

INDEX

Sr. No	TOPIC	DATE	SIGN
1	Understanding Remote Process Communication.		
	i) Develop a program for one way client and server communication.	29-07-19	
	ii) Develop a program for client server chat using Java Socket.	29-07-19	
	iii) Develop a program for client server GUI Chat.	07-08-19	
	iv) Implement a server which calculates sum of 2 numbers.	07-08-19	
	v) Develop a program for client n server communication using UDP.	07-08-19	
	vi) Implement a server to find whether an entered number is even [UDP].	08-08-19	
	vii) Implement a program for multi-client chat server.	08-08-19	
2	Understanding Remote Procedure Call.		
	i) Implement a Date time server containing date () and time () remote.	11-08-19	
	ii) Implement a Server to do the following: (Use RPC)	11-08-19	
	a) Get two number from the client.		
	b) Server Processing summation of two number		
	c) Server send the processed data to client and client checks whether the sum is greater than 100 or not. And display appropriate msg.		
	iii) Write a program to implement server calculator containing add(), sub(), mul(), div(). Implement using RPC.	14-08-19	
3	Understanding Remote Method Invocation.		
	i) Retrieve time and date function from server to client by implementing RMI.	19-08-19	
	ii) Implement a server to solve equation $c = (a+b)^2$ & $c = (a+b)^3$.	19-08-19	
	iii) Implement a server to solve equation $(ax^2 + bx + c = 0)$.	21-08-19	
	iv) Design a Graphical User Interface to find greatest of two numbers.	21-08-19	
	v) Design a Graphical User Interface (GUI) based Basic calculator using RMI.	28-08-19	
4	Implementation of Shared Memory		
	i) Write a program to increment counter in Shared Memory	29-08-19	
5	Understanding Remote Object Communication.		
	i) Retrieve the students information from the College Database.	25-09-19	
	ii) Retrieve the list of book available in the library.	25-09-19	
	iii) Retrieve the billing information from the MTNL Database.	26-09-19	
6	Understanding Enterprise Java Beans.		
	i) Write a program for basic Arithmetic operation using Stateless session Bean.	03-10-19	
	ii) Write a program for basic Arithmetic operation using Stateful session Bean.	03-10-19	
	iii) Write a program to demonstrate Message Driven Bean.	09-10-19	
	iv) Write a program to demonstrate Entity Bean.	10-10-19	
7	Implementation of Mutual Exclusion using Token Ring.		
	i) Write a program to demonstrate token ring technique.	16-10-19	
8	Study of Cloud Virtualization Technologies.	17-10-19	
9	Study of Grid Services.	17-10-19	

Practical No 1

AIM: Understanding Remote Process Communication.

Description:

Socket Programming refers to writing programs that execute across multiple computers in which the devices are all connected to each other using a network. A Server runs on a specific computer and has a socket that is bound to a specific port number. The server just waits, listening to the socket for a client to make a connection request. The client and server can communicate by writing to or reading from their socket.

A **Socket** is simply an endpoint for communication between machines.

There are two communication protocol that one can use for Socket Programming:

- 1) Transmission control protocol (TCP)
- 2) User Datagram Protocol (UDP)

1) Transmission Control Protocol.

TCP is a connection oriented protocol. In order to do communication over the TCP Protocol, a connection must be established between the pair of sockets. While one of the socket listen for connection request (Server), the other ask for connection (client). Once two socket have been connected, they can be used to transmit data in both direction.

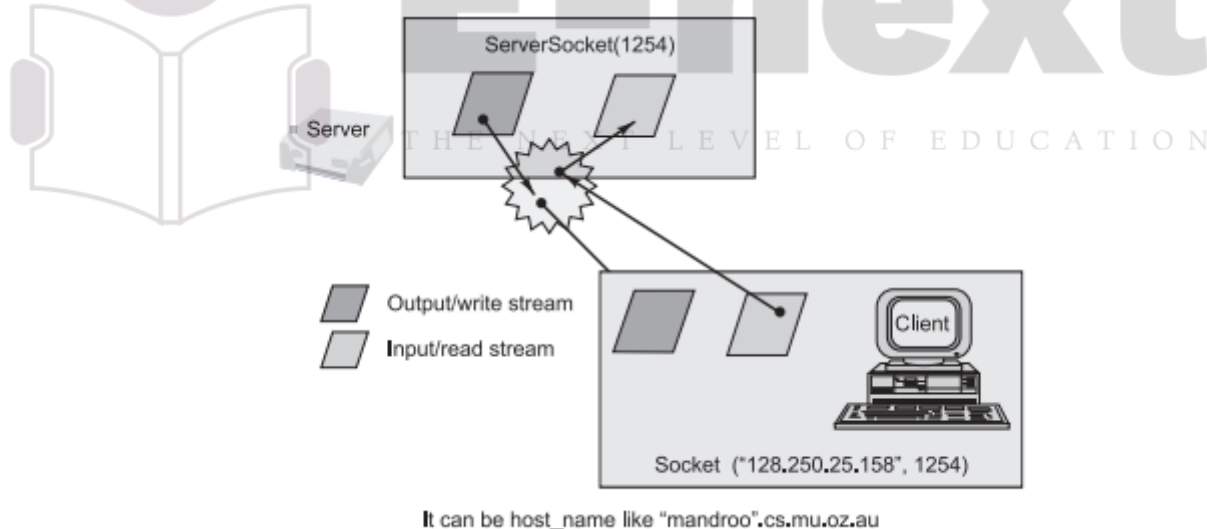


Fig: Socket-based client and server programming.

Classes used for Socket programming :

1) Socket

Socket class represents the socket that both the client and the server use to communicate with each other.

2) ServerSocket

The ServerSocket class used by server application to obtain a port and listen for client requests.

Steps for creating a simple Server Program :**Step 1) Open the Server Socket.**

```
ServerSocket ss=new ServerSocket(PORT);
```

Step 2) Wait for the client request.

```
Socket client=ss.accept();
```

Step 3) Create I/O Streams for the communicating to the client.

```
DataInputStream dis=new DataInputStream(client.getInputStream());
```

```
DataOutputStream dos=new DataOutputStream(client.getOutputStream());
```

Step 4) Perform communication with the client.

```
Receive from client : => String str=dis.readUTF( );
```

```
Send data to client : => dos.writeUTF("Hello");
```

Step 5) Close the socket.

```
Client.close();
```

Steps for creating a simple Client Program :**Step 1) Create a Socket Object.**

```
Socket s=new Socket(server,port_id);
```

Step 2) Create I/O Streams for the communicating to the server.

```
DataInputStream dis=new DataInputStream(client.getInputStream());
```

```
DataOutputStream dos=new DataOutputStream(client.getOutputStream());
```

Step 3) Perform communication with the server.

```
Receive data from server : => String str=dis.readUTF( );
```

```
Send data to the server : => dos.writeUTF("Hello");
```

Step 4) Close the socket.

```
s.close();
```

2) User Datagram Protocol.

UDP is a connection less protocol that allows for packets of data to be transmitted. Datagram Packets are used to implement a connection less packet delivery service supported by the UDP Protocol. Each message is transferred from source machine to destination based on information contained within that packet.

The format of Datagram Packet is :

Msg length Host serverPort

The class **DatagramPacket** contain several constructors that can be used for creating packet object.

For Example : **DatagramPacket(byte[] buff, int length, InetAddress address, int port);**

The class **datagramSocket** support various methods that can be used for transmitting or receiving data over the network, The two key methods are :

void send(DatagramPacket p) => send a datagram packet from this socket.

void receive(DatagramPacket p) => receive a datagram packet from this socket.

- 1) Develop a program for one way client and server communication using java Socket , where client sends a message to the server, then the server reads the message and print it.

Source Code:

Filename : DemoClient.java :

```
import java.net.*;
import java.io.*;
public class DemoClient
{
    public static void main(String args[]) throws Exception
    {
        Socket s=new Socket("localhost",1234);
        DataOutputStream dis=new DataOutputStream(s.getOutputStream());
        dis.writeUTF("Hello !! How are you");
        s.close();
    }
}
```

Filename : DemoServer.java :

```
import java.net.*;
import java.io.*;
public class DemoServer
{
    public static void main(String args[]) throws Exception
    {
        ServerSocket ss=new ServerSocket(1234);
        Socket s=ss.accept();
        DataInputStream dis=new DataInputStream(s.getInputStream());
        String msg=dis.readUTF();
        System.out.println("Message from client : "+msg);
    }
}
```

OUTPUT:

DemoClient.java



```
C:\Windows\system32\cmd.exe
G:\adc>javac DemoClient.java
G:\adc>java DemoClient
G:\adc>
```

DemoServer.java



```
C:\Windows\system32\cmd.exe
G:\adc>javac DemoServer.java
G:\adc>java DemoServer
Message from client : Hello !! How are you
G:\adc>
```

2) Develop a program for client server chat using java socket.

Source Code:

Filename : MyClient.java :

```
import java.net.*;
import java.io.*;
import java.util.*;
public class My_Client
{
    public static void main(String args[]) throws Exception
    {
        String str;
        Socket s=new Socket("localhost",3333);
        DataInputStream dis=new DataInputStream(s.getInputStream());
        DataOutputStream dos=new DataOutputStream(s.getOutputStream());
        Scanner in=new Scanner(System.in);
        while(true)
        {
            System.out.print("Client says : ");
            str=in.nextLine();
            dos.writeUTF(str);
            str=dis.readUTF();
            System.out.println("Server says : "+str);
            if(str.equals("exit"))
                break;
        }
        s.close();
    }
}
```

Filename : MyServer.java :

```
import java.net.*;
import java.io.*;
import java.util.*;
public class My_Server
{
    public static void main(String args[]) throws Exception
    {
        String str;
        ServerSocket ss=new ServerSocket(3333);
        Socket s=ss.accept();
        DataInputStream dis=new DataInputStream(s.getInputStream());
        DataOutputStream dos=new DataOutputStream(s.getOutputStream());
        Scanner in=new Scanner(System.in);

        while(true)
```

```

        {
            str=dis.readUTF();
            if (str.equals("exit"))
            {
                dos.writeUTF("exit");
                break;
            }
            System.out.println(" Client says: "+str);
            System.out.print("Server says : ");
            str=in.nextLine();
            dos.writeUTF(str);
        }
        ss.close();
        s.close();
    }
}

```

OUTPUT:

My_Client.java

```

G:\adc>javac My_Client.java

G:\adc>java My_Client
Client says : hii
Server says : hello
Client says : how are you
Server says : i am fine
Client says : exit
Server says : exit

G:\adc>_

```

My_Server.java

```

G:\adc>javac My_Server.java

G:\adc>java My_Server
Client says: hii
Server says : hello
Client says: how are you
Server says : i am fine

G:\adc>_

```

3) Develop a program for client server GUI chat.

Source Code:

Filename : ClientChat.java :

```
import java.io.*;
import java.net.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class ClientChat extends JFrame implements ActionListener,Runnable
{
    JButton b;
    JTextField tf;
    JTextArea ta;
    Socket s;
    PrintWriter pw;
    BufferedReader br;
    Thread th;
    public ClientChat()
    {
        b=new JButton("Send");
        b.addActionListener(this);
        tf=new JTextField(20);
        ta=new JTextArea(19,30);
        add(ta);
        add(tf);
        add(b);
        try
        {
            s=new Socket("localhost",1234);
            br=new BufferedReader(new InputStreamReader(s.getInputStream()));
            pw=new PrintWriter(s.getOutputStream(),true);
        }
        catch(Exception e) { }
        th=new Thread(this);
        th.start();
    }
    public void actionPerformed(ActionEvent ae)
    {
        pw.println(tf.getText());
        ta.append("Client says : "+tf.getText()+"\n");
        tf.setText("");
    }
}
```

public void run()

Khan S Alam

```

    {
        while(true)
        {
            try
            {
                ta.append("Server says : "+br.readLine()+"\n");
            }
            catch(Exception e) {}
        }
    }
    public static void main(String args[])
    {
        ClientChat c = new ClientChat();
        c.setLayout(new FlowLayout());
        c.setSize(400,400);
        c.setTitle("Client");
        c.setVisible(true);
    }
}

```

Filename : ServerChat.java :

```

import java.io.*;
import java.net.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class ServerChat extends JFrame implements ActionListener,Runnable
{

```

```

    JButton b;
    JTextField tf;
    JTextArea ta;
    ServerSocket ss;
    Socket s;
    PrintWriter pw;
    BufferedReader br;
    Thread th;
    public ServerChat()
    {

```

```

        b=new JButton("Send");
        b.addActionListener(this);
        tf=new JTextField(20);
        ta=new JTextArea(19,30);
        add(ta);
        add(tf);
        add(b);
        try
        {

```



```

        ss=new ServerSocket(1234);
        s=ss.accept();
        br=new BufferedReader(new InputStreamReader(s.getInputStream()));
        pw=new PrintWriter(s.getOutputStream(),true);
    }
    catch(Exception e) { }
    th=new Thread(this);
    th.start();
}
public void actionPerformed(ActionEvent ae)
{
    pw.println(tf.getText());
    ta.append("Server says : "+tf.getText()+"\n");
    tf.setText("");
}
public void run()
{
    while(true)
    {
        try
        {
            ta.append("Client says : "+br.readLine()+"\n");
        }
        catch(Exception e) {}
    }
}
public static void main(String args[])
{
    ServerChat as = new ServerChat();
    as.setLayout(new FlowLayout());
    as.setSize(400,400);
    as.setTitle("Server");
    as.setVisible(true);
}
}

```

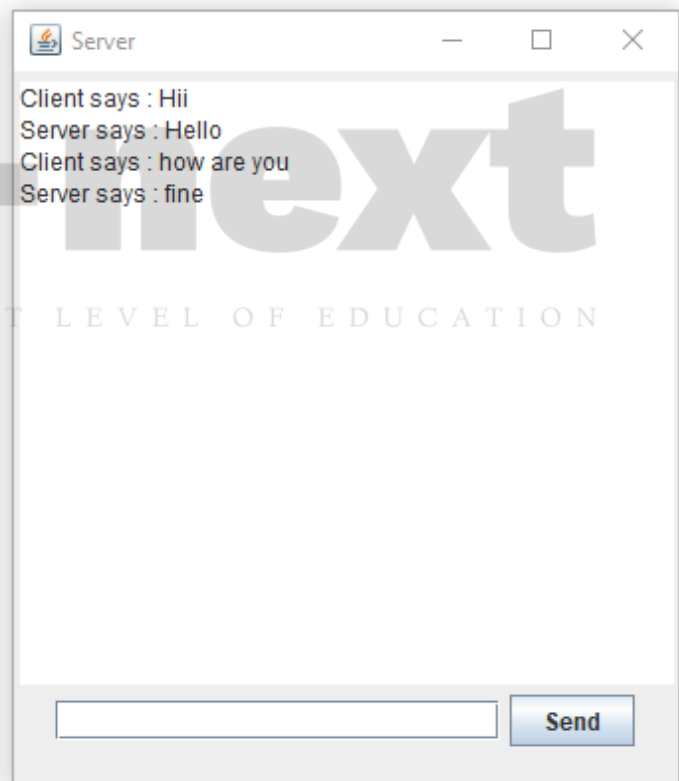
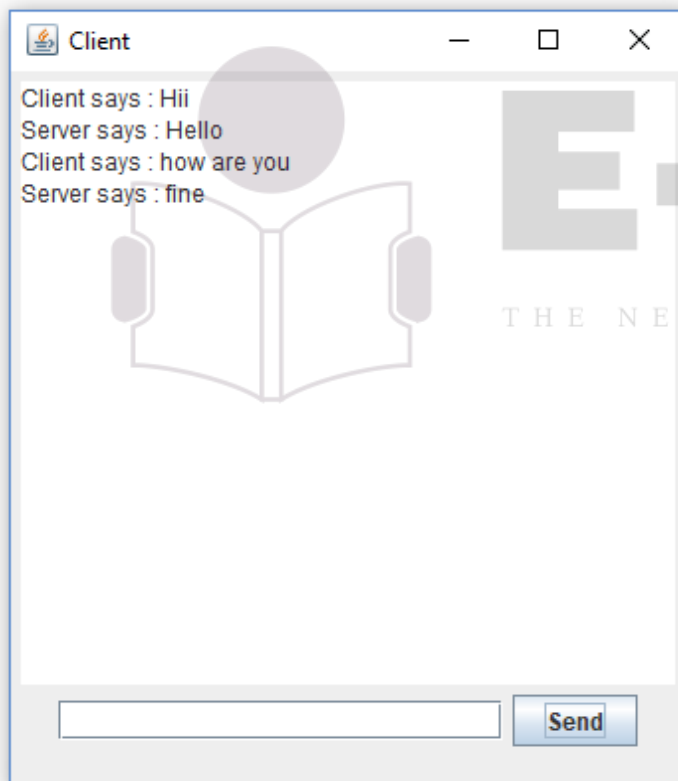
OUTPUT:

ClientChat.java

```
C:\Windows\system32\cmd.exe - java ClientChat
G:\adc>javac ClientChat.java
G:\adc>java ClientChat
```

ServerChat.java

```
C:\Windows\system32\cmd.exe - java ServerChat
G:\adc>javac ServerChat.java
G:\adc>java ServerChat
```



4) Implement a server which calculates sum of two numbers using Java Socket.

Source Code:

Filename : Client-Add.java :


```
import java.net.*;
import java.io.*;
import java.util.*;
public class Client_Add
{
    public static void main(String args[]) throws Exception
    {
        int a,b;
        Socket s=new Socket("localhost",1234);
        DataInputStream dis=new DataInputStream(s.getInputStream());
        DataOutputStream dos=new DataOutputStream(s.getOutputStream());
        Scanner in=new Scanner(System.in);
        System.out.print("Enter first number : ");
        a=in.nextInt();
        System.out.print("Enter second number : ");
        b=in.nextInt();
        dos.write(a);
        dos.write(b);
        int ans=dis.read();
        System.out.println("Result : "+ans);
        s.close();
    }
}
```

Filename : Server-Add.java :

```
import java.net.*;
import java.io.*;
import java.util.*;
public class Server_Add
{
    public static void main(String args[]) throws Exception
    {
        ServerSocket ss=new ServerSocket(1234);
        Socket s=ss.accept();
        DataInputStream dis=new DataInputStream(s.getInputStream());
        DataOutputStream dos=new DataOutputStream(s.getOutputStream());
        int a=dis.read();
        int b=dis.read();
        int res=a+b;
        dos.write(res);
    }
}
```


OUTPUT:

Client_Add.java



```
C:\Windows\system32\cmd.exe
G:\adc>javac Client_Add.java
G:\adc>java Client_Add
Enter first number : 23
Enter second number : 10
Result : 33
G:\adc>
```

Server_Add.java



```
C:\Windows\system32\cmd.exe
G:\adc>javac Server_Add.java
G:\adc>java Server_Add
G:\adc>
```

5) Develop a program for one way client and server communication using Datagram Socket.

