

TÀI LIỆU THỰC TẬP LẬP TRÌNH MẠNG: LAB 12

DANH MỤC THUẬT NGỮ TIẾNG ANH

Từ	Nghĩa của từ
abstract	Trừu tượng
break	Dừng vòng lặp
catch	Từ khóa đầu của một khối bắt ngoại lệ
continue	Bỏ qua phần cuối vòng lặp, tiếp tục sang bước tiếp theo
default	Giá trị mặc định của phương thức switch()
extends	Kế thừa
final	Một hằng số, phương thức hay một lớp không được ghi đè
finally	Một phần của khối xử lý ngoại lệ try luôn được thực hiện
implements	Thực hiện giao diện
import	Khai báo một gói thư viện
instanceof	Kiểm tra một đối tượng là một thể hiện của lớp
interface	Giao diện
new	Tạo một đối tượng mới của lớp
null	Tham chiếu rỗng
package	Gói
private	Tiền tố chỉ được truy cập bởi phương thức của lớp
protected	Tiền tố được truy cập bởi phương thức của lớp, lớp con của và các lớp khác trong cùng một gói
public	Tiền tố có thể được truy cập bởi phương thức của tất cả các lớp
return	Trả về của một phương thức
super	Gọi phương thức của lớp cha
synchronized	Đồng bộ
this	Tham chiếu đến đối tượng hiện tại

DANH MỤC CHỮ VIẾT TẮT

Chữ viết tắt	Ý nghĩa
UDP	User Datagram Protocol
TCP	Transmission Control Protocol
IP	Internet Protocol
URL	Uniform Resource Locator
CSDL	Cơ Sở Dữ Liệu
JDBC	Java Database Connectivity
CNTT	Công Nghệ Thông Tin
HĐH	Hệ Điều Hành
MVC	Model-View-Control
DNS	Domain Name System
API	Application Programming Interface
FTP	File Transfer Protocol
JDK	Java Development Kit
GB	GigaByte
UCLN	Ước Chung Lớn Nhất
BCNN	Bội Chung Nhỏ Nhất
RAM	Random Access Memory
RMI	Remote Method Invocation
JVM	Java Virtual Machine
NIC	Network Interface Card
ĐH KTKT CN	Đại học Kinh tế Kỹ thuật Công nghiệp

LỜI NÓI ĐẦU

Ngày nay do nhu cầu thực tế và do sự phát triển mạnh mẽ của nhiều công nghệ tích hợp, dẫn đến các chương trình ứng dụng hầu hết đều có khả năng thực hiện trên môi trường mạng. Ngôn ngữ JAVA là ngôn ngữ phù hợp để viết các ứng dụng mạng. So với lập trình thông thường, lập trình mạng đòi hỏi người lập trình hiểu biết và có kỹ năng tốt để viết các chương trình giao tiếp và trao đổi dữ liệu giữa các máy tính với nhau.

Để hỗ trợ sinh viên chuyên ngành CNTT trong nhà trường tiếp cận với kỹ thuật lập trình mới này, tiếp theo cuốn tài liệu học tập lý thuyết “**Công nghệ JAVA**”, chúng tôi xây dựng cuốn “**Bài tập lập trình mạng**”, nhằm cung cấp cho sinh viên những kiến thức và kỹ thuật cơ bản nhất để phát triển các chương trình ứng dụng mạng, thông qua các dạng bài tập từ cơ bản đến nâng cao qua các chủ đề: lập trình cơ bản, lập trình hướng đối tượng, lập trình CSDL JDBC, lập trình mạng dùng socket, lập trình phân tán với RMI. Sinh viên sẽ thực hiện các bài thực hành này trên phòng máy nhà trường.

Nội dung cuốn tài liệu bao gồm 12 bài lab chia thành các chủ đề khác nhau. Trong mỗi chủ đề chúng tôi đưa ra tóm tắt lý thuyết, bài tập mẫu, sau đó là bài tập tương tự, và bài tập tổng hợp. Kết quả qua những bài lab, sinh viên được rèn và thành thạo các kỹ năng lập trình hướng đối tượng, lập trình CSDL, lập trình với giao thức truyền thông có sẵn và khả năng tích hợp trong các ứng dụng khác nhau, nhất là các giao thức truyền thông thời gian thực, từ đó sinh viên có thể viết được các phần mềm quản lý theo mô hình MVC, xây dựng được các ứng dụng mạng, các ứng dụng tích hợp và triệu gọi lẫn nhau trên mạng Intranet (mạng cục bộ), mạng Internet (mạng toàn cầu), các hệ thống xử lý truy xuất dữ liệu phân tán hoàn chỉnh. Nội dung biên soạn phù hợp với chuẩn đầu ra của ngành CNTT và ngành mạng máy tính và truyền thông dữ liệu về kỹ năng và kiến thức. Sau khi học xong học phần này sinh viên có thể viết phần mềm quản lý, truyền thông.

Chúng tôi xin chân thành cảm ơn Thầy Nguyễn Hoàng Chiến, phó chủ nhiệm khoa, phụ trách khoa CNTT trường ĐH KTKT CN cùng với các đồng nghiệp đã đóng góp ý kiến cho cuốn tài liệu này. Vì tài liệu được biên soạn lần đầu, chúng tôi đã cố gắng hoàn chỉnh, song không tránh khỏi thiếu sót. Rất mong nhận được sự góp ý của bạn đọc để tài liệu học tập được hoàn thiện hơn.

Xin trân trọng cảm ơn!

Nhóm tác giả

LAB 12. LẬP TRÌNH PHÂN TÁN VỚI RMI [5]

A. MỤC TIÊU

- Trang bị cho sinh viên kỹ thuật lập trình phân tán và công nghệ RMI
- Ứng dụng công nghệ RMI để xây dựng các ứng dụng phân tán
- Chuyển tham số cho phương thức triệu gọi từ xa và nhận kết quả trả về từ phương thức triệu gọi từ xa.

B. NỘI DUNG

- Xây dựng giao diện từ xa
- Triển khai giao diện từ xa
- Cài đặt và đăng ký đối tượng từ xa
- Cài đặt chương trình trên máy khách, triệu gọi phương thức của đối tượng cài đặt từ xa

C. KẾT QUẢ SAU KHI HOÀN THÀNH

Viết chương trình ứng dụng phân tán với RMI, áp dụng vào các bài toán cụ thể: Xây dựng ứng dụng tra cứu thông tin sách phân tán, mô phỏng bài toán nạp, rút tiền ngân hàng, phát triển hệ thống ngân hàng phân tán.

