Chapter 5.
Network Programming

Content

- Networking Basics
- Working with URL
- Socket
- Remote Method Invocation

1- Networking Basics

- Some definitions related to networking
- Client-Server Model

Definitions

- **Platform**: hardware + operating system.
- **Client**: an application running in a computer (such as browser) can receive data from another (server).
- **Server**: an application running in a computer (such as IIS-Windows Internet Information Service) can supply data to others (clients).
- **IP address** (internet protocol): unsigned integer helps identifying a network element(computer, router,…).
- **IPv4**: 4-byte IP address, such as 192.143.5.1
- **IPv6**: 16-byte IP address
- **Port**: unsigned 2-byte integer helps operating system differentiating a network communicating process.
- **Protocol**: Rules for packaging data of a network communication because client and server can be working in different platform. Two common basic protocols are TCP and UDP

Definitions

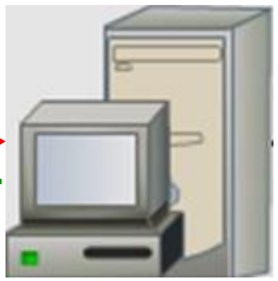
- **TCP**: (*Transmission Control Protocol*) is a connection-based protocol (only one connecting line only) that provides a reliable flow of data between two computers based on the acknowledge mechanism.
- **UDP**: (*User Datagram Protocol*) is a protocol that sends independent packets of data, called datagrams, from one computer to another with no guarantees about arrival (many connecting lines can be used, acknowledge mechanism is not used). Many firewalls and routers have been configured not to allow UDP packets. Ask your system administrator if UDP is permitted.
- **Serialization**: a process that converts object's state (values in fields of the object) to a byte stream.
- **De- serialization**: a process that splits data in a byte stream then set data to fields of an object.

Client-Server Model

Step 1: Client sends a request to server

Step 2: Server analyzes the request then process it

Request can be
(file.html, file.txt
Script file -.asp, .aspx. .php, .jsp….
Execute a method of running object



Client
(can be a browser)

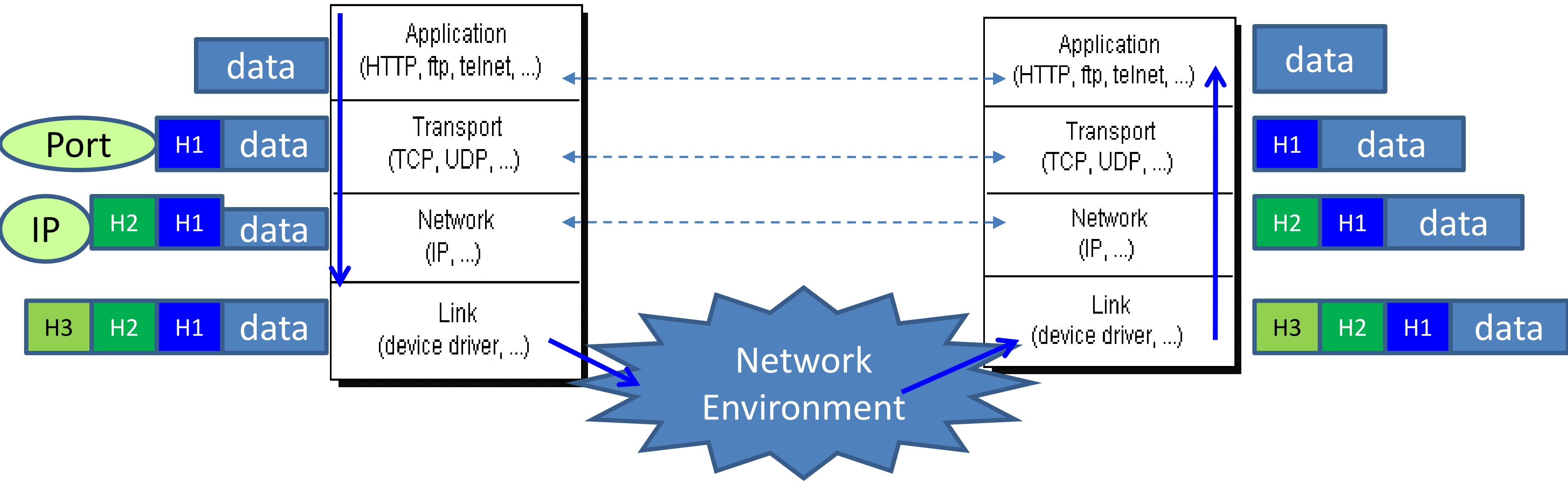
Response:
File.html, .txt,…
Result of processing

Server
(can be a container,
web container or
Application container)