Filename : UDPCClient.java :

```
import java.net.*;
import java.io.*;
public class UDPCClient
{
    public static void main(String args[]) throws Exception
    {
        String s="How are you";
        DatagramSocket ds=new DatagramSocket();
        InetAddress ip=InetAddress.getByName("localhost");
        DatagramPacket p=new DatagramPacket(s.getBytes(),s.length(),ip,2222);
        ds.send(p);
    }
}
```

Filename : UDPServer.java :

```
import java.net.*;
import java.io.*;
public class UDPServer
{
    public static void main(String args[]) throws Exception
    {
        DatagramSocket ds=new DatagramSocket(2222);
        byte[] b=new byte[1024];
        DatagramPacket p=new DatagramPacket(b,1024);
        ds.receive(p);
        String msg=new String(p.getData(),0,p.getLength());
        System.out.println("Message from client : "+msg);
    }
}
```

OUTPUT:

UDPCClient.java



```
C:\Windows\system32\cmd.exe
G:\adc>javac UDPCClient.java
G:\adc>java UDPCClient
G:\adc>
```

UDPServer.java



```
C:\Windows\system32\cmd.exe
G:\adc>javac UDPServer.java
G:\adc>java UDPServer
Message from client : How are you
G:\adc>
```

6) Implement a server to find whether an entered number is odd or even using Datagram Socket.

Source Code:

Filename : ClientUDP.java :

```
import java.net.*;
import java.io.*;
import java.util.*;
public class ClientUDP
{
    public static void main(String args[]) throws Exception
    {
        Scanner in=new Scanner(System.in);
        System.out.println("Enter a number");
        String a=in.nextLine();
        byte[] t1=a.getBytes();
        InetAddress ip=InetAddress.getByName("localhost");
        DatagramPacket p=new DatagramPacket(t1,a.length(),ip,2222);
        DatagramSocket ds=new DatagramSocket();
        ds.send(p);
    }
}
```

Filename : ServerUDP.java :

```
import java.net.*;
import java.io.*;
import java.util.*;
public class ServerUDP
{
    public static void main(String args[]) throws Exception
    {
        DatagramSocket ds=new DatagramSocket(2222);
        byte[] b=new byte[1024];
        DatagramPacket p=new DatagramPacket(b,1024);
        ds.receive(p);
        String s1=new String(p.getData(),0,p.getLength());
        int n=Integer.parseInt(s1);
        if(n%2==0)
            System.out.println(n+" is even");
        else
            System.out.println(n+" is odd");
    }
}
```

E-next
THE NEXT LEVEL OF EDUCATION

OUTPUT:**ClientUDP.java**

```
C:\Windows\system32\cmd.exe
G:\adc>javac ClientUDP.java
G:\adc>java ClientUDP
Enter a number
23
G:\adc>_
```

ServerUDP.java

```
C:\Windows\system32\cmd.exe
G:\adc>javac ServerUDP.java
G:\adc>java ServerUDP
23 is odd
G:\adc>_
```

7) Implement a Program for multi-client chat server.**Filename : chatClient.java :**

```

import java.net.*;
import java.io.*;
import java.util.*;
import java.awt.*;
class chatClient extends Frame implements Runnable
{
    Socket soc;
    TextField tf;
    TextArea ta;
    Button btnSend,btnClose;
    String sendTo;
    String LoginName;
    Thread t=null;
    DataOutputStream dout;
    DataInputStream din;
    chatClient(String LoginName,String chatwith) throws Exception
    {
        super(LoginName);
        this.LoginName=LoginName;
        sendTo=chatwith;
        tf=new TextField(50);
        ta=new TextArea(50,50);
        btnSend=new Button("Send");
        btnClose=new Button("Close");
        soc=new Socket("127.0.0.1",5217);
        din=new DataInputStream(soc.getInputStream());
        dout=new DataOutputStream(soc.getOutputStream());
        dout.writeUTF(LoginName);
        t=new Thread(this);
        t.start();
    }
    void setup()
    {
        setSize(600,400);
        setLayout(new GridLayout(2,1));
        add(ta);
        Panel p=new Panel();
        p.add(tf);
        p.add(btnSend);
        p.add(btnClose);
        add(p);
        show();
    }
    public boolean action(Event e,Object o)
    {

```



```

if(e.arg.equals("Send"))
{
    try
    {
        dout.writeUTF(sendTo + " " + "DATA" + " " + tf.getText().toString());
        ta.append("\n" + LoginName + " Says:" + tf.getText().toString());
        tf.setText("");
    }
    catch(Exception ex)
    {
    }
}
else if(e.arg.equals("Close"))
{
    try
    {
        dout.writeUTF(LoginName + " LOGOUT");
        System.exit(1);
    }
    catch(Exception ex)
    {
    }
}
return super.action(e,o);
}
public static void main(String args[]) throws Exception
{
    chatClient Client1=new chatClient(args[0],args[1]);
    Client1.setup();
}
public void run()
{
    while(true)
    {
        try
        {
            ta.append( "\n" + sendTo + " Says :" + din.readUTF());

        }
        catch(Exception ex)
        {
            ex.printStackTrace();
        }
    }
}
} }

```

Filename : chatServer.java :

```

import java.net.*;
import java.util.*;
import java.io.*;
class chatServer
{
    static Vector ClientSockets;
    static Vector LoginNames;
    chatServer() throws Exception
    {
        ServerSocket soc=new ServerSocket(5217);
        ClientSockets=new Vector();
        LoginNames=new Vector();
        while(true)
        {
            Socket CSoc=soc.accept();
            AcceptClient obClient=new AcceptClient(CSoc);
        }
    }
    public static void main(String args[]) throws Exception
    {
        chatServer ob=new chatServer();
    }
}
class AcceptClient extends Thread
{
    Socket ClientSocket;
    DataInputStream din;
    DataOutputStream dout;
    AcceptClient (Socket CSoc) throws Exception
    {
        ClientSocket=CSoc;
        din=new DataInputStream(ClientSocket.getInputStream());
        dout=new DataOutputStream(ClientSocket.getOutputStream());
        String LoginName=din.readUTF();
        System.out.println("User Logged In :" + LoginName);
        LoginNames.add(LoginName);
        ClientSockets.add(ClientSocket);
        start();
    }
    public void run()
    {
        while(true)
        {
            try
            {
                String msgFromClient=new String();
                msgFromClient=din.readUTF();
                StringTokenizer st=new StringTokenizer(msgFromClient);

```

```

String Sendto=st.nextToken();
String MsgType=st.nextToken();
int iCount=0;
if(MsgType.equals("LOGOUT"))
{
    for(iCount=0;iCount<LoginNames.size();iCount++)
    {
        if(LoginNames.elementAt(iCount).equals(Sendto))
        {
            LoginNames.removeElementAt(iCount);
            ClientSockets.removeElementAt(iCount);
            System.out.println("User " + Sendto + " Logged Out ...");
            break;
        }
    }
}
else
{
    String msg="";
    while(st.hasMoreTokens())
    {
        msg=msg+" " +st.nextToken();
    }
    for(iCount=0;iCount<LoginNames.size();iCount++)
    {
        if(LoginNames.elementAt(iCount).equals(Sendto))
        {
            Socket tSoc=(Socket)ClientSockets.elementAt(iCount);
            DataOutputStream tdout=new DataOutputStream(tSoc.getOutputStream());
            tdout.writeUTF(msg);
            break;
        }
    }
    if(iCount==LoginNames.size())
        dout.writeUTF("I am offline");
    else { }
}
if(MsgType.equals("LOGOUT"))
{
    break;
}
}
}
catch(Exception ex)
{
    ex.printStackTrace();
}
}
}
}

```

OUTPUT:**ChatServer.java**

```

C:\Windows\system32\cmd.exe - java chatServer
G:\adc>java chatServer
User Logged In :anand
User Logged In :aman

```

ChatClient.java

```

C:\Windows\system32\cmd.exe - java chatClient aman anand
G:\adc>java chatClient aman anand

```

ChatClient.java

```

C:\Windows\system32\cmd.exe - java chatClient anand aman
G:\adc>java chatClient anand aman

```



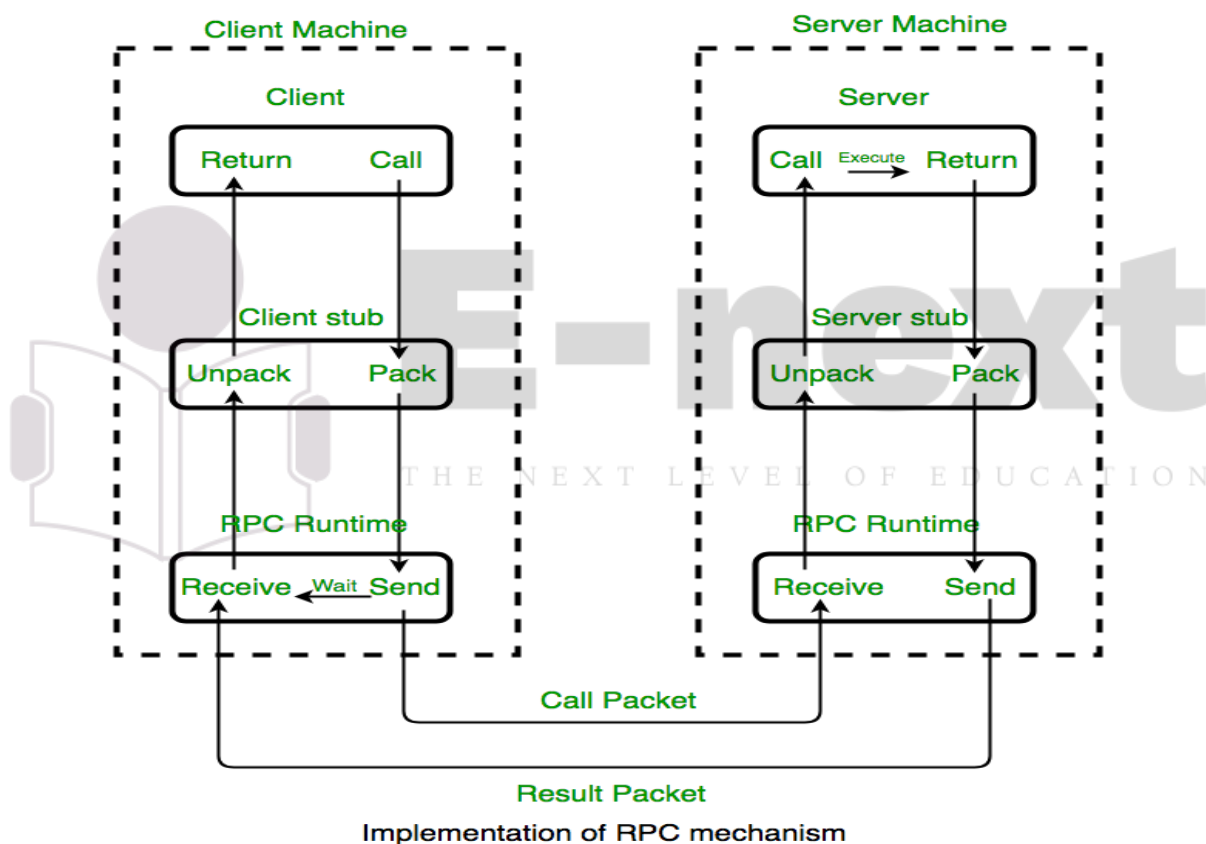
Practical No 2

AIM: Understanding Remote Procedure Call.

Description:

Remote Procedure Call (RPC) is a powerful technique for constructing **distributed, client-server based applications**. A remote procedure call is an inter-process communication technique that is used for client-server based applications. It is also known as a subroutine call or a function call. It is based on extending the conventional local procedure calling so that the **called procedure need not exist in the same address space as the calling procedure**. The two processes may be on the same system, or they may be on different systems with a network connecting them.

RPC Working:



The sequence of events in a remote procedure call are given as follows:

- 1> The client stub is called by the client.
- 2> The client stub makes a system call to send the message to the server and puts the parameters in the message.
- 3> The message is sent from the client to the server by the client's operating system.
- 4> The message is passed to the server stub by the server operating system.
- 5> The parameters are removed from the message by the server stub.
- 6> Then, the server procedure is called by the server stub.

8) Implement a Date time server containing date() and time() remote , through socket programming.

Source Code:

Filename : DateServer.java :

```
import java.net.*;
import java.io.*;
import java.util.*;
import java.text.*;
public class DateServer
{
    DateServer() throws Exception
    {
        ServerSocket ss=new ServerSocket(4444);
        Socket s=ss.accept();
        DataOutputStream dos=new DataOutputStream(s.getOutputStream());
        dos.writeUTF(date( ));
        dos.writeUTF(time( ));
        dos.flush();
    }
    public String date( )
    {
        return new SimpleDateFormat("dd/mm/yyyy").format(new Date()).toString();
    }
    public String time( )
    {
        return new SimpleDateFormat("hh:mm:ss").format(new Date()).toString();
    }

    public static void main(String args[ ]) throws Exception
    {
        DateServer d=new DateServer();
    }
}
```

Filename : DateClient.java :

```
import java.net.*;
import java.io.*;
public class DateClient
{
    public static void main(String args[]) throws Exception
    {
        Socket s=new Socket("localhost",4444);
        DataInputStream dis=new DataInputStream(s.getInputStream());
        String dt=dis.readUTF();
        String tm=dis.readUTF();
        System.out.println("Date : "+dt);
        System.out.println("Time : "+tm);
    }
}
```

OUTPUT:**DateServer.java**

```
C:\Windows\system32\cmd.exe
G:\adc\rpc>javac DateServer.java
G:\adc\rpc>java DateServer
G:\adc\rpc>
```

DateClient.java

```
C:\Windows\system32\cmd.exe
g:\adc\rpc>javac DateClient.java
g:\adc\rpc>java DateClient
Date : 25/28/2019
Time : 10:28:09
g:\adc\rpc>
```

9) Implement a Server to do the following: (Use RPC)

- i) Get two numbers from the client.**
- ii) Server processing the summation of the above two numbers**
- iii) Server sends the processed data to the client and client checks whether the sum is greater than 100 or not**
- iv) Client displays appropriate message**

Source Code:**Filename : ServerSum.java :**

```

import java.net.*;
import java.io.*;
import java.util.*;
public class ServerSum
{
    ServerSum() throws Exception
    {
        ServerSocket ss=new ServerSocket(4444);
        Socket s=ss.accept();
        DataInputStream dis=new DataInputStream(s.getInputStream());
        DataOutputStream dos=new DataOutputStream(s.getOutputStream());
        String str=dis.readUTF();
        StringTokenizer st = new StringTokenizer(str," ");
        String method = st.nextToken();
        int a = Integer.parseInt(st.nextToken());
        int b = Integer.parseInt(st.nextToken());
        dos.write(getSum(a,b));
        System.out.println("Sum of "+a+" and "+b+" is "+getSum(a,b));
        dos.flush();
    }
    public int getSum(int a,int b)
    {
        return a+b;
    }
    public static void main(String args[]) throws Exception
    {
        ServerSum d=new ServerSum();
    }
}

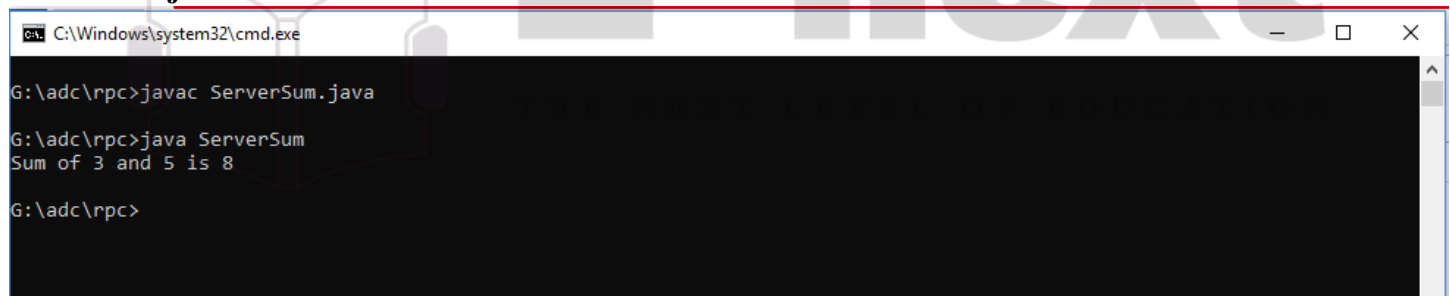
```


Filename : ClientSum.java :

```

import java.net.*;
import java.io.*;
public class ClientSum
{
    public static void main(String args[]) throws Exception
    {
        Socket s=new Socket("localhost",4444);
        DataInputStream dis=new DataInputStream(s.getInputStream());
        DataOutputStream dos=new DataOutputStream(s.getOutputStream());
        dos.writeUTF("sum 3 5");
        int sum=dis.read();
        if(sum>100)
            System.out.println("Sum is greater that 100");
        else
            System.out.println("SUM is not greater than 100");
    }
}

```

OUTPUT:**ServerSum.java**


```

C:\Windows\system32\cmd.exe
G:\adc\rpc>javac ServerSum.java
G:\adc\rpc>java ServerSum
Sum of 3 and 5 is 8
G:\adc\rpc>

```

ClientSum.java


```

C:\Windows\system32\cmd.exe
g:\adc\rpc>javac ClientSum.java
g:\adc\rpc>java ClientSum
SUM is not greater than 100
g:\adc\rpc>_

```

10) Write a program to implement a Server calculator containing ADD() , MUL() , SUB() , DIV(). Implement using RPC.

