Assignment1-

Circle.java=

```
//DS Assignment 1: Implement multi-threaded client/server Process communication using RMI.
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=

```
//DS Assignment 1: Implement multi-threaded client/server Process communication using RMI.
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;
    }
}
```

Server.java=

```
//DS Assignment 1: Implement multi-threaded client/server Process communication using RMI.
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);
        }
    }
}
```

Client.java=

```
//DS Assignment 1: Implement multi-threaded client/server Process communication using RMI.
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);
        }
    }
}
```

OUTPUT=
1st terminal= javac *.java
java Server
2nd terminal= java Client

Assignment-3
DistributedSum.java=

```
//DS Assignment 3: 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.
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();
    }
}
```

output=
1. Download & extract jar file in home directory from below link

https://sourceforge.net/projects/mpjexpress/

2. Open terminal in home directory & type below command

```
sudo gedit ~/.bashrc
```

3. Add below 2 lines in open bash rc

```
export MPJ_HOME="/home/pvg/mpj-v0_44"

export PATH=$MPJ_HOME/bin:$PATH
```

4. Compile and run assignment 3 using below commands

```
javac -cp "/home/pvg/mpj-v0_44/lib/mpj.jar" DistributedSum.java


mpjrun.sh -np 6 DistributedSum
```

Assignment-4

code=
BerkelyAlgorithm.java=

```
//DS Assignment 4: Implement Berkeley algorithm for clock synchronization.
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;
    }
}
```

output=
java BerkelyAlgorithm

Assignment-5
code=

TokenRing.java=

```
import java.util.*;
public class TokenRing{
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
```

```
System.out.print("Enter Number Of Nodes You Want In The Ring : ");
int n = sc.nextInt();

System.out.println("Ring Formed Is As Below: ");
for(int i=0; i<n; i++){
   System.out.print(i + " ");
}

System.out.println("0");


int choice = 0;
int token = 0;

do{
   System.out.print("Enter Sender : ");
   int sender = sc.nextInt();

   System.out.print("Enter Receiver : ");
   int receiver = sc.nextInt();

   System.out.print("Enter Data To Send : ");
   int data = sc.nextInt();


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

   for(int i=token; i<sender; i++){
      System.out.print(" " + i + "->");
   }

   if(receiver == sender + 1){
      System.out.println("Sender: " + sender + " Sending The Data: " + data);
      System.out.println("Receiver: " + receiver + " Received The Data: " + data);
   }
   else{
      System.out.println(" " + sender);
      System.out.println("Sender:" + sender + " Sending Data: " + data);

      for(int i=sender; i!=receiver; i = (i+1)%n){
         System.out.println("Data: " + data + " Forwarded By: " + i);
      }
```

```
            System.out.println("Receiver: " + receiver + " Received The Data: " + data);
        }


        token = sender;

        System.out.print("Do You Want To Send Data Again? If YES Enter 1, If NO Enter 0: ");
        choice = sc.nextInt();

    }while(choice == 1);

  }
}
```

OUTput=
java TokenRing

Assignment 6
code=

BullyRing.java=

```
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=
java BullyRing
```
