from and send output to the client.**
   We use getInputStream() and getOutputStream() methods for reading input from and sending to the client.
4. **Close the connection.**
   Final, we use close() method to close all the stream and connection.

Note: We also can use DataInputStream to read and write in console and file.

```
import java.io.*;
class Demo
{
        public static void main(String [] args) throws IOException
        {
                DataInputStream dis=new DataInputStream(System.in);
                System.out.println("Enter your name");
                String nm=dis.readLine();
                //System.out.println("Name : "+nm);

                DataOutputStream dos=new DataOutputStream(System.out);
                dos.writeBytes(nm);  //
                dis.close();
        }
}
```

Solved Examples:

1. Write a socket program to receive and display the "Welcome " message on a client machine from the server.

Client side Program:

```
import java.io.*;
import java.net.*;
class Clientside
{
        public static void main(String [] args) throws IOException
        {
                Socket soc=new Socket("localhost",1212);

                DataInputStream dis=new DataInputStream(soc.getInputStream());
                System.out.println(dis.readLine());
                dis.close();
                soc.close();
        }
}
```

Server side Program:

```
import java.io.*;
import java.net.*;
class Serverside
{
        public static void main(String [] args) throws IOException
        {
                ServerSocket ss=new ServerSocket(1212);
                while(true)
                {
                        System.out.println("Waiting for connection");
                        Socket acc=ss.accept();

                        PrintStream ps=new PrintStream(acc.getOutputStream());
                        ps.println("Welcome Bhesh Bdr Thapa");
```

```
                ps.close();
            }
        }
    }
```

2. Write a socket program in which the client receive name of a student. The server then displays this information.

Client side Program

```
import java.net.*;
import java.io.*;
import java.util.Scanner;
public class Client
{
    public static void main(String [] args) throws Exception
    {
        Socket cs=new Socket("localhost",1010);
        Scanner stdin=new Scanner(System.in);
        System.out.println("Enter the Name");
        String name=stdin.nextLine();

        PrintStream ps=new PrintStream(cs.getOutputStream());
        ps.println(name);

        ps.close();
        cs.close();

    }
}
```

Server side Program

```
import java.net.*;
import java.io.*;
public class Server
{
    public static void main(String [] args) throws Exception
    {
        ServerSocket ss=new ServerSocket(1010);
        while(true)
        {
            Socket cs=ss.accept();

            DataInputStream in=new DataInputStream(cs.getInputStream());
            String name=in.readLine();
            System.out.println("Name="+name);

            in.close();
        }
    }

}
```

3. Write a socket program in which the client receive name and roll of a student. The server then displays these informations.

Client side program:

```
import java.net.*;
import java.io.*;
import java.util.Scanner;
public class Client1
{
    public static void main(String [] args) throws Exception
    {
            Socket cs=new Socket("localhost",1010);

            Scanner stdin=new Scanner(System.in);
            System.out.println("Enter the Name");
            String name=stdin.nextLine();
            System.out.println("Enter the roll");
            String roll=stdin.nextLine();

            PrintStream out=new PrintStream(cs.getOutputStream());
            out.println(name);
            out.println(roll);

            out.close();
            cs.close();
    }
}
```

Server side program:

```
import java.net.*;
import java.io.*;
public class Server1
{
        public static void main(String [] args) throws Exception
        {
                ServerSocket ss=new ServerSocket(1010);
                while(true)
                {
                        Socket cs=ss.accept();

                        DataInputStream in=new DataInputStream(cs.getInputStream());
                        String name=in.readLine();
                        String roll=in.readLine();
                        System.out.println("Name="+name);
                        System.out.println("Roll="+roll);

                        in.close();
                }
        }
}
```

4. Write a socket program in which the client program receives radius from the user and sends to the server. The server then computes radius of the circle and sends the result back to client. The client then displays the result.

Client side program:

```
import java.net.*;
import java.util.Scanner;
import java.io.*;
public class Client
{
    public static void main(String [] args) throws Exception
    {
            Socket cs=new Socket("localhost",2000);

            Scanner sc=new Scanner(System.in);
            System.out.println("Enter the radius of circle");
            String rad=sc.nextLine();

            PrintStream out=new PrintStream(cs.getOutputStream());
            out.println(rad);

            DataInputStream in=new DataInputStream(cs.getInputStream());
            System.out.println("The area of circle is "+in.readLine());

            out.close();
            in.close();
            cs.close();
    }
}
```

Server side program:

```
import java.net.*;
import java.io.*;
public class Server
{
        public static void main(String [] args) throws Exception
        {
                ServerSocket ss=new ServerSocket(2000);
                while(true)
                {
                        Socket cs=ss.accept();

                        DataInputStream dis=new DataInputStream(cs.getInputStream());
                        String rad=dis.readLine();

                        PrintStream ps=new PrintStream(cs.getOutputStream());
                        double area=3.14*Double.parseDouble(rad)*Double.parseDouble(rad);
                        ps.println(area);

                        ps.close();
                        dis.close();
                }
        }
}
```

5. Write a socket program to accept user ID and password from client side and check whether the user id is “sa” and password is “Bhesh” by the server. If it matches, send message to the client “you are welcome” otherwise send “access denied”.

