

## **Assignment No .1**

### **AddClient.java :**

```
import java.rmi.*;

public class AddClient {

    public static void main(String args[]) {

        try { String addServerURL = "rmi://" + args[0] + "/AddServer";

            AddServerIntf addServerIntf =

                (AddServerIntf)Naming.lookup(addServerURL);

            System.out.println("The first number is: " + args[1]);

            double d1 = Double.valueOf(args[1]).doubleValue();

            System.out.println("The second number is: " + args[2]);

            double d2 = Double.valueOf(args[2]).doubleValue();

            System.out.println("The sum is: " + addServerIntf.add(d1, d2));

        } catch (Exception e) { System.out.println("Exception: " + e); } } }
```

### **AddServer.java :**

```
import java.net.*;
import java.rmi.*;

public class AddServer {

    public static void main(String args[]) {

        try { AddServerImpl addServerImpl = new AddServerImpl();

            Naming.rebind("AddServer", addServerImpl);

        } catch (Exception e) { System.out.println("Exception: " + e); } } }
```

### **AddServerImpl.java :**

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

public class AddServerImpl extends UnicastRemoteObject

    implements AddServerIntf { public AddServerImpl() throws RemoteException {

    } public double add(double d1, double d2) throws RemoteException {

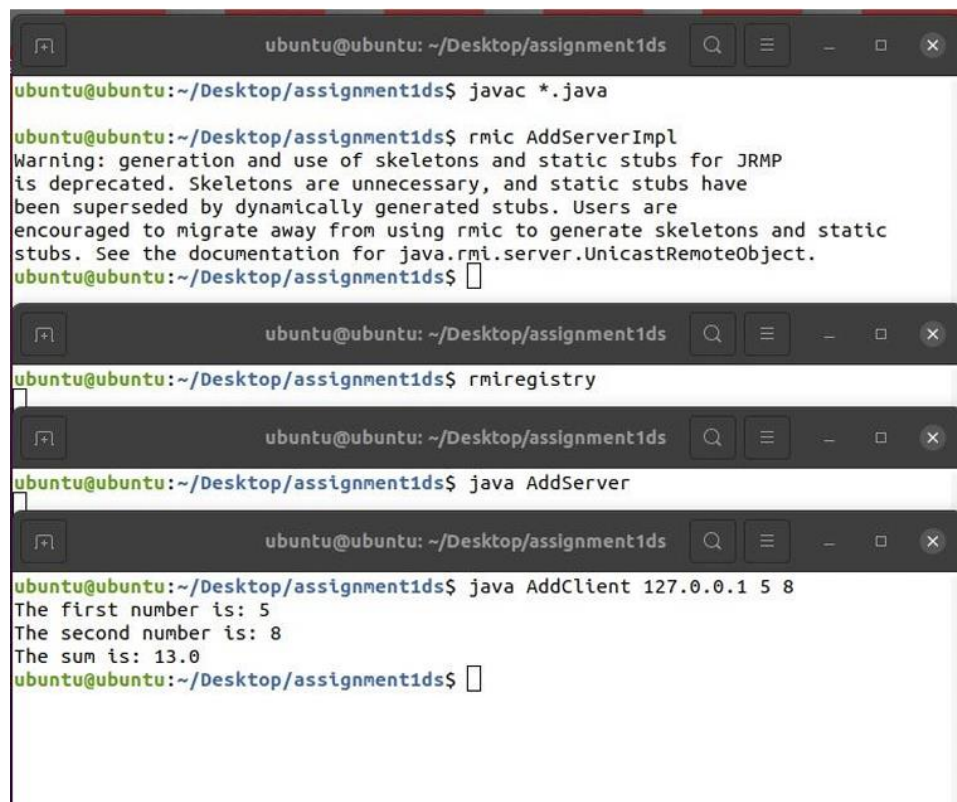
        return d1 + d2; } }
```

### **AddServerIntf.java :**

```
import java.rmi.*;

public interface AddServerIntf extends Remote { double add(double d1, double d2) throws

    RemoteException; }
```



```
ubuntu@ubuntu: ~/Desktop/assignment1ds
ubuntu@ubuntu:~/Desktop/assignment1ds$ javac *.java

ubuntu@ubuntu:~/Desktop/assignment1ds$ rmic AddServerImpl
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been superseded by dynamically generated stubs.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.
ubuntu@ubuntu:~/Desktop/assignment1ds$

ubuntu@ubuntu:~/Desktop/assignment1ds$ rmiregistry

ubuntu@ubuntu:~/Desktop/assignment1ds$ java AddServer

ubuntu@ubuntu:~/Desktop/assignment1ds$ java AddClient 127.0.0.1 5 8
The first number is: 5
The second number is: 8
The sum is: 13.0
ubuntu@ubuntu:~/Desktop/assignment1ds$
```

#### AddClient.java:

- This is a client-side Java program that interacts with a remote server via RMI (Remote Method Invocation).
- It accepts command-line arguments for the server's hostname or IP address, as well as two numbers to be added.
- It constructs an RMI URL to locate the remote server.
- It uses the Naming.lookup() method to obtain a reference to the remote object from the RMI registry.
- It calls the add() method on the remote object to perform the addition operation and prints the result.

#### AddServer.java:

- This is the main class for the server-side program.
- It creates an instance of AddServerImpl, which is the implementation of the remote interface.
- It uses Naming.rebind() to bind the remote object to the RMI registry, making it available for remote invocation.

#### AddServerImpl.java:

- This class implements the remote interface AddServerIntf.
- It extends UnicastRemoteObject to make the object available for remote method invocation.
- It provides an implementation for the add() method, which adds two numbers and returns the result.

#### AddServerIntf.java:

- This is the remote interface that defines the method(s) that can be invoked remotely.
- It extends Remote to mark it as a remote interface.
- It declares a single method add() which takes two doubles as parameters and returns their sum.

## **Assignment No .2**

ReverseClient.java :

