Chapter- 7
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);
		}
	}
}
```

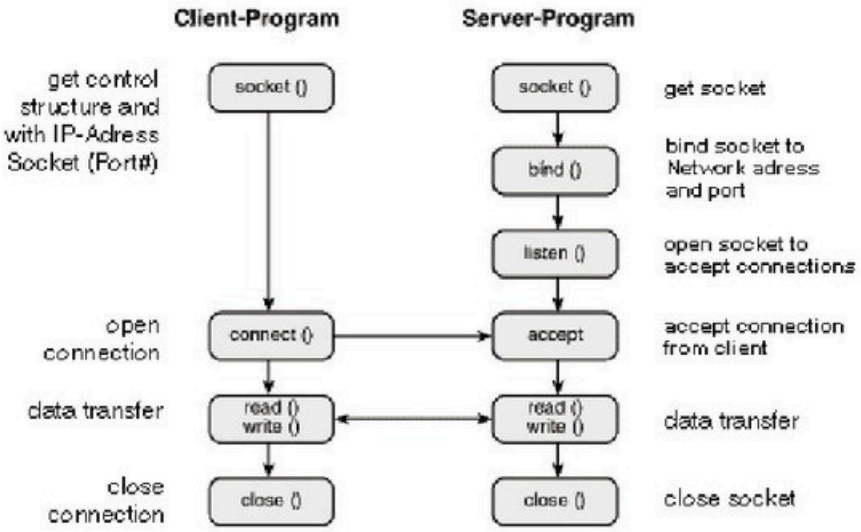
Ouput:

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();
                    }
            }
    }
```

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

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

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();
            }
        }
    }
```

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.

```
1  import java.net.*;
2
3  public class client {
4      public static void main(String[] args) throws Exception {
5          // Create a DatagramSocket
6          DatagramSocket socket = new DatagramSocket();
7
8          // Buffers for sending and receiving data
9          byte[] sendData = new byte[200];
10         byte[] receiveData = new byte[200];
11
12         // Define the server address and port
13         InetAddress serverAddress = InetAddress.getByName("localhost");
14         int serverPort = 5020;
15
16         // Create and send a DatagramPacket to the server
17         DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, serverAddress, serverPor
18 t);       socket.send(sendPacket);
19
20         // Receive a response from the server
21         DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
22         socket.receive(receivePacket);
23
24         // Extract and print the received message
25         String message = new String(receivePacket.getData(), 0, receivePacket.getLength());
26         System.out.println("Received Message: " + message);
27
28         // Close the socket
29         socket.close();
30     }
31 }
```

```java
import java.net.*;

public class Server {
    public static void main(String[] args) throws Exception {
        // Create a DatagramSocket and bind it to port 5020
        DatagramSocket socket = new DatagramSocket(5020);

        // Buffers for sending and receiving data
        byte[] sendData = new byte[200];
        byte[] receiveData = new byte[200];

        System.out.println("Server is running...");

        while (true) {
            // Receive a DatagramPacket from the client
            DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
            socket.receive(receivePacket);

            // Extract the client's address and port
            InetAddress clientAddress = receivePacket.getAddress();
            int clientPort = receivePacket.getPort();

            // Prepare the response message
            String message = "Welcome Client";
            sendData = message.getBytes();

            // Create and send a DatagramPacket to the client
            DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, clientAddress, clientPort);
            socket.send(sendPacket);

            // Print a message indicating the response has been sent
            System.out.println("Sent response to client at " + clientAddress + ":" + clientPort);
        }
    }
}
```

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.

Clientside :

```java
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.Scanner;

public class CircleAreaClient {
    public static void main(String[] args) {
        try {
            DatagramSocket clientSocket = new DatagramSocket();
            InetAddress serverAddress = InetAddress.getByName("localhost");

            Scanner scanner = new Scanner(System.in);
            System.out.print("Enter the radius of the circle: ");
            double radius = scanner.nextDouble();
            String radiusStr = String.valueOf(radius);

            byte[] sendBuffer = radiusStr.getBytes();
            byte[] receiveBuffer = new byte[1024];

            // Send radius to server
            DatagramPacket sendPacket = new DatagramPacket(sendBuffer, sendBuffer.length, serverAddress, 9876);
            clientSocket.send(sendPacket);

            // Receive area from server
            DatagramPacket receivePacket = new DatagramPacket(receiveBuffer, receiveBuffer.length);
            clientSocket.receive(receivePacket);

            String areaStr = new String(receivePacket.getData(), 0, receivePacket.getLength());
            System.out.println("The area of the circle is: " + areaStr);

            clientSocket.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Serverside :

```java
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

public class CircleAreaServer {
    public static void main(String[] args) {
        try {
            DatagramSocket serverSocket = new DatagramSocket(9876);
            System.out.println("Server is running and waiting for a radius...");

            byte[] receiveBuffer = new byte[1024];
            byte[] sendBuffer;

            while (true) {
                // Receive radius from client
                DatagramPacket receivePacket = new DatagramPacket(receiveBuffer, receiveBuffer.length);
                serverSocket.receive(receivePacket);

                String radiusStr = new String(receivePacket.getData(), 0, receivePacket.getLength());
                double radius = Double.parseDouble(radiusStr);

                // Calculate area of the circle
                double area = Math.PI * radius * radius;
                String areaStr = String.format("%.2f", area);

                // Send area back to client
                InetAddress clientAddress = receivePacket.getAddress();
                int clientPort = receivePacket.getPort();
                sendBuffer = areaStr.getBytes();

                DatagramPacket sendPacket = new DatagramPacket(sendBuffer, sendBuffer.length, clientAddress, clientPort);
                serverSocket.send(sendPacket);

                System.out.println("Processed radius: " + radius + ", Sent area: " + areaStr);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```