CHAPTER 1

[JAVA DATA COMMUNICATION & NETWORKING]

[SEN2321]

PRINCIPLES OF PROGRAMMING II

SULEIMAN HAYAT TUGE

CHAPTER DURATION 2 WEEKS

# WEEK 1

## COMPUTER NETWORK

A computer network is a group of computers linked to each other that enables the computer to communicate with another computer and share their resources, data, and applications.

## DATA COMMUNICATION

Data communications refers to the transmission of digital data between two or more computers on a computer network or data network is a telecommunications network that allows computers to exchange data. The physical connection between networked computing devices is established using either cable media or wireless media. The best-known computer network is the Internet.

## IP ADDRESSES

An IP address is the identifier that enables your device to send or receive data packets across the internet. It holds information related to your location and therefore making devices available for two-way communication. The internet requires a process to distinguish between different networks, routers, and websites. Therefore, IP addresses provide the mechanism of doing so, and it forms an indispensable part in the working of the internet. You will notice that most of the IP addresses are essentially numerical. Still, as the world is witnessing a colossal growth of network users, the network developers had to add letters and some addresses as internet usage grows.

## PORT ADDRESS

The TCP port is a unique number assigned to different applications. For example, we have opened the email and games applications on our computer; through email application, we want to send the mail to the host, and through games application, we want to play the online games. In order to do all these tasks, different unique numbers are assigned to these applications. Each protocol and address have a port known as a port number. The TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) protocols mainly use the port numbers.

A port number is a unique identifier used with an IP address. A port is a 16-bit unsigned integer, and the total number of ports available in the TCP/IP model is 65,535 ports. Therefore, the range of port numbers is 0 to 65535. In the case of TCP, the zero-port number is reserved and cannot be used, whereas, in UDP, the zero port is not available. IANA (Internet Assigned Numbers Authority) is a standard body that assigns the port numbers. The port numbers are divided into three categories:

* Well-known ports from 0 to 1023
* Registered ports from 1024 to 49151
* Dynamic ports from 49152 to 65535

At your own time you need to read about the OSI Model and TCP/IP Model.

## NETWORK PROTOCOLS

* TCP
* FTP
* UDP
* Telnet
* SMTP
* POP

## 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. The java.net package supports two protocols:

* **TCP:** Transmission Control Protocol provides reliable communication between the sender and receiver. TCP is used along with the Internet Protocol referred as TCP/IP.
* **UDP:** User Datagram Protocol provides a connection-less protocol service by allowing packet of data to be transferred along two or more nodes.

## 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. The client in socket programming must know two information IP Address of Server, and Port number.

Here, we are going to make one-way client and server communication. In this application, client sends a message to the server, server reads the message and prints it. Here, two classes are being used: Socket and ServerSocket. The Socket class is used to communicate client and server. Through this class, we can read and write message. The ServerSocket class is used at server-side. The accept() method of ServerSocket class blocks the console until the client is connected. After the successful connection of client, it returns the instance of Socket at server-side.

**Sample Code:**

Receiver or Server

package datacommunication;

import java.io.\*;

import java.net.\*;

/\*\*

\*

\* **@author** HSMLTUGE

\*/

public class **DataCommunication** {

public static void ***main***(String[] args) {

try {

ServerSocket serverSocket = new ServerSocket(6666);

Socket socket = serverSocket.accept();

//establishes connection

DataInputStream dis = new DataInputStream(socket.getInputStream());

String str = (String) dis.readUTF();

System.*out*.println("message= " + str);

serverSocket.close();

} catch (Exception e) {

System.*out*.println(e);

}

}

}

Sender or Client

package datacommunication;

import java.io.\*;

import java.net.\*;

/\*\*

\*

\* **@author** HSMLTUGE

\*/

public class **Client** {

public static void ***main***(String[] args) {

try {

Socket socket = new Socket("localhost", 6666);

DataOutputStream dout = new DataOutputStream(socket.getOutputStream());

dout.writeUTF("Hello Server");

dout.flush();

dout.close();

s.close();

} catch (Exception e) {

System.*out*.println(e);

}

}

}

To execute the above program the student will have to run the Receiver or Server first, that way when the Sender or Client try to connect it will get acknowledged by the Receiver or Server. The output of the above code after the Receiver is executed first and Sender is executed second, the program will Receiver and show:

> message = Hello Server

The reason is because the system sees localhost as itself thereby it doesn’t require any cable or wireless connection to connect to itself. A computer trying to communicate to itself can use either **localhost** or **127.0.0.1.**

## JAVA SOCKET PROGRAMMING (TWO WAY COMMUNICATION)

In real world scenario communication between server and client is always a two-way communication, in the above example we dealt with a one-way communication between the Sender and Receiver. In the example below we will write a console java chat application that allows users to communicate over a network.

