HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

BÀI GIẢNG MÔN

Lập trình mạng

Giảng viên: TS. Nguyễn Trọng Khánh

Điện thoại/E-mail:

Bộ môn: CNPM- Khoa CNTT1

Học kỳ/Năm biên soạn: 2022

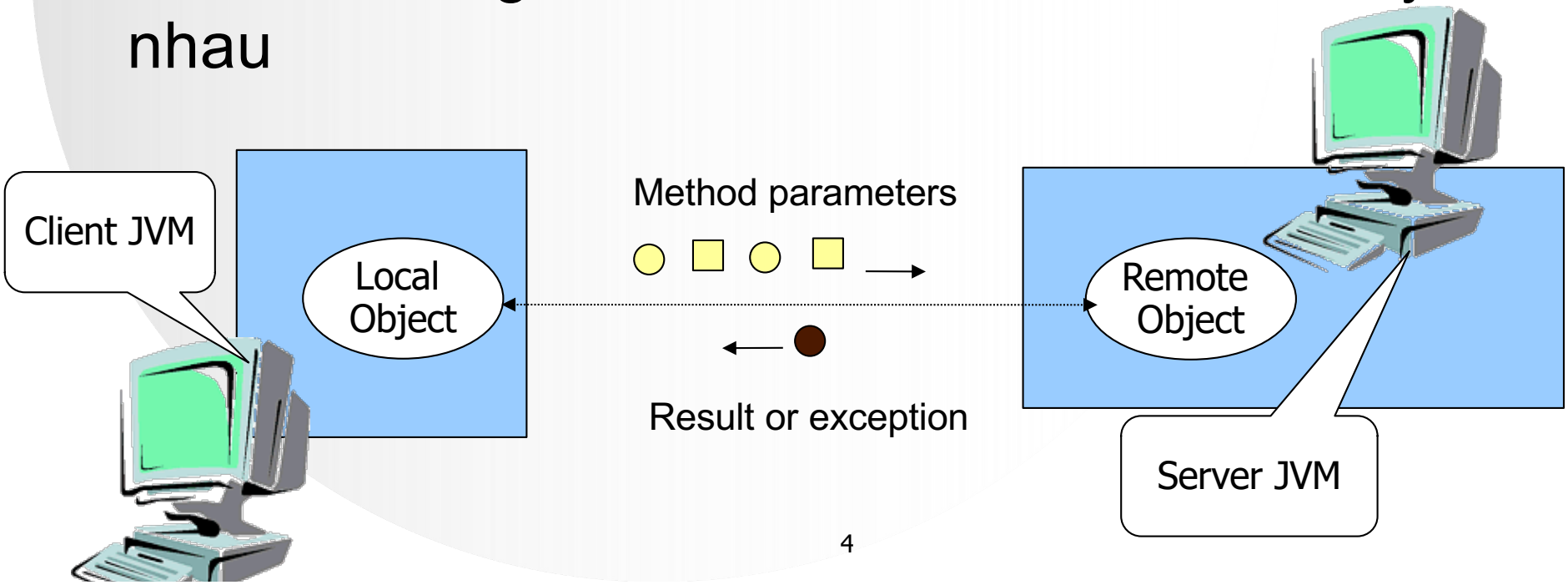
RMI

Nội dung

- Giới thiệu RMI
- Stub & Skeleton
- Cài đặt ứng dụng RMI
- Demo
- Bài tập

Giới thiệu

- Local method Invocation: trên máy cục bộ
  - code nạp thẳng vào bộ nhớ
  - thực thi ngay trên máy cục bộ
- Remote Method Invocation (RMI) : giao tiếp giữa các đối tượng Java có được cài đặt trên máy khác nhau



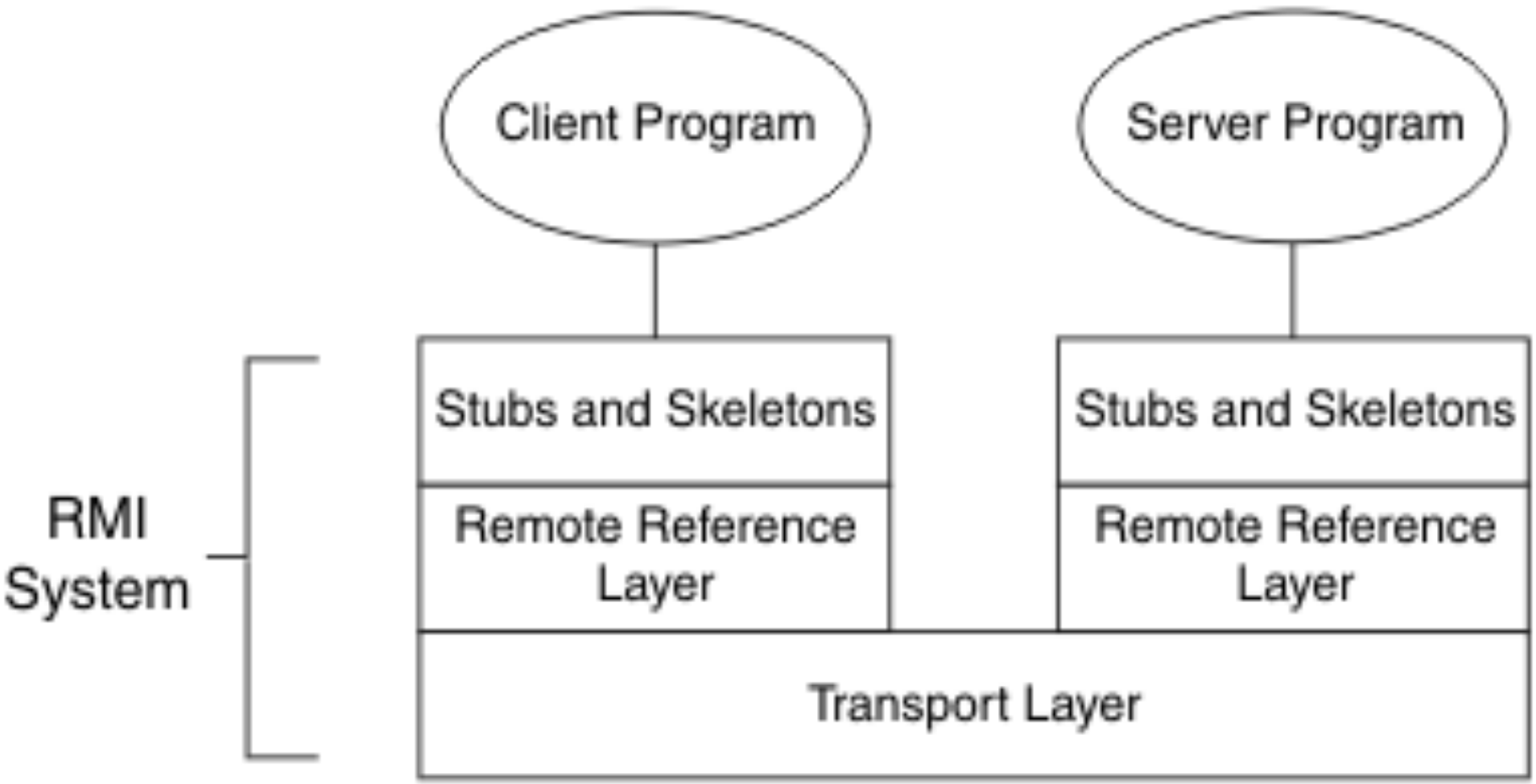
Vấn đề phát sinh

- Gọi phương thức từ xa phức tạp hơn :
  - Tham chiếu đến biến, địa chỉ của đối tượng khác nhau ở các máy khác nhau
  - Truyền tham số cho phương thức từ xa
  - Gọi phương thức từ xa: thông qua mạng → có thể bị ngắt ngang

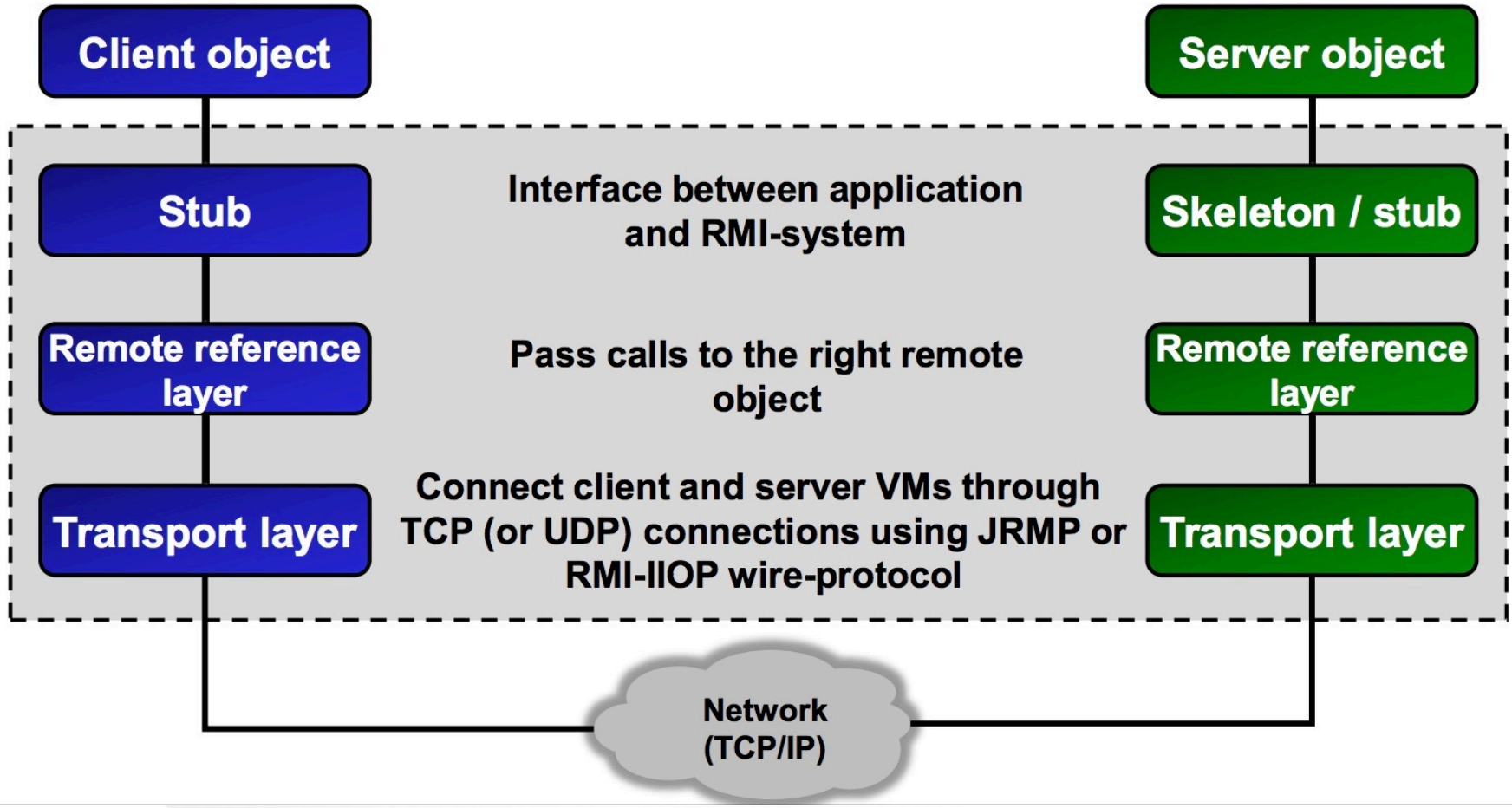
Giải pháp

- Đối tượng trên hai máy khác nhau không gọi trực tiếp mà thông qua lớp trung gian.
- Lớp trung gian tồn tại ở cả hai phía Client và Server.
  - Lớp ở máy Client gọi là Stub,
  - Lớp ở máy Server gọi là Skel(Skeleton)
- Lớp trung gian Stub sẽ biết cách thông báo lỗi khi có các sự cố về mạng cho Client

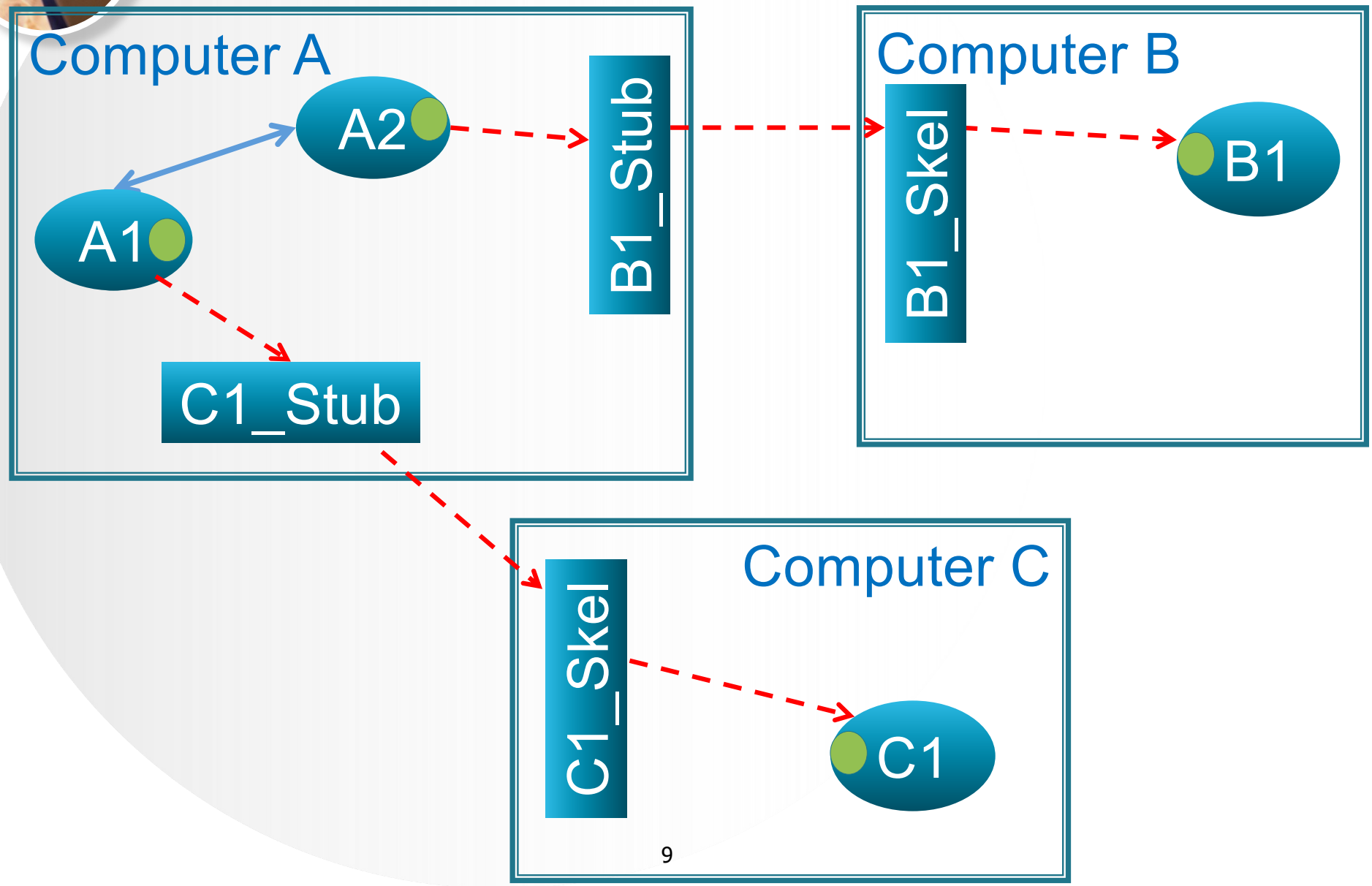
RMI


Client Program
Server Program
Stubs and Skeletons
Stubs and Skeletons
Remote Reference Layer
Remote Reference Layer
RMI System
Transport Layer

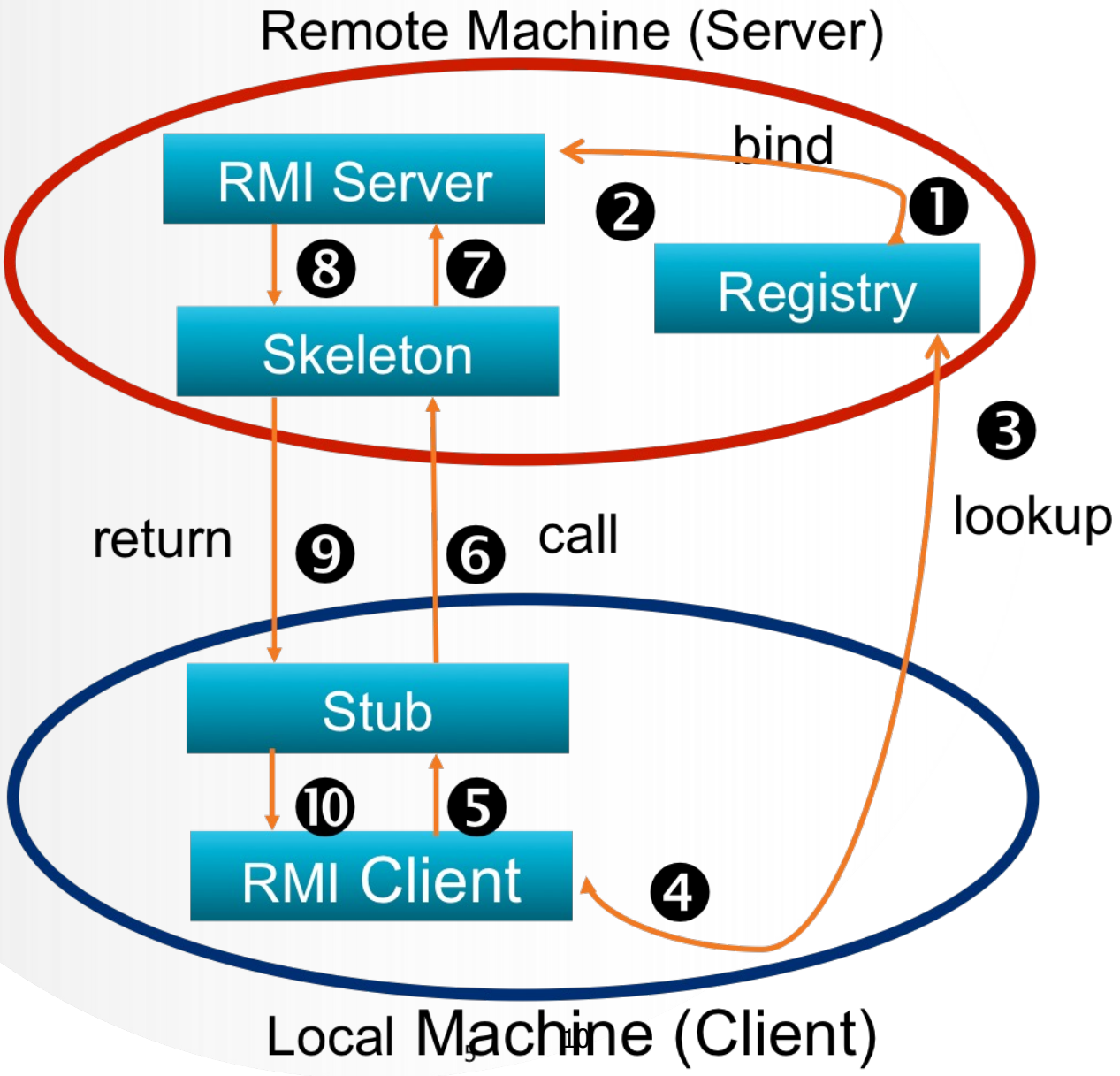
Các lớp trong RMI



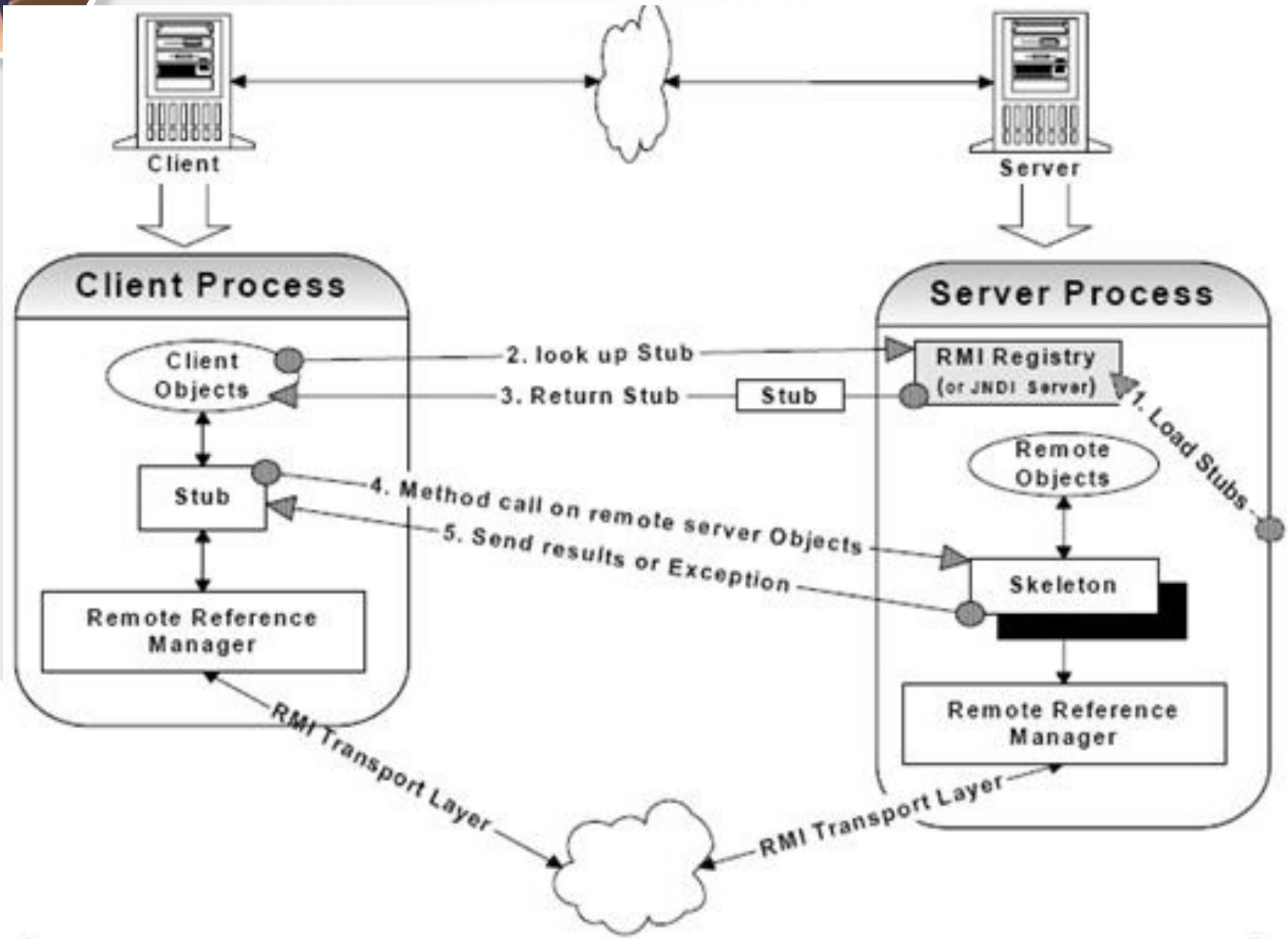
Stub và Skeleton



RMI: Đăng ký và tìm kiếm



Đăng ký và tìm kiếm



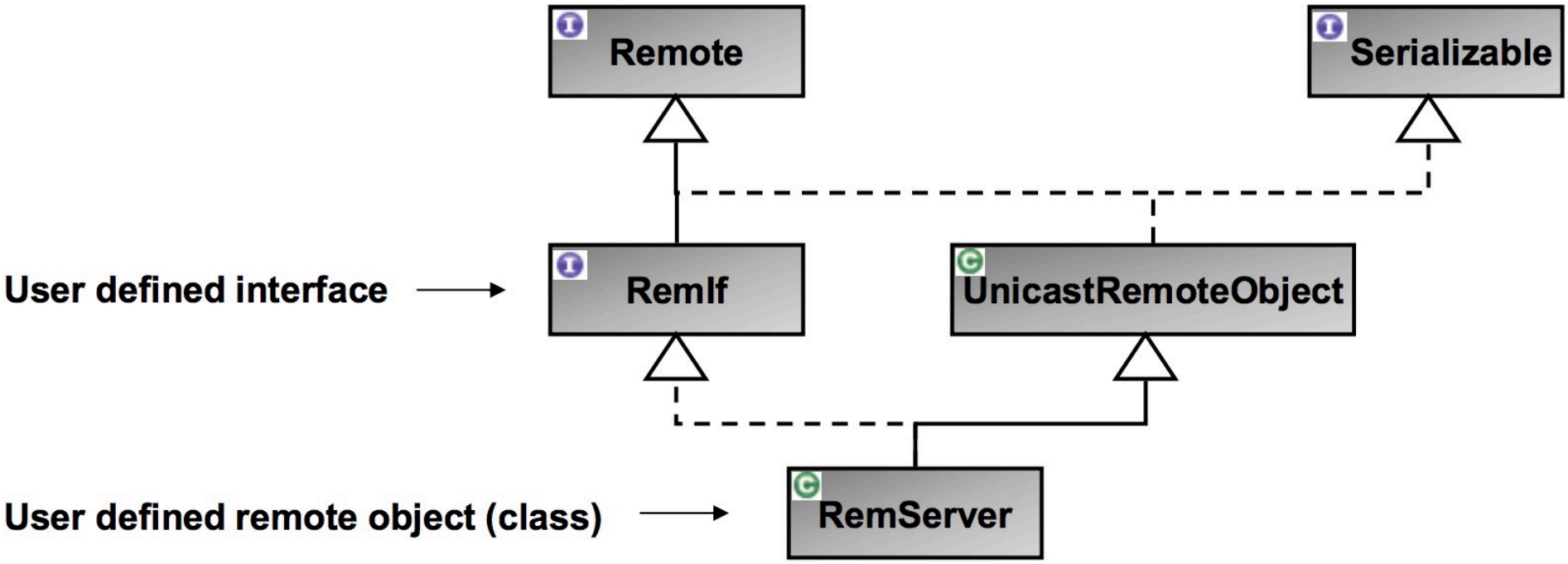
Gói java.rmi

Package	Description
`java.rmi.*`	Core RMI package with classes and interfaces used by both client and server. Contains interface `Remote`, classes `Naming` and `RMISecurityManager` and some basic exception classes.
`java.rmi.activation.*`	Classes and interfaces for dynamic activation of remote objects together with RMI daemon (`rmid`). More information on dynamic invocation see below.
`java.rmi.dgc.*`	Classes and interfaces for distributed garbage collection (DGC).
`java.rmi.registry.*`	`Registry` and `LocateRegistry` classes for directly interacting with a (remote or local) registry. `Registry` class provides `lookup()`, `rebind()`, `list()` and other methods.
`java.rmi.server.*`	Classes for use on the server side like class loader (`RMIClassLoader`) and `UnicastRemoteObject` (base class for remote objects).
`javax.rmi.*`	APIs for RMI-IIOP (interoperability between RMI and CORBA).

Cài đặt RMI

1. Server
   - Đặc tả giao tiếp từ xa.
   - Thực thi giao tiếp từ xa.
   - Sinh Stub phía client và Skeleton phía server.
     - Có thể thực hiện bằng code
   - Xây dựng chương trình phía Server
2. Client
   - Xây dựng chương trình phía Client.
   - Khởi động bộ đăng ký RMI (RMI registry)
   - Khởi động các đối tượng từ xa phía server
   - Chạy chương trình phía client.

RMI Server


Remote
Serializable
User defined interface
RemIf
UnicastRemoteObject
User defined remote object (class)
RemServer

Đăng ký đối tượng RMI

- Cho phép client tìm được đối tượng từ xa
- Thông qua giao tiếp Registry hoặc lớp Naming
- Chung một số phương thức
  - `String[] list(String url)`
  - `Remote lookup(String url)`
  - `void bind(String url, Remote object)`
  - `void rebind(String url, Remote obj)`
  - `void unbind()`

Naming class - methods

- `public static String[] list(String url) throws RemotException`
  - trả về một mảng các xâu ký tự, mỗi xâu là một URL đã được gắn với một tham chiếu. url là URL của trình đăng ký Naming.
- `public static Remote lookup(String url) throws RemotException, NotBoundException, AccessException, MalformedURLException`
  - Client lookup để tìm kiếm một đối tượng từ xa gắn liền với tên đối tượng.
  - **`NotBoundException`**: server ở xa không nhận ra tên của nó.

Naming class - methods

- `Public static void bind(String url, Remote object) throws RemotException, AlreadyBoundException, MalformedURLException, AccessException`
  - để liên kết một tên với một đối tượng ở xa. Nếu thành công thì client có thể tìm kiếm đối tượng stub từ trình đăng ký.
- Exception có thể có khi gán tên
  - **MalformedURLException:** url không đúng cú pháp.
  - **RemoteException**: không thể liên lạc được với trình đ.ký
  - **AccessException**: client không được phép gán các đối tượng trong trình đăng ký.
  - **AlreadyBoundException**: nếu đối tượng URL đã gắn với một đối tượng cục bộ

Naming class - methods

- `public static void rebind(String url, Remote obj)throws RemoteException, AccessException, MalformedURLException`
  - Phương thức này giống như phương thức bind() ngoại trừ việc là nó gán URL cho đối tượng ngay cả khi URL đã được gán.

Đăng ký và tìm kiếm

- RMI Registry: máy chủ cung cấp dịch vụ đăng ký và tìm kiếm.
- Đăng ký server:

```
RemServer localObject = new RemServer();
Naming.rebind("MyServ", localObject);
```

- Tìm kiếm

```
rmi://<host_name> [:<name_service_port>] /<service_name>
RemIf remObject = (RemIf)Naming.lookup("rmi://" + host + "/MyServ");
```

Giao tiếp Registry

- Cho phép các client tìm kiếm các đối tượng ở xa theo tên.
- Các phương thức
  - **Bind()** để gán một tên với một đối tượng từ xa cụ thể
  - **List()** liệt kê tất cả các tên đã được đăng ký với trình đăng ký
  - **Lookup()** tìm một đối tượng từ xa cụ thể với một URL cho trước gắn với nó
  - **Rebind()** gán một tên với một đối tượng ở xa khác
  - **Unbind()** loại bỏ một tên đã được gán cho một đối tượng ở xa trong trình đăng ký
  - **Registry.REGISTRY_PORT** là cổng mặc định để lắng nghe các các yêu cầu. Giá trị mặc định là 1099.

Chuyển tham số trong RMI

- Đối với việc truyền tham số qua mạng theo cơ chế RMI thì có hai nguyên tắc sau:
  - Tất cả các kiểu dữ liệu đơn giản như int, char,… đều được truyền theo tham trị.
  - Tất cả các dữ liệu kiểu đối tượng muốn truyền qua mạng đều buộc phải cài đặt một trong hai giao tiếp là Remote hoặc Serializable.
    - Các đối tượng cài giao tiếp Remote sẽ được truyền theo tham chiếu
    - Các đối tượng cài đặt giao tiếp Serializable sẽ được truyền theo tham trị.
    - Các đối tượng ko được cài đặt giao tiếp Serializable hay Remote thì ko thể dùng làm tham số truyền qua mạng được

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{
    // goi ham tu xa
    return myServer.reverse(du lieu 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("rmiServer", this);
}catch (Exception e){
    System.out.println(e);
}
```

Client

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

Bài tập:Chương trình Calculator

- Xây dựng ứng dụng phân tán cho phép thực thi các phép toán sau
  - Cộng
  - Trừ
  - Nhân
  - Chia

1 – Đặc tả Giao tiếp từ xa

```
public interface Calculator extends java.rmi.Remote {
    public long add(long a, long b) throws java.rmi.RemoteException;

    public long sub(long a, long b) throws java.rmi.RemoteException;

    public long mul(long a, long b) throws java.rmi.RemoteException;

    public long div(long a, long b) throws java.rmi.RemoteException;
}
```

2 – Thực thi giao tiếp từ xa

```
public class CalculatorImpl extends java.rmi.server.UnicastRemoteObject implements Calculator {

   public CalculatorImpl() throws java.rmi.RemoteException {
        super();
    }
    public long add(long a, long b)  throws java.rmi.RemoteException {
        return a + b;
    }
    public long sub(long a, long b)  throws java.rmi.RemoteException {
        return a - b;
    }
    public long mul(long a, long b) throws java.rmi.RemoteException {
        return a * b;
    }
    public long div(long a, long b) throws java.rmi.RemoteException {
        return a / b;
    }
}
```

3 – Tạo Server

```
public class CalculatorServer {

    public CalculatorServer() {
      try {
        Calculator c = new CalculatorImpl();
        LocateRegistry.createRegistry(1099);
        Naming.rebind("rmi://localhost:1099/CalculatorService", c);
      } catch (Exception e) {
          e.printStackTrace();
        System.out.println("Trouble: " + e);
      }
    }

    public static void main(String args[]) {
      new CalculatorServer();
    }
}
```

4 – Tạo Client

```
public class CalculatorClient {

    public static void main(String[] args) {
        try {
            Calculator c = (Calculator) Naming.lookup("rmi://localhost/CalculatorService");

            System.out.println( c.sub(9, 3) );
            System.out.println( c.add(10, 5) );
            System.out.println( c.mul(3, 6) );
            System.out.println( c.div(9, 3) );
        }
```

Policy

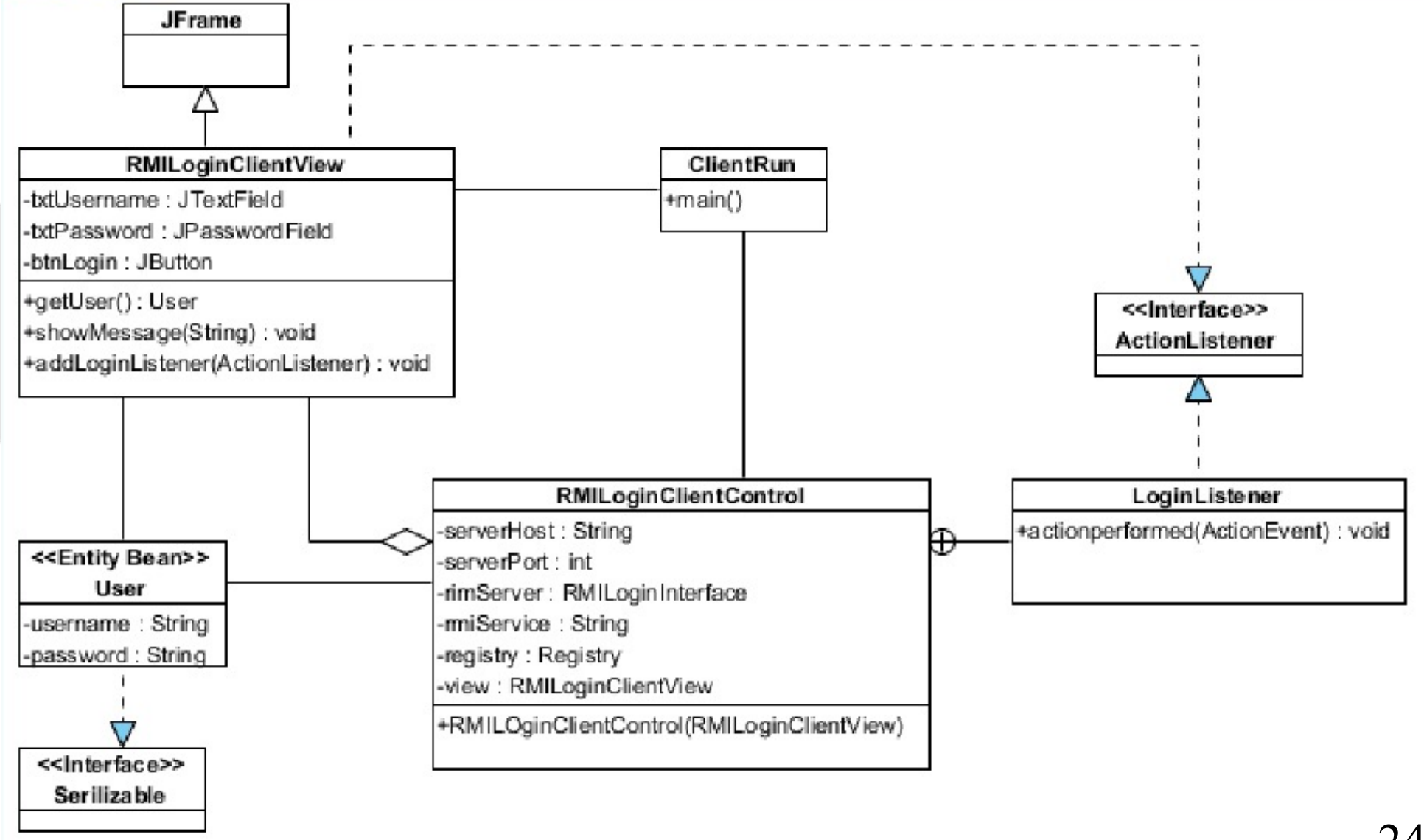
- Với Java tất cả các thao tác kết nối và chép tập tin lạ từ một máy khác về một máy đều phải thông qua lớp bảo vệ: *SecurityManager*
  - Tạo file *client.policy* và *server.policy* trên client và server
    - permission java.security.AllPermission;

Ví dụ:
Login từ xa dùng RMI

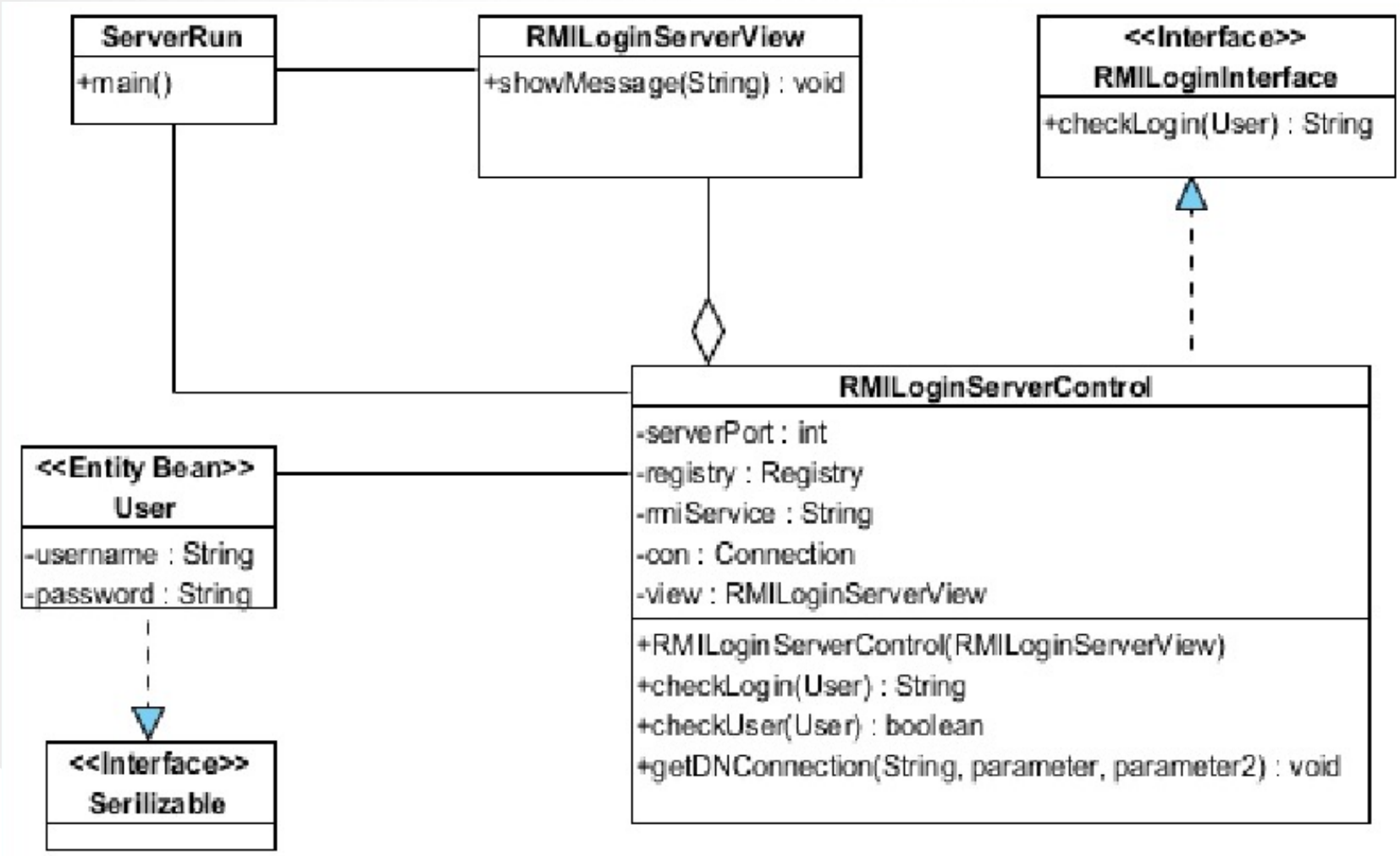
Bài toán: Login dùng RMI

- Thông tin user được lưu trên client.
- 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

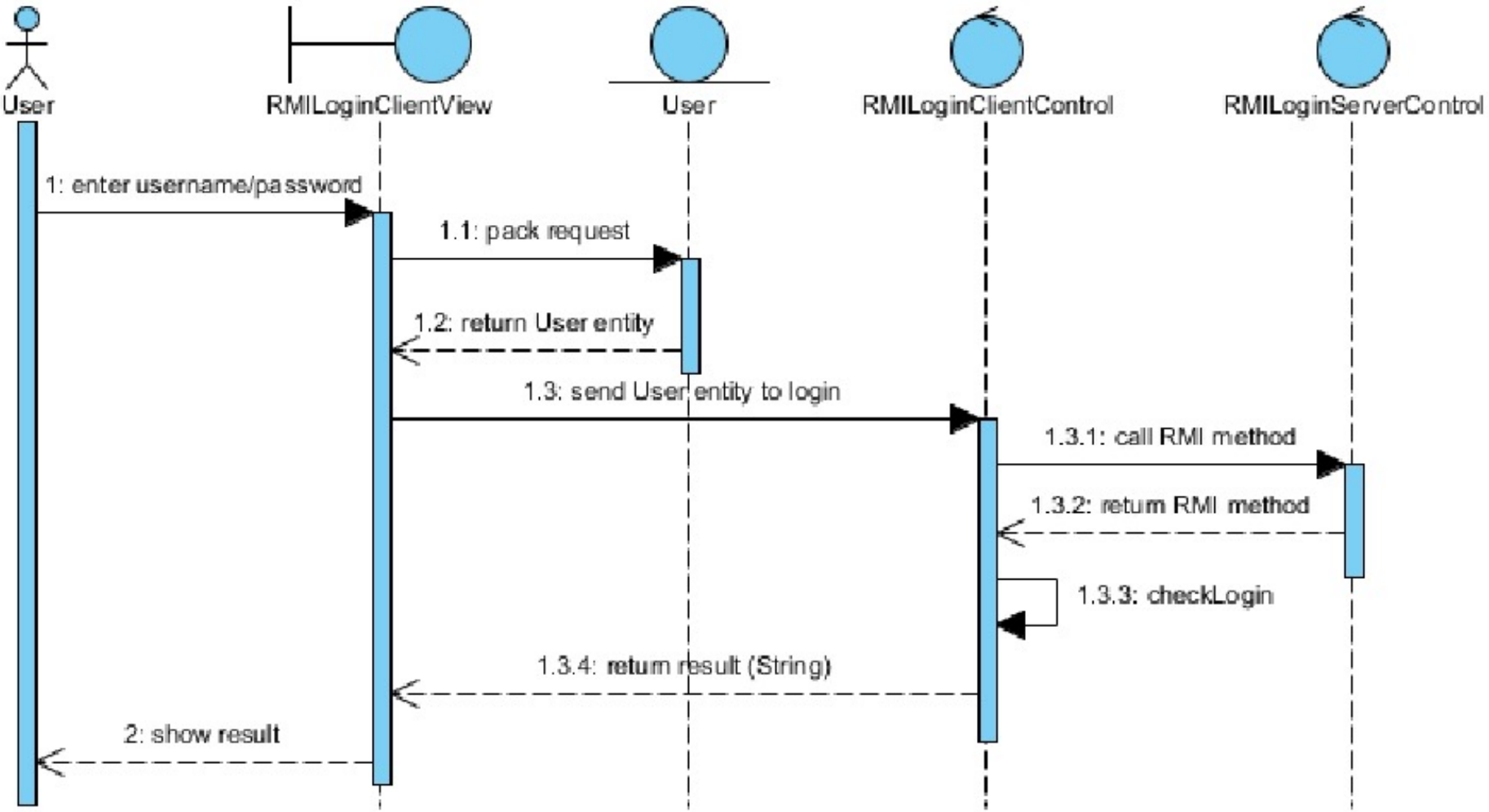
Sơ đồ lớp phía client



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.showMessage("Login succesfully!");
                    }else{
                    view.showMessage("Invalid username and/or
password!");
                    }
                } catch (Exception ex) {
                    view.showMessage(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 = "";
        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.showMessage(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.showMessage(e.getStackTrace().toString());
        }

        return false;
      }
}
```

ServerRun

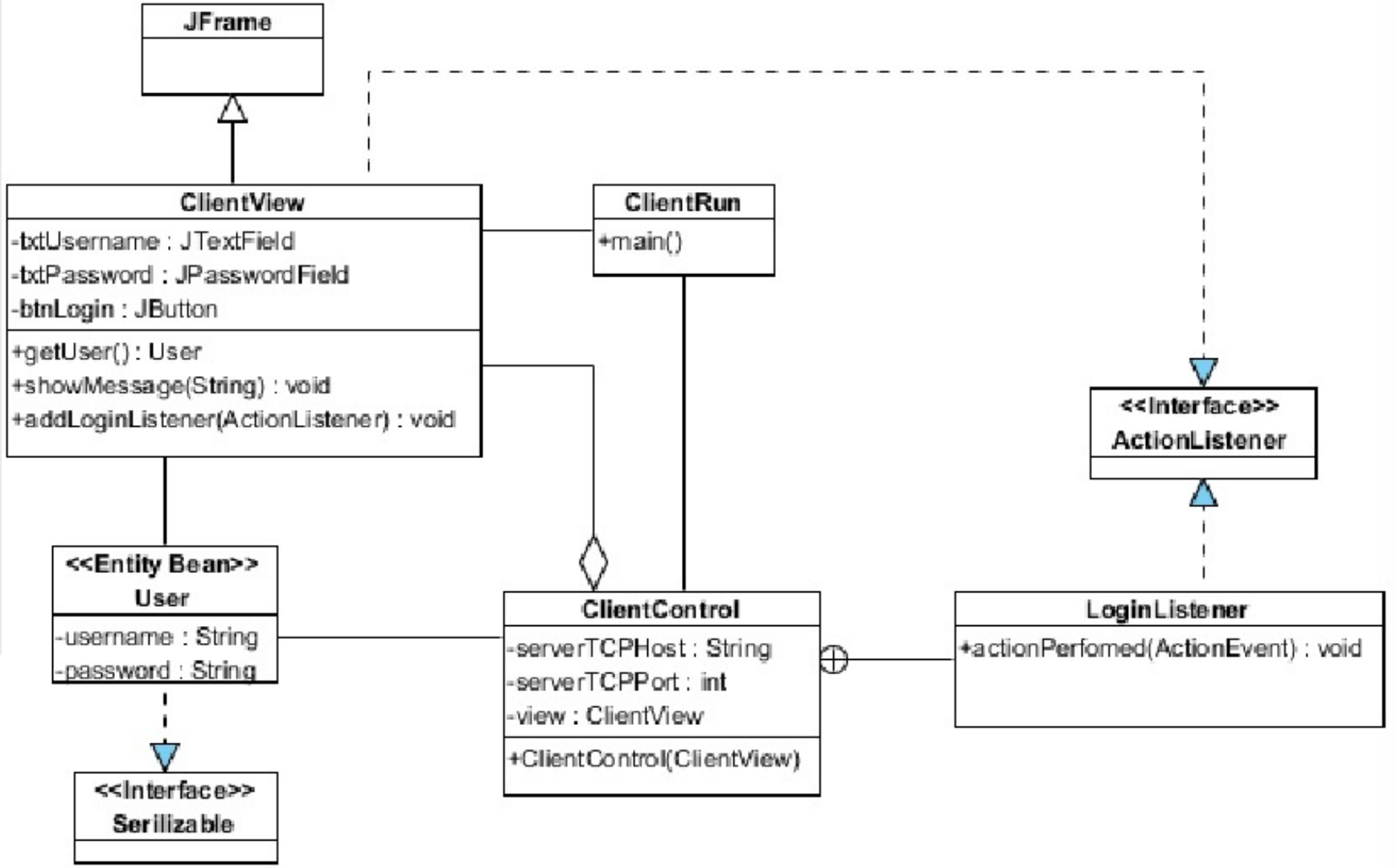
```
public class ServerRun {
    public static void main(String[] args) {
        RMILoginServerView view = new RMILoginServerView();
        try{
            RMILoginServerControl control = new
                RMILoginServerControl(view);
        }catch(Exception e){
            e.printStackTrace();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

}
```

Ví dụ:
Login từ xa dùng TCP/IP-RMI

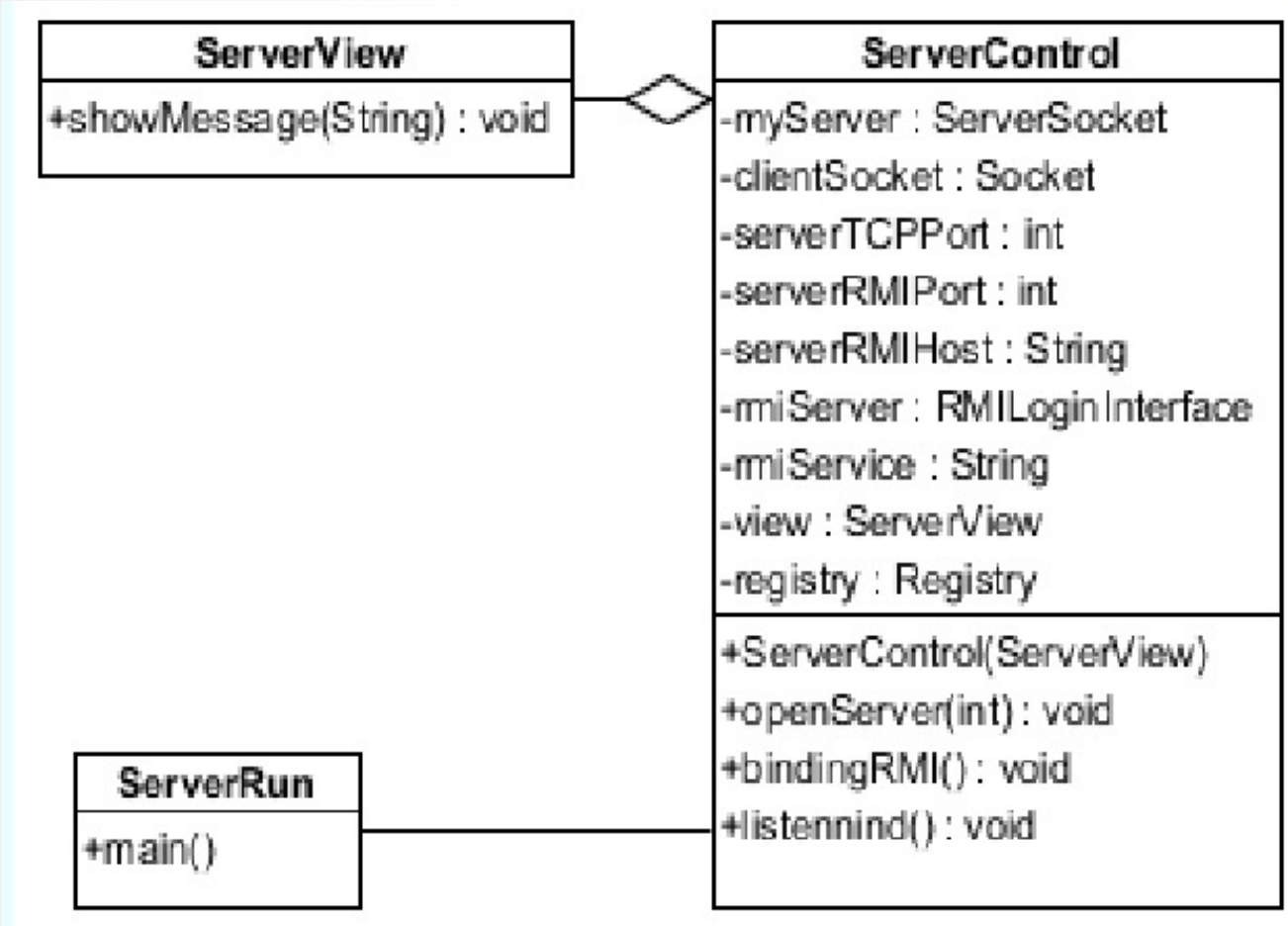
Bài toán

- 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

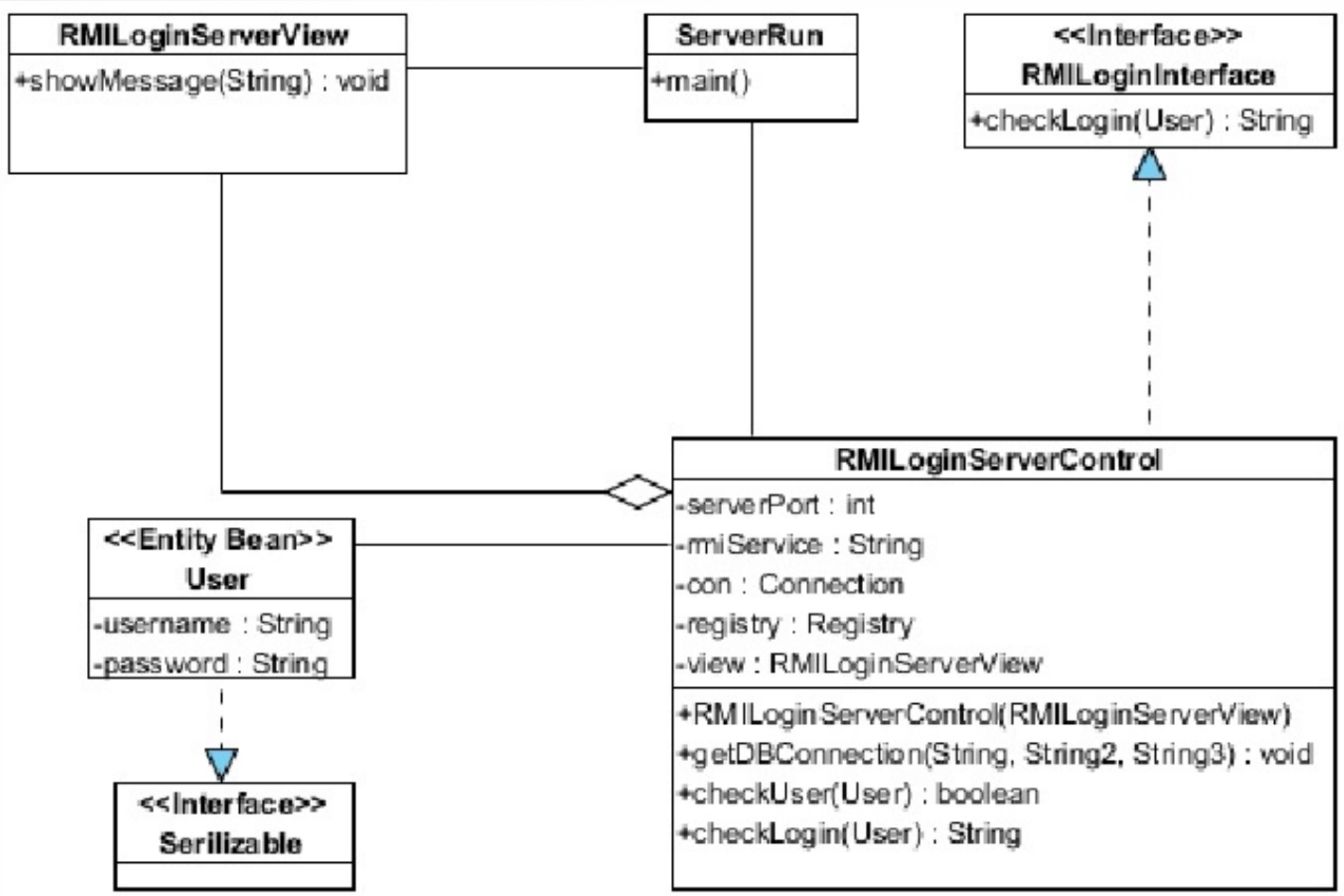
Sơ đồ lớp phía client TCP



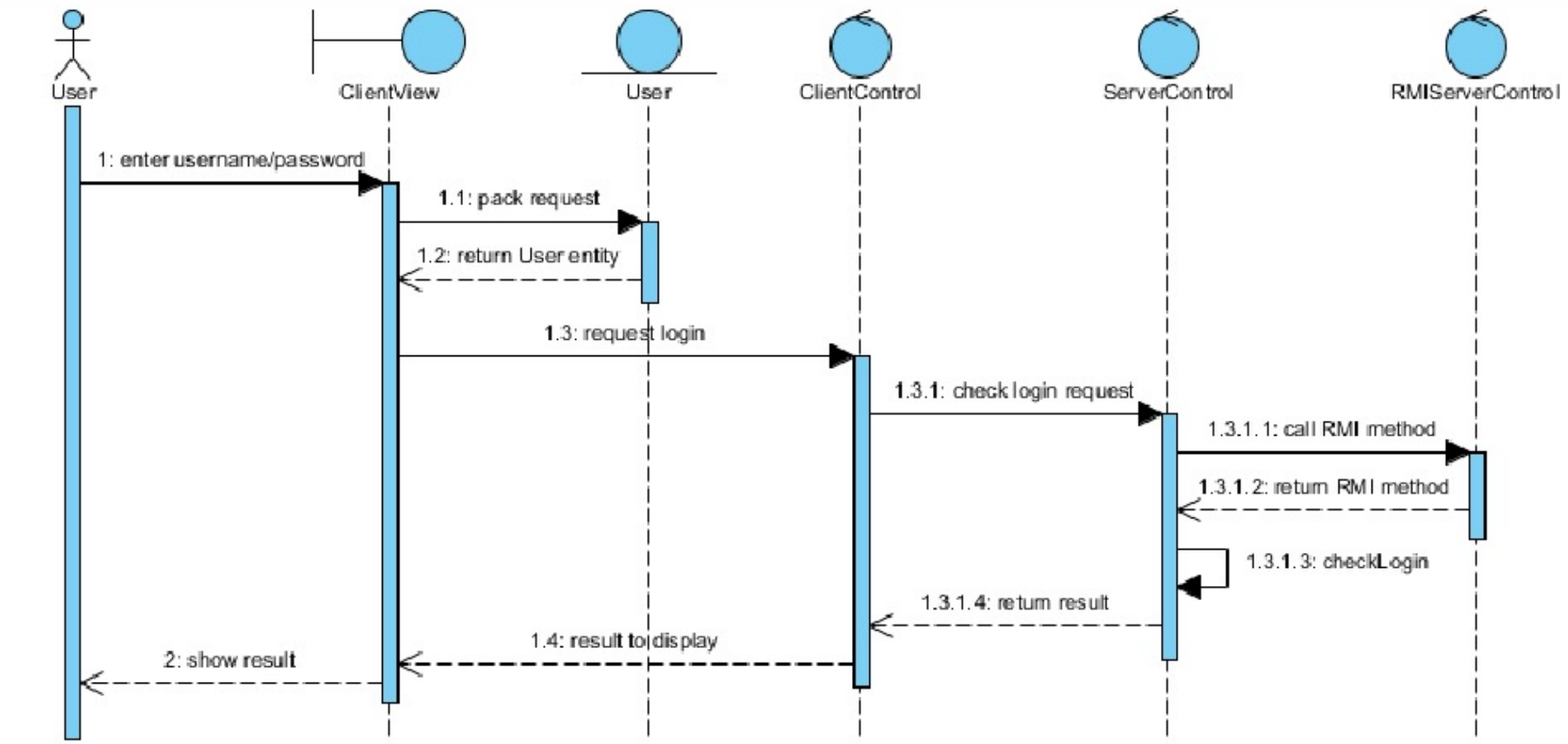
Sơ đồ lớp phía server TCP



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;
    }
}
```

Lớp: 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());
    }
```

Lớp: 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.showMessage("Login succesfully!");
                    else
                        view.showMessage("Invalid username and/or password!");
                }
                mySocket.close();
            } catch (Exception ex) {
                view.showMessage(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.showMessage(e.getStackTrace().toString());
        }catch(NotBoundException e){
                view.showMessage(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.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 = "";
        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.showMessage(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.showMessage(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