Step 3: Server sends response to client

Client-Server Model: Anatomy

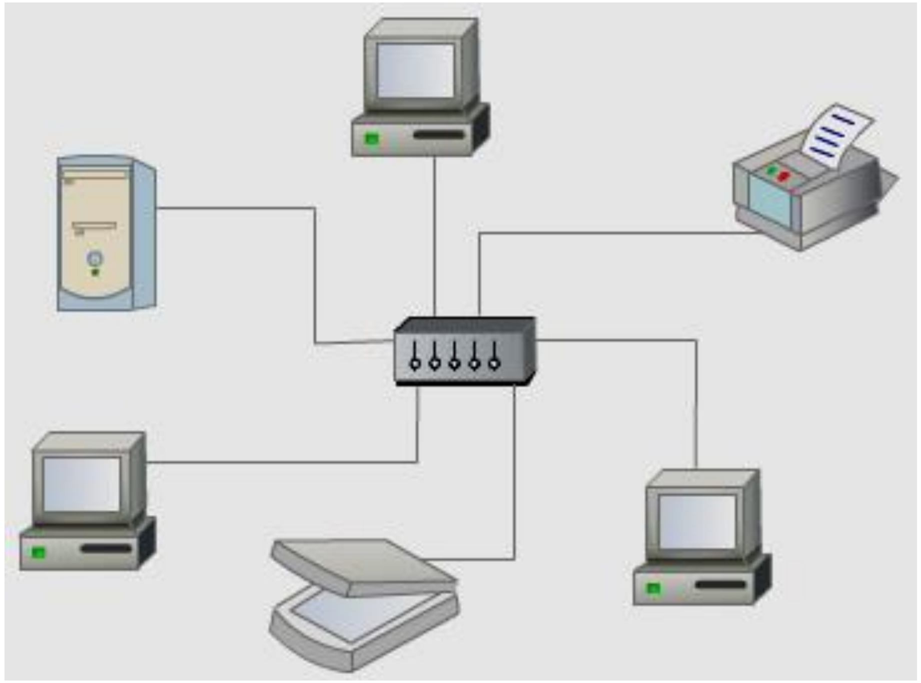
- Computers running on the Internet communicate to each other:



A package is attached an appropriate header (H-identifiable data) when it is transferred to each layer. A layer is an applications or a function library of network managing system

Client-Server Model: Anatomy…

- How to distinguish a computer in a network?



IP:152.3.21.121 or Hostname

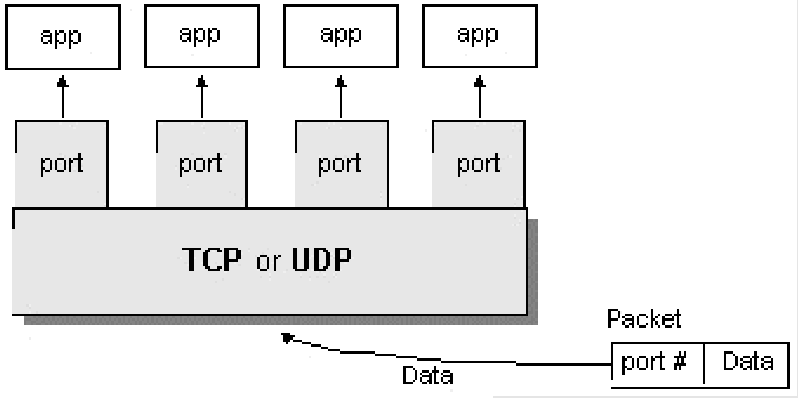
Personal computer IP: 127.0.0.1

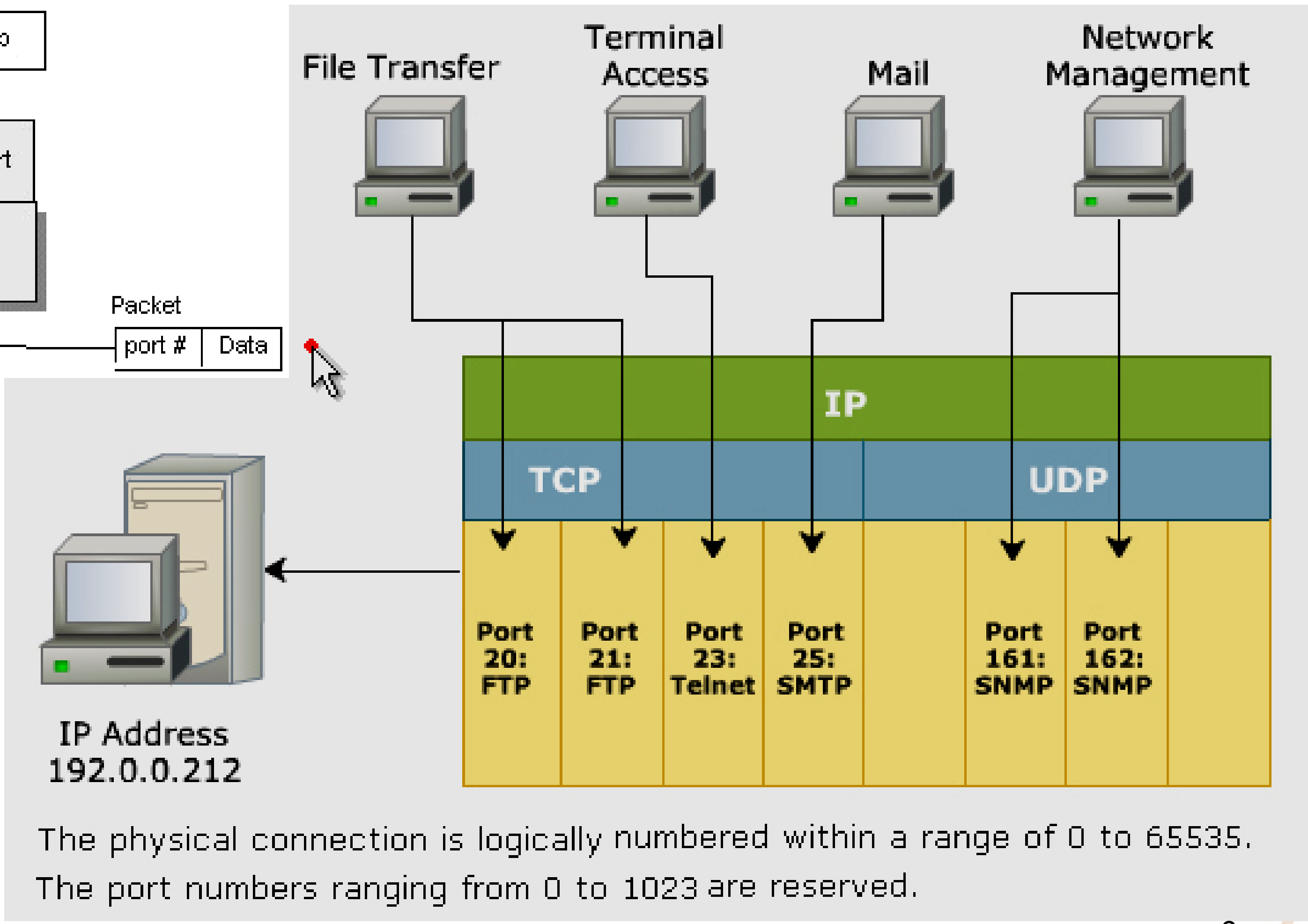
An IP address is either a 32-bit or 128-bit unsigned number used by IP, a lower-level protocol on which protocols like UDP and TCP are built. The IP address architecture is defined by *RFC 790:*