Task:

Create a java application that allows two users to enter their names, the IP or PORT addresses of the other’s computer and transfer messages between each other. The application should say the name of the sender and message sent:

We start by analyzing the application requirement, The application consist of two users, 1 is sender and the second is the receiver.

Receiver’s requirement consists of:

* To be able to enter names,
* To enter a port number that will be opened for the communication,
* To be able to write message,
* To be able to read response

Sender’s requirement consists of:

* To be able to enter names,
* To enter IP address of the receiver.
* To enter a port number that will be opened for the communication,
* To be able to write message
* To be able to read response

**Receiver Sample Code:**

package datacommunication;

import java.io.\*;

import java.net.\*;

import java.util.Scanner;

/\*\*

\*

\* @author HSMLTUGE

\*/

public class DataCommunication {

//this is the receiver

//notice we are throwing exception from the main function

//this way we do not need to wrap the code in a try and catch block

public static void main(String[] args) throws IOException {

//we instantiate scanner object

Scanner scan = new Scanner(System.in);

//we ask the name of the receiver

System.out.println("Enter your name:");

//we receive the name of the receiver

String name = scan.nextLine();

// we ask for the port to be used for the communication

System.out.println("Enter port number you wish to communicate on:");

// we receive the port number to be used

int port = scan.nextInt();

// we inform the receiver to wait for the sender to setup his side of the system

System.out.println("Waiting for communication...");

// we open the port to listen for requests

ServerSocket serverSocket = new ServerSocket(port);

// we accept any request coming from the port

Socket socket = serverSocket.accept();

//we instantiate data input stream to be used to receive input from the sender

DataInputStream din = new DataInputStream(socket.getInputStream());

// we instantiate data output stream to send data to the sender

DataOutputStream dout = new DataOutputStream(socket.getOutputStream());

// we instantiate buffered reader to receive input from the receiver

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

// we tell the receiver if connection successful and to start communicating

System.out.println("Start Communicating, connection successful");

// we declare two string variables,

//str we use to read message from the sender and display it,

//while str2 we receive our input to be sent to the sender

String str = "", str2 = "";

//we start a loop that check if the input of the user is not equal to stop

// in the event the input is stop this end the chat application and no data can be transferred any longer

while (!str.equals("stop")) {

//we read input from the sender

str = din.readUTF();

// we display input from the sender

System.out.println(str);

// we display message for receiver to write his/her own message

System.out.println(name + " writes:");

//we read what the receiver what to send to the sender

str2 = br.readLine();

//we now send the message saying receiver says

dout.writeUTF(name + " says: " + str2);

// Flushes the output stream and forces any buffered output bytes to be written out.

dout.flush();

}

//we close the data input stream

din.close();

//we close the socket

socket.close();

//we close the server socket

serverSocket.close();

}

}

**Sender Sample Code:**

package datacommunication;

import java.io.\*;

import java.net.\*;

import java.util.Scanner;

/\*\*

\*

\* @author HSMLTUGE

\*/

public class ClientApplication {

//this is the sender

//notice we are throwing exception from the main function

//this way we do not need to wrap the code in a try and catch block

public static void main(String[] args) throws IOException {

//we instantiate scanner object

Scanner scan = new Scanner(System.in);

//we ask the name of the sender

System.out.println("Enter your name:");

//we receive the name of the sender

String name = scan.nextLine();

// we ask for the ip address to be used for the communication

System.out.println("Enter the other computer's IP Address:");

// we receive the ip address to be used

String ip = scan.nextLine();

// we ask for the port to be used for the communication

System.out.println("Enter port number you wish to communicate on:");

// we receive the port number to be used

int port = scan.nextInt();

// we inform the receiver to wait for the sender to setup his side of the system

System.out.println("Waiting for communication...");

// we try to establish a connection

Socket socket = new Socket(ip, port);

//we instantiate data input stream to be used to receive input from the receiver

DataInputStream din = new DataInputStream(socket.getInputStream());

// we instantiate data output stream to send data to the receiver

DataOutputStream dout = new DataOutputStream(socket.getOutputStream());

// we instantiate buffered reader to receive input from the sender

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

// we tell the receiver if connection successful and to start communicating

System.out.println("Start Communicating, connection successful");

// we declare two string variables,

//str we use to read message from the receiver and display it,

//while str2 we receive our input to be sent to the receiver

String str = "", str2 = "";

//we start a loop that check if the input of the user is not equal to stop

// in the event the input is stop this end the chat application and no data can be transferred any longer

while (!str.equals("stop")) {

// we display message for sender to write his/her message

System.out.println(name + " write:");

