Name: Omkar Kadam
Roll No. : 3025
BE IT
Subject: Distributed System

Assignment 1

Title: Remote Method Invocation

Aim: To implement multi-threaded client/server Process communication using RMI.

Objectives:
1. To understand the basics of multi-threaded client/server Process communication.
2. To develop interprocess communication application using RMI.

Tools / Environment:
Java Programming Environment, jdk 1.8, rmiregistry

Steps:
Steps to run this program
(Make sure you are in the folder containing the java files)

Step 1 - On Terminal 1
```
javac *.java
```

Step 2 - On Terminal 2
```
java Server
```

Step 3 - On Terminal 3
```
java Client
```

1. Define the remote interface.
2. Implement the remote interface.
3. Create the server class, register, and bind the remote object.
4. Create the client class, locate the remote object, and invoke remote methods.
5. Compile all classes.
6. Start the RMI registry (if not already running).
7. Start the server.
8. Start the client.

Code:
Circle.java:-

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Circle extends Remote {

	public double getArea(int radius) throws RemoteException;

	public double getPerimeter(int radius) throws RemoteException;
}
```

CircleImpl.java:-

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class CircleImpl extends UnicastRemoteObject implements Circle {

	private double PI;

	public CircleImpl() throws RemoteException {
		super();
		PI = 22.0 / 7.0;
	}

	@Override
	public double getArea(int radius) {
		return PI * radius * radius;
	}

	@Override
	public double getPerimeter(int radius) {
		return 2 * PI * radius;
	}
}
```

Client.java:-

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.Scanner;
```

```
public class Client {

	public static void main(String[] args) {
		try {
			// Locate the registry
			Registry registry = LocateRegistry.getRegistry("127.0.0.1", 4000);

			// Get the references of exported objects from the registry
			Circle circle = (Circle) registry.lookup("rmi://localhost:4000/circle");

			int radius;
			Scanner in=new Scanner(System.in);
			System.out.print("Enter the radius of the circle: ");
			radius=in.nextInt();
			System.out.println("\nThe Area of the circle is "+ circle.getArea(radius));
			System.out.println("The Perimeter of the circle is "+
circle.getPerimeter(radius));
		} catch (Exception e)
		{
			System.out.println("Client Error: " + e);
		}
	}
}
```

Server.java:-

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class Server
{

    public static void main(String[] args) {
        try {
            System.out.println("Server started...");

            // Set hostname for the server using javaProperty
            System.setProperty("java.rmi.server.hostname", "127.0.0.1");

            // Register the exported class in RMI registry with some name,
            // Client will use that name to get the reference of those exported object
            // Get the registry to register the object.
            Registry registry = LocateRegistry.createRegistry(4000);

            // create product objects
            Circle stub = new CircleImpl();
            registry.rebind("rmi://localhost:4000/circle", stub);
        } catch (Exception e) {
            System.out.println("Server error: " + e);
        }
    }
}
```

Output:-

```
Enter the radius of the circle: 18

The Area of the circle is 1018.2857142857142
The Perimeter of the circle is 113.14285714285714
```

Conclusion:
Remote Method Invocation (RMI) allows you to build Java applications that are distributed among several machines. Remote Method Invocation (RMI) allows a Java object that executes on one machine to invoke a method of a Java object that executes on another machine.
This is an important feature, because it allows you to build distributed applications.

Name: Omkar Kadam
Roll No. : 3025
BE IT
Subject: Distributed System

Assignment 2

Title: Common Object Request Broker Architecture (CORBA)

Aim: To develop any distributed application with CORBA program using JAVA IDL.

Objective:
1. To understand the basics steps for implementation of CORBA.
2. To develop any distributed application using CORBA to demonstrate object brokering.

Tools / Environment:
Java Programming Environment, JDK 1.8

Steps:-

1. Define the IDL interface.
2. Compile the IDL file to generate Java stubs and skeletons.
3. Implement the server class.
4. Create the server application and register the server object with the naming service.
5. Implement the client class.
6. Compile all classes.
7. Start the ORB.
8. Start the server.
9. Run the client.

