LAB RMI

Reference book:

[1] Uttam Kumar Roy. *Advanced Java Programming.* Published in India by Oxford University Press 2015. Chappter 14 Remote method Invocation.

External links:

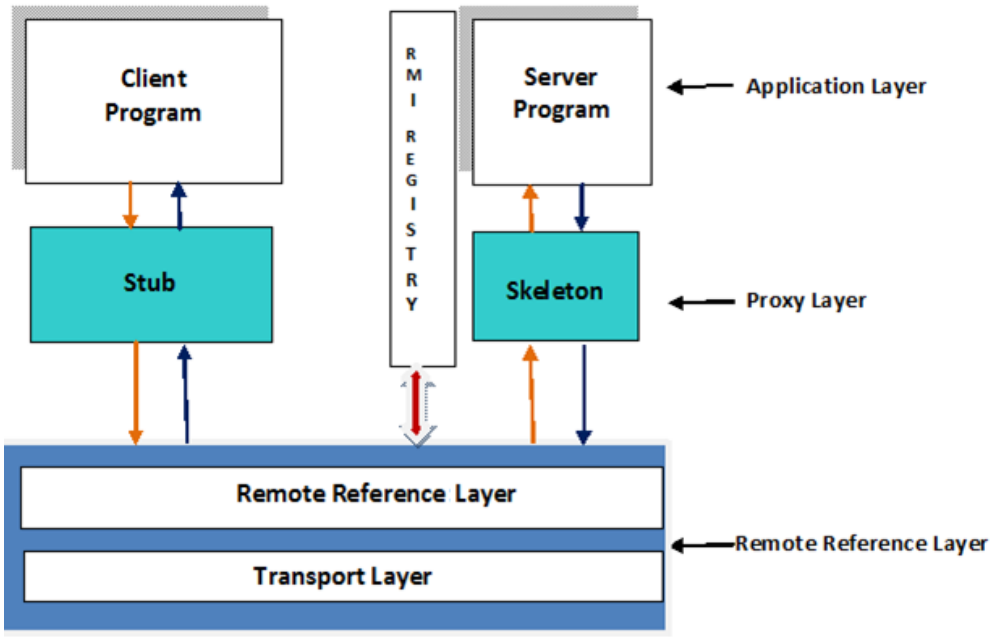
[2] RMI: https://docs.oracle.com/javase/tutorial/rmi/

[3] Remote Method Invocation (Level I): https://www.javacamp.org/moreclasses/rmi/rmi.html

[4] Remote Method Invocation (Level II): https://www.javacamp.org/moreclasses/rmi/rmi21.html

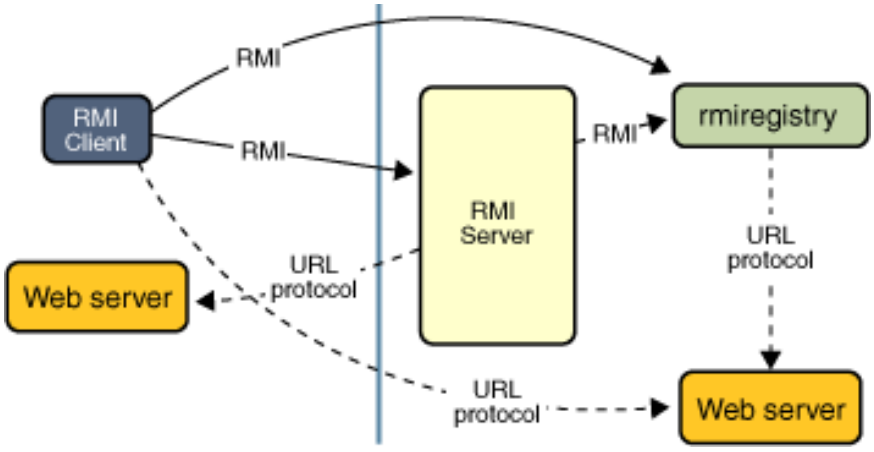
INTRODUCTION

RMI Architecture



RMI Applications

The following illustration depicts an RMI distributed application that uses the RMI registry to obtain a reference to a remote object. The server calls the registry to associate (or bind) a name with a remote object. The client looks up the remote object by its name in the server's registry and then invokes a method on it. The illustration also shows that the RMI system uses an existing web server to load class definitions, from server to client and from client to server, for objects when needed.