Client side Program:

```
import java.net.*;
import java.io.*;
import java.util.Scanner;
public class Client
{
    public static void main(String [] args) throws Exception
    {
        Socket cs=new Socket("localhost",1010);

        String lid;
        String lid1,pwd;
        String s,s1,s2;
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter the login id");
        lid=sc.nextLine();
        System.out.println("Enter your password");
        pwd=sc.nextLine();

        PrintStream out=new PrintStream(cs.getOutputStream());
        out.println(lid);
        out.println(pwd);

        DataInputStream in=new DataInputStream(cs.getInputStream());
        s=in.readLine();
        s1=in.readLine();
        s2=in.readLine();
        System.out.println(s);
        System.out.println(s1);
        System.out.println(s2);

        out.close();
        in.close();
        cs.close();
    }
}
```

Server side program:

```
import java.net.*;
import java.io.*;
public class Server
{
    public static void main(String [] args) throws Exception
    {
        ServerSocket ss=new ServerSocket(1010);

        String pwd,lid,s;
        while(true)
        {
            Socket cs=ss.accept();
            DataInputStream in=new DataInputStream(cs.getInputStream());
            lid=in.readLine();
            pwd=in.readLine();

            PrintStream out=new PrintStream(cs.getOutputStream());
            if(lid.equals("sa")&& pwd.equals("Bhesh"))
            {
                out.println("You are welcome");
            }
```

```
                else
                {
                        out.println("Access Denied");
                }
                out.println("Logine id ="+lid);
                out.println("password ="+pwd);

                out.close();
                in.close();
        }
    }
}
```

6. Write a socket program to accept a line of string input from the client and change it to upper case string and send it to client. Client machine display the sentence received from server.

Client side Program:

```
import java.net.*;
import java.util.Scanner;
import java.io.*;
public class Client
{
    public static void main(String [] args) throws Exception
    {
        Socket cs=new Socket("localhost",1010);

        Scanner sc=new Scanner(System.in);
        System.out.println("Enter the string");
        String str=sc.nextLine();

        PrintStream out=new PrintStream(cs.getOutputStream());
        out.println(str);

        DataInputStream dis=new DataInputStream(cs.getInputStream());
        String res=dis.readLine();
        System.out.println("String in upper case = "+res);

        out.close();
        dis.close();
        cs.close();
    }
}
```

Server side Program:

```
import java.net.*;
import java.io.*;
public class Server
{
        public static void main(String [] args) throws Exception
        {
                ServerSocket ss=new ServerSocket(1010);
                while(true)
                {
```

```
                    Socket cs=ss.accept();

                    DataInputStream dis=new DataInputStream(cs.getInputStream());
                    String str=dis.readLine();

                    PrintStream out=new PrintStream(cs.getOutputStream());
                    String res=str.toUpperCase();
                    out.println(res);

                    out.close();
                    dis.close();
                }
            }
        }
```

7. Write a socket program to accept a line of string input from the client and change it to alternate case string and send it to the client. Client machine display the sentence received from the server.

Client side Program:

```
import java.net.*;
import java.util.Scanner;
import java.io.*;
public class Client
{
    public static void main(String [] args) throws Exception
    {
        Socket cs=new Socket("localhost",1010);

        Scanner sc=new Scanner(System.in);
        System.out.println("Enter the string");
        String str=sc.nextLine();

        PrintStream out=new PrintStream(cs.getOutputStream());
        out.println(str);

        DataInputStream dis=new DataInputStream(cs.getInputStream());
        String res=dis.readLine();
        System.out.println("String in alternate case = "+res);

        out.close();
        dis.close();
        cs.close();
    }
}
```

Server side Program:

```
import java.net.*;
import java.io.*;
public class Server
{
    public static void main(String [] args) throws Exception
    {
        ServerSocket ss=new ServerSocket(1010);
        while(true)
        {
```

```
                Socket cs=ss.accept();

                DataInputStream dis=new DataInputStream(cs.getInputStream());
                String str=dis.readLine();

                PrintStream out=new PrintStream(cs.getOutputStream());
                String res="";
                for(int i=0;i<str.length();i++)
                {
                        String ext=str.substring(i,i+1);
                        if(i%2==0)
                        {
                                res=res+ext.toLowerCase();
                        }
                        else
                        {
                                res=res+ext.toUpperCase();
                        }
                }
                out.println(res);

                out.close();
                dis.close();

        }
    }
}
```

8. Write a client server program using TCP in which the client program reads the length and breadth of a rectangle from the user and sends to the server program. The server program computes the area and perimeter and return to the client program which display the obtained area and perimeter.

