Experiment No. 1

Aim-: Program to demonstrate Client/Server application Using RMI

The RMI Application is composed of four programs:

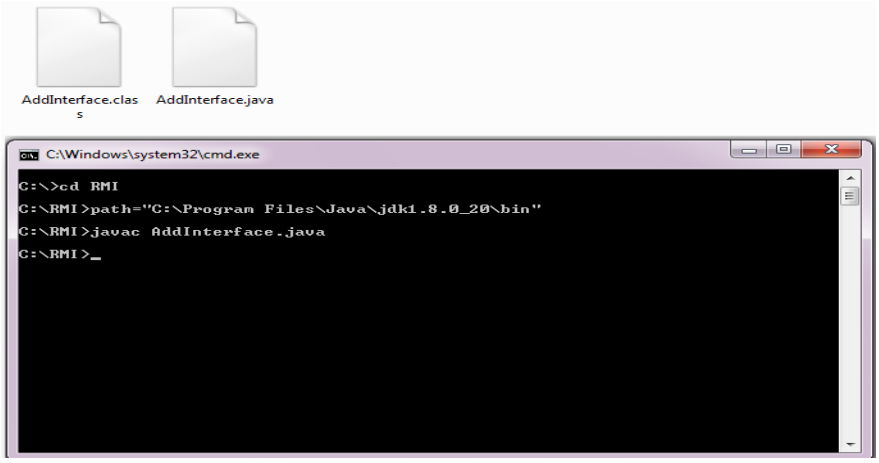
a) **Interface program**– which contains the declaration of methods to be called by a client and defined by the server program.
b) **Implementation program**– which contains definition of method that is declared in the interface.
c) **Server Program**– which contains statement like Naming.rebind to bind the objects called by the client.
d) **Client program**– which contains Method calling and Naming.lookup method to locate the object in registry.

The following steps will explain the RMI program to add two numbers:

Step 1 – create a new directory, named RMI, for this application. Start your favorite text editor and create a file named AddInterface.java in this directory. In your file, enter the following code which has declaration of add() method.

```
import java.rmi.*;

public interface AddInterface extends Remote

{

        public int sum(int n1,int n2) throws RemoteException;

}
```

Save the file and Compile this program using the java compiler.



Step 2 – Write and Compile Implementation file Add.java, which contains definition of the method declared in the interface.

Add.java

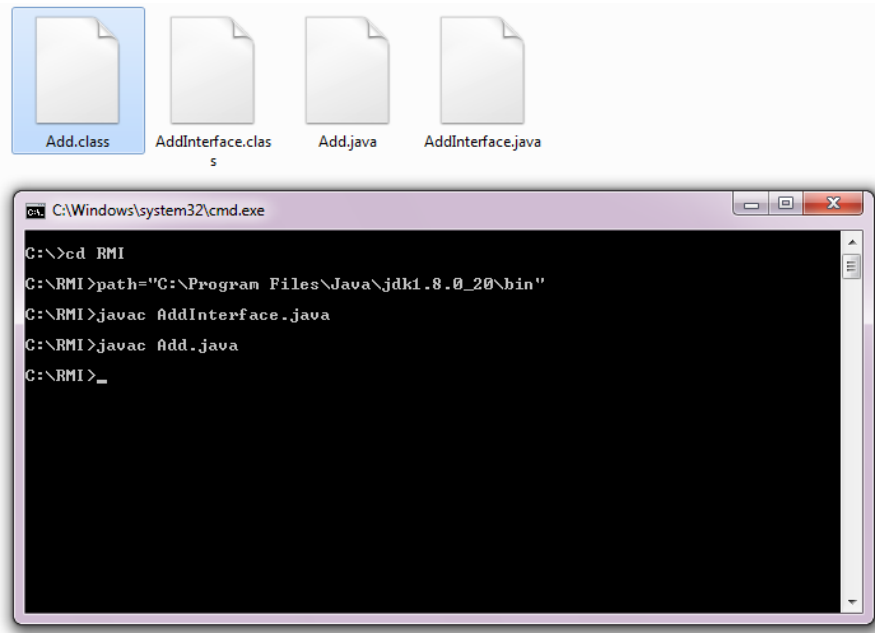
```
import java.rmi.*;

import java.rmi.server.*;

public class Add extends UnicastRemoteObject implements AddInterface

{

int num1,num2;

public Add() throws RemoteException

{

}

public int sum(int n1,int n2) throws RemoteException

{

num1=n1;

num2=n2;
```

return num1+num2;

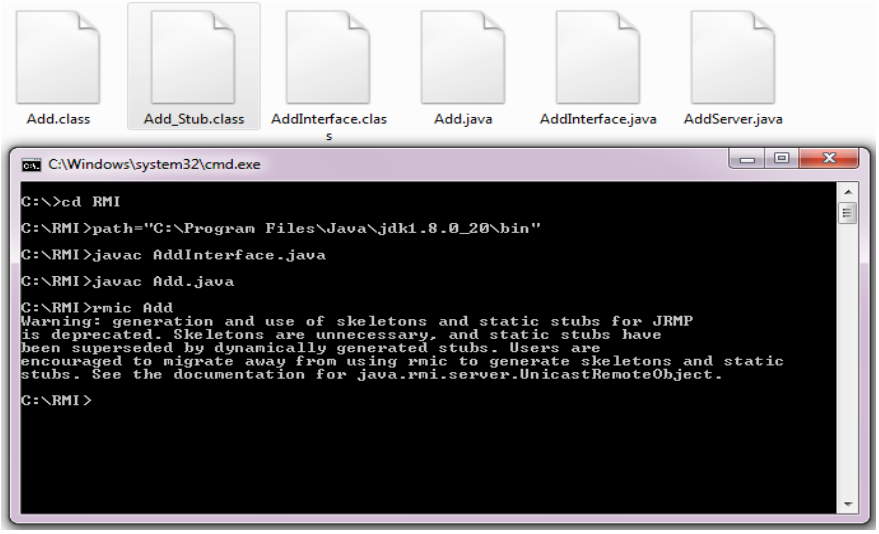
}

}

Compile the program as follows:



Step 3 – Create stubs and skeletons using rmic command by specifying implementation class name, i.e. rmic Add



Step 4 – Create AddServer.java file and add the following code:

AddServer.java

```
import java.rmi.Naming;
public class AddServer
{
public static void main(String args[])
{
try
{
Naming.rebind("Add",new Add());
System.out.println("Server is connected and waiting for the client");
}
catch(Exception e)
```

{

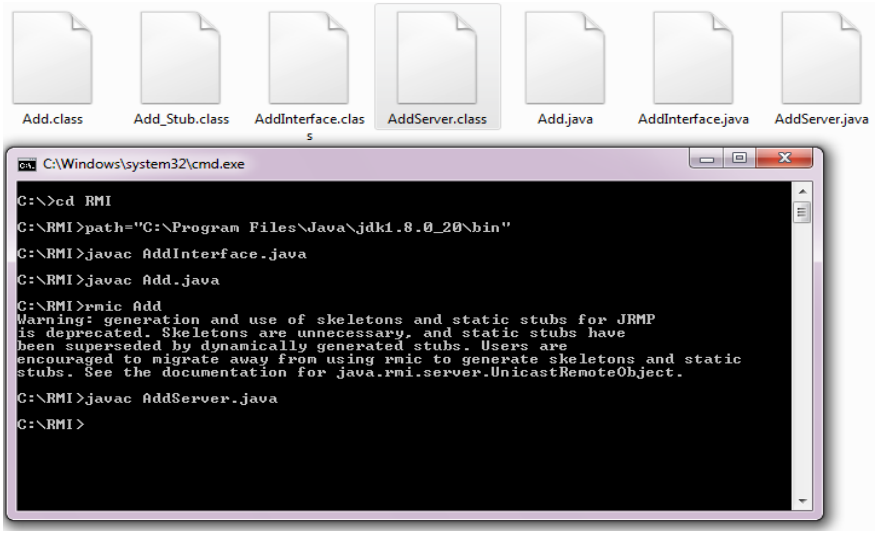
System.out.println("Server could not connect: "+e);

}

}

}

Compile it using javac command



Step 4 – Create AddClient.java file and add the following code, which contains calling of sum() method with parameters 10 and 2:

AddClient.java

---

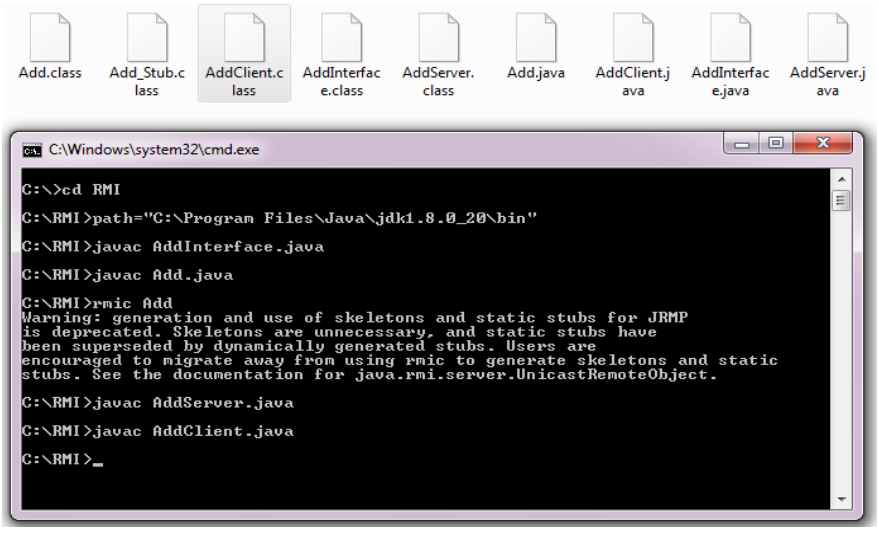
import java.rmi.Naming;

public class AddClient

{

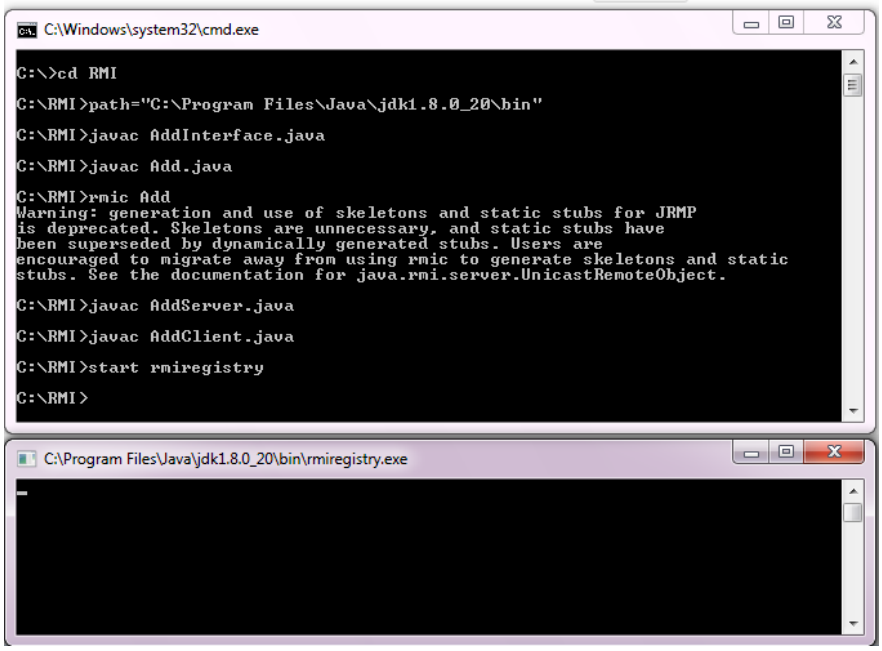
```
public static void main(String args[])
{
try
{
AddInterface ai=(AddInterface)Naming.lookup("//localhost/Add");
System.out.println("The sum of 2 numbers is: "+ai.sum(10,2));
}
catch(Exception e)
{
System.out.println("Client Exception: "+e);
}
}
}
```

Compile the program which generates class file



Step 5– Start RMI Registry and minimize it.