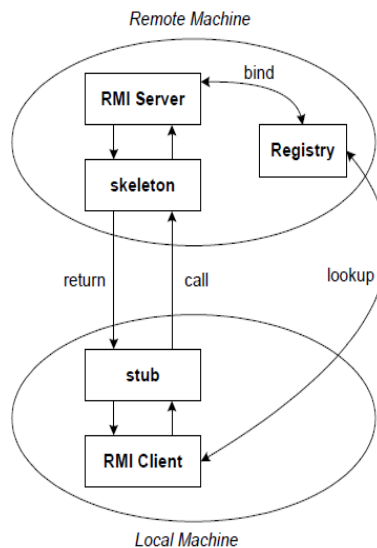
D. YÊU CẦU PHẦN CỨNG, PHẦN MỀM

- Máy tính cài HĐH windows, RAM tối thiểu 1GB.
- Phần mềm NETBEAN 8.0, JDK 1.8, SQL Server 2015.

E. HƯỚNG DẪN

1. Tạo giao diện từ xa (**Remote interface**) cần tuân theo các bước sau:

- Giao diện từ xa phải là một giao diện **public**
- Phải được kế thừa từ giao diện **Remote**
- Tất cả các phương thức trong giao diện từ xa đều bung ra ngoại lệ **RemoteException**
- Nếu tham số truyền cho phương thức hoặc giá trị nhận về từ phương thức triệu gọi từ xa là một đối tượng thì đối tượng đó phải **implements** giao diện **Remote** hoặc giao diện **Serializable**



Hình 15. Kiến trúc RMI

Tạo lớp cài đặt giao diện từ xa

```

public class class_name extends UnicastRemoteObject implements
interface_name {
    Phương thức khởi tạo
    public class_name() throws RemoteException {
    }
    Cài đặt xử lý cho tất cả các phương thức có trong giao diện
    public datatype|void method_name([parameter list]) throws RemoteException
    {
        Viết xử lý cho phương thức
    }
}

```

Viết xử lý phía Client

```

Tạo đối tượng Registry
Registry reg = LocateRegistry.getRegistry(Servername, portnumber);
//portnumber nằm trong phạm vi 1024-65535
Truy xuất remoteObject
interface_name remoteObject = (interface_name)reg.lookup(registername);
Gọi phương thức thông qua remoteObject
remoteObject.method_name([parameter list])

```

Viết xử lý phía Server

```

Tạo đối tượng Registry
Registry reg = LocateRegistry.createRegistry(portnumber);
Đăng ký Remote Object
class_name remoteObject = new class_name();
reg.bind(registername, remoteObject);

```

Bài 1. Xây dựng chương trình gửi số thực a từ máy Client lên máy chủ Server (sử dụng cổng 1102). Server tính giá trị bình phương của số thực a vừa nhận được rồi gửi về cho Client. Kết quả hiển thị trên máy Client.

Hướng dẫn:

Bước 1: Tạo giao diện Calculator

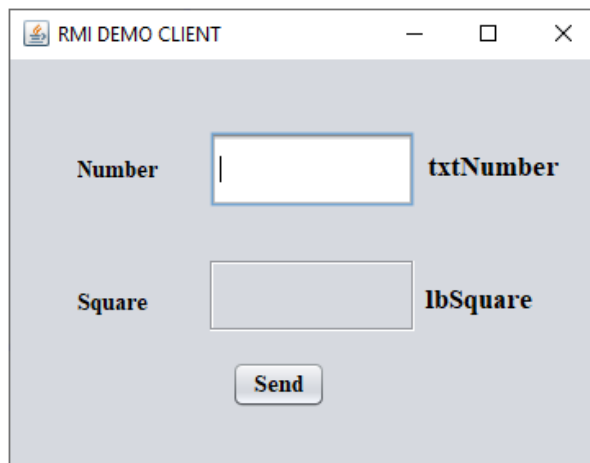
```
package rmi_demo;  
import java.rmi.Remote;  
import java.rmi.RemoteException;  
public interface Calculator extends Remote {  
    // Khai báo phương thức tính bình phương  
    public double square(double a) throws RemoteException;  
}
```

Bước 2: Tạo lớp cài đặt interface (Sq_calculator)

```
package rmi_demo;  
import java.rmi.RemoteException;  
import java.rmi.Server.UnicastRemoteObject;  
public class Sq_calculator extends UnicastRemoteObject implements  
Calculator{  
    // Khai báo phương thức khởi tạo  
    public Sq_calculator() throws RemoteException{  
    }  
    // Viết xử lý cho phương thức tính bình phương  
    public double square(double a) throws RemoteException {  
        return a*a;  
    }  
}
```

Bước 3: Xây dựng lớp xử lý ở phía Client. Tạo File RMI_Client

- Xây dựng JFrame Form:



- Viết code xử lý sự kiện khi người sử dụng bấm nút Send:

```
private void btSendActionPerformed(java.awt.event.ActionEvent evt) {
```

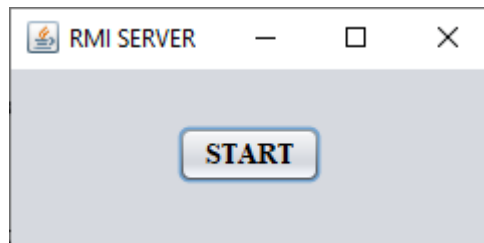
```

try {
    // Gọi Server đang lắng nghe tại cổng 1102
    Registry reg = LocateRegistry.getRegistry("localhost", 1102);
    // Lấy đối tượng từ xa
    Calculator cal = (Calculator) reg.lookup("RMICalSer");
    // Gọi phương thức từ xa
    double result = cal.square(Double.parseDouble(txtNum.getText()));
    // Hiển thị kết quả
    lblResult.setText(String.valueOf(result));
} catch (Exception e) {
    }
}

```

Bước 4. Viết chương trình xử lý phía Server:

- Xây dựng JFrame Form:



- Viết code xử lý khi người dùng nhấp chuột vào nút Start:

```

private void btStartActionPerformed(java.awt.event.ActionEvent evt) {
try {
    //Tạo đối tượng Registry
    Registry reg = LocateRegistry.createRegistry(1102);
    //Đăng ký Remote Object
    Sq_calculator ci = new Sq_calculator();
    reg.bind("RMICalSer", ci);
} catch (Exception e) {
    e.printStackTrace();
}
}

```

Bài 2.

Xây dựng chương trình tra cứu thông tin sách từ xa như sau: Client gửi thông tin mã isbn đến phía Server, Server tra cứu thông tin nếu có sách trùng mã isbn gửi đến thì hiện thông tin sách, nếu không có hiện thông báo không thấy. Ngoài ra phía Client thực hiện tra cứu thông tin sách sẽ hiện toàn thông tin các quyển sách ra màn hình.

