LIST OF ASSIGNMENTS

SAVITRIBAI PHULE PUNE UNIVERSITY
सावित्रीबाई फुले पुणे विद्यापीठ

Sr. No.	Assignments	Mapping	
		CO	PO
1	To develop any distributed application through implementing client-server communication programs based on **Java Sockets and RMI** techniques.	CO-1	PO-1, PO-3, PO-5
2	To develop any distributed application using Message Passing Interface **(MPI)**.	CO-1	PO-1, PO-3, PO-5
3	To develop any distributed application with **CORBA program using JAVA IDL**.	CO-2	PO-2, PO-3 PO-5
4	To develop any distributed algorithm for **leader election**.	CO-1	PO-1, PO-3
5	To create a simple **web service** and write any distributed application to consume the web service.	CO-2, CO-3	PO-2,3 PO-5
6	To develop any distributed application using Messaging System in **Publish - Subscribe paradigm**.	CO-2, CO-3	PO-2,3 PO-5
7	To develop **Microservices** framework based distributed application.	CO-2, CO-3	PO-2,3 PO-5

ASSIGNMENT NO. 1	
Assignment No. 1 A)	To develop any distributed application through implementing client-server communication programs based on **TCP /UDP Java Sockets**
Objective(s):	By the end of this assignment, the student will be able to implement any distributed multi-threaded client-server programs using Java sockets.
Assignment No. 1 B)	To develop any distributed application through implementing client-server communication programs based on **Java RMI.**
Objective(s):	By the end of this assignment, the student will be able to implement any distributed applications based on RMI.
Tools	Java Programming Environment, jdk 1.8, rmiregistry.

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.
- **DatagramSocket** and **DatagramPacket** classes are used for connection-less socket programming.
- **Socket class and methods:**
  - A socket is simply an endpoint for communications between the machines. The Socket class can be used to create a socket.
  - public InputStream getInputStream(): returns the InputStream attached with this socket.
  - public OutputStream getOutputStream():returns the OutputStream attached with this socket.
  - public synchronized void close(): closes this socket

JAVA API FOR STREAM COMMUNICATION

- **ServerSocket class and methods:**
  - The ServerSocket class can be used to create a server socket. This object is used to establish communication with the clients.
  - public Socket accept(): returns the socket and establish a connection between server and client.
  - public synchronized void close(): closes the server socket.
- **Java.net.DatagramSocket class :**
  - Every packet sent from a datagram socket is individually routed and delivered.
  - It can also be used for sending and receiving broadcast messages.
  - Datagram Sockets is the java's mechanism for providing network communication via UDP instead of TCP.
  - **DatagramSocket() :** Creates a datagramSocket and binds it to any available port on local machine. If this constructor is used, the OS would assign any port to this socket.

JAVA API FOR DATAGRAM COMMUNICATION

- **DatagramPacket** : A datagram is an independent, self-contained message sent over the network whose arrival, arrival time, and content are not guaranteed. In Java, `DatagramPacket` represents a datagram.
- You can create a DatagramPacket object by using one of the following constructors:
- `DatagramPacket(byte[] buf, int length)`
- `DatagramPacket(byte[] buf, int length, InetAddress address, int port)`
- The data must be in the form of an array of bytes.
- The first constructor is used to create a `DatagramPacket` to be received.
- The second constructor creates a `DatagramPacket` to be sent, so you need to specify the address and port number of the destination host.
- The parameter length specifies the amount of data in the byte array to be used, usually is the length of the array (`buf.length`).