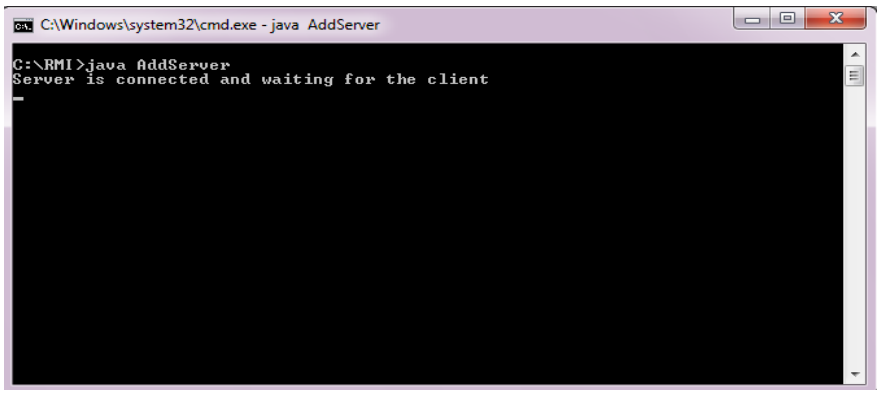
(Note – Please do not close the Registry till we get the results otherwise server and client both the programs will generate errors)



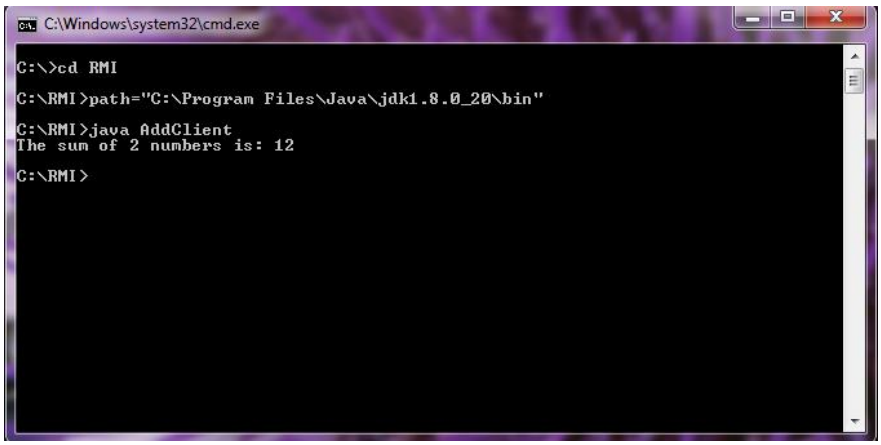
Step 5 – Run Server Program and keep it running.

Commented [AS1]: Is this really a necessary command?

Commented [A2R1]: Yes



Step 6 – Open one more command prompt and run Client program to get the output.



Experiment No. 2

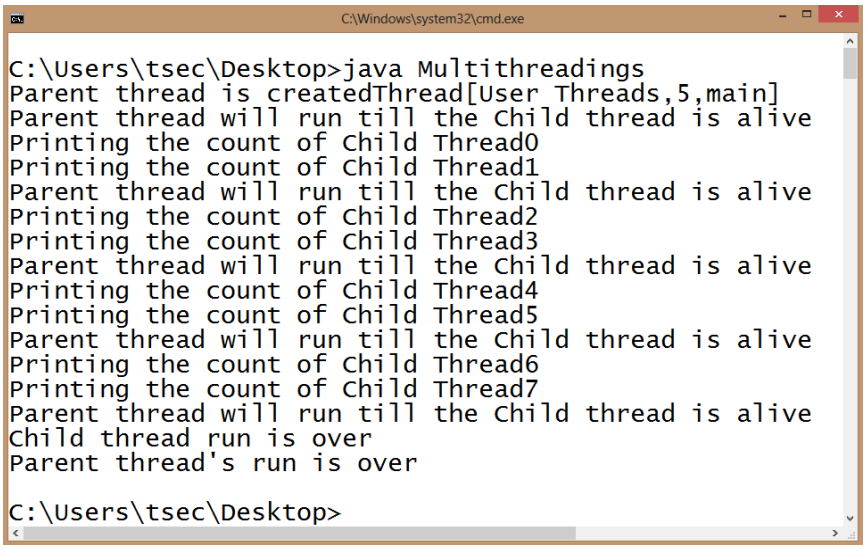
Aim– Program to demonstrate the implementation of a multi-threaded Application using Java

The program for implementing multi-threaded Application using Java is as follows:

```
class threads extends Thread
{
  threads()
  {
   super("User Threads");
   System.out.println("User thread is created" + this);
   start();
  }
  public void run()
  {
   try
   {
     for (int i=0 ;i<8;i++)
     {
       System.out.println("Printing the count of Child Thread" + i);
       Thread.sleep(800);
     }
   }
   catch(InterruptedException e)
   {
     System.out.println("User thread interrupted");
```

```
    }
    System.out.println("Child thread run is over" );
  }
}
class Multithreadings
{
  public static void main(String args[])
  {
    threads th = new threads();
    try
    {
      while(th.isAlive())
      {
       System.out.println("Parent thread will run till the Child thread is alive");
       Thread.sleep(1500);
      }
    }
    catch(InterruptedException e)
    {
     System.out.println("Parent thread interrupted");
    }
    System.out.println("Parent thread's run is over" );
  }
}
```

The Output of above program is shown as follows:



Experiment No. 3

Aim– Program to demonstrate Inter-Process communication using Java

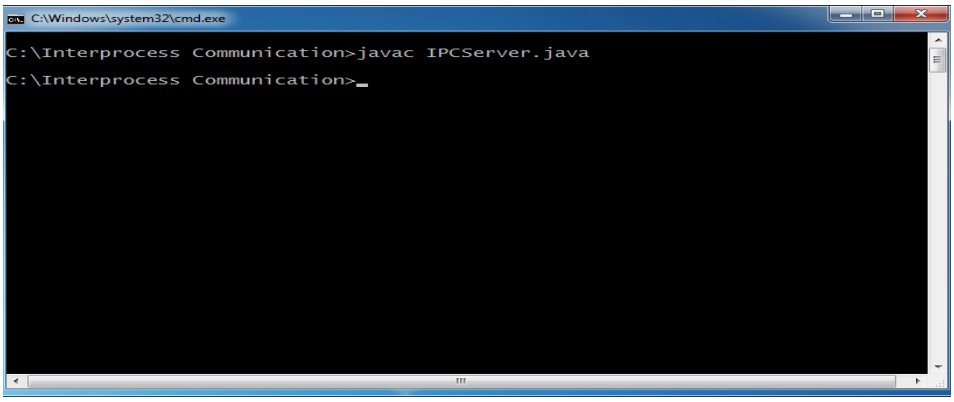
Steps to perform Inter-process Communication in Java using Client and Server programs are as follows

Steps 1– Write Server Program IPCServer.java and compile it.

IPCServer.java

```
import java.net.*;
import java.io.*;
public class IPCServer
{
public static void main (String args[])
{
System.out.println("\n **** INTERPROCESS COMMUNICATION ****\n");
System.out.println("\n *** SERVER PROCESS STARTED ***\n");
System.out.println("\n * SERVER IS READY AND WAITING TO RECEIVE DATA FROM
CLIENT PROCESS ON PORT " +1200);
try
 {
ServerSocket ss = new ServerSocket(1200); // 1200 is port number
Socket clientSocket = ss.accept();
System.out.println("\n * Client is connected with IP address " +clientSocket.getInetAddress()
+ " and port Number " + clientSocket.getPort());
DataOutputStream dos = new DataOutputStream(clientSocket.getOutputStream());
DataInputStream dis = new DataInputStream(clientSocket.getInputStream());
int a=dis.readInt();
System.out.println("\n SERVER RECEIVED");
System.out.println("\n Number 1 ====>"+a);
int b=dis.readInt();
System.out.println("\n Number 2 ====>"+b);
int c=a+b;
```

```
dos.writeInt(c);
System.out.println("\n SERVER PROCESS HAS EXECUTED REQUESTED PROCESS AND
SENT RESULT "+c+" TO THE CLIENT \n");
clientSocket.close();
System.out.println("\n SERVER PROCESS EXITING.....");
ss.close();
}
catch (Exception e) {
System.out.println("Exception: " + e);
}
}
}
```

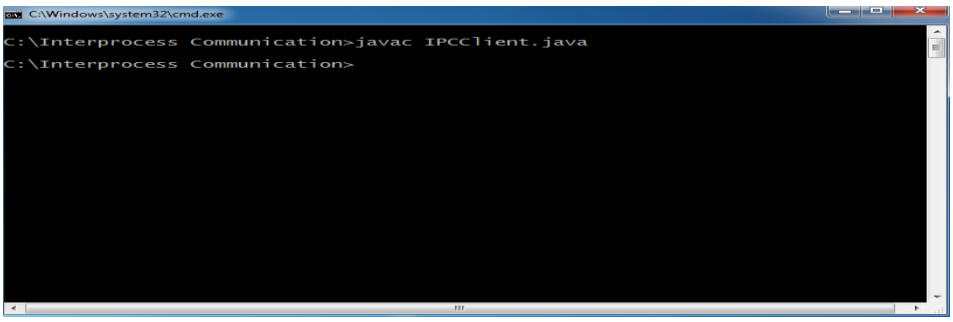


Step 2– Write Client Program IPCClient.java and compile it.

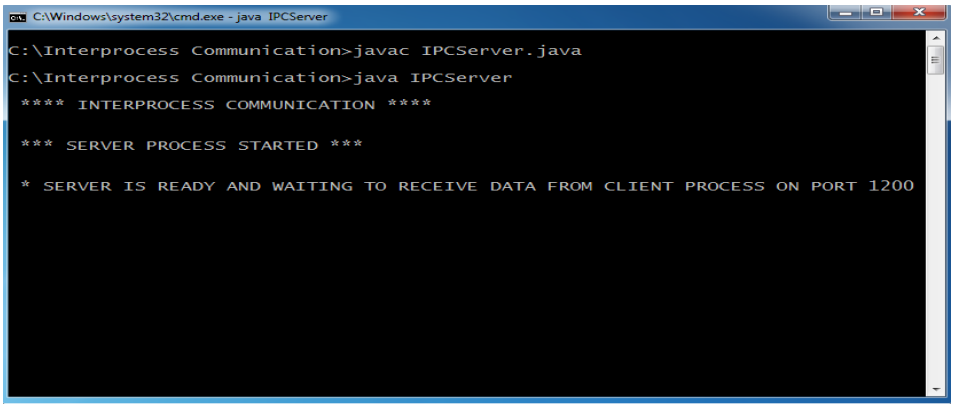
IPCClient.java

```
import java.net.*;
import java.io.*;
public class IPCClient
{
public static void main (String args[])
 {
```

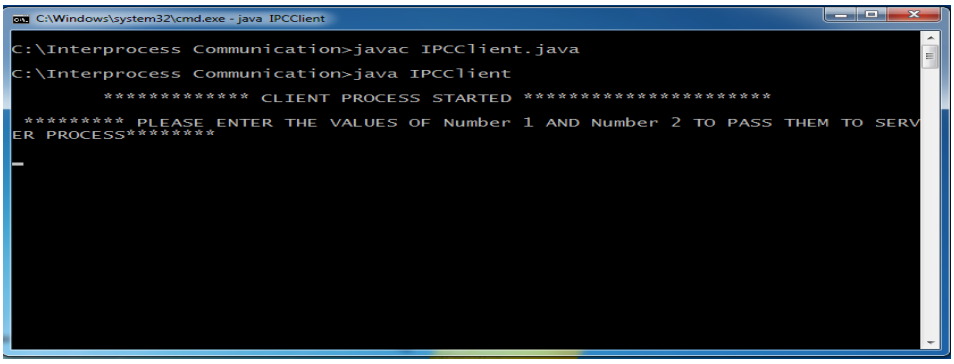
```
try
{
Socket s = new Socket("localhost",1200);
DataOutputStream dos = new DataOutputStream(s.getOutputStream());
DataInputStream dis = new DataInputStream(s.getInputStream());
InputStreamReader isr=new InputStreamReader(System.in);
System.out.println(" \n \t********** CLIENT PROCESS STARTED
********** ");
System.out.println("\n ********* PLEASE ENTER THE VALUES OF Number 1 AND
Number 2 TO PASS THEM TO SERVER PROCESS******** \n");
BufferedReader br=new BufferedReader(isr);
int a=Integer.parseInt(br.readLine());
System.out.println("Number 1 ====>"+a);
dos.writeInt(+a);
int b= Integer.parseInt(br.readLine());
System.out.println("Number 2 ====>"+b);
dos.writeInt(+b);
int result=dis.readInt();
System.out.println("\n.....CLIENT PROCESS HAS RECEIVED RESULT FROM
SERVER.....\n");
System.out.println("\n THE ADDITION OF " + a +" AND "+ b + " IS " +result);
s.close();
}
catch (Exception e)
{
System.out.println("Exception is "+e);
}
}
}
```



Step 3– Run Server Program.