```
// Client

import ReverseModule.*;

import org.omg.CosNaming.*;

import org.omg.CosNaming.NamingContextPackage.*;

import org.omg.CORBA.*;

import java.io.*;

class ReverseClient

{

public static void main(String args[])

{

Reverse ReverselImpl=null;

try

{

// initialize the ORB object request broker

org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);

org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");

NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

String name = "Reverse";

// narrow converts generic object into string type

ReverselImpl = ReverseHelper.narrow(ncRef.resolve_str(name));

System.out.println("Enter String=");

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

String str= br.readLine();

String tempStr= ReverselImpl.reverse_string(str);

System.out.println(tempStr);

}

catch(Exception e)

{

e.printStackTrace();

}

}

}
```

ReverselImpl.java :

```

import ReverseModule.ReversePOA;

import java.lang.String;

class ReverseImpl extends ReversePOA
{
    ReverseImpl()
    {
        super();

        System.out.println("Reverse Object Created");
    }

    public String reverse_string(String name)
    {
        StringBuffer str=new StringBuffer(name);

        str.reverse();

        return (("Server Send "+str));
    }
}

ReverseModule.idl :
module ReverseModule
{
    interface Reverse
    {
        string reverse_string(in string str);
    };
};

ReverseServer.java :
import ReverseModule.Reverse;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;

class ReverseServer
{
    public static void main(String[] args)
    {
        try

```

```

{
// initialize the ORB
org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);
// initialize the BOA/POA
POA rootPOA = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
rootPOA.the_POAManager().activate();
// creating the object
ReverseImpl rvr = new ReverseImpl();
// get the object reference from the servant class
org.omg.CORBA.Object ref = rootPOA.servant_to_reference(rvr);
System.out.println("Step1");
Reverse h_ref = ReverseModule.ReverseHelper.narrow(ref);
System.out.println("Step2");
org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
System.out.println("Step3");
NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
System.out.println("Step4");
String name = "Reverse";
NameComponent path[] = ncRef.to_name(name);
ncRef.rebind(path,h_ref);
System.out.println("Reverse Server reading and waiting....");
orb.run();
} catch(Exception e)
{ e.printStackTrace(); } } }

```

```
ubuntu@ubuntu: ~/Desktop/Assignment2
ubuntu@ubuntu:~/Desktop/Assignment2$ idlj -fall ReverseModule.idl
ubuntu@ubuntu:~/Desktop/Assignment2$ javac *.java ReverseModule/*.java
ReverseModule/_ReverseStub.java:46: warning: IORCheckImpl is internal proprietary API and may be removed in a future release
    com.sun.corba.se.impl.orbutil.IORCheckImpl.check(str, "ReverseModule._ReverseStub");
                                         ^
Note: ReverseModule/ReversePOA.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
1 warning
ubuntu@ubuntu:~/Desktop/Assignment2$ orbd -ORBInitialPort 1056&
[1] 3657
ubuntu@ubuntu:~/Desktop/Assignment2$ java ReverseServer -ORBInitialPort 1056&
[2] 3681
ubuntu@ubuntu:~/Desktop/Assignment2$ Reverse Object Created
Step1
Step2
Step3
Step4
Reverse Server reading and waiting....

ubuntu@ubuntu:~/Desktop/Assignment2
ubuntu@ubuntu:~/Desktop/Assignment2$ java ReverseClient -ORBInitialPort 1056 -ORBInitialHost localhost
Enter String=
assignment 2
Server Send  2 tnenmgissa
ubuntu@ubuntu:~/Desktop/Assignment2$
```

### **Explanation:-**

- **Client-Server Architecture:**
- The code implements a simple client-server architecture using Common Object Request Broker Architecture (CORBA).
- It consists of a client (ReverseClient.java) and a server (ReverseServer.java).
- **IDL Definition:**
- The ReverseModule.idl file defines a module named ReverseModule containing an interface Reverse.
- The interface declares a method reverse\_string that takes a string input and returns a reversed string.
- **Server Implementation (ReverseImpl.java):**
- The ReverseImpl class implements the Reverse interface.
- It provides the implementation for the reverse\_string method, which reverses the input string and returns it.
- **Server Setup (ReverseServer.java):**
- The server code initializes the ORB (Object Request Broker) and the Portable Object Adapter (POA).
- It creates an instance of ReverseImpl and activates the POA.
- The server registers the ReverseImpl object with the naming service so that clients can look it up.
- **Client Implementation (ReverseClient.java):**
- The client code initializes the ORB and obtains a reference to the naming service.
- It looks up the Reverse object from the naming service and obtains its reference.
- The client prompts the user to input a string, sends it to the server for reversal, and prints the reversed string received from the server.
- **Naming Service:**
- The naming service (obtained via orb.resolve\_initial\_references("NameService")) allows objects to be registered and looked up by name.
- **Communication:**
- Communication between the client and server is established through CORBA, using method invocation on remote objects.
- **Exception Handling:**
- Both client and server code have exception handling to catch and print any errors that occur during execution.
- **Output:**
- The server and client code print various status messages and error stack traces to the console for debugging and monitoring purposes.

### **Assignment No .3**

