Assignment 3

Name : Omkar Deshpande

Roll No : 43212

Batch - Q10

Problem statement -To develop Microservices framework based distributed application.

CODE -

Publisher.java

```
import java.rmi.registry.Registry;
import java.util.HashMap;
import java.util.List;
import java.util.Scanner;
import java.util.concurrent.locks.ReentrantLock;
import java.rmi.registry.LocateRegistry;
import java.io.File;
import java.io.FileNotFoundException;
import java.lang.management.ManagementFactory;
import java.rmi.AccessException;
import java.rmi.NotBoundException;
import java.rmi.Remote;
import java.rmi.RemoteException;

/*
 * Publisher class is a simple class because it will not be binded to
 * any RMI id. Thus, we can make the entire class static.
 */
public class Publisher {
    static String UUID = "";
    static String logFile = "";
    static Registry registry;

    /*
     *  reqCounter  -   Request Counter, to generate unique request Ids.
     *  counterLock -   To lock the reqCounter.
     */
```

```
    static Integer reqCounter = 0;
    static ReentrantLock counterLock = new ReentrantLock(true);

    /*
     *  publish(topic, dt, ReqID) - Publish dt to the subscribers of
topic.
     */

    private static void publish(String topic, Data dt, String ReqID)
            throws AccessException, RemoteException, NotBoundException {

        System.out.println("Publishing @"+topic+" Data: "+
dt.getData()+" with Request ID: "+ ReqID);
        registry = LocateRegistry.getRegistry();
        ServerInterface server = (ServerInterface)
registry.lookup("master");
        // System.out.println(server);
        server.sendToSubscribers(topic, dt, ReqID);
    }

    // private static void outputToLog(String log) {

    // }

    /*
     *  executeCommand(String line) -   Extract and execute the command
from line.
     */
    static void executeCommand(String line) {
        String[] splitStrings = line.split(" ");
        if(splitStrings.length != 2)
        {
            System.out.println("Invalid Command: "+line);
            return;
        }
        Data dt = new Data();
        dt.setStringData(splitStrings[1]);
        String reqId = UUID;
        counterLock.lock();
```

```
        reqId += reqCounter;
        reqCounter++;
        counterLock.unlock();
        try {
            publish(splitStrings[0], dt, reqId);
        } catch (AccessException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (RemoteException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (NotBoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    /*
     *  executeCommandsFromFile(filename)   -   Read the filename file
and execute commands from this file line by line.
     */
    static void executeCommandsFromFile(String filename) {
        File testfile = new File(filename);
        Scanner reader;
        try {
            reader = new Scanner(testfile);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
            return;
        }
        while (reader.hasNextLine()) {
            String line = reader.nextLine();
            executeCommand(line);
        }
        reader.close();
    }

    /*
     *  takeInputFromCommandLine()      -   Take input from terminal and
```

```
execute commands line by line.
     */
    static void takeInputFromCommandLine() {
        Scanner scanner = new Scanner(System.in);
        while (true) {
            String cmd = scanner.nextLine();
            if (cmd.compareTo("exit")==0) {
                scanner.close();
                return;
            }
            executeCommand(cmd);
        }
    }


    public static String getUUID() {
        return UUID;
    }

    /*
     *  main() -
     *
     *  UUID is the unique identifier of JVM, usually it is of the form
pid@hostname.
     */

    public static void main(String[] args) {
        UUID = ManagementFactory.getRuntimeMXBean().getName();
        try {
            registry = LocateRegistry.getRegistry();
        } catch (RemoteException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        if (args.length == 1) {
            executeCommandsFromFile(args[0]);
        } else {
            takeInputFromCommandLine();
        }
```

```
    }
}
```

Subscriber.java

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
import java.util.Scanner;
import java.util.concurrent.locks.ReentrantLock;
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.lang.management.ManagementFactory;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

public class Subscriber implements SubscriberInterface {
    String UUID = "";
    String logFile = "log_sub";
    Integer reqCounter = 0;
    ReentrantLock counterLock = new ReentrantLock(true);
    Registry registry;