Client-Server Model: Anatomy…

How to distinguish a network-communicating process in a computer?

PORT





The physical connection is logically numbered within a range of 0 to 65535.

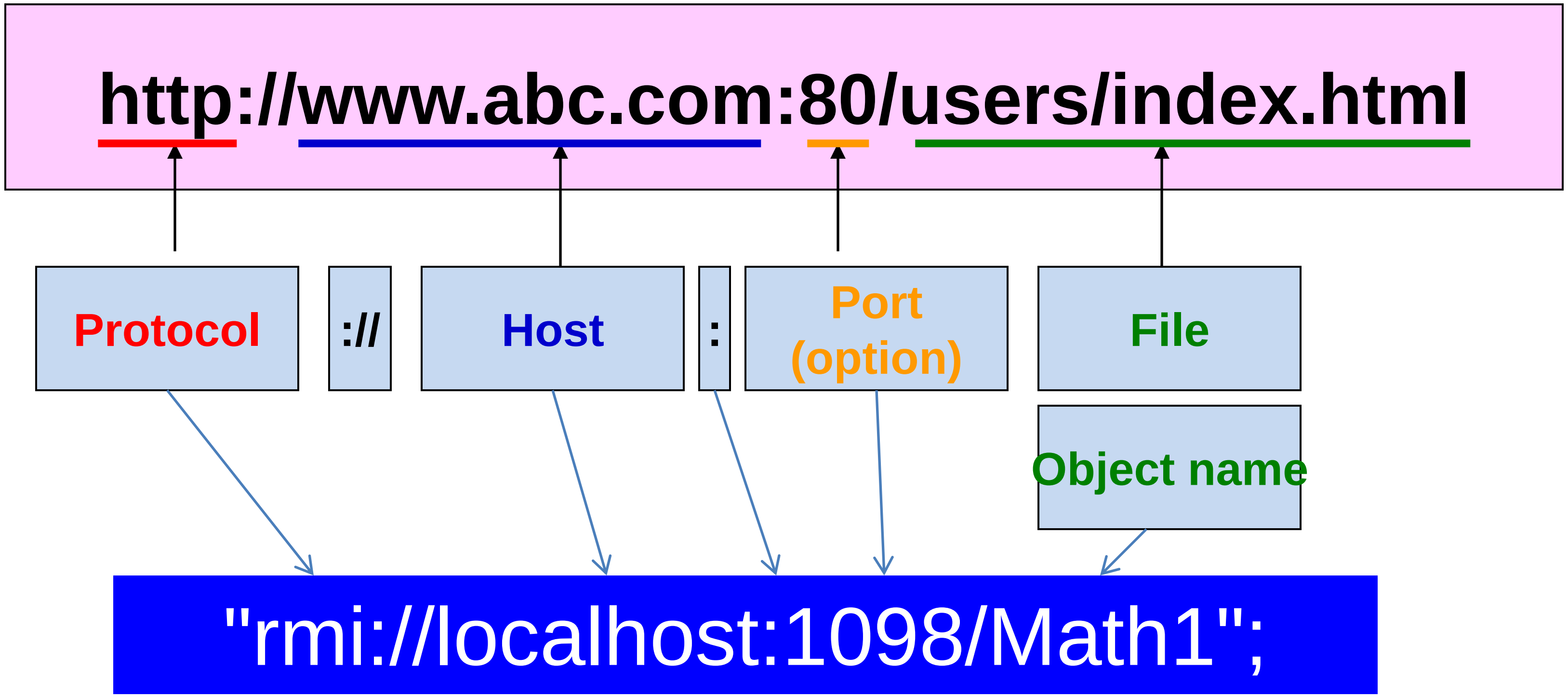
The port numbers ranging from 0 to 1023 are reserved.

Client-Server Model: Anatomy…

How to specify a resource in internet/network?

URL: Uniform Resource Locator

URN: Uniform Resource Name. It involves URL and pathname



http://www.abc.com:80/users/index.html
Protocol
://
Host
:
Port (option)
File
Object name
"rmi://localhost:1098/Math1";

2- Working With URL

- The package ***java.net***
- The class ***java.net.URL***
- Demonstrations for using the URL and URLConnection classes to get contents from urls.

The java.net package

- It contains basic APIs for connecting computer networks.
- Reference: docs-Java8/api/java/net/package-tree.html
- Common used classes:
  - java.net.**URL** (implements java.io.Serializable)
  - java.net.**URLConnection** (abstract class)
    - java.net.**HttpURLConnection**
    - java.net.**JarURLConnection**
  - java.net.**URLDecoder**
  - java.net.**URLEncoder**
  - java.net.**URLStreamHandler**
  - java.net.**ServerSocket** (implements java.io.Closeable)
  - java.net.**Socket** (implements java.io.Closeable)
- This session will introduce the URL only.