BÀI TẬP 1

Hướng dẫn từng bước với bài toán xây dựng calculator server

1. Đầu tiên ta tạo một service interface. Interface này nằm ở cả 2 phía client và server, đóng vai trò như sự đảm bảo dịch vụ sẽ cung cấp từ server cho phía client cũng như là định nghĩa dịch vụ phải cài đặt của phía client.

Interface Calculator

```
public interface Calculator extends java.rmi.Remote{
   int cong(int a, int b) throws java.rmi.RemoteException;
   int tru(int a, int b)throws java.rmi.RemoteException;
   int nhan(int a, int b)throws java.rmi.RemoteException;
   int chia(int a, int b)throws java.rmi.RemoteException;
}
```

2. Lớp implements cài đặt dịch vụ sẽ phục vụ cho client

Lớp CalculatorImpl

```
public class CalculatorImpl
      extends  java.rmi.server.UnicastRemoteObject
      implements Calculator{
   public CalculatorImpl() throws java.rmi.RemoteException{
   }
   public int cong(int a, int b) throws java.rmi.RemoteException{
      return a+b;
   }
   public int tru(int a, int b)throws java.rmi.RemoteException{
      return a-b;
   }
   public int nhan(int a, int b)throws java.rmi.RemoteException{
      return a*b;
   }
   public int chia(int a, int b)throws java.rmi.RemoteException{
      return a/b;
   }
}
```

3. Lớp Server

   Tạo một rmiregistry bind ở cổng 1099. Trong trường hợp hợp tổng quát thì ta phải start **rmiregistry** như là 1 server riêng bên ngoài và sau đó bind dịch vụ của ta vào đó

   Tạo một service instance và đăng ký với server

```
import javax.naming.*;
import java.rmi.registry.LocateRegistry;
public class CalculatorServer{
   public static void main(String []args)throws Exception{
```

```
        //create local registry instead of using rmiregistry program
        LocateRegistry.createRegistry(1099);
        System.out.println("rmiregistry started on port 1099");

        //create implemnt instant
        Calculator calc=new CalculatorImpl();
        //bin to server - use JNDI
        Context ctx=new InitialContext();
        ctx.bind("rmi://localhost:1099/teo",calc);
        //ctx.bind("rmi:met",calc);
        System.out.println("Service has bound to rmiregistry");
    }
}
```

4. Lớp client

Tạo lớp client dùng **JNDI** lookup dịch vụ trên rmiregistry sao đó triệu gọi từ xa

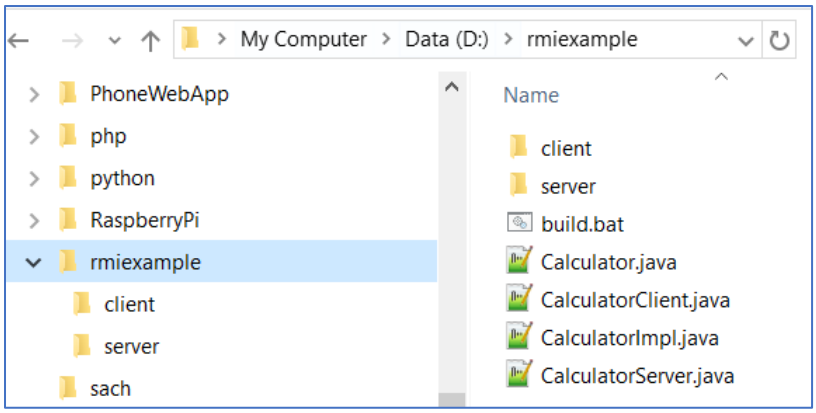
```
import javax.naming.*;
public class CalculatorClient{
    public static void main(String []args)throws Exception{
        String svr="localhost";
        if(args.length>0) svr=args[0];
         //lookup reference using JNDI
        Context ctx=new InitialContext();
        Object obj=ctx.lookup("rmi://"+svr+":1099/teo");
        Calculator calc=(Calculator)obj;

        //calling method
        int c1=calc.cong(3,6);
        int c2=calc.tru(3,6);
        int c3=calc.chia(3,6);

        //processing results
        System.out.println(c1);
        System.out.println(c2);
        System.out.println(c3);
    }
}
```

