**Name:** Arzanish Sarwar

**ID:**  R00170508

**Group:**  SDH3-B

Q4:

Server Class Code:

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

// **TODO** Auto-generated method stub

ServerSocket server = **new** ServerSocket(3001);

Socket client = server.accept();

System.***out***.println("Welcome !!");

InputStream inputStream = client.getInputStream();

// create a DataInputStream so we can read data from it.

DataInputStream dataInputStream = **new** DataInputStream(inputStream);

OutputStream outputStream = client.getOutputStream();

// create a data output stream from the output stream so we can send data through it

DataOutputStream dataOutputStream = **new** DataOutputStream(outputStream);

String number = dataInputStream.readUTF();

String[] numbers = number.split(" ");

IntegerConversion obj;

**switch**(numbers[0]) {

**case** "1":

obj = **new** IntegerConversion(Integer.*valueOf*((numbers[1])), Integer.*valueOf*((numbers[2])));

**int** result1 = obj.sum();

dataOutputStream.writeUTF(String.*valueOf*(result1));

**break**;

**case** "2":

obj = **new** IntegerConversion(Integer.*valueOf*((numbers[1])), Integer.*valueOf*((numbers[2])));

**int** result2 = obj.product();

dataOutputStream.writeUTF(String.*valueOf*(result2));

**break**;

**case** "3":

obj = **new** IntegerConversion(Integer.*valueOf*((numbers[1])), Integer.*valueOf*((numbers[2])));

**int** result3 = obj.power();

dataOutputStream.writeUTF(String.*valueOf*(result3));

**break**;

**default**:

System.***out***.println("Enter in the formet 1 12 13 or 2 13 56 or 3 33 45 !!");

}

}

Client Class Code:

**public** **class** Client {

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

// **TODO** Auto-generated method stub

**try** {

Socket socket = **new** Socket("localhost", 3001);

System.***out***.println("Connected!");

OutputStream outputStream = socket.getOutputStream();

// create a data output stream from the output stream so we can send data through it

DataOutputStream dataOutputStream = **new** DataOutputStream(outputStream);

InputStream inputStream = socket.getInputStream();

// create a DataInputStream so we can read data from it.

DataInputStream dataInputStream = **new** DataInputStream(inputStream);

Scanner s = **new** Scanner(System.***in***);

System.***out***.println("Enter Command in the formet 1 12 13 or 2 13 56 or 3 33 45 !! 1 is for sum, 2 is for product and 3 is for power");

String string = s.nextLine();

dataOutputStream.writeUTF(string);

String ans = dataInputStream.readUTF();

System.***out***.println("Anser is " + ans);

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

IntegerConversion Class code:

**public** **class** IntegerConversion {

**int** num1;

**int** num2;

**public** IntegerConversion(**int** num1 , **int** num2) {

**this**.num1 = num1;

**this**.num2 = num2;

}

**public** **int** sum() {

**return** **this**.num1 + **this**.num2;

}

**public** **int** product() {

**return** **this**.num1\***this**.num2;

}

**public** **int** power() {

**return** (**int**) Math.*pow*(**this**.num1,**this**.num2);

}

}

**B)**

So in Server class we will change the code for handling clients as following

server = **new** ServerSocket(3001);

System.***out***.println("Server Started ....");

**while**(**true**){

Socket serverClient = server.accept();

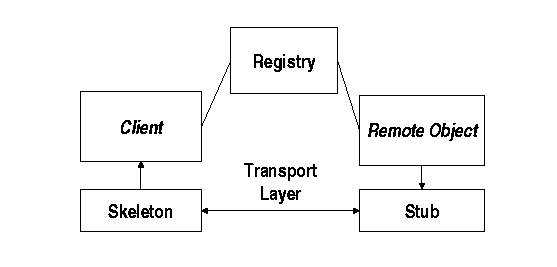
ServerClientThread sct = **new** ServerClientThread(serverClient); //send the request to a separate thread

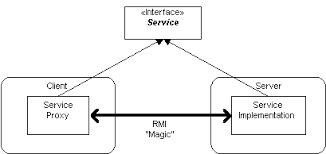
sct.start();

}

As you can see in above code, each time client is connected to server it makes a child thread and let that thread deal with the socket, While server.acccept() is free and listening for new clients and in this way it can handle multiple clients.

