Distributed Systems Lab
Car Hire Booking Distributed System

Team Members:
Tania Rajabally (2017130047)
Shruti Rampure(2017130048)
Rahul Ramteke(2017130049)

Batch: C

Problem Statement:
The purpose of this system is to monitor and control the bookings of cars in a distributed environment. All the features of a typical Car Hire Booking System are discussed here by considering a distributed system.

CLIENT SERVER COMMUNICATION

Client/Server communication involves two components, namely a client and a server. They are usually multiple clients in communication with a single server. The clients send requests to the server and the server responds to the client requests.
The RMI (Remote Method Invocation) is an API that provides a mechanism to create distributed applications in java. The RMI allows an object to invoke methods on an object running in another JVM.
The RMI provides remote communication between the applications using two objects: stub and skeleton.
The stub is an object, and acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object.
The skeleton is an object, and acts as a gateway for the server side object. All the incoming requests are routed through it.
Remote Method Invocation (RMI) is a distributed object model for the Java Platform.
RMI is unique in that it is a language-centric model that takes advantage of a common network type system. uses object serialization to convert object graphs to byte-streams for transport. Any Java object type can be passed during invocation, including primitive types, core classes, user-defined classes, and JavaBeansTM. Java RMI could be described as a natural progression of procedural RPC, adapted to an object-oriented paradigm
RMI is implemented as three layers:
- A stub program in the client side of the client/server relationship, and a corresponding skeleton at the server end. The stub appears to the calling program to be the program being called for a service. (Sun uses the term proxy as a synonym for stub.)
- A Remote Reference Layer that can behave differently depending on the parameters passed by the calling program. For example, this layer can determine whether the request is to call a single remote service or multiple remote programs as in a multicast.
- A Transport Connection Layer, which sets up and manages the request.

Code:

ClientNew.java

```java
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.Scanner;

public class ClientNew {
    private ClientNew() {}
    Run | Debug
    public static void main(String[] args) {
        try {
            // Getting the registry
            Registry registry = LocateRegistry.getRegistry(null);

            // Looking up the registry for the remote object
            Rmi stub = (Rmi) registry.lookup("ServerA");
            Scanner myObj = new Scanner(System.in);  // Create a Scanner object
            System.out.println("Enter Car Name");
            // Calling the remote method using the obtained object
            String carName = myObj.nextLine();
            stub.putCars(carName,0);
            System.out.println(stub.printMsg());

        } catch (Exception e) {
            System.out.println("Client exception: " + e.toString());
            e.printStackTrace();
        }
    }
}
```

Implementation.java

```java
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.*;

// Creating Remote interface for our application
interface Rmi extends Remote {
    String printMsg() throws RemoteException;
    void putCars(String car, int flag) throws RemoteException;
    void replicatePut(String car, int flag) throws RemoteException;
}


public class ImplExample implements Rmi {
    static HashMap<String, Integer> map  = new HashMap<>();
    static int i = 0;
    // Implementing the interface method
    public String printMsg() {
        return map.toString();
    }
    public void replicatePut(String car,int flag){
        map.put(car,i);
        this.i+=1;
        System.out.println(map);
    }
    public void putCars(String car,int flag){
            map.put(car,i);
        this.i+=1;
        System.out.println(map);
        if (flag==0){
            try {
                // Getting the registry
```

ClientNew.java | ImplExample.java

ImplExample.java > ImplExample > putCars(String, int)

```java
                // Getting the registry
                Registry registry = LocateRegistry.getRegistry(null);
                // Looking up the registry for the remote object
                Rmi stub = (Rmi) registry.lookup("ServerB");
//                stub.putBooks(book,1);
                stub.replicatePut(car,1);
                System.out.println(stub.printMsg());
            } catch (Exception e) {
                System.out.println("Client exception: " + e.toString());
                e.printStackTrace();
            }
            try {
                // Getting the registry
                Registry registry = LocateRegistry.getRegistry(null);
                // Looking up the registry for the remote object
                Rmi stub = (Rmi) registry.lookup("ServerC");
//                stub.putBooks(book,2);
                stub.replicatePut(car,2);
                System.out.println(stub.printMsg());
            } catch (Exception e) {
                System.out.println("Client exception: " + e.toString());
                e.printStackTrace();
            }
        }
        else if (flag==1){
            try {
                // Getting the registry
                Registry registry = LocateRegistry.getRegistry(null);
                // Looking up the registry for the remote object
                Rmi stub = (Rmi) registry.lookup("ServerA");
                stub.replicatePut(car,0);
                System.out.println(stub.printMsg());
            } catch (Exception e) {
```