Client side Program:

```
import java.net.*;
import java.util.Scanner;
import java.io.*;
public class Client
{
        public static void main(String [] args) throws Exception
        {
                Socket cs=new Socket("localhost",1010);

                Scanner sc=new Scanner(System.in);
                System.out.println("Enter the length of rectangle");
                String len=sc.nextLine();
                System.out.println("Enter the breadth of rectangle");
                String bre=sc.nextLine();

                PrintStream out=new PrintStream(cs.getOutputStream());
                out.println(len);
                out.println(bre);

                DataInputStream dis=new DataInputStream(cs.getInputStream());
                String area=dis.readLine();
```

```
                String per=dis.readLine();
                System.out.println("The area of rectangleis "+area);
                System.out.println("The perimeter of rectangele is"+per);

                out.close();
                dis.close();
                cs.close();
        }
}
```

Server side Program:

```
import java.net.*;
import java.io.*;
public class Server
{
        public static void main(String [] args) throws Exception
        {
                ServerSocket ss=new ServerSocket(1010);
                while(true)
                {
                        Socket cs=ss.accept();

                        DataInputStream dis=new DataInputStream(cs.getInputStream());
                        String len=dis.readLine();
                        String bre=dis.readLine();

                        PrintStream out=new PrintStream(cs.getOutputStream());
                        double area=Integer.parseInt(len)*Integer.parseInt(bre);
                        double per=2*(Integer.parseInt(len)+Integer.parseInt(bre));
                        out.println(area);
                        out.println(per);

                        out.close();
                        dis.close();
                }
        }
}
```

9. Write a client server program using TCP in which the client program reads the radius of a circle from the user and sends to the server program. The server program computes the area and circumference of circle and return results to the client program which display the obtained area and circumference.

Client side program:

```
import java.net.*;
import java.util.Scanner;
import java.io.*;
public class Client
{
        public static void main(String [] args) throws Exception
        {
                Socket cs=new Socket("localhost",1010);

                Scanner sc=new Scanner(System.in);
                System.out.println("Enter the radius of circle");
                String rad=sc.nextLine();
```

```
                PrintStream ps=new PrintStream(cs.getOutputStream());
                ps.println(rad);

                DataInputStream dis =new DataInputStream(cs.getInputStream());
                String area=dis.readLine();
                String cir=dis.readLine();

                System.out.println("The area of circle is"+area);
                System.out.println("The circumference of circle is "+cir);

                dis.close();
                ps.close();
                cs.close();
        }
    }
```

Server side program:

```
import java.net.*;
import java.io.*;
public class Server
{
    public static void main(String [] args) throws Exception
    {
        ServerSocket ss=new ServerSocket(1010);
        while(true)
        {
            Socket cs=ss.accept();
            DataInputStream dis=new DataInputStream(cs.getInputStream());
            String rad=dis.readLine();
            double area=3.14*Integer.parseInt(rad)*Integer.parseInt(rad);
            double circum=2 * 3.14 * Integer.parseInt(rad); //Double.parseDouble(rad);

            PrintStream ps=new PrintStream(cs.getOutputStream());
            ps.println(area);
            ps.println(circum);

            dis.close();
            ps.close();
        }
    }
}
```

Assignments:

1. Write a client server program using TCP in which the client program reads a string from the user and sends to the server program. The server program then checks if the string is palindrome or not and return the result to the client. The client then displays the result.
2. Write a client server program using TCP in which the client program reads a number from the user and sends to the server program. The server program then reverses the digit in the number and return the result. The client then displays the result.

Creating Client/Server application using UDP

- To establish the UDP based communication, we use **DatagramSocket and DatagramPacket class.**
- Two methods **send() and receive()** are used to send and receive datagram packets.
- Application that communicate through datagrams send and receive completely independent packets of information.
- The delivery of datagrams to their destinations is not guaranteed, and order of arrival is not either.
- The DatagramPacket class has many constructors one of which is

DatagramPacket(byte[] buf, int length, InetAddress address, int port)

Steps for Creating the server application

1. Obtain the address of client to which the information is to be sent.
2. Create the DatagramSocket object.
3. Prepare DatagramPacket
4. Send the prepared DatagramPacket using send() method.