Source Code:

Filename : RPCServer.java :

```
import java.net.*;
import java.util.*;
class RPCServer
{
    DatagramSocket ds;
    DatagramPacket dp;
    String str, methodName, result;
    int val1, val2;
    RPCServer()
    {
        try
        {
            ds = new DatagramSocket(1200);
            byte b[] = new byte[4096];
            while(true)
            {
                dp = new DatagramPacket(b,b.length);
                ds.receive(dp);
                str = new String(dp.getData(),0,dp.getLength());
                if(str.equalsIgnoreCase("quit"))
                    System.exit(1);
                else
                {
                    StringTokenizer st = new StringTokenizer(str," ");
                    int i = 0;
                    while(st.hasMoreTokens()){
                        String token = st.nextToken();
                        methodName = token;
                        val1 = Integer.parseInt(st.nextToken());
                        val2 = Integer.parseInt(st.nextToken());
                    }
                }
                System.out.println("\nClient Selected \""+str+"\" Method :");
                System.out.println("\nFirst Value : "+val1);
                System.out.println("Second Value : "+val2);
                InetAddress ia = InetAddress.getLocalHost();
```

```

        if(methodName.equalsIgnoreCase("add"))
            result= "" + add(val1,val2);
        else if(methodName.equalsIgnoreCase("sub"))
            result= "" + sub(val1,val2);
        else if(methodName.equalsIgnoreCase("mul"))
            result= "" + mul(val1,val2);
        else if(methodName.equalsIgnoreCase("div"))
            result= "" + div(val1,val2);
        byte b1[] = result.getBytes();
        DatagramSocket ds1 = new DatagramSocket();
        DatagramPacket dp1 = new DatagramPacket(b1,b1.length,InetAddress.getLocalHost(), 1300);
        System.out.println("Result : "+result+"\n");
        ds1.send(dp1);
    }
}
catch (Exception e)
{
    e.printStackTrace();
}
}
public int add(int val1, int val2)
{
    return val1 + val2;
}
public int sub(int val3, int val4)
{
    return val3 - val4;
}
public int mul(int val3, int val4)
{
    return val3 * val4;
}
public int div(int val3, int val4)
{
    return val3 / val4;
}
public static void main(String[] args)
{
    new RPCServer();
}
}

```

Filename : RPCClient.java :

```

import java.io.*;
import java.net.*;
class RPCClient
{
    RPCClient()
    {
        try
        {
            InetAddress ia = InetAddress.getLocalHost();
            DatagramSocket ds = new DatagramSocket();
            DatagramSocket ds1 = new DatagramSocket(1300);
            System.out.println("\nRPC Client");
            System.out.println("-----\n");
            System.out.println("Enter Method Name with Parameter like add 3 4\n");
            while (true)
            {
                BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
                String str = br.readLine();
                byte b[] = str.getBytes();
                DatagramPacket dp = new DatagramPacket(b,b.length,ia,1200);
                ds.send(dp);
                dp = new DatagramPacket(b,b.length);
                ds1.receive(dp);
                String s = new String(dp.getData(),0,dp.getLength());
                System.out.println("\nResult =" + s + "\n");
                System.out.println("\n\nEnter Method Name with Parameter like add 3 4\n");
            }
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
    public static void main(String[] args)
    {
        new RPCClient();
    }
}

```

OUTPUT:**RPCServer.java**

```

C:\Windows\system32\cmd.exe
g:\adc\rpc>javac RPCServer.java
g:\adc\rpc>java RPCServer
Client Selected "add 3 4" Method :
First Value : 3
Second Value : 4
Result : 7

Client Selected "sub 4 2" Method :
First Value : 4
Second Value : 2
Result : 2

Client Selected "mul 4 5" Method :
First Value : 4
Second Value : 5
Result : 20

Client Selected "div 4 2" Method :
First Value : 4
Second Value : 2
Result : 2

g:\adc\rpc>_

```

RPCClient.java

```

C:\Windows\system32\cmd.exe - java RPCClient
G:\adc\rpc>javac RPCClient.java
G:\adc\rpc>java RPCClient
RPC Client
-----
Enter Method Name with Parameter like add 3 4
add 3 4
Result = 7

Enter Method Name with Parameter likeadd 3 4
sub 4 2
Result = 2

Enter Method Name with Parameter likeadd 3 4
mul 4 5
Result = 20

Enter Method Name with Parameter likeadd 3 4
div 4 2
Result = 2

Enter Method Name with Parameter likeadd 3 4
quit

```

Practical No 3

AIM: Understanding Remote Method Invocation.

Description:

Remote Method Invocation (RMI) is an API which allows an object to invoke a method on an object that exists in another address space, which could be on the same machine or on a remote machine. Through RMI, object running in a JVM present on a computer (Client side) can invoke methods on an object present in another JVM (Server side). RMI creates a public remote server object that enables client and server side communications through simple method calls on the server object. RMI is used for building distributed application.

The RMI provides remote communication between the applications using two objects **Stub** and **Skeleton**.

Stubs:

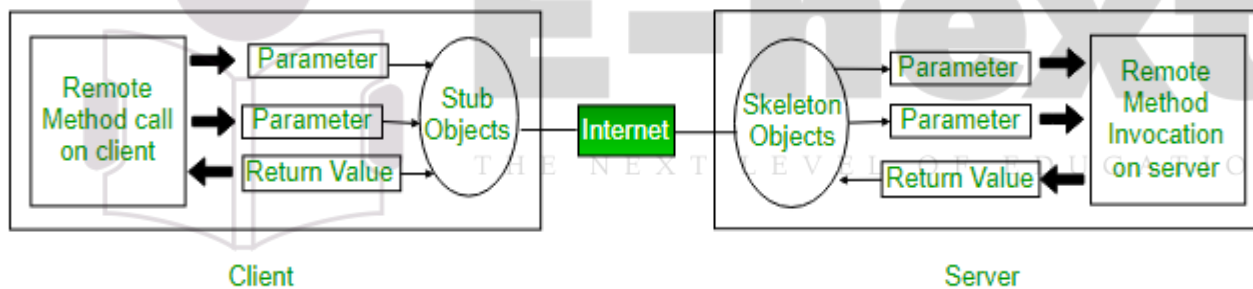
The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object

Skeleton:

The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it.

RMI Working:

Two Intermediate objects are used for the communication between client and server [Stubs and Skeleton].



Working of RMI

Steps to Implement RMI:

Step 1) Defining a Remote Interface.

Step 2) Implementing the Remote Interface.

Step 3) Creating Stub and Skeleton objects from the implementation class using rmic (rmi compiler).

Step 4) Start the rmiregistry using the command start rmiregistry.

Step 5) Create and execute the server application program.

Step 6) Create and execute the client application program.

11) Retrieve time and date function from server to client. This program should display server date and time by implementing RMI.

Source Code:

Filename : MyInterface.java :

```
import java.rmi.*;
public interface MyInterface extends Remote
{
    public String getDate() throws RemoteException;
    public String getTime() throws RemoteException;
}
```

Filename : MyServer.java :

```
import java.rmi.*;
import java.util.*;
import java.text.*;
import java.rmi.server.*;
public class MyServer extends UnicastRemoteObject implements MyInterface
{
    MyServer() throws RemoteException
    {
        super();
    }
    public String getDate() throws RemoteException
    {
        return new SimpleDateFormat("dd/mm/yyyy").format(new Date()).toString();
    }
    public String getTime() throws RemoteException
    {
        return new SimpleDateFormat("hh:mm:ss").format(new Date()).toString();
    }
}
```

Filename : Register.java :

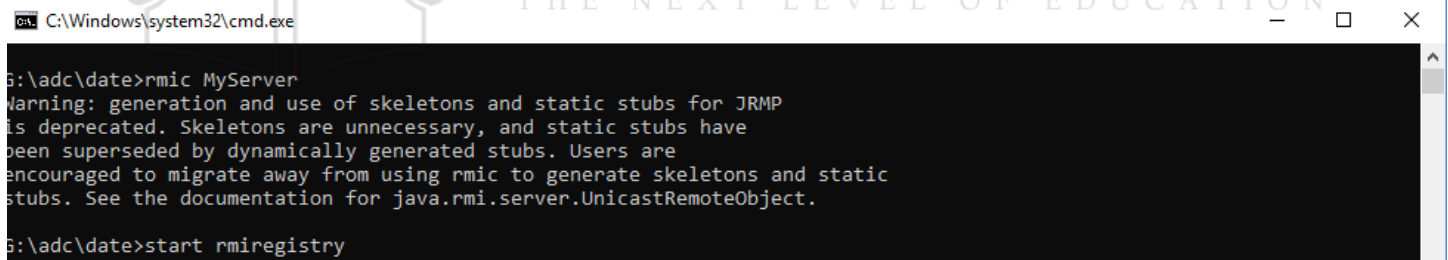
```
import java.rmi.*;
import java.rmi.registry.*;
public class Register
{
    public static void main(String[] args)
    {
        try {
            Registry reg=LocateRegistry.createRegistry(2099);
            MyServer obj=new MyServer();
            Naming.rebind("rmi://localhost:2099/dt",obj);
        }
        catch(Exception e) { }
    }
}
```

Filename : MyClient.java :

```
import java.rmi.*;
public class MyClient
{
    public static void main(String[] args)
    {
        try
        {
            MyInterface obj=(MyInterface)Naming.lookup("rmi://localhost:2099/dt");
            System.out.println("Date is : "+obj.getDate());
            System.out.println("Time is : "+obj.getTime());
        }
        catch(Exception e) { }
    }
}
```

OUTPUT:**1> Compile all 4 programs.**


```
C:\Windows\system32\cmd.exe
G:\adc\date>javac MyInterface.java
G:\adc\date>javac MyServer.java
G:\adc\date>javac Register.java
G:\adc\date>javac MyClient.java
```

2> Create stubs and skeleton object using following command and start rmiregistry


```
C:\Windows\system32\cmd.exe
G:\adc\date>rmic MyServer
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.
G:\adc\date>start rmiregistry
```

3> Run Server Program.


```
C:\Windows\system32\cmd.exe - java Register
G:\adc\date>java Register
```

4> Run Client Program.


```
C:\Windows\system32\cmd.exe
G:\adc\date>java MyClient
Date is : 18/47/2019
Time is : 02:47:09
G:\adc\date>_
```


12) The client should provide the values of a, b. The server will solve the equation $c = (a+b)^2$ & $c = (a+b)^3$ and will give back the value of c. If $a = 2$, $b = 3$ and then return value will be $c = 25$ and $c = 125$.

Source Code:

Filename : MyInterface.java :

```
import java.rmi.*;
public interface MyInterface extends Remote
{
    public int solveEq1(int a,int b) throws RemoteException;
    public int solveEq2(int a,int b) throws RemoteException;
}
```

Filename : MyServer.java :

```
import java.rmi.*;
import java.rmi.server.*;
import java.math.*;
public class MyServer extends UnicastRemoteObject implements MyInterface
{
    public MyServer()throws RemoteException
    {
        super();
    }
    public int solveEq1(int a,int b) throws RemoteException
    {
        int ans= (a*a)+(b*b)+(2*a*b);
        return ans;
    }
    public int solveEq2(int a,int b) throws RemoteException
    {
        int ans= (a*a*a)+(3*a*a*b)+(3*a*b*b)+(b*b*b);
        return ans;
    }
}
```

Filename : Register.java :

```
import java.rmi.*;
import java.rmi.registry.*;
public class Register
{
    public static void main(String[] args)
    {
        try {
            Registry reg=LocateRegistry.createRegistry(2099);
            MyServer obj=new MyServer();
            Naming.rebind("rmi://localhost:2099/equ",obj);
        }
        catch(Exception e) { }
    }
}
```

Filename : MyClient.java :

```

import java.rmi.*;
import java.util.*;
public class MyClient
{
    public static void main(String[] args)
    {
        try {
            int num1,num2,num3,res1,res2;
            MyInterface object=(MyInterface)Naming.lookup("rmi://localhost:2099/eq");
            Scanner in= new Scanner(System.in);
            System.out.println("Enter 1st number");
            num1=in.nextInt();
            System.out.println("Enter 2st number");
            num2=in.nextInt();
            System.out.println("(a+b)2 = "+ object.solveEq1(num1,num2));
            System.out.println("(a+b)3 = "+ object.solveEq2(num1,num2));
        }
        catch(Exception e) { }
    }
}

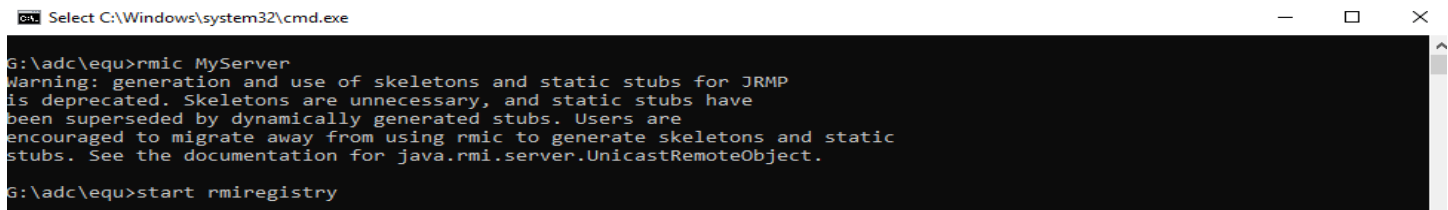
```

OUTPUT:**1> Compile all 4 programs.**


```

C:\Windows\system32\cmd.exe
G:\adc\equ>javac MyInterface.java
G:\adc\equ>javac MyServer.java
G:\adc\equ>javac Register.java
G:\adc\equ>javac MyClient.java

```

2> Create stubs and skeleton object using following command and start rmiregistry


```

C:\Windows\system32\cmd.exe
G:\adc\equ>rmic MyServer
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.
G:\adc\equ>start rmiregistry

```

3> Run Server Program.


```

C:\Windows\system32\cmd.exe - java Register
G:\adc\equ>java Register

```

4> Run Client Program


```

C:\Windows\system32\cmd.exe
G:\adc\equ>java MyClient
Enter 1st number
2
Enter 2st number
3
(a+b)2 = 25
(a+b)3 = 125
G:\adc\equ>

```

13) The client should provide the values of a, b & c. The server will solve the equation ($ax^2 + bx + c = 0$) and will give back the value of x. If a = 1, b = 5 and c = 6 then return value will be x = -2 or x = -3..

Source Code:

Filename : MyInterface.java :

```
import java.rmi.*;
public interface MyInterface extends Remote
{
    public double solveEq1(int a,int b,int c) throws RemoteException;
    public double solveEq2(int a,int b,int c) throws RemoteException;
}
```

Filename : MyServer.java :

```
import java.rmi.*;
import java.rmi.server.*;
import java.math.*;
public class MyServer extends UnicastRemoteObject implements MyInterface
{
    public MyServer()throws RemoteException
    {
        super( );
    }
    public double solveEq1(int a,int b,int c) throws RemoteException
    {
        double ans= ((-b)+ Math.sqrt(Math.abs((b*b)-(4*a*c))))/(2*a);
        return ans;
    }
    public double solveEq2(int a,int b,int c) throws RemoteException
    {
        double ans= ((-b)- Math.sqrt(Math.abs((b*b)-(4*a*c))))/(2*a);
        return ans;
    }
}
```

Filename : Register.java :

```
import java.rmi.*;
import java.rmi.registry.*;
public class Register
{
    public static void main(String[] args)
    {
        try {
            Registry reg=LocateRegistry.createRegistry(2099);
            MyServer obj=new MyServer();
            Naming.rebind("rmi://localhost:2099/eq2",obj);
        }
        catch(Exception e) { }
    }
}
```

Filename : MyClient.java :

```

import java.rmi.*;
import java.util.*;
public class MyClient {
    public static void main(String[] args) throws Exception
    {
        int num1,num2,num3;
        double res1,res2;
        MyInterface object=(MyInterface)Naming.lookup("rmi://localhost:2099/eq2");
        Scanner in= new Scanner(System.in);
        System.out.println("Enter 1st number");
        num1=in.nextInt();
        System.out.println("Enter 2st number");
        num2=in.nextInt();
        System.out.println("Enter 3st number");
        num3=in.nextInt();
        System.out.println("X = "+object.solveEq1(num1,num2,num3));
        System.out.println("X = "+object.solveEq2(num1,num2,num3));
    }
}

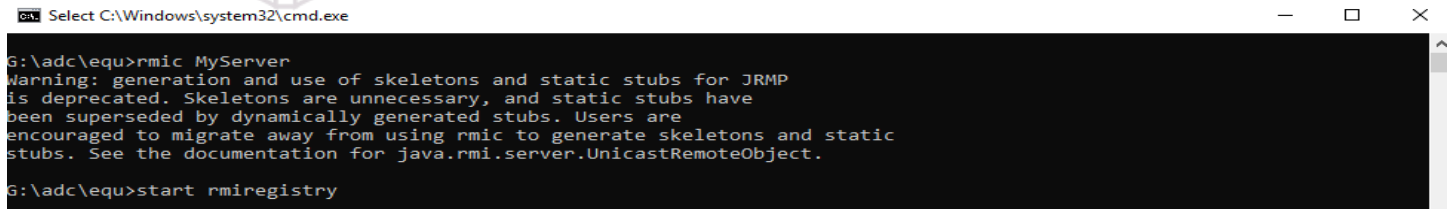
```

OUTPUT:**1> Compile all 4 programs.**


```

C:\Windows\system32\cmd.exe
G:\adc\equ>javac MyInterface.java
G:\adc\equ>javac MyServer.java
G:\adc\equ>javac Register.java
G:\adc\equ>javac MyClient.java

```

2> Create stubs and skeleton object using following command and start rmiregistry


```

C:\Windows\system32\cmd.exe
G:\adc\equ>rmic MyServer
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.
G:\adc\equ>start rmiregistry

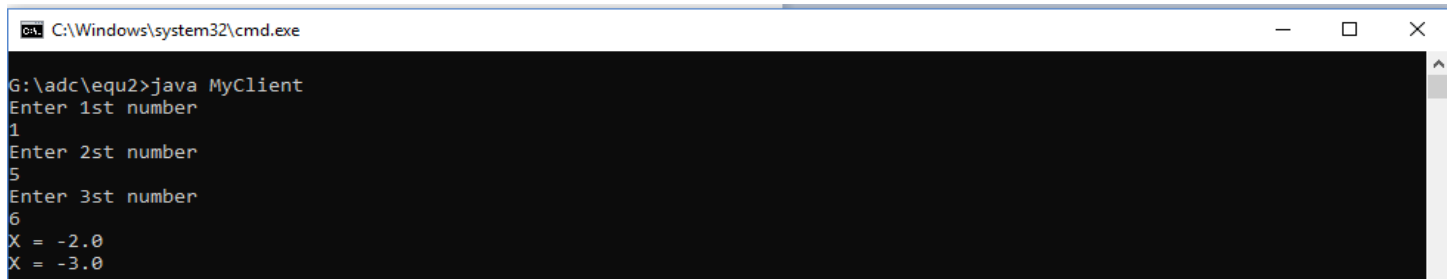
```