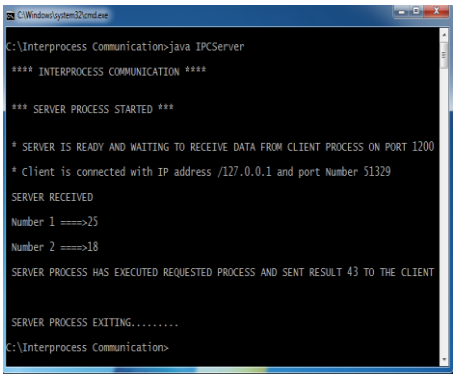
Step 4– Run Client Program.



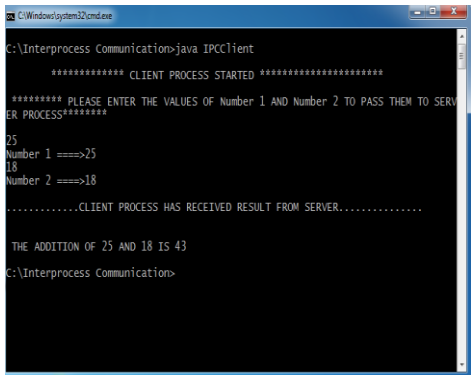
The Output of Inter-process Communication is shown as follows

Commented [AS3]: This is not a step. It's the result of all the steps taken above. Amend it.

Server Side



Client Side



Experiment No. 4

Aim– Program to demonstrate Group Communication using Java

This Application has two important programs, namely– client and server. The client application is copied thrice and given names as Master, slave1 and slave2.The master program is responsible for broadcasting the message among their group while slave can read those messages. The server is only a handler, which binds multiple clients together into a group. The steps of implementation are as follows:

Step 1– Write Server Program Server.java

server.java

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Vector;
public class Server
{
private static Vector<PrintWriter> writers = new Vector<PrintWriter>();
```

```
public static void main(String[] args) throws Exception
{
ServerSocket listener = new ServerSocket(9001);
System.out.println("The server is running at port 9001.");
while (true)
new Handler(listener.accept()).start();
}
private static class Handler extends Thread
{
private Socket socket;
public Handler(Socket socket)
{
this.socket = socket;
}
public void run()
{
try
{
BufferedReader in = new BufferedReader(newInputStreamReader(socket.getInputStream()));
PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
out.println("SUBMITNAME");
String name = in.readLine();
System.out.println(name+" joined");
writers.add(out);
while (true)
{
String input = in.readLine();
for (PrintWriter writer : writers)
writer.println("MESSAGE " + name + ": " + input);
}
}
catch (Exception e) {System.err.println(e);}
}
}
```

```
}
```

Step 2– Write Master Client Program master.java

master.java

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.Scanner;
public class master
{
public static void main(String[] args) throws Exception {
Scanner sc = new Scanner(System.in);
Socket socket = new Socket("localhost", 9001);
BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
System.out.print("Enter your name: ");
String name = sc.nextLine();
while (true) {
String line = in.readLine();
if (line.startsWith("SUBMITNAME")) out.println(name);
else if (line.startsWith("MESSAGE"))
System.out.println(line.substring(8));
if(name.startsWith("master")){
System.out.print("Enter a message: ");
out.println(sc.nextLine());
}
}
}
}
```

Step 3– Write slave1 Client Program slave1.java that is same as the master.

slave1.java

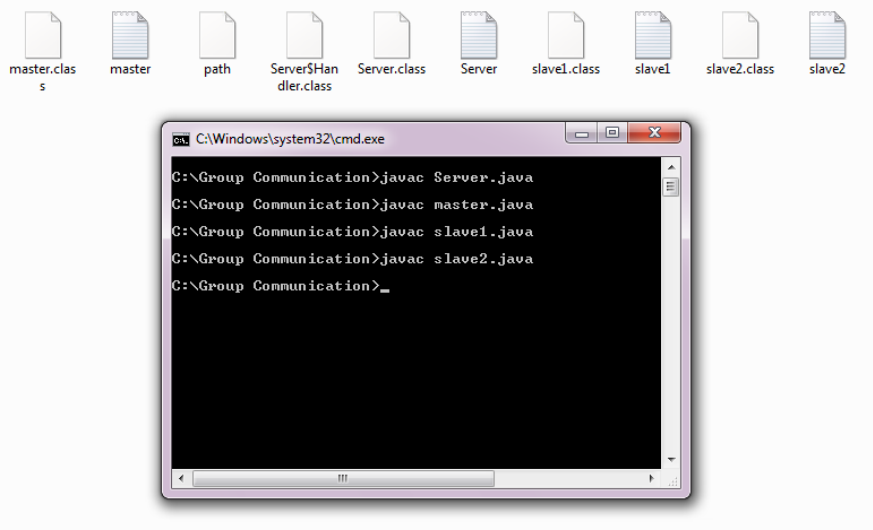
```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.Scanner;
public class slave1
{
public static void main(String[] args) throws Exception {
Scanner sc = new Scanner(System.in);
Socket socket = new Socket("localhost", 9001);
BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
System.out.print("Enter your name: ");
String name = sc.nextLine();
while (true) {
String line = in.readLine();
if (line.startsWith("SUBMITNAME")) out.println(name);
else if (line.startsWith("MESSAGE"))
System.out.println(line.substring(8));
if(name.startsWith("master")){
System.out.print("Enter a message: ");
out.println(sc.nextLine());
}
}
}
}
```

Step 4– Write slave2 Client Program slave2.java, same as the master.

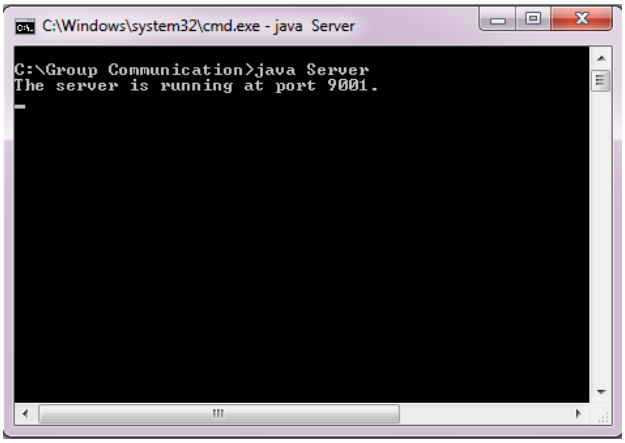
slave2.java

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.Scanner;
public class slave2
{
public static void main(String[] args) throws Exception
{
Scanner sc = new Scanner(System.in);
Socket socket = new Socket("localhost", 9001);
BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
System.out.print("Enter your name: ");
String name = sc.nextLine();
while (true)
{
String line = in.readLine();
if (line.startsWith("SUBMITNAME"))
out.println(name);
else if (line.startsWith("MESSAGE"))
System.out.println(line.substring(8));
if(name.startsWith("master")){
System.out.print("Enter a message: ");
out.println(sc.nextLine());
}
}
}
}
```

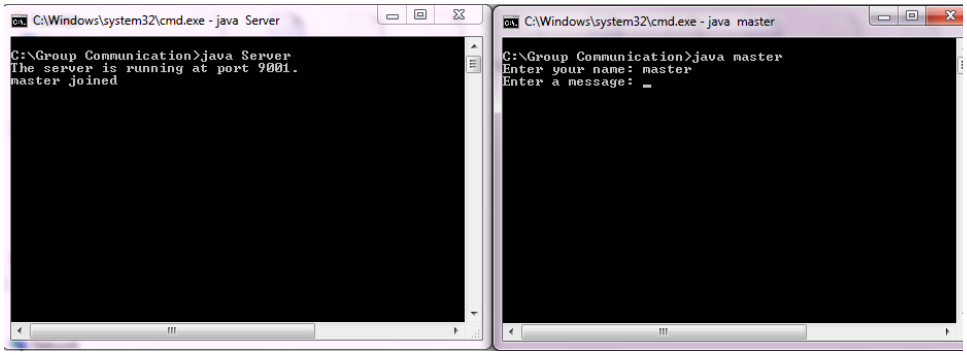
Step 5 – Compile Server, master and slave programs.



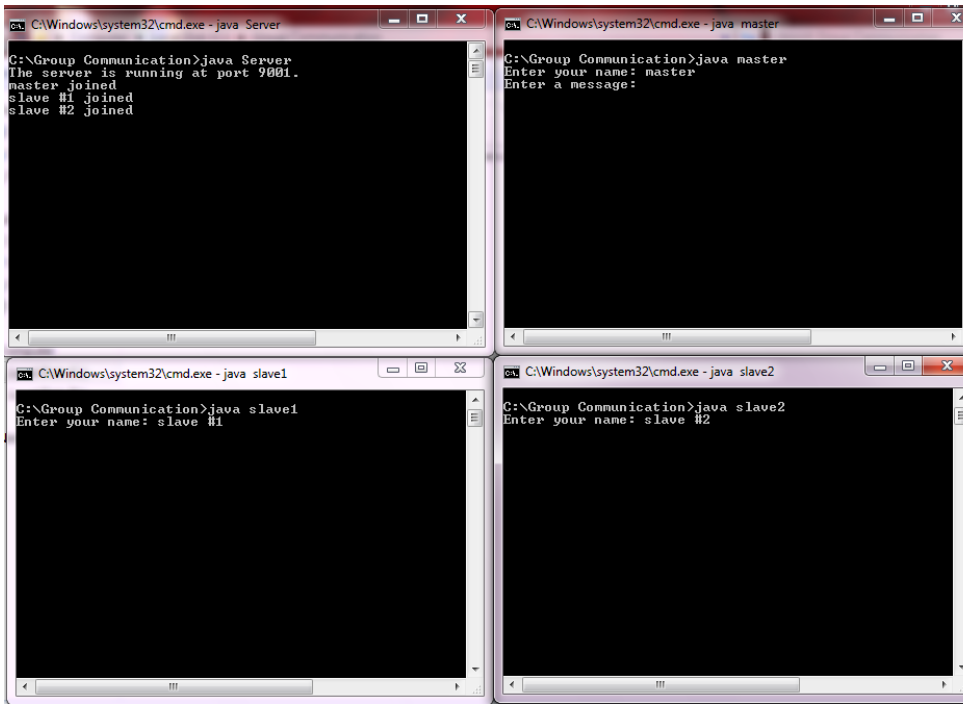
Step 5 – Run Server program.



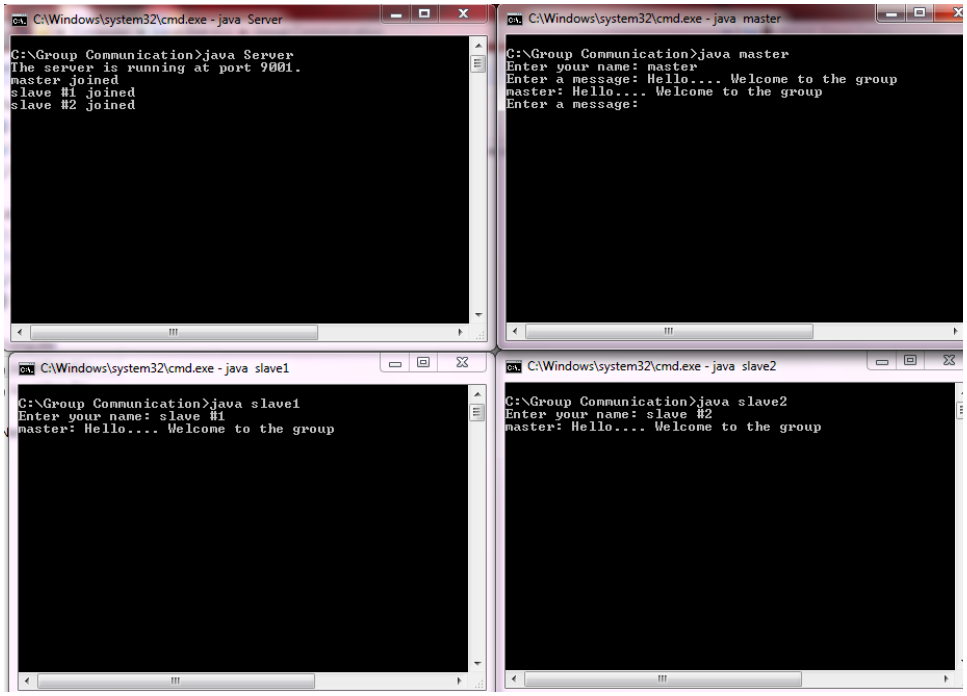
Step 6 – Run Master program.



Step 7– Run slave1 and slave 2 programs.



Step 8 – In the final output, the master broadcasts the message within the group, which will be delivered at slave1 and slave 2 terminals.



Experiment No. 5

Aim– Program to demonstrate Load Balancing Algorithm using Java.

Program to Simulate Load Balancing Environment using Java

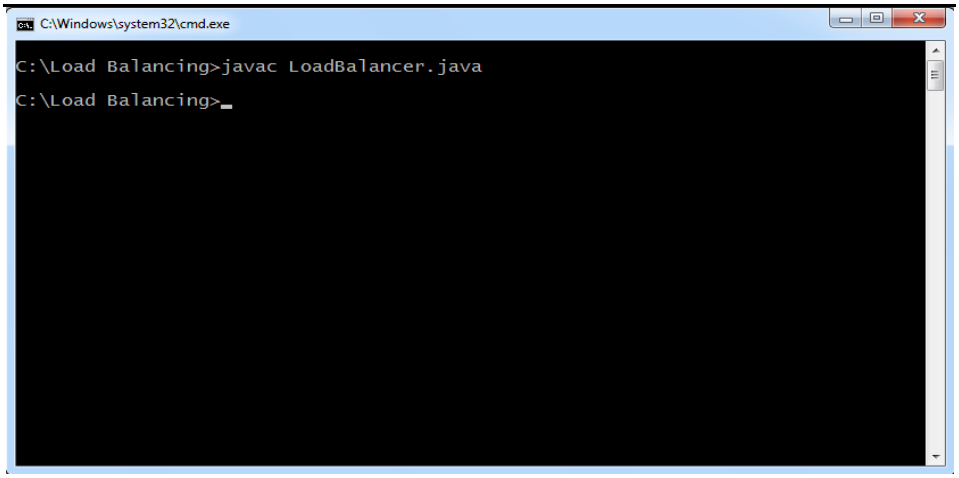
LoadBalancer.java

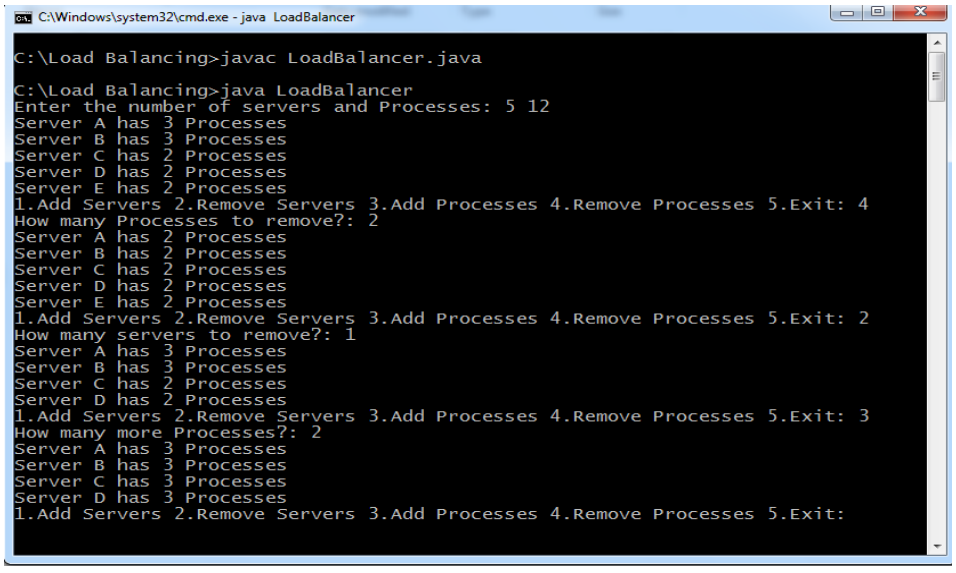
```
import java.util.Scanner;
class LoadBalancer{
static void printLoad(int servers,int Processes){
int each = Processes/servers;
int extra = Processes%servers;
int total = 0;
```