Steps for creating the client application

1. Client side also uses both DatagramSocket and DatagramPacket objects and receive() method.
2. A buffer is created to hold character received.
3. A socket is created to listen at a particular port and DatagramPacket object is created to collect data.
4. Once the socket and packet objects are setup, all that is required to receive the data is a call to DatagramSocket class's receive() method.

Example

1. Write a client server program using Datagram to receive and display the "Welcome " message on a client machine from the server.

Client Side

```
import java.net.*;
import java.io.*;
import java.util.Arrays;
public class Client
{
	public static void main(String [] args) throws Exception
	{
		DatagramSocket cs=new DatagramSocket();

		byte []senddata=new byte[200];
		byte []recdata=new byte[200];

		InetAddress ia=InetAddress.getByName("localhost");

		DatagramPacket dpsend=new DatagramPacket(senddata,senddata.length,ia,5020);
		cs.send(dpsend);

		DatagramPacket dprec=new DatagramPacket(recdata,recdata.length);
		cs.receive(dprec);

		String msg=new String (dprec.getData());
		System.out.println(msg);

		/*
			Also we can display result from buffer
			String m="";
			for(int i=0;i<recdata.length;i++)
			{
				char x=(char)recdata[i];
				m=m+x;
			}
			System.out.println(m);
		*/
		cs.close();


	}
```

```
	}
```

Server Side

```
import java.net.*;
import java.io.*;
public class Server
{
	public static void main(String [] args) throws Exception
	{
		DatagramSocket cs=new DatagramSocket(5020);
		byte []senddata=new byte[200];
		byte []recdata=new byte[200];

		while(true)
		{
			DatagramPacket dprec=new DatagramPacket(recdata,recdata.length);
			cs.receive(dprec);

			int pn=dprec.getPort();
			InetAddress ia=dprec.getAddress();

			String msg="Welcome Client";
			senddata=msg.getBytes();
			DatagramPacket dpsend=new DatagramPacket(senddata,senddata.length,ia,pn);
			cs.send(dpsend);
		}


	}

}
```

2. Write a client server program using Datagram in which the client program receives the radius of the circle form the use and sends to the server program that computes the area of circle and sends to the client. The client then displays the result.

Client side program:

```
import java.io.*;
import java.net.*;
import java.util.*;
public class Client
{
	public static void main(String [] args) throws Exception
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the radius");
		String rad=sc.next();

		byte[] sendData=new byte[10]; // byte array
		sendData=rad.getBytes();

		InetAddress adrs=InetAddress.getByName("localhost");
		DatagramPacket dpsend=new DatagramPacket(sendData,sendData.length,adrs,1020);
		DatagramSocket cs=new DatagramSocket();
		cs.send(dpsend);

		byte[] receiveData=new byte[10];
```

```
                    DatagramPacket dprec=new DatagramPacket(receiveData,receiveData.length);
                    cs.receive(dprec);
                    String res=new String(dprec.getData());
                    System.out.println("The area of circle is"+res);
                    sc.close();
                }
            }
```

Server side program:

```
import java.io.*;
import java.net.*;
public class Server
{
    public static void main(String [] args) throws Exception
    {
        DatagramSocket sc=new DatagramSocket(1020);
        byte sendData[]=new byte[10];
        byte receiveData[] =new byte[10];

        while(true)
        {
            DatagramPacket dprec=new DatagramPacket(receiveData,receiveData.length);
            sc.receive(dprec);
            String rad=new String(dprec.getData());
            double area=3.14 *Double.parseDouble(rad)*Double.parseDouble(rad);
            sendData=Double.toString(area).getBytes();//convert to string to get bytes

            InetAddress addrs=dprec.getAddress();
            int port=dprec.getPort();
            DatagramPacket dpsend=new DatagramPacket(sendData,sendData.length,addrs,port);
            sc.send(dpsend);
        }

`       }
}
```

Serving Multiple Clients

- The multicast datagram socket class is useful for sending and receiving IP multicast packets.
- A MulticastSocket is a (UDP) DatagramSocket, with additional capabilities for joining "groups" of other multicast hosts on the internet.
- A multicast group is specified by a class D IP address and by a standard UDP port number.
- Class D IP addresses are in the range 224.0.0.0 to 239.255.255.255, inclusive.
- The address 224.0.0.0 is reserved and should not be used.
- One would join a multicast group by first creating a MulticastSocket with the desired port, then invoking the joinGroup(InetAddress groupAddr) method
- When one sends a message to a multicast group, all subscribing recipients to that host and port receive the message (within the time-to-live range of the packet, see below). The socket needn't be a member of the multicast group to send messages to it.
- One can leave a multicast group by invoking the leaveGroup(InetAddress groupAddr) method