3> Run Server Program.


```

C:\Windows\system32\cmd.exe - java Register
G:\adc\equ>java Register

```

4> Run Client Program


```

C:\Windows\system32\cmd.exe
G:\adc\equ>java MyClient
Enter 1st number
1
Enter 2st number
5
Enter 3st number
6
X = -2.0
X = -3.0

```

14) Design a Graphical User Interface to find greatest of two numbers. Implement using RMI.**Source Code:****Filename : MyInterface.java :**

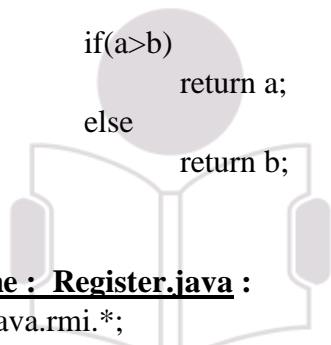
```
import java.rmi.*;
public interface MyInterface extends Remote
{
    public int find(int a,int b) throws RemoteException;
}
```

Filename : MyServer.java :

```
import java.rmi.*;
import java.rmi.server.*;
public class MyServer extends UnicastRemoteObject implements MyInterface
{
    MyServer() throws RemoteException
    {
        super( );
    }
    public int find(int a, int b) throws RemoteException
    {
        if(a>b)
            return a;
        else
            return b;
    }
}
```

Filename : Register.java :

```
import java.rmi.*;
import java.rmi.registry.*;
public class Register
{
    public static void main(String[] args)
    {
        try
        {
            Registry reg=LocateRegistry.createRegistry(2099);
            MyServer obj=new MyServer();
            Naming.rebind("rmi://localhost:2099/g",obj);
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
}
```



E-next

THE NEXT LEVEL OF EDUCATION

Filename : MyClient.java :

```

import java.rmi.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class MyClient extends JFrame implements ActionListener
{
    JTextField tf1,tf2;
    JButton btn;
    JLabel lb,lb1,lb2;
    MyClient()
    {
        tf1=new JTextField(10);
        tf2=new JTextField(10);
        lb=new JLabel("");
        lb1=new JLabel("Enter First Number ");
        lb2=new JLabel("Enter Second Number ");
        btn=new JButton("Submit");
        add(lb1);
        add(tf1);
        add(lb2);
        add(tf2);
        add(btn);
        add(lb);
        btn.addActionListener(this);
    }
    public void actionPerformed(ActionEvent ae)
    {
        try
        {
            MyInterface obj=(MyInterface)Naming.lookup("rmi://localhost:2099/g");
            int a=Integer.parseInt(tf1.getText());
            int b=Integer.parseInt(tf2.getText());
            lb.setText(obj.find(a,b)+" is greatest");
        }
        catch(Exception e){ }
    }
    public static void main(String args[])
    {
        MyClient c=new MyClient();
        c.setLayout(new GridLayout(6,1));
        c.setVisible(true);
        c.setSize(300,300);
    }
}

```

OUTPUT:**1> Compile all 4 programs.**

```

C:\Windows\system32\cmd.exe

G:\adc\greatest>javac MyInterface.java
G:\adc\greatest>javac MyServer.java
G:\adc\greatest>javac Register.java
G:\adc\greatest>javac MyClient.java
G:\adc\greatest>_

```

2> Create stubs and skeleton object using following command and start rmiregistry

```

C:\Windows\system32\cmd.exe

G:\adc\greatest>rmic MyServer
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.

G:\adc\greatest>start rmiregistry

```

3> Run Server Program.

```

C:\Windows\system32\cmd.exe - java Register

G:\adc\greatest>java Register

```

4> Run Client Program

```

C:\Windows\system32\cmd.exe - java MyClient

G:\adc\greatest>java MyClient
_

```

Enter First Number

24

Enter Second Number

35

Submit

35 is greatest

15) Design a Graphical User Interface (GUI) based Basic calculator by implementing RMI.**Source Code:****Filename : MyInterface.java :**

```
import java.rmi.*;
public interface MyInterface extends Remote
{
    public int add(int a,int b) throws RemoteException;
    public int sub(int a,int b) throws RemoteException;
    public int mul(int a,int b) throws RemoteException;
    public int div(int a,int b) throws RemoteException;
}
```

Filename : MyServer.java :

```
import java.rmi.*;
import java.rmi.server.*;

public class MyServer extends UnicastRemoteObject implements MyInterface
{
    MyServer() throws RemoteException
    {
        super();
    }
    public int add(int a, int b) throws RemoteException
    {
        return a+b;
    }
    public int sub(int a, int b) throws RemoteException
    {
        return a-b;
    }
    public int mul(int a, int b) throws RemoteException
    {
        return a*b;
    }
    public int div(int a, int b) throws RemoteException
    {
        return a/b;
    }
}
```


Filename : Register.java :

```

import java.rmi.*;
import java.rmi.registry.*;
public class Register
{
    public static void main(String[] args)
    {
        try {
            Registry reg=LocateRegistry.createRegistry(2099);
            MyServer obj=new MyServer();
            Naming.rebind("rmi://localhost:2099/calc",obj);
        }
        catch(Exception e) { }
    }
}

```

Filename : MyClient.java :

```

import java.rmi.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class MyClient extends JFrame implements ActionListener
{
    JTextField tf;
    JLabel lb;
    JButton b[]=new JButton[16];
    JPanel p1,p2;
    double res=0;
    String op,n1="",n2="";
    MyClient()
    {
        for(int i=0;i<10;i++)
        {
            b[i]=new JButton(""+i);
        }
        b[10]=new JButton("+");
        b[11]=new JButton("-");
        b[12]=new JButton("*");
        b[13]=new JButton("/");
        b[14]=new JButton("=");
        b[15]=new JButton("C");
        lb=new JLabel("");
        tf=new JTextField(15);
        p1=new JPanel();
        p2=new JPanel();
        p1.setLayout(new GridLayout(2,1));
        p2.setLayout(new GridLayout(4,4));
        p1.add(lb);
        p1.add(tf);
    }
}

```

```

for(int i=0;i<16;i++)
{
    b[i].addActionListener(this);
    p2.add(b[i]);
}
add(p1,"North");
add(p2,"Center");
}
public void actionPerformed(ActionEvent ae)
{
    try
    {
        MyInterface obj=(MyInterface)Naming.lookup("rmi://localhost:2099/calc");

        for(int i=0;i<10;i++)
        {
            if(ae.getSource()==b[i])
            {
                tf.setText(tf.getText()+ae.getActionCommand());
            }
        }
        if(ae.getSource()==b[10])
        {
            n1=tf.getText();
            tf.setText("");
            op="+";
        }
        if(ae.getSource()==b[11])
        {
            n1=tf.getText();
            tf.setText("");
            op="-";
        }
        if(ae.getSource()==b[12])
        {
            n1=tf.getText();
            tf.setText("");
            op="*";
        }
        if(ae.getSource()==b[13])
        {
            n1=tf.getText();
            tf.setText("");
            op="/";
        }

        if(ae.getSource()==b[14])
        {

```

```

lb.setText("");
n2=tf.getText();
lb.setText(n1+" "+op+" "+n2);
int a=Integer.parseInt(n1);
int b=Integer.parseInt(n2);
switch(op)
{
    case "+":
        res=obj.add(a,b);
        break;
    case "-":
        res=obj.sub(a,b);
        break;
    case "*":
        res=obj.mul(a,b);
        break;
    case "/":
        res=obj.div(a,b);
        break;
}
tf.setText(String.valueOf(res));
}
if(ae.getSource()==b[15])
{
    tf.setText("");
    lb.setText("");
}
}
catch(Exception e)
{
    System.out.println(e);
}
}
public static void main(String args[])
{
    MyClient c=new MyClient();
    c.setVisible(true);
    c.setSize(300,300);
}
}

```

E-next
THE NEXT LEVEL OF EDUCATION

OUTPUT:**1> Compile all 4 programs.**

```

C:\Windows\system32\cmd.exe
G:\adc\calc>javac MyInterface.java
G:\adc\calc>javac MyServer.java
G:\adc\calc>javac Register.java
G:\adc\calc>javac MyClient.java

```

2> Create stubs and skeleton object using following command and start rmiregistry

```

C:\Windows\system32\cmd.exe
G:\adc\calc>rmic MyServer
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.
G:\adc\calc>start rmiregistry

```

3> Run Server Program.

```

C:\Windows\system32\cmd.exe - java Register
G:\adc\calc>java Register

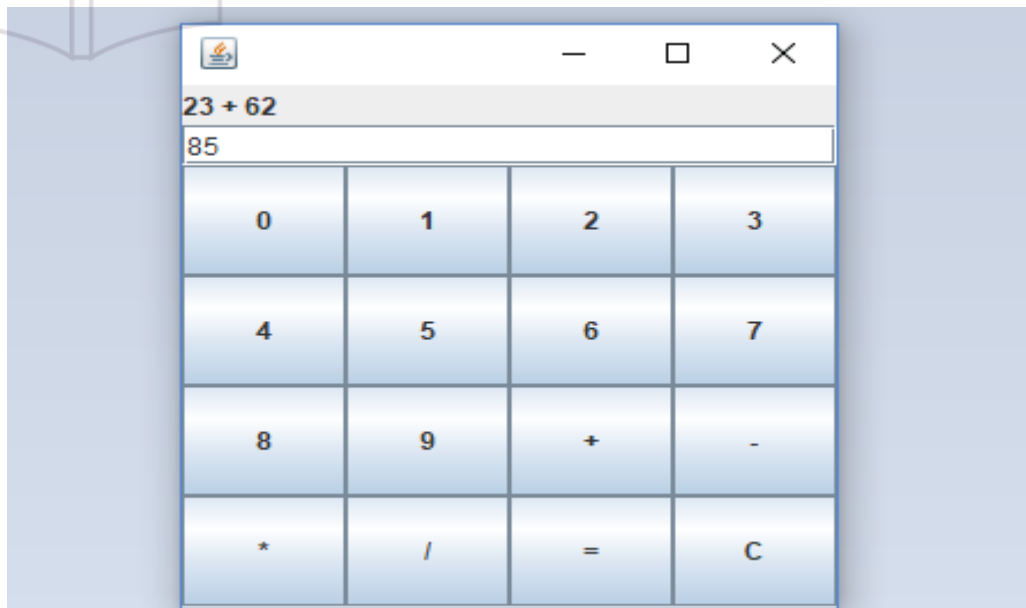
```

4> Run Client Program

```

C:\Windows\system32\cmd.exe - java MyClient
G:\adc\calc>java MyClient

```



Practical No 4

AIM: Implementation of Shared Memory.

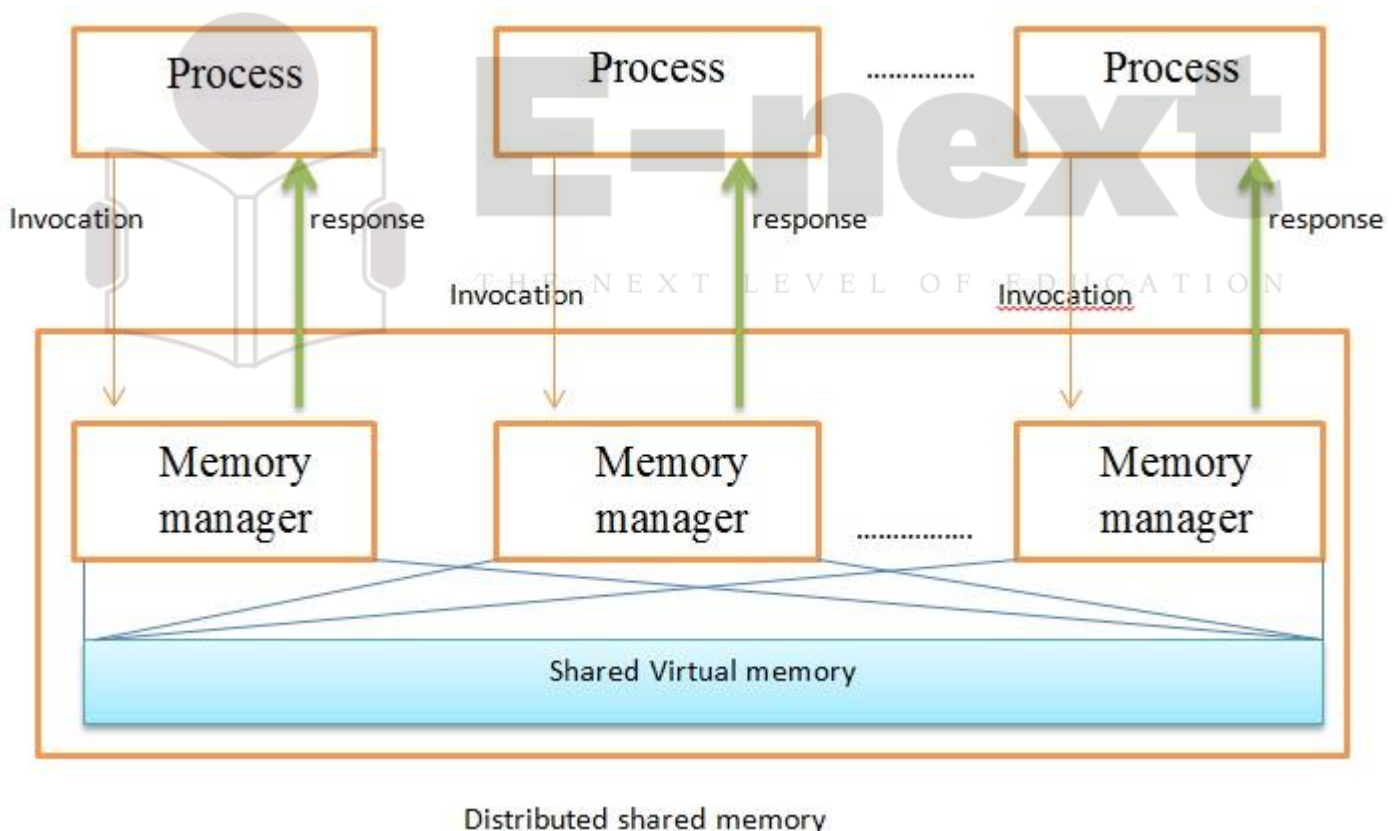
Description:

Memory Management :

Memory management is the process of controlling and coordinating computer memory, assigning portions called blocks to various running program to optimize overall system performance. Memory management resides in hardware ,in the OS (Operating System), and in the programs and applications.

Distributed Shared Memory :

Distributed Shared Memory (DSM) is a resource management component of a distributed operating system that implements the shared memory model in distributed systems, which have no physically shared memory. The shared memory model provides a virtual address space that is shared among all computers in a distributed system.



In DSM, data is accessed from a shared address space similar to the way that virtual memory is accessed. Data moves between secondary and main memory, as well as, between the distributed main memories of different nodes. Ownership of pages in memory starts out in some pre-defined state but changes during the course of normal operation. Ownership changes take place when data moves from one node to another due to an access by a particular process.

16) Write a program to increment counter in Shared Memory .

Source Code:

Filename : SM-Server.java :

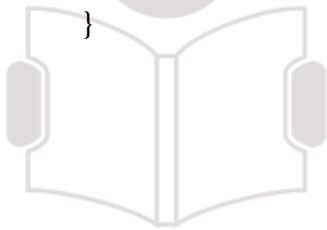
```
import java.io.*;
import java.net.*;
import java.util.*;
public class SM_Server
{
    static int a=50;
    static int count=0;
    public static int getA(PrintStream cout)
    {
        count++;
        cout.println(a);
        return a;
    }
    public void setA(int a)
    {
        this.a = a;
    }
    public static void main(String[] args) throws Exception
    {
        ServerSocket ss=new ServerSocket(2000);
        while ( true )
        {
            Socket sk=ss.accept();
            BufferedReader cin = new BufferedReader(new InputStreamReader(sk.getInputStream()));
            PrintStream cout=new PrintStream(sk.getOutputStream());
            System.out.println("Client from "+sk.getInetAddress().getHostAddress()+" accepted");
            BufferedReader stdin=new BufferedReader(new InputStreamReader(System.in));
            String s=cin.readLine();
            Scanner sc=new Scanner(s);
            String op=sc.next();
            if (op.equalsIgnoreCase("show"))
            {
                getA(cout);
            }
            else
            {
                cout.println("Check Syntax");
                break;
            }
            System.out.println("Count : " +count);
        }
    }
}
```

Filename : SM-Client.java :

```

import java.io.*;
import java.net.*;
import java.util.*;
public class SM_Client
{
    public static void main(String args[]) throws Exception
    {
        Socket sk = new Socket("localhost",2000);
        BufferedReader sin= new BufferedReader(new InputStreamReader(sk.getInputStream()));
        PrintStream sout = new PrintStream(sk.getOutputStream());
        BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));
        String s;
        while(true)
        {
            System.out.print("Client : ");
            s=stdin.readLine();
            sout.println(s);
            s=sin.readLine();
            System.out.println("Answer : " +s);
            break;
        }
    }
}

```



E-next

THE NEXT LEVEL OF EDUCATION

OUTPUT:**File Name: SM_Server.java -> Server**

```

C:\Windows\system32\cmd.exe - java SM_Server
G:\adc\shared memory>javac SM_Server.java
G:\adc\shared memory>java SM_Server
Client from 127.0.0.1 accepted
Count : 1
Client from 127.0.0.1 accepted
Count : 2
Client from 127.0.0.1 accepted
Count : 3

```

File Name : SM_Client.java -> Client 1

```

C:\Windows\system32\cmd.exe
G:\adc\shared memory>javac SM_Client.java
G:\adc\shared memory>java SM_Client
Client : show
Answer : 50
G:\adc\shared memory>

```

File Name : SM_Client.java -> Client 2

```

C:\Windows\system32\cmd.exe
G:\adc\shared memory>javac SM_Client.java
G:\adc\shared memory>java SM_Client
Client : show
Answer : 50
G:\adc\shared memory>

```

File Name : SM_Client.java -> Client 3

```

C:\Windows\system32\cmd.exe
G:\adc\shared memory>javac SM_Client.java
G:\adc\shared memory>java SM_Client
Client : show
Answer : 50
G:\adc\shared memory>

```


Practical No 5

AIM: Understanding Remote Object Communication.