```
import mpi.MPI;
import java.util.Scanner;
import mpi.*;
public class ArrSum {
    public static void main(String[] args) throws Exception{
        MPI.Init(args);
        int rank = MPI.COMM_WORLD.Rank();
        int size = MPI.COMM_WORLD.Size();
        int unitsize = 5;
        int root = 0;
        int send_buffer[] = null;
        // 1 process is expected to handle 4 elements
        send_buffer = new int [unitsize * size];
        int recieve_buffer[] = new int [unitsize];
        int new_recieve_buffer[] = new int [size];
        // Set data for distribution
        if(rank == root) {
            int total_elements = unitsize * size;
            System.out.println("Enter " + total_elements + " elements");
            for(int i = 0; i < total_elements; i++) {
                System.out.println("Element " + i + "\t = " + i);
                send_buffer[i] = i;
            }
        }

        // Scatter data to processes
        MPI.COMM_WORLD.Scatter(
            send_buffer,
            0,
            unitsize,
            MPI.INT,
            recieve_buffer,
            0,
            unitsize,
```

```

        MPI.INT,

        root

    );

    // Calculate sum at non root processes
    // Store result in first index of array
    for(int i = 1; i < unitsize; i++) {
        recieve_buffer[0] += recieve_buffer[i];
    }

    System.out.println(

        "Intermediate sum at process " + rank + " is " + recieve_buffer[0]

    );

    // Gather data from processes
    MPI.COMM_WORLD.Gather(

        recieve_buffer,

        0,

        1,

        MPI.INT,

        new_recieve_buffer,

        0,

        1,

        MPI.INT,

        root

    );


    // Aggregate output from all non root processes
    if(rank == root) {
        int total_sum = 0;
        for(int i = 0; i < size; i++) {
            total_sum += new_recieve_buffer[i];
        }

        System.out.println("Final sum : " + total_sum);
    }

    MPI.Finalize();
}
}

```



```
ubuntu@ubuntu: ~/Desktop/Assignment3
ubuntu@ubuntu:~/Desktop/Assignment3$ export MPJ_HOME=/home/ubuntu/Downloads/mpj-v0_44
ubuntu@ubuntu:~/Desktop/Assignment3$ export PATH=$MPJ_HOME/bin:$PATH
ubuntu@ubuntu:~/Desktop/Assignment3$ javac -cp $MPJ_HOME/lib/mpj.jar ArrSum.java
ubuntu@ubuntu:~/Desktop/Assignment3$ $MPJ_HOME/bin/mpjrun.sh -np 4 ArrSum
MPJ Express (0.44) is started in the multicore configuration
Enter 20 elements
Element 0      = 0
Element 1      = 1
Element 2      = 2
Element 3      = 3
Element 4      = 4
Element 5      = 5
Element 6      = 6
Element 7      = 7
Element 8      = 8
Element 9      = 9
Element 10     = 10
Element 11     = 11
Element 12     = 12
Element 13     = 13
Element 14     = 14
Element 15     = 15
Element 16     = 16
Element 17     = 17
Element 18     = 18
Element 19     = 19
Intermediate sum at process 0 is 10
Intermediate sum at process 3 is 85
Intermediate sum at process 2 is 60
Intermediate sum at process 1 is 35
Final sum : 190
ubuntu@ubuntu:~/Desktop/Assignment3$
```

### **Explanation**

- This Java program utilizes the MPI (Message Passing Interface) library for parallel computing.
- The program calculates the sum of elements in an array using parallel processing.
- It divides the array among multiple processes, where each process computes the sum of a subset of the array.
- The array is initially created and populated with data on the root process (rank 0).
- Data is scattered from the root process to other processes using `MPI.COMM_WORLD.Scatter`.
- Each non-root process calculates the sum of elements in its assigned subset.
- The intermediate sums from each process are gathered back to the root process using `MPI.COMM_WORLD.Gather`.
- Finally, the root process aggregates the intermediate sums to compute the final total sum of all elements.
- The MPI library functions used include `MPI.Init()`, `MPI.COMM_WORLD.Rank()`, `MPI.COMM_WORLD.Size()`, `MPI.COMM_WORLD.Scatter()`, `MPI.COMM_WORLD.Gather()`, and `MPI.Finalize()`.
- This program demonstrates parallel computing, which can significantly speed up the computation of tasks that can be parallelized, such as array summation, on a multi-core or distributed computing environment.

## **Assignment No .4**

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

public class Berkeley {
    // Define the port number that will be used for communication
    private static final int PORT = 9876;

    public static void main(String[] args) throws Exception {
        // Create a server socket to listen for incoming messages
        ServerSocket serverSocket = new ServerSocket(PORT);

        // Create a list to store the time differences for each node
        List<Long> timeDiffs = new ArrayList<Long>();

        // Create a new thread to handle the time requests from nodes
        Thread timeServerThread = new Thread(new Runnable() {
            public void run() {
                while (true) {
                    try {
                        // Wait for a node to connect and request the current time
                        Socket clientSocket = serverSocket.accept();
                        ObjectInputStream in = new ObjectInputStream(clientSocket.getInputStream());

                        // Read the current time from the node's request
                        Date clientTime = (Date) in.readObject();

                        // Send the current time to the node as a response
                        ObjectOutputStream out = new ObjectOutputStream(clientSocket.getOutputStream());
                        out.writeObject(new Date());

                        // Calculate the time difference between the server and the node
                        long timeDiff = (new Date().getTime() - clientTime.getTime()) / 2;
                        timeDiffs.add(timeDiff);
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
            }
        });
        timeServerThread.start();
    }
}
```

```

        // Close the input/output streams and the socket
        in.close();
        out.close();
        clientSocket.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

});

timeServerThread.start();

// Create a new thread to periodically send time requests to the server
Thread timeClientThread = new Thread(new Runnable() {
    public void run() {
        while (true) {
            try {
                // Connect to the server and send a time request
                Socket socket = new Socket("localhost", PORT);
                ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream());
                out.writeObject(new Date());

                // Read the current time from the server's response
                ObjectInputStream in = new ObjectInputStream(socket.getInputStream());
                Date serverTime = (Date) in.readObject();

                // Calculate the time difference between the node and the server
                long timeDiff = (serverTime.getTime() - new Date().getTime()) / 2;
                timeDiffs.add(timeDiff);

                // Close the input/output streams and the socket
                in.close();
                out.close();
                socket.close();
            }
        }
    }
});

