GOVERNMENT COLLEGE OF ENGINEERING, JALGAON
(An Autonomous Institute of Government of Maharashtra)
Department of Computer Engineering

Name: Vaibhav Bharat Sontakke	PRN No: 1942207
Class: T. Y. B. Tech.	Batch: L4
Date of Performance: _____	Date of Completion: _____
Subject: DSL	Sign. of Teacher with Date: _____

Experiment No.: 06

Aim: Program for Distributed Application using RMI

Title: Design and develop a distributed Hotel booking application using Java RMI. A distributed hotel booking system consists of the hotel server and the client machines. The server manages hotel rooms booking information. A customer can invoke the following operations at his machine

i) Book the room for the specific guest
ii) Cancel the booking of a guest

Software Requirements : Ubuntu OS, Eclipse IDE 4.11

Theory :

Java RMI - Introduction

RMI stands for **Remote Method Invocation**. It is a mechanism that allows an object residing in one system (JVM) to access/invoke an object running on another JVM.

RMI is used to build distributed applications; it provides remote communication between Java programs. It is provided in the package **java.rmi**.

Architecture of an RMI Application

In an RMI application, we write two programs, a **server program** (resides on the server) and a **client program** (resides on the client).

- Inside the server program, a remote object is created and reference of that object is made available for the client (using the registry).

- The client program requests the remote objects on the server and tries to invoke its methods. The following Figure 1.1 shows the architecture of an RMI application.

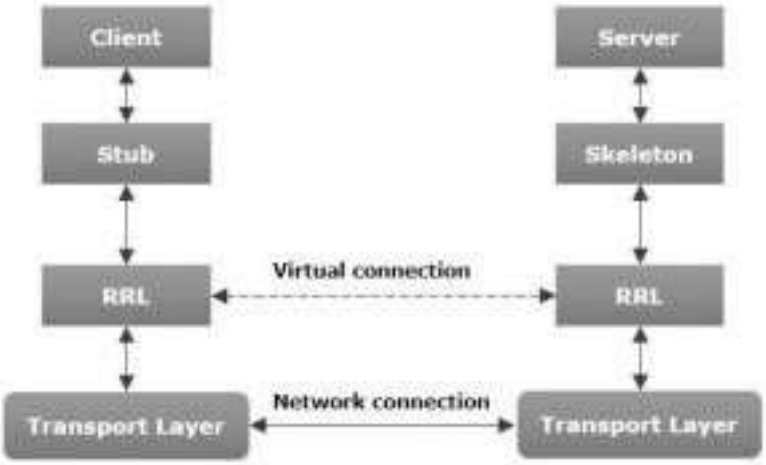

Figure 1.1 - Architecture of an RMI application

The components of this architecture are :

- **Transport Layer** − This layer connects the client and the server. It manages the existing connection and also sets up new connections.
- **Stub** − A stub is a representation (proxy) of the remote object at client. It resides in the client system; it acts as a gateway for the client program.
- **Skeleton** − This is the object which resides on the server side. **Stub** communicates with this skeleton to pass request to the remote object.
- **RRL (Remote Reference Layer)** − It is the layer which manages the references made by the client to the remote object.

Working of an RMI Application

The following points summarize how an RMI application works −

- When the client makes a call to the remote object, it is received by the stub which eventually passes this request to the RRL.
- When the client-side RRL receives the request, it invokes a method called **invoke()** of the object **remoteRef**. It passes the request to the RRL on the server side.

- The RRL on the server side passes the request to the Skeleton (proxy on the server) which finally invokes the required object on the server.
- The result is passed all the way back to the client.

RMI Registry

RMI registry is a namespace on which all server objects are placed. Each time the server creates an object, it registers this object with the RMIregistry (using **bind()** or **reBind()** methods). These are registered using a unique name known as **bind name**.

To invoke a remote object, the client needs a reference of that object. At that time, the client fetches the object from the registry using its bind name (using **lookup()** method).