```
for(int i = 0; i < servers; i++){
if(extra-->0) total = each+1;
else total = each;
System.out.println("Server "+(char)('A'+i)+" has "+total+" Processes");
}
}
public static void main(String[] args){
Scanner sc = new Scanner(System.in);
System.out.print("Enter the number of servers and Processes: ");
int servers = sc.nextInt();
int Processes = sc.nextInt();
while(true){
printLoad(servers, Processes);
System.out.print("1.Add Servers 2.Remove Servers 3.Add Processes 4.Remove Processes 5.Exit: ");
switch(sc.nextInt()){
case 1:
System.out.print("How many more servers?: ");
servers+=sc.nextInt();
break;
case 2:
System.out.print("How many servers to remove?: ");
servers-=sc.nextInt();
break;
case 3:
System.out.print("How many more Processes?: ");
Processes+=sc.nextInt();
break;
case 4:
System.out.print("How many Processes to remove?: ");
Processes-=sc.nextInt();
break;
case 5:
return;
}
```

```
}
}
}
```

OUTPUT


C:\Windows\system32\cmd.exe
C:\Load Balancing>javac LoadBalancer.java
C:\Load Balancing>


C:\Windows\system32\cmd.exe - java LoadBalancer
C:\Load Balancing>javac LoadBalancer.java
C:\Load Balancing>java LoadBalancer
Enter the number of servers and Processes: 5 12
Server A has 3 Processes
Server B has 3 Processes
Server C has 2 Processes
Server D has 2 Processes
Server E has 2 Processes
1.Add Servers 2.Remove Servers 3.Add Processes 4.Remove Processes 5.Exit: 4
How many Processes to remove?: 2
Server A has 2 Processes
Server B has 2 Processes
Server C has 2 Processes
Server D has 2 Processes
Server E has 2 Processes
1.Add Servers 2.Remove Servers 3.Add Processes 4.Remove Processes 5.Exit: 2
How many servers to remove?: 1
Server A has 3 Processes
Server B has 3 Processes
Server C has 2 Processes
Server D has 2 Processes
1.Add Servers 2.Remove Servers 3.Add Processes 4.Remove Processes 5.Exit: 3
How many more Processes?: 2
Server A has 3 Processes
Server B has 3 Processes
Server C has 3 Processes
Server D has 3 Processes
1.Add Servers 2.Remove Servers 3.Add Processes 4.Remove Processes 5.Exit:

Experiment No. 6

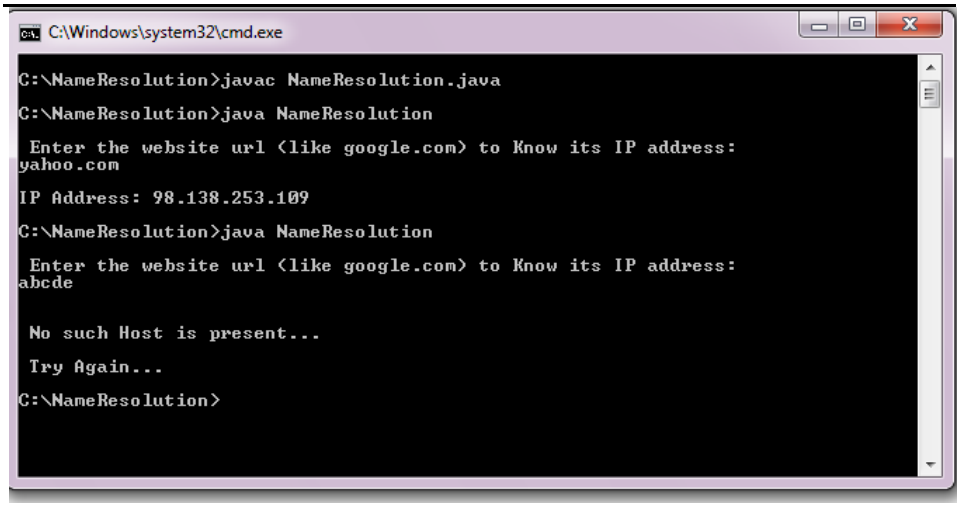
Aim– Program to demonstrate Name Resolution Protocol using Java.

Program to Demonstrate Name resolution using Java

NameResolution.java

```
import java.io.*;
import java.net.*;
public class NameResolution
{
public static void main(String args[]) throws IOException
{
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
System.out.println("\n Enter the website url (like google.com) to Resolve its Name to Address: ");
String name = br.readLine();
try
{
InetAddress ip = InetAddress.getByName(name);
System.out.println("\nIP Address: "+ip.getHostAddress());
}
catch(UnknownHostException e)
{
System.out.println("\n\n No such Host is present...");
System.out.println("\n Try Again...");
}
}
}
```

OUTPUT



Experiment No. 7

Aim– Program to demonstrate Bully Election Algorithm using Java.

Program to demonstrate Bully Election Algorithm

BullyAlgo.java

```
import java.io.*;
class BullyAlgo
{
int cood,ch,crash;
int prc[];
public void election(int n) throws IOException
{
```

```
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
System.out.println("\nThe Coordinator Has Crashed!");
int flag=1;
while(flag==1)
{
crash=0;
for(int i1=0;i1<n;i1++)
if(prc[i1]==0)
crash++;
if(crash==n)
{
System.out.println("\n*** All Processes Are Crashed ***");
break;
}
else
{
System.out.println("\nEnter The Intiator");
int init=Integer.parseInt(br.readLine());
if((init<1)||(init>n)||(prc[init-1]==0))
{
System.out.println("\nInvalid Initiator");
continue;
}
for(int i1=init-1;i1<n;i1++)
System.out.println("Process "+(i1+1)+" Called For Election");
System.out.println("");
```

```
for(int i1=init-1;i1<n;i1++)
{
if(prc[i1]==0)
{
System.out.println("Process "+(i1+1)+ " Is Dead");
}
else
System.out.println("Process "+(i1+1)+" Is In");
}
for(int i1=n-1;i1>=0;i1--)
if(prc[i1]==1)
{
cood=(i1+1);
System.out.println("\n*** New Coordinator Is "+(cood)+" ***");
flag=0;
break;
}
}
}
}
public void Bully() throws IOException
{
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
System.out.println("Enter The Number Of Processes: ");
int n=Integer.parseInt(br.readLine());
prc=new int[n];
```