JAVA API FOR DATAGRAM COMMUNICATION

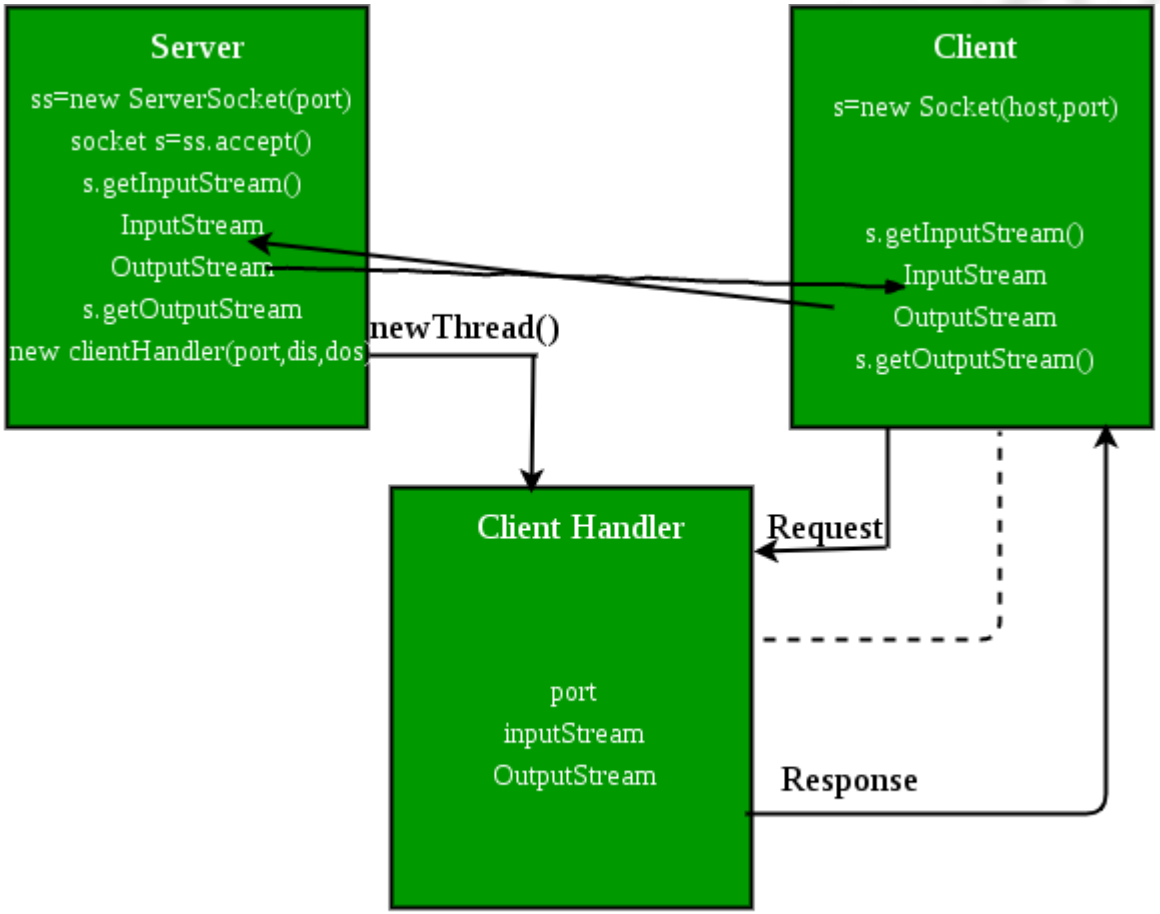
- **DatagramSocket :** `DatagramSocket` is used to send and receive `DatagramPackets`.
- In Java, `DatagramSocket` is used for both client and server. There are no separate classes for client and server like TCP sockets.
- `DatagramSocket` creates object to establish a UDP connection for sending and receiving datagram, by using the following constructors:
- `DatagramSocket(int port, InetAddress laddr)`**:** This constructor binds the server to the specified IP address (in case the computer has multiple IP addresses).
- The key methods of the `DatagramSocket` include:
- **`send`**`(DatagramPacket p)`**:** sends a datagram packet.
- **`receive`**`(DatagramPacket p)`: receives a datagram packet.
- **`close`**`()`**:** closes the socket.

JAVA TCP SOCKET PROGRAMMING

- Create two java files, **Server.java** and **Client.java**.
- Server file contains two classes namely: **Sever** (public class for creating server) and **ClientHandler** (for handling any client using multithreading).
- Client file contain only one public class **Client** (for creating a client).
- Java API networking package (java.net) to be imported which takes care of all network programming.