```

```

        // Wait for a short period of time before sending the next time request
        Thread.sleep(1000);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
}

});
timeClientThread.start();

// Wait for a sufficient number of time differences to be recorded
Thread.sleep(10000);

// Compute the average time difference and adjust the node's clock
long sumTimeDiff = 0;
for (Long timeDiff : timeDiffs) {
    sumTimeDiff += timeDiff;
}

long avgTimeDiff = sumTimeDiff / timeDiffs.size();
System.out.println("Average time difference: " + avgTimeDiff);

// Adjust the node's clock by adding the average time difference
Calendar calendar = Calendar.getInstance();
calendar.setTime(new Date());
calendar.add(Calendar.MILLISECOND, (int) avgTimeDiff);
System.out.println("Adjusted time: " + calendar.getTime());
}
}

```

The screenshot shows an IDE with a Java file named `Berkeley.java`. The code implements a basic Berkeley algorithm for time synchronization. It features a `main` method that creates a `timeClientThread` and starts it. The thread's `run` method handles incoming connections, closes streams, waits for a short period, and catches exceptions. After a sufficient number of time differences are recorded, it computes the average time difference and adjusts the node's clock. The output window shows the results of the execution.

```
1 public class Berkeley {
10     public static void main(String[] args) throws Exception {
51         Thread timeClientThread = new Thread(new Runnable() {
52             public void run() {
68                 // Close the input/output streams and the socket
69                 in.close();
70                 out.close();
71                 socket.close();
72
73                 // Wait for a short period of time before sending the next time request
74                 Thread.sleep(millis:1000);
75             } catch (Exception e) {
76                 e.printStackTrace();
77             }
78         }
79     }
80 }
81 timeClientThread.start();
82
83 // Wait for a sufficient number of time differences to be recorded
84 Thread.sleep(millis:10000);
85
86 // Compute the average time difference and adjust the node's clock
87 long sumTimeDiff = 0;
88 for (Long timeDiff : timeDiffs) {
89     sumTimeDiff += timeDiff;
90 }
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS SEARCH ERROR

```
PS C:\Users\ASUS\Downloads\DS Codes> javac Berkeley.java
PS C:\Users\ASUS\Downloads\DS Codes> java Berkeley.java
Average time difference: 0
Adjusted time: Sun Apr 21 22:40:22 IST 2024
```

Link Explain Code Comment Code Find Bugs Code Chat Java: Ready Search Error

### **Explanation: -**

- The code implements a basic Berkeley algorithm for time synchronization among multiple nodes in a distributed system.
- It utilizes Java's networking capabilities to enable communication between a server and multiple client nodes.
- The server listens for incoming connections from client nodes and responds to time requests.
- Each client periodically sends time requests to the server and adjusts its clock based on the server's response.
- Time differences between the server and each client are calculated and stored in a list.
- After a certain period, the average time difference is computed to synchronize the clocks.
- The average time difference is then used to adjust the client's clock, aiming to minimize time discrepancies within the network.
- The adjusted time is printed out for each client, indicating the synchronized time.
- This synchronization process helps ensure that all nodes in the distributed system have closely aligned clocks, which is essential for coordinating actions and maintaining consistency across the system.

## Assignment No. 5

