Assignment 3

Mehtab Kayani
ID: 9497

RMI Chat

1. present your algorithm design, showing in particular what RMI interactions occur among the components of your application
Answer:

RMI is a technique which facilitates calling a method or an object from a remote location. In case of my server client relationship there is a client object and a server object, when a client port invokes a remote method on a remote object, it is done by calling an ordinary method of java program language, i-e encapsulated in packaged object called stub, which resides on client machine not on a server.

This technique makes the use of device independent encoding the parameter, which is called the parameter marshalling.

An object running on one java virtual machine (JVM) can invoke methods on an object running on an another JVM through Java RMI. It allows remote communication to occur between two programs which are written in Java.

My application architecture is as follows.
Firstly defined remote interfaces **ServerInterface.java** and **ClientInterface.java**
The remote interface contains the declarations of the methods that can be called from other machines.
My remote interfaces extend the java.rmi.Remote interface.
The data type of any remote object that is passed as an argument or return value is declared as the remote interface type. For example ClientInterface not client.
Secondly I created the classes **Client.java** and **Server.java** which implements the above-mentioned interfaces respectively.

SERVER implementation:
The server program creates remote objects, makes references to these objects and waits for the clients to invoke methods on these objects.
The main method creates an instance of the remote object implementation, and then binds that instance to a name in a Java RMI *registry*. Every object has a global name and each name is registered in a registry.

```
main()

  try
      Server kayani = new Server();
             /* creating the registry*/
      LocateRegistry.createRegistry(Registry.REGISTRY_PORT);
             Naming.bind("server", kayani);
```

```
          Server is running and is available for users to connect...

       catch (RemoteException e)
          Error
       catch (MalformedURLException e)
          Error
       catch (AlreadyBoundException e)
          Error
```

In this example, the main method for the server is defined in the class Server which also implements the remote interface. The server's main method does the following:

a. **Starts the server**
   The statement “the server is running and is available for the users to connect is printed out on the console.
b. **Creates the registry and does the binding**

Client implementation:

On the server’s host the client program obtains a stub for the registry, looks up the remote objects stub by name in the registry, and then invokes the method on the remote object using the stub.

```
Client(String name, String ip)

            this.userName = name;

         try
this.server = (ServerInterface) Naming.lookup("rmi://"+ ip + "/server")
                  server.connect(this)

            catch (RemoteException e)
                  setLoginStatus(false)

                  IP-Address not found
```

- The client-side runtime opens a connection to the server using the host and port information in the remote object's stub and then serializes the call data.
- The server-side runtime accepts the incoming call, dispatches the call to the remote object, and serializes the result, asking user for the user name and ip address and once the process is completed, the connection is established and the user sees the success message.
- The client-side runtime receives, deserializes, and returns the result to the caller.

The response message returned from the remote invocation on the remote object is then printed to System.out.

```
 main( )
            signIn();
            users();
            duringChat();

            while (connectionStatus)
                  message()
            if message.equals("/connectedusers")
                        users();
                        duringChat();

                       show the list of connected users

             else if message.equals("/quit")
                        client.disconnect();
                        connectionStatus = false;
                     log the user out.

                  try
                        client.broadcast(message)
                   catch (RemoteException e)

                        Error message
```

Once the client program is run the client can signin with the help of **signIn()** method. The client has to set the user name and provide the ip address. If the name is already taken the user is informed with the message, and in case of wrong IP address the program throws an exception.
During the chat session the method **message()** takes care of the messages, the user can write messages as long as he is loggedIn, the and the connection status is set to true. The message is broadcasted if all the if statements are true with the help of the method **broadcast().** User can also type **/connectedusers** to check the users connected and the user can exit the program by typing **/quit,** user is connected with the help of **disconnect()** method.

Reference: ORACLE
http://docs.oracle.com/javase/7/docs/technotes/guides

2. develop an RMI-based Java implementation of both client and server. The Java solution should be robust wrt. client and server failures;
See the java files
Answer: See the java code for RMIchat program

3.Test your implementation and report on possible bugs and/or unexpected behaviours you should find.
Answer: I successfully tested the server in the main method by creating a new object of the server class and running the server, and also by connecting multiple users (clients) to the server.

4. discuss how the new version differs from the previous one based immediately on TCP and point out which changes and improvements or drawbacks were brought about by the new RMI version.
Answer:
RMI is simple method for developing and deploying distributed object applications in Java environment. It is for implementing application-level networking between Java applications. It is based on network interactions as Java method calls made to objects that live in other applications. It takes into consideration the higher level of abstraction. It doesn’t reveal the details of the sockets, connection, and transmission of data. The

RMI ports are blocked by default and a significant amount of effort is required to configure RMI-based applications.

RMI clients can directly invoke the server method, whereas socket-level programming is limited to passing values. Socket-level programming is very primitive and is like programming in assembly language, while RMI programming is like programming in a high-level language.

Sockets are ideal for programming communication between participants, streaming media, games and file transfer. The streams can be optimized by buffering or compressing data, but RMI is better for Inter-Application communication i-e client server communication.

Comparison between socket programming and RMI

Parameters	Socket programming	RMI
Platform	Independent	Dependent on Java strictly
Multithreading	Explicit	Implicit
Abstraction	low	High
Execution overhead	less	More
Speed	Faster	Comparatively slower
Security	High	Moderate
Control	Tighter control over sent data	Flexible can’t guarantee for the usage of same thread
Port Configuration	Easy	Difficult
Suited for	Communication between participants, gaming, file transfer	Inter-application communication