Code:

CalculatorImpl.java
```
import org.omg.CORBA.ORB;
import CalculatorApp.CalculatorPOA;
public class CalculatorImpl extends CalculatorPOA {
   private ORB orb;
   public CalculatorImpl(ORB orb) {
      super();
```

```
        this.orb = orb;
    }
    @Override
    public double add(double x, double y) {
        return x + y;
    }
    @Override
    public double subtract(double x, double y) {
        return x - y;
    }
    @Override
    public double multiply(double x, double y) {
        return x * y;
    }
    @Override
    public double divide(double x, double y) {
        if (y == 0) {
            return Double.MAX_VALUE;
        }
        return x / y;
    }
    @Override
    public void shutdown() {
        orb.shutdown(false);
    }
}
//Calculator.idl
module CalculatorApp
{
    interface Calculator
    {
        double add(in double x, in double y);
        double subtract(in double x, in double y);
        double multiply(in double x, in double y);
        double divide(in double x, in double y);
        oneway void shutdown();
    };
};
```

Client.java

```
import java.util.Scanner;
import org.omg.CORBA.ORB;
import org.omg.CosNaming.NamingContextExt;
import org.omg.CosNaming.NamingContextExtHelper;
import CalculatorApp.Calculator;
import CalculatorApp.CalculatorHelper;
public class Client {
	public static void main(String[] args) {
		try {
			ORB orb = ORB.init(args, null);
			org.omg.CORBA.Object objRef =
orb.resolve_initial_references("NameService");
			NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
			Calculator calculator =
CalculatorHelper.narrow(ncRef.resolve_str("ABC"));
			Scanner sc = new Scanner(System.in);
			System.out.println("Welcome to addition system:");
			while (true) {
				System.out.println("1. Add");
				System.out.println("2. Subtract");
				System.out.println("3. Multiply");
				System.out.println("4. Divide");
				System.out.println("Press any key to exit");
				System.out.print("Enter your choice: ");
				int choice = sc.nextInt();
				if (choice == 5) {
					break;
				}
				System.out.print("Enter first no: ");
				double x = sc.nextDouble();
				System.out.print("Enter second no: ");
				double y = sc.nextDouble();
				if (choice == 1) {
					System.out.println("Ans = " + x + " + " + y + " = " +
calculator.add(x, y));
				} else if (choice == 2) {
					System.out.println("Ans = " + x + " - " + y + " = " +
calculator.subtract(x, y));
				} else if (choice == 3) {
					System.out.println("Ans = " + x + " * " + y + " = " +
calculator.multiply(x, y));
				} else if (choice == 4) {
					System.out.println("Ans = " + x + " / " + y + " = " +
calculator.divide(x, y));
				}
				System.out.println("--------------------------------------------------\n");
			}
			sc.close();
		} catch (Exception e) {
			System.out.println("Client Error: " + e);
			e.printStackTrace(System.out);
		}
	}
}
```

Server.java

```
import org.omg.CORBA.ORB;
import org.omg.CosNaming.NameComponent;
import org.omg.CosNaming.NamingContextExt;
import org.omg.CosNaming.NamingContextExtHelper;
import org.omg.PortableServer.POA;
import org.omg.PortableServer.POAHelper;
import CalculatorApp.Calculator;
import CalculatorApp.CalculatorHelper;
public class Server {
	public static void main(String[] args) {
		try {
			// create and initialize the orb
			ORB orb = ORB.init(args, null);
			// get reference to rootPOA
			POA rootPOA =
POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
			// activate the POAManager
			rootPOA.the_POAManager().activate();
			// create servant and register it with the ORB
			CalculatorImpl calculator = new CalculatorImpl(orb);
			// get object reference from the servant
			org.omg.CORBA.Object ref = rootPOA.servant_to_reference(calculator);
			Calculator href = CalculatorHelper.narrow(ref);
			org.omg.CORBA.Object objRef =
orb.resolve_initial_references("NameService");
			NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
			NameComponent[] path = ncRef.to_name("ABC");
			ncRef.rebind(path, href);
			System.out.println("Server ready and waiting...");
			while (true) {
				orb.run();
			}
		} catch (Exception e) {
			System.out.println("Server Error: " + e);
			e.printStackTrace(System.out);
```

```
                    }
                    System.out.println("Server Exiting...");
          }
}
```

OUTPUT:

```
Welcome to Calculator System
1. Add
2. Subtract
3. Multiply
4. Divide
Press any key to exit
Enter your choice: 1
Enter first no: 22
Enter second no: 8
Ans = 22.0 + 2.0 = 30.0
```

Name: Omkar Kadam
Roll No. : 3025
BE IT
Subject: Distributed System

Assignment 3

Title: Message Passing Interface (MPI)

1.Aim: Develop a distributed system, to find sum of N elements in an array by distributing N/n elements to n number of processors MPI or OpenMP. Demonstrate by displaying the intermediate sums calculated at different processors

2.Objective:
1. To introduce you to the fundamentals of MPI.
2. To understand widely used standard for writing message passing programs.