```
import java.util.Scanner;

class Tring {

    public static void main(String args[]) {

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the number of nodes: ");

        int n = sc.nextInt();

        // Decides the number of nodes forming the ring

        int token = 0;

        for (int i = 0; i < n; i++)

            System.out.print(" " + i);

        System.out.println(" " + 0);

        try {

            while (true) {

                System.out.print("Enter sender: ");

                int s = sc.nextInt();

                System.out.print("Enter receiver: ");

                int r = sc.nextInt();

                System.out.print("Enter Data: ");

                String d = sc.next();

                System.out.print("Token passing:");

                //current token not equal to sender, increment i by 1 and j by j+1%n
                for (int i = token, j = token; (i % n) != s; i++, j = (j + 1) % n) {

                    System.out.print(" " + j + "->");

                }

                System.out.println(" " + s);

                System.out.println("Sender " + s + " sending data: " + d);

                // start forwarding from node after sender until it becomes equal to receiver and increment by
                i+1%n
                for (int i = (s + 1) % n; i != r; i = (i + 1) % n) {

                    System.out.println("Data " + d + " forwarded by " + i);

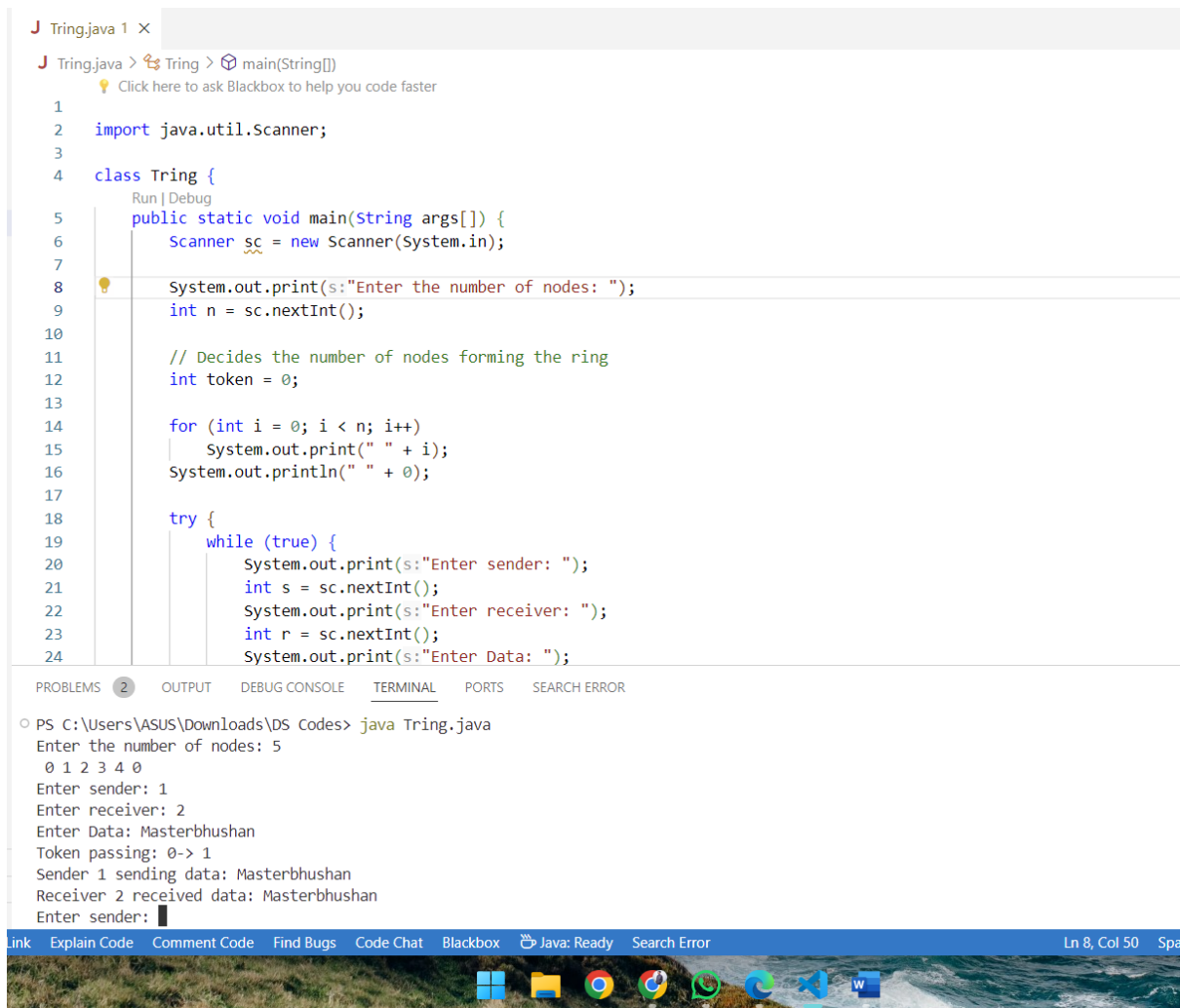
                }

                System.out.println("Receiver " + r + " received data: " + d);

                token = s;

            } } catch (Exception e) {

                System.out.println("Error occurred: " + e.getMessage()); } }
```



```
Tring.java 1 X
Tring.java > Tring > main(String[])
Click here to ask Blackbox to help you code faster