```
crash=0;
for(int i=0;i<n;i++)
prc[i]=1;
cood=n;
do
{
System.out.println("\n\t1. Crash A Process");
System.out.println("\t2. Recover A Process");
System.out.println("\t3. Display New Cordinator");
System.out.println("\t4. Exit");
ch=Integer.parseInt(br.readLine());
switch(ch)
{
case 1: System.out.println("\nEnter A Process To Crash");
int cp=Integer.parseInt(br.readLine());
if((cp>n)||(cp<1)){
System.out.println("Invaid Process! Enter A Valid Process");
}
else if((prc[cp-1]==1)&&(cood!=cp))
{
prc[cp-1]=0;
System.out.println("\nProcess "+cp+ " Has Been Crashed");
}
else if((prc[cp-1]==1)&&(cood==cp))
{
prc[cp-1]=0;
```

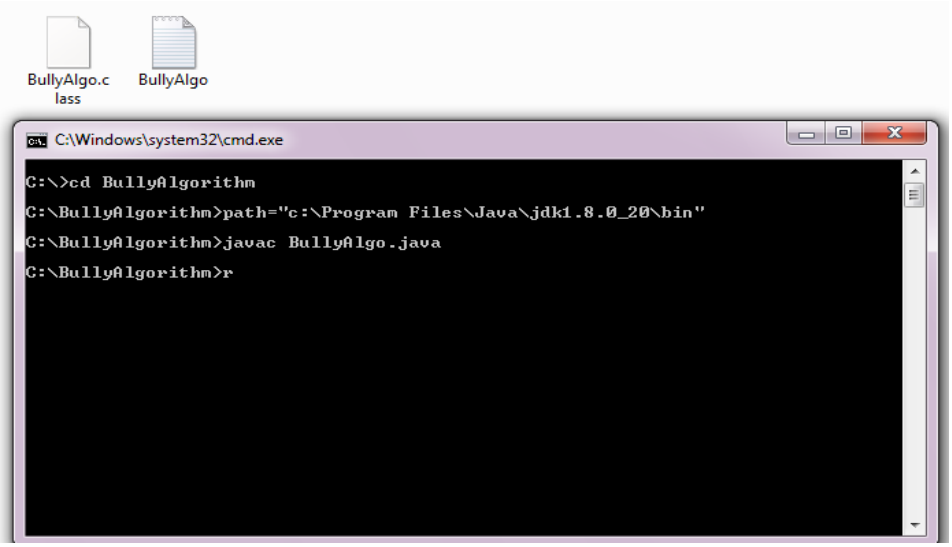
```
election(n);
}
else
System.out.println("\nProcess "+cp+" Is Already Crashed");
break;
case 2: System.out.println("\nCrashed Processes Are: \n");
for(int i=0;i<n;i++)
{
if(prc[i]==0)
System.out.println(i+1);
crash++;
}
System.out.println("Enter The Process You Want To Recover");
int rp=Integer.parseInt(br.readLine());
if((rp<1)||(rp>n))
System.out.println("\nInvalid Process. Enter A Valid ID");
else if((prc[rp-1]==0)&&(rp>cood))
{
prc[rp-1]=1;
System.out.println("\nProcess "+rp+" Has Recovered");
cood=rp;
System.out.println("\nProcess "+rp+ " Is The New Coordinator");
}
else if(crash==n)
{
prc[rp-1]=1;
```

```
cood=rp;
System.out.println("\nProcess "+rp+ " Is The New Coordinator");
crash--;
}
else if((prc[rp-1]==0)&&(rp<cood))
{
prc[rp-1]=1;
System.out.println("\nProcess "+rp+" Has Recovered");
}
else
System.out.println("\nProcess "+rp+" Is Not A Crashed Process");
break;
case 3: System.out.println("\nCurrent Coordinator Is "+cood);
break;
case 4: System.exit(0);
break;
default: System.out.println("\nInvalid Entry!");
 break;
}
}
while(ch!=4);
}
public static void main(String args[]) throws IOException
{
BullyAlgo ob=new BullyAlgo();
ob.Bully();
```

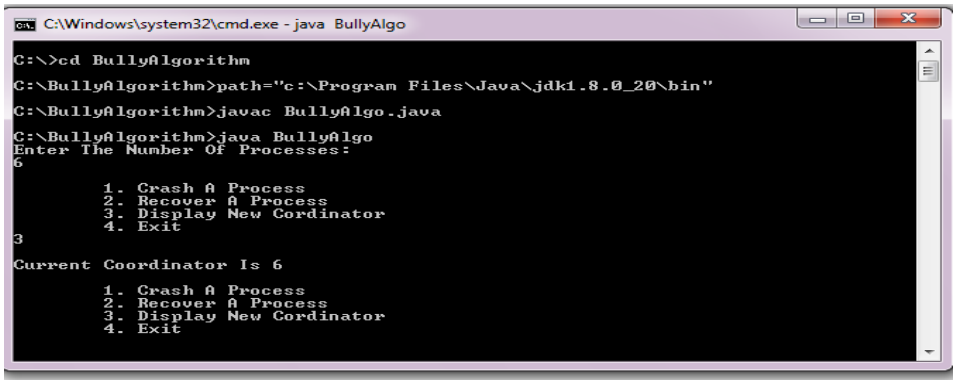
}

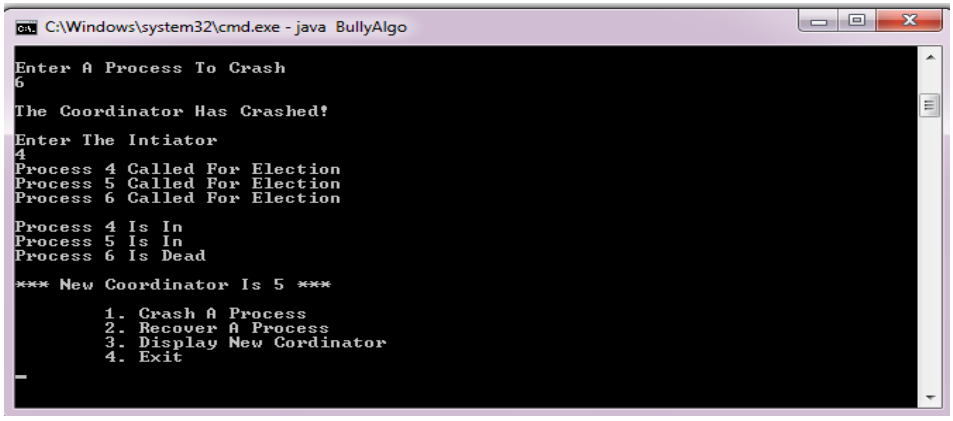
}

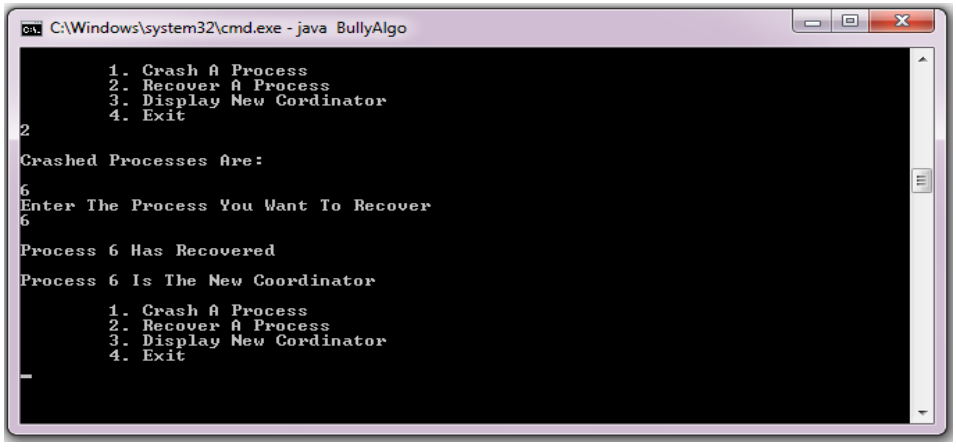
Compile the program as follows:



Output of the program is showed as follows:


C:\Windows\system32\cmd.exe - java BullyAlgo
C:\>cd BullyAlgorithm
C:\BullyAlgorithm>path="c:\Program Files\Java\jdk1.8.0_20\bin"
C:\BullyAlgorithm>javac BullyAlgo.java
C:\BullyAlgorithm>java BullyAlgo
Enter The Number Of Processes:
6
1. Crash A Process
2. Recover A Process
3. Display New Cordinator
4. Exit
3
Current Coordinator Is 6
1. Crash A Process
2. Recover A Process
3. Display New Cordinator
4. Exit


C:\Windows\system32\cmd.exe - java BullyAlgo
Enter A Process To Crash
6
The Coordinator Has Crashed!
Enter The Intiator
4
Process 4 Called For Election
Process 5 Called For Election
Process 6 Called For Election
Process 4 Is In
Process 5 Is In
Process 6 Is Dead
*** New Coordinator Is 5 ***
1. Crash A Process
2. Recover A Process
3. Display New Cordinator
4. Exit


C:\Windows\system32\cmd.exe - java BullyAlgo
1. Crash A Process
2. Recover A Process
3. Display New Cordinator
4. Exit
2
Crashed Processes Are:
6
Enter The Process You Want To Recover
6
Process 6 Has Recovered
Process 6 Is The New Coordinator
1. Crash A Process
2. Recover A Process
3. Display New Cordinator
4. Exit

Experiment No. 8

Aim– Program to demonstrate Clock Synchronization Algorithm using Java.

Program to demonstrate Berkeley clock synchronization algorithm

Berkeley.java

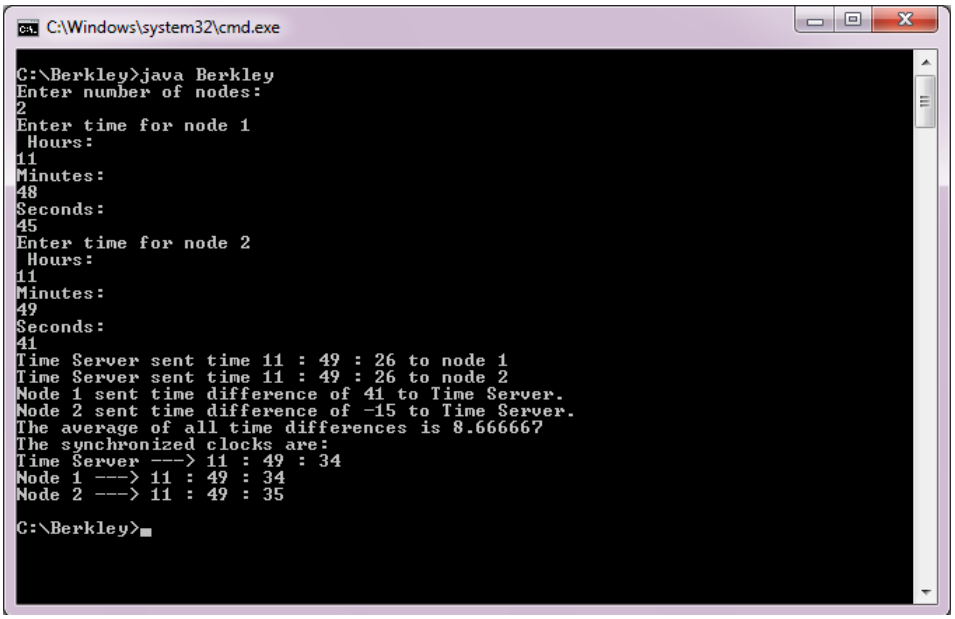
```
import java.io.*;
import java.util.*;
public class Berkley
 {
float diff(int h, int m, int s, int nh, int nm, int ns){
int dh = h-nh;
int dm = m-nm;
int ds = s-ns;
int diff = (dh*60*60)+(dm*60)+ds;
return diff;
}
float average(float diff[], int n){
int sum=0;
for(int i=0; i<n; i++)
{
sum+=diff[i];
}
float average = (float)sum/(n+1);
System.out.println("The average of all time differences is "+average);
return average;
}
void sync(float diff[], int n, int h, int m, int s, int nh[], int nm[], int ns[], float average)
 {
for(int i=0;i<n;i++)
{
```

```
diff[i]+=average;
int dh=(int)diff[i]/(60*60);
diff[i]%=(60*60);
int dm=(int)diff[i]/60;
diff[i]%=60;
int ds=(int)diff[i];
nh[i]+=dh;
if(nh[i]>23)
{
nh[i]%=24;
}
nm[i]+=dm;
if(nm[i]>59)
{
nh[i]++;
nm[i]%=60;
}
ns[i]+=ds;
if(ns[i]>59)
{
nm[i]++;
ns[i]%=60;
}
if(ns[i]<0)
{
nm[i]--;
ns[i]+=60;
}
}
h+=(int)(average/(60*60));
if(h>23)
```

```
{
h%=24;
}
m+=(int)(average/(60*60*60));
if(m>59)
{
h++;
m%=60;
}
s+=(int)(average%(60*60*60));
if(s>59)
{
 m++;
s%=60;
}
if(s<0)
{
m--;
s+=60;
}
System.out.println("The synchronized clocks are:\nTime Server ---> "+h+" : "+m+" : "+s);
for(int i=0;i<n;i++)
{
System.out.println("Node "+(i+1)+" ---> "+nh[i]+" : "+nm[i]+" : "+ns[i]);
}
}
public static void main(String[] args) throws IOException {
Berkley b = new Berkley();
Date date = new Date();
BufferedReader obj = new BufferedReader(new InputStreamReader(System.in));
System.out.println("Enter number of nodes:");
```