QUICK OVERVIEW

- The diagram shows how these three classes interact with each other.



HOW THESE PROGRAM WORKS TOGETHER?

- When a client, say client1 sends a request to connect to server, the server assigns a new thread to handle this request.
- The newly assigned thread is given the access to streams for communicating with the client.
- After assigning the new thread, the server via its while loop, again comes into accepting state.
- When a second request comes while first is still in process, the server accepts this requests and again assigns a new thread for processing it. In this way, multiple requests can be handled even when some requests are in process.

SERVER SIDE PROGRAMMING (Server.java)

- **Server class :** The steps involved on server side:

1. **Establishing the Connection:** Server socket object is initialized and inside a while loop a socket object continuously accepts incoming connection.
2. **Obtaining the Streams:** The inputstream object and outputstream object is extracted from the current requests' socket object. Streams reads data (numbers) instead of just bytes.
3. **Creating a handler object:** After obtaining the streams and port number, a new clientHandler object is created with these parameters.
4. **Invoking the start() method :** The start() method is invoked on this newly created thread object.

SERVER SIDE PROGRAMMING (Server.java)

```
// Java implementation of Server side
// It contains two classes : Server and ClientHandler
// Save file as Server.java

import java.io.*;
import java.text.*;
import java.util.*;
import java.net.*;

// Server class
public class Server
{
    public static void main(String[] args) throws IOException
    {
        // server is listening on port 5056
        ServerSocket ss = new ServerSocket(5056);

        // running infinite loop for getting
        // client request
        while (true)
        {
            Socket s = null;

            try
            {
                // socket object to receive incoming client requests
                s = ss.accept();

                System.out.println("A new client is connected : " + s);
```

SERVER SIDE PROGRAMMING (Server.java)

```
// ClientHandler class
class ClientHandler extends Thread
{
    DateFormat fordate = new SimpleDateFormat("yyyy/MM/dd");
    DateFormat fortime = new SimpleDateFormat("hh:mm:ss");
    final DataInputStream dis;
    final DataOutputStream dos;
    final Socket s;


    // Constructor
    public ClientHandler(Socket s, DataInputStream dis, DataOutputStream dos)
    {
        this.s = s;
        this.dis = dis;
        this.dos = dos;
    }

    @Override
    public void run()
    {
        String received;
        String toreturn;
        while (true)
        {
            try {

                // Ask user what he wants
                dos.writeUTF("What do you want?[Date | Time]..\n"+
                             "Type Exit to terminate connection.");

                // receive the answer from client
                received = dis.readUTF();
```

SERVER SIDE PROGRAMMING (Server.java)

```
            if(received.equals("Exit"))
            {
                System.out.println("Client " + this.s + " sends exit...");
                System.out.println("Closing this connection.");
                this.s.close();
                System.out.println("Connection closed");
                break;
            }

            // creating Date object
            Date date = new Date();

            // write on output stream based on the
            // answer from the client
            switch (received) {

                case "Date" :
                    toreturn = fordate.format(date);
                    dos.writeUTF(toreturn);
                    break;

                case "Time" :
                    toreturn = fortime.format(date);
                    dos.writeUTF(toreturn);
                    break;

                default:
                    dos.writeUTF("Invalid input");
                    break;
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

```
                try
                {
                    // closing resources
                    this.dis.close();
                    this.dos.close();

                }catch(IOException e){
                    e.printStackTrace();
                }
            }
        }
```

CLIENT SIDE PROGRAMMING (Client.java)

- Client side programming is similar as in general socket programming program with the following steps-

1. Establish a Socket Connection **2.** Communication **3.** Close the connection

```
import java.io.*;
import java.net.*;
import java.util.Scanner;

// Client class
public class Client
{
    public static void main(String[] args) throws IOException
    {
        try
        {
            Scanner scn = new Scanner(System.in);