3.Tools / Environment:
Java Programming Environment, JDK1.8 or higher, MPI Library (mpi.jar), MPJ Express (mpj.jar)

4.Steps:

1. **Import jar files:**

Jar file is present in: mpj-v0_44 -> lib -> mpj.jar
The file "mpjrun.sh" which is required to run is present in: mpj-v0_44 -> bin -> mpjrun.sh

   a. Initialize MPI environment using MPI_Init().

2. **Get Processor Information**:
   a. Get the total number of processors and the rank of the current processor using MPI_Comm_size() and MPI_Comm_rank().
3. **Allocate Array and Distribute Data**:
   a. Create the array of N elements on the root processor.
   b. Distribute the data among processors using MPI_Scatter() or MPI_Scatterv().
4. **Calculate Partial Sum**:
   a. Each processor calculates the sum of its assigned chunk of elements.
5. **Gather Partial Sums**:
   a. Gather the partial sums calculated by each processor onto the root processor using MPI_Gather() or MPI_Gatherv().
6. **Combine Partial Sums**:

    a. The root processor combines the partial sums to get the final sum.

7. **Finalize MPI**:
    a. Finalize MPI environment using MPI_Finalize().

2. **Code:**

```
import mpi.*;

public class DistributedSum {
   public static void main(String[] args) throws MPIException {
      MPI.Init(args);
      int rank = MPI.COMM_WORLD.Rank(); // get the rank of the current process
      int size = MPI.COMM_WORLD.Size(); // get the total number of processes

      int[] array = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}; // sample input array
      int n = array.length; // total number of elements
      int local_n = n / size; // number of elements to be processed by each process
      int remainder = n % size; // number of remaining elements

      int[] local_array = new int[local_n + (rank < remainder ? 1 : 0)]; // local array to hold the
elements for each process
      int offset = rank * local_n + Math.min(rank, remainder); // compute the offset for the current
process
      for (int i = 0; i < local_array.length; i++) {
         local_array[i] = array[offset + i];
      }

      int local_sum = 0; // compute the sum of the local elements
      for (int i = 0; i < local_array.length; i++) {
         local_sum += local_array[i];
      }

      int[] global_sums = new int[size]; // array to hold the global sum from each process
      MPI.COMM_WORLD.Allgather(new int[]{local_sum}, 0, 1, MPI.INT, global_sums, 0, 1,
MPI.INT); // gather the local sums to all processes

      if (rank == 0) { // print the intermediate and final sums
         System.out.println("Number of Processes Entered: "+ size);
         System.out.println("\nIntermediate Sums:");
         int sum = 0;
         for (int i = 0; i < size; i++) {
            sum += global_sums[i];
```

```
                System.out.println("Process " + i + ": " + global_sums[i]);
            }
            System.out.println("\nTotal Sum: " + sum);
        }

        MPI.Finalize();
    }
}
```

3.Enter this commands in terminal:

```
-javac -cp "C:\Users\dell\Downloads\mpj-v0_44\mpj-v0_44\lib\mpj.jar" DistributedSum.java
-C:\Users\dell\Downloads\mpj-v0_44\mpj-v0_44\bin\mpjrun.bat" -np 6 DistributedSum
```

4.Output:

```
Number of Processes Entered: 6
Intermediate Sums:
Process 0: 3
Process 1: 7
Process 2: 11
Process 3: 15
Process 4: 9
Process 5: 10
Total Sum: 55
```

5.Conclusion:

Thus Student develop a distributed system application, to find sum of N elements in an array by distributing N/n elements to n number of processors MPI or OpenMP. Demonstrate by displaying the intermediate sums calculated at different processors.

6.Outcome:

1. Students learn fundamentals of parallel programming and MPI.
2. Students design parallel algorithms and write parallel programs using the MPI library.

Name: Omkar Kadam
Roll No. : 3025
BE IT
Subject: Distributed System

Assignment 4

Title: Clock Synchronization

Aim: To implement Berkeley algorithm for clock synchronization

Objective:
1. To understand the basics of physical and Logical clock in DS.
2. To develop an n-node distributed system that implements Berkeley's time synchronization algorithm.