```
int n = Integer.parseInt(obj.readLine());
int h = date.getHours();
int m = date.getMinutes();
int s = date.getSeconds();
int nh[] = new int[n];
int nm[] = new int[n];
int ns[] = new int[n];
for(int i=0; i<n; i++)
{
System.out.println("Enter time for node "+(i+1)+"\n Hours:");
nh[i]=Integer.parseInt(obj.readLine());
System.out.println("Minutes:");
nm[i]=Integer.parseInt(obj.readLine());
System.out.println("Seconds:");
ns[i]=Integer.parseInt(obj.readLine());
}
for(int i=0; i<n; i++)
{
System.out.println("Time Server sent time "+h+" : "+m+" : "+s+" to node "+(i+1));
}
float diff[] = new float[n];
for(int i=0;i<n;i++)
{
diff[i] = b.diff(h,m,s,nh[i],nm[i],ns[i]);
System.out.println("Node "+(i+1)+" sent time difference of "+(int)diff[i]+" to Time Server.");
}
float average = b.average(diff,n);
b.sync(diff, n, h, m, s, nh, nm, ns, average);
}
}
```

Output of above program is shown as follows:



Experiment No. 9

Aim– Program to demonstrate Mutual Exclusion Algorithm using Java.

Program to demonstrate Ring Algorithm for Mutual Exclusion

The program for mutual exclusion is composed of three sub programs, namely a server program to co-ordinate the clients and two client programs to exchange the tokens amongst them for sending and receiving messages.

Step 1 – Write and Compile Server program MutualServer.java.

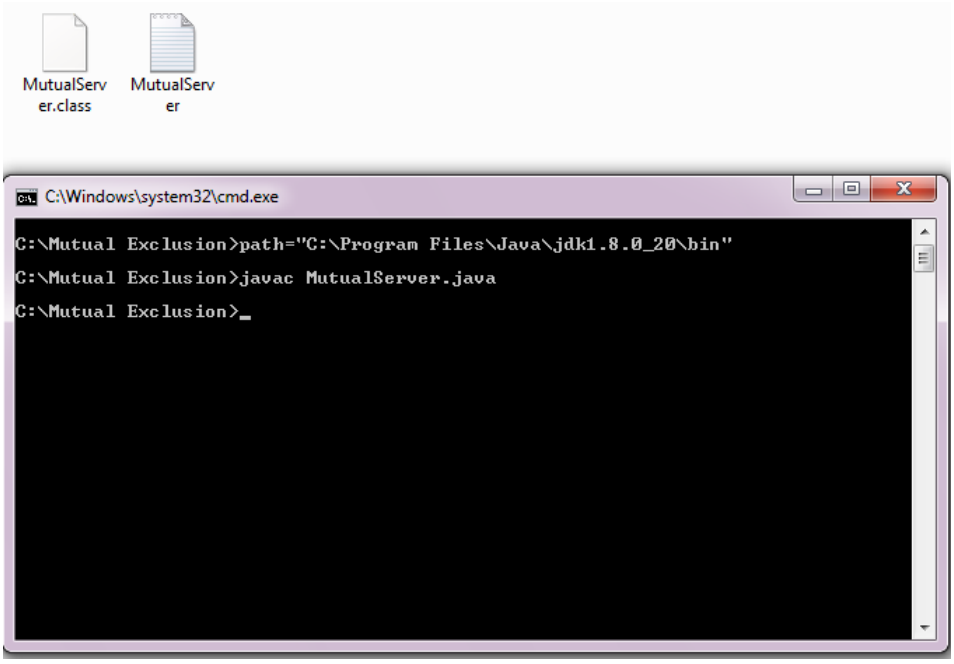
MutualServer.java.

```
import java.io.*;
import java.net.*;
public class MutualServer implements Runnable
```

```
{
Socket socket=null;
static ServerSocket ss;
MutualServer(Socket newSocket)
{
this.socket=newSocket;
}
public static void main(String args[]) throws IOException
{
ss=new ServerSocket(7000);
System.out.println("Server Started");
while(true)
{
Socket s = ss.accept();
MutualServer es = new MutualServer(s);
Thread t = new Thread(es);
t.start();
}
}
public void run()
{
Try
 {
BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
while(true)
```

```
{
System.out.println(in.readLine());
}
}
catch(Exception e){ }
}
}
```

Compile Server Program as follows:



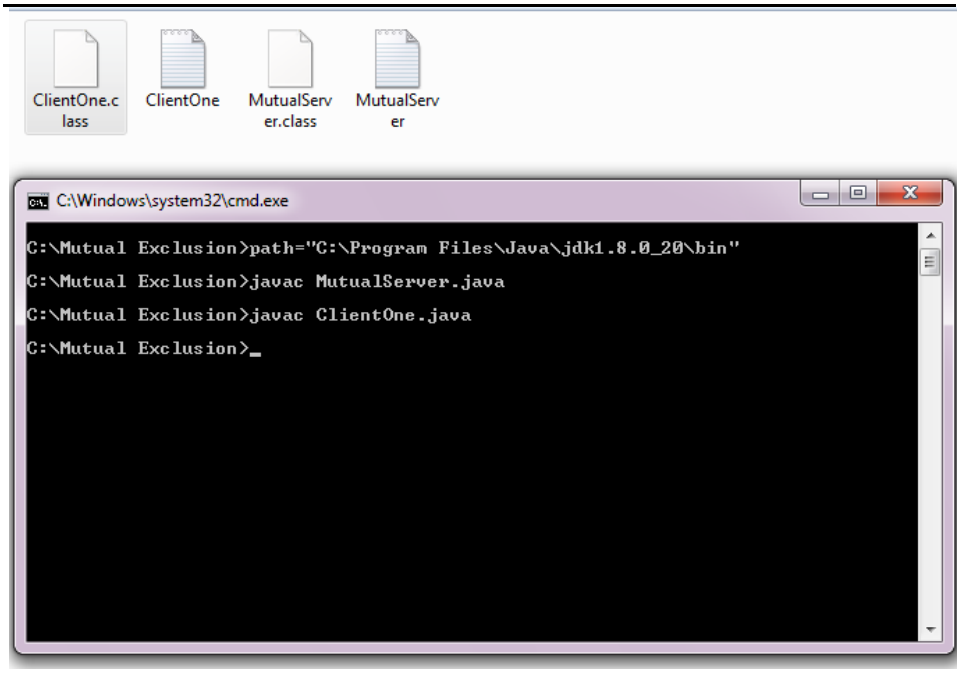
Step 2 – Write and Compile First client program ClientOne.java.

ClientOne.java

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

```
import java.net.*;
public class ClientOne
{
public static void main(String args[])throws IOException
{
Socket s=new Socket("localhost",7000);
PrintStream out = new PrintStream(s.getOutputStream());
ServerSocket ss = new ServerSocket(7001);
Socket s1 = ss.accept();
BufferedReader in1 = new BufferedReader(new
InputStreamReader(s1.getInputStream()));
PrintStream out1 = new PrintStream(s1.getOutputStream());
BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
String str="Token";
while(true)
{
if(str.equalsIgnoreCase("Token"))
{
System.out.println("Do you want to send some data");
System.out.println("Enter Yes or No");
str=br.readLine();
if(str.equalsIgnoreCase("Yes"))
{
System.out.println("Enter the data");
str=br.readLine();
```

```
out.println(str);
}
out1.println("Token");
}
System.out.println("Waiting for Token");
str=in1.readLine();
}
}
}
```

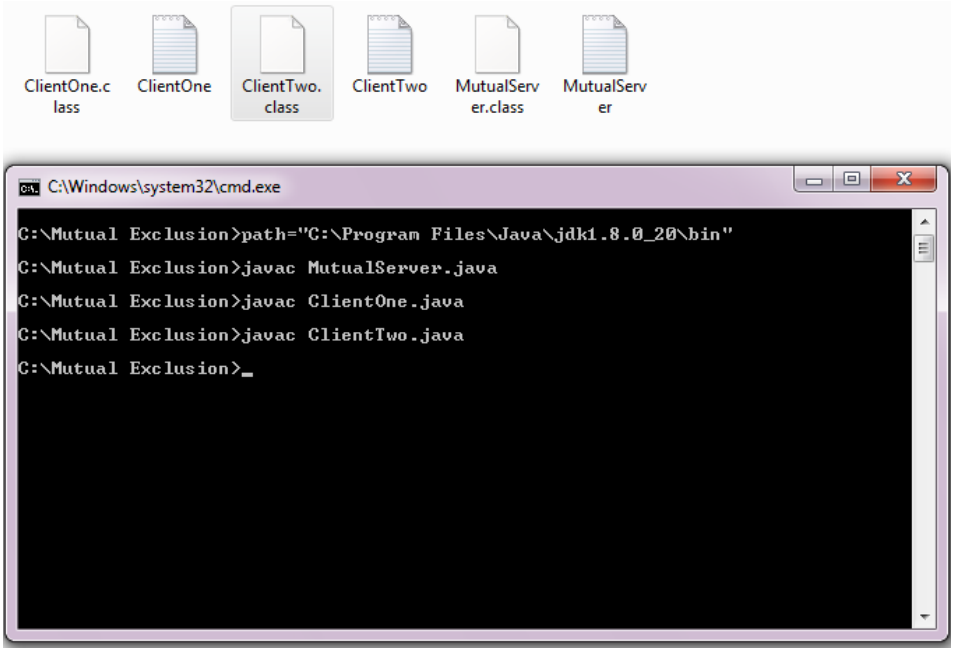


Step 3 – Write and Compile Second client program ClientTwo.java.

ClientTwo.java

