DSCC Lab Week1

1. Implement echo server using TCP.

Client:

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

public class Client
{
	private Socket socket		= null;
	private DataInputStream input = null;
	private DataOutputStream out	= null;
	public Client(String address, int port)
	{
	try
	{
		socket = new Socket(address, port);
		System.out.println("Connected");
		input = new DataInputStream(System.in);
		out = new DataOutputStream(socket.getOutputStream());
	}
	catch(UnknownHostException u)
	{
		System.out.println(u);
	}
	catch(IOException i)
	{
		System.out.println(i);
	}
	String line = "";
	while (!line.equals("Over"))
	{
		try
		{
			line = input.readLine();
			out.writeUTF(line);
		}
		catch(IOException i)
		{
			System.out.println(i);
		}
	}
	try
	{
		input.close();
		out.close();
		socket.close();
```

```
		}
		catch(IOException i)
		{
			System.out.println(i);
		}
}

		public static void main(String args[])
		{
			Client client = new Client("127.0.0.1", 5000);
		}
}

Server:
import java.net.*;
import java.io.*;

public class Server
{
private Socket socket = null;
private ServerSocket server = null;
private DataInputStream in	= null;
public Server(int port)
{
		try
		{
		server = new ServerSocket(port);
		System.out.println("Server started");

		System.out.println("Waiting for a client ...");

		socket = server.accept();
		System.out.println("Client accepted");
		in = new DataInputStream(new bufferedInputStream(socket.getInputStream()));

		String line = "";
		while (!line.equals("Over"))
		{
			try
			{
				line = in.readUTF();
				System.out.println(line);

			}
			catch(IOException i)
			{
				System.out.println(i);
			}
```

```
                }
                System.out.println("Closing connection");
                socket.close();
                in.close();
            }
            catch(IOException i)
            {
                System.out.println(i);
            }
        }

        public static void main(String args[])
        {
            Server server = new Server(5000);
        }
        }
```

2. Implement echo server using UDP.

Client:

```
import java.io.*;
import java.net.*;
class UDPClient{
 public static void main(String args[]) throws Exception {
  BufferedReader inFromUser =new BufferedReader(new InputStreamReader(System.in));
  DatagramSocket clientSocket = new DatagramSocket();
  InetAddress IPAddress = InetAddress.getByName("localhost");
  byte[] sendData = new byte[1024];
  byte[] receiveData = new byte[1024];
  String sentence = inFromUser.readLine();
  sendData = sentence.getBytes();
  DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, IPAddress, 9876);
  clientSocket.send(sendPacket);
  DatagramPacket receivePacket = new DatagramPacket(receiveData,receiveData.length);
  clientSocket.receive(receivePacket);
  String modifiedSentence = new String(receivePacket.getData());
  System.out.println("FROM SERVER:" + modifiedSentence);
  clientSocket.close();
 }
}
```

Server:

```
import java.io.*;
import java.net.*;
class UDPServer{
 public static void main(String args[]) throws Exception {
  DatagramSocket serverSocket = new DatagramSocket(9876);
  byte[] receiveData = new byte[1024];
  byte[] sendData = new byte[1024];
  while(true) {
   DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
   serverSocket.receive(receivePacket);
```

```
  String sentence = new String( receivePacket.getData());
  System.out.println("RECEIVED: " + sentence);
  InetAddress IPAddress = receivePacket.getAddress();
  int port = receivePacket.getPort();
  String capitalizedSentence = sentence.toUpperCase();
  sendData = capitalizedSentence.getBytes();
  DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, IPAddress, port);
  serverSocket.send(sendPacket);
 }
 }
}
```

DSCC Lab Week2

1 .Create a Web Service class with the web service method that returns a MD5-hahsed value of an input string.

2 Create a server class for deploying the above Web Service.

3. Create a client program to invoke the above web service method.

Source Code:

```
// MD5WebService.java
package vce.webservices.server;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import javax.jws.WebMethod;
import javax.jws.WebService;

@WebService
public class MD5WebService {
@WebMethod
public String hashString(String input) {
try {
MessageDigest msgDigest = MessageDigest.getInstance("MD5");
byte[] inputBytes = input.getBytes();
byte[] hashedBytes = msgDigest.digest(inputBytes);

StringBuffer sb = new StringBuffer();
for (int i = 0; i < hashedBytes.length; i++) {
sb.append(Integer.toString((hashedBytes[i] & 0xff) + 0x100, 16)
.substring(1));
}

return sb.toString();
```

```
} catch (NoSuchAlgorithmException ex) {
ex.printStackTrace();
return "";
}
}
}
```

//WebServiceServer.java

```
package vce.webservices.server;

import javax.xml.ws.Endpoint;

public class WebServiceServer {

/**
* Starts a simple server to deploy the web service.
*/
public static void main(String[] args) {
String bindingURI = "http://localhost:9898/md5WebService";
MD5WebService webService = new MD5WebService();
Endpoint.publish(bindingURI, webService);
System.out.println("Server started at: " + bindingURI);
}
}
```

//WebServiceClient.java