WAP to demonstrate serving multiple clients

Server-side

```
import java.io.*;
import java.net.*;
class MultiCastSender
{
        public static void main(String [] args) throws IOException
        {
                InetAddress group=InetAddress.getByName("228.5.6.7");
                MulticastSocket ms=new MulticastSocket(6789);

                byte [] senddata=new byte[200];
                String msg="hello this is multicasted message";
                senddata=msg.getBytes();
                DatagramPacket dpsend=new DatagramPacket(senddata,senddata.length,group,6789);
                ms.send(dpsend);
        }
}
```

Client-side

```
import java.io.*;
import java.net.*;
class MultiCastReceiver
{
        public static void main(String [] args) throws Exception
        {
                InetAddress group=InetAddress.getByName("228.5.6.7");

                MulticastSocket ms=new MulticastSocket(6789);                    -
                ms.joinGroup(group);

                byte [] recdata=new byte[200];
                DatagramPacket dprec=new DatagramPacket(recdata,recdata.length,group,6789);
                ms.receive(dprec);

                String msg=new String (dprec.getData());
                System.out.println(msg);

        }
}
```

Note:

1. Run two version of client. Both will wait for server to response.
2. Run Server then we will see both the client will get the response at same time from server

Modify the above program so that server can type a message until he presses "quit" and send to all the connected clients.

Client :

```
import java.io.*;
import java.net.*;
class Client
{
        public static void main(String [] args) throws Exception
        {
                InetAddress group=InetAddress.getByName("228.5.6.7");
```

```
                MulticastSocket ms=new MulticastSocket(6789);
                ms.joinGroup(group);

                while(true)
                {

                        byte [] recdata=new byte[200];
                        DatagramPacket dprec=new DatagramPacket(recdata,recdata.length,group,6789);
                        ms.receive(dprec);

                        String msg=new String (dprec.getData());
                        System.out.println(msg);
                }

        }
}
```

Server:

```
import java.io.*;
import java.net.*;
import java.util.*;
class Server
{
        public static void main(String [] args) throws IOException
        {
                InetAddress group=InetAddress.getByName("228.5.6.7");
                MulticastSocket ms=new MulticastSocket(6789);
                byte [] senddata=new byte[200];

                String txt="";
                Scanner sc=new Scanner(System.in);
                while(txt!="quit")
                {
                        txt=sc.nextLine();
                        senddata=txt.getBytes();
                        DatagramPacket dpsend=new DatagramPacket(senddata,senddata.length,group,6789);
                        ms.send(dpsend);
                }
        }
}
```

Half Closed Socket

- The close() method shutdown both the input and output from the socket.
- So when it is need to shutdown only half of the connection either input or output we use the following two methods to close half of the connection.

    1. void shutdownInput()
    2. void shutdownOutput()

- so once the above methods are executed further read from the socket will return -1, and further write in the socket will throw and IOException.

Example:

```
Socket soc=new Socket("localhost",1212);
```

So, the statement soc.close() will shutdown both input and output from the socket. But if we only need to shutdown output from the socket we can call soc.shutdownOutput() method which will still enable us to input to the socket but not output.

Assignment

1. **WAP to demonstrate sending email in java.**

Sending Email in Java

```
import java.util.*;
import javax.mail.*;
import javax.mail.internet.*;
import javax.activation.*;

public class SendEmail
{

	public static void main(String [] args)  throws Exception
	{
		// Recipient's email ID needs to be mentioned.
		String to = "bheeshmathapa@gmail.com";

		// Sender's email ID needs to be mentioned
		String from = "prabin@gmail.com";

		// Assuming you are sending email from localhost
		String host = "localhost";

		// Get system properties
		Properties properties = System.getProperties();

		// Setup mail server
		properties.setProperty("mail.smtp.host", host);



		// Get the default Session object.
		Session session = Session.getDefaultInstance(properties);

		// Create a default MimeMessage object.
		MimeMessage message = new MimeMessage(session);

		// Set From: header field of the header.
		message.setFrom(new InternetAddress(from));

		// Set To: header field of the header.
		message.addRecipient(Message.RecipientType.TO, new InternetAddress(to));

		// Set Subject: header field
		message.setSubject("This is the Subject Line!");

		// Now set the actual message
		message.setText("This is actual message");

		// Send message
		Transport.send(message);
		System.out.println("Sent message successfully....");
	}
}
```