5. Thực thi ứng dụng

Giả sử ta có thư mục rmiexample lưu trữ các lớp trên



a. Ta tạo file build.bat để build các file java

*javac *.java*

pause

Thực thi file bat này và đảm bảo không xuất hiện lỗi

b. Tạo 2 thư mục server và client trong thư mục này

Copy các file class vào các thư mục tương ứng:

- Calculator.class vào cả 2 thư mục client và server
- CalculatorImpl.class và CalculatorServer.class vào thư mục server
- CalculatorClient.class vào thư mục client.

c. Trong thư mục server tạo file server.bat có nội dung sau để start server

java CalculatorServer

d. Trong thư mục client

- Tạo file client.policy có nội dung sau

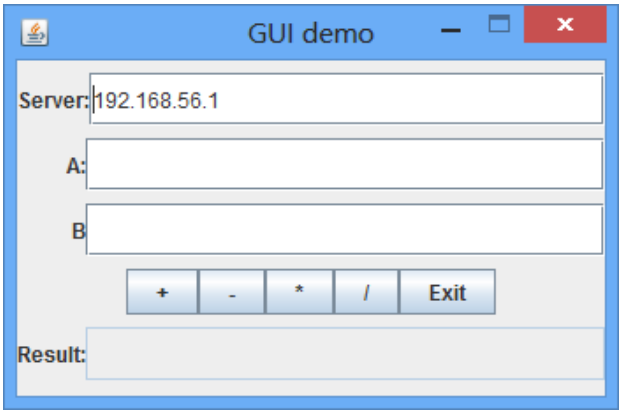
```
grant{
        permission java.net.SocketPermission
                        "*:1024-65535", "connect,listen,resolve,accept";
        //permission java.security.AllPermission;
};
```

- Tạo file client.bat có nội dung sau

java -Djava.security.policy=teo.policy CalculatorClient

pause

*** **Yêu cầu thêm:** Tạo client như sau sau đó kiểm tra kết nối trên máy khác để thực thi



BÀI TẬP 2

Cho mô tả sau:

Ngân hàng ABC cần lưu trữ thông tin của các tài khoản tiết kiệm gồm số tài khoản (gồm 11 chữ số), họ tên chủ sở hữu tài khoản và số tiền có trong tài khoản.

Số tài khoản là một CHUỖI số có mẫu được kiểm tra theo thuật toán Luhn.

Hãy viết một ứng dụng dùng kỹ thuật RMI xây dựng một server với các phương thức với yêu cầu và bản mẫu sau:

1. Phương thức kiểm tra số tài khoản có hợp lệ không.
   + *validateCardNumber(cardNumber: String): boolean*
2. Phương thức sinh một số tài khoản hợp lệ theo thuật toán Lahn
   + *generateCardNumber(): String*

Viết RMI client thử các tính năng này.

***Gợi ý:* Thuật toán Luhn - Cho một dãy số cần kiểm tra**

Bước 1: Từ phải sang, lấy các số cộng dồn với nhau. Tuy nhiên ở vị chẵn, nhân đôi số này, nếu kết quả nhân đôi lớn hơn 9 thì tiếp tục cộng dồn 2 số hàng chục và hàng đơn vị.

Bước 2: Kiểm tra nếu số tổng chia hết cho 10 thì số cần kiểm tra là hợp lệ.

Ví dụ: Cần kiểm tra số tài khoản **49927398716**

Vị trí	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*
Số tk	4	9	9	2	7	3	9	8	7	1	6
Double	4	18	9	4	7	6	9	16	7	2	6
Sum	4	**9**	9	**4**	7	**6**	9	**7**	7	**2**	6

Tổng của các số ở dòng Sum là 70. Vậy đây là một số tài khoản hợp lệ

- Đối với trường hợp tạo ngẫu nhiên một số tài khoản, ta để ý số cuối cùng của dãy số gọi là checkdigit, ta sinh ngẫu nhiên 10 chữ số còn lại, sau đó tính check-digit sao cho dãy số vừa sinh cộng với check-digit tạo ra một chuỗi số hợp lệ.