**Q3**





**RMI interface :**

**import** java.rmi.Remote;

**import** java.rmi.RemoteException;

**import** java.util.ArrayList;

**public** **interface** RMiinterface **extends** Remote {

**public** ArrayList<Integer> connectedClient() **throws** RemoteException;

}

**Object implementation:**

**import** java.rmi.RemoteException;

**import** java.rmi.server.UnicastRemoteObject;

**import** java.util.ArrayList;

**public** **class** RMIserver **implements** RMiinterface{

**protected** RMIserver() **throws** RemoteException {

UnicastRemoteObject.*exportObject*(**this**,0);

}

@Override

**public** ArrayList<Integer> connectedClient() **throws** RemoteException {

ArrayList<Integer> list = **new** ArrayList<Integer>();

**for**(**int** i = 10;i<0;i--) {

list.add(i);

}

**return** list;

}

}

**Server Class:**

**import** java.rmi.Remote;

**import** java.rmi.RemoteException;

**import** java.rmi.registry.LocateRegistry;

**import** java.rmi.registry.Registry;

**public** **class** Main **implements** Remote{

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

RMiinterface rmiServer = **new** RMIserver();

Registry registry = LocateRegistry.createRegistry(1099);

registry.rebind("RMIserver",rmiServer);

}

}

**Client Class:**

**import** java.io.DataInputStream;

**import** java.io.DataOutputStream;

**import** java.io.IOException;

**import** java.io.InputStream;

**import** java.io.OutputStream;

**import** java.net.MalformedURLException;

**import** java.net.Socket;

**import** java.rmi.Naming;

**import** java.rmi.NotBoundException;

**import** java.rmi.RemoteException;

**import** java.rmi.registry.LocateRegistry;

**import** java.rmi.registry.Registry;

**import** java.util.ArrayList;

**import** java.util.Scanner;

**public** **class** Client {

**private** **static** ArrayList<Integer> *list*;

**public** **static** **void** main(String[] args) **throws** RemoteException, MalformedURLException, NotBoundException {

Registry registry = LocateRegistry.*getRegistry*("localhost",1099);

RMiinterface rmi = (RMiinterface) Naming.*lookup*("RMIserver");

System.***out***.println("Client is Connected !! ");

*list* = rmi.connectedClient();

**for**(**int** i=0;i<*list*.size();i++) {

System.***out***.print(*list*.get(i)+" , ");

}

}

}

**d) What changes are needed so that several connected clients get their own unique set of numbers (all the numbers 10 ...1).**

In client class we will make a new list so that each client receiving the list can make its own deep copy in that way, each client will have their own unique set of numbers.

**public** **class** Client {

**private** **static** ArrayList<Integer> *lis = new* ArrayList<Integer>(); //this is the change

**public** **static** **void** main(String[] args) **throws** RemoteException, MalformedURLException, NotBoundException {

Registry registry = LocateRegistry.*getRegistry*("localhost",1099);

RMiinterface rmi = (RMiinterface) Naming.*lookup*("RMIserver");

System.***out***.println("Client is Connected !! ");

*list* = rmi.connectedClient();

**for**(**int** i=0;i<*list*.size();i++) {

System.***out***.print(*list*.get(i)+" , ");

}

}

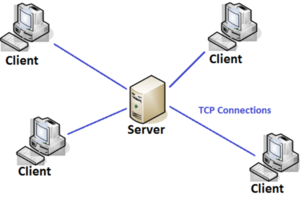
}

Q1

1. Draw a diagram of and explain in your own words what is meant by

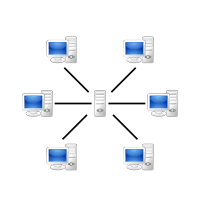
**Client-Server:**

Client-Server model is a distributed application structure, in this structure, Server provides resources and service and client connect to server and utilize its services and resources. In this model, only Server can share its resources. If server is down, whole system gets down



**Peer-to-Peer:**

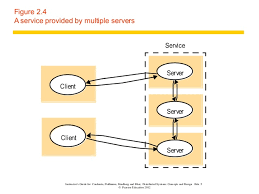
In Peer-to-Peer, each computer shares its resources and connects to each host/device in network, unlike client-server model, each computer or device has equal status and if one computer is down, we can use other host can be used to communicate.



**Services provided by multiple servers:**

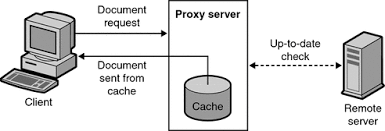
Multiple-server architecture: cluster used for highly scalable web services, e.g., search engines, online stores.

Client can connect to any server in the cluster and use the resources. In this mode, server cluster can handle many clients with efficient and fast response.



**Proxy servers and caches:**

Proxy caching allows a server to act as an intermediary between a user and a provider of web content. When a user accesses a website, proxies interpret and respond to requests on behalf of the original server.



**Mobile code and mobile agents:**

 a **mobile agent** is a composition of computer software and data that is able to migrate (move) from one computer to another autonomously and continue its execution on the destination computer.

**Mobile code** is any program, application, or content capable of movement while embedded in an email, document or website. **Mobile code** uses network or storage media, such as a Universal Serial Bus (USB) flash drive, to execute local **code** execution from another computer system