    /*
     *  (un)subscribe(topic, ReqID) -   calls the master server and
performs the appropriate function.
     */
    private void unsubscribe(String topic, String ReqID) {
        // call server.unregisterSubscriber()
        try {
```

```
            ServerInterface server = (ServerInterface)
registry.lookup("master");
            server.unregisterSubscriber(topic, UUID, ReqID);
            System.out.println("UnSubscribe @"+topic);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    private void subscribe(String topic, String ReqID) {
        // call server.registerSubscriber()
        try {
            ServerInterface server = (ServerInterface)
registry.lookup("master");
            server.registerSubscriber(topic, UUID, ReqID);
            System.out.println("Subscribe @"+topic);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    /*
     *  receiveDta(topic, dt, ReqID)    -   This function is called by
server to send data to the subscriber object.
     */
    public void receiveData(String topic, Data dt, String ReqID) {
        // receive data from server. This is called by server
        System.out.println("Received @"+topic+" Data: "+dt.getData()+"
with reqID: "+ReqID);
        outputToLog(dt.getData());
    }

    /*
     *  outputToLog(log)    -   Append log to the logfile.
     */
    private void outputToLog(String log){
        try {
```

```
            BufferedWriter writer = new BufferedWriter(new
FileWriter("./logs/"+logFile+".txt",true));
            writer.write(log + "\n");
            writer.flush();
            writer.close();
        } catch (IOException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
    }

    public void updateLogFileName(String name){
        logFile = name;
        Path path = Paths.get("./logs/"+logFile+".txt");
        try {
            Files.deleteIfExists(path);
        }
        catch (IOException e) {
            e.printStackTrace();
        }
        File logfile = new File("./logs/"+logFile+".txt");
    }

    /*
     *  executeCommand(String line) -   Extract and execute the command
from line.
     */
    private void executeCommand(String line) {
        String[] splitStrings = line.split(" ");
        if(splitStrings.length != 2)
            return;
        String reqId = UUID;
        counterLock.lock();
        reqId += reqCounter;
        reqCounter++;
        counterLock.unlock();
        if(splitStrings[0].compareTo("S") == 0)
            subscribe(splitStrings[1], reqId);
        else if(splitStrings[0].compareTo("U") == 0)
```

```
            unsubscribe(splitStrings[1], reqId);
        else {
            System.err.println("ERROR: Invalid Command line: "+line);
        }
    }

    /*
     *  executeCommandsFromFile(filename)   -   Read the filename file
and execute commands from this file line by line.
     */
    public void executeCommandsFromFile(String filename) {
        File testfile = new File(filename);
        Scanner reader;
        try {
            reader = new Scanner(testfile);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
            return;
        }
        while (reader.hasNextLine()) {
            String line = reader.nextLine();
            executeCommand(line);
        }
        reader.close();
    }

    /*
     *  takeInputFromCommandLine()      -   Take input from terminal and
execute commands line by line.
     */
    public void takeInputFromCommandLine() {
        Scanner scanner = new Scanner(System.in);
        while (true) {
            String cmd = scanner.nextLine();
            if (cmd.compareTo("exit")==0) {
                scanner.close();
                return;
            }
            executeCommand(cmd);
```

```
        }
    }

    /*
     *  UUID is the unique identifier of JVM, usually it is of the form
pid@hostname.
     */
    public Subscriber(String name) {
        if (name == null){
            UUID = ManagementFactory.getRuntimeMXBean().getName();
        } else {
            UUID = name;
        }

        try {
            registry = LocateRegistry.getRegistry();
        } catch (RemoteException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    public String getUUID() {
        return UUID;
    }

    public void register() {
        Registry registry;
        try {
            registry = LocateRegistry.getRegistry();
            registry.rebind(UUID, this);
        } catch (RemoteException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