BÀI TẬP 3

Hướng dẫn từng bước với bài toán xây dựng calculator server với cơ chế Activation

Step 1: Writing the Remote Interface

- It must extend the marker Remote interface.
- All methods in a remote interface must throw a RemoteException or an exception, which is its superclass such as IOException or Exception.

```
package rmiserverside;

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

public interface ICalActivatable extends Remote{
    public int plus(int a, int b) throws RemoteException;
```

```
        public int minus(int a, int b) throws RemoteException;
    }
```

Step 2: Writing implementation class

- The heart of RMI object activation is java.rmi.activation.Activatable class that provides support for making remote objects activatable.
  ➔ *The implementation class of an activatable remote object usually extends this class.*
- Based on the RMI specification, we declared a two-argument constructor passing ActivationID to register the object with the activation system and a MarshalledObject.
  This constructor is required and should throw RemoteException.
  The super(id, 0) method calls Activatable constructor to pass an activation ID and a port number. In this case, the port number is default 1099.

```
package rmiserverside;

import java.rmi.MarshalledObject;
import java.rmi.RemoteException;
import java.rmi.activation.Activatable;
import java.rmi.activation.ActivationID;

@SuppressWarnings("serial")
public class CalActivatableImpl extends Activatable implements ICalActivatable{

        public CalActivatableImpl(ActivationID id, MarshalledObject<?> data) throws RemoteException {
                super(id, 0);
        }

        @Override
        public int plus(int a, int b) throws RemoteException {
                return b + a;
        }

        @Override
        public int minus(int a, int b) throws RemoteException {
                return a - b;
        }
}
```

Step 3: Writing server class

1) Install security manager for the ActivationGroup VM.
2) Set security policy
3) Create an instance of ActivationGroupDesc class
4) Register the instance and get an ActivationGroupID.

5) Create an instance of ActivationDesc.
6) Register the instance with rmid.
7) Bind or rebind the remote object instance with its name
8) Exit the system.

```
package rmiserverside;

import java.rmi.Remote;
import java.rmi.activation.Activatable;
import java.rmi.activation.ActivationDesc;
import java.rmi.activation.ActivationGroup;
import java.rmi.activation.ActivationGroupDesc;
import java.rmi.activation.ActivationGroupID;
import java.util.Properties;

import javax.naming.Context;
import javax.naming.InitialContext;

public class Server {
    public static void main(String[] args) throws Exception {

//              Install security manager for the ActivationGroup VM
                SecurityManager securityManager = System.getSecurityManager();
                if(securityManager == null)
                        System.setSecurityManager(new SecurityManager());

//              Set security policy
                Properties props = new Properties();
                props.setProperty("java.security.policy", "javapolicy.policy");

//              Create an instance of ActivationGroupDesc class
                ActivationGroupDesc.CommandEnvironment aec = null;
                ActivationGroupDesc groupDesc = new ActivationGroupDesc(props, aec);

//              Register the instance and get an ActivationGroupID.
                ActivationGroupID groupID = ActivationGroup.getSystem().registerGroup(groupDesc);

//              Create an instance of ActivationDesc.
                ActivationDesc desc = new ActivationDesc(groupID, CalActivatableImpl.class.getName(),
null, null);

//              Register the instance with rmid.
                Remote obj = Activatable.register(desc);
```

```
    //          Bind or rebind the remote object instance with its name
                Context ctx = new InitialContext();
                ctx.rebind("rmi://localhost:1099/myobj", obj);

                System.out.println("Exported!");

    //          Exit the system
                System.exit(0);
        }
    }
```

Step 4: Writing the RMI Client Program

```
package rmiclientside;
import java.net.MalformedURLException;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

import rmiserverside.ICalActivatable;

public class Client {
        public static void main(String[] args) throws MalformedURLException, RemoteException,
NotBoundException, NamingException {
                if(System.getSecurityManager()==null)       {
                        System.setSecurityManager(new SecurityManager());
                }
                Context ctx = new InitialContext();
                ICalActivatable obj = (ICalActivatable) ctx.lookup("rmi://localhost:1099/myobj");
                System.out.println(obj.plus(100, 500));
        }
}
```