            // getting localhost ip
            InetAddress ip = InetAddress.getByName("localhost");

            // establish the connection with server port 5056
            Socket s = new Socket(ip, 5056);

            // obtaining input and out streams
            DataInputStream dis = new DataInputStream(s.getInputStream());
            DataOutputStream dos = new DataOutputStream(s.getOutputStream());
```

CLIENT SIDE PROGRAMMING (Client.java)

```
            // the following loop performs the exchange of
            // information between client and client handler
            while (true)
            {
                System.out.println(dis.readUTF());
                String tosend = scn.nextLine();
                dos.writeUTF(tosend);

                // If client sends exit,close this connection
                // and then break from the while loop
                if(tosend.equals("Exit"))
                {
                    System.out.println("Closing this connection : " + s);
                    s.close();
                    System.out.println("Connection closed");
                    break;
                }

                // printing date or time as requested by client
                String received = dis.readUTF();
                System.out.println(received);
            }

            // closing resources
            scn.close();
            dis.close();
            dos.close();
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

STEPS FOR COMPILATION AND EXECUTION

- **If you're using Eclipse :**
- Compile both of them on two different terminals or tabs
- First run the Server.java followed by the Client.java.
- Run multiple instances from the same program
- Type messages in the Client Window which will be received and showed by the Server Window simultaneously.
- Type 'Exit' to end.

EXPECTED OUTPUT

```
What do you want?[Date | Time]..
Type Exit to terminate connection.
Date
2019/01/17
What do you want?[Date | Time]..
Type Exit to terminate connection.
Time
05:35:28
What do you want?[Date | Time]..
Type Exit to terminate connection.
Over
Invalid input
What do you want?[Date | Time]..
Type Exit to terminate connection.
Exit Closing this connection :
  Socket[addr=localhost/127.0.0.1,port=5056,localport=605
  36] Connection closed
```

JAVA UDP CLIENT PROGRAM

- Write a code for a client program that requests for quotes from a server that implements the Quote of the Day (QOTD) service .
- The code of the full client program that parameterizes the hostname and port number, handles exceptions and gets a quote from the server for every 10 seconds:

```
import java.io.*;
import java.net.*;

/**
 * This program demonstrates how to implement a UDP client program.
 *
 *
 * @author www.codejava.net
 */
public class QuoteClient {

    public static void main(String[] args) {
        if (args.length < 2) {
            System.out.println("Syntax: QuoteClient <hostname> <port>");
            return;
        }

        String hostname = args[0];
        int port = Integer.parseInt(args[1]);
```

JAVA UDP CLIENT PROGRAM

- Write a code for a client program that requests for quotes from a server that implements the Quote of the Day (QOTD) service .

```
        try {
            InetAddress address = InetAddress.getByName(hostname);
            DatagramSocket socket = new DatagramSocket();

            while (true) {

                DatagramPacket request = new DatagramPacket(new byte[1], 1, address, port);
                socket.send(request);

                byte[] buffer = new byte[512];
                DatagramPacket response = new DatagramPacket(buffer, buffer.length);
                socket.receive(response);

                String quote = new String(buffer, 0, response.getLength());

                System.out.println(quote);
                System.out.println();

                Thread.sleep(10000);
            }

        } catch (SocketTimeoutException ex) {
            System.out.println("Timeout error: " + ex.getMessage());
            ex.printStackTrace();
        } catch (IOException ex) {
            System.out.println("Client error: " + ex.getMessage());
            ex.printStackTrace();
        } catch (InterruptedException ex) {
            ex.printStackTrace();
        }
    }
}
```