The following illustration explains the entire process −

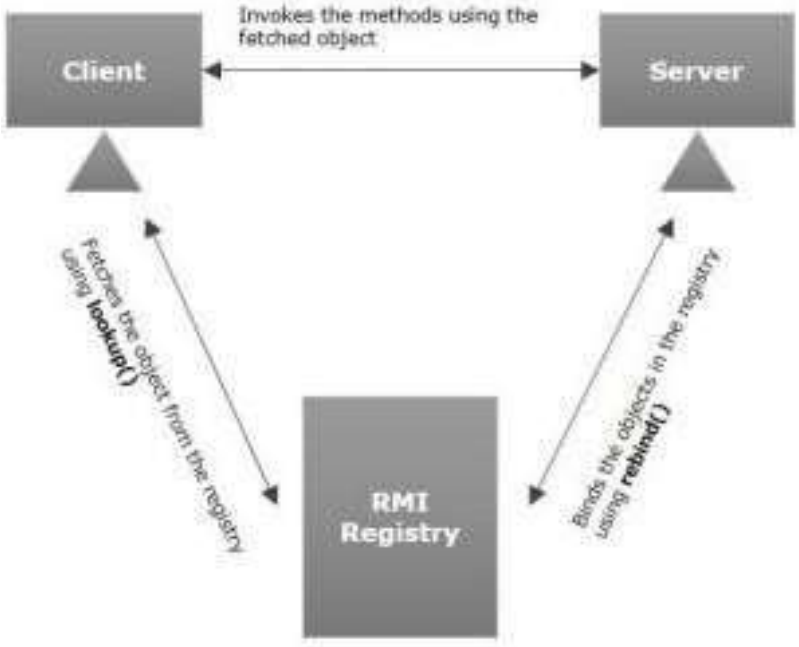

Figure 1.2 - Entire process RMI application

Goals of RMI

Following are the goals of RMI −

- To minimize the complexity of the application.
- To preserve type safety.
- Distributed garbage collection.
- Minimize the difference between working with local and remote objects.

Java RMI Application

To write an RMI Java application, you would have to follow the steps given below −

- Define the remote interface

- Develop the implementation class (remote object)
- Develop the server program
- Develop the client program
- Compile the application
- Execute the application.

Programming Requirement: You are required to write this client-server hotel booking system program, which communicates via RMI. Specifically, your program should consist of two parts: one for the client and another for the server. The client (the customer) will initiate a request by sending request message to the hotel server to execute a specified procedure (e.g. booking) with parameters.

Required files: Included are all files needed to run this application. These files are:

1. RoomBookingClient.java
2. RoomBookingServer.java
3. RoomBookingInterface.java
4. Room.java
5. RoomList.java
6. Connect.policy
7. Rooms.txt
8. rooms.txt

Design Requirement: You can use the following interface and class definitions as a starting point for your project. However, you are free to develop your own interface and class definitions.

1- Interface public interface HotelInterface extends Remote

```
{
public void book (int NumberOfGuests, int NumberOfNights, String
CustomerName, Date checkInDate)throws RemoteException; public boolean
cancelBooking(String CustomerName) throws RemoteException; public Date
inquiry(String CustomerName) throws RemoteException;
}
```

2- Server public class HotelServer extends UnicastRemoteObject implements HotelInterface

```
{
```

```
private Vector Customers; //guests' name private Vector
CheckInDate; //guests' check-in Date private Vector
NumOfNights; //guests' number of nights staying
```



```
private Vector NumbOfGuest; // number of guests boolean done;

public HotelServer() throws RemoteException
{ //implementation
}
public void book (int NumberOfGuests, int NumberOfNights, String CustomerName, Date checkInDate) throws RemoteException
{ //implementation
}
public boolean cancelBooking(String CustomerName) throws RemoteException
{ //implementation
}
public Date inquiry(String CustomerName) throws RemoteException {
//implementation
}
public static void main(String args[]) throws Exception
{ //init Hotel server
}
}
```

The Hotel server has only one input parameter "server_port", which specifies the port of rmiregistry. The default port of rmiregistry is 1099, but we may have to use other ports, if 1099 has already been used by some other programs.

3- Customer (Client)

```
public class Customer
{
public static void (String args[]) throws Exception
{ //get user's input, and perform the operations
}
}
```

The input parameters of a Customer must include:

- server_address: the address of the rmiregistry
- server_port: the port of the rmiregistry
- Operation: one of "book", "cancelBooking", and "Inquiry"
- CustomerName: assume it is unique for every user
- CheckInDate: entered in the book operation
- NumberOfNights: entered in the book operation

- NumberOfGuests: entered in the book operation

Algorithm Steps to run Distributed Application:

1. **Compile all java files**javac *.java

2. **Create Stub and Skeleton class file**

for RoomBookingServerrmic

RoomBookingServer **3. Run RMIC Registry**

rmiregistry &

4. **Run RoomBookingServer**java

RoomBookingServer

5. **In new terminal Run**

RoomBookingClientjava

RoomBookingClient

Note: Make sure that you are running the programs in same directory

Conclusion:

In this practical we learnt about Designing and Developing an Hotel Booking Distributed application using Java RMI.

DOC: 20/04/2021

.....

Miss. Archana Chitte

Name & Sign of Course Teacher