Step 5: Compiling and running the program

1. Compile java classes

   Using ***javac*** tool

   javac -d . *.java

2. Start the rmiregistry

   Using command: ***start rmiregistry***

3. Start the activation daemon, rmid

   Using command: ***start rmid -J-Djava.security.policy=rmi.policy***

```
grant{
        permission com.sun.rmi.rmid.ExecPermission "${java.home}${/}bin${/}java";
        permission com.sun.rmi.rmid.ExecOptionPermission "-Djava.security.policy=*";
};
```

4. Run the server program

   Using tool: ***java -Djava.security.policy=javapolicy.policy*** **YourServer**

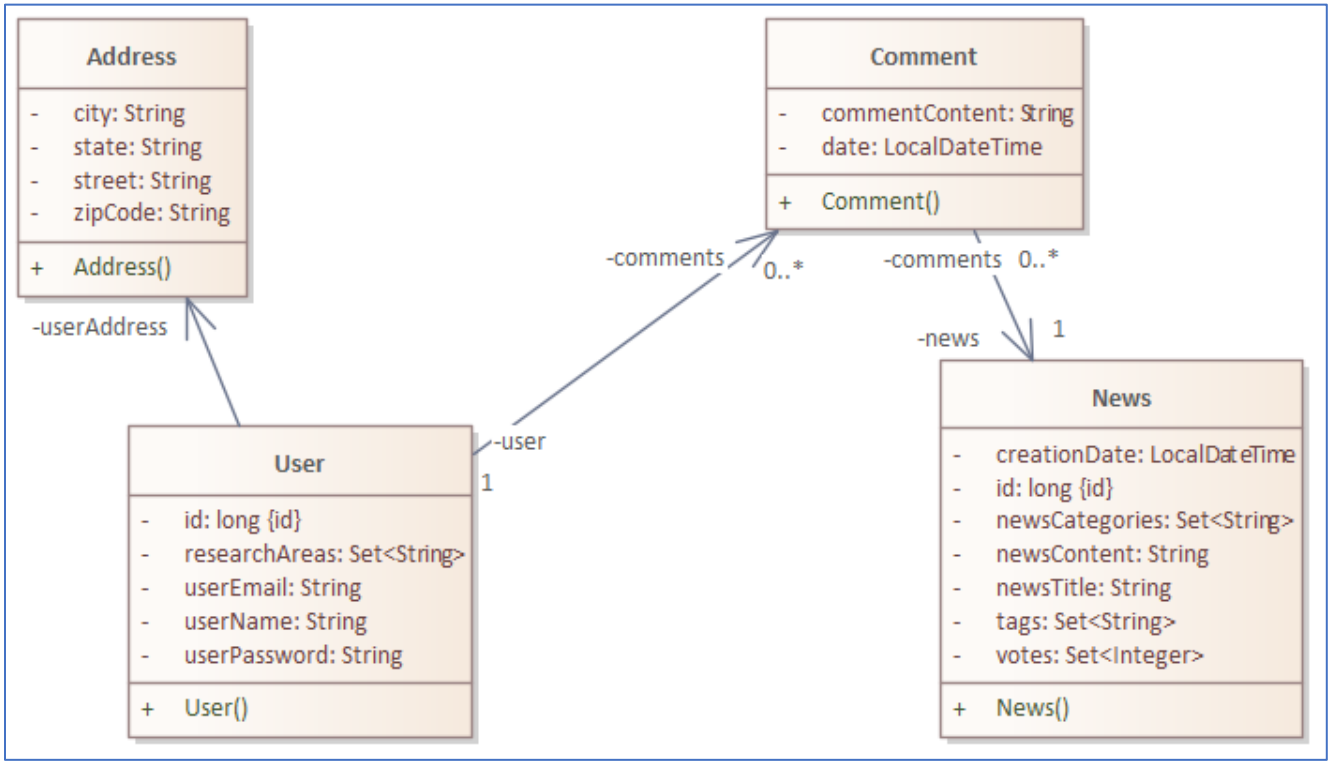
5. Run the client program

   Using tool: ***java -Djava.security.policy=javapolicy.policy YourClient***

BÀI TẬP 4

RMI và JPA

Cho mô hình lớp của một ứng dụng quản lý thông tin về tin tức và các bình luận cho tin tức của một website tin tức hàng ngày. Một người dùng *(User)* muốn đăng lời bình luận *(Comment)* về tin tức *(News)* cần có một tài khoản. Một người dùng có thể tham gia bình luận về nhiều tin tức, mỗi một tin tức sẽ có nhiều bình luận với nội dung vào thời gian khác nhau:



1. Thao tác với Java Persistence API (JPA).

   a. Tạo các lớp Entity phù hợp với mô hình lớp.

   b. Dùng ORM/OGM ánh xạ các lớp Entity vào MS SQL Server/MongoDB *(ORM nên xét 2 chiều)*

2. Dùng RMI xây dựng một Server với các chức năng cho phép thao tác với CSDL vừa tạo, thực hiện:

   a. Thêm dữ liệu người dùng (User) và lời bình luận (Comment) vào cơ sở dữ liệu NoSQL MongoDB hoặc cơ sở dữ liệu quan hệ SQL Server.

   + addUser(u: User): boolean

   b. Tìm các tin tức theo tags hoặc theo loại tin tức (tìm tương đối), yêu cầu dùng text search trên các cột tags và loại tin tức (newsCategories).

   + getNewsByTagsOrNewsCategories (value: String): List<News>

   c. Thống kê tổng số lượng các tin tức có các lời bình luận từ từng người dùng (chỉ thống kê những người dùng có ít nhất 3 lượt bình luận trở lên). Thông tin kết quả gồm thông tin của người dùng và số lượng bình luận cho tin tức.

   + getStatistics(): Map<User, Integer>

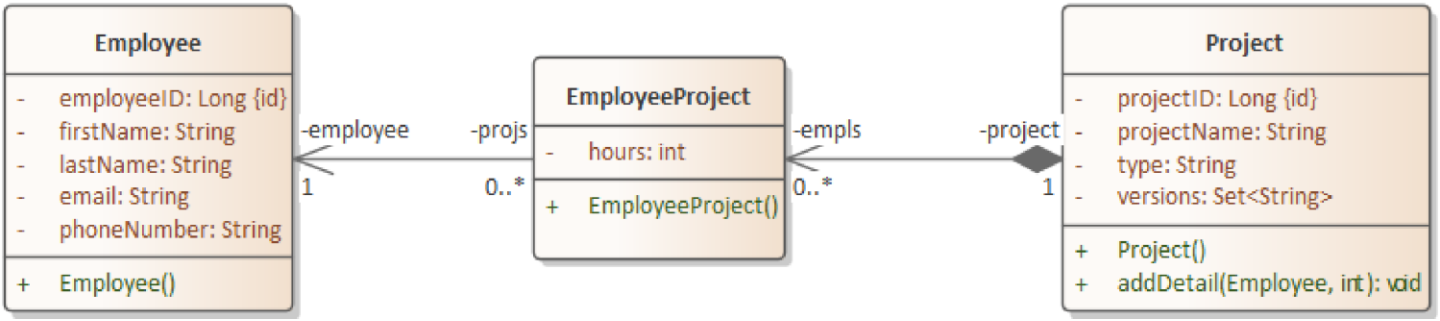
3. RMI Client để thực hiện các chức năng trong Câu 2a, 2b, 2c.
4. Thực hiện chương trình với cơ chế Activation.

BÀI TẬP 5

Xây dựng ứng dụng dựa trên mô hình 3-layers phân tán

Một nhân viên *(Employee)* tham gia nhiều dự án *(Project)*. Ngược lại, một dự án được nhiều nhân viên tham gia thực hiện. Mỗi nhân viên khi tham gia dự án sẽ lưu thông tin về số giờ *(hours)* làm.

Với mô hình lớp như sau:



I. Thao tác với Java Persistence API (JPA).

a) Tạo các lớp Entity phù hợp với mô hình lớp.
b) Dùng ORM/OGM ánh xạ các lớp Entity vào MS SQL Server/MongoDB (ORM nên xét 2 chiều)

II. Dùng RMI xây dựng một Server với các chức năng cho phép thao tác với CSDL vừa tạo, thực hiện:

1) Thêm một đối tượng Project và EmployeeProject vào MongoDB

   + addProject(prj: Project) : boolean

2) Tìm 1 dự án khi biết tên dự án (projectName).