    }

    public static void main(String[] args) throws RemoteException {
```

```
        Subscriber obj;
        if (args.length == 2) {
            obj = new Subscriber(args[0]);
        } else {
            obj = new Subscriber(null);
        }

        // Create object, bind to UUID and call
executeCommandsFromFile();
        SubscriberInterface robj = (SubscriberInterface)
UnicastRemoteObject.exportObject(obj,0);
        robj.register();
        if (args.length == 1){
            robj.executeCommandsFromFile(args[0]);
        } else if (args.length == 2) {
            robj.updateLogFileName(args[1]);
            robj.takeInputFromCommandLine();
        } else {
            robj.takeInputFromCommandLine();
        }
    }
}
```

Server.java

```
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
import java.util.concurrent.locks.ReentrantReadWriteLock;
import java.rmi.registry.LocateRegistry;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
```

```java
import java.io.IOException;
import java.lang.ProcessBuilder.Redirect;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.rmi.NotBoundException;
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.ArrayList;

/*
 *  Server Class - This class is the manager server b/w the publishers and subscribers.
 *
 *  This class have 2 modes of working,
 *  a. As Master - When the class is working as master, it receives requests and data from publishers and subscribers.
 *                  Master also propogates requests to the registered slave. This makes sure both the master and slave have
 *                  same topicSubscriberList HashMap.
 *  b. As Slave - When the class is working as slave, it receives requests from the master. When slave starts up, it does a bulk-
 *                  transfer to sync the topicSubscriberList, after that requests are received incrementally.
 *
 */

public class Server implements ServerInterface{
    /*
     * topicSubscriberList - Maintain RMI ids of subscribers registered to topics.
     * reentrantReadWriteLock - Used to lock the topicSubscriberList.
     */
    HashMap<String, Set<String> > topicSubscriberList;
    ReentrantReadWriteLock reentrantReadWriteLock = new ReentrantReadWriteLock();

    public Server() {
        topicSubscriberList = new HashMap<>();
```

```
    }

    /*
     *  amIUp()     -   This skeleton function is called by the Slave to
make sure that master is working or not.
     *
     *  isMasterUp()-   Get stub for master server and try calling the
amIUp(). If the function executes then
     *                  the master server is up.
     */
    public boolean amIUp() {
        return true;
    }

    public boolean isMasterUp() {
        // Get slave instance from RMI
        try {
            // Get the registry
            Registry registry = LocateRegistry.getRegistry();
            // Look up the registry for the remote object
            try {
                ServerInterface stub = (ServerInterface)
registry.lookup("master");
                return stub.amIUp();
            } catch(NotBoundException e) {
                return false;
            }
        } catch (Exception e) {
            System.err.println("Client exception: " + e.toString());
            e.printStackTrace();
        }
        return false;
    }

    /*
     *  becomeMaster()  -   This function binds the current object to
"master" key in the RMI registry.
     *
     *  becomeSlave()   -   This function first sync the
```

```
topicSubscriberList with master and then binds current object
     *                      to the "slave" key in the RMI registry.
     */
    public int becomeMaster() {

        try {
            // Export the remote object to the stub
            // ServerInterface stub = (ServerInterface)
UnicastRemoteObject.exportObject(this, 0);
            // Bind the remote object (stub) in the registry
            Registry registry = LocateRegistry.getRegistry();
            registry.rebind("master", this);
            return 0;
        } catch (Exception e) {
            System.err.println("Server exception: " + e.toString());
            e.printStackTrace();
        }
        return -1;
    }
    public void becomeSlave() {
        Registry registry;
        try{
            registry = LocateRegistry.getRegistry();
            ServerInterface server = (ServerInterface)
registry.lookup("master");
            topicSubscriberList = server.syncWithSlave();
        }
        catch(Exception e) {
            e.printStackTrace();
        }
        try {
            // Export the remote object to the stub
            // ServerInterface stub = (ServerInterface)
UnicastRemoteObject.exportObject(this, 0);
            // Bind the remote object (stub) in the registry
            registry = LocateRegistry.getRegistry();
            registry.rebind("slave", this);
        } catch (Exception e) {
            System.err.println("Server exception: " + e.toString());
```