The URL Class

- A URL takes the form of a string that describes how to find a resource on the Internet. URLs have two main components: the protocol needed to access the resource and the location of the resource.
- public final class **URL** extends Object implements Serializable
- **Constructors:**

URL(**String** spec) Creates a URL object from the String representation.

URL(**String** protocol, **String** host, int port, **String** file) Creates a URL object from the specified protocol, host, port number, and file.

URL(**String** protocol, **String** host, **String** file) Creates a URL from the specified protocol name, host name, and file name.

…

Demo 1: Parse a URL

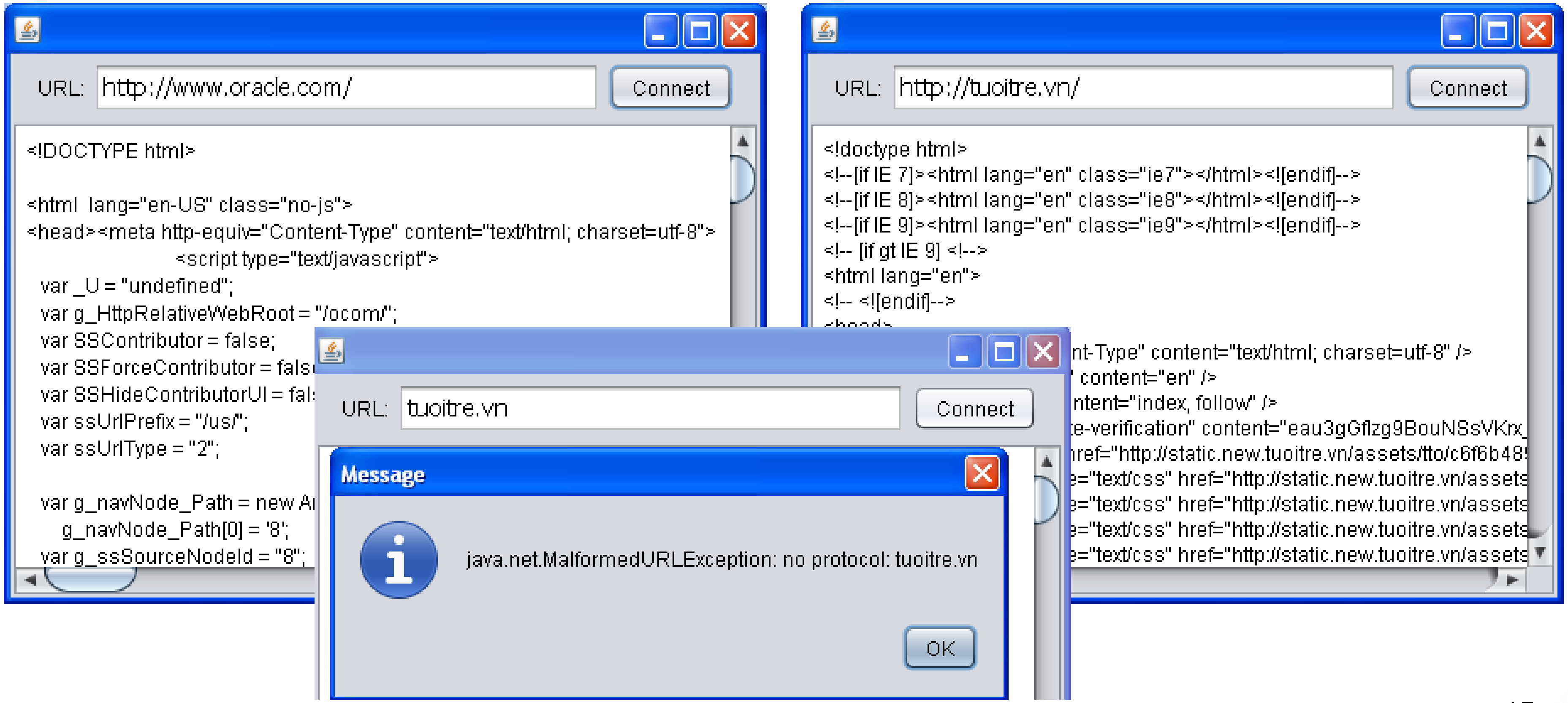
```java
package netPkg;
import java.net.*;
public class ParseURL {
    public static void main(String[] args) throws Exception {

        URL aURL = new URL("http://example.com:80/docs/books/tutorial"
                           + "/index.html?name=networking#DOWNLOADING");
        System.out.println("protocol = " + aURL.getProtocol());
        System.out.println("authority = " + aURL.getAuthority());
        System.out.println("host = " + aURL.getHost());
        System.out.println("port = " + aURL.getPort());
        System.out.println("path = " + aURL.getPath());
        System.out.println("query = " + aURL.getQuery());
        System.out.println("filename = " + aURL.getFile());
        System.out.println("ref = " + aURL.getRef());
    }
}
```

```
run:
protocol = http
authority = example.com:80
host = example.com
port = 80
path = /docs/books/tutorial/index.html
query = name=networking
filename = /docs/books/tutorial/index.html?name=networking
ref = DOWNLOADING
```

- This program will get components in a URL and no connection is carried out.

Demo 2: Read a URL

In the following program, if user enters a URL then clicks the button **Connect**, the content of this URL will be shown. If inputted URL string is malformed, a message will be shown.