JAVA UDP SERVER PROGRAM

- The sample program demonstrates how to implement a server for the above client. The code creates a UDP server listening on port 17 and waiting for client's request:

```
import java.io.*;
import java.net.*;
import java.util.*;

/**
 * This program demonstrates how to implement a UDP server program.
 *
 *
 * @author www.codejava.net
 */
public class QuoteServer {
    private DatagramSocket socket;
    private List<String> listQuotes = new ArrayList<String>();
    private Random random;

    public QuoteServer(int port) throws SocketException {
        socket = new DatagramSocket(port);
        random = new Random();
    }

    public static void main(String[] args) {
        if (args.length < 2) {
            System.out.println("Syntax: QuoteServer <file> <port>");
            return;
        }

        String quoteFile = args[0];
        int port = Integer.parseInt(args[1]);
```

JAVA UDP SERVER PROGRAM

- The sample program demonstrates how to implement a server for the above client. The code creates a UDP server listening on port 17 and waiting for client's request:

```
    try {
        QuoteServer server = new QuoteServer(port);
        server.loadQuotesFromFile(quoteFile);
        server.service();
    } catch (SocketException ex) {
        System.out.println("Socket error: " + ex.getMessage());
    } catch (IOException ex) {
        System.out.println("I/O error: " + ex.getMessage());
    }
}

private void service() throws IOException {
    while (true) {
        DatagramPacket request = new DatagramPacket(new byte[1], 1);
        socket.receive(request);

        String quote = getRandomQuote();
        byte[] buffer = quote.getBytes();

        InetAddress clientAddress = request.getAddress();
        int clientPort = request.getPort();

        DatagramPacket response = new DatagramPacket(buffer, buffer.length, clientAddress, clientPort);
        socket.send(response);
    }
}
```

JAVA UDP SERVER PROGRAM

- The sample program demonstrates how to implement a server for the above client. The code creates a UDP server listening on port 17 and waiting for client's request:

```
    private void loadQuotesFromFile(String quoteFile) throws IOException {
        BufferedReader reader = new BufferedReader(new FileReader(quoteFile));
        String aQuote;

        while ((aQuote = reader.readLine()) != null) {
            listQuotes.add(aQuote);
        }

        reader.close();
    }

    private String getRandomQuote() {
        int randomIndex = random.nextInt(listQuotes.size());
        String randomQuote = listQuotes.get(randomIndex);
        return randomQuote;
    }
}
```

- Suppose we have a Quotes.txt file with the following content (each quote is in a single line):

```
1  Whether you think you can or you think you can't, you're right - Henry Ford
2  There are no traffic jams along the extra mile - Roger Staubach
3  Build your own dreams, or someone else will hire you to build theirs - Farrah Gray
4  What you do today can improve all your tomorrows - Ralph Marston
5  Remember that not getting what you want is sometimes a wonderful stroke of luck - Dalai Lama
```

COMPILE AND RUN

- Type the following command to run the server program:

```
java QuoteServer Quotes.txt 17
```

- And run the client program (on the same computer):

```
java QuoteClient localhost 17
```

CONCLUSION

- Thus students have learnt how to develop a client/server distributed application relying on TCP protocol. Based on this knowledge, they are able to develop client programs that communicate with servers via TCP, and developing their own TCP client/server applications.

- Thus students have learnt how to develop a client/server distributed application relying on UDP protocol. Based on this knowledge, they are able to develop client programs that communicate with servers via UDP, and developing their own UDP client/server applications.

CONCLUSION

- Thus students have learnt how to develop a client/server distributed application relying on TCP protocol. Based on this knowledge, they are able to develop client programs that communicate with servers via TCP, and developing their own TCP client/server applications.

- Thus students have learnt how to develop a client/server distributed application relying on UDP protocol. Based on this knowledge, they are able to develop client programs that communicate with servers via UDP, and developing their own UDP client/server applications.

REFERENCES

- Java UDP Client Server Program Example: https://www.codejava.net/java-se/networking/java-udp-client-server-program-example.
- DatagramPacket Javadoc: https://docs.oracle.com/javase/8/docs/api/java/net/DatagramPacket.html
- DatagramSocket Javadoc: https://docs.oracle.com/javase/8/docs/api/java/net/DatagramSocket.html
- Socket Programming in Java: https://www.geeksforgeeks.org/socket-programming-in-java/