```
import java.io.*;
import java.net.*;
public class ClientTwo
{
public static void main(String args[])throws IOException
{
Socket s=new Socket("localhost",7000);
PrintStream out = new PrintStream(s.getOutputStream());
Socket s2=new Socket("localhost",7001);
BufferedReader in2 = new BufferedReader(new
InputStreamReader(s2.getInputStream()));
PrintStream out2 = new PrintStream(s2.getOutputStream());
BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
String str;
while(true)
{
System.out.println("Waiting for Token");
str=in2.readLine();
if(str.equalsIgnoreCase("Token"))
{
System.out.println("Do you want to send some data");
System.out.println("Enter Yes or No");
str=br.readLine();
if(str.equalsIgnoreCase("Yes"))
{
```

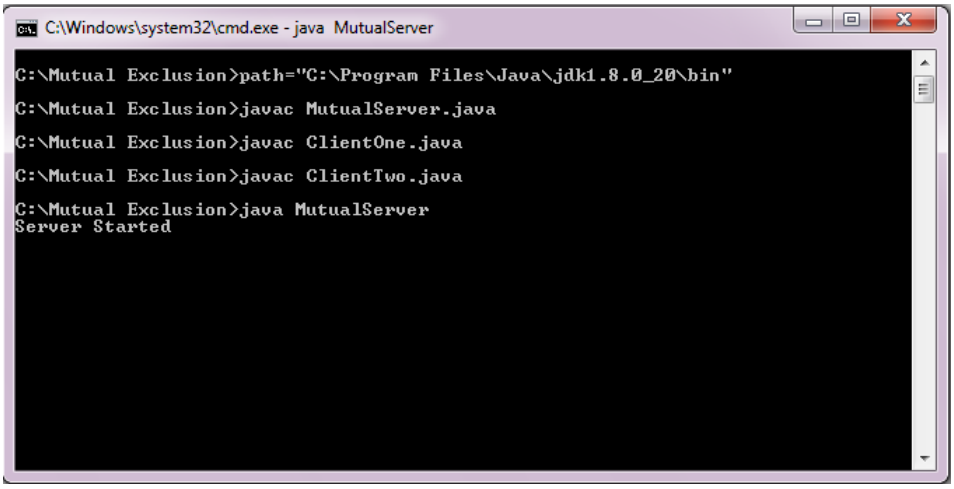
```
System.out.println("Enter the data");
str=br.readLine();
out.println(str);
}
out2.println("Token");
}
}
}
}
```



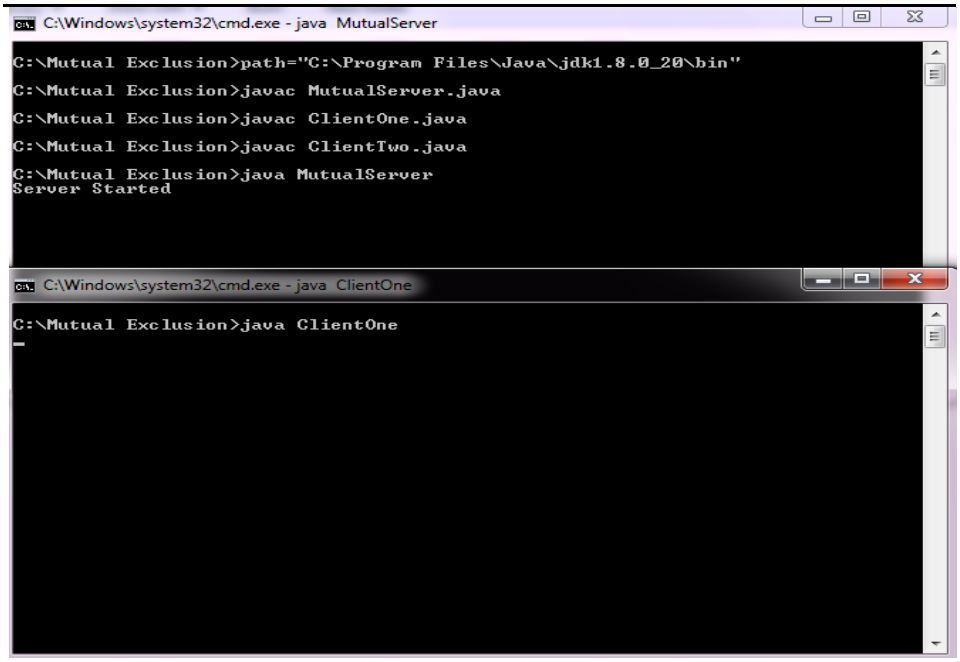
Step 4 – Run Server Program and keep it running till we connect the clients.

Commented [AS4]: Refer to the same comment earlier.

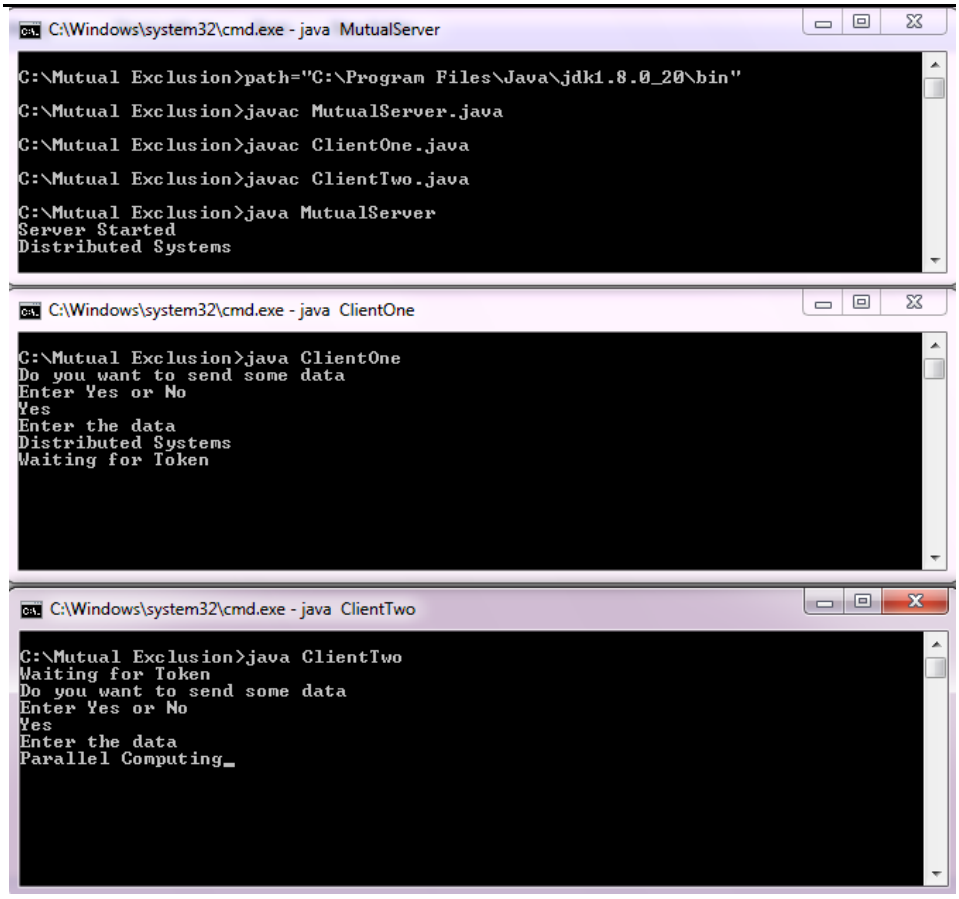
Commented [A5R4]: Modifies the sentences



Step 5 – Open new Command prompt and Run ClientOne Program on it and keep it running till ClientTwo starts.



Step 6 – Open one more Command prompt to Run ClientTwo Program. The output allows both the clients to use token and share their messages with each other using Token Ring. To send the message, the client has to accept the token by typing type Yes followed by the message alternately and has to type No to release the token.



Experiment No. 10

Aim– Program to demonstrate Deadlock Management in Distributed Systems.

Steps to Demonstrate CORBA Application using Java

The CORBA Application is composed of three programs:

a) **idl program–** which contains the declaration of methods to be called by the client and defined by the server program. The return type of method or parameters should not be integer as CORBA does not support integer data type; instead short or double can be used.

b) **Server Program** – which contains definition of the methods which are declared in idl file and called by the client program.

c) **Client program** – which contains Method calling defined at the server.

The following steps will explain the CORBA program to print Hello World message:

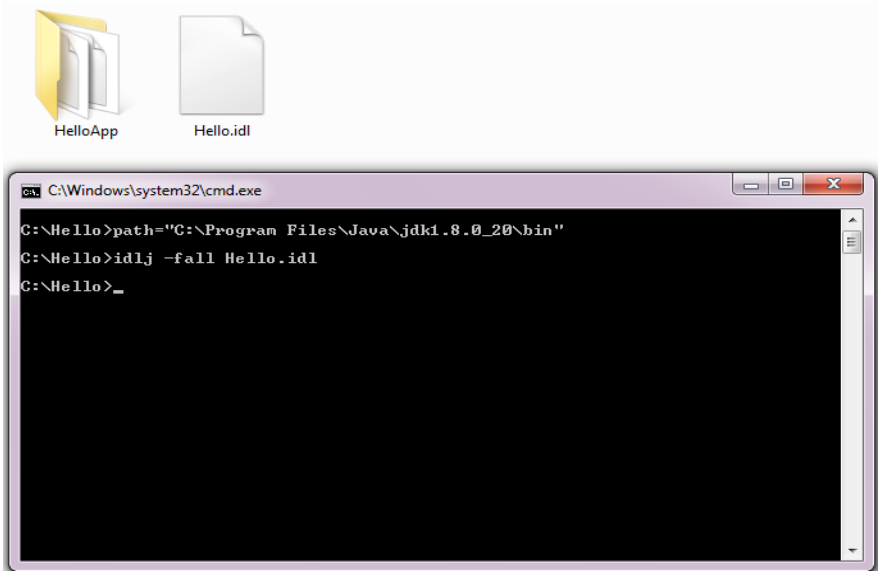
Step 1 – Create Hello.idl file

To create the Hello.idl file, create a new directory, named Hello, for this application. Start your favorite text editor and create a file named Hello.idl in this directory. In your file, enter the code for the interface definition, **Hello.idl**:

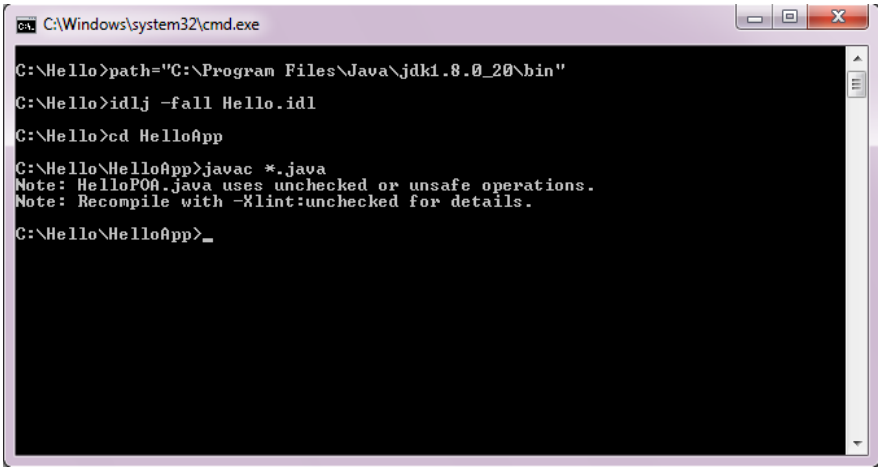
```
module HelloApp
{
 interface Hello
 {

  //sayHello() is declared which can be replaced with own method declaration
```

```
    string sayHello();
    oneway void shutdown();
  };
};
```

Save the file and Compile this program using idlj compiler using– fall options. After the compilation of idl file, it generates HelloApp directory, which contains supporting files to run CORBA application like helper, holder, poa etc.



Step 2 – Compile all the files inside HelloApp directory using javac compiler.



Step 3 – Write Server program which contains definition of sayHello() method declaired in idl file. Compile this program using java compiler.

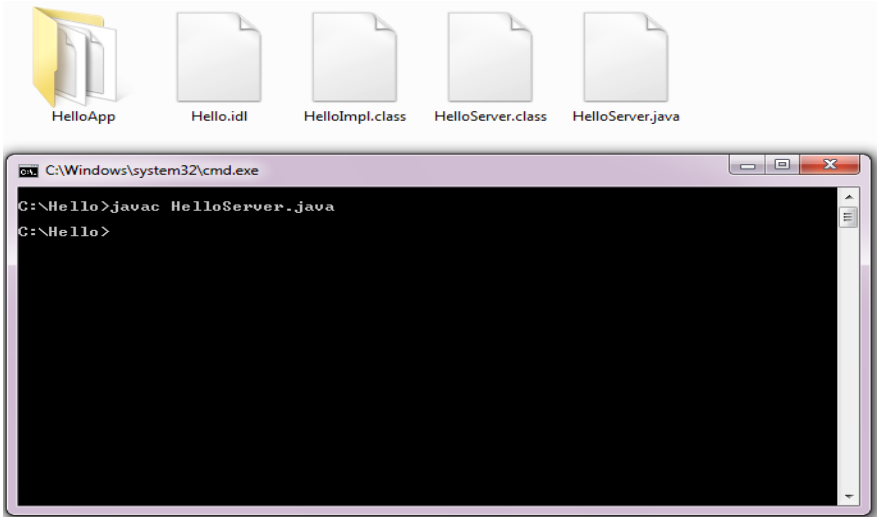
HelloServer.java

```
import HelloApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import org.omg.PortableServer.POA;
import java.util.Properties;
class HelloImpl extends HelloPOA
{
private ORB orb;
public void setORB(ORB orb_val)
{
 orb = orb_val;
```

```
}
// implement sayHello() method this definition can be replaced with own method
public String sayHello()
{
 return "\nHello world !!\n";
}
 // implement shutdown() method
public void shutdown()
{
orb.shutdown(true);
}
}
public class HelloServer
{
public static void main(String args[])
{
try
{
 // create and initialize the ORB
 ORB orb = ORB.init(args, null);
 // get reference to rootpoa & activate the POAManager
 POA rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
 rootpoa.the_POAManager().activate();
// create servant and register it with the ORB
 HelloImpl helloImpl = new HelloImpl();
 helloImpl.setORB(orb);
// get object reference from the servant
org.omg.CORBA.Object ref = rootpoa.servant_to_reference(helloImpl);
Hello href = HelloHelper.narrow(ref);
 // get the root naming context
org.omg.CORBA.Object objRef =
```