1
2 import java.util.Scanner;
3
4 class Tring {
5     public static void main(String args[]) {
6         Scanner sc = new Scanner(System.in);
7
8         System.out.print(s:"Enter the number of nodes: ");
9         int n = sc.nextInt();
10
11         // Decides the number of nodes forming the ring
12         int token = 0;
13
14         for (int i = 0; i < n; i++)
15             System.out.print(" " + i);
16         System.out.println(" " + 0);
17
18         try {
19             while (true) {
20                 System.out.print(s:"Enter sender: ");
21                 int s = sc.nextInt();
22                 System.out.print(s:"Enter receiver: ");
23                 int r = sc.nextInt();
24                 System.out.print(s:"Enter Data: ");
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS SEARCH ERROR

```
PS C:\Users\ASUS\Downloads\DS Codes> java Tring.java
Enter the number of nodes: 5
0 1 2 3 4 0
Enter sender: 1
Enter receiver: 2
Enter Data: Masterbhushan
Token passing: 0-> 1
Sender 1 sending data: Masterbhushan
Receiver 2 received data: Masterbhushan
Enter sender: 
```

Link Explain Code Comment Code Find Bugs Code Chat Blackbox Java: Ready Search Error Ln 8, Col 50 Spa

- The code implements a token passing algorithm for communication among nodes arranged in a ring network topology.
- It begins by prompting the user to input the number of nodes in the ring.
- Then it prints the sequence of nodes forming the ring.
- Inside a try-catch block, it enters an infinite loop for continuous interaction.
- Within each iteration, it prompts the user to input the sender node, receiver node, and data to be transmitted.
- It then simulates the token passing process, starting from the sender node and forwarding the data until it reaches the receiver node.
- The token passing is visualized by printing the sequence of nodes through which the token/data is being forwarded.
- After the data transmission, it updates the token to the sender node for the next iteration.
- The program catches and handles any exceptions that may occur during its execution.
- The code can be used for educational purposes to understand the basics of token passing algorithms and network communication in a ring topology. It can also serve as a simple simulation tool for demonstrating how data transmission works in such a network.

## Assignment No.6

### **BullyAlgorithm**

```
import java.util.Scanner;

public class BullyAlgorithm {

    static boolean[] state = new boolean[5];

    static int coordinator = 5;

    public static void up(int up) {
        if (state[up - 1]) {
            System.out.println("Process " + up + " is already up.");
        } else {
            state[up - 1] = true;

            System.out.println("Process " + up + " is up.");

            // If the newly up process has a higher ID than the current coordinator,
            // initiate an election

            if (up > coordinator) {
                System.out.println("Process " + up + " initiates an election.");
                election(up);
            } } }

    public static void down(int down) {
        if (!state[down - 1]) {
            System.out.println("Process " + down + " is already down.");
        } else {
            state[down - 1] = false;

            System.out.println("Process " + down + " is down.");

            if (down == coordinator) {
                System.out.println("Coordinator (Process " + down + ") is down.");
                election(down);
            } } }

    public static void mess(int mess) {
        if (!state[mess - 1]) {
            System.out.println("Process " + mess + " is down.");
        } else {
            if (mess == coordinator) {
                System.out.println("Coordinator (Process " + coordinator + ") received the message: OK");
            } else {
```



```

        System.out.println("Process " + mess + " sends a message.");
    } } }

public static void election(int initiator) {
    System.out.println("Election initiated by Process " + initiator);
    for (int i = initiator + 1; i <= 5; i++) {
        if (state[i - 1]) {
            System.out.println("Election message sent from Process " + initiator + " to Process " + i);
        }
    }
    // If no higher priority process responds, declare the initiator as the new coordinator
    coordinator = initiator;
    System.out.println("Process " + coordinator + " becomes the new coordinator.");
    System.out.println("Coordinator message sent from Process " + coordinator + " to all.");
}

public static void main(String[] args) {
    int choice;
    Scanner sc = new Scanner(System.in);
    for (int i = 0; i < 5; ++i) {
        state[i] = true; }
    System.out.println("5 active processes are:");
    System.out.println("Process up = p1 p2 p3 p4 p5");
    System.out.println("Process 5 is coordinator");
    do {
        System.out.println(".....");
        System.out.println("1. Up a process.");
        System.out.println("2. Down a process.");
        System.out.println("3. Send a message.");
        System.out.println("4. Exit.");
        choice = sc.nextInt();
        switch (choice) {
            case 1: {
                System.out.println("Bring process up:");
                int up = sc.nextInt();
                if (up > 5) {
                    System.out.println("Invalid process number.");
                    break;
                }
            }
        }
    } while (choice != 4);
}

```

```

    }
    up(up);
    break;
}
case 2: {
    System.out.println("Bring down any process:");
    int down = sc.nextInt();
    if (down > 5) {
        System.out.println("Invalid process number.");
        break;
    }
    down(down);
    break;
}
case 3: {
    System.out.println("Which process will send message:");
    int mess = sc.nextInt();
    if (mess > 5) {
        System.out.println("Invalid process number.");
        break; }
    mess(mess);
    break; } }
} while (choice != 4);
}
}