ClientNew.java | ImplExample.java

ImplExample.java > ImplExample > putCars(String, int)

```java
                stub.replicatePut(car,2);
                System.out.println(stub.printMsg());
            } catch (Exception e) {
                System.out.println("Client exception: " + e.toString());
                e.printStackTrace();
            }
        }
        else {
            try {
                // Getting the registry
                Registry registry = LocateRegistry.getRegistry(null);
                // Looking up the registry for the remote object
                Rmi stub = (Rmi) registry.lookup("ServerA");
                stub.replicatePut(car,0);
                System.out.println(stub.printMsg());
            } catch (Exception e) {
                System.out.println("Client exception: " + e.toString());
                e.printStackTrace();
            }
            try {
                // Getting the registry
                Registry registry = LocateRegistry.getRegistry(null);
                // Looking up the registry for the remote object
                Rmi stub = (Rmi) registry.lookup("ServerB");
                stub.replicatePut(car,1);
                System.out.println(stub.printMsg());
            } catch (Exception e) {
                System.out.println("Client exception: " + e.toString());
                e.printStackTrace();
            }
        }
```

ServerA.java

```java
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class ServerA extends ImplExample {
    public ServerA() {}
    Run | Debug
    public static void main(String args[]) {
        try {
            // Instantiating the implementation class
            ImplExample obj = new ImplExample();

            // Exporting the object of implementation class
            // (here we are exporting the remote object to the stub)
            Rmi stub = (Rmi) UnicastRemoteObject.exportObject(obj, 0);

            // Binding the remote object (stub) in the registry
            Registry registry = LocateRegistry.getRegistry();

            registry.bind("ServerA", stub);
            System.out.println("ServerA ready");

        } catch (Exception e) {
            System.out.println("Server exception: " + e.toString());
            e.printStackTrace();
        }
    }
}
```

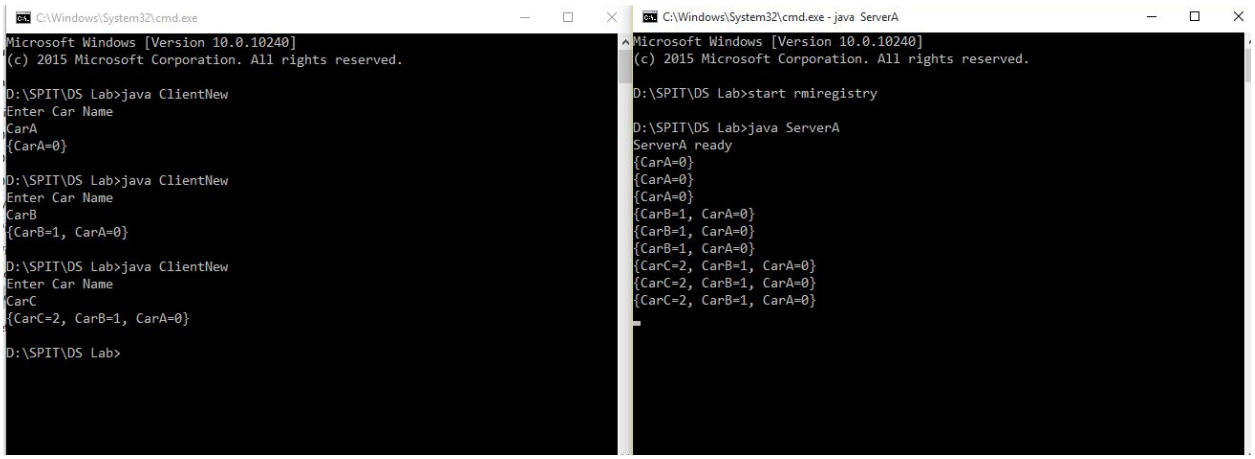
Explanation of Implementation:

We first created a remote interface which provides the description of all the methods of a particular remote object. The client communicates with this remote interface. This remote interface extends the predefined interface Remote which belongs to the package.
We then created another class and implemented the remote interface.
For the server program we create a remote object and bind it to the RMIregistry. Here ServerA acts as our server. We create a unicast remote object which we then bind with the registry.
In the client program, we look up for the registry created in the server program. We then invoke the required method using the obtained remote object.
In our implementation, ClientNew is the client side program and serverA is the server side program. Once, the connection is established, the server is ready. We are using a hashmap to store the details entered by the user. The same details are stored on the server as displayed in the screenshots below.Everytime a new car is added, the server is updated and the details are added in the hashmap created.

Steps to run:

1. Start rmiregistry
2. Compile the files : javac *.java
3. Run the command: java ServerA so that the server will be ready
4. In a new window, run the command: java ClientNew to run the client side program. The client can then add new cars.

Screenshots:



The client has added 3 cars i.e CarA, CarB and CarC. Each time a request is sent to the server and this data is stored in a hashmap in the server.

Conclusion:

Client server communication was implemented through remote method invocation. Every time the client made a request, the server made the required changes and updated depending on the request. In this manner, client server communication was implemented and integrated for our car booking system.