Description:

Remote Object for Database Access:

Pass Remote Object from server to the Client. The Client will receive the stub object (through remote interface) and save it in an object variable with the same type as remote interface. Then the client can access actual object on the server through the variable. Make Use of JDBC and RMI for accessing multiple data access objects.

Here we are using MYSQL Database.

Steps involved in Remote Object for database access through JDBC and RMI.

Prerequisite:

- Download Mysql connector jar file.
- Paste this jar file inside C:\Program Files\Java\jre1.8.0_121\lib\ext folder.
- Create a database inside mysql application.
- Now Create a table and insert few records.

RMI Steps:

Step 1) Defining a Remote Interface.

Step 2) Implementing the Remote Interface.

Step 3) Creating Stub and Skeleton objects from the implementation class using rmic (rmi compiler).

Step 4) Start the rmiregistry using the command start rmiregistry.

Step 5) Create and execute the server application program.

Step 6) Create and execute the client application program.

JDBC Steps while implementing RMI:

Step 1) Import package java.sql.*.

Step 2) Define and loading the driver that would be used for connection.

Class.forName("name of the driver ");

Step 3) Establish the connection.

Connection con=DriverManager.getConnection("url", "username", "password");

Step 4) Create the statement.

Statement st=con.createStatement();

Step 5) Executes the Statements / Query.

st.executeQuery("Select * from table_name");

Step 6) Retrieve the results.

ResultSet rs=st.executeQuery("Select * from table_name");

Step 7) Close the connection.

st.close() and con.close(); .

17) Retrieve the Student information from the College database.**Source Code:****Filename : MyInterface.java :**

```
import java.rmi.*;
public interface MyInterface extends Remote
{
    public String getData() throws RemoteException;
}
```

Filename : MyServer.java :

```
import java.sql.*;
import java.rmi.*;
import java.rmi.server.*;
public class MyServer extends UnicastRemoteObject implements MyInterface
{
    String str=" ";
    MyServer() throws RemoteException
    {
        super();
    }
    public String getData()
    {
        try
        {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection con=DriverManager.getConnection("jdbc:mysql://localhost/adc","root","");
            Statement st=con.createStatement();
            ResultSet rs=st.executeQuery("select * from student");
            while(rs.next())
            {
                str+=rs.getString(1)+" "+rs.getString(2)+" \n ";
            }
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
        return str;
    }
}
```

Filename : Register.java :

```

import java.rmi.*;
import java.rmi.registry.*;
public class Register
{
    public static void main(String[] args)
    {
        try
        {
            Registry reg=LocateRegistry.createRegistry(2099);
            MyServer obj=new MyServer();
            Naming.rebind("rmi://localhost:2099/db",obj);
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
}

```

Filename : MyClient.java :

```

import java.rmi.*;
public class MyClient
{
    public static void main(String args[]) throws Exception
    {
        MyInterface obj=(MyInterface)Naming.lookup("rmi://localhost:2099/db");
        String s=obj.getData();
        System.out.println(s[i]);
    }
}

```

OUTPUT:

5> Compile all 4 programs.

```
C:\Windows\system32\cmd.exe

G:\adc\db>javac MyInterface.java
G:\adc\db>javac MyServer.java
G:\adc\db>javac Register.java
G:\adc\db>javac MyClient.java
G:\adc\db>_
```

6> Create stubs and skeleton object using following command and start rmiregistry

```
C:\Windows\system32\cmd.exe

G:\adc\db>rmic MyServer
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.

G:\adc\db>start rmiregistry
G:\adc\db>
```

7> Run Server Program.

```
C:\Windows\system32\cmd.exe - java Register

G:\adc\db>java Register
```

8> Run Client Program.

```
C:\Windows\system32\cmd.exe

G:\adc\db>java MyClient
10 ashish
20 anil
30 vinay
G:\adc\db>
```

18) Retrieve the list of books available in the library.**Source Code:****Filename : MyInterface.java :**

```
import java.rmi.*;
public interface MyInterface extends Remote
{
    public String[] getData() throws RemoteException;
}
```

Filename : MyServer.java :

```
import java.sql.*;
import java.rmi.*;
import java.rmi.server.*;
public class MyServer extends UnicastRemoteObject implements MyInterface
{
    int count=0,k=0;
    String str[ ];
    MyServer() throws RemoteException
    {
        super();
    }
    public String[] getData()
    {
        try
        {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection con=DriverManager.getConnection("jdbc:mysql://localhost/adc","root","");
            Statement st=con.createStatement();
            ResultSet rs1=st.executeQuery("select books from library");
            while(rs1.next())
            {
                count++;
            }
            str=new String[count];
            ResultSet rs=st.executeQuery("select books from library");
            for(int i=0;i<count;i++)
            {
                rs.next();
                str[i]=rs.getString(1);
            }
        }
    }
}
```

```

        catch(Exception e)
        {
            System.out.println(e);
        }
        return str;
    }
}

```

Filename : Register.java :

```

import java.rmi.*;
import java.rmi.registry.*;
public class Register
{
    public static void main(String[] args)
    {
        try
        {
            Registry reg=LocateRegistry.createRegistry(2099);
            MyServer obj=new MyServer();
            Naming.rebind("rmi://localhost:2099/db",obj);
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
}

```

Filename : MyClient.java :

```

import java.rmi.*;
public class MyClient
{
    public static void main(String args[]) throws Exception
    {
        MyInterface obj=(MyInterface)Naming.lookup("rmi://localhost:2099/db");
        String s[]=obj.getData();
        int size=s.length;
        for(int i=0;i<size;i++)
        {
            System.out.println(s[i]);
        }
    }
}

```

OUTPUT:**1> Compile all 4 programs.**

```

C:\Windows\system32\cmd.exe
G:\adc\db>javac MyInterface.java
G:\adc\db>javac MyServer.java
G:\adc\db>javac Register.java
G:\adc\db>javac MyClient.java
G:\adc\db>_

```

2> Create stubs and skeleton object using following command and start rmiregistry

```

C:\Windows\system32\cmd.exe
G:\adc\db>rmic MyServer
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.
G:\adc\db>start rmiregistry
G:\adc\db>

```

3> Run Server Program.

```

C:\Windows\system32\cmd.exe - java Register
G:\adc\db>java Register

```

1> Run Client Program.

```

Select C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.17134.706]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\4v1Sh3k>g:
G:\>cd adc
G:\adc>cd db
G:\adc\db>java MyClient
Java
DBMS
Operating System
Digital marketing
Asp.net
G:\adc\db>

```

19) Retrieve the MTNL billing information from the MTNL database.**Source Code:****Filename : MyInterface.java :**

```
import java.rmi.*;
public interface MyInterface extends Remote
{
    public String[] getData() throws RemoteException;
}
```

Filename : MyServer.java :

```
import java.sql.*;
import java.rmi.*;
import java.rmi.server.*;
public class MyServer extends UnicastRemoteObject implements MyInterface
{
    int count=0,k=0;
    String str[ ];
    MyServer() throws RemoteException
    {
        super();
    }
    public String[] getData()
    {
        try
        {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection con=DriverManager.getConnection("jdbc:mysql://localhost/adc","root","");
            Statement st=con.createStatement();
            ResultSet rs1=st.executeQuery("select * from MtnlBill");
            while(rs1.next())
            {
                count++;
            }
            str=new String[count];
            ResultSet rs=st.executeQuery("select * from MtnlBill");
            ResultSetMetaData rsmd=rs.getMetaData();
            int col=rsmd.getColumnCount();
            while(rs.next())
            {
                str[k]="";
                for(int i=1;i<=col;i++)
                {
                    str[k]=str[k]+rs.getString(i)+" ";
                }
                k++;
            }
        }
    }
}
```



```

        catch(Exception e)
        {
            System.out.println(e);
        }
        return str;
    }
}

```

Filename : Register.java :

```

import java.rmi.*;
import java.rmi.registry.*;
public class Register
{
    public static void main(String[] args)
    {
        try
        {
            Registry reg=LocateRegistry.createRegistry(2099);
            MyServer obj=new MyServer();
            Naming.rebind("rmi://localhost:2099/db",obj);
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
}

```

Filename : MyClient.java :

```

import java.rmi.*;
public class MyClient
{
    public static void main(String args[]) throws Exception
    {
        MyInterface obj=(MyInterface)Naming.lookup("rmi://localhost:2099/db");
        String s[]=obj.getData();
        int size=s.length;
        for(int i=0;i<size;i++)
        {
            System.out.println(s[i]);
        }
    }
}

```

OUTPUT:**1> Compile all 4 programs.**

```

C:\Windows\system32\cmd.exe
G:\adc\db>javac MyInterface.java
G:\adc\db>javac MyServer.java
G:\adc\db>javac Register.java
G:\adc\db>javac MyClient.java
G:\adc\db>_

```

2> Create stubs and skeleton object using following command and start rmiregistry

```

C:\Windows\system32\cmd.exe
G:\adc\db>rmic MyServer
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.
G:\adc\db>start rmiregistry
G:\adc\db>

```

3> Run Server Program.

```

C:\Windows\system32\cmd.exe - java Register
G:\adc\db>java Register

```

4> Run Client Program.

```

C:\Windows\system32\cmd.exe
G:\adc\db>java MyClient
9820947884 222342312 2019-01-20 00:00:00.0 2019-01-30 00:00:00.0 252323 2000
9862326743 222232442 2019-07-30 00:00:00.0 2019-08-30 00:00:00.0 213324 4120
9364527324 222234332 2019-03-10 00:00:00.0 2019-03-30 00:00:00.0 342323 5650
G:\adc\db>

```

Practical No 6

AIM: Understanding Enterprise Java Beans (EJB).

Description:

Enterprise Java Beans:

EJB is an acronym for enterprise java bean. It is a specification provided by Sun Microsystems to develop secured, robust and scalable distributed applications.

EJB application is deployed on the server, so it is called server side component also.

EJB is like COM (Component Object Model) provided by Microsoft. But, it is different from Java Bean, RMI and Web Services.

Types of Enterprise java Beans :

1) Session Bean:

Session bean encapsulates business logic only, it can be invoked by local, remote and webservice client.

There are 3 types of Session Bean.

i) **Stateless Session Bean:** It doesn't maintain state of a client between multiple method calls.

ii) **Stateful Session Bean:** It maintains state of a client across multiple requests.

iii) **Singleton :** One instance per application, it is shared between clients and supports concurrent access.

2) Message Driven Bean:

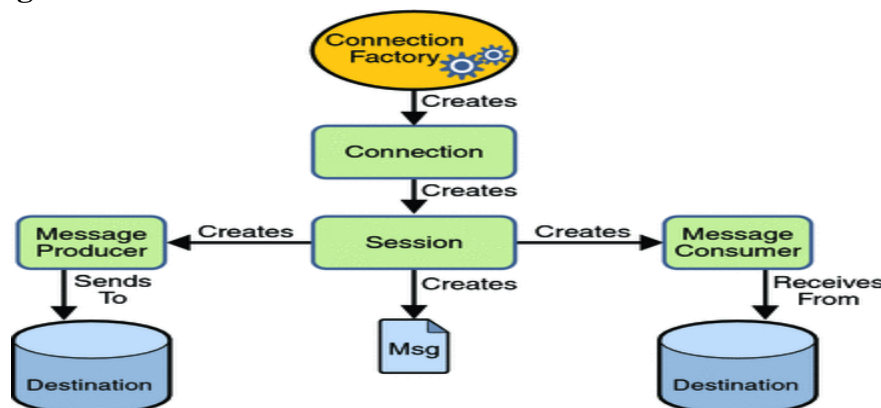
A message driven bean (MDB) is a bean that contains business logic. But, it is invoked by passing the message. So, it is like JMS Receiver. MDB asynchronously receives the message and processes it.

Java Messaging service (JMS):

JMS (Java Message Service) is an API that provides the facility to create, send and read messages. It provides loosely coupled, reliable and asynchronous communication.

JMS is also known as a messaging service.

JSM Programming Model:



3) Entity Bean :

Entity bean represents the persistent data stored in the database. It is a server-side component.

In EJB 2.x, there were two types of entity beans: **bean managed persistence (BMP)** and **container managed persistence (CMP)**.

Since EJB 3.x, it is deprecated and replaced by JPA (Java Persistence API) .

1> Session Bean Example .

i) **Write a program for Basic arithmetic operations implemented in Session Bean (stateless).**

Step 1 > Create a new Project [Enterprise Application] names as SessionDemo.

Step 2 > Now Right click on 'SessionDemo-ejb' and select new Session Bean.

Step 3> Enter name of bean as 'CalcBean' and name of the package as 'sessionBean' and select the Session

Type as Stateful and check the 'Local' checkbox.

Step 4> Now you will see 2 file will be created one is an Interface and another is an Implementation class.

FileName : CalcBeanLocal.java:

```
package sessionBean;
import javax.ejb.Local;
@Local
public interface CalcBeanLocal
{
    int add(int a,int b);
    int sub(int a,int b);
    int mul(int a,int b);
    int div(int a,int b);
}
```

File Name : CalcBean.java :

```
package sessionBean;
import javax.ejb.Stateless;
@Stateless
public class Calculator implements CalculatorLocal {

    public int add(int a, int b) {
        return a+b;
    }
    public int sub(int a, int b) {
        return a-b;
    }
    public int mul(int a, int b) {
        return a*b;
    }
    public int div(int a, int b) {
        return a/b;
    }
}
```

E-next

THE NEXT LEVEL OF EDUCATION

Step 5 > Now Inside 'SessionDemo-war' > webpages > index.html.

File Name : index.html:

```
<html>
<body>
    <form method="post" action="ProcessC">
        <table bgcolor="orange" align="center" cellpadding="10" cellspacing="10">
            <tr><td> Enter First Number : <td><input type="number" name="n1">
            <tr><td>Enter Second Number : <td><input type="number" name="n2">
            <tr align="center"><td colspan="2"><input type="submit" value="A D D" name="add">
            <input type="submit" value="S U B" name="sub">
            <input type="submit" value="M U L" name="mul">
            <input type="submit" value="D I V" name="div">
        </table>
    </form>
</body>
</html>
```

Step 6 > Now Inside source package create a Servlet named as 'ProcessC.java'

File Name : Process.java

```
import sessionBean.CalcbeanLocal;
import java.io.*;
import javax.sessionBean.EJB;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;
@WebServlet(urlPatterns = {"/Arithmetic"})
public class ProcessC extends HttpServlet {
    @EJB
    CalcBeanLocal cal;

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            out.println("<html>");
            out.println("<body>");
            int a=Integer.parseInt(request.getParameter("n1"));
            int b=Integer.parseInt(request.getParameter("n2"));
            String add=request.getParameter("add");
            String sub=request.getParameter("sub");
            String mul=request.getParameter("mul");
            String div=request.getParameter("div");
```

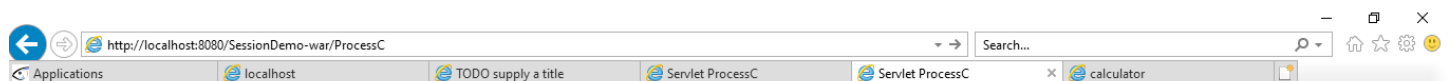
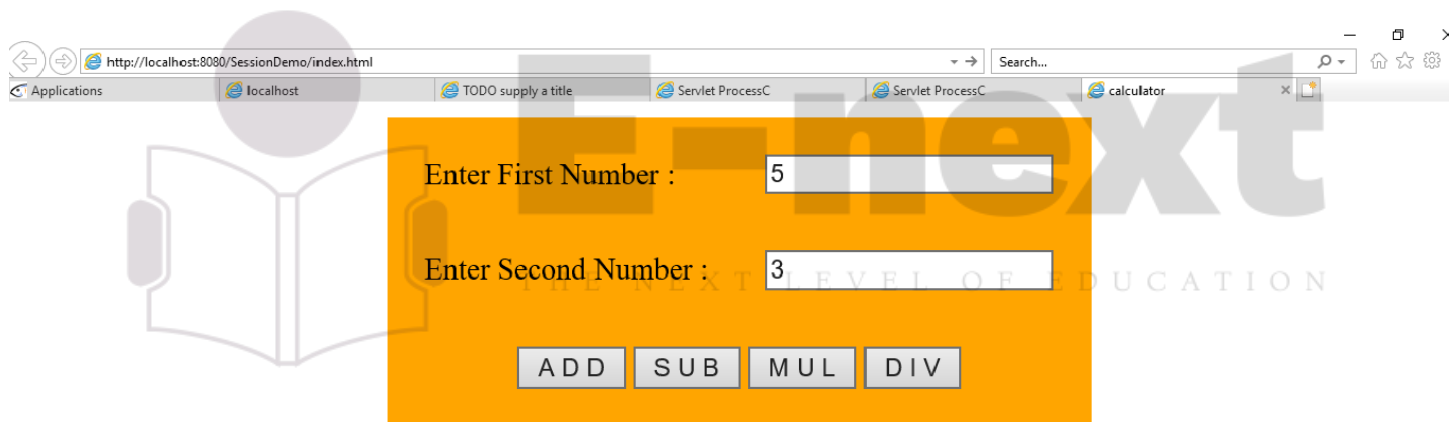
```

    if(add!=null)
        out.println("<h1>Addition is : "+cal.add(a, b)+"</h1>");
    if(sub!=null)
        out.println("<h1>Subtraction is : "+cal.sub(a, b)+"</h1>");
    if(mul!=null)
        out.println("<h1>Multiplication is : "+cal.mul(a, b)+"</h1>");
    if(div!=null)
        out.println("<h1>Division is : "+cal.div(a, b)+"</h1>");
    out.println("</body>");
    out.println("</html>");
}
}
}

```

Step 7 > Now Deploy the 'SessionDemo' project and then click on Run.

OUTPUT:



Addition is : 8

ii) Write a Program to add two number implemented in Session Bean (stateful).