Steps:-

1. Select a Coordinator: Choose one node as the coordinator.
2. Coordinator Sends Requests: The coordinator periodically sends synchronization requests to all other nodes.
3. Nodes Respond: Each node receiving the request responds with its current time.
4. Calculate Time Differences: The coordinator calculates the time differences between its clock and the clocks of other nodes.
5. Calculate Average: Calculate the average time difference among all nodes.
6. Adjust Clocks: The coordinator sends adjustment messages to all nodes based on the calculated average.
7. Nodes Adjust Clocks: All nodes adjust their clocks based on the received adjustment messages.

Code:-

```
import java.text.*;
import java.util.*;

public class BerkelyAlgorithm {

    public static void main(String[] args) throws ParseException {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter number of clients in your network: ");
        int clientCount = sc.nextInt();
        sc.nextLine();

        String[] timeString = new String[1 + clientCount];  // 1 server + clientCount clients

        for (int i = 0; i < timeString.length; i++) {
            if (i == 0) {
                System.out.print("Enter time displayed in Server (HH:mm): ");
            } else {
                System.out.print("Enter time displayed in Client " + i + " (HH:mm): ");
            }

            String time = sc.nextLine();
            timeString[i] = appendCurrentDateToTime(time);
        }

        System.out.println("\nBefore Synchronization");
        displayTime(timeString, "");

        berkeleyAlgorithm(timeString);

        System.out.println("\nAfter Synchronization");
        displayTime(timeString, "Synchronized ");

        sc.close();
    }

    public static void berkeleyAlgorithm(String[] timeString) throws ParseException {
        int n = timeString.length;

        SimpleDateFormat simpleDateFormat = new SimpleDateFormat("HH:mm | yyyy-MM-dd");

        // Converting time to milliseconds
        long[] timeInMilliseconds = new long[n];
        for (int i = 0; i < n; i++) {
            timeInMilliseconds[i] = simpleDateFormat.parse(timeString[i]).getTime();
        }

        // Calculating time difference w.r.t. server
        long serverTime = timeInMilliseconds[0];
        long[] differenceInTimeWithServer = new long[n];
        for (int i = 0; i < n; i++) {
            differenceInTimeWithServer[i] = timeInMilliseconds[i] - serverTime;
        }

        // Calculating Fault tolerant average
        long avg = 0;
        for (int i = 0; i < n; i++) {
            avg += differenceInTimeWithServer[i];
        }
        avg /= n;
        System.out.println("Fault tolerant average = " + avg / (1000 * 60));    // Displaying
fault-tolerant average in minutes

        // Adjusting the time in Server and Clients
        for (int i = 0; i < n; i++) {
            long offset = avg - differenceInTimeWithServer[i];
            timeInMilliseconds[i] += offset;
            if (i == 0) {
                continue;
            }
            System.out.println("Clock " + i + " adjustment = " + offset / (1000 * 60)); // Displaying
adjustment value in minutes
        }

        // Converting milliseconds to actual time
        for (int i = 0; i < n; i++) {
            timeString[i] = simpleDateFormat.format(new Date(timeInMilliseconds[i]));
        }
    }

    private static void displayTime(String[] time, String prefix) {
        System.out.println(prefix + "Server Clock:\t" + time[0]);
        for (int i = 1; i < time.length; i++) {
            System.out.println(prefix + "Client " + i + " Clock:\t" + time[i]);
        }
        System.out.println();
    }

    private static String appendCurrentDateToTime(String time) {
        Calendar calendar = new GregorianCalendar();
        calendar.setTime(new Date());
        int year = calendar.get(Calendar.YEAR);
        int month = calendar.get(Calendar.MONTH) + 1;
        int day = calendar.get(Calendar.DAY_OF_MONTH);
        return time + " | " + year + "-" + month + "-" + day;
    }
}
```

Commands:

```
~/Documents/assignments$ javac BerkeleyAlgorithm.java
~/Documents/assignments$ java BerkeleyAlgorithm
```