```

The screenshot shows a Java IDE with the file `BullyAlgorithm.java` open. The code defines a `BullyAlgorithm` class with a `main` method that implements the Bully Algorithm. The terminal output shows the program's execution, including the initial state of processes, the election process, and the selection of a coordinator.

```
Go Run Terminal Help
J Tring.java 1 J Berkeley.java 1 J BullyAlgorithm.java 1 x

BullyAlgorithm.java > BullyAlgorithm
3 public class BullyAlgorithm {
61 public static void main(String[] args) {
88     case 2: {
89         System.out.println(x:"Bring down any process:");
90         int down = sc.nextInt();
91         if (down > 5) {
92             System.out.println(x:"Invalid process number.");
93             break;
94         }
    }

PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL PORTS SEARCH ERROR

PS C:\Users\ASUS\Downloads\DS Codes> java BullyAlgorithm.java
5 active processes are:
Process up = p1 p2 p3 p4 p5
Process 5 is coordinator
.....
1. Up a process.
2. Down a process.
3. Send a message.
4. Exit.
2
Bring down any process:
2
Process 2 is down.
.....
1. Up a process.
2. Down a process.
3. Send a message.
4. Exit.
3
Which process will send message:
3
Process 3 sends a message.
.....
1. Up a process.
2. Down a process.
3. Send a message.
4. Exit.
4
```

### **Bully Algorithm Explanation:**

- The provided Java code implements the Bully Algorithm, a leader election algorithm used in distributed systems to elect a coordinator among a group of processes.
- Key components of the code:
  - An array state representing the state (up/down) of each process.
  - An integer coordinator representing the current coordinator process.
  - Methods for bringing up (up), bringing down (down), and messaging (mess) processes, along with an election method (election).
- Functionality:
  - `up(int up)`: Brings a process up, updates its state, and initiates an election if it has a higher ID than the current coordinator.
  - `down(int down)`: Brings a process down, updates its state, and initiates an election if it was the coordinator.
  - `mess(int mess)`: Sends a message to a process, indicating if it's down or if the coordinator received the message.
  - `election(int initiator)`: Initiates an election by sending election messages to higher priority processes, and if no response is received, declares the initiator as the new coordinator.
- The main method provides a menu-driven interface for interacting with the processes, allowing users to bring processes up or down and send messages.
- The program continues to run until the user chooses to exit.
- Use cases:
  - Simulating fault tolerance in distributed systems: Processes can elect a new coordinator if the current one fails.
  - Coordinating tasks among multiple processes: The coordinator process can manage and delegate tasks among others.

## Ring Algorithm

```
import java.util.Scanner;

public class RingAlgorithm {

    public static void main(String[] args) {

        int i, j;

        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter the number of processes: ");

        int numberOfProcesses = scanner.nextInt();

        Process[] processes = new Process[numberOfProcesses];

        // Object initialization

        for (i = 0; i < processes.length; i++)

            processes[i] = new Process();

        // Getting input from users

        for (i = 0; i < numberOfProcesses; i++) {

            processes[i].index = i;

            System.out.println("Enter the ID of process " + (i + 1) + ": ");

            processes[i].id = scanner.nextInt();

            processes[i].state = "active";

            processes[i].hasSentMessage = false;

        }

        // Sorting the processes based on their IDs

        for (i = 0; i < numberOfProcesses - 1; i++) {

            for (j = 0; j < numberOfProcesses - i - 1; j++) {

                if (processes[j].id > processes[j + 1].id) {

                    Process temp = processes[j];

                    processes[j] = processes[j + 1];

                    processes[j + 1] = temp;

                }

            }

        }

        System.out.println("Processes in sorted order:");

        for (i = 0; i < numberOfProcesses; i++) {
```

```

        System.out.println "[" + processes[i].index + " ] " + processes[i].id);
    }

    // Initiating coordinator selection

    int initiatorIndex;

    while (true) {

        System.out.println("\n1. Initiate Election\n2. Quit");

        int choice = scanner.nextInt();

        switch (choice) {

            case 1:

                System.out.println("Enter the index of the process initiating the election: ");

                initiatorIndex = scanner.nextInt();

                initiateElection(processes, initiatorIndex, numberOfProcesses);

                break;

            case 2:

                System.out.println("Program terminated.");

                return;

            default:

                System.out.println("Invalid response.");

        }
    }

    public static void initiateElection(Process[] processes, int initiatorIndex, int numberOfProcesses) {

        System.out.println("Election initiated by Process " + processes[initiatorIndex].id);

        processes[initiatorIndex].hasSentMessage = true;

        int templIndex = initiatorIndex;

        int nextIndex = (initiatorIndex + 1) % numberOfProcesses;

        while (nextIndex != initiatorIndex) {

            if (processes[nextIndex].state.equals("active") && !processes[nextIndex].hasSentMessage) {

                System.out.println("Process " + processes[initiatorIndex].id + " sends message to Process " +
processes[nextIndex].id);

                processes[nextIndex].hasSentMessage = true;

                templIndex = nextIndex;

            }

            nextIndex = (nextIndex + 1) % numberOfProcesses;

        }

        System.out.println("Process " + processes[templIndex].id + " sends message to Process " +