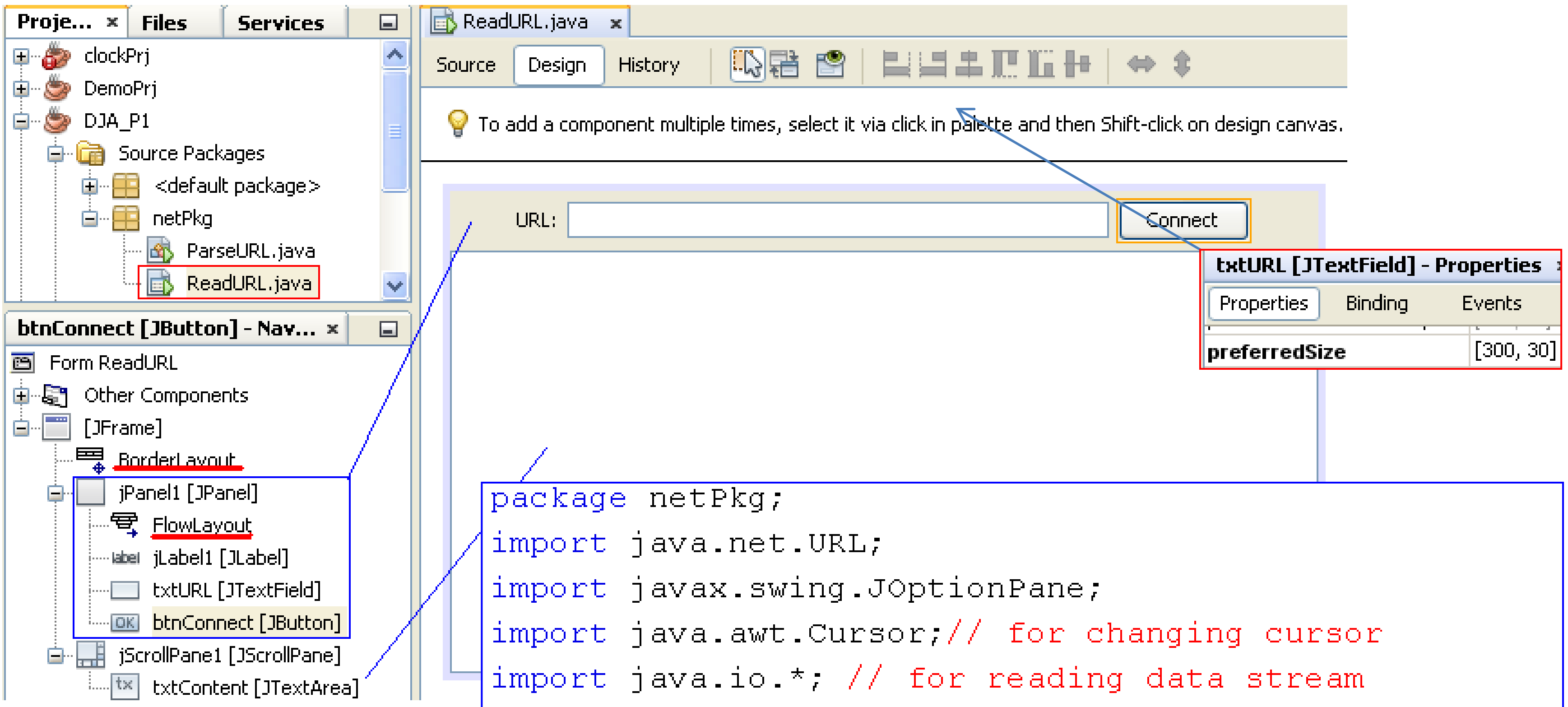


```
<!DOCTYPE html>

<html  lang="en-US" class="no-js">
<head><meta http-equiv="Content-Type" content="text/html; charset=utf-8">
                    <script type="text/javascript">
  var _U = "undefined";
  var g_HttpRelativeWebRoot = "/ocom/";
  var SSContributor = false;
  var SSForceContributor = fals
  var SSHideContributorUI = fal
  var ssUrlPrefix = "/us/";
  var ssUrlType = "2";

  var g_navNode_Path = new A
     g_navNode_Path[0] = '8';
  var g_ssSourceNodeId = "8";
```

```
<!doctype html>
<!--[if IE 7]><html lang="en" class="ie7"></html><![endif]-->
<!--[if IE 8]><html lang="en" class="ie8"></html><![endif]-->
<!--[if IE 9]><html lang="en" class="ie9"></html><![endif]-->
<!-- [if gt IE 9] <!-->
<html lang="en">
<!-- <![endif]-->
<head>
nt-Type" content="text/html; charset=utf-8" />
' content="en" />
ntent="index, follow" />
te-verification" content="eau3gGflzg9BouNSsVKrx_
href="http://static.new.tuoitre.vn/assets/tto/c6f6b48
e="text/css" href="http://static.new.tuoitre.vn/assets
e="text/css" href="http://static.new.tuoitre.vn/assets
e="text/css" href="http://static.new.tuoitre.vn/assets
e="text/css" href="http://static.new.tuoitre.vn/assets
```