OUTPUT:

```
Enter number of clients in your network: 2
Enter time displayed in Server (HH:mm): 03:00
Enter time displayed in Client 1 (HH:mm): 03:25
Enter time displayed in Client 2 (HH:mm): 02:50

Before Synchronization
Server Clock:   03:00 | 2023-5-10
Client 1 Clock: 03:25 | 2023-5-10
Client 2 Clock: 02:50 | 2023-5-10

Fault tolerant average = 5
Clock 1 adjustment = -20
Clock 2 adjustment = 15

After Synchronization
Synchronized Server Clock:      03:05 | 2023-05-10
Synchronized Client 1 Clock:    03:05 | 2023-05-10
Synchronized Client 2 Clock:    03:05 | 2023-05-10
```

Conclusion:

The Berkeley algorithm is a simple yet effective solution for clock synchronization in distributed systems. Its decentralized approach allows for resilience against failures, and it remains a relevant and widely used tool in ensuring accurate timekeeping between machines.

Name: Omkar Kadam
Roll No. : 3025
BE IT
Subject: Distributed System

Assignment 5

Title: Mutual Exclusion

Aim: Implement token ring based mutual exclusion algorithm.

Objective:
1. To understand the concept of starvation, Mutual Exclusion in DS.
2. To Achieve Mutual Exclusion In Distributed System based on Token Exchange.

Steps:-

1. **Token Initialization**: Start with a token that circulates among the nodes in the system.

2. **Request for Critical Section**: When a node wants to enter the critical section, it waits until it possesses the token.

3. **Use Critical Section**: Once a node possesses the token, it enters the critical section and executes its task.

4. **Pass Token**: After completing the critical section, the node passes the token to the next node in the ring.

5. **Repeat**: Nodes continue to pass the token around the ring, allowing each node to enter the critical section one at a time.

By following these steps, mutual exclusion can be achieved in a distributed system using a token ring based algorithm, ensuring that only one node accesses the critical section at a time.

Code:-
```
import java.util.*;

public class TokenRing {

    private int n; // Number of processes
    private int[] state;
private boolean token;

public TokenRing(int n) {
    this.n = n;
    state = new int[n];
}

public void run() {
    // Initialize the ring
    for (int i = 0; i < n; i++) {
        state[i] = 0; // not in critical section
    }

    // Randomly select the process that will hold the token initially
    int curr = new Random().nextInt(n);
    token = true;

    while (true) {
        if (token) {
            if (state[curr == 0) { // not in critical section
                enterCritical(curr);
                state[curr] = 1; // in critical section
            } else { // in critical section
                exitCritical(curr);
                state[curr] = 0; // not in critical section

                curr = (curr + 1) % n; // pass the token
                token = true;
            }
        }
    }
}

private void enterCritical(int id) {
    // Enter critical section
    System.out.println("Process " + id + " enters critical section");
}

private void exitCritical(int id) {
    // Exit critical section
    System.out.println("Process " + id + " exits critical section");
    token = false;
}
```

```
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter Number of Processes To Be Created: ");
        int n = sc.nextInt();
        sc.close();

        TokenRing tr = new TokenRing(n); // Instantiate a ring of n processes
        tr.run(); // Run the token ring algorithm
    }
}
```

Commands:

```
~/Documents/assignments$ javac TokenRing.java
~/Documents/assignments$ java TokenRing*/
```

Output:

```
Enter Number of Processes To Be Created: 8
Process 5 enters critical section
Process 5 exits critical section
Process 6 enters critical section
Process 6 exits critical section
Process 7 enters critical section
Process 7 exits critical section
Process 0 enters critical section
Process 0 exits critical section
Process 1 enters critical section
Process 1 exits critical section
Process 2 enters critical section
Process 2 exits critical section
Process 3 enters critical section
Process 3 exits critical section
Process 4 enters critical section
Process 4 exits critical section
Process 5 enters critical section
Process 5 exits critical section
Process 6 enters critical section
Process 6 exits critical section
Process 7 enters critical section
Process 7 exits critical section
Process 0 enters critical section
Process 0 exits critical section
Process 1 enters critical section
Process 1 exits critical section
Process 2 enters critical section
Process 2 exits critical section
.
.
.
.
(program runs infinitely to exit press Ctrl + C)
```

Conclusion:
The token ring-based mutual exclusion algorithm is a well-known solution for coordinating access to shared resources in distributed systems. Its simple and efficient design ensures that only one process can access a shared resource at a time, thus preventing conflicts and ensuring
consistency. While the algorithm can suffer from potential delays and network congestion, it remains a widely used and effective solution for achieving mutual exclusion in distributed Systems.