Hướng dẫn:

Bước 1: Tạo interface RMInterface

```

public interface RMInterface extends Remote {

```

```
        Book findBook(Book b) throws RemoteException;
        ArrayList<Book> allBooks() throws RemoteException;
    }
```

Bước 2: Tạo Class Book

```
public class Book implements Serializable {
    private static final long serialVersionUID = 1190476516911661470L;
    private String title;
    private String isbn;
    private double cost;
    public Book(String isbn) {
        this.isbn = isbn;
    }
    public Book(String title, String isbn, double cost) {
        this.title = title;
        this.isbn = isbn;
        this.cost = cost;
    }
    public String getTitle() {
        return title;
    }
    public String getIsbn() {
        return isbn;
    }
    public double getCost() {
        return cost;
    }
    public String toString() {
        return "> " + this.title + " (" + this.cost + ")";
    }
}
```

Bước 3:

```
public class BookStore extends UnicastRemoteObject implements
RMIIInterface {
    private static final long serialVersionUID = 1L;
    private ArrayList<Book> bookList;
    protected BookStore(ArrayList<Book> list) throws RemoteException {
        super();
        this.bookList = list;
    }
}
```



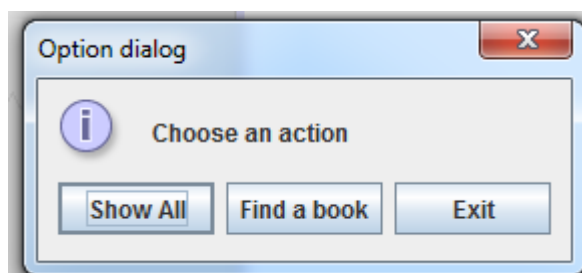
```

    }
    @Override
    public Book findBook(Book book) throws RemoteException {
        Predicate<Book> predicate = x -> x.getIsbn().equals(book.getIsbn());
        return
        bookList.stream().filter(predicate).findFirst().get();
    }
    @Override
    public ArrayList<Book> allBooks() throws RemoteException {
        return bookList;
    }
    private static ArrayList<Book> initializeList() {
        ArrayList<Book> list = new ArrayList<Book>();
        list.add(new Book("Head First Java, 2nd Edition", "978-0009205", 31.41));
        list.add(new Book("Java In A Nutshell", "978-059737", 10.90));
        list.add(new Book("Java:The Complete Reference","978-8082", 40.18));
        list.add(new Book("Head First Servlets and JSP", "978-16680", 35.41));
        list.add(new Book("Java Puzzlers", "978-0321336781", 39.99));
        return list;
    }

    public static void main(String[] args) {
        try {
            Registry reg=LocateRegistry.createRegistry(1122);
            BookStore bst = new BookStore(initializeList());
            reg.bind("abc", bst);
            System.err.println("Server ready");
        } catch (Exception e) {
            System.err.println("Server exception: " + e.getMessage());
        }
    }
}

```

Bước 4: Tạo class Custommer thực hiện phía Client. Giao diện phía Client gồm các chức năng sau:



```

public class Customer {
    private static RMIInterface rmi;
    public static void main(String[] args) throws
        MalformedURLException, RemoteException, NotBoundException
    {
        Registry reg = LocateRegistry.getRegistry("localhost", 1122);
        // Lấy đối tượng từ xa
        RMIInterface rmi = (RMIInterface) reg.lookup("abc");
        // Gọi phương thức từ xa
        boolean findmore;
        do {
            String[] options = {"Show All", "Find a book", "Exit"};
            int choice = JOptionPane.showOptionDialog(null, "Choose an
action", "Option dialog",JOptionPane.DEFAULT_OPTION,
                JOptionPane.INFORMATION_MESSAGE,
                null, options, options[0]);
            switch (choice) {
                case 0:
                    ArrayList<Book> list = rmi.allBooks();
                    StringBuilder message = new StringBuilder();
                    list.forEach(x -> {
                        message.append(x.toString() + "\n");
                    });
                    JOptionPane.showMessageDialog(null, new String(message));
                    break;
                case 1:
String isbn = JOptionPane.showInputDialog("Type the isbn of the book you
                    want to find.");
                    try {
                        Book response = rmi.findBook(new Book(isbn));
                        JOptionPane.showMessageDialog(null, "Title: " +response.getTitle() +
"\n" + "Cost: $" +response.getCost(),response.getIsbn(),
JOptionPane.INFORMATION_MESSAGE);
                    } catch (NoSuchElementException ex) {
                        JOptionPane.showMessageDialog(null, "Not found");
                    }
                    break;
                default:
                    System.exit(0);
            }
        } while (findmore);
    }
}

```

```

        break;
    }
    findmore = (JOptionPane.showConfirmDialog(null, "Do you want to
    exit?", "Exit", JOptionPane.YES_NO_OPTION) == JOptionPane.NO_OPTION);
    } while (findmore);
}
}

```

Bài 3. Dùng RMI mô phỏng giao dịch gửi, rút tiền trong ngân hàng

Bước 1. Tạo giao diện Account có các phương thức sau:

```

import java.rmi.Remote;
import java.rmi.RemoteException;
public interface Account extends Remote {
    public String getName() throws RemoteException;
    public float getBalance() throws RemoteException;
    public void withdraw(float amt) throws RemoteException;
    public void deposit(float amt) throws RemoteException;
    public void transfer(float amt, Account src) throws RemoteException;
}

```

Bước 2. Tạo lớp cài đặt interface trên (AccountImpl)

```

public class AccountImpl extends UnicastRemoteObject implements Account {
    private float balance = 0;
    private String name = "";
    // Constructor creates a new account with the given name
    public AccountImpl() throws RemoteException
    {

    }
    public AccountImpl(String aName) throws RemoteException {
        name = aName;
    }
    public String getName() throws RemoteException {
        return name;
    }
    public float getBalance() throws RemoteException {
        return balance;
    }
    //Withdraw some funds
    public void withdraw(float amt) throws RemoteException {
        balance -= amt;
    }
}

```

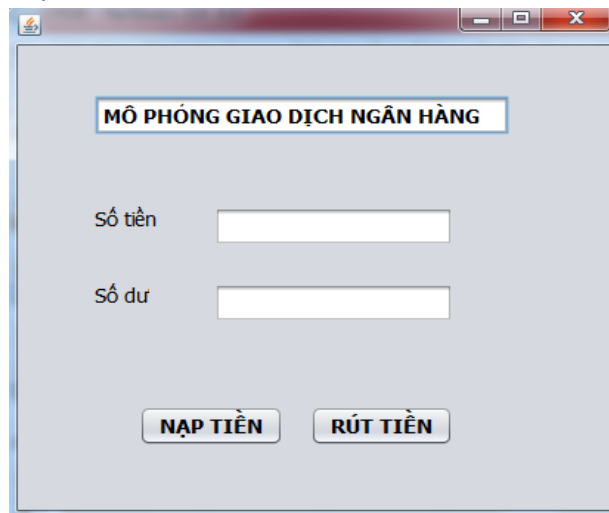
```

        //Ensure balance never drops below zero
        balance = Math.max(balance, 0);
    }
    //Deposit some funds
    public void deposit(float amt) throws RemoteException {
        balance += amt;
    }
    //Transfer some funds from another (remote) account Into this one
    public void transfer(float amt, Account src) throws RemoteException
    {
        src.withdraw(amt);
        this.deposit(amt);
    }
}