```
            e.printStackTrace();
        }
    }

    /*
     *  startNewSlave() -   Boots up a new Server jvm and redirects the
input, output and error streams to the original terminal.
     */
    private void startNewSlave() {
        String[] args = new String[] { "java", "Server"};
        ProcessBuilder pb = new ProcessBuilder(args);
        try {
            pb.redirectOutput(Redirect.INHERIT);
            pb.redirectError(Redirect.INHERIT);
            pb.redirectInput(Redirect.INHERIT);     // CTRL + C brings
everything down and we don't have to kill all processes manually.
            pb.start();
        } catch (IOException e) {
            System.out.println(e);
        }
    }

    /*
     *  (un)subscribeToSlave(topic, UUID, ReqID) - Sends the
(un)subscribe request to the slave.
     */

    private void subscribeToSlave(String topic, String UUID, String
ReqID) {
        // Get slave instance from RMI
        try {
            // Get the registry
            Registry registry = LocateRegistry.getRegistry();
            // Look up the registry for the remote object
            ServerInterface stub = (ServerInterface)
registry.lookup("slave");
            // Pass a message to the remote object
            stub.__registerSubscriber(topic, UUID, ReqID);
        } catch (Exception e) {
```

```
            System.err.println("Client exception: " + e.toString());
            e.printStackTrace();
        }
    }
    private void unsubscribeToSlave(String topic, String UUID, String
ReqID) {
        // Get slave instance from RMI
        try {
            // Get the registry
            Registry registry = LocateRegistry.getRegistry();
            // Look up the registry for the remote object
            ServerInterface stub = (ServerInterface)
registry.lookup("slave");
            // Pass a message to the remote object
            stub.__unregisterSubscriber(topic, UUID, ReqID);
        } catch (Exception e) {
            System.err.println("Client exception: " + e.toString());
            e.printStackTrace();
        }
    }
    /*
     *  syncWithSlave() - Returns the topicSubscriberList of master.
     */
    public HashMap<String,Set<String> > syncWithSlave() {
        return topicSubscriberList;
    }

    public void lockMaster() {
        reentrantReadWriteLock.writeLock().lock();
    }

    public void unlockMaster() {
        reentrantReadWriteLock.writeLock().unlock();
    }

    /*
     *  sendToSubscribers(topic, dt, ReqID) - This function is called by
the publisher to send the data to the subscribers.
     */
```

```
    public void sendToSubscribers(String topic, Data dt, String ReqID) {
        // Acquire lock on topicSubsriberList
        reentrantReadWriteLock.readLock().lock();
        Set <String> topicSubscribers = topicSubscriberList.get(topic);
        reentrantReadWriteLock.readLock().unlock();

        try {
            outputToLog("P " + topic + " " + dt.getData());
        } catch(IOException e){
            e.printStackTrace();
        }

        if(topicSubscribers == null)
            return;

        // Iterate over all the subscribers of this topic and send the
data.
        for (String sub: topicSubscribers) {
            try {
                Registry registry = LocateRegistry.getRegistry();
                SubscriberInterface stub = (SubscriberInterface)
registry.lookup(sub);
                stub.receiveData(topic, dt, ReqID);
            } catch (Exception e) {
                System.err.println("Client exception: " + e.toString());
                e.printStackTrace();
            }
        }
    }