Step 1 > Create a new Project [Enterprise Application] names as SessionDemo.

Step 2 > Now Right click on 'SessionDemo-ejb' and select new Session Bean.

Step 3> Enter name of bean as 'CalcBean' and name of the package as 'sessionBean' and select the Session

Type as Stateful and check the 'Local' checkbox.

Step 4> Now you will see 2 file will be created one is an Interface and another is Implementation class.

Step 5 > Now Inside 'SessionDemo-war' > webpages > index.html.

Step 6 > Now Inside source package create a Servlet named as 'ProcessC.java'

Step 7 > Now Deploy the 'SessionDemo' project and then click on Run.

File Name : CalcBeanLocal.java:

```
package sessionbean;
import javax.ejb.Local;
@Local
public interface CalcBeanLocal
{
    Integer add(int a, int b);
}
```

File Name : ClacBean.java :

```
package sessionbean;
import javax.ejb.Stateful;
@Stateful
public class CalcBean implements CalcBeanLocal
{
    @Override
    public Integer add(int a, int b)
    {
        return (a+b);
    }
}
```

E-next

THE NEXT LEVEL OF EDUCATION

File Name : index.html:

```

<html>
|<body>
|   <form method="post" action="ProcessC">
|       <input type="text" name="t1">
|       <input type="text" name="t2">
|       <input type="submit" name="add" value="add">
|   </form>
|</body>
</html>

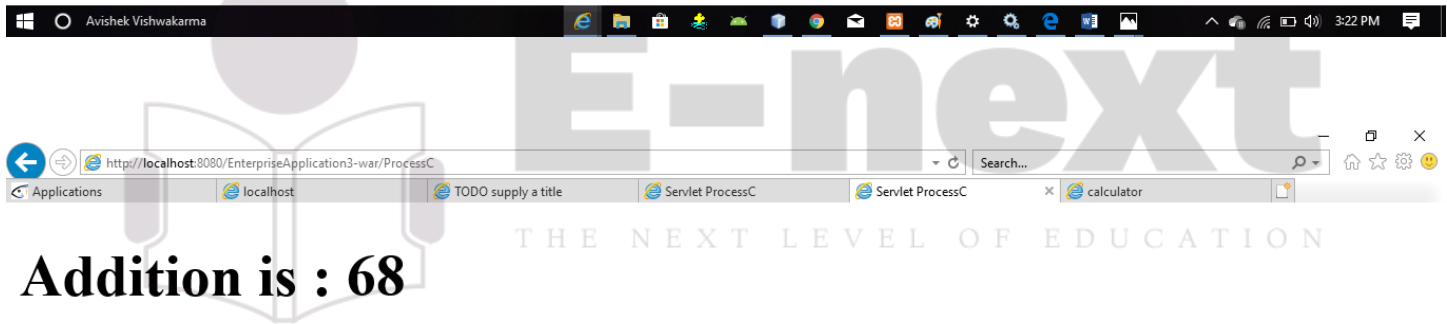
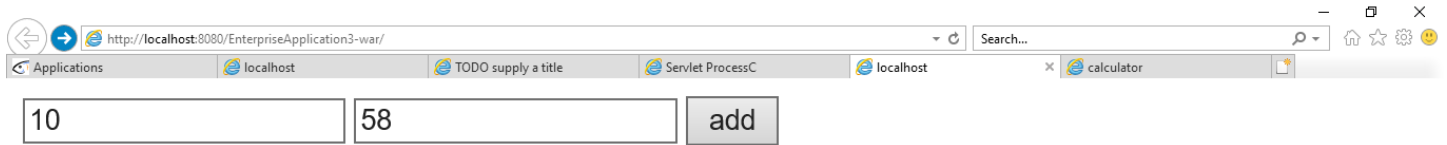
```

File Name : ProcessC.java:

```

import java.io.IOException;
import java.io.PrintWriter;
import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import sessionbean.CalcBeanLocal;
public class ProcessC extends HttpServlet {
    @EJB
    CalcBeanLocal c;
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet ProcessC</title>");
            out.println("</head>");
            out.println("<body>");
            int a=Integer.parseInt(request.getParameter("t1"));
            int b=Integer.parseInt(request.getParameter("t2"));
            out.println("<h1>Addition is : "+c.add(a, b)+"</h1>");
            out.println("</body>");
            out.println("</html>");
        }
    }
}

```


OUTPUT:

2> Message Driven Bean Example.

i) Write a program to demonstrate Message Driven Bean.

Step 1> Create 2 JNDI (here we are using glassfish server).

1> Connection Factory.

2> Destination Resource

Now Creating connection factory and destination resource.

1.1> Open Netbeans and Go to Services Tab.

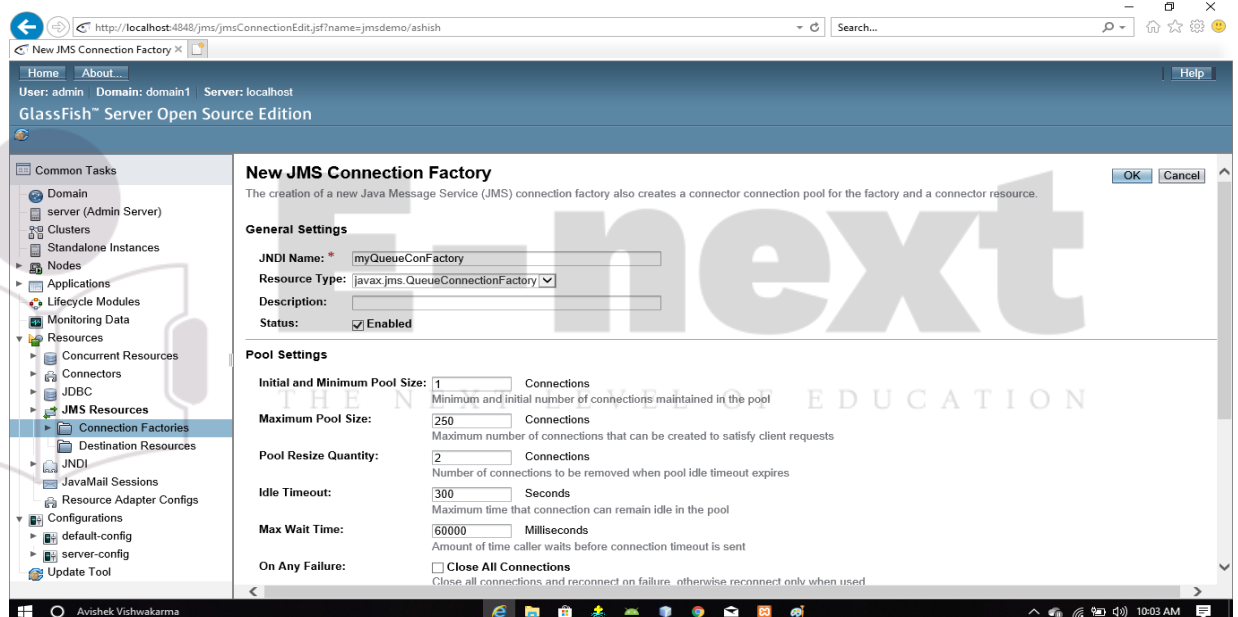
1.2> Inside Service tab Expand the Server Directory and you will find glassfish server.

1.3> Right Click on glassfish server and click on Start.

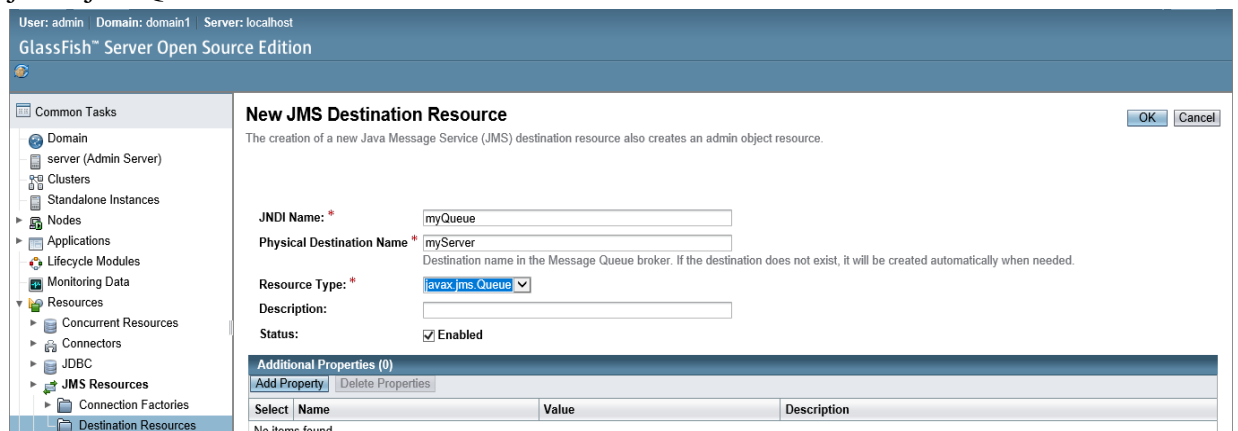
1.4> Now Open any browser and inside url type `http://localhost:4848`

1.5> Now glassfish server admin console will be open.

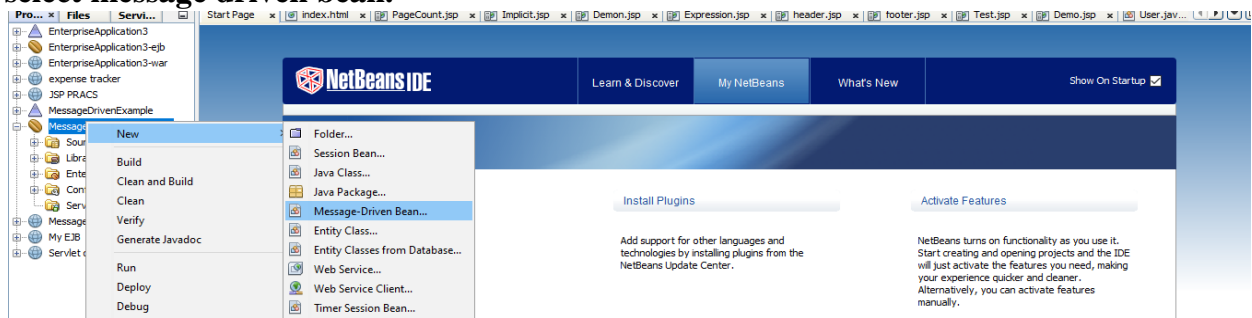
1.6> Click on the **JMS Resource -> Connection Factories -> New**, now write the JNDI name as `myQueueConFactory` and select the Resource Type as `QueueConnectionFactory` then click on ok button.



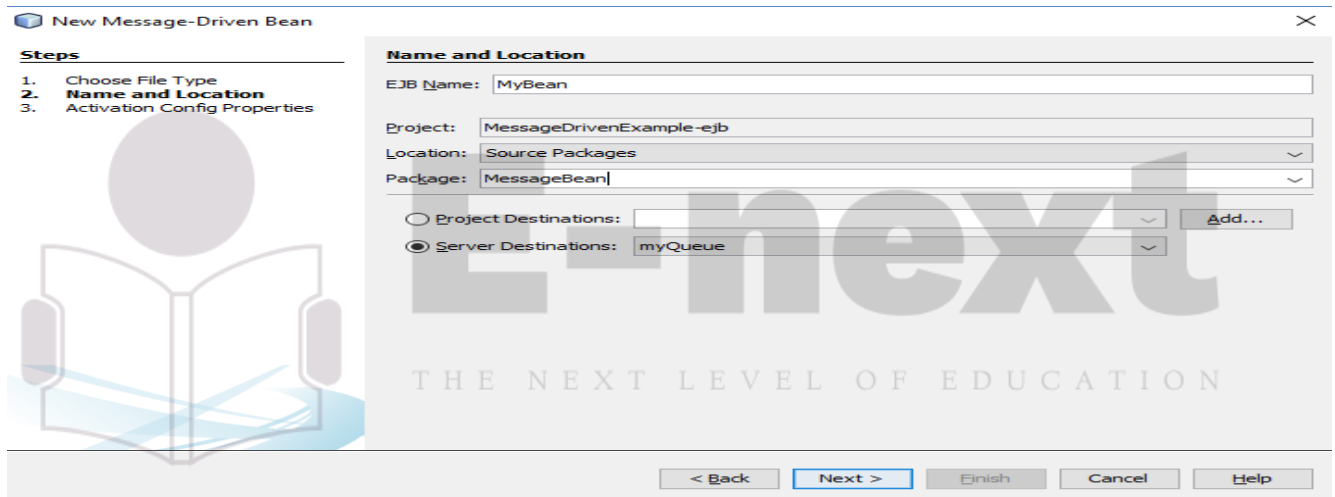
1.7> Now Click on the **JMS Resource -> Destination Resources -> New**, now write the JNDI name as `myQueue` and physical destination name as `myServer` and resource type will be `javax.jms.Queue` then click on ok button.



Step 2> Now Create an new Project i.e. Enterprise Application named as “MessageDrivenExample” and now Create a Message driven Bean by right clicking on ‘MessageDrivenExample-ejb’ and select message driven bean.



Step 3> Now a dialog box will open and give EJB Name as ‘MyBean’ and package name as Messagebean. And change the radio button from Project destination to Server Destination and make sure your Destination resource will be selected that u created in step 1 and now click on next and the Finish.



Now a MyBean.java file will open. And write the following code.

Filename: MyBean.java :

```
package MessageBean;
import javax.ejb.ActivationConfigProperty;
import javax.ejb.MessageDriven;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.TextMessage;
```

```
@MessageDriven(activationConfig = {
    @ActivationConfigProperty(propertyName = "destinationLookup", propertyValue = "myQueue"),
    @ActivationConfigProperty(propertyName = "destinationType", propertyValue = "javax.jms.Queue")
})
public class MyBean implements MessageListener
{
    public MyBean() {
```

```

}
@Override
public void onMessage(Message message)
{
    try
    {
        TextMessage msg=(TextMessage) message;
        System.out.println("Message is : "+msg.getText());
    }
    catch(Exception e){ }
}
}

```

Step 4> Now create client application under MessageDrivenExample-war directory.

Inside **web-pages** create Index.html(already present). And under **Source package** create a servlet named as 'MyServlet'.

File Name : Index.html :

```

<html>
<head>
    <title>TODO supply a title</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
<body>
    <form method="post" action="MyServlet">
        Enter your message : <input type="text" name="txt1">
        <input type="submit" value="SEND">
    </form>
</body>
</html>

```

File Name : MyServlet.java :

```

import javax.jms.*;
import java.io.PrintWriter;
import javax.annotation.Resource;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class MyServlet extends HttpServlet {

    @Resource(mappedName = "myQueue")
    private Queue myQueue;
    @Resource(mappedName = "myQueueConnectionFactory")
    private ConnectionFactory queue;

```

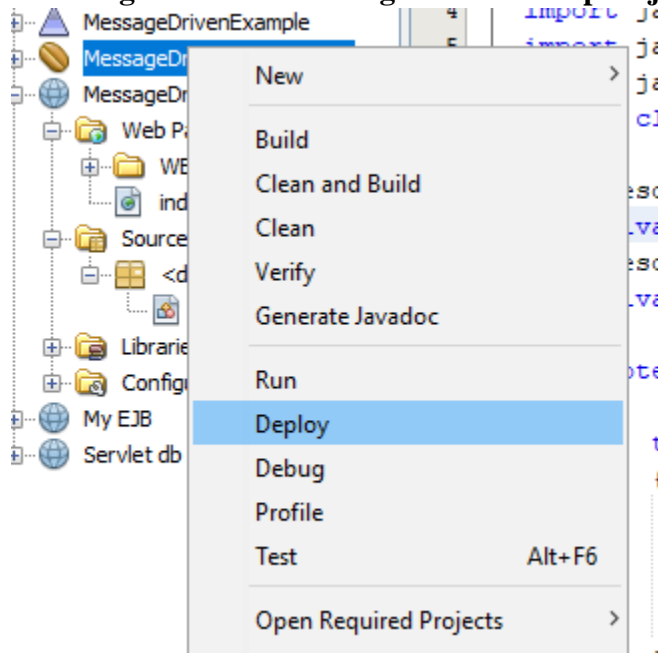
```

protected void doPost(HttpServletRequest request, HttpServletResponse response)
{
    try
    {
        PrintWriter out=response.getWriter();
        String msg=request.getParameter("txt1");
        sendJMSMessageToMyQueue(msg);
        out.println("Your message is queued");
    }
    catch(Exception e){ }
}

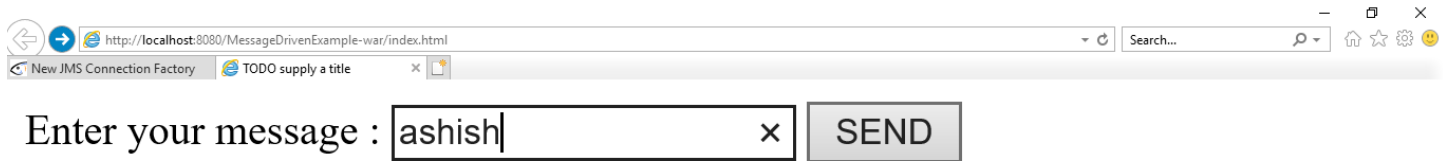
private void sendJMSMessageToMyQueue(String messageData)
{
    try
    {
        Connection con=queue.createConnection();
        Session s=con.createSession();
        MessageProducer mp=s.createProducer(myQueue);
        TextMessage tm=s.createTextMessage();
        tm.setText(messageData);
        mp.send(tm);
    }
    catch(Exception e)
    {
        System.out.println("Error");
    }
}
}

```

Step 5> Now right Click on MessageDrivenExample-ejb and select deploy.



