

Lập trình phân tán: RMI

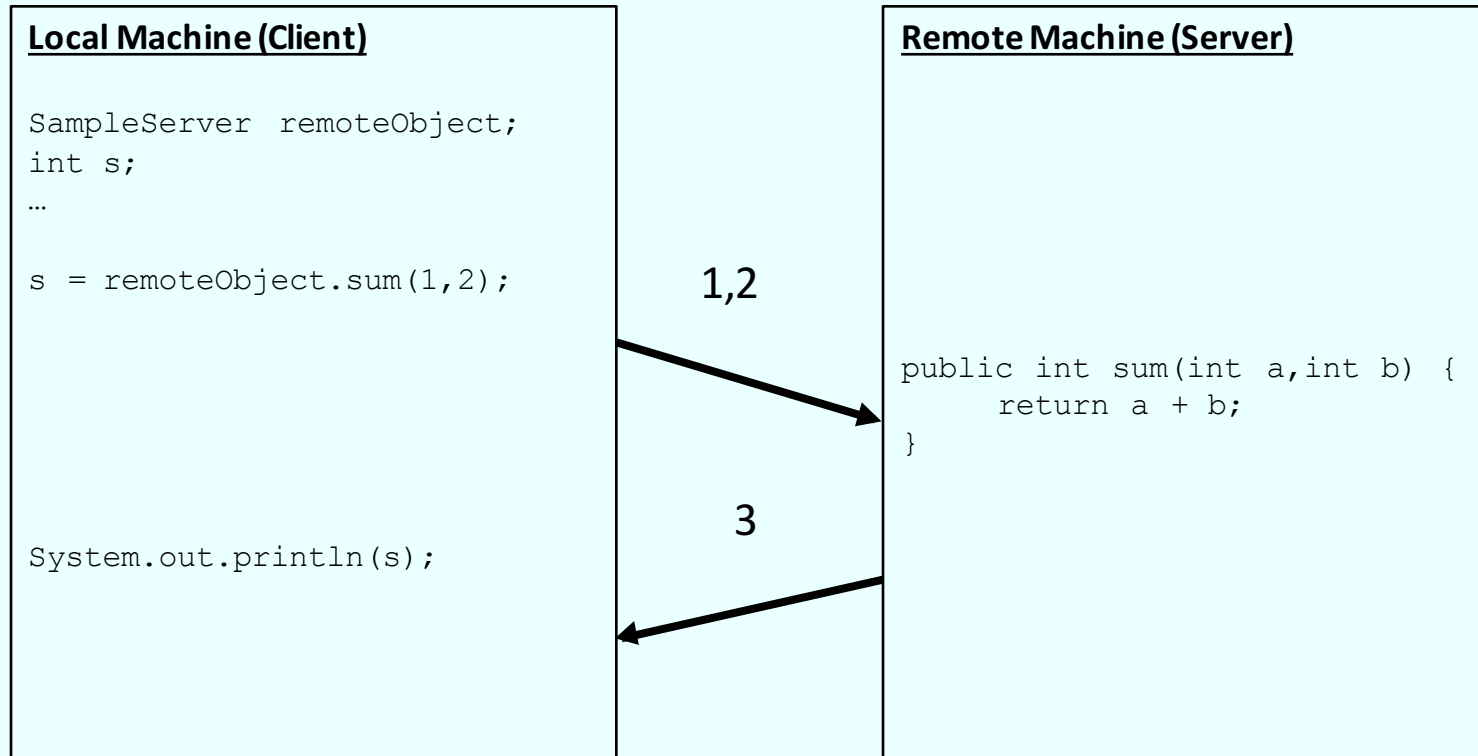
Remote Invoke Method

- RMI là một cơ chế cho phép một đối tượng đang chạy trên một máy ảo Java này (Java Virtual Machine) gọi các phương thức của một đối tượng đang tồn tại trên một máy ảo Java khác (JVM)
- RMI tạo ra các ứng dụng phân tán có độ tin cậy một cách dễ dàng

RMI

- Server: Cung cấp dịch vụ RMI (phương thức từ xa)
- Client: Gọi các phương thức từ xa được cung cấp bởi server.

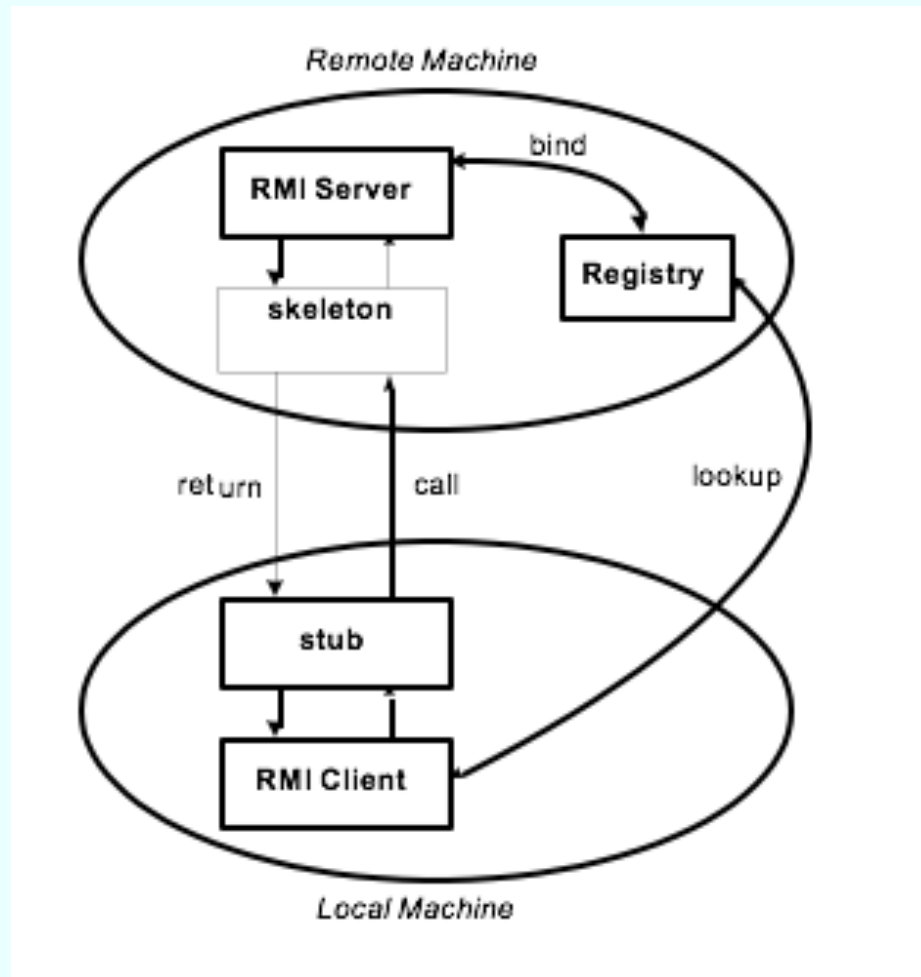
RMI Ví dụ



Truyền tin trong RMI

- RMI sử dụng các lớp trung gian để truyền tin: Skeleton và Stub
- Lớp Stub dùng ở client.
- Lớp Skeleton dùng phía server.
- Java sử dụng rmic.exe để tạo các lớp trung gian.
- TCP Socket

Kiến trúc RMI



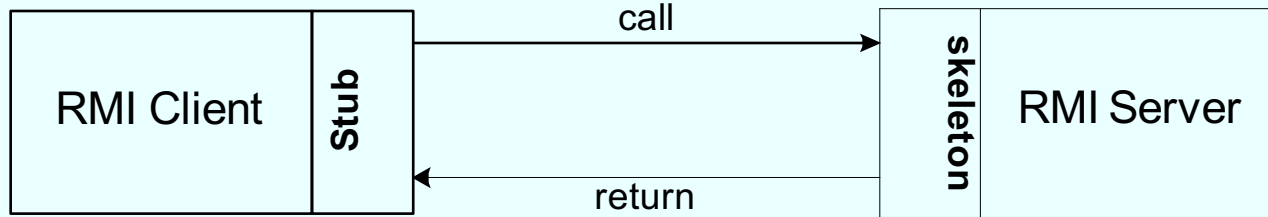
Hoạt động của RMI

- Server RMI phải đăng ký với một dịch vụ tra tìm và đăng ký tên (rmiregistry)
- Sau khi server được đăng ký, nó sẽ chờ các yêu cầu RMI từ các client
- Nếu một dịch vụ chuyển từ server này sang một server khác, client chỉ cần tra tìm trình đăng ký để tìm ra vị trí mới
- Các client RMI sẽ gửi các thông điệp RMI để gọi một phương thức trên một đối tượng từ xa

Hoạt động của RMI

- Ứng dụng client yêu cầu một tên dịch vụ cụ thể, và nhận một URL trỏ tới tài nguyên từ xa
- *rmi://hostname:port/servicename*

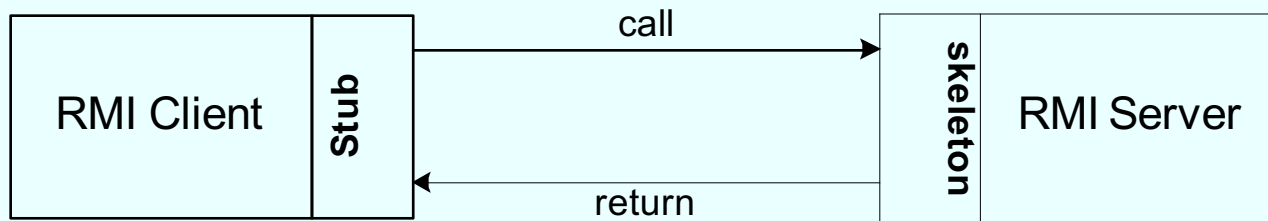
Stub



- Stub: một đối tượng ủy quyền, truyền tải yêu cầu đối tượng tới server RMI
- Người phát triển ứng dụng không cần quan tâm đến tài nguyên RMI nằm ở đâu, nó đang chạy trên nền nào, nó đáp ứng đầy đủ yêu cầu như thế nào

`> Client RMI gọi một phương thức trên đối tượng ủy quyền

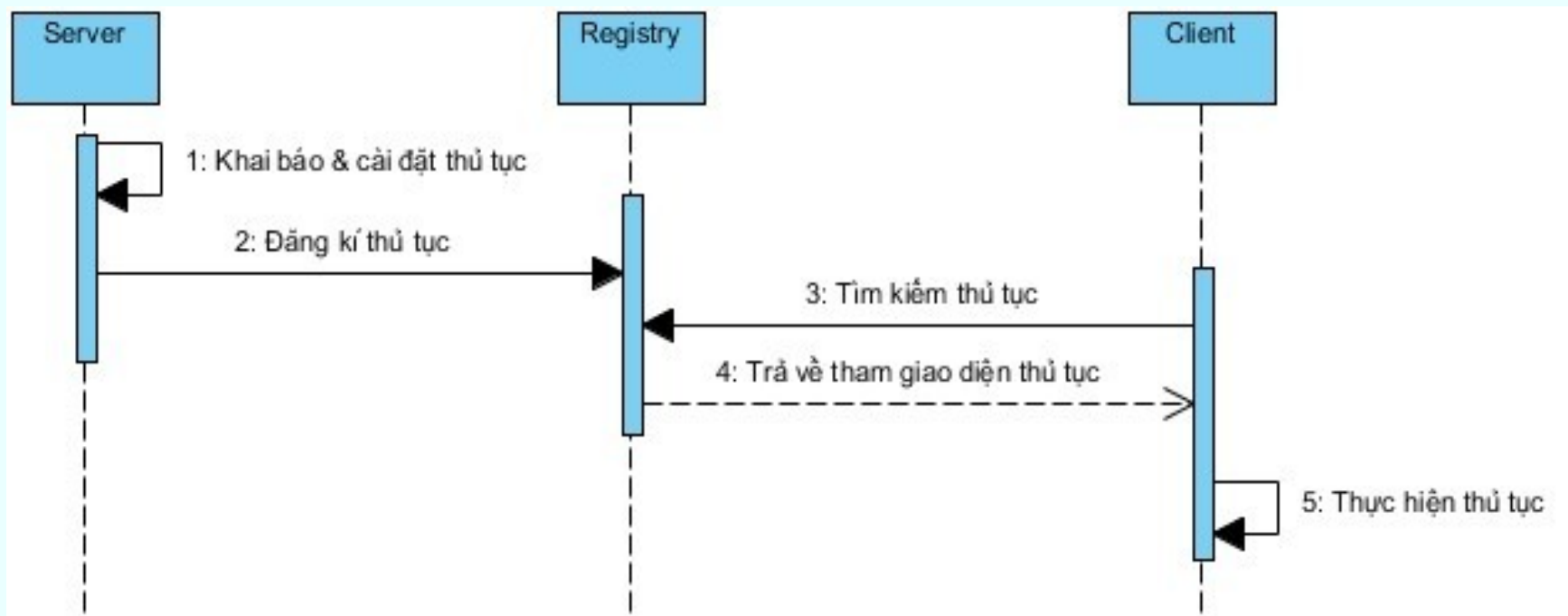
Skeleton

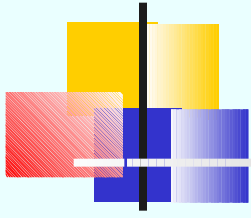


- Skeleton có nhiệm vụ lắng nghe các yêu cầu RMI đến và truyền các yêu cầu này tới dịch vụ RMI
- Skeleton không cung cấp bản cài đặt của dịch vụ RMI. Nó chỉ đóng vai trò như là chương trình nhận các yêu cầu, và truyền các yêu cầu

- java.rmi.server.*
- java.rmi.*

RMI: quan điểm lập trình

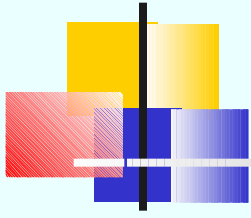




Server (1)

Bước 1: Khai báo Interface cho RMI server, ví dụ chỉ có duy nhất phương thức đổi chiều chuỗi kí tự

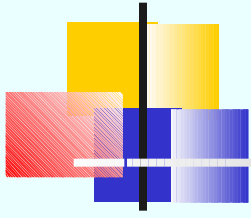
```
import java.rmi.Remote;  
import java.rmi.RemoteException;  
  
public interface RMIInterface extends Remote{  
    public String reverse(String str) throws RemoteException;  
}
```



Server (2)

Bước 2: Cài đặt các phương thức đã khai báo trong Interface, ví dụ với bài toán đổi chiều chuỗi kí tự

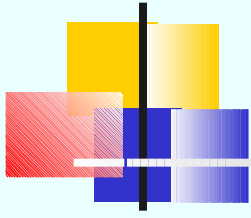
```
public String reverse(String str) throws RemoteException{  
    ReverseString tmp = new ReverseString(str);  
    tmp.reverse();  
    return tmp.get_string();  
}
```



Server (3)

Bước 3: Đăng kí đối tượng RMI vào registry, có thể thực hiện ngay trong hàm khởi tạo, hoặc có thể thực hiện khi gọi đối tượng RMI server (trong hàm main)

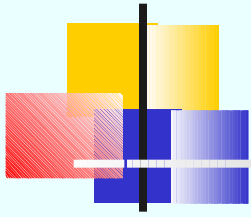
```
// dang ki RMI server
try{
    registry = LocateRegistry.createRegistry(thisPort);
    registry.rebind("rmiServer", this);
} catch (RemoteException e) {
    throw e;
}
```



Client (1)

Bước 1: Tìm kiếm đối tượng RMI trên server

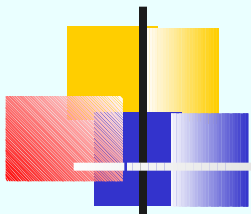
```
try{  
    // lay the dang ki  
    Registry registry =  
        LocateRegistry.getRegistry(address,port);  
  
    // tim kiem RMI server  
    RMIInterface myServer =  
        (RMIInterface) (registry.lookup("rmiServer"));  
} catch (RemoteException e) {  
    e.printStackTrace();  
} catch (NotBoundException e) {  
    e.printStackTrace();  
}
```

Client (2)

Bước 2: Gọi phương thức tương ứng của đối tượng

```
try{  
    // gọi hàm tu xa  
    return myServer.reverse(du liệu cần xử lí);  
} catch (RemoteException e) {  
    e.printStackTrace();  
}
```



Lưu ý

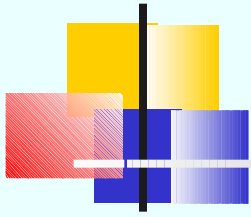
Nếu dùng Naming để đăng kí đối tượng từ xa (bước 3 của server) thì việc tìm kiếm đối tượng từ xa từ phía client cũng khác

Server

```
try{
    Naming.rebind("rmi://localhost:4444/rmiServer", this);
} catch (Exception e) {
    System.out.println(e);
}
```

Client

```
try{
    RMIServer myServer =
(RMIServer)Naming.lookup("rmi://localhost:4444/rmiServer");
} catch (Exception e) {
    System.out.println(e);
}
```



Ví dụ: đảo chuỗi (1)

```
import java.lang.String;

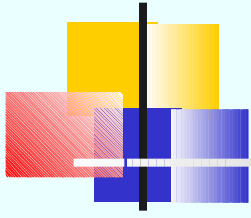
public class ReverseString {
    private String _string;

    // khởi tạo không tham số
    public ReverseString() {
    }

    // khởi tạo có tham số
    public ReverseString(String _string) {
        this._string = _string;
    }

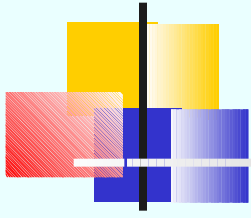
    public String get_string() {
        return _string;
    }

    public void set_string(String _string) {
        this._string = _string;
    }
}
```



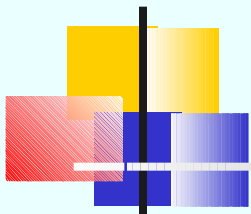
Ví dụ: đảo chuỗi (2)

```
//phuong thuc dao nguoc chuoì kí tu của lớp này
public void reverse(){
    String tmp = "";
    for(int i=_string.length() - 1; i >=0 ;i--){
        tmp += _string.substring(i, i+1);
    }
    this._string = tmp;
}
```



Ví dụ: đảo chuỗi – server (1)

```
import java.rmi.Remote;  
import java.rmi.RemoteException;;  
  
public interface RMIIInterface extends Remote{  
    public String reverse(String str) throws RemoteException;  
}
```



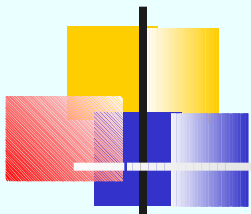
Ví dụ: đảo chuỗi – server (2)

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

public class RMIServer extends UnicastRemoteObject implements
RMIInterface{

    int      thisPort = 3232; // this port (registry's port)
    String   thisAddress;
    Registry registry; // dang ki RMI

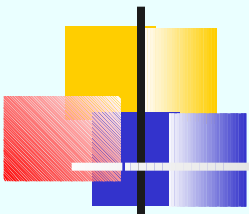
    public String reverse(String str) throws RemoteException{
        ReverseString tmp = new ReverseString(str);
        tmp.reverse();
        return tmp.get_string();
    }
}
```



Ví dụ: đảo chuỗi – server (3)

```
// khởi tạo đồng thời đăng kí RMI server
public RMIServer() throws RemoteException{

    // đăng kí RMI server
    try{
        registry=LocateRegistry.createRegistry(thisPort);
        registry.rebind("rmiServer", this);
    } catch (RemoteException e) {
        throw e;
    }
}
```

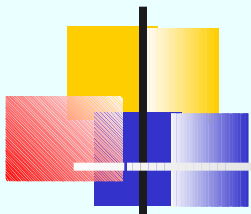


Ví dụ: đảo chuỗi – client (1)

```
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.NotBoundException;

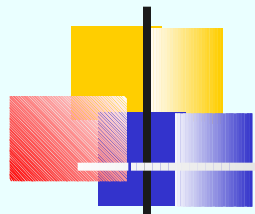
public class RMIClient {
    RMIInterface rmiServer;
    Registry registry;

    public RMIClient(String address, int port) {
        try {
            // lay the dang ki
            registry=LocateRegistry.getRegistry(address,port);
            // tim kiem RMI server
            RmiServer =
                (RMIInterface) (registry.lookup("rmiServer"));
        } catch (RemoteException e) {
            e.printStackTrace();
        } catch (NotBoundException e) {
            e.printStackTrace();
        }
    }
}
```

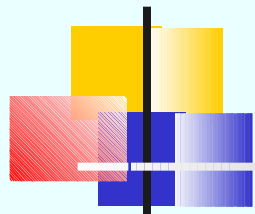
Ví dụ: đảo chuỗi – client (2)

```
// tra ve ket qua
public String getResult(String input) {
    try{
        // goi ham tu xa
        return rmiServer.reverse(input);
    } catch (RemoteException e) {
        e.printStackTrace();
    }
    return null;
}
}
```



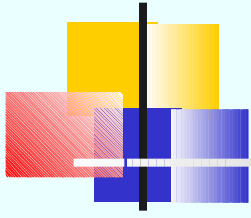
Bài tập (1)

- Cài đặt theo mô hình RMI cho ví dụ trong bài, dùng Naming thay vì dùng registry



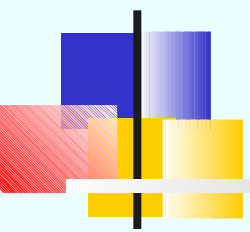
Bài tập (2)

- Cài đặt theo mô hình RMI cho thủ tục tính USCLN của hai số nguyên dương a và b, dùng Naming thay vì dùng registry
- Viết lại bài tập này theo mô hình MVC



Bài tập (3)

- Cài đặt theo mô hình RMI cho thủ tục kiểm tra đăng nhập theo username và password, thông tin này được lưu ở một CSDL trên server khác.
- Viết lại bài tập này theo mô hình MVC



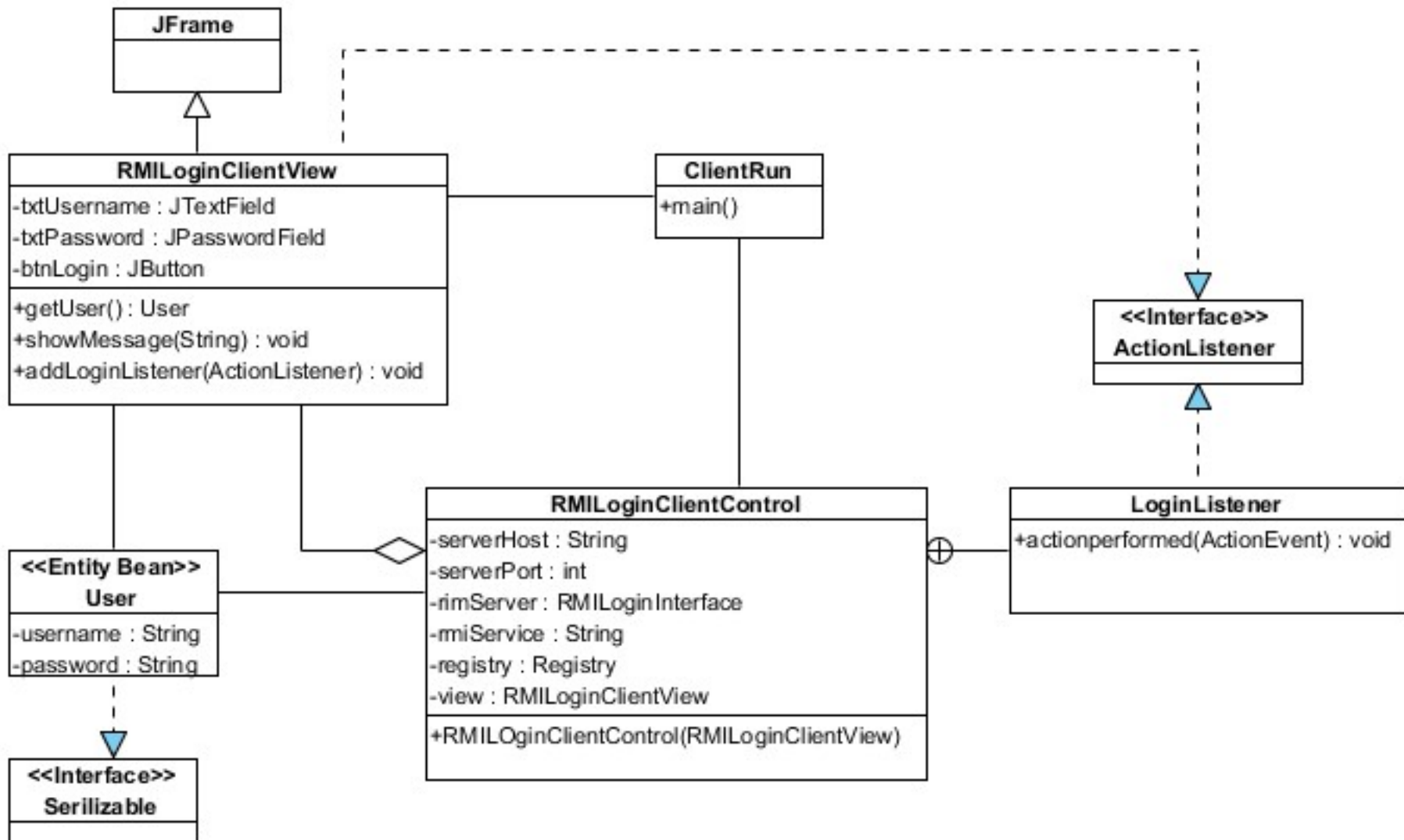
Ví dụ: Login từ xa dùng RMI



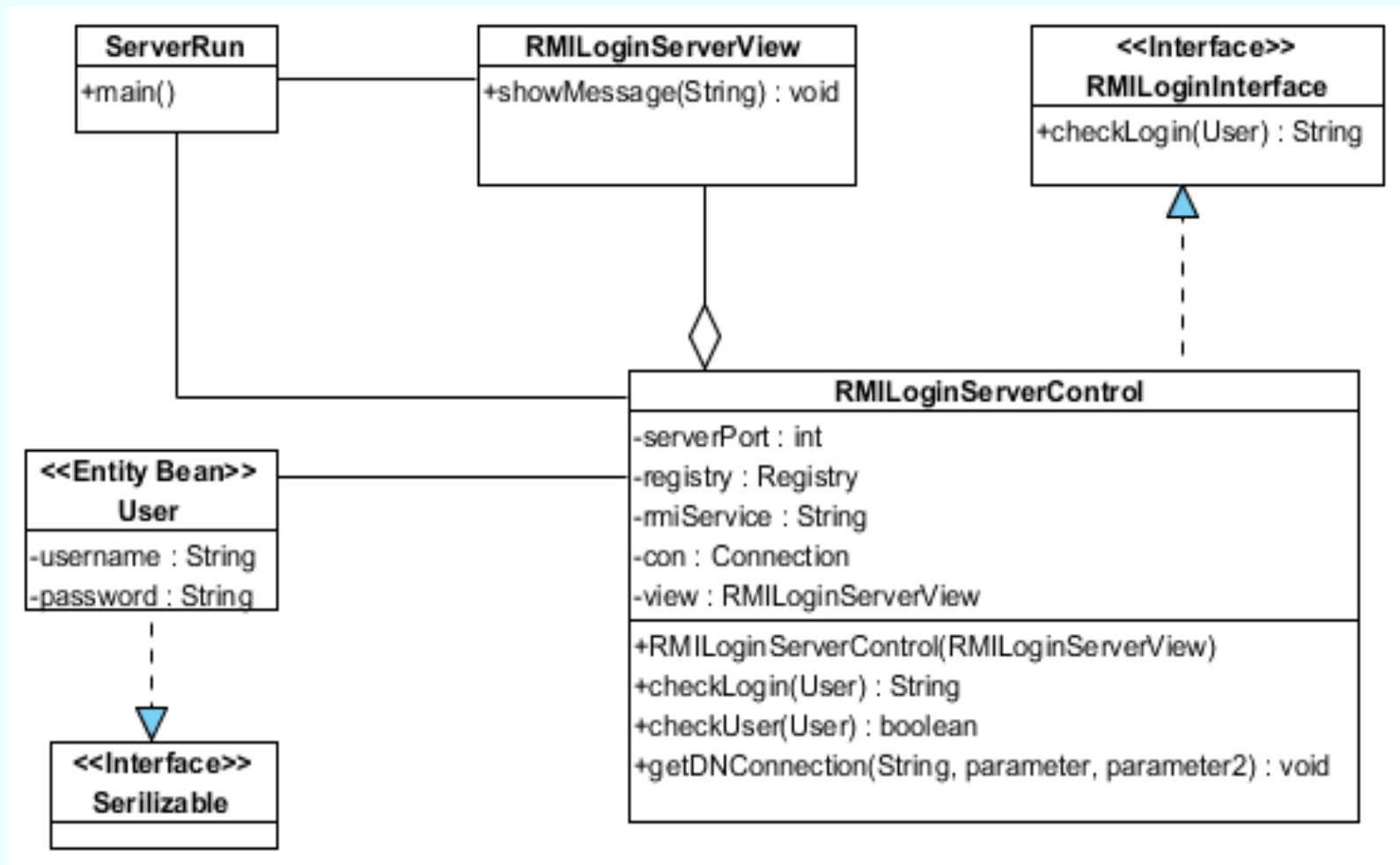
Bài toán: Login dùng RMI

- Thông tin user được lưu trên server.
- Server RMI cung cấp phương thức checkLogin bởi RMI
- Chương trình hiện cửa sổ đăng nhập GUI (username, password)
- Khi click vào nút login, chương trình sẽ triệu gọi phương thức checkLogin của RMI để kiểm tra đăng nhập
- Kết quả đăng nhập được thông báo lại cho người dùng

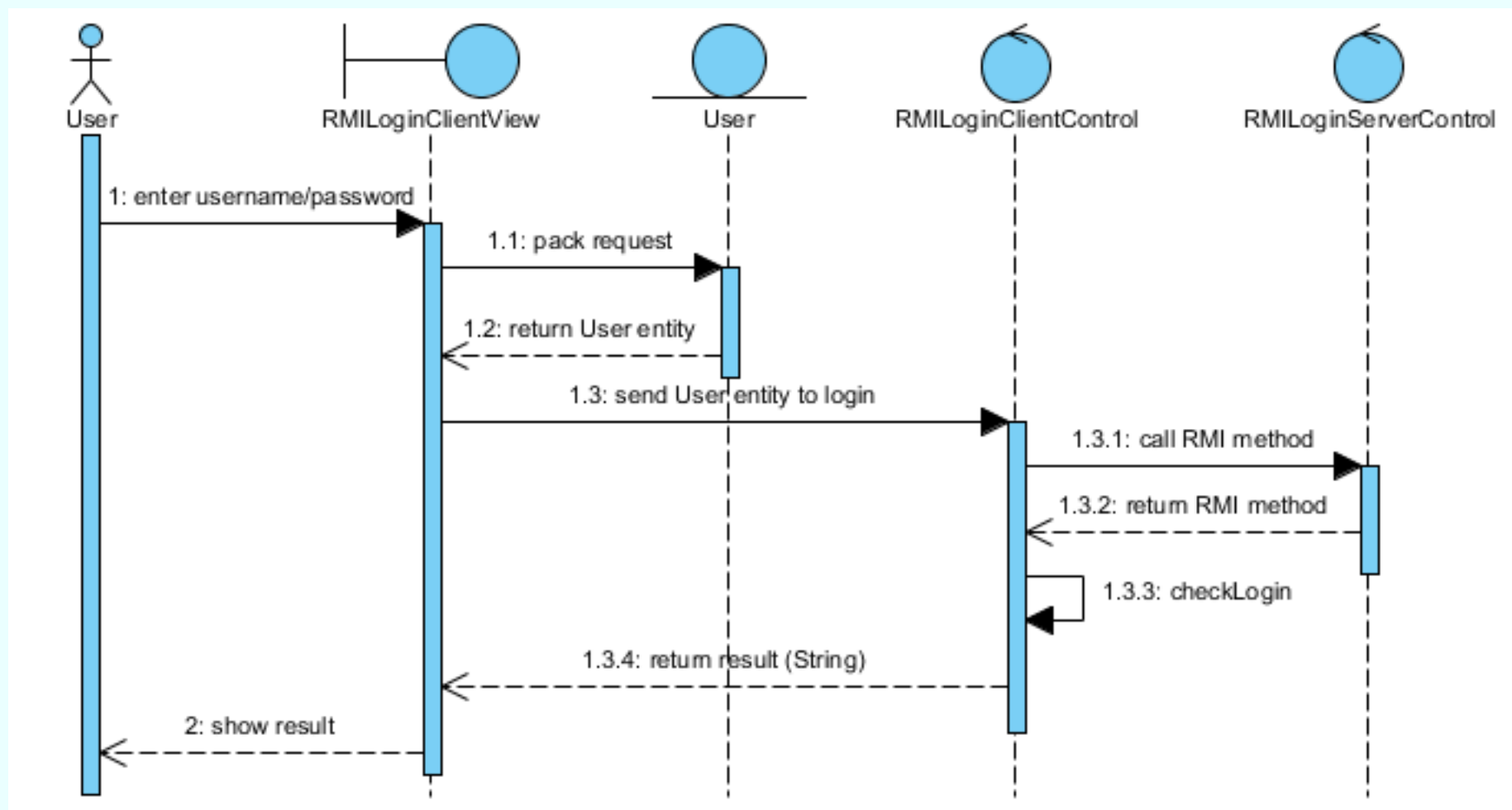
Sơ đồ lớp phía client



Sơ đồ lớp phía server



Tuần tự thực hiện





Lớp: User

```
import java.io.Serializable;

public class User implements Serializable{
    private String userName;
    private String password;

    public User(){
    }

    public User(String username, String password){
        this.userName = username;
        this.password = password;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getUserName() {
        return userName;
    }

    public void setUserName(String userName) {
        this.userName = userName;
    }
}
```

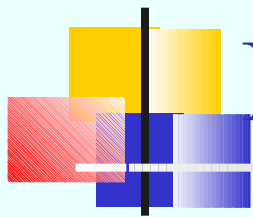


Lớp: RMILoginClientView (1)

```
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.JTextField;

public class RMILoginClientView extends JFrame implements
ActionListener{
    private JTextField txtUsername;
    private JPasswordField txtPassword;
    private JButton btnLogin;
```



Lớp: RMILoginClientView (2)

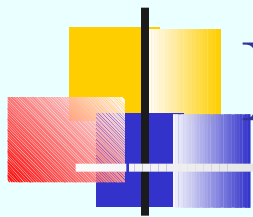
```
public RMILoginClientView(){
    super("RMI Login MVC");

    txtUsername = new JTextField(15);
    txtPassword = new JPasswordField(15);
    txtPassword.setEchoChar('*');
    btnLogin = new JButton("Login");

    JPanel content = new JPanel();
    content.setLayout(new FlowLayout());
    content.add(new JLabel("Username:"));
    content.add(txtUsername);
    content.add(new JLabel("Password:"));
    content.add(txtPassword);
    content.add(btnLogin);

    this.setContentPane(content);
    this.pack();

    this.addWindowListener(new WindowAdapter(){
        public void windowClosing(WindowEvent e){
            System.exit(0);
        }
    });
}
```



Lớp: RMILoginClientView (3)

```
public void actionPerformed(ActionEvent e) {  
}  
  
public User getUser(){  
    User model = new User(txtUsername.getText(),  
        txtPassword.getText());  
    return model;  
}  
  
public void showMessage(String msg){  
    JOptionPane.showMessageDialog(this, msg);  
}  
  
public void addLoginListener(ActionListener log) {  
    btnLogin.addActionListener(log);  
}  
}
```

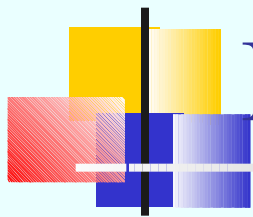


RMILoginClientControl (1)

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

import rmi.server.RMILoginInterface;

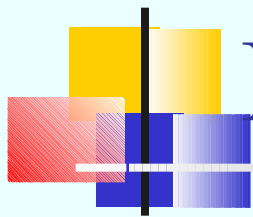
public class RMILoginClientControl {
    private RMILoginClientView view;
    private String serverHost = "localhost";
    private int serverPort = 3232;
    private RMILoginInterface rmiServer;
    private Registry registry;
    private String rmiService = "rmiLoginServer";
```



RMILoginClientControl (2)

```
public RMILoginClientControl(RMILoginClientView view){
    this.view = view;
    view.addLoginListener(new LoginListener());

    try{
        // lay the dang ki
        registry = LocateRegistry.getRegistry(serverHost,
            serverPort);
        // tim kiem RMI server
        rmiServer = (RMILoginInterface)
            (registry.lookup(rmiService));
    }catch(RemoteException e){
        view.showMessage(e.getStackTrace().toString());
        e.printStackTrace();
    }catch(NotBoundException e){
        view.showMessage(e.getStackTrace().toString());
        e.printStackTrace();
    }
}
```



RMILoginClientControl (3)

```
class LoginListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        try {
            User model = view.getUser();
            if(rmiServer.checkLogin(model).equals("ok")){
                view.showMessageDialog("Login succesfully!");
            }else{
                view.showMessageDialog("Invalid username and/or
password!");
            }
        } catch (Exception ex) {
            view.showMessageDialog(ex.getStackTrace().toString());
            ex.printStackTrace();
        }
    }
}
```




ClientRun

```
public class ClientRun {  
    public static void main(String[] args) {  
        RMILoginClientView view      = new RMILoginClientView();  
        RMILoginClientControl control = new  
RMILoginClientControl(view);  
        view.setVisible(true);  
    }  
}
```



RMILoginInterface

```
import java.rmi.Remote;  
import java.rmi.RemoteException;  
  
import rmi.client.User;  
  
public interface RMILoginInterface extends Remote{  
    public String checkLogin(User user) throws RemoteException;  
}
```



RMILoginServerView

```
public class RMILoginServerView {  
    public RMILoginServerView(){  
    }  
  
    public void showMessage(String msg){  
        System.out.println(msg);  
    }  
}
```



RMILoginServerControl (1)

```
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

import rmi.client.User;

public class RMILoginServerControl extends UnicastRemoteObject
implements RMILoginInterface{
    private int serverPort = 3232;
    private Registry registry;
    private Connection con;
    private RMILoginServerView view;
    private String rmiService = "rmiLoginServer";
```

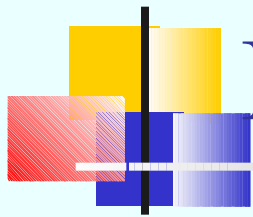


RMILoginServerControl (2)

```
public RMILoginServerControl(RMILoginServerView view) throws
RemoteException{
    this.view = view;
    getDBConnection("myDBName", "admin", "123456");
    view.showMessage("RMI server is running...");

    // dang ki RMI server
    try{
        registry = LocateRegistry.createRegistry(serverPort);
        registry.rebind(rmiService, this);
    }catch(RemoteException e){
        throw e;
    }
}

public String checkLogin(User user) throws RemoteException{
    String result = "";
    if(checkUser(user))
        result = "ok";
    return result;
}
```



RMILoginServerControl (3)

```
private void getDBConnection(String dbName, String username,
String password){
    String dbUrl = "jdbc:mysql://your.database.domain/" + dbName;
    String dbClass = "com.mysql.jdbc.Driver";

    try {
        Class.forName(dbClass);
        con = DriverManager.getConnection (dbUrl, username,
password);
    }catch(Exception e) {
        view.showMessageDialog(e.getStackTrace().toString());
    }
}
```



RMILoginServerControl (4)

```
private boolean checkUser(User user) {
    String query = "Select * FROM users WHERE username =" +
user.getUserName()
        + "' AND password =" + user.getPassword() + "'";

    try {
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery(query);

        if (rs.next()) {
            return true;
        }
    } catch (Exception e) {
        view.showMessageDialog(e.getStackTrace().toString());
    }

    return false;
}
```



ServerRun

```
public class ServerRun {  
    public static void main(String[] args) {  
        RMILoginServerView view = new RMILoginServerView();  
        try{  
            RMILoginServerControl control = new  
                RMILoginServerControl(view);  
        }catch(Exception e){  
            e.printStackTrace();  
        }  
    }  
}
```


Ví dụ:

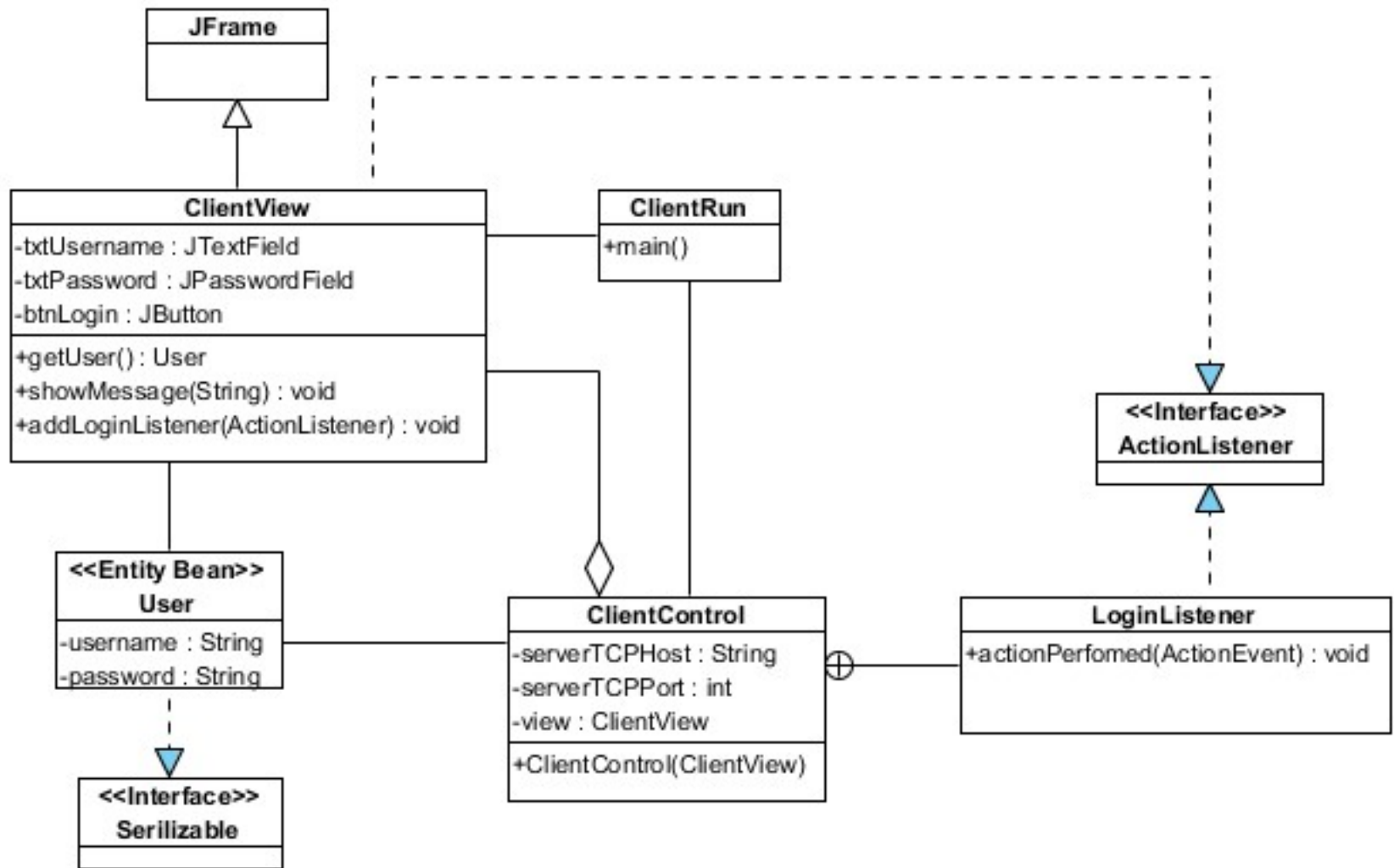
 Login từ xa dùng TCP/IP-RMI



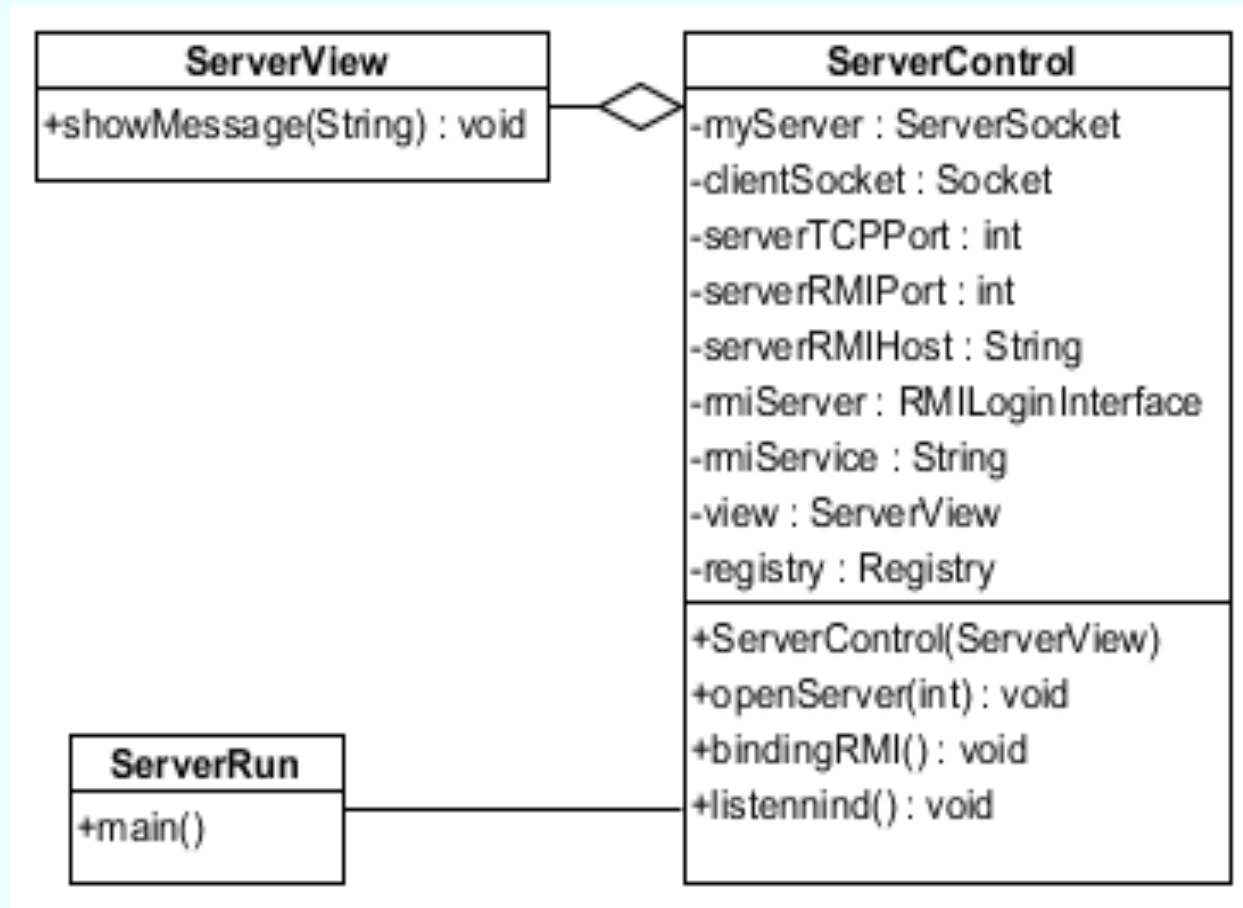
Bài toán

- Thông tin user được lưu trên server TCP.
- Server RMI và cung cấp phương thức checkLogin bởi RMI
- Chương trình hiện cửa sổ đăng nhập GUI (username, password) ở phía client TCP
- Khi click vào nút login, client TCP sẽ gửi thông tin đăng nhập đến server TCP xử lí
- Server TCP sẽ triệu gọi phương thức checkLogin của RMI để kiểm tra đăng nhập
- Kết quả đăng nhập được trả từ server RMI về server TCP, server TCP lại trả về cho client TCP

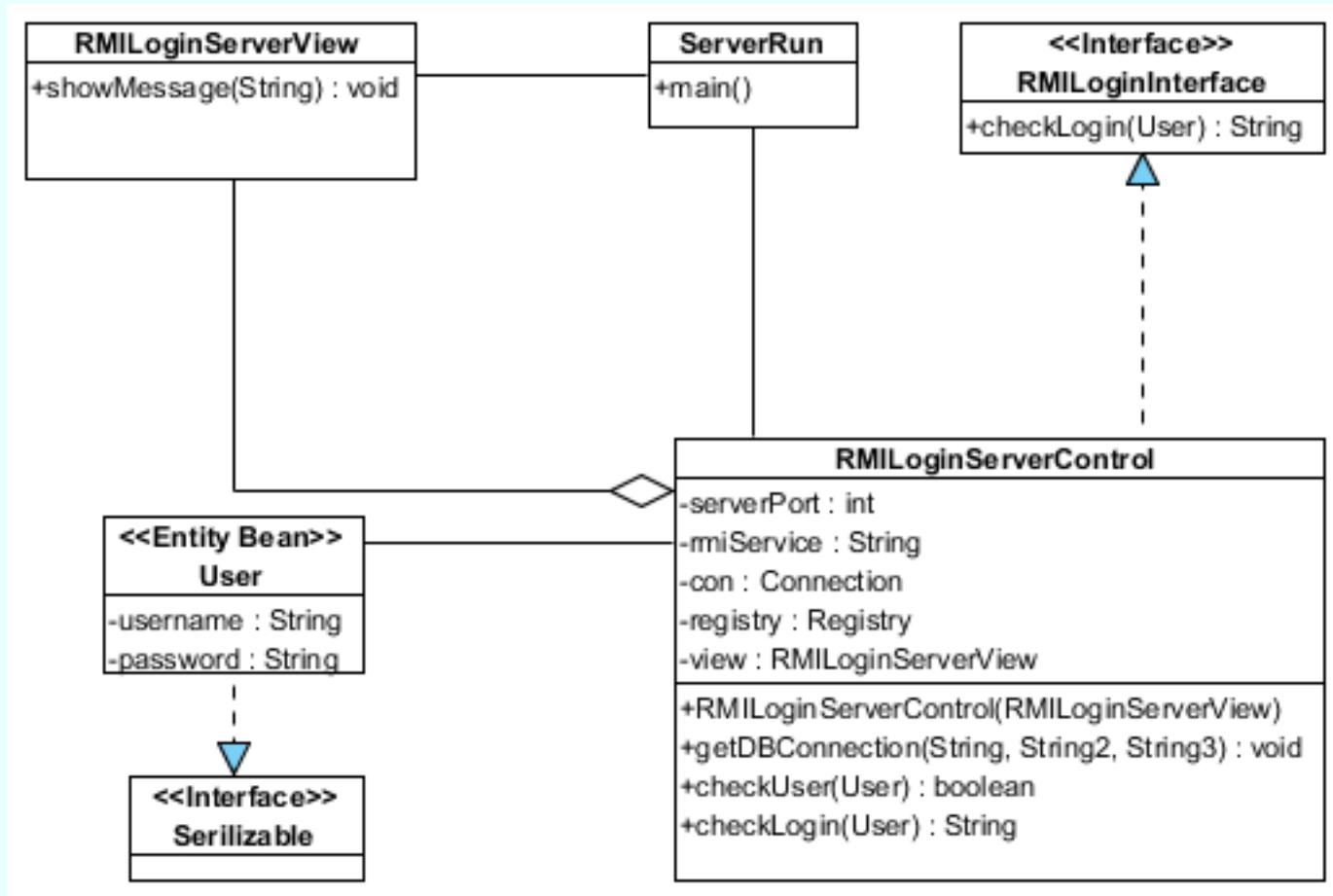
Sơ đồ lớp phía client TCP



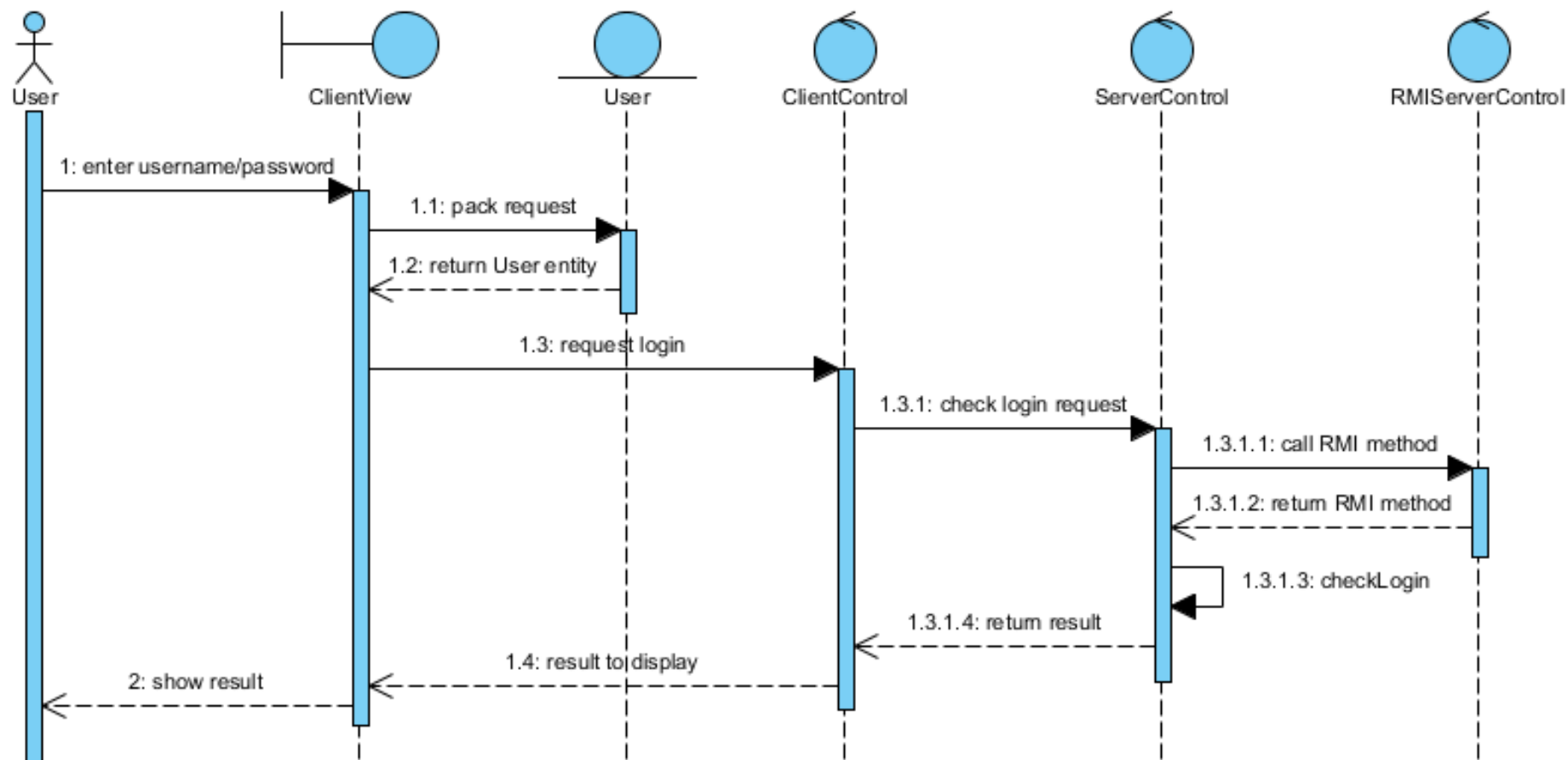
Sơ đồ lớp phía server TCP



Sơ đồ lớp phía server RMI



Tuần tự thực hiện





Lớp: User

```
import java.io.Serializable;

public class User implements Serializable{
    private String userName;
    private String password;

    public User(){
    }

    public User(String username, String password){
        this.userName = username;
        this.password = password;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getUserName() {
        return userName;
    }

    public void setUserName(String userName) {
        this.userName = userName;
    }
}
```

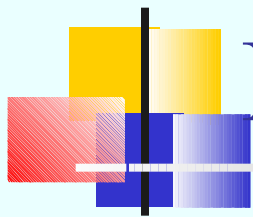


Lóp: ClientView (1)

```
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.JTextField;

public class ClientView extends JFrame implements ActionListener{
    private JTextField txtUsername;
    private JPasswordField txtPassword;
    private JButton btnLogin;
```

Lớp: ClientView (2)

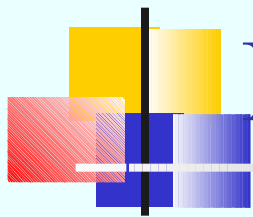
```
public ClientView(){
    super("RMI - TCP Login MVC");

    txtUsername = new JTextField(15);
    txtPassword = new JPasswordField(15);
    txtPassword.setEchoChar('*');
    btnLogin = new JButton("Login");

    JPanel content = new JPanel();
    content.setLayout(new FlowLayout());
    content.add(new JLabel("Username:"));
    content.add(txtUsername);
    content.add(new JLabel("Password:"));
    content.add(txtPassword);
    content.add(btnLogin);

    this.setContentPane(content);
    this.pack();

    this.addWindowListener(new WindowAdapter(){
        public void windowClosing(WindowEvent e){
            System.exit(0);
        }
    });
}
```



Lớp: ClientView (3)

```
public void actionPerformed(ActionEvent e) {  
}  
  
public User getUser(){  
    User model = new User(txtUsername.getText(),  
                           txtPassword.getText());  
    return model;  
}  
  
public void showMessage(String msg){  
    JOptionPane.showMessageDialog(this, msg);  
}  
  
public void addLoginListener(ActionListener log) {  
    btnLogin.addActionListener(log);  
}  
}
```



Lớp: ClientControl (1)

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;

public class ClientControl {
    private ClientView view;
    private String serverTCPHost = "localhost";
    private int serverTCPPort = 8000;

    public ClientControl(ClientView view){
        this.view = view;
        this.view.addLoginListener(new LoginListener());
    }
}
```



Lép: ClientControl (2)

```
class LoginListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        try {
            User user = view.getUser();
            Socket mySocket = new Socket(serverTCPHost, serverTCPPort);
            ObjectOutputStream oos = new
ObjectOutputStream(mySocket.getOutputStream());
            oos.writeObject(user);

            ObjectInputStream ois = new
                ObjectInputStream(mySocket.getInputStream());
            Object o = ois.readObject();
            if(o instanceof String){
                String result = (String)o;
                if(result.equals("ok"))
                    view.showMessageDialog("Login succesfully!");
                else
                    view.showMessageDialog("Invalid username and/or password!");
            }
            mySocket.close();
        } catch (Exception ex) {
            view.showMessageDialog(ex.getStackTrace().toString());
        }
    }
}
```



Lớp: ClientRun

```
public class ClientRun {  
  
    public static void main(String[] args) {  
        ClientView view = new ClientView();  
        ClientControl control = new ClientControl(view);  
        view.setVisible(true);  
    }  
}
```



Lớp: ServerView

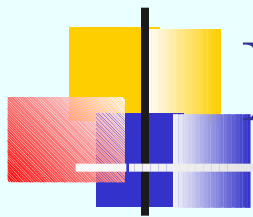
```
public class ServerView {  
    public ServerView(){  
    }  
  
    public void showMessage(String msg){  
        System.out.println(msg);  
    }  
}
```



Lớp: ServerControl (1)

```
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import rmi_tcp.rmiServer.RMILoginInterface;
import rmi_tcp.tcpClient.User;

public class ServerControl {
    private ServerView view;
    private ServerSocket myServer;
    private Socket clientSocket;
    private String serverRMIHost = "localhost";
    private int serverRMIPort = 3535;
    private int serverTCPPort = 8000;
    private RMILoginInterface rmiServer;
    private Registry registry;
    private String rmiService = "rmitcpLoginServer";
```

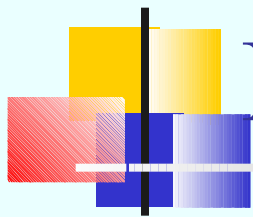


Lớp: ServerControl (2)

```
public ServerControl(ServerView view){
    this.view = view;
    openServer(serverTCPPort);
    bindingRMI();
    view.showMessage("TCP server is running...");

    while(true){
        listenning();
    }
}

private void openServer(int portNumber){
    try {
        myServer = new ServerSocket(portNumber);
    }catch(IOException e) {
        view.showMessage(e.toString());
        e.printStackTrace();
    }
}
```

Lớp: ServerControl (3)

```
private void bindingRMI(){
    try{
        // lay the dang ki
        registry =
LocateRegistry.getRegistry(serverRMIHost,
                                serverRMIPort);
        // tim kiem RMI server
        rmiServer = (RMILoginInterface)
                                (registry.lookup(rmiService));
    }catch(RemoteException e){
        view.showMessageDialog(e.getStackTrace().toString());
    }catch(NotBoundException e){
        view.showMessageDialog(e.getStackTrace().toString());
    }
}
```



Lớp: ServerControl (4)

```
private void listenning(){
    try {
        clientSocket = myServer.accept();
        ObjectInputStream ois = new
            ObjectInputStream(clientSocket.getInputStream());

        Object o = ois.readObject();
        if(o instanceof User){
            User user = (User)o;
            String result = rmiServer.checkLogin(user);
            ObjectOutputStream oos = new
                ObjectOutputStream(clientSocket.getOutputStream());
            oos.writeObject(result);
        }
    } catch (Exception e) {
        view.showMessage(e.toString());
    }
}
```



Lớp: ServerRun

```
public class ServerRun {  
    public static void main(String[] args) {  
        ServerView view      = new ServerView();  
        ServerControl control = new ServerControl(view);  
    }  
}
```



Lớp: RMILoginInterface

```
import java.rmi.Remote;  
import java.rmi.RemoteException;  
  
import rmi_tcp.tcpClient.User;  
  
public interface RMILoginInterface extends Remote{  
    public String checkLogin(User user) throws RemoteException;  
}
```



Lớp: RMILoginServerView

```
public class RMILoginServerView {  
    public RMILoginServerView(){  
    }  
  
    public void showMessage(String msg){  
        System.out.println(msg);  
    }  
}
```



RMILoginServerControl (1)

```
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

import rmi_tcp.tcpClient.User;

public class RMILoginServerControl extends UnicastRemoteObject
implements RMILoginInterface{
    private int serverPort = 3535;
    private Registry registry;
    private Connection con;
    private RMILoginServerView view;
    private String rmiService = "rmitcpLoginServer";
```



RMILoginServerControl (2)

```
public RMILoginServerControl(RMILoginServerView view) throws
RemoteException{
    this.view = view;
    view.showMessageDialog("RMI server is running...");

    // dang ki RMI server
    try{
        registry = LocateRegistry.createRegistry(serverPort);
        registry.rebind(rmiService, this);
    }catch(RemoteException e){
        throw e;
    }
}

public String checkLogin(User user) throws RemoteException{
    String result = "";
    getDBConnection("myDBName", "admin", "123456");
    if(checkUser(user))
        result = "ok";
    return result;
}
```



RMILoginServerControl (3)

```
private void getDBConnection(String dbName, String username,
String password){
    String dbUrl = "jdbc:mysql://your.database.domain/" + dbName;
    String dbClass = "com.mysql.jdbc.Driver";

    try {
        Class.forName(dbClass);
        con = DriverManager.getConnection (dbUrl, username, password);
    }catch(Exception e) {
        view.showMessageDialog(e.getStackTrace().toString());
    }
}
```




RMILoginServerControl (4)

```
private boolean checkUser(User user) {  
    String query = "Select * FROM users WHERE username ="  
        + user.getUserName()  
        + "' AND password =" + user.getPassword() + "'";  
  
    try {  
        Statement stmt = con.createStatement();  
        ResultSet rs = stmt.executeQuery(query);  
  
        if (rs.next()) {  
            return true;  
        }  
    } catch (Exception e) {  
        view.showMessageDialog(e.getStackTrace().toString());  
    }  
  
    return false;  
}
```

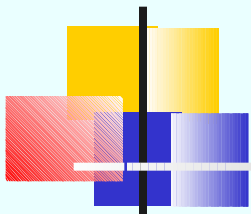


ServerRun

```
public class ServerRun {  
    public static void main(String[] args) {  
        RMILoginServerView view = new RMILoginServerView();  
        try{  
            RMILoginServerControl control = new  
                RMILoginServerControl(view);  
        }catch(Exception e){  
            e.printStackTrace();  
        }  
    }  
}
```

Lưu ý: thứ tự chạy là:

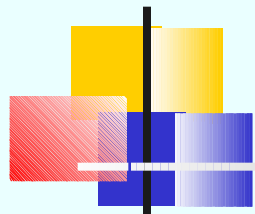
- 1 - chạy serverRun của RMI
- 2 - chạy serverRun của TCP
- 3 - chạy clientRun của TCP



Bài tập (1)

Cài đặt đúng mô hình MVC (cũ hoặc mới) cho bài toán quản lí người dùng theo mô hình RMI:

- Server chứa CSDL về người dùng, có bảng tbluser chứa các cột: id, username, password, address, birthday, sex, description
- Client có giao diện nhập thông tin đăng kí người dùng mới
- Sau khi nhập thông tin và click submit, client gọi thủ tục kiểm tra và thêm mới người dùng từ server
- Server có thủ tục kiểm tra xem có trùng username không, nếu không thì thêm vào CSDL và báo thành công, nếu trùng thì thông báo trùng
- Client sẽ hiển thị yêu cầu người dùng nhập lại khi trùng, hoặc báo đăng kí thành công.



Bài tập (2)

Cài đặt đúng mô hình MVC (cũ hoặc mới) cho bài toán quản lí người dùng theo mô hình RMI:

- Server chứa CSDL về người dùng, có bảng tbluser chứa các cột: id, username, password, address, birthday, sex, description
- Client có giao diện nhập thông tin tìm kiếm người dùng theo tên
- Sau khi nhập thông tin và click submit, client gọi thủ tục tìm kiếm người dùng từ server
- Server có thủ tục tìm kiếm thông tin người dùng từ CSDL
- Client nhận sẽ hiển thị danh sách người dùng có tên chứa từ khóa đã nhập.



Questions?