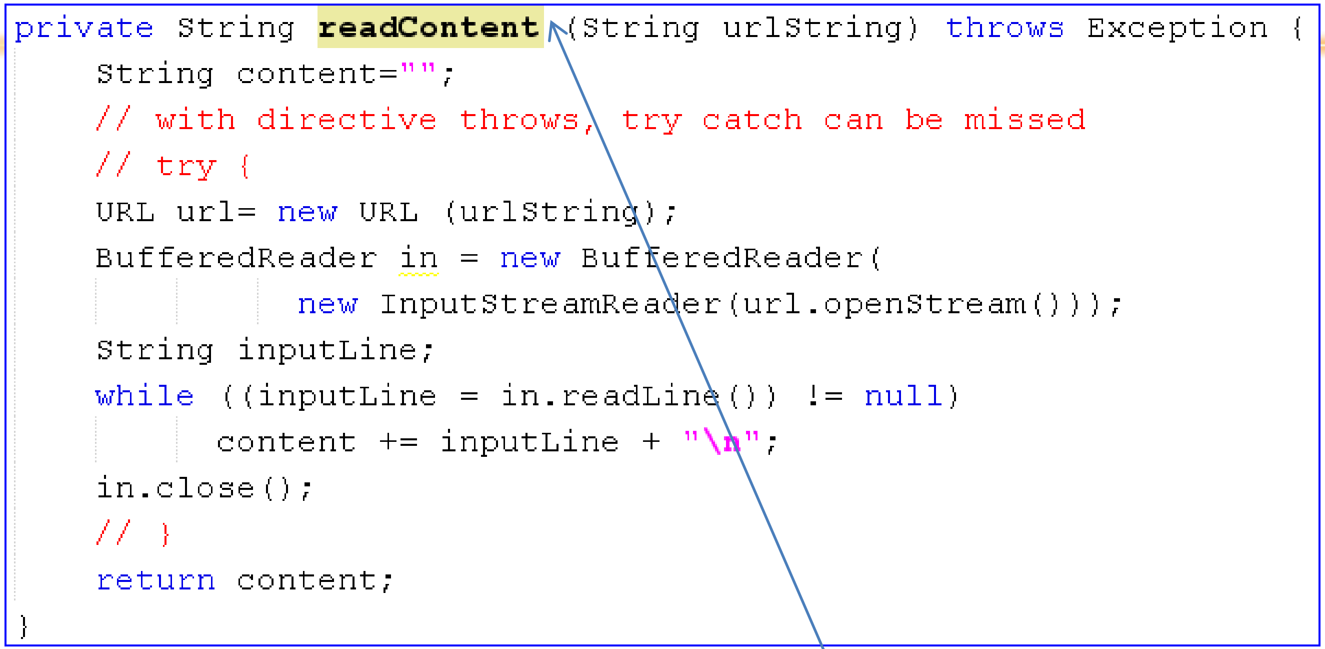
```
java.net.MalformedURLException: no protocol: tuoitre.vn
```

Demo 2: Read a URL...

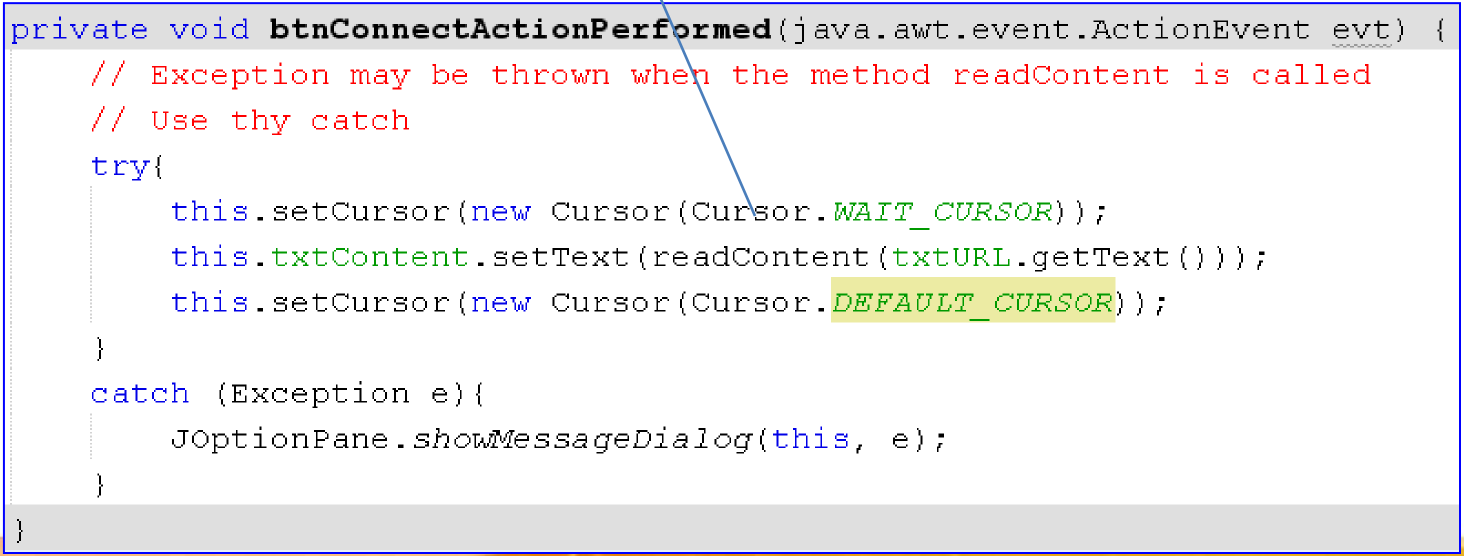


```
package netPkg;
import java.net.URL;
import javax.swing.JOptionPane;
import java.awt.Cursor;// for changing cursor
import java.io.*; // for reading data stream
public class ReadURL extends javax.swing.JFrame {
    /** Creates new form GetURLContents ...3 lines */
    public ReadURL() {
        initComponents();
        this.setSize(450, 300);
    }
```