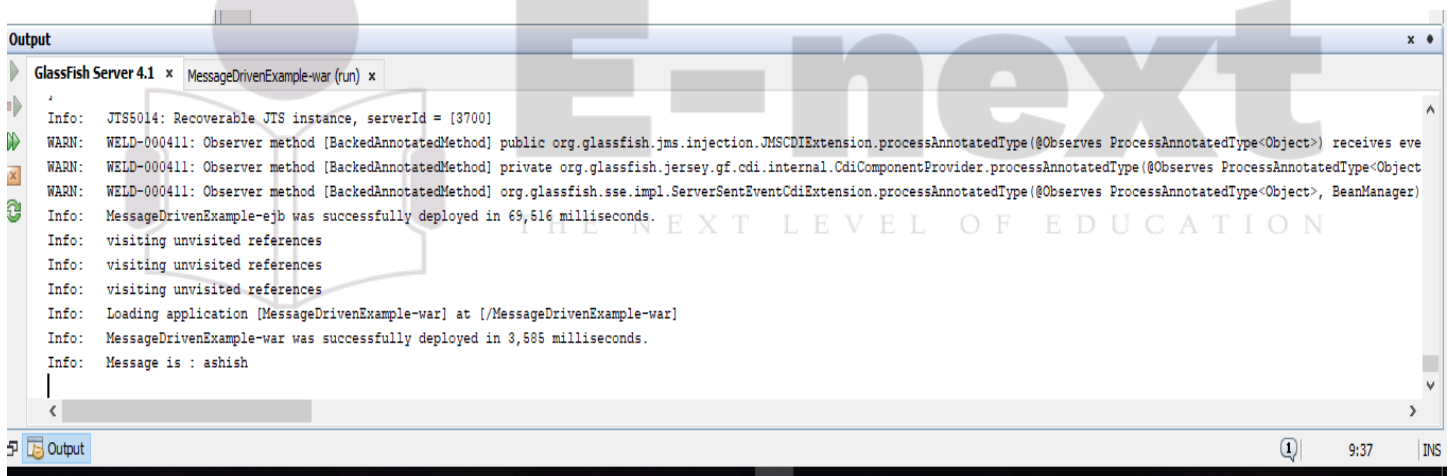
Step 6> Now Right Click Click on Index.html and Select Run.



➔ Now Click On SEND button



➔ Now Go to Netbean and under Output section inside glassfish server you will see the following output.

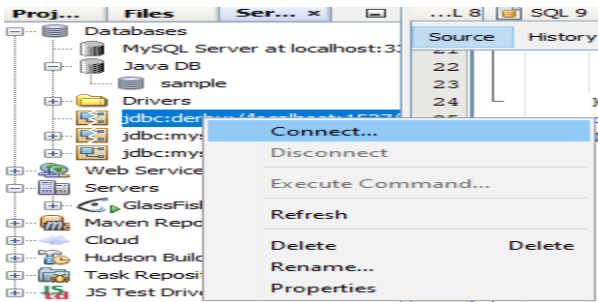


3> Entity Bean Example.

i) Write a program to demonstrate Entity Bean.

Prerequisites:

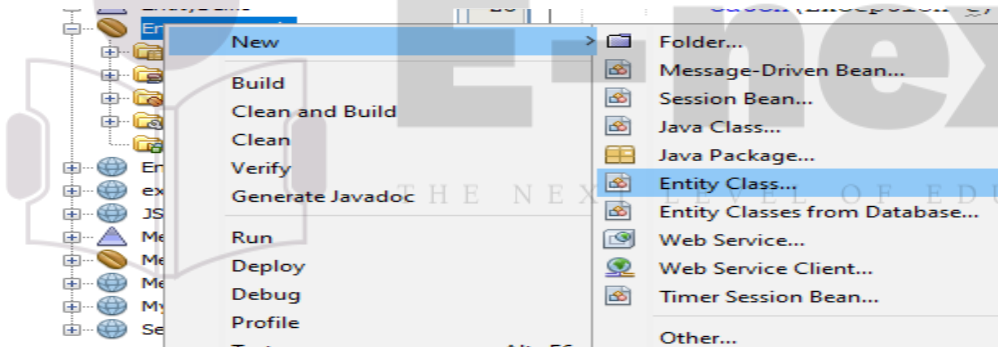
- Open Netbeans and under Services tab expand the database directory and right click on jdbc:derby://localhost:1527/sample and select Connect.



- Now under Services tab > Servers > Glassfish > right click on glassfish server and select start.

Step 1> Create a new Enterprise Application and named as “EntityDemo”.

Step 2> Right Click on ‘EntityDemo-ejb’ and select New > Entity class.



Step 3> A dialog will open and write entity name as Employee and package name as emp and click on next. Then a window will open and under data source dropdown list select jdbc/sample.

Step 4> Now you will see a file will be created as Employee.java and inside Employee.java add some

Code . i.e. declare 2 variable name and age and right click on page and select insert code
And inside that select ‘Generate Getter and setter’ and click on next and finish.

File Name : Employee.java:

```
package emp;
import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
```

```

import javax.persistence.Id;

@Entity
public class Employee implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String name;
    private String age;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getAge() {
        return age;
    }
    public void setAge(String age) {
        this.age = age;
    }
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    @Override
    public int hashCode() {
        int hash = 0;
        hash += (id != null ? id.hashCode() : 0);
        return hash;
    }
    @Override
    public boolean equals(Object object) {
        if (!(object instanceof Employee)) {
            return false;
        }
        Employee other = (Employee) object;
        if ((this.id == null && other.id != null) || (this.id != null && !this.id.equals(other.id))) {
            return false;
        }
        return true;
    }
}

```

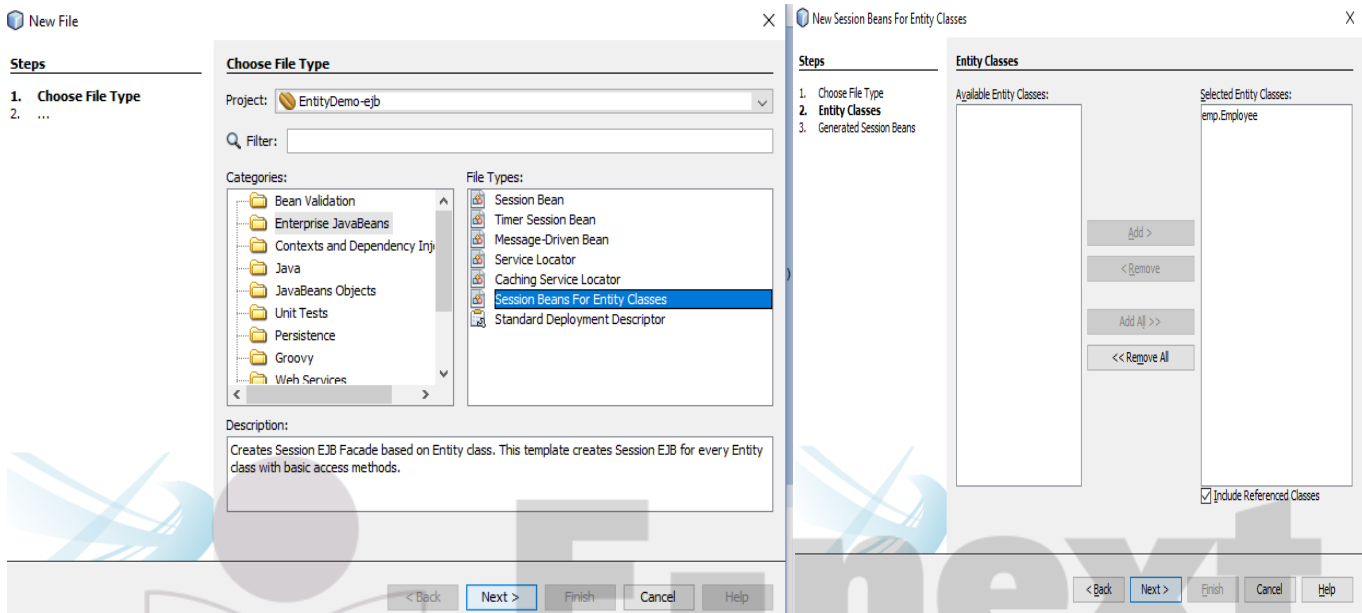


```

@Override
public String toString() {
    return "emp.Employee[ id=" + id + " ]";
}
}

```

Step 5> Now Right click on 'EntityDemo-ejb' and select 'Session Bean for Entity class' and then click on Next then click on All all and then click next.



Step 6> Now a dialog will open and select the Checkbox 'Local' and create interface.

Step 7> Now you will see 3 extra file will be created inside emp package.

Step 8> Now Inside EntityDemo-war > web-pages create index.html(already present) and a servlet File inside source package.

File Name Index.html:

```

<html>
<head>
    <title>TODO supply a title</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
<body>
    <form method="post" action="MyServlet">
        Enter your Name : <input type="text" name="txt1"><br>
        Enter your Age : <input type="text" name="txt2"><br>
        <input type="submit" value="Submit">
    </form>
</body>
</html>

```

File Name : MyServlet.java

```

package ent;
import emp.Employee;
import emp.EmployeeFacadeLocal;
import java.io.IOException;
import java.io.PrintWriter;
import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class MyServlet extends HttpServlet {

    @EJB
    private EmployeeFacadeLocal employeeFacade;

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
            out.println("</head>");
            Employee obj=new Employee();
            obj.setName(request.getParameter("txt1"));
            obj.setAge(request.getParameter("txt2"));
            employeeFacade.create(obj);
            out.println("<body>");
            out.println("<h1> User craeted Successfully</h1>");
            out.println("</body>");
            out.println("</html>");
        }
    }
}

```

Step 9> Deploy the Project and then Run the Project.

OUTPUT:

Enter your Name :

Enter your Age :

User created Successfully

➔ Now again go back to NetBeans and under Services tab > jdbc:derby:localhost > APP > Tables
You will see a table named Employee is created.

Connection: jdbc:derby:localhost:1527/sample [app on APP]

```
select * from APP.EMPLOYEE;
```

select * from APP.EMPLOYEE...

#	ID	AGE	NAME
1		1.22	Ashish

Practical No 7

AIM: Implementation of Mutual Exclusion using Token Ring technique.

Description:

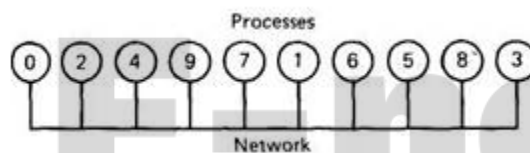
Mutual Exclusion :

Mutual exclusion is a concurrency control property which is introduced to prevent race conditions. It is the requirement that a process can not enter its critical section while another concurrent process is currently present or executing in its critical section i.e only one process is allowed to execute the critical section at any given instance of time.

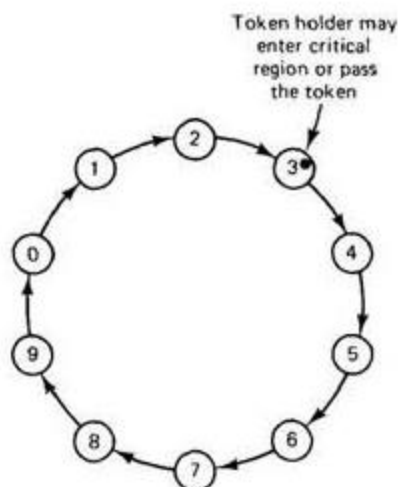
Token Ring Technique:

Token Ring algorithm achieves mutual exclusion in a distributed system by creating a bus network of processes. A logical ring is constructed with these processes and each process is assigned a position in the ring. Each process knows who is next in line after itself.

An Unordered Group of Processes on network:



A Logical Ring constructed in software:



20) Write a program to demonstrate token ring technique.

Source Code:

Filename : TokenServer.java :

```
import java.net.*;
import java.io.*;
class TokenServer
{
    public static DatagramSocket ds;
    public static DatagramPacket dp;
    public static void main(String[] args)throws Exception
    {
        ds=new DatagramSocket(1000);
        while(true)
        {
            byte buff[]=new byte[1024];
            ds.receive(dp=new DatagramPacket(buff,buff.length));
            String str=new String(dp.getData(),0,dp.getLength());
            System.out.println("Message from " +str);
        }
    }
}
```

Filename : TokenClient1.java :

```
import java.net.*;
import java.io.*;
class TokenClient1
{
    public static DatagramSocket ds;
    public static DatagramPacket dp;
    public static BufferedReader br;
    public static void main(String args[])throws Exception
    {
        boolean hasToken=true;
        ds=new DatagramSocket(100);
        while(true)
        {
            if(hasToken==true)
            {
                System.out.println("Do you want to write data...(yes/no)");
                br=new BufferedReader(new InputStreamReader(System.in));
                String ans=br.readLine();
                if(ans.equalsIgnoreCase("yes"))
                {
                    System.out.println("ready to write");
                    System.out.println("enter the data");
                }
            }
        }
    }
}
```

```

        br=new BufferedReader(new InputStreamReader(System.in));
        String str="Client-1====>"+br.readLine();
        byte buff[]=new byte[1024];
        buff=str.getBytes();
        ds.send(new DatagramPacket(buff,buff.length,InetAddress.getLocalHost(),1000));
        System.out.println("now sending");
    }
    else if(ans.equalsIgnoreCase("no"))
    {
        System.out.println("I am Busy state");
        String msg="token";
        byte bf1[]=new byte[1024];
        bf1=msg.getBytes();
        ds.send(new DatagramPacket(bf1,bf1.length,InetAddress.getLocalHost(),200));
        hasToken=false;
    }
}
else
{
    System.out.println("Entering in receiving mode");
    byte bf[]=new byte[1024];
    ds.receive(dp=new DatagramPacket(bf,bf.length));
    String clientmsg=new String(dp.getData(),0,dp.getLength());
    System.out.println("The data is "+clientmsg);
    if(clientmsg.equals("token"))
    {
        hasToken=true;
        System.out.println("I am leaving busy state");
    }
}
}
}
}

```

File Name : TokenClient2.java

```

import java.net.*;
import java.io.*;
class TokenClient2
{
    public static DatagramSocket ds;
    public static DatagramPacket dp;
    public static BufferedReader br;
    public static void main(String args[])throws Exception
    {
        boolean hasToken=true;
        ds=new DatagramSocket(200);
        while(true)
        {
            if(hasToken==true)

```

```

{
    System.out.println("Do you want to write data...(yes/no)");
    br=new BufferedReader(new InputStreamReader(System.in));
    String ans=br.readLine();
    if(ans.equalsIgnoreCase("yes"))
    {
        System.out.println("ready to write");
        System.out.println("enter the data");
        br=new BufferedReader(new InputStreamReader(System.in));
        String str="Client-2===>"+br.readLine();
        byte buff[]=new byte[1024];
        buff=str.getBytes();
        ds.send(new DatagramPacket(buff,buff.length,InetAddress.getLocalHost(),1000));
        System.out.println("now sending");
    }
    else if(ans.equalsIgnoreCase("no"))
    {
        System.out.println("I am Busy state");
        String msg="token";
        byte bf1[]=new byte[1024];
        bf1=msg.getBytes();
        ds.send(new DatagramPacket(bf1,bf1.length,InetAddress.getLocalHost(),100));
        hasToken=false;
    }
    else
    {
        try
        {
            System.out.println("Entering in receiving mode");
            byte bf[]=new byte[1024];
            ds.receive(dp=new DatagramPacket(bf,bf.length));
            String clientmsg=new String(dp.getData(),0,dp.getLength());
            System.out.println("The data is "+clientmsg);
            if(clientmsg.equals("token"))
            {
                hasToken=true;
                System.out.println("I am leaving busy state");
            }
        }
        catch(Exception e){ }
    }
}
}
}

```

OUTPUT:**File Name: TokenServer.java -> Server**

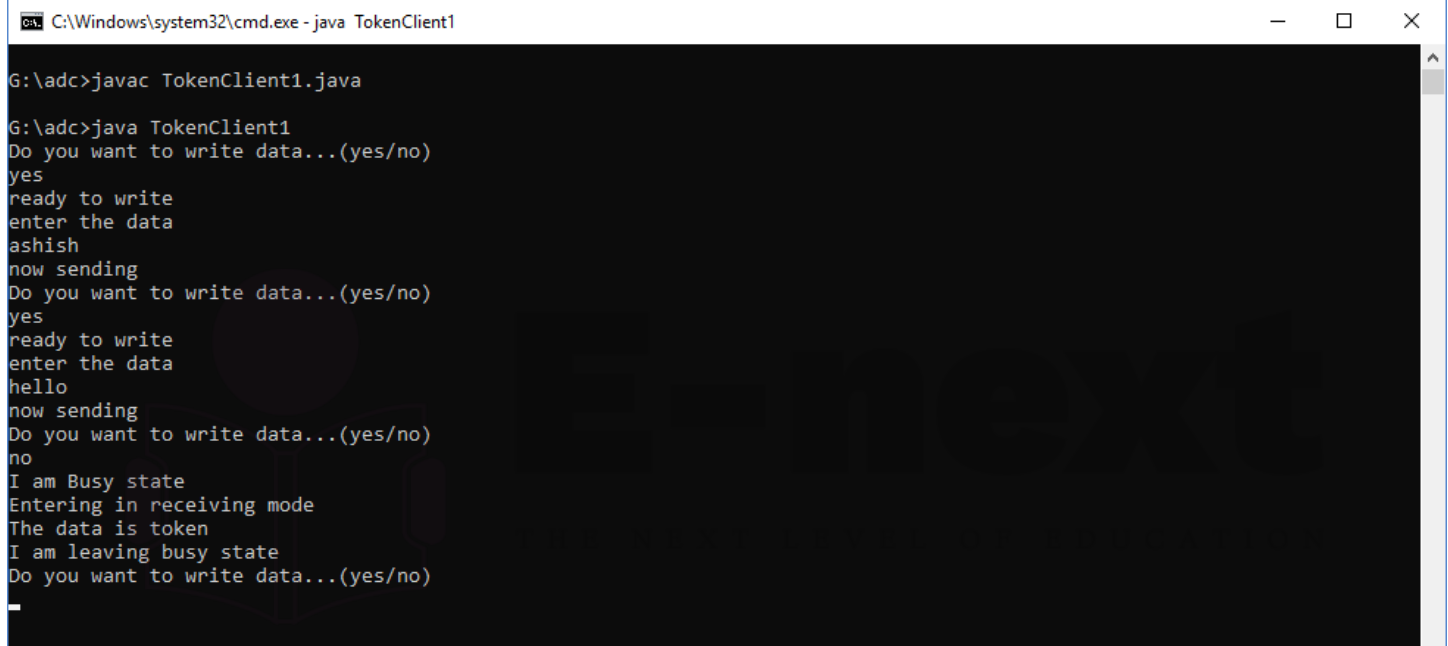

```

C:\Windows\system32\cmd.exe - java TokenServer

G:\adc>javac TokenServer.java

G:\adc>java TokenServer
Message from Client-1==>ashish
Message from Client-1==>hello
Message from Client-2==>vinay
Message from Client-2==>hii