    /*
     *  outputToLog(str)    -   This function appends str to the log file
of the server.
     */
    public void outputToLog(String str) throws IOException {
        BufferedWriter writer = new BufferedWriter(new
FileWriter("./logs/server.txt",true));
        writer.write(str+ "\n");
        writer.flush();
```

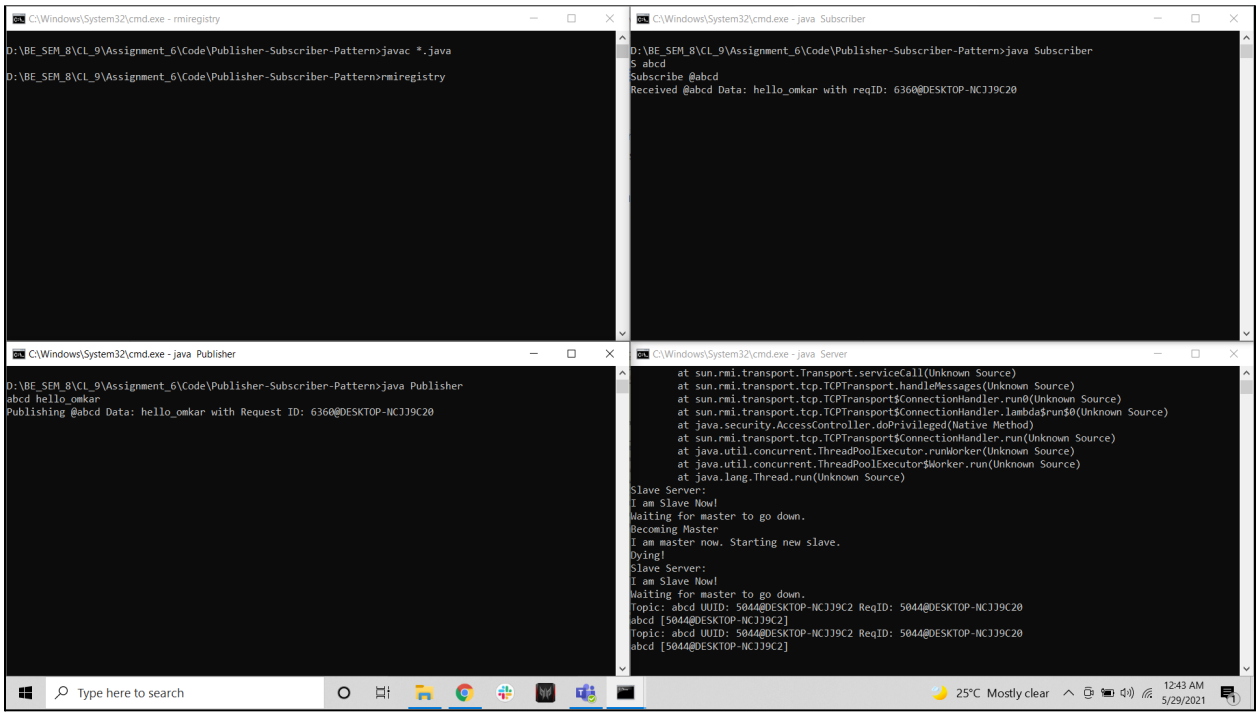
```
        writer.close();
    }

    /*
     *  createLogFile() -   Deletes old log file and creates a blank
file.
     */
    public void createLogFile(){
        Path path = Paths.get("./logs/server.txt");
        try {
            Files.deleteIfExists(path);
        }
        catch (IOException e) {
            e.printStackTrace();
        }
        File newFile = new File("./logs/server.txt");
    }

    /*
     *  __(un)registerSubscriber(topic, UUID, ReqID)    -   Directly
changes the topicSubscriberList based on the request.
     *
     *  (un)registerSubscriber(topic, UUID, ReqID)  -   Do the changes
locally and propogate the changes to the slave.
     */
    public void __registerSubscriber(String topic, String UUID, String
ReqID) {
        // Update the hashmap
        if (!topicSubscriberList.containsKey(topic)) {
            topicSubscriberList.put(topic, new HashSet<String>());
        }
        topicSubscriberList.get(topic).add(UUID);
        System.out.println("Topic: "+topic +" UUID: "+UUID+" ReqID:
"+ReqID);
        printTopicList();
    }
    public void registerSubscriber(String topic, String UUID, String
ReqID) {
        // Acquire lock on topicSubsriberList
```

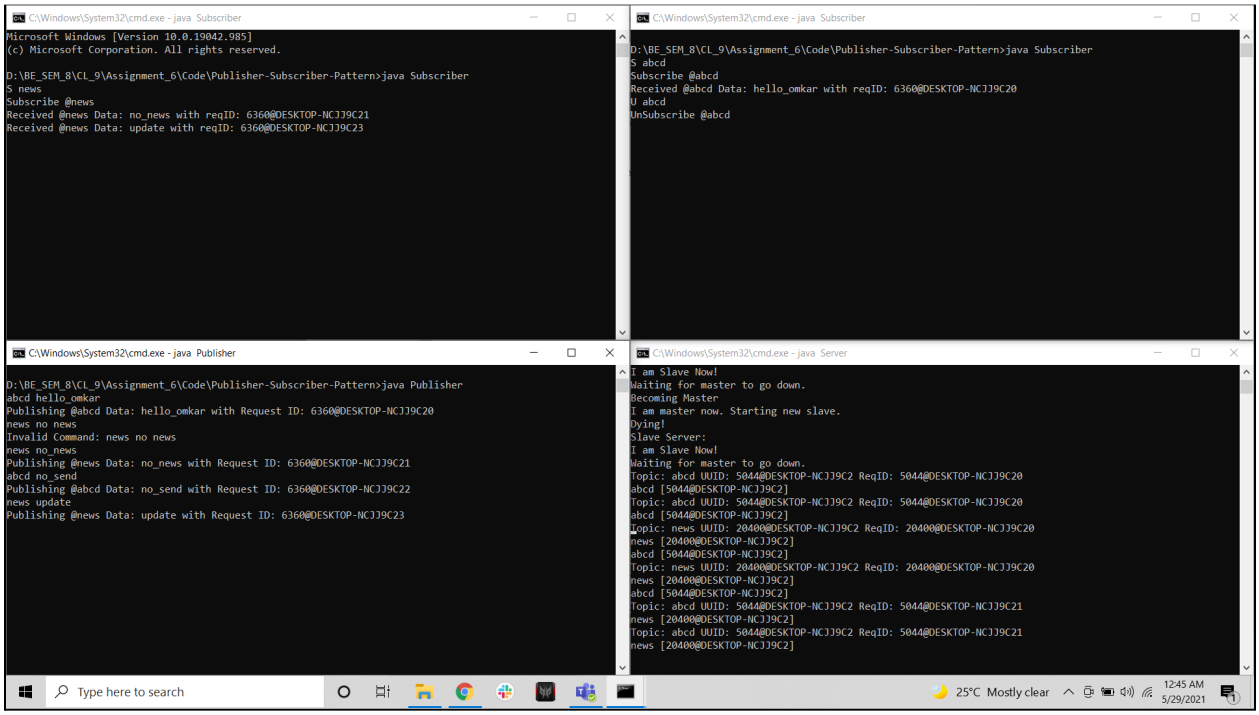
```
        reentrantReadWriteLock.writeLock().lock();
        try {
            // Send this data to slave
            __registerSubscriber(topic, UUID, ReqID);
            try {
                outputToLog("S " + UUID + " " + topic);
            } catch(IOException e){
                e.printStackTrace();
            }
        } finally {
            // Release lock
            reentrantReadWriteLock.writeLock().unlock();
            subscribeToSlave(topic, UUID, ReqID);
        }
    }
    public void __unregisterSubscriber(String topic, String UUID, String
ReqID) {
        // Update the hashmap
        if (topicSubscriberList.containsKey(topic)) {
            topicSubscriberList.get(topic).remove(UUID);
            if (topicSubscriberList.get(topic).size()==0){
                topicSubscriberList.remove(topic);
            }
        }
        System.out.println("Topic: "+topic +" UUID: "+UUID+" ReqID:
"+ReqID);
        printTopicList();
    }
    public void unregisterSubscriber(String topic, String UUID, String
ReqID) {
        // Acquire lock on topicSubsriberList
        reentrantReadWriteLock.writeLock().lock();
        try {
            // Send this data to slave
            __unregisterSubscriber(topic, UUID, ReqID);
            try {
                outputToLog("U " + UUID + " " + topic);
            } catch(IOException e){
                e.printStackTrace();
```

```
            }
        } finally {
            // Release lock
            reentrantReadWriteLock.writeLock().unlock();
            unsubscribeToSlave(topic, UUID, ReqID);
        }
    }

    /*
     *  printTopicList()    -   Simply print the current topicSubscriber
List.
     */
    public void printTopicList() {
        for (String ls: topicSubscriberList.keySet()) {
            System.out.println(ls+" "+topicSubscriberList.get(ls));
        }
    }

    public static void main(String[] args) throws InterruptedException,
RemoteException {
        Server sobj = new Server();
        ServerInterface rmobj = (ServerInterface)
UnicastRemoteObject.exportObject(sobj, 0);
        sobj.createLogFile();
        rmobj.becomeSlave();    //  Start the server as a slave.

        System.out.println("Slave Server: ");
        rmobj.printTopicList(); //  Prints the initial
topicSubsriberList of slave.
        System.out.println("I am Slave Now!");

        System.out.println("Waiting for master to go down.");
        while(rmobj.isMasterUp()) {     // Poll the master every 0.5
secs.
            Thread.sleep(500);
        }

        System.out.println("Becoming Master");
        if(rmobj.becomeMaster() != 0 ) {
```

```
            System.out.println("Unable to become master. I will not
start a slave.");
            System.out.println("Dying!");
            return;
        }

        System.out.println("I am master now. Starting new slave.");
        sobj.startNewSlave();
        System.out.println("Dying!");
    }
}
```

OUTPUT :


C:\Windows\System32\cmd.exe - rmiregistry
D:\BE_SEM_8\CL_9\Assignment_6\Code\Publisher-Subscriber-Pattern>javac *.java
D:\BE_SEM_8\CL_9\Assignment_6\Code\Publisher-Subscriber-Pattern>rmiregistry
C:\Windows\System32\cmd.exe - java Subscriber
D:\BE_SEM_8\CL_9\Assignment_6\Code\Publisher-Subscriber-Pattern>java Subscriber
S abcd
Subscribe @abcd
Received @abcd Data: hello_omkar with reqID: 6360@DESKTOP-NCJJ9C20
C:\Windows\System32\cmd.exe - java Publisher
D:\BE_SEM_8\CL_9\Assignment_6\Code\Publisher-Subscriber-Pattern>java Publisher
abcd hello_omkar
Publishing @abcd Data: hello_omkar with Request ID: 6360@DESKTOP-NCJJ9C20
C:\Windows\System32\cmd.exe - java Server
at sun.rmi.transport.Transport.serviceCall(Unknown Source)
at sun.rmi.transport.tcp.TCPTransport.handleMessages(Unknown Source)
at sun.rmi.transport.tcp.TCPTransport$ConnectionHandler.run0(Unknown Source)
at sun.rmi.transport.tcp.TCPTransport$ConnectionHandler.lambda$run$0(Unknown Source)
at java.security.AccessController.doPrivileged(Native Method)
at sun.rmi.transport.tcp.TCPTransport$ConnectionHandler.run(Unknown Source)
at java.util.concurrent.ThreadPoolExecutor.runWorker(Unknown Source)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(Unknown Source)
at java.lang.Thread.run(Unknown Source)
Slave Server:
I am Slave Now!
Waiting for master to go down.
Becoming Master
I am master now. Starting new slave.
Dying!
Slave Server:
I am Slave Now!
Waiting for master to go down.
Topic: abcd UUID: 5044@DESKTOP-NCJJ9C2 ReqID: 5044@DESKTOP-NCJJ9C20
abcd [5044@DESKTOP-NCJJ9C2]
Topic: abcd UUID: 5044@DESKTOP-NCJJ9C2 ReqID: 5044@DESKTOP-NCJJ9C20
abcd [5044@DESKTOP-NCJJ9C2]
Type here to search
25°C Mostly clear
12:43 AM
5/29/2021


C:\Windows\System32\cmd.exe - java Subscriber
Microsoft Windows [Version 10.0.19042.985]
(c) Microsoft Corporation. All rights reserved.
D:\BE_SEM_8\CL_9\Assignment_6\Code\Publisher-Subscriber-Pattern>java Subscriber
S news
Subscribe @news
Received @news Data: no_news with reqID: 6360@DESKTOP-NCJJ9C21
Received @news Data: update with reqID: 6360@DESKTOP-NCJJ9C23
C:\Windows\System32\cmd.exe - java Subscriber
D:\BE_SEM_8\CL_9\Assignment_6\Code\Publisher-Subscriber-Pattern>java Subscriber
S abcd
Subscribe @abcd
Received @abcd Data: hello_omkar with reqID: 6360@DESKTOP-NCJJ9C20
U abcd
UnSubscribe @abcd
C:\Windows\System32\cmd.exe - java Publisher
D:\BE_SEM_8\CL_9\Assignment_6\Code\Publisher-Subscriber-Pattern>java Publisher
abcd hello_omkar
Publishing @abcd Data: hello_omkar with Request ID: 6360@DESKTOP-NCJJ9C20
news no_news
Invalid Command: news no news
news no_news
Publishing @news Data: no_news with Request ID: 6360@DESKTOP-NCJJ9C21
abcd no_send
Publishing @abcd Data: no_send with Request ID: 6360@DESKTOP-NCJJ9C22
news update
Publishing @news Data: update with Request ID: 6360@DESKTOP-NCJJ9C23
C:\Windows\System32\cmd.exe - java Server
I am Slave Now!
Waiting for master to go down.
Becoming Master
I am master now. Starting new slave.
Dying!
Slave Server:
I am Slave Now!
Waiting for master to go down.
Topic: abcd UUID: 5044@DESKTOP-NCJJ9C2 ReqID: 5044@DESKTOP-NCJJ9C20
abcd [5044@DESKTOP-NCJJ9C2]
Topic: abcd UUID: 5044@DESKTOP-NCJJ9C2 ReqID: 5044@DESKTOP-NCJJ9C20
abcd [5044@DESKTOP-NCJJ9C2]
Topic: news UUID: 20400@DESKTOP-NCJJ9C2 ReqID: 20400@DESKTOP-NCJJ9C20
news [20400@DESKTOP-NCJJ9C2]
abcd [5044@DESKTOP-NCJJ9C2]
Topic: news UUID: 20400@DESKTOP-NCJJ9C2 ReqID: 20400@DESKTOP-NCJJ9C20
news [20400@DESKTOP-NCJJ9C2]
abcd [5044@DESKTOP-NCJJ9C2]
Topic: abcd UUID: 5044@DESKTOP-NCJJ9C2 ReqID: 5044@DESKTOP-NCJJ9C21
news [20400@DESKTOP-NCJJ9C2]
Topic: abcd UUID: 5044@DESKTOP-NCJJ9C2 ReqID: 5044@DESKTOP-NCJJ9C21
news [20400@DESKTOP-NCJJ9C2]
Type here to search
25°C Mostly clear
12:45 AM
5/29/2021