Demo 2: Read a URL…

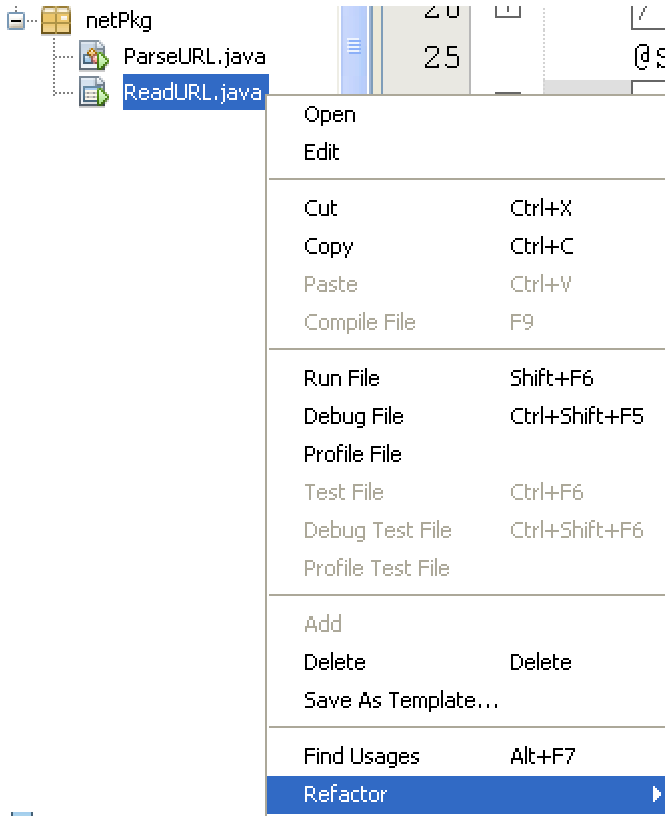


```java
private String readContent (String urlString) throws Exception {
    String content="";
    // with directive throws, try catch can be missed
    // try {
    URL url= new URL (urlString);
    BufferedReader in = new BufferedReader(
                new InputStreamReader(url.openStream()));
    String inputLine;
    while ((inputLine = in.readLine()) != null)
          content += inputLine + "\n";
    in.close();
    // }
    return content;
}
```

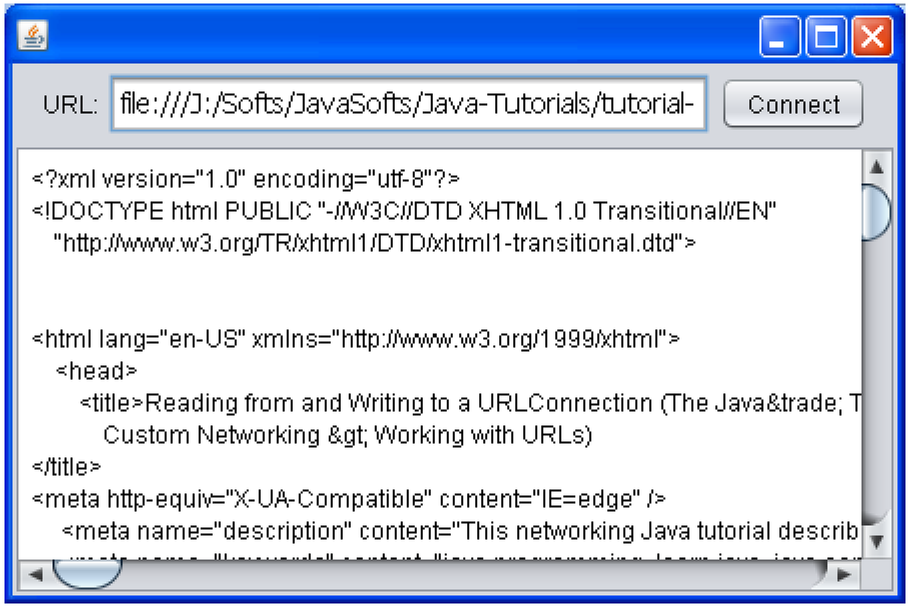


```java
private void btnConnectActionPerformed(java.awt.event.ActionEvent evt) {
    // Exception may be thrown when the method readContent is called
    // Use thy catch
    try{
        this.setCursor(new Cursor(Cursor.WAIT_CURSOR));
        this.txtContent.setText(readContent(txtURL.getText()));
        this.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
    }
    catch (Exception e){
        JOptionPane.showMessageDialog(this, e);
    }
}
```

Demo 3: Using URLConnection



Use the function REFRACTOR of NetBeans to copy and rename the class ReadURL to ReadURLConnection, modify code to gain the similar result as following:

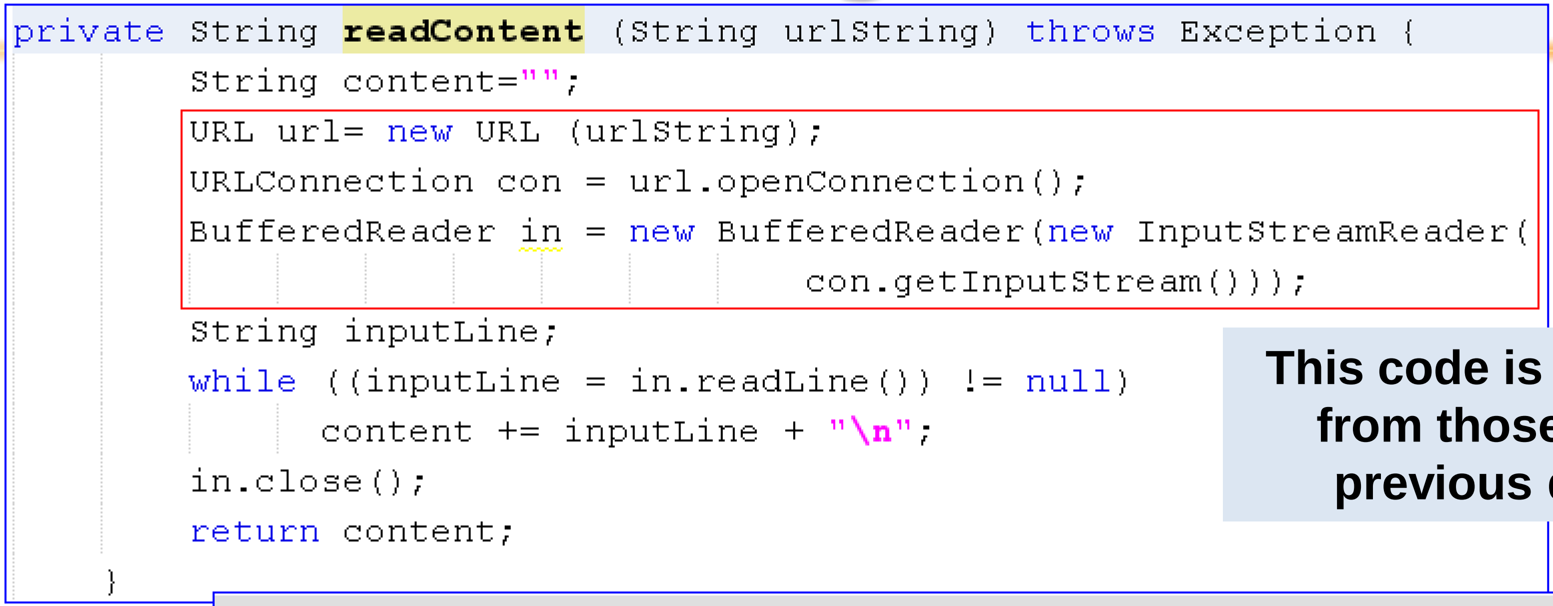


```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">


<html lang="en-US" xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Reading from and Writing to a URLConnection (The Java&trade; T
      Custom Networking &gt; Working with URLs)
</title>
<meta http-equiv="X-UA-Compatible" content="IE=edge" />
   <meta name="description" content="This networking Java tutorial describ
```