```

Bước 3. Xây dựng lớp xử lý ở phía Client. Tạo File RMI_Client

- Xây dựng JFrame Form:



- Viết code xử lý sự kiện khi người sử dụng bấm nút NẠP TIỀN

```

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        // Gọi Server đang lắng nghe tại cổng 1102
        Registry reg = LocateRegistry.getRegistry("localhost", 1102);
        // Lấy đối tượng từ xa
        Account ac = (Account) reg.lookup("JhonAdams");
        // Gọi phương thức từ xa
        float sotien=Float.parseFloat(jTextField1.getText());
        ac.deposit(sotien);
        float balance = ac.getBalance();
        jTextField2.setText(String.valueOf(balance));
    }
    catch (Exception e) {

```

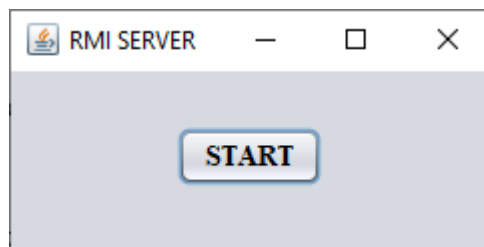
```
}  
}
```

Viết code xử lý sự kiện khi người sử dụng bấm nút RÚT TIỀN

```
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {  
    try {  
        Registry reg = LocateRegistry.getRegistry("localhost", 1102);  
        Account ac = (Account) reg.lookup("JhonAdams");  
        float sotien=Float.parseFloat(jTextField1.getText());  
        ac.withdraw(sotien);  
        float balance = ac.getBalance();  
        jTextField2.setText(String.valueOf(balance));  
    }  
    catch (Exception e) {  
    }  
}
```

Bước 4. Viết chương trình xử lý phía Server:

- Xây dựng JFrame Form:



- Viết code xử lý khi người dùng nhấp chuột vào nút Start:

```
private void btStartActionPerformed(java.awt.event.ActionEvent evt) {  
    try {  
        Registry reg = LocateRegistry.createRegistry(1102);  
        AccountImpl acct = new AccountImpl();  
        reg.bind("JhonAdams ", acct);  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

Bài 4. Kết hợp cơ sở dữ liệu và lập trình RMI, xây dựng chương trình Login từ xa

Bài toán login từ xa dùng RMI đặt ra như sau:

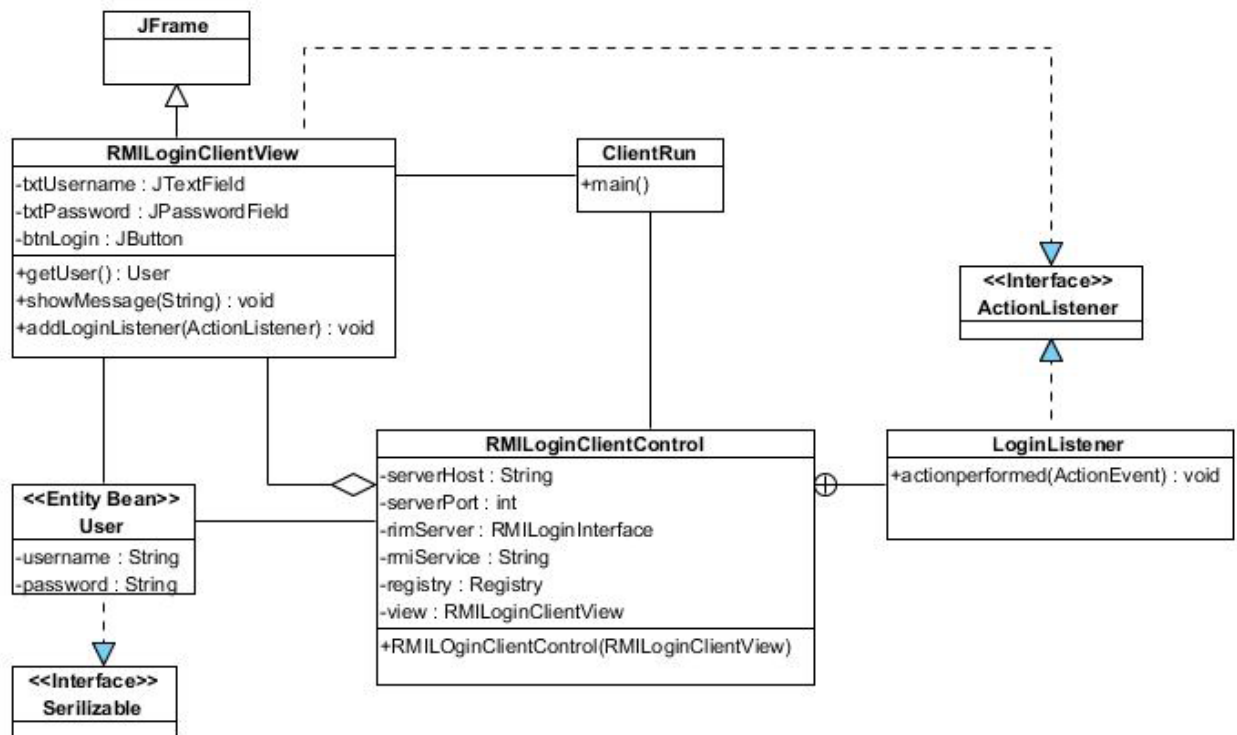
- CSDL được lưu trữ và quản lý trên Server RMI, trong đó có bảng users chứa ít nhất hai cột: cột username và cột password.
- Tại phía Server, có khai báo, định nghĩa, và đăng kí một đối tượng từ xa có phương thức kiểm tra đăng nhập, nó sẽ tiến hành kiểm tra trong CSDL xem có tài khoản nào trùng với thông tin đăng nhập nhận được hay không.

- Chương trình phía Client phải hiện giao diện đồ họa, trong đó có một ô text để nhập username, một ô text để nhập password, và một nút nhấn Login.
- Khi nút Login được click, chương trình Client sẽ triệu gọi làm kiểm tra login từ Server RMI, lấy thông tin đăng nhập (username/password) trên form giao diện để kiểm tra. Sau khi có kết quả kiểm tra (đăng nhập đúng, hoặc sai), Client sẽ hiển thị thông báo tương ứng với kết quả nhận được: nếu đăng nhập đúng thì thông báo login thành công. Nếu đăng nhập sai thì thông báo là username/password không đúng.
- Yêu cầu kiến trúc hệ thống ở cả hai phía Client và Server RMI đều được thiết kế theo mô hình MVC.

Gợi ý:

Sơ đồ lớp của phía client được thiết kế theo mô hình MVC bao gồm 3 lớp chính tương ứng với sơ đồ M-V-C như sau:

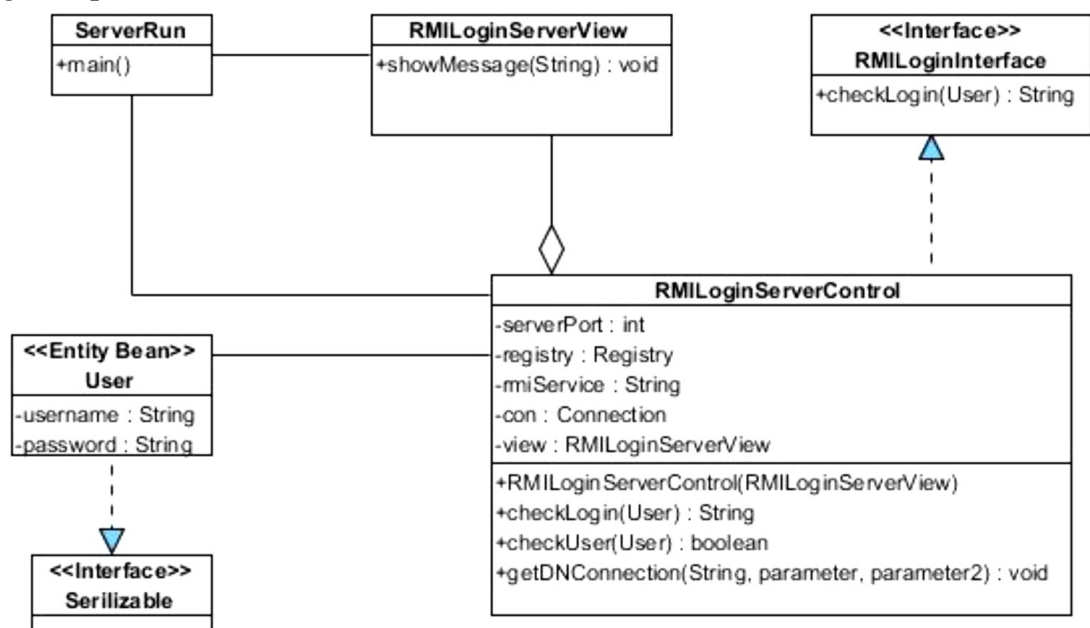
- Lớp User: là lớp tương ứng với thành phần model (M), bao gồm hai thuộc tính username và password, các hàm khởi tạo và các cặp getter/setter tương ứng với các thuộc tính.
- Lớp RMILoginClientView: Lớp tương ứng với thành phần view (V), là lớp form nên kế thừa từ lớp JFrame của Java, nó chứa các thuộc tính là các thành phần đồ họa bao gồm ô text nhập username, ô text nhập password, nút nhấn Login.
- Lớp RMILoginClientControl: Lớp tương ứng với thành phần control (C), nó chứa một lớp nội tại là LoginListener. Khi nút Login trên tầng view bị click thì sẽ chuyển tiếp sự kiện xuống lớp nội tại này để xử lý. Tất cả các xử lý đều gọi từ trong phương thức actionPerformed của lớp nội tại này, bao gồm: lấy thông tin trên form giao diện, triệu gọi thủ tục từ xa RMI về kiểm tra đăng nhập và yêu cầu form giao diện hiển thị.



Sơ đồ lớp phía RMI Client

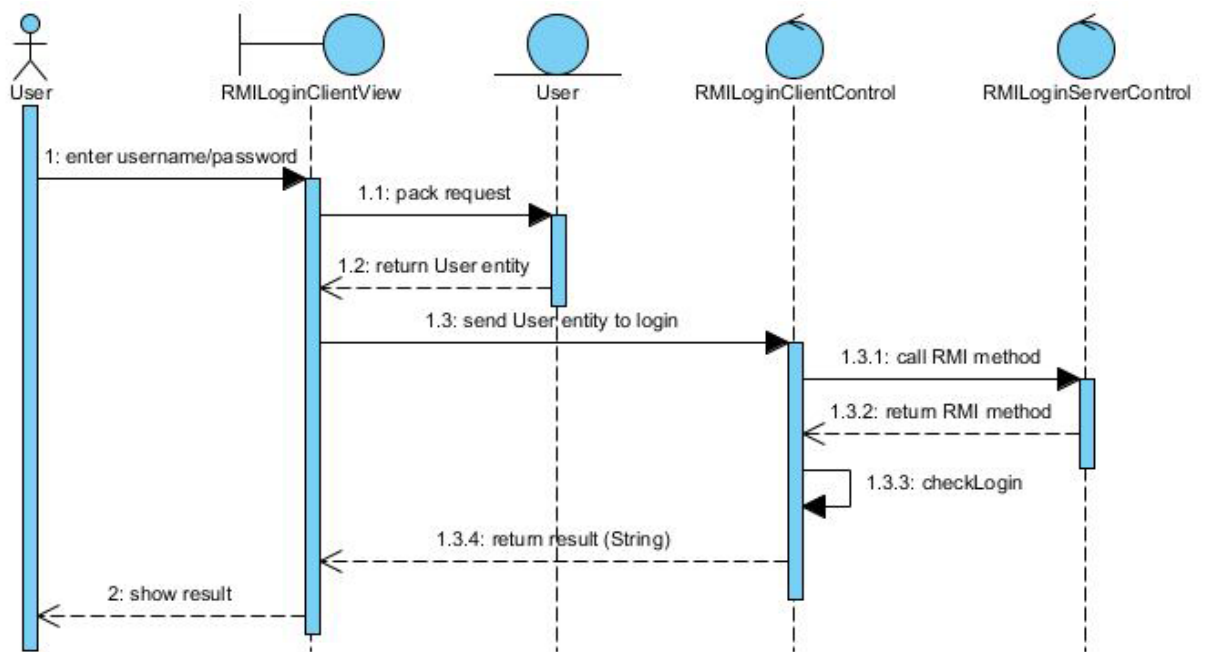
Sơ đồ lớp của phía server được thiết kế theo mô hình MVC bao gồm 3 lớp chính tương ứng với sơ đồ M-V-C như sau:

- Lớp User: là lớp thực thể, dùng chung thống nhất với lớp phía bên client.
- Lớp RMILoginServerView: Tương ứng với thành phần view (V), là lớp dùng hiển thị các thông báo và trạng thái hoạt động bên server RMI.
- Giao diện RMILoginInterface: Là giao diện (interface) khai báo đối tượng từ xa, trong đó nó khai báo thủ tục checkLogin(): Thủ tục nhận vào một tham số kiểu User, trả kết quả về dạng String.
- Lớp RMILoginServerControl: Tương ứng với thành phần control (C), nó đảm nhiệm vai trò xử lý của server RMI, trong đó định nghĩa cụ thể lại phương thức đã được khai báo trong RMILoginInterface, sau đó đăng kí bản thân nó vào server RMI để phục vụ các lời triệu gọi từ phía các client.



Sơ đồ lớp phía Server

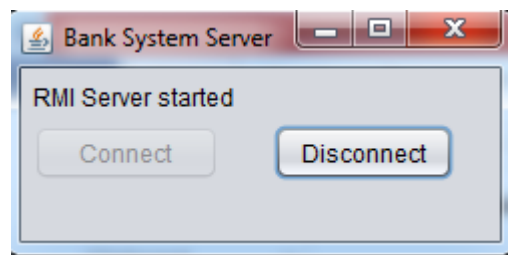
Tuần tự các bước thực hiện



Tuần tự các bước xử lý như sau

1. Ở phía client, người dùng nhập username/password và click vào giao diện của lớp RMILoginClientView
2. Lớp RMILoginClientView sẽ đóng gói thông tin username/password trên form vào một đối tượng model User bằng phương thức getUser() và chuyển xuống cho lớp RMILoginClientControl xử lý.
3. Lớp RMILoginClientControl sẽ triệu gọi làm checkLogin() từ phía server RMI
4. Server trả về cho bên client một skeleton của phương thức checkLogin().
5. Bên phía client, khi nhận được skeleton, nó gọi phương thức checkLogin() để kiểm tra thông tin đăng nhập.
6. Kết quả kiểm tra sẽ được lớp RMILoginClientControl sẽ chuyển cho lớp RMILoginClientView hiển thị bằng phương thức showMessage()
7. Lớp RMILoginClientView hiển thị kết quả đăng nhập lên cho người dùng. Sinh viên vận dụng các kiến thức đã học về lập trình RMI và kết nối CSDL tiến hành viết các lớp tương ứng.

Bài 5. Tổng hợp. Kết hợp kiến thức về RMI, database. Xây dựng hệ thống giao dịch ngân hàng phân tán.



Phía Server Bấm button Connect: Server kết nối với RMI.

Phía người dùng thực hiện những chức năng sau:

The account was credited
as follows: An Nguyễn
Balance: 78000.0

Customer ID

Account ID

Account ID to deposit

Amount to deposit

Account ID to withdraw

Amount to withdraw

Bài toán phát triển hệ thống ngân hàng phân tán đặt ra như sau:

- Cơ sở dữ liệu được lưu trữ và quản lý trên Server, trong đó có 3 bảng:
Bảng Account lưu thông tin tài khoản gồm hai trường IDaccount và Balance

Column Name	Data Type	Allow Nulls
IDAccount	int	<input type="checkbox"/>
Balance	float	<input type="checkbox"/>

Bảng AccountCustomer lưu thông tin tài khoản khách hàng gồm 2 trường idAccount và IdCustomer

Column Name	Data Type	Allow Nulls
iDAccount	int	<input type="checkbox"/>
IdCustomer	int	<input type="checkbox"/>

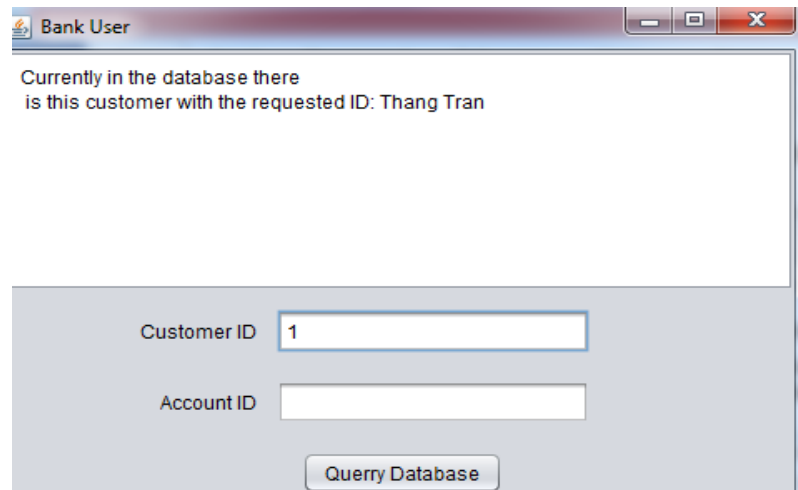
Bảng Customer lưu thông tin như sau

Column Name	Data Type	Allow Nulls
idCustomer	int	<input type="checkbox"/>
Name	varchar(50)	<input checked="" type="checkbox"/>
SurName	varchar(50)	<input checked="" type="checkbox"/>

- Chương trình phía Client hiện giao diện đồ họa, trong đó có ô text để nhập

CustomerID và AccountID, và một nút QueryDatabase.

- Khi nút QueryDatabase được click, chương trình Client sẽ gửi thông tin đăng nhập trên form giao diện, và gửi sang Server
- Tại phía Server, mỗi khi nhận được thông tin gửi từ Client, sẽ tiến hành kiểm tra trong CSDL xem có tài khoản nào trùng với thông tin nhận được hay không.
- Phía Client, sau khi nhận được kết quả từ Server, sẽ hiển thị thông báo tương ứng



Sau đó người dùng thực hiện các chức năng gửi tiền và rút tiền từ xa.

Hướng dẫn:

Bước 1: Tạo Interface Account

```
public interface Account extends Remote
{
    //Trả về ngân hàng quản lý tài khoản này
    public BankManager getBankManager() throws RemoteException;
    //Trả về Client của tài khoản này
    public Client getClient() throws RemoteException;
    //Lấy số dư tài khoản
    public float getBalance() throws RemoteException;
    //Chỉnh sửa số dư tài khoản
    public void setBalance(float bal) throws RemoteException;
    //phương thức rút tiền có kiểm tra số dư trước khi rút. Trả về amount
    public long getCash (long amount) throws NoCashAvailableException,
    RemoteException;
    //gửi tiền
    public void deposit(float amount) throws RemoteException;
    //rút tiền
    public void withdraw(float amount) throws RemoteException;
}
```

Bước 2: Tạo class AccountImpl thực hiện interface Account

```

public class AccountImpl implements Account, Serializable
{
    private BankManager bankManager;

    private Client Client;

    private float balance;

    private String accountNumber;

    public AccountImpl (BankManager bankManager, Client Client, String
accountNumber, float bal)
    {
        this.bankManager = bankManager;
        this.Client = Client;
        this.balance = bal;
        this.accountNumber = accountNumber;
    }

    public void deposit(float amount)
    {
        balance += amount;
    }

    public void withdraw(float amount)
    {
        balance -= amount;
    }

    public BankManager getBankManager() throws RemoteException
    {
        return bankManager;
    }

    public Client getClient() throws RemoteException
    {
        return Client;
    }
}

```

```

public float getBalance() throws RemoteException
{
    return balance;
}

public void setBalance(float bal) throws RemoteException
{
    balance = bal;
}

public long getCash(long amount) throws NoCashAvailableException,
RemoteException
{
    if (amount > balance)
    {
        throw new NoCashAvailableException();
    }
    balance = balance - amount;
    return amount;
}
}

```

Bước 3: Tạo interface BankManager

```

public interface BankManager extends Remote
{
    //Phương thức lấy tài khoản theo mã
    public Account getAccount(String accountNumber) throws RemoteException;
    //Phương thức lấy Client theo tên
    public Client getClient(String ClientName) throws RemoteException;
    //phương thức lấy mã khách hàng theo mã tài khoản
    public int getCustomerId(int accountId) throws RemoteException;
    //phương thức gửi tiền vào tài khoản
    public void deposit(String idAccount, float Amount) throws
RemoteException;
    //phương thức rút tiền
    public void withdraw(String idAccount, float Amount) throws
RemoteException;
}

```

```
}
```

Bước 4: Tạo class BankManagerImpl thực hiện interface BankManager

```
public class BankManagerImpl extends UnicastRemoteObject implements
BankManager
{
    private Hashtable accounts;
    private Hashtable Clients;
    private Connection conn;
    private Statement s;
    private int CustomerID;
    public BankManagerImpl() throws RemoteException
    {
        super();
        initialize();
    }
    public Account getAccount(String accountNumber) throws RemoteException
    {
        AccountImpl account = (AccountImpl)accounts.get(accountNumber);
        return account;
    }
    public Client getClient(String ClientID) throws RemoteException
    {
        ClientImpl Client = (ClientImpl)Clients.get(ClientID);
        return Client;
        //Phương thức rút tiền
    }
    public void deposit(String idAccount, float amount)throws RemoteException
    {
        Account theAccount = (Account)accounts.get(idAccount);
        theAccount.deposit(amount);
        accounts.remove(idAccount);
        accounts.put(idAccount,theAccount);
        try
        {
            Statement s = conn.createStatement();
            String sql = "Update account Set Balance =' " +
theAccount.getBalance() +"' where idAccount = '" + idAccount + "'";
            s.executeUpdate(sql);
            /// update in the dataabase now
        }
        catch(Exception e)
```

```

        {
            e.printStackTrace();
        }
    }

    public void withdraw(String idAccount, float amount) throws
RemoteException
    {
        Account theAccount = (Account)accounts.get(idAccount);
        theAccount.withdraw(amount);
        accounts.remove(idAccount);
        accounts.put(idAccount,theAccount);
        try
        {
            Statement s = conn.createStatement();
            String sql = "Update account Set Balance =' " +
theAccount.getBalance() +"' where idAccount = '" + idAccount + "'";
            s.executeUpdate(sql);
            /// update in the dataabase now
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }

    public void getClientsFromDatabase()
    {
        public void initialize() throws java.rmi.RemoteException
        {
            // Create the hashtables
            accounts = new Hashtable(20);
            Clients = new Hashtable(10);
            CreateConnection();
            getCustomersFromDatabase();
            getAccountsFromDatabase();
        }

        public boolean initializeConnection(String SERVER, String
DATABASE, String USER_ID, String PASSWORD) throws ClassNotFoundException,
SQLException
        {
            try

```

```

        {
            String connString = "jdbc:sqlserver://" + SERVER +
":1433;integratedSecurity=true;databaseName=" + DATABASE;
            conn = DriverManager.getConnection(connString);
            s = conn.createStatement();
            return true;
        }
        catch (SQLException e)
        {
            return false;
        }
        catch (Exception e)
        {
            e.printStackTrace();
            return false;
        }
    }
}

public void CreateConnection()
{
    if(conn == null)
        try
        {
            initializeConnection("localhost", "QLNH", "root", "");
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }

public int getCustomerId(int idAccount)
    {
        ArrayList<Integer> ids = new ArrayList<Integer>();
        try
        {
            Statement s = conn.createStatement();
            String sql = "Select IdCustomer from accountCustomer where
idAccount='" + idAccount + "'";
            ResultSet r = s.executeQuery(sql);
            while(r.next())
            {
                ids.add(r.getInt("IdCustomer"));
            }
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}

```

```

        }
    }
    catch (Exception ex)
    {
        ex.printStackTrace();
    }
    return ids.get(0).intValue();
}

public void getCustomersFromDatabase()
{
    try
    {
        Statement s = conn.createStatement();
        String sql = "Select * from customer";
        ResultSet r = s.executeQuery(sql);
        while(r.next())
        {
            int idCustomer = r.getInt("IdCustomer");
            String name = r.getString("Name");
            String surname = r.getString("Surname");
            Client newClient = new ClientImpl(this, name + " " + surname);
            Clients.put(String.valueOf(idCustomer), newClient);
        }
    }
    catch (Exception ex)
    {
        ex.printStackTrace();
    }
}

public void getAccountsFromDatabase()
{
    System.out.println("-----");
    System.out.println("Reading accounts from the database:");
    try
    {
        int counter = 0;
        Statement s = conn.createStatement();
        Statement s1 = conn.createStatement();
        String sql = "Select * from accountcustomer";
        ResultSet r = s.executeQuery(sql);
    }
}

```

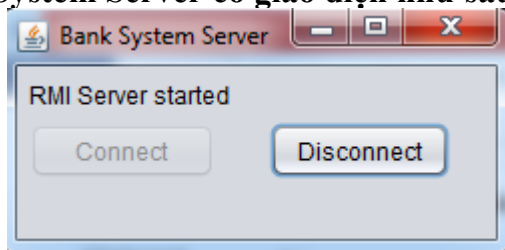


```

        while(r.next())
        {
            int idCustomer = r.getInt("IdCustomer");
            int idAccount = r.getInt("idAccount");
            Client theClient =(ClientImpl)Clients.get(String.valueOf(idCustomer));
            Account newAccount = new
            AccountImpl(this,theClient,String.valueOf(idAccount),0);
            accounts.put(String.valueOf(idAccount), newAccount);
            System.out.println("Customer:" + newAccount.getClient().getName() + "
- Account:" + String.valueOf(idAccount));
            counter++;
        }
        for(int i=1;i<=counter;i++)
        {
            if(accounts.containsKey(String.valueOf(i)))
            {
                sql = "Select Balance from account where idAccount = '" + i + "'";
                ResultSet r1 = s1.executeQuery(sql);
                r1.next();
                float balance = r1.getFloat("Balance");
                Account theAccount = (Account)accounts.get(String.valueOf(i));
                theAccount.setBalance(balance);
                accounts.remove(String.valueOf(i));
                accounts.put(String.valueOf(i),theAccount);
            }
        }
        s.close();
    }
    catch (Exception ex)
    {
        ex.printStackTrace();
    }
}

```

Bước 5: Tạo Form BankSystem Server có giao diện như sau



Thực hiện code cho button Connect và Disconnect

Phương thức ConnectServer

```
private void ConnectServer(  
{.....  
    m_bankManager = new BankManagerImpl();  
        if(m_Registry == null)  
            m_Registry = LocateRegistry.createRegistry(1234);  
            m_Registry.rebind("BankSystem", m_bankManager);  
            m_connected = true;  
        .....  
    }
```

Phương thức DisconnectServer()

```
private void DisconnectServer()  
{  
    m_bankManager = null;  
    m_Registry.unbind("BankSystem");  
    m_connected = false;  
}
```

Về phía người dùng tạo interface Client

```
public interface Client extends Remote  
{  
    //Phương thức trả về ngân hàng quản lý Client này  
    public BankManager getBankManager() throws RemoteException;  
    //Phương thức trả về tên của Client  
    public String getName() throws RemoteException;  
}
```

Tạo ClientImpl thực hiện interface Client

```

public class ClientImpl implements Client, Serializable
{
    private BankManager bankManager;
    private String ClientName;
    public ClientImpl(BankManager bm, String name)
    {
        this.bankManager = bm;
        this.ClientName = name;
    }
    public BankManager getBankManager() throws RemoteException
    {
        return bankManager;
    }
    public String getName() throws RemoteException
    {
        return ClientName;
    }
}

```

Tạo JFrame BankUser có giao diện như yêu cầu đề bài:

```

public class BankUser extends javax.swing.JFrame {
    public BankUser() {
        initComponents();
    }
    // Code chức năng mở cửa sổ
    private void formWindowOpened(java.awt.event.WindowEvent evt) {
        try
        {
            m_Registry = LocateRegistry.getRegistry("localhost" ,1234);
            m_bankManager = (BankManager) m_Registry.lookup("BankSystem");
        }
        catch (NotBoundException notBoundException)
        {
            System.out.println("Not Bound: " + notBoundException);
        }
        catch (RemoteException remoteException)
        {
            System.out.println("Remote Exception: " + remoteException);
        }
    }
}

```

Code cho button QueryDatabase

```

private void btnQrDatabaseActionPerformed(java.awt.event.ActionEvent evt)
{
    try
    {
        if(txtCustomerID.getText().length() > 0)
        {
            System.out.println("oops: " + txtCustomerID.getText() + "000");
            Client Client = m_bankManager.getClient(txtCustomerID.getText());
            txtCustomerID.setText("");
            System.out.println("Currently in the database there \n is this
customer with the requested ID: " + Client.getName());
            txa_printArea.setText("Currently in the database there \n is this
customer with the requested ID: " + Client.getName());
        }
        else
        {
            if(txtAccountID.getText().length() > 0)
            {
                Account account = m_bankManager.getAccount(txtAccountID.getText());
                txtAccountID.setText("");
                System.out.println("Currently in the database there is \n this
account with the requested ID: " + account.getClient().getName());
                txa_printArea.setText("Currently in the database there is \n this
account with the requested ID: " + account.getClient().getName() + "\n
Balance: " + account.getBalance());
            }
        }
        catch (RemoteException remoteException)
        {
        }
    }
}

```

Code cho button Deposit

```

private void btnDepositActionPerformed(java.awt.event.ActionEvent evt) {
    try
    {
        if(txtAccIDtoDeposit.getText().length() > 0 &&
txtAmountToDeposit.getText().length() > 0)
        {
            m_bankManager.deposit(txtAccIDtoDeposit.getText(),
Float.parseFloat(txtAmountToDeposit.getText()));
            Account account =
m_bankManager.getAccount(txtAccIDtoDeposit.getText());
            txa_printArea.setText("The account was credited \n as follows:" +
account.getClient().getName() + "\nBalance: " + account.getBalance());
        }
    }
    catch (RemoteException remoteException)
    {
        System.err.println(remoteException);
    }
}

```

Code cho button Withdraw

```

private void btnWithdrawActionPerformed(java.awt.event.ActionEvent evt) {
    try
    {
        if(txtAccIDtoWithdraw.getText().length() > 0 &&
txtAmountToWithdraw.getText().length() > 0)
        {
            m_bankManager.withdraw(txtAccIDtoWithdraw.getText(),
Float.parseFloat(txtAmountToWithdraw.getText()));
            Account account = m_bankManager.getAccount(txtAccIDtoWithdraw.getText());
            txa_printArea.setText("The account was credited \n as follows:" +
account.getClient().getName() + "\nBalance: " + account.getBalance());
        }
    }
    catch (RemoteException remoteException)
    {
        System.err.println(remoteException);
    }
}

```

TÀI LIỆU THAM KHẢO

- [1]. Cay S. Horstmann, *Core Java Volum I - Fundamentals, Tenth Edition*, NewYork : Prentice Hall, 2016.
- [2]. Cay S. Horstmann. *Core Java Volum II - Advanced Features, Tenth Edition*, New York : Prentice Hall, 2017.
- [3].Eng.haneen Ei-masry, *Java database connection*, Islamic University of Gaza Faculty of Engineering Department of Computer Engineering ECOM 4113: DataBase Lab, 2014.
- [4]. Angelos Stavrou, *Advanced Network Programming Lab using Java*, Network Security, ISA 656, Angelos Stavrou.
- [5]. Marenglen Biba, Ph.D, *Manual for Lab practices, Remote Method Invocation Three Tier Application with a Database Server*, Department of Comsputer Science, University of New York.
- [6].Elliotte Rusty Harold, *Java Network Programming, Fourth Edition*, O'Reilly Media, 2013.
- [7]. Đoàn Văn Ban, *Lập trình hướng đối tượng với JAVA*, Nhà xuất bản Khoa học và Kỹ thuật, 2005.
- [8]. ThS. Dương Thành Phết, *Bài tập thực hành Chuyên đề 1 CNPM- Java*, Khoa CNTT- Trường ĐH Công nghệ TP.HCM.
- [9]. <https://www.oracle.com/technetwork/java/socket-140484.html#>
- [10]. https://personales.unican.es/corcuerp/java/Labs/LAB_22.htm
- [11]. <http://www.nrcmec.org/pdf/Manuals/CSE/student/2-2%20java16-17.pdf>
- [12]. <http://cse.mait.ac.in/pdf/LAB%20MANUAL/JAVA.pdf>
- [13]. https://www.academia.edu/35283541/Bài_tập_môn_lập_trình_hướng_đối_tượng