//we read what the send what to send to the receiver

str = br.readLine();

//we now send the message saying sender says

dout.writeUTF(name + " says: " + str);

// Flushes the output stream and forces any buffered output bytes to be written out.

dout.flush();

//we read input from the receiver

str2 = din.readUTF();

// we display input from the receiver

System.out.println(str2);

}

//we close the data output stream

dout.close();

//we close the socket

socket.close();

}

}

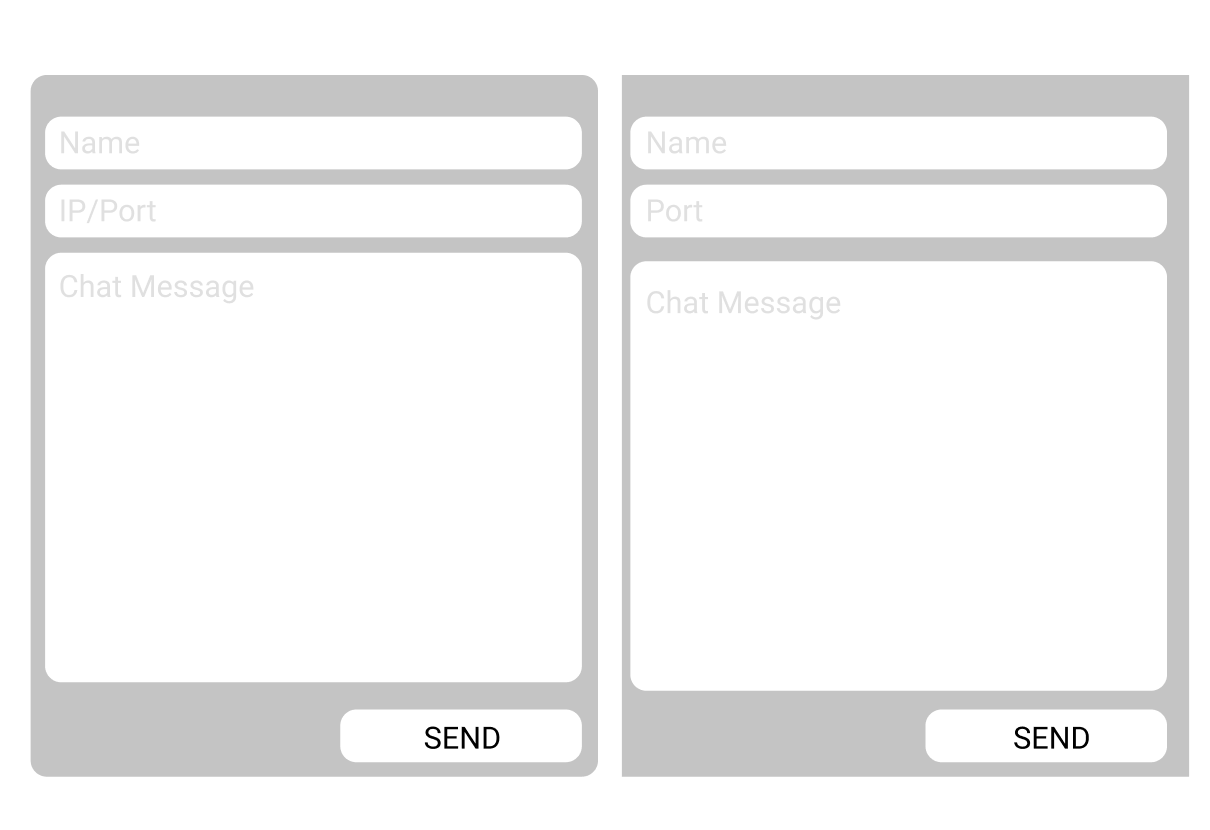
**Sample Code Output:**

|  |  |
| --- | --- |
| Receiver Output | Sender Output |
| run:  Enter your name:  Hayat  Enter port number you wish to communicate on:  9800  Waiting for communication...  Start Communicating, connection successful  Solo says: Hello Hayat  Hayat writes:  Hi Solo  Solo says: How are you Hayat  Hayat writes: | run:  Enter your name:  Solo  Enter the other computer's IP Address:  localhost  Enter port number you wish to communicate on:  9800  Waiting for communication...  Start Communicating, connection successful  Solo write:  Hello Hayat  Hayat says: Hi Solo  Solo write:  How are you Hayat |

# WEEK 2

## DESIGNING THE CHAT GUI

What is missing in this diagram?



## INTEGRATING THE GUI TO THE CHAT APPLICATION

## INTRODUCTION TO DATABASE CONNECTION