```
orb.resolve_initial_references("NameService");
 // Use NamingContextExt which is part of the Interoperable
 // Naming Service (INS) specification.
NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
 // bind the Object Reference in Naming
String name = "Hello";
NameComponent path[] = ncRef.to_name( name );
ncRef.rebind(path, href);
System.out.println("HelloServer ready and waiting ...");
 // wait for invocations from clients
orb.run();
}
 catch (Exception e) {
System.err.println("ERROR: " + e);
e.printStackTrace(System.out);
}
System.out.println("HelloServer Exiting ...");
}
}
```

Compile this program, which generates HelloServer and HelloImpl class files.

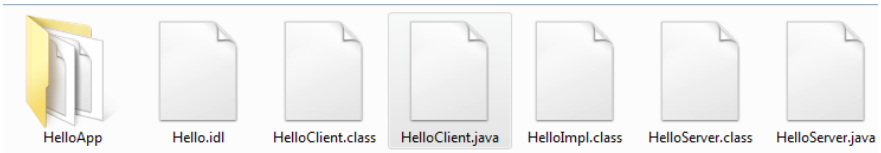


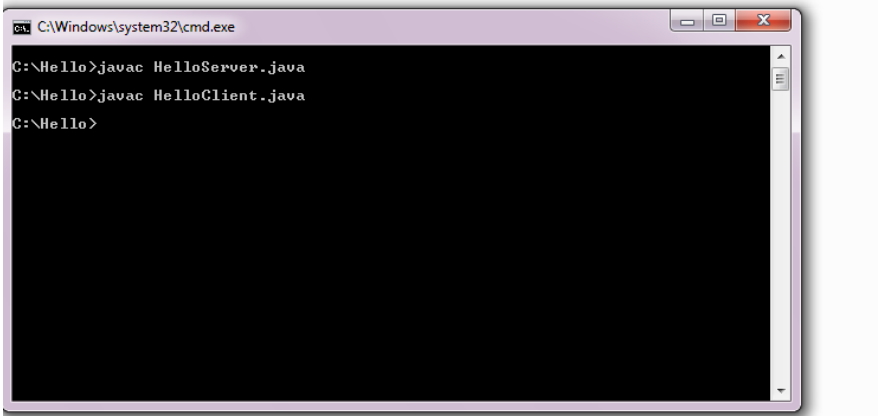
Step 4 – Write Client program, which contains calling of sayHello() method. Compile this program using java compiler only.

HelloClient.java

```
import HelloApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
public class HelloClient
{
static Hello helloImpl;
public static void main(String args[])
{
try
{
 // create and initialize the ORB
```

```
ORB orb = ORB.init(args, null);
// get the root naming context
org.omg.CORBA.Object objRef =
orb.resolve_initial_references("NameService");
// Use NamingContextExt instead of NamingContext. This is
// part of the Interoperable naming Service.
NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
// resolve the Object Reference in Naming
String name = "Hello";
helloImpl = HelloHelper.narrow(ncRef.resolve_str(name));
System.out.println("Obtained a handle on server object: " + helloImpl);
System.out.println(helloImpl.sayHello());
helloImpl.shutdown();
}
catch (Exception e) {
System.out.println("ERROR : " + e) ;
e.printStackTrace(System.out);
}
}
}
```

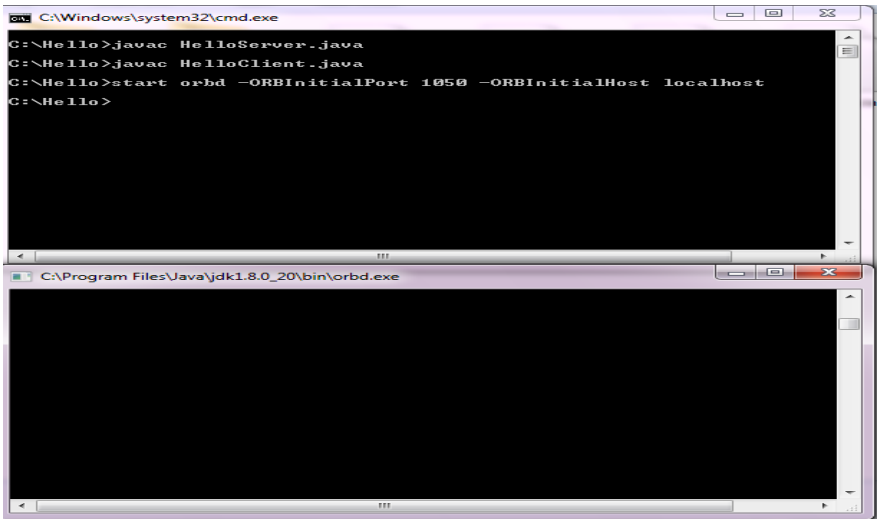




Step 5 – Start CORBA Registry using the following command and minimize it.

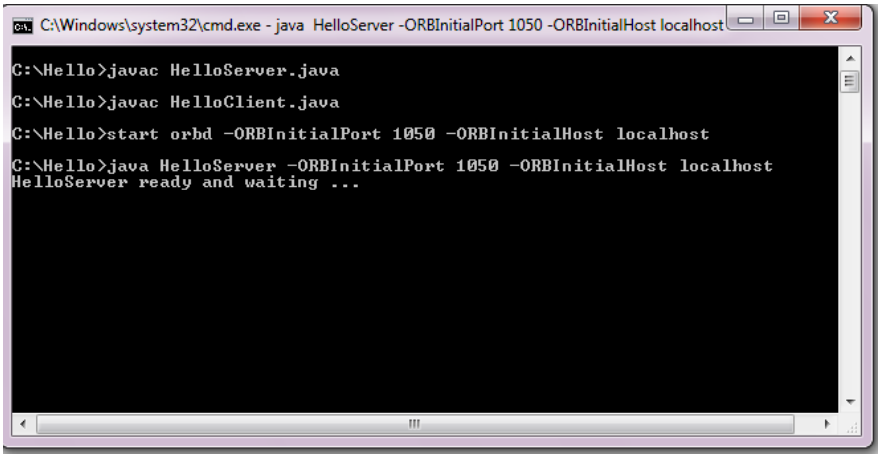
(Note – Please do not close Registry in between otherwise server and client programs will not run or will generate errors)

start orbd -ORBInitialPort 1050 -ORBInitialHost localhost



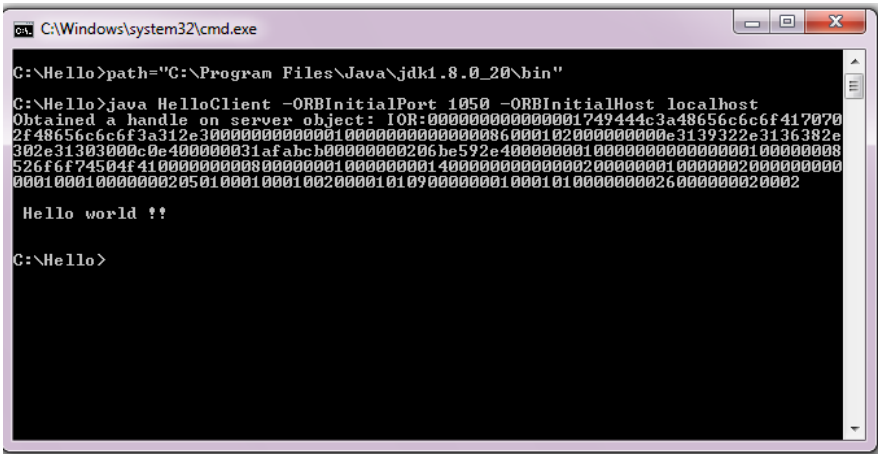
Step 6 – Run Server Program using the following command and keep it running.

java HelloServer -ORBInitialPort 1050 -ORBInitialHost localhost



Step 7 – Open one more command prompt and run client program on it using the following command:

java HelloClient -ORBInitialPort 1050 -ORBInitialHost localhost



Finally Hello World output will be displayed followed by bytecode.

Experiment No. 11

Aim– Demonstrate Hadoop Distributed File System

The Apache Hadoop is an open source software framework that enables distributed processing of large data sets across clusters of commodity servers using programming models. It is designed to scale up from a single server to thousands of machines, with a very high degree of fault tolerance. The Hadoop framework consist of two main components namely: Hadoop distributed file system (HDFS), Map-reduce programming model. The HDFS is a Hadoop implementation of distributed file system design that holds a large amount of data and provides easier way of access to many clients distributed across the network. It is highly fault tolerant and designed to be run on low cost hardware (called commodity hardware). The files in HDFS are stored across the multiple machines in a redundant fashion to recover the data loss in the case of failure. The commands to perform different file management operations on HDFS are as follows:

Commented [AS6]: Meaning unclear.

Commented [A7R6]: Rewritten

Sr. No.	Command	Description
1	#hadoop fs –ls	Lists the files
2	#hadoop fs –count hdfs:/	Counts the number of directories, files and bytes under the paths
3	#hadoop fs –mkdir /user /hadoop	Creates a new directory hadoop under user directory
4	#hadoop fs –rm hadoop/cust	Deletes file cust from hadoop directory
5	#hadoop fs –mv /user/training/cust hadoop/	Moves file cust from /user/training directory to hadoop directory
6	#hadoop fs –cp /user/training/cust hadoop/	Copies file cust from /user/training directory to hadoop directory
7	#hadoop fs –copyToLocal hadoop/a.txt /home/training/	Copies file a.txt to local disk from HDFS
8	#hadoop fs –copyFromLocal /home/training/a.txt hadoop/	Copies file a.txt from local directory /home/training to HDFS

Experiment No. 12

Aim– Deadlock Management in Distributed Systems

A deadlock is a condition in a system where a set of processes (or threads) have requests for resources that can never be satisfied. Essentially, a process cannot proceed because it needs to obtain a resource held by another process; but, it itself is holding a resource that the other process needs. There are four conditions to be met for a deadlock to occur in a system:

1. Mutual exclusion: A resource can be held by at most one process.
2. Hold and wait: Processes that already hold resources can wait for another resource.
3. Non-preemption: A resource, once granted, cannot be taken away.
4. Circular wait: Two or more processes are waiting for resources held by one of the other processes.

The banker's algorithm is a resource allocation and deadlock avoidance algorithm used in distributed system. The implementation of banker's algorithm in Java is as follows:

Bankers algorithm for deadlock detection

```
import java.util.Scanner;
public class Bankers{
   private int need[][],allocate[][],max[][],avail[][],np,nr;
   private void input(){
    Scanner sc=new Scanner(System.in);
    System.out.print("Enter no. of processes and resources : ");
    np=sc.nextInt();  //no. of process
    nr=sc.nextInt();  //no. of resources
    need=new int[np][nr];  //initializing arrays
    max=new int[np][nr];
    allocate=new int[np][nr];
    avail=new int[1][nr];
```

```
 System.out.println("Enter allocation matrix -->");
 for(int i=0;i<np;i++)
      for(int j=0;j<nr;j++)
     allocate[i][j]=sc.nextInt();  //allocation matrix

 System.out.println("Enter max matrix -->");
 for(int i=0;i<np;i++)
      for(int j=0;j<nr;j++)
     max[i][j]=sc.nextInt();  //max matrix

    System.out.println("Enter available matrix -->");
    for(int j=0;j<nr;j++)
     avail[0][j]=sc.nextInt();  //available matrix

    sc.close();
}

private int[][] calc_need(){
   for(int i=0;i<np;i++)
     for(int j=0;j<nr;j++)  //calculating need matrix
      need[i][j]=max[i][j]-allocate[i][j];

   return need;
}

private boolean check(int i){
   //checking if all resources for ith process can be allocated
   for(int j=0;j<nr;j++)
   if(avail[0][j]<need[i][j])
      return false;
```

```
return true;
}

public void isSafe(){
  input();
  calc_need();
  boolean done[]=new boolean[np];
  int j=0;

  while(j<np){  //until all process allocated
  boolean allocated=false;
  for(int i=0;i<np;i++)
   if(!done[i] && check(i)){  //trying to allocate
       for(int k=0;k<nr;k++)
       avail[0][k]=avail[0][k]-need[i][k]+max[i][k];
    System.out.println("Allocated process : "+i);
    allocated=done[i]=true;
          j++;
         }
     if(!allocated) break;  //if no allocation
  }
  if(j==np)  //if all processes are allocated
   System.out.println("\nSafely allocated");
  else
   System.out.println("All processes cannot be allocated safely");
}

public static void main(String[] args) {
  new Bankers().isSafe();
}
```

}

The Output of above program is as follows: