ALPER SIMSEK HW2 34293 REPORT

I have used Java’s RMI implemtation for this project.

There are 6 different java files.

2 for interfaces.

2 for drivers.

2 for classes.

To Implement it correctly, developer has to create interfaces of the server and client part of the project.

These interfaces extends Remote interface of the RMI.

Server Interface includes two function;
First one is for registering the client. This takes chatClient interface as input.
Second one is for broadcasting the message. This takes message as input.
These two function throws RemoteException to hanle unexpected server problem.

Chat Interface includes only one function.
And this is for retrieving message. But this throws RemoteException to handle unexpected server problems.

Chat Client is for handling everything in the client part of the server.
It extends UnicastRemoteObject and implements Chat Client Interface and Runnable.

It has 4 variable.

Chat server interface,

Name as String,

Id for process id,

Timestamp vector as ArrayList;

Id stores the unique id of the process.

Timestamp list stores the vector timestamps of the processes in the server.

Init time stamp function initializes the timestamp vector as zeros in the beginning.
Add time stamp function adds +1 for the processes vector timestamps.
Retrieve message function takes message as input and splits it to the message and vector timestamps.
Create str of list function creates a string for broadcasting timestamps as string which takes timestamp list as input.
Run function is for waiting new line of input and broadcasting the message to the all.

Char Client Driver is for main function of the Client side.
It looks up the url of the server with Naming.lookup(url) and binds to it.
Then creates a new thread of ChatClient with name and id which is given in the args.

Chat Server is implements Server Interface and extends UnicastRemoteObject as client.
This class has only one variable which stores the clients in the array list.
ChatServer initializes the Server and throws RemoteException.
Registerchatclient is a void function and ads client to the client list. Also this function is synchronized.
Broadcast message is also a void function and broadcasts the message to the all clients.
This function is also synchronized and in a for loop each client retrieves messages.

Chat Server Driver is for main function of the server.
Which do’s only one thing, rebinds the server to the rmi server.

When all classes are written, in the src file developer has to use this command in the terminal to create classes.
“javac *.java”

Then it has to register rmi in the current folder. Which is also same in the server side of the AWS with

“rmiregistry” command.

Then to start server open new terminal and go to the source code of the project and

“java ChatServerDriver” copy this to the terminal.

When server initializes

Open 5 different terminals and do the followings.

1) Go to the source code folder
2) java ChatClientDriver name id
3) then start writing

For this all of to work in AWS EC2 machines,

There is two different way.

First, create 6 different machines in the same VPC.

Second, create 6 different machines and open all the ports to the world.

Becase rmi can use different port in the different times. So opening all the ports works, but this is not secure.

Go to the AWS console and open EC2 page.

Click Launch Instance and pick ubuntu 18.04

In the networks part do the things about VPC.

Launch instance with a key pair you have created before in the EC2 page.

After instance is created goto your terminal and write this command

“ssh -i <pem file>.pem ubuntu@<id-of-machine>”

Then update with

“sudo apt install update” (18.04) otherwise use apt-get.

Then install java

“sudo apt install default-jdk”

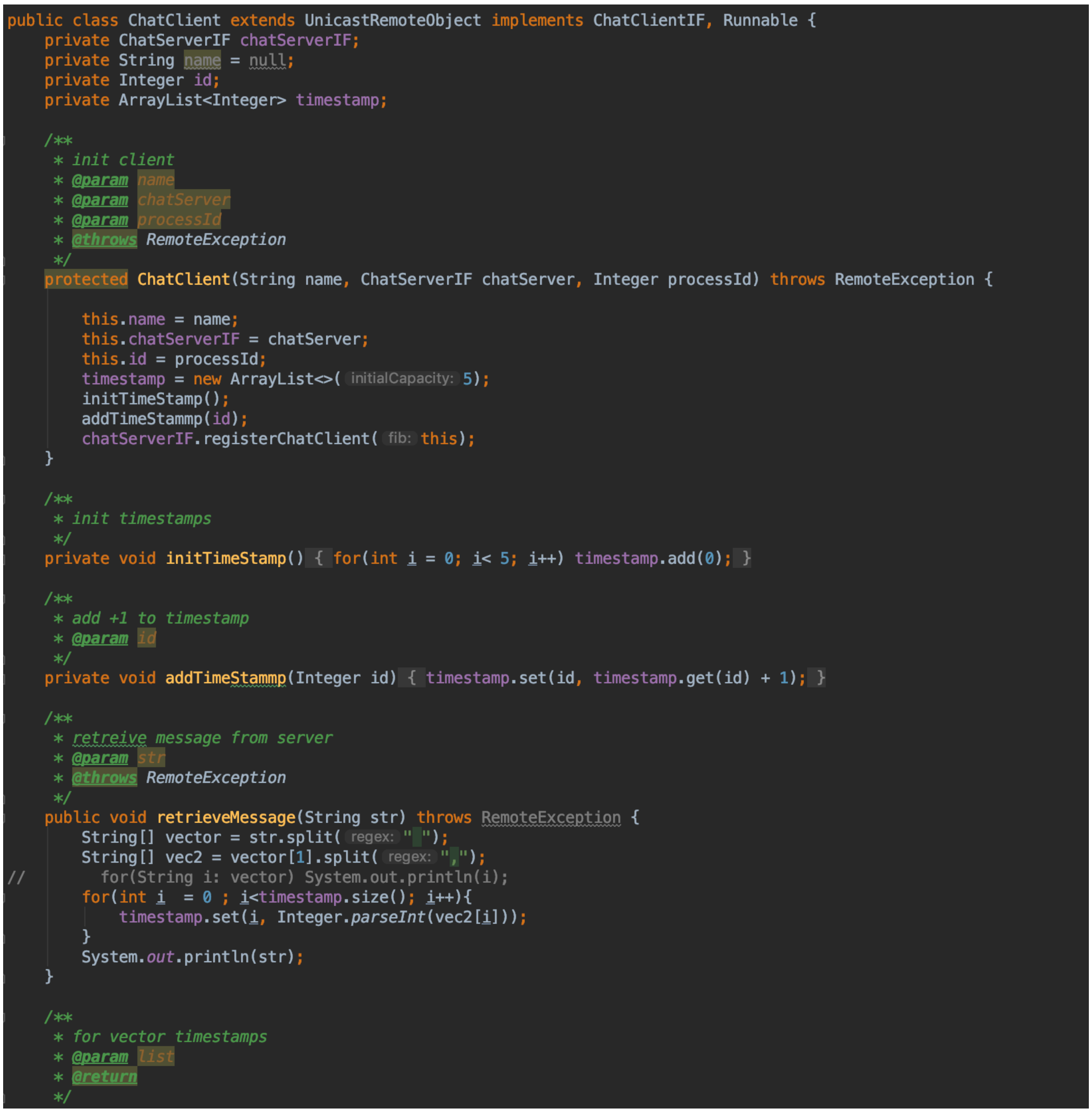
After that copy all the files in your projects source code with the;

“scp -I <pem file>.pem *.java ubuntu@<id-of-machine>”

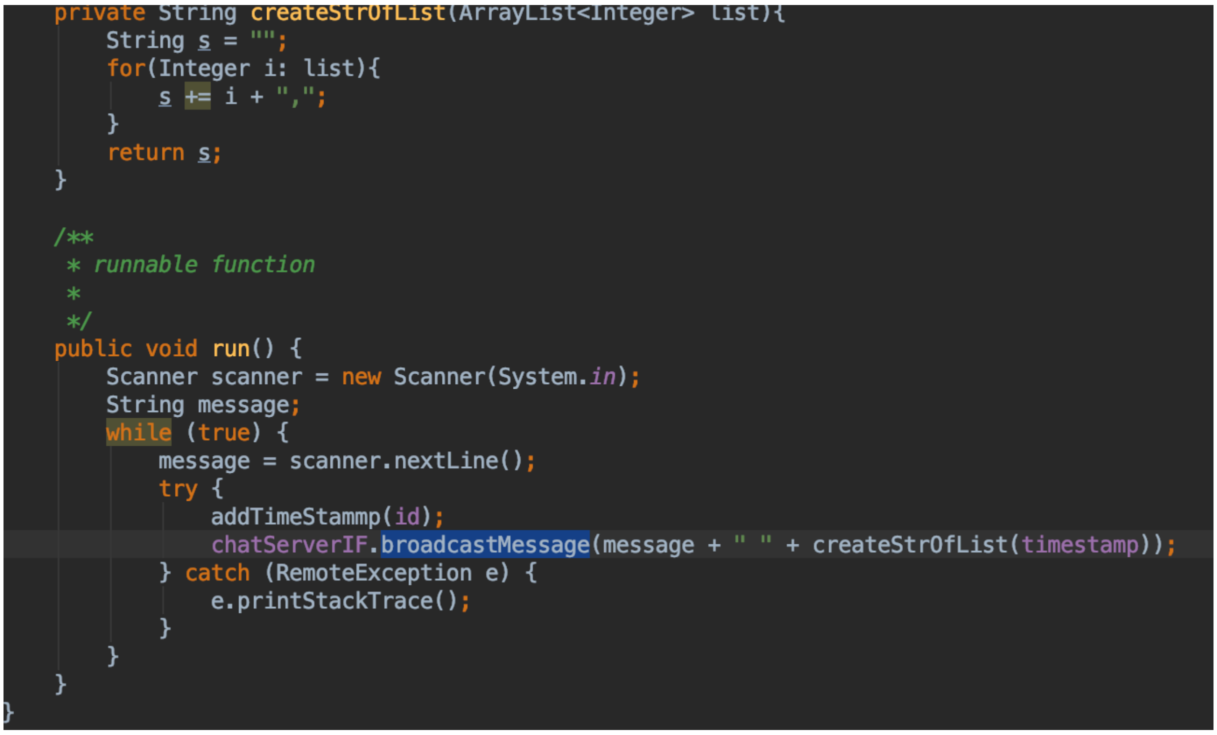
Then repeat the same processes like in your local machine.

You have to open two terminal in your server instance. Because you have to register rmi and then create server.

Chat Client

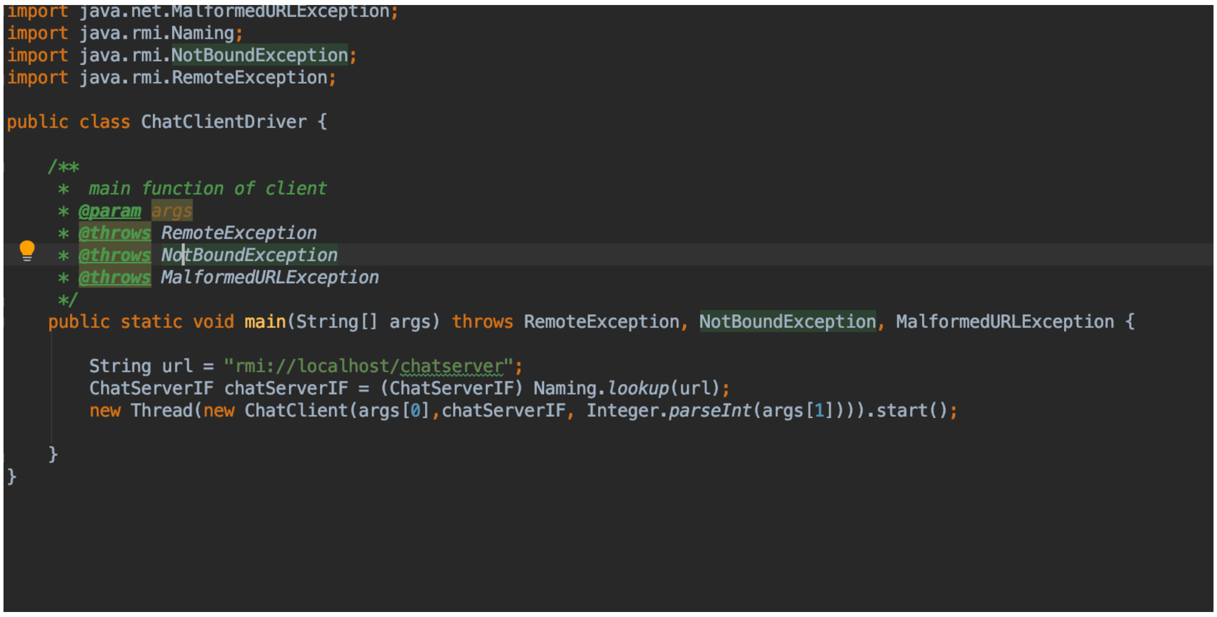


```java
public class ChatClient extends UnicastRemoteObject implements ChatClientIF, Runnable {
    private ChatServerIF chatServerIF;
    private String name = null;
    private Integer id;
    private ArrayList<Integer> timestamp;

    /**
     * init client
     * @param name
     * @param chatServer
     * @param processId
     * @throws RemoteException
     */
    protected ChatClient(String name, ChatServerIF chatServer, Integer processId) throws RemoteException {

        this.name = name;
        this.chatServerIF = chatServer;
        this.id = processId;
        timestamp = new ArrayList<>( initialCapacity: 5);
        initTimeStamp();
        addTimeStammp(id);
        chatServerIF.registerChatClient( fib: this);
    }

    /**
     * init timestamps
     */
    private void initTimeStamp() { for(int i = 0; i< 5; i++) timestamp.add(0); }

    /**
     * add +1 to timestamp
     * @param id
     */
    private void addTimeStammp(Integer id) { timestamp.set(id, timestamp.get(id) + 1); }

    /**
     * retreive message from server
     * @param str
     * @throws RemoteException
     */
    public void retrieveMessage(String str) throws RemoteException {
        String[] vector = str.split( regex: " ");
        String[] vec2 = vector[1].split( regex: ",");
//          for(String i: vector) System.out.println(i);
        for(int i  = 0 ; i<timestamp.size(); i++){
            timestamp.set(i, Integer.parseInt(vec2[i]));
        }
        System.out.println(str);
    }

    /**
     * for vector timestamps
     * @param list
     * @return
     */
```



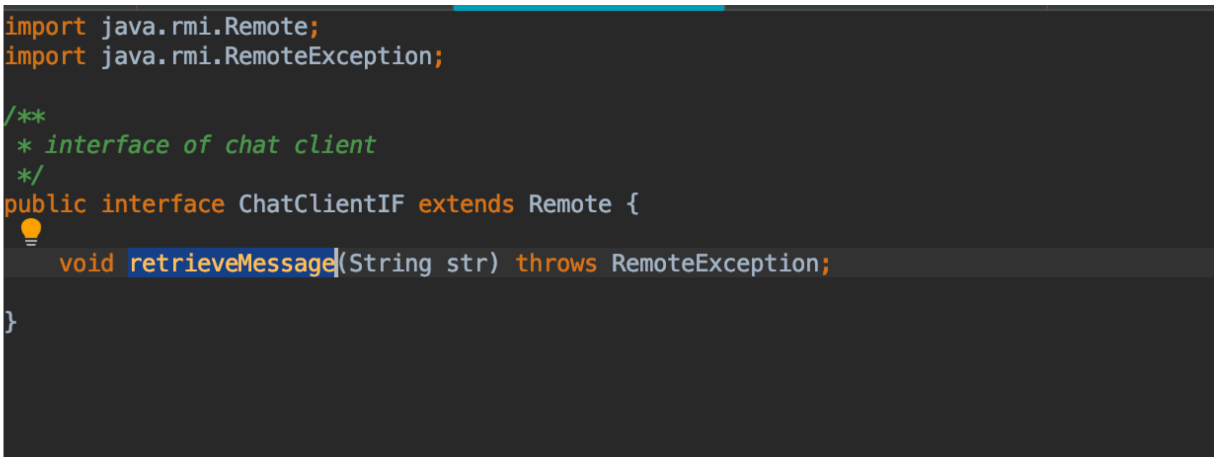
```java
    private String createStrOfList(ArrayList<Integer> list){
        String s = "";
        for(Integer i: list){
            s += i + ",";
        }
        return s;
    }

    /**
     * runnable function
     *
     */
    public void run() {
        Scanner scanner = new Scanner(System.in);
        String message;
        while (true) {
            message = scanner.nextLine();
            try {
                addTimeStammp(id);
                chatServerIF.broadcastMessage(message + " " + createStrOfList(timestamp));
            } catch (RemoteException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Chat Client Driver



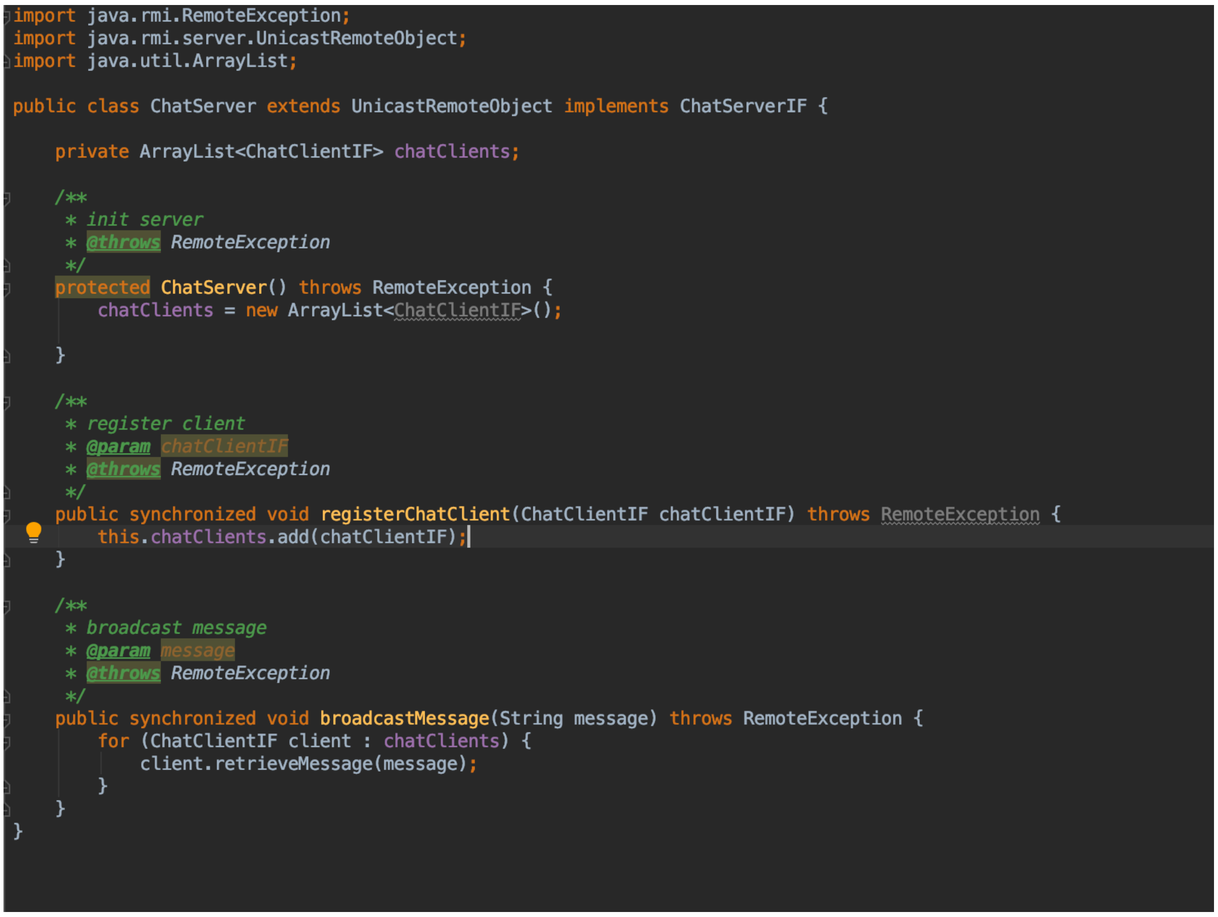
```java
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;

public class ChatClientDriver {

    /**
     *  main function of client
     * @param args
     * @throws RemoteException
     * @throws NotBoundException
     * @throws MalformedURLException
     */
    public static void main(String[] args) throws RemoteException, NotBoundException, MalformedURLException {

        String url = "rmi://localhost/chatserver";
        ChatServerIF chatServerIF = (ChatServerIF) Naming.lookup(url);
        new Thread(new ChatClient(args[0],chatServerIF, Integer.parseInt(args[1]))).start();

    }
}
```

CHATCLIENT INTERFACE



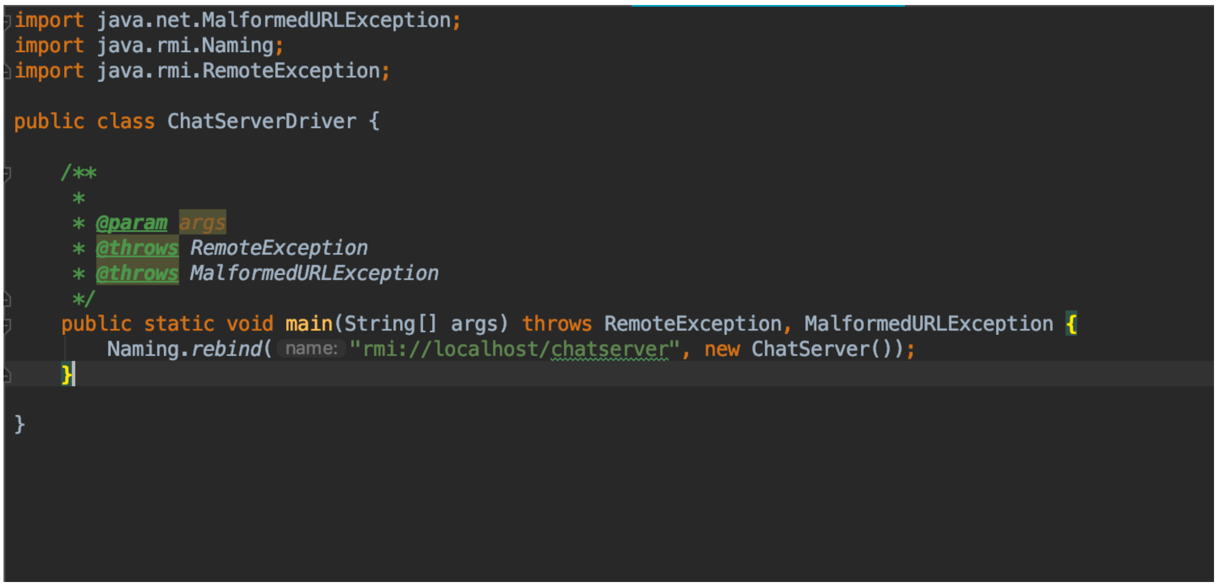
```java
import java.rmi.Remote;
import java.rmi.RemoteException;

/**
 * interface of chat client
 */
public interface ChatClientIF extends Remote {

    void retrieveMessage(String str) throws RemoteException;

}
```

SERVER CLASS



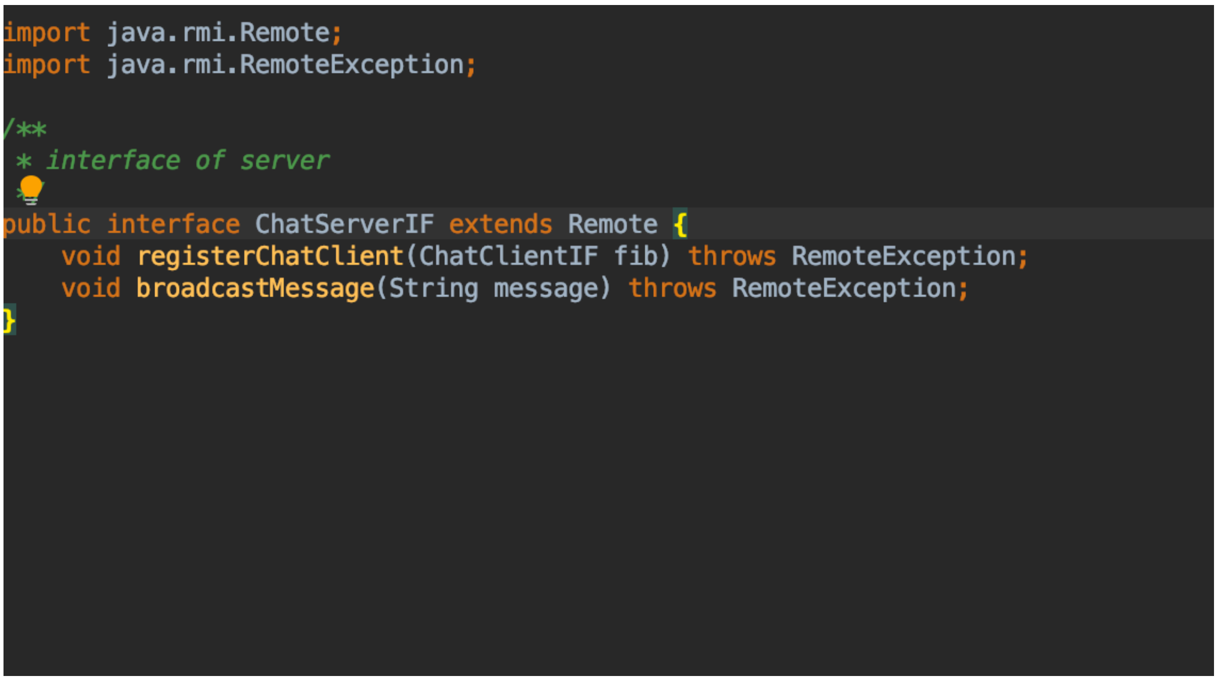
```java
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.ArrayList;

public class ChatServer extends UnicastRemoteObject implements ChatServerIF {

    private ArrayList<ChatClientIF> chatClients;

    /**
     * init server
     * @throws RemoteException
     */
    protected ChatServer() throws RemoteException {
        chatClients = new ArrayList<ChatClientIF>();

    }

    /**
     * register client
     * @param chatClientIF
     * @throws RemoteException
     */
    public synchronized void registerChatClient(ChatClientIF chatClientIF) throws RemoteException {
        this.chatClients.add(chatClientIF);
    }

    /**
     * broadcast message
     * @param message
     * @throws RemoteException
     */
    public synchronized void broadcastMessage(String message) throws RemoteException {
        for (ChatClientIF client : chatClients) {
            client.retrieveMessage(message);
        }
    }
}
```

CHAT SERVER DRIVER



```java
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.RemoteException;

public class ChatServerDriver {

    /**
     *
     * @param args
     * @throws RemoteException
     * @throws MalformedURLException
     */
    public static void main(String[] args) throws RemoteException, MalformedURLException {
        Naming.rebind( name: "rmi://localhost/chatserver", new ChatServer());
    }

}
```

CHAT SERVER INTERFACE



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

/**
 * interface of server
 */
public interface ChatServerIF extends Remote {
    void registerChatClient(ChatClientIF fib) throws RemoteException;
    void broadcastMessage(String message) throws RemoteException;
}
```