```

```

processes[initiatorIndex].id);

    System.out.println("Process " + processes[initiatorIndex].id + " becomes the coordinator.");

    processes[initiatorIndex].state = "inactive";

}}

class Process {

public int index;

public int id;

// Index of the process

// ID of the process

public boolean hasSentMessage; // Flag to indicate whether the process has sent a message

public String state;

}

```

```

Go Run Terminal Help
PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL PORTS SEARCH ERROR

○ PS C:\Users\ASUS\Downloads\DS Codes> java RingAlgorithm.java
Enter the number of processes:
5
Enter the ID of process 1:
1
Enter the ID of process 2:
2
Enter the ID of process 3:
3
Enter the ID of process 4:
4
Enter the ID of process 5:
5
Processes in sorted order:
[0] 1
[1] 2
[2] 3
[3] 4
[4] 5

1. Initiate Election
2. Quit
1
Enter the index of the process initiating the election:
3
Election initiated by Process 4
Process 4 sends message to Process 5
Process 4 sends message to Process 1
Process 4 sends message to Process 2
Process 4 sends message to Process 3
Process 3 sends message to Process 4
Process 4 becomes the coordinator.

1. Initiate Election
2. Quit
█

```

### **Ring Algorithm Explanation: -**

- The code implements the Ring Algorithm, a distributed algorithm used for coordinator selection in a network of processes.
- It begins by prompting the user to input the number of processes and the IDs for each process.
- Processes are represented by the Process class, which includes attributes like index, ID, whether it has sent a message, and its state (active or inactive).
- The processes are then sorted based on their IDs to establish a logical ring structure.
- The user is given the option to initiate an election for selecting a coordinator or to quit the program.
- When an election is initiated, the process designated as the initiator sends messages around the ring until it receives its own message back, indicating that it has traversed the entire ring.
- Along the way, each active process that has not yet sent a message receives and forwards the election message.
- Finally, the initiator process becomes the coordinator after completing the ring traversal, and its state is updated too inactive.
- The process continues until the user chooses to quit the program.
- The algorithm ensures that each process participates in the election and only forwards the message if it has not done so already, preventing message loops and ensuring eventual termination.
- This algorithm is commonly used in distributed systems for tasks like leader election and resource allocation.

## **Assignment No.7**

### **Calculator.java**

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/WebServices/WebService.java to edit this
 * template
 */

package com.unique;

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

@WebService(serviceName = "Calculator")

public class Calculator {

    @WebMethod(operationName = "getNumber")

    public int getNumber(@WebParam(name = "num1") int num1, @WebParam(name = "num2") int num2)
    {

        int sum=num1+num2;

        return sum; }}
```

### **Calculator.java (Client side):**

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/WebServices/WebService.java to edit this
 * template
 */

package com.unique;

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

@WebService(serviceName = "Calculator")

public class Calculator {

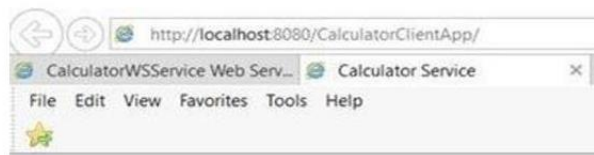
    @WebMethod(operationName = "getNumber")

    public int getNumber(@WebParam(name = "num1") int num1, @WebParam(name = "num2") int num2)
    {

        int sum=num1+num2;

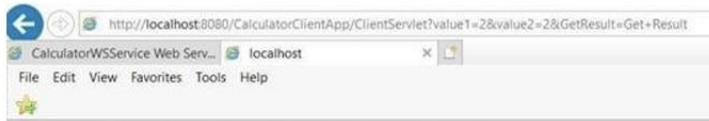
        return sum; }}}
```





## Calculator Service

2 + 2 =



Servlet ClientServlet at /CalculatorClientApp

Result: 2 + 2 = 4

### Explanation: -

- Two Java classes named Calculator are provided, one for the server-side and one for the client-side.
- Both classes are packaged under com.unique.
- They utilize Java's JAX-WS (Java API for XML Web Services) to define a simple web service for performing arithmetic operations.
- The server-side Calculator class defines a web service endpoint named Calculator with an operation named getNumber.
- This operation takes two integer parameters num1 and num2, adds them together, and returns the result.
- The client-side Calculator class mirrors the server-side class structure and functionality.
- It also defines a web service endpoint named Calculator with an operation named getNumber.
- This operation takes two integer parameters num1 and num2, and it's intended to invoke the corresponding server-side operation.
- Both classes use annotations like @WebService, @WebMethod, and @WebParam to define the web service and its operations.
- The @WebService annotation is used at the class level to mark the class as a web service endpoint.
- The @WebMethod annotation is used at the method level to specify the web service operation.
- The @WebParam annotation is used at the method parameter level to specify the names of the parameters in the web service operation.
- Uses:
  - This code can be used to create a simple web service for performing addition operations over the network.
  - The server-side class can be deployed on a web server, allowing clients to invoke its getNumber operation remotely.
  - The client-side class can be used by other applications to consume the web service provided by the server-side class, allowing them to perform addition operations without needing to implement the logic themselves.