Name: Omkar Kadam
Roll No. : 3025
BE IT
Subject: Distributed System

Assignment 6

Title: Election Algorithms

Aim: To Implement Bully and Ring algorithm for leader election.

Objective:
1. To enable distributed systems to select a leader in a decentralized manner, without requiring a centralized control mechanism.
2. To ensure that the leader selection process is reliable and efficient, even in the presence of failures, network delays, and other forms of uncertainty.
3. To provide a fair and deterministic method for selecting the leader, such that all nodes in the system have an equal chance of being chosen.

Tools / Environment:
Java Programming Environment, JDK 1.8, Eclipse Neon(EE).

Steps:-

Bully Algorithm:

1. **Node Failure Detection**:
   - Nodes continuously monitor the state of other nodes to detect failures.

2. **Election Initiation**:
   - If a node detects a leader failure or timeouts occur, it initiates an election.

3. **Broadcast Election Message**:
   - The initiating node sends an election message to all nodes with higher IDs.

4. **Candidacy Announcement**:
   - Nodes with higher IDs respond, announcing their candidacy.

5. **Vote Collection**:

- The initiating node collects responses from higher-ID nodes.

6. **Election Resolution**:
- If no higher-ID nodes respond, the initiating node becomes the leader and broadcasts a victory message.

7. **Leader Announcement**:
- All nodes acknowledge the new leader.

8. **Leader Heartbeat**:
- The new leader periodically sends heartbeat messages to confirm its status.

Ring Algorithm:

1. **Node Initialization**:
- Each node is assigned an ID and knows the IDs of its neighboring nodes in the ring.

2. **Election Initiation**:
- If a node detects leader failure or timeouts occur, it initiates an election.

3. **Message Passing**:
- The initiating node sends an election message to its neighboring node with the highest ID.

4. **Message Forwarding**:
- Nodes forward the election message to their neighboring nodes.

5. **Election Propagation**:
- The election message travels along the ring until it reaches the node with the highest ID.

6. **Election Resolution**:
- The node with the highest ID becomes the leader and sends a victory message back along the ring.

7. **Leader Announcement**:
- All nodes acknowledge the new leader.

8. **Leader Heartbeat**:
- The new leader periodically sends heartbeat messages to confirm its status.

By following these 10 steps, you can implement Bully and Ring algorithms for leader election in a distributed system, ensuring decentralized leader selection with reliability and efficiency.

Code:-

```
import java.util.Scanner;

// create process class for creating a process having id and status
class Process {
	public int id;
	public String status;

	public Process(int id) {
		this.id = id;
		this.status = "active";
	}
}

public class BullyRing {

	Scanner sc;
	Process[] processes;
	int n;

	// initialize Scanner class object in constructor
	public BullyRing() {
		sc = new Scanner(System.in);
	}

	// create ring() method for initializing the ring
	public void ring() {

		// get input from the user for processes
		System.out.print("Enter total number of processes: ");
		n = sc.nextInt();

		// initialize processes array
		processes = new Process[n];
		for (int i = 0; i < n; i++) {
			processes[i] = new Process(i);
		}
	}

	// create election() method for electing process
	public void performElection() {

		// we use the sleep() method to stop the execution of the current thread
```

```
                try {
                        Thread.sleep(1000);
                } catch (InterruptedException e) {

                        e.printStackTrace();
                }

                // show failed process
                System.out.println("Process " + processes[getMaxValue()].id + " fails");

                // change status to Inactive of the failed process
                processes[getMaxValue()].status = "Inactive";

                int idOfInitiator = 0;
                boolean overStatus = true;

                while (overStatus) {

                        boolean higherProcesses = false;

                        // iterate all the processes
                        System.out.println();
                        for (int i = idOfInitiator + 1; i < n; i++) {
                                if (processes[i].status == "active") {
                                        System.out.println("Process " + idOfInitiator + " Passes
        Election(" + idOfInitiator          + ") message to Process " + i);
                                        higherProcesses = true;
                                }
                        }

                        // check for higher process
                        if (higherProcesses) {

                                System.out.println();
                                for (int i = idOfInitiator + 1; i < n; i++) {
                                        if (processes[i].status == "active") {
                                                System.out.println("Process " + i + " passes Ok(" + i
        + ") message to Process " + idOfInitiator);
                                        }
                                }
                                idOfInitiator++;
                        }

                        else {
```

```
                                        // get the last process from the processes that will become
coordinator
                                        int coord = processes[getMaxValue()].id;

                                        // show process that becomes the coordinator
                                        System.out.println("Finally Process " + coord + " Becomes
Coordinator");

                                        for (int i = coord - 1; i >= 0; i--) {
                                                if (processes[i].status == "active") {
                                                        System.out.println("Process " + coord + " passes
Coordinator(" + coord + ") message to Process " + i);
                                                }
                                        }
                                        System.out.println("\nEnd of Election");
                                        overStatus = false;
                                        break;
                                }
                        }
                }

                // create getMaxValue() method that returns index of max process
                public int getMaxValue() {
                        int mxId = -99;
                        int mxIdIndex = 0;
                        for (int i = 0; i < processes.length; i++) {
                                if (processes[i].status == "active" && processes[i].id > mxId) {
                                        mxId = processes[i].id;
                                        mxIdIndex = i;
                                }
                        }
                        return mxIdIndex;
                }

                public static void main(String[] args) {

                        BullyRing bully = new BullyRing();

                        bully.ring();
                        bully.performElection();
                }
}
```