   + getProject(name: String) : Project

3) Tìm tất cả các dự án có nhân viên nào đó thực hiện khi biết địa chỉ email.

   + getProjects(empl_email: String): List<Project>

4) Tìm các nhân viên khi biết tên (tìm tương đối): Yêu cầu dùng Text search trên cột firstName và lastName.

+ getEmployeeByName(aValue: String): List<Employee>

5) Lập bảng lương cho các nhân viên. Thông tin gồm thông tin của nhân viên, tổng số giờ nhân viên đó đã làm và tổng tiền lương. *Biết rằng, tiền lương 1 giờ là 6$.*

+ getStatistics(): Map<Employee, Object[]>

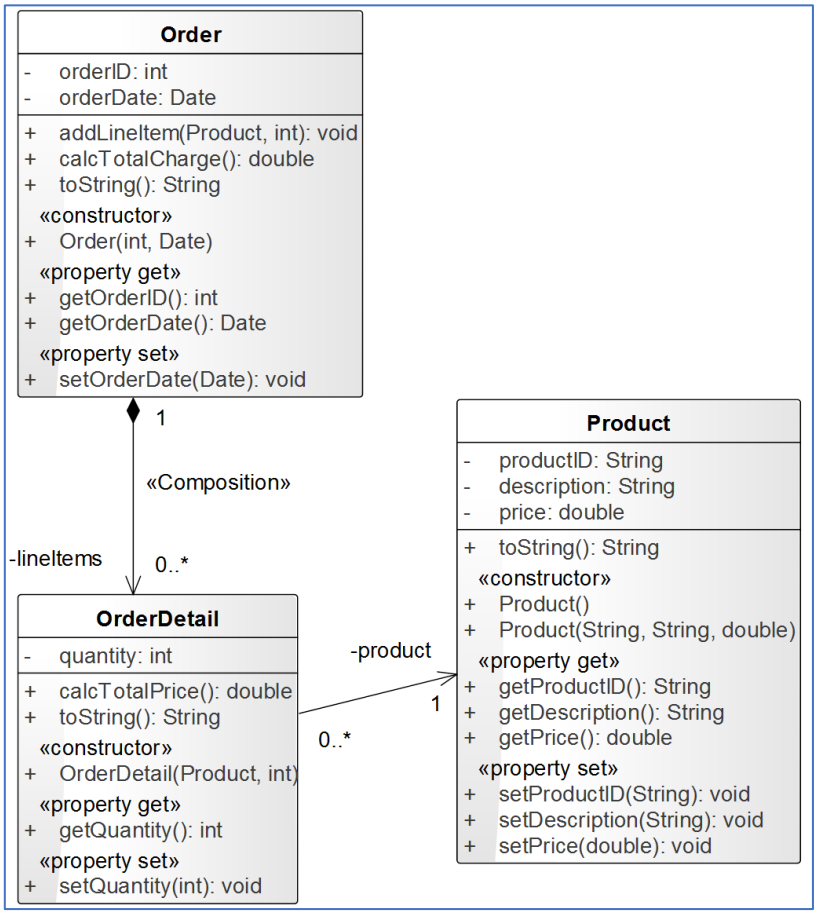
III/ Viết client để kiểm thử các chức năng *(kiểm thử chi tiết)*

IV/ Thực hiện chương trình với cơ chế Activation.

BÀI TẬP 6

Mục tiêu: Xây dựng ứng dụng dựa trên mô hình 3-tiers phân tán *(JPA OGM RMI).*

Yêu cầu: Xây dựng order-processing server cho phép các client tạo đơn đặt hàng từ xa. Tạo đơn hàng, tìm kiếm đơn hàng khi biết mã đơn hàng ….



Trong đó:

- totalPrice = quantity * price
  *(thành tiền = số lượng * đơn giá sản phẩm)*
- Tổng tiền hóa đơn *(totalCharge)* = $\sum_{count=0}^{n}$ thành tiền.
- Phương thức *addLineItem(Product p, int q) : void*, dùng để thêm một sản phẩm p với số lượng q vào hóa đơn.

Entities

Data Access

Business Logic

Control

Façade

Implement using RMI as façade

1. Service
2. Implementation
3. Server

Client

Triển khai và thực thi