```
package vce.webservices.client;

public class WebServiceClient {


/** * Starts the web service client. */
public static void main(String[] args) {
MD5WebServiceService client = new MD5WebServiceService();
MD5WebService md5Webservice = client.getMD5WebServicePort();
String hash = md5Webservice.hashString("hyderabad");
System.out.println("MD5 hash string: " + hash);
}
}
```

Week-3

Implement a 2PC for distributed transaction management.

Server.java :

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

```
{
static int n;
static String[] status= new String[2];
Clients(int num){
n=num;
for (int j=0;j<n ;j++ )
{
status[j] = new String("NotPrepared");
}
}
}
class Coordinator implements Runnable{
public static int i=-1;
int flag=1;
Socket s; Thread t;
MulticastSocket ms =null;
InetAddress group ;
Coordinator(Socket c){
try{
ms = new MulticastSocket(8899);
group= InetAddress.getByName("228.5.6.7");
ms.joinGroup(group);
}
catch (Exception e){
e.printStackTrace();
}
s=c;
t = new Thread(this);
t.start();
i++;
}
public void run(){
int index = i;
String clientSattus;
try{
DataInputStream input=new
DataInputStream(s.getInputStream());
DataOutputStream output=new
DataOutputStream(s.getOutputStream());
while (true){
clientSattus = input.readUTF();
System.out.println("Client "+index+" "+clientSattus);
 Clients.status[index] = new String (clientSattus);
for (int k=0;k<Clients.n; k++){
System.out.println(Clients.status[k]);
if (Clients.status[k].equalsIgnoreCase("prepared"))
 continue;
else
flag=0;
if (flag==1){
byte[] msg = new String("commit").getBytes();
DatagramPacket msgpack = new
DatagramPacket(msg,msg.length, group, 8899);
```

```
ms.send(msgpack);
System.out.println("BroadCasted msg "+new String(msg));
}
flag=1;
}
}
catch (Exception e){
e.printStackTrace();
}
}
}

class Server {
public static ServerSocket ss; Server(){}
public static void main(String args[]) throws Exception{
ss = new ServerSocket(8088);
int num;
num = Integer.parseInt(args[0]);
 new Clients(num);
while (true){
System.out.println("Server waiting: ");
Socket s = ss.accept();
new Coordinator(s);
}
}
}
```

MulticastPeer.java:

```
import
java.net.*;
import
java.io.*;
public class MulticastPeer{
   public static void main(String args[]){
        // args give message contents and destination multicast group (e.g. "228.5.6.7")
        MulticastSocket s =null;
        try {
                InetAddress group = InetAddress.getByName(args[1]);
                s = new MulticastSocket(6789);
                s.joinGroup(group);
                byte [] m = args[0].getBytes();
                DatagramPacket messageOut = new DatagramPacket(m, m.length, group, 6789);
                s.send(messageOut);
                byte[] buffer = new byte[1024];
                for(int i=0; i< 3;i++) {                        // get messages from others in group
                        DatagramPacket messageIn = new DatagramPacket(buffer, buffer.length);
                        s.receive(messageIn);
                        System.out.println("Received:" + new String(messageIn.getData()));
                }
                s.leaveGroup(group);
```

```
        }catch (SocketException e){System.out.println("Socket: " + e.getMessage());
        }catch (IOException e){System.out.println("IO: " + e.getMessage());
        }finally {if(s != null) s.close();}}}
```

DBConnector.java :

```
        import java.sql.*;
        class  DBConnector
{
          public static Connection getDBConnection(String dsn) throws Exception{
                  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                  return DriverManager.getConnection("jdbc:odbc:"+dsn);
          }
}
```

Client.java :

```
        import java.io.*;
        import java.net.*;
        import java.awt.*;
        import java.awt.event.*;
        import javax.swing.*;
        import java.sql.*;
        import java.util.*;
        class Client extends JFrame implements ActionListener{
        JButton b1,b2,b4,b5;
        JPanel p1,p2;
        JTextField t1;
        JLabel l1;
        ServerSocket ss;
        Socket s;
        DataOutputStream output;
        DataInputStream input;
        Connection con;
        Statement stmt;
        String serverMessage="Prepared";
        static int port;
        Client(){
        b1=new JButton("Prepared");
        b2=new JButton("NotPrepared");
        b4=new JButton("Execute");
        b5=new JButton("Exit");
        t1=new JTextField("",35);
        l1=new JLabel("SQL");
        p1=new JPanel();
        p2=new JPanel();
        p1.setLayout(new FlowLayout());
        p1.add(l1);
```