```

File Name : TokenClient1.java -> Client 1


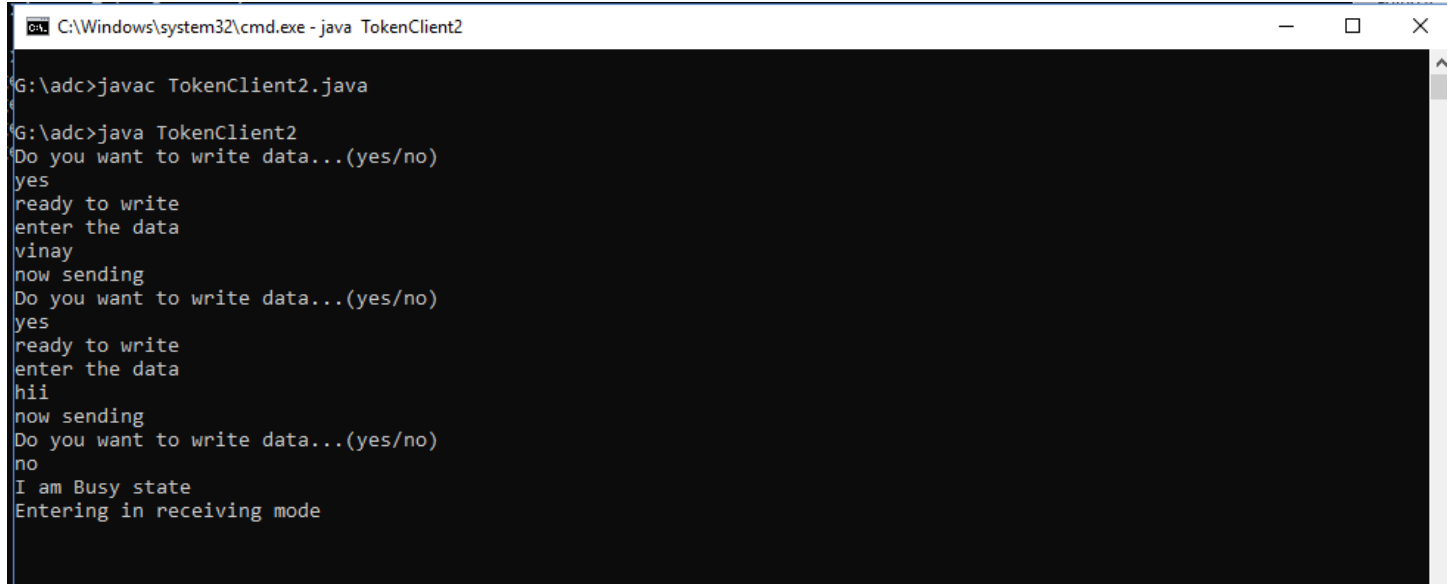
```

C:\Windows\system32\cmd.exe - java TokenClient1

G:\adc>javac TokenClient1.java

G:\adc>java TokenClient1
Do you want to write data...(yes/no)
yes
ready to write
enter the data
ashish
now sending
Do you want to write data...(yes/no)
yes
ready to write
enter the data
hello
now sending
Do you want to write data...(yes/no)
no
I am Busy state
Entering in receiving mode
The data is token
I am leaving busy state
Do you want to write data...(yes/no)
-

```

File Name : TokenClient2.java -> Client 2


```

C:\Windows\system32\cmd.exe - java TokenClient2

G:\adc>javac TokenClient2.java

G:\adc>java TokenClient2
Do you want to write data...(yes/no)
yes
ready to write
enter the data
vinay
now sending
Do you want to write data...(yes/no)
yes
ready to write
enter the data
hii
now sending
Do you want to write data...(yes/no)
no
I am Busy state
Entering in receiving mode

```


Practical No 8

AIM : Study of Cloud Virtualization Technologies Description:

Virtualization:

Virtualization refers to running multiple virtual computers, or virtual machines, inside a single physical computer. While the basic idea of virtualization is old (dating back to mainframe computers in 1960s), it has become mainstream only in the last 10-15 years. Today, most new servers are virtualized.

- ☐ Hypervisor is the operating system running on actual hardware. Virtual machines run as processes within this operating system. Sometimes the hypervisor is called Dom0 or Domain 0.
- ☐ Virtual machine is a virtual computer running under a hypervisor.
- ☐ Container is a lightweight virtual machine that runs under the same operating system instance (kernel) as the hypervisor. Essentially, a container is nothing but a group of processes with their own name space for process identifiers etc.
- ☐ Virtual network is a logical network within a server, or possibly extending to multiple servers or even multiple data centres.
- ☐ Virtualization software is software that implements virtualization on a computer. It can be part of an operating system (or a special version of an operating system) or an application software package.

Virtualization is the basis of modern cloud computing.

WHY VIRTUALIZE?

Virtualization drivers in recent years have included:

- More powerful hardware, allowing each machine to run multiple applications simultaneously
- Pressure to lower IT costs and simplify IT administration
- Need to manage large-scale installations and clusters, such as server farms
- Improved security, reliability, scalability, and device independence Ability to mix multiple operating systems on same hardware.

Virtualization has become a critically important focus of the IT world in recent years. Virtualization technologies are used by countless thousands of companies to consolidate their workloads and to make their IT environments scalable and more flexible. If you want to learn cloud computing, you'll simply have to absorb the basic virtualization technology concepts at some point.

This course will give you all the **fundamental concepts to understand how Virtualization works**: why it's so important and how we moved from Virtualization to cloud computing. As a beginner course, you will find how Virtualization helps companies and professionals achieving better TCO and how it works from a technical point of view. Learn what is an hypervisor, how virtual machines are separated inside the same physical host and how they communicate with lower hardware levels. If you want to start a career in the cloud computing industry, you will need to know how the most common virtualization technologies works and how they are used in cloud infrastructures.

A consistent part of this course is dedicated to the description of the most common technologies like: VMware, XEN, KVM and Microsoft Hyper-V. You will learn how they are used in the most common public cloud infrastructures and when to use them based on your needs.

Cloud Data Center:

Datacenter Design and Interconnection Networks

Datacenter Growth and Cost Breakdown : A large datacenter may be built with ten thousands or more servers. Smaller ones are built with hundreds or thousands of servers. The costs to build and maintain datacenter servers are increasing over the years. To keep a datacenter running well, typically, only 30% costs are due to purchase of IT equipments (such as servers and disks, etc), 33% costs are attributed to chiller, 18% on UPS

(uninterruptible power supply), 9% on CRAC (computer room air conditioning), and the remaining 7% due to power distribution, lighting, and transformer costs. Thus the cost to run a datacenter is dominated by about 60% in management and maintenance costs. The server purchase cost did not increase much with time. The cost of electricity and cooling did increase from 5% to 14% in 15 years.

Low-Cost Design Philosophy: High end switches or routers cost a lot of money. Thus using high-end network devices does not fit the economics of cloud computing. However, the cost only plays one side of the story. The other side is to provide more bandwidth. Given a fix number of budget, much more low-end commodity switches can be purchased than the high end devices. The large number of low-end commodity switches can provide network redundancies as well as much larger bandwidth. This is also the same story while choosing the large number of commodity x86 servers instead of small number of mainframes. While using the large number of low cost commodity switches, software developer should design the software layer for handling the network traffic balancing, fault tolerant as well as expandability. This is also the same on the server side. The network topology design must face such situation. Currently, near all the cloud computing datacenters are using the Ethernet as the fundamental network technology. 7.2.1 Warehouse-Scale Datacenter Design Figure 7.8 shows a programmer's view of storage hierarchy of a typical WSC. A server consists of a number of processor sockets, each with a multicore CPU and its internal cache hierarchy, local shared and coherent DRAM, and a number of directly attached disk drives. The DRAM and disk resources within the rack are accessible through the firstlevel rack switches (assuming some sort of remote procedure call API to them), and all resources in all racks are accessible via the cluster-level switch.

Consider a datacenter built with 2,000 servers, each with 8 GB of DRAM and four 1-TB disk drives. Each group of 40 servers is connected through a 1-Gbps link to a rack-level switch that has an additional eight 1Gbps ports used for connecting the rack to the cluster-level switch. It was estimated by Barroso and Holze [9] that the bandwidth available from local disks is 200 MB/s, whereas the bandwidth from off-rack disks is just 25 MB/s via the shared rack uplinks. On the other hand, total disk storage in the cluster is almost ten million times larger than local DRAM. A large application that requires many more servers than can fit on a single rack must deal with these large discrepancies in latency, bandwidth, and capacity. In a large scale datacenter, each component is relatively cheap and easily obtained from the commercial market. The components used datacenters are very different from those in building supercomputer systems. With a scale of thousands of servers, concurrent failure, either hardware failure or software failure, of tens of nodes is common. There are many failures that can happen in hardware, for example CPU failure, disk IO failure, and network failure etc. It is even quite possible that the whole datacenter does not work while facing the situation of power crash. And also, some failures are brought by software. The service and data should not be lost in failure situation. Reliable can be achieved by redundant hardware. The software must keep multiple copies of data in different location and keep the data accessible while facing hardware or software errors.

Datacenter Interconnection Networks A critical core design of a datacenter is the interconnection network among all servers in the datacenter cluster. This network design must meet five special requirements: low latency, high bandwidth, low cost, support MPI communications, and fault-tolerant. The design of an interserver network must satisfy both point-to-point and collective communication patterns among all server nodes. Specific design considerations are given below: Application Traffic Support: The network topology should support all MPI communication patterns. Both point-to-point and collective MPI communications must be supported. The network should have high bi-section bandwidth to meet this requirement. For example, one-to-many communications are used for supporting distribute file accesses. One can use one or a few servers as metadata master servers which need to communicate with slave server nodes in the cluster. To support the MapReduce programming paradigm, the network must be designed to perform the map and reduce functions (to be treated in Chapter 7) in high speed. In other word, the underline network structure should support various network traffic patterns demanded by user applications.

Network Expandability : The interconnection network should be expandable. With thousands or even hundreds of thousands of server nodes, the cluster network interconnection should be allow to expand, once more servers are added into a datacenter. The network topology should be restructured while facing such an expected growth in the future. Also the network should be designed to support load balancing and data movement among the servers. None of the links should become a bottleneck to slow down the application performance. The topology of the interconnection should avoid such bottlenecks.

Fault Tolerance and Graceful Degradation: The interconnection network should provide some mechanism to tolerate link or switch failures. Multiple paths should be established between any two server nodes in a datacenter. Fault tolerant of servers is achieved by replicating data and computing among redundant servers. Similar redundancy technology should apply to the network structure. Both software and hardware network redundancy apply to cope with potential failures. One the software side, the software layer should be aware of network failure. Packets forwarding should avoid using the broken links. The network support software drivers should handle this transparently without affecting the cloud operations.

Switch-centric Datacenter Design : Currently, there are two approaches to building datacenter-scale networks : One is switch-centric and the other is server-centric. In a switch-centric network, the switches are used to connect the server nodes. The switch centric design does not affect the server side. No modifications to the servers are needed. The server-centric design does modify the operating system running on servers. Special drivers are designed for relaying the traffic. Switches still have to be organized for achieving the connections.

Container Datacenter Construction: The datacenter module is housed in a container. The modular container design include the network gear, compute, storage, and cooling. Just plug-in power, network, and chilled water, the datacenter should work. One needs to increase the cooling efficiency by varying the water and air flow with better air flow management. Another concern is to meet the seasonal load requirements. The construction of the container-based datacenter may start with one system (server), then move to rack system design and finally the container system. The staged development may take different amounts of time and demand an increasing cost. For example, building one server system may take a few hours in racking and networking. Building a rack of 40 servers may take a half day's effort.

Practical No 9

AIM : Study of Grid Services.

Description:

The World Wide Web began as a technology for scientific collaboration and was later adopted for e-business. We foresee—and indeed are experiencing—a similar trajectory for Grid technologies. The scientific resource sharing applications that motivated the early development of Grid technologies include the pooling of expertise through collaborative visualization of large scientific data sets, the pooling of computer power and storage through distributed computing for computation ally demanding data analyses, and increasing functionality and availability by coupling scientific instruments with remote computers and archives.^{1,7} We expect similar applications to become important in commercial settings—initially for scientific and technical computing applications, where we can already point to success stories—and then for commercial distributed computing applications. However, we expect that rather than enhancing raw capacity, the most important role for Grid concepts in commercial computing will be to offer solutions to new challenges that relate to the construction of reliable, scalable, and secure distributed systems. These challenges derive from the current rush, driven by technology trends and commercial pressures, to decompose and distribute through the network previously monolithic host-centric services.

OPEN GRID SERVICES ARCHITECTURE Enterprise computing systems must increasingly operate within virtual organizations (VO) with similarities to the scientific collaborations that originally motivated Grid computing. Depending on the context, the dynamic ensembles of resources, services, and people that comprise a scientific or business VO can be small or large, short- or long-lived, single- or multi-institutional, and homogeneous or heterogeneous. Individual ensembles can be structured hierarchically from smaller systems and may overlap in membership. Regardless of these differences, VO application developers face common requirements as they seek to deliver QoS—whether measured in terms of common security semantics, distributed workflow and resource management, coordinated fail-over, problem determination services, or other metrics— across a collection of resources with heterogeneous and often dynamic characteristics. Service orientation The Open Grid Services Architecture (OGSA) ³ supports the creation, maintenance, and application of the service ensembles that VOs maintain. OGSA adopts a common representation for computational and storage resources, networks, programs, databases, and the like. All are treated as services—network-enabled entities that provide some capability through the exchange of messages. Arguably, we could use the term object instead, as in systems like SOS⁸ and Legion,⁹ but we avoid it because of its overloaded meaning and because OGSA does not require object-oriented implementations. Adopting this uniform service-oriented model makes all components of the environment virtual—although the model must be grounded on implementations of physical resources. This service-oriented view partitions the interoperability problem into two subproblems: the definition of service interfaces and the identification of protocols that can invoke a particular interface. A service-oriented view addresses the need for standard interface definition mechanisms, local and remote transparency, adaptation to local OS services, and uniform service semantics. A service-oriented view also simplifies virtualization through encapsulation of diverse implementations behind a common interface. Virtualization Virtualization enables consistent resource access across multiple heterogeneous platforms. Virtualization also enables mapping of multiple logical resource instances onto the same physical resource and facilitates management of resources within a VO based on composition from lower-level basic services to form more sophisticated services— without regard for how these services are implemented. Virtualizing Grid services also underpins the ability to map common service semantic behavior seamlessly onto native platform facilities. This virtualization is easier if we can express service functions in a standard form, so that any implementation of a service is invoked in the same man ner. We adopt the Web Services Description Language (WSDL) for this purpose.

Service semantics: The Grid service Our ability to virtualize and compose services depends on more than standard interface definitions. We also require standard semantics for service interactions so that, for example, we have standard mechanisms for discovering service properties and different services follow the same conventions for error notification. To this end, OGSA defines a Grid service—a Web service that provides a set

of well-defined interfaces and that follows specific conventions. The interfaces address discovery, dynamic service creation, lifetime management, notification, and manageability; the conventions address naming and upgradeability. Grid services also address authorization and concurrency control. This core set of consistent interfaces, from which we implement all Grid services, facilitates the construction of hierarchical, higher-order services that can be treated uniformly across layers of abstraction. As Figure 1 shows, a set of interfaces configured as a WSDL portType defines each Grid service. Every Grid service must support the GridService interface; in addition, OGSA defines a variety of other interfaces for notification and instance creation. Of course, users also can define arbitrary application-specific interfaces. The Grid service's serviceType, a WSDL extensibility element, defines the collection of portTypes that a Grid service supports, along with some additional information relating to versioning.

Grid Services Tools:

ASKALON Architecture ASKALON has been designed as a set of distributed Grid services (see Figure 1). The services are based on the OGSI-technology and expose a platform independent standard API, expressed in the standard Web Services Description Language (WSDL) [11]. Platform dependent and proprietary services are pre-installed on specific appropriate sites from where they can be remotely accessed in a portable way, via the Simple Object Access Protocol (SOAP) [46] over HTTP. By isolating platform dependencies on critical resources, extra flexibility in installing and managing the tools is achieved. Each tool provides its own graphical User Portal to be accessed in a friendly and intuitive way. The User Portals are light-weight clients, easy to be installed and managed by the end-users. User Portals reside on the user's local machine (e.g. a notebook) and provide gateways to performance tools by dynamically creating and connecting to remote services. ASKALON services can be persistent (e.g. Registry and Factory) or transient, as specified by OGSI. All services can be accessed concurrently by multiple clients, which is an essential feature in a Grid environment and enables tool interoperability.

The Grid SECURITY Infrastructure (GSI) [26] based on single signon, credential delegation, and Web services security [5] through XML digital signature and XML encryption is employed for authentication across ASKALON User Portals and Grid services. Remote service instances are created by a general-purpose Factory service (see Section 3.2 using the information from the Service Repository (see Section 3.1). At the same time, the portals discover and bind to existing service instances using the Registry service, described in Section 3.3. Additionally, the Data Repository (see Section 3.4) with a common standard schema definition, stores and shares common performance and output data of the applications under evaluation. It thus provides an additional mode of integration and interoperability among tools. To increase reliability of the system by avoiding single point of failures, multiple Registry, Service, and Data Repository instances are replicated on multiple sites and run independently.

An OGSI-based asynchronous event framework enables Grid services to notify clients about interesting system and application events. ASKALON services support both push and pull event models, as specified by the Grid Monitoring Architecture (GMA) [50]. Push events are important for capturing dynamic information about running applications and the overall Grid system on-the-fly and avoids expensive continuous polling. Pull events are crucial for logging important information, for instance in cases when tools like ZENTURIO run in off-line mode, with disconnected off-line users. ASKALON classifies the Grid sites on which the services can run into two categories (see Figure 1): (1) Compute sites are Grid locations where end applications run and which host services intimately related to the application execution. Such services include the Experiment Executor of ZENTURIO, in charge of submitting and controlling jobs on the local sites and the Overhead Analyzer of SCALEA, which transforms raw performance data collected from the running applications into higher-level more meaningful performance overheads. (2) Service sites are arbitrary Grid locations on which ASKALON services are pre-installed or dynamically created by using the Factory service