Output:

```
Enter total number of processes: 6
Process 5 fails

Process 0 Passes Election(0) message to Process 1
Process 0 Passes Election(0) message to Process 2
Process 0 Passes Election(0) message to Process 3
Process 0 Passes Election(0) message to Process 4

Process 1 passes Ok(1) message to Process 0
Process 2 passes Ok(2) message to Process 0
Process 3 passes Ok(3) message to Process 0
Process 4 passes Ok(4) message to Process 0

Process 1 Passes Election(1) message to Process 2
Process 1 Passes Election(1) message to Process 3
Process 1 Passes Election(1) message to Process 4

Process 2 passes Ok(2) message to Process 1
Process 3 passes Ok(3) message to Process 1
Process 4 passes Ok(4) message to Process 1

Process 2 Passes Election(2) message to Process 3
Process 2 Passes Election(2) message to Process 4

Process 3 passes Ok(3) message to Process 2
Process 4 passes Ok(4) message to Process 2

Process 3 Passes Election(3) message to Process 4

Process 4 passes Ok(4) message to Process 3

Finally Process 4 Becomes Coordinator
Process 4 passes Coordinator(4) message to Process 3
Process 4 passes Coordinator(4) message to Process 2
Process 4 passes Coordinator(4) message to Process 1
Process 4 passes Coordinator(4) message to Process 0

End of Election
```

Name: Omkar Kadam
Roll No. : 3025
BE IT
Subject: Distributed System

Assignment 7

Title: Web Services

Aim: To create a simple web service and write any distributed application to consume the web Service.

Objective:
1. To understands the fundaments of web services, architecture and its types.
2. To create a simple web service.

Tools / Environment:
Java Programming Environment, JDK 8, Netbeans IDE with GlassFish Server

Steps:-

1. Create a new Java Web Application project in Netbeans.
2. Create the SimpleWebService.java class in your project and add the web service code.
3. Deploy your web service project.
4. Create a new Java Application project in Netbeans.
5. Add the JAX-RS client library (javax.ws.rs-api.jar) to your Java Application project.
6. Create the WebServiceConsumer.java class in your Java Application project and add the client code.
7. Run the WebServiceConsumer class.

Code:-

Simple Web Service Code (SimpleWebService.java):

```
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("data")
public class SimpleWebService {
  @GET
  @Produces(MediaType.TEXT_PLAIN)
  public String getDummyData() {
     return "Hello, this is dummy data from the web service!";
  }
}
```

Java Application to Consume the Web Service (WebServiceConsumer.java):

```
import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.core.MediaType;

public class WebServiceConsumer {

  public static void main(String[] args) {
     Client client = ClientBuilder.newClient();
     String response = client.target("http://localhost:8080/<your_project_name>/data")
                        .request(MediaType.TEXT_PLAIN)
                        .get(String.class);
     System.out.println("Response from web service: " + response);
     client.close();
  }
}
```

Output:-

```
Response from web service: Hello, this is dummy data from the web service!
```

Conclusion:
This assignment, described the Web services approach to the Service Oriented Architecture concept. Also, described the Java APIs for programming Web services and demonstrated examples of their use by providing detailed step-by-step examples of how to program Web services in Java.