```
p1.add(t1);
p2.add(b1);
p2.add(b2);
p2.add(b4);
p2.add(b5);
add(p1);
add(p2,"South");
setSize(600,300);
setTitle("DNS Client");
setVisible(true);
b1.addActionListener(this);
b2.addActionListener(this);
b4.addActionListener(this);
b5.addActionListener(this);
setDefaultCloseOperation(EXIT_ON_CLOSE);
MulticastSocket ms =null;
InetAddress group ;
try{
s = new Socket("localhost",8088);
System.out.println("Client Connected");
output=new DataOutputStream(s.getOutputStream());
input=new DataInputStream(s.getInputStream());
con = DBConnector.getDBConnection("mydb");
stmt = con.createStatement();
con.setAutoCommit(false);
ms = new MulticastSocket(8899);
group=InetAddress.getByName("228.5.6.7");
ms.joinGroup(group);
byte[] buffer = new byte[1024];
while (true){
        DatagramPacket serMsg= new DatagramPacket(buffer, buffer.length);
        ms.receive(serMsg);
        String commitMsg = new String (serMsg.getData()).trim();
        if (commitMsg.equals("commit"))
        {
                System.out.println("Received "+commitMsg); con.commit();
                System.out.println("Transactions Committed");
      }}}
        catch (Exception e){
                e.printStackTrace();
        }
    }
public void actionPerformed(ActionEvent ae){
try{
String str=ae.getActionCommand();
```

```
if(str.equals("Execute")){

            String query = t1.getText(); System.out.println(stmt.executeUpdate(query));
            t1.setText("Query Executed(NotPrepared)"); output.writeUTF("NotPrepared");
}
if(str.equals("Prepared")){
            output.writeUTF("Prepared");
}
```

Query.txt :

```
update account set bal = 500 where accno = 1
```

week-4

multi chat application:

server.java

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

// Server class
public class Server
{
   static Vector<ClientHandler> ar = new Vector<>();

   // counter for clients
   static int i = 0;

   public static void main(String[] args) throws IOException
   {
      // server is listening on port 1234
      ServerSocket ss = new ServerSocket(1234);

      Socket s;
      while (true)
      {
         // Accept the incoming request
         s = ss.accept();

         System.out.println("New client request received : " + s);

         // obtain input and output streams
         DataInputStream dis = new DataInputStream(s.getInputStream());
         DataOutputStream dos = new DataOutputStream(s.getOutputStream());

         System.out.println("Creating a new handler for this client...");
         ClientHandler mtch = new ClientHandler(s,"client " + i, dis, dos);
         Thread t = new Thread(mtch);

         System.out.println("Adding this client to active client list");

         // add this client to active clients list
```

```
            ar.add(mtch);

            // start the thread.
            t.start();
            i++;
}}}

// ClientHandler class
class ClientHandler implements Runnable
{
   Scanner scn = new Scanner(System.in);
   private String name;
   final DataInputStream dis;
   final DataOutputStream dos;
   Socket s;
   boolean isloggedin;

   // constructor
   public ClientHandler(Socket s, String name,
                        DataInputStream dis, DataOutputStream dos) {
      this.dis = dis;
      this.dos = dos;
      this.name = name;
      this.s = s;
      this.isloggedin=true;
   }

   @Override
   public void run() {

      String received;
      while (true)
      {
         try
         {
            // receive the string
            received = dis.readUTF();

            System.out.println(received);

            if(received.equals("logout")){
               this.isloggedin=false;
               this.s.close();
               break;
            }

            // break the string into message and recipient part
            StringTokenizer st = new StringTokenizer(received, "#");
            String MsgToSend = st.nextToken();
            String recipient = st.nextToken();

            // search for the recipient in the connected devices list.
            // ar is the vector storing client of active users
```

```
            for (ClientHandler mc : Server.ar)
            {
               // if the recipient is found, write on its
               // output stream
               if (mc.name.equals(recipient) && mc.isloggedin==true)
               {
                  mc.dos.writeUTF(this.name+" : "+MsgToSend);
                  break;
               }
            }
         } catch (IOException e) {

            e.printStackTrace();
         }

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

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

Client.java

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

public class Client
{
   final static int ServerPort = 1234;

   public static void main(String args[]) throws UnknownHostException, IOException
   {
      Scanner scn = new Scanner(System.in);
      InetAddress ip = InetAddress.getByName("localhost");

      // establish the connection
      Socket s = new Socket(ip, ServerPort);

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

      // sendMessage thread
      Thread sendMessage = new Thread(new Runnable()
      {
         @Override
         public void run() {
            while (true) {
```

```
            // read the message to deliver.
            String msg = scn.nextLine();

            try {
               // write on the output stream
               dos.writeUTF(msg);
            } catch (IOException e) {      e.printStackTrace();                  }
         }
      }
   });

   // readMessage thread
   Thread readMessage = new Thread(new Runnable()
   {
      @Override
      public void run() {

         while (true) {
            try {
               // read the message sent to this client
               String msg = dis.readUTF();
               System.out.println(msg);
            } catch (IOException e) {     e.printStackTrace();     }
         }
      }
   });

   sendMessage.start();
   readMessage.start();

  }
}
```