Demo 3: Using URLConnection

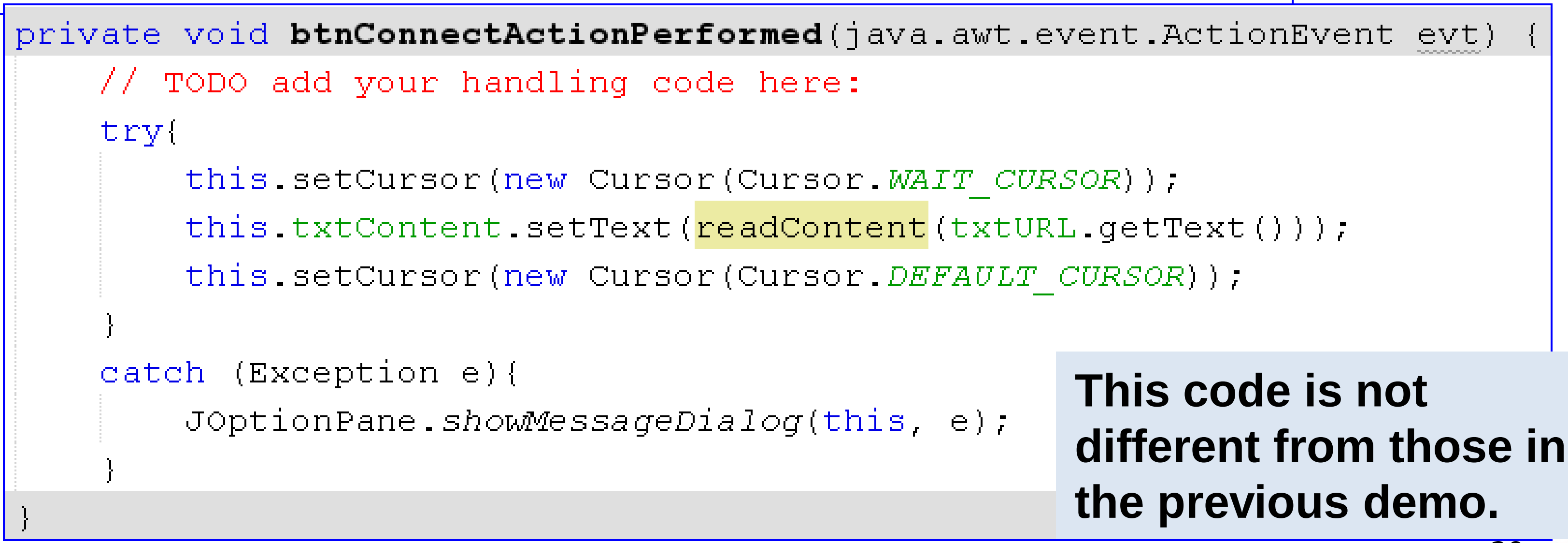
```java
package netPkg;
import java.net.URL;
import java.net.URLConnection;
import java.awt.Cursor;
import javax.swing.JOptionPane;
import java.io.*;
public class ReadURLConnection extends javax.swing.JFrame {
    /** Creates new form GetURLContents ...3 lines */
    public ReadURLConnection() {
        initComponents();
        this.setSize(450, 300);
    }
```

Demo 3: Using URLConnection



```
private String readContent (String urlString) throws Exception {
        String content="";
        URL url= new URL (urlString);
        URLConnection con = url.openConnection();
        BufferedReader in = new BufferedReader(new InputStreamReader(
                                    con.getInputStream()));
        String inputLine;
        while ((inputLine = in.readLine()) != null)
              content += inputLine + "\n";
        in.close();
        return content;
    }
```

This code is modified from those in the previous demo.



```
private void btnConnectActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    try{
        this.setCursor(new Cursor(Cursor.WAIT_CURSOR));
        this.txtContent.setText(readContent(txtURL.getText()));
        this.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
    }
    catch (Exception e){
        JOptionPane.showMessageDialog(this, e);
    }
}
```

This code is not different from those in the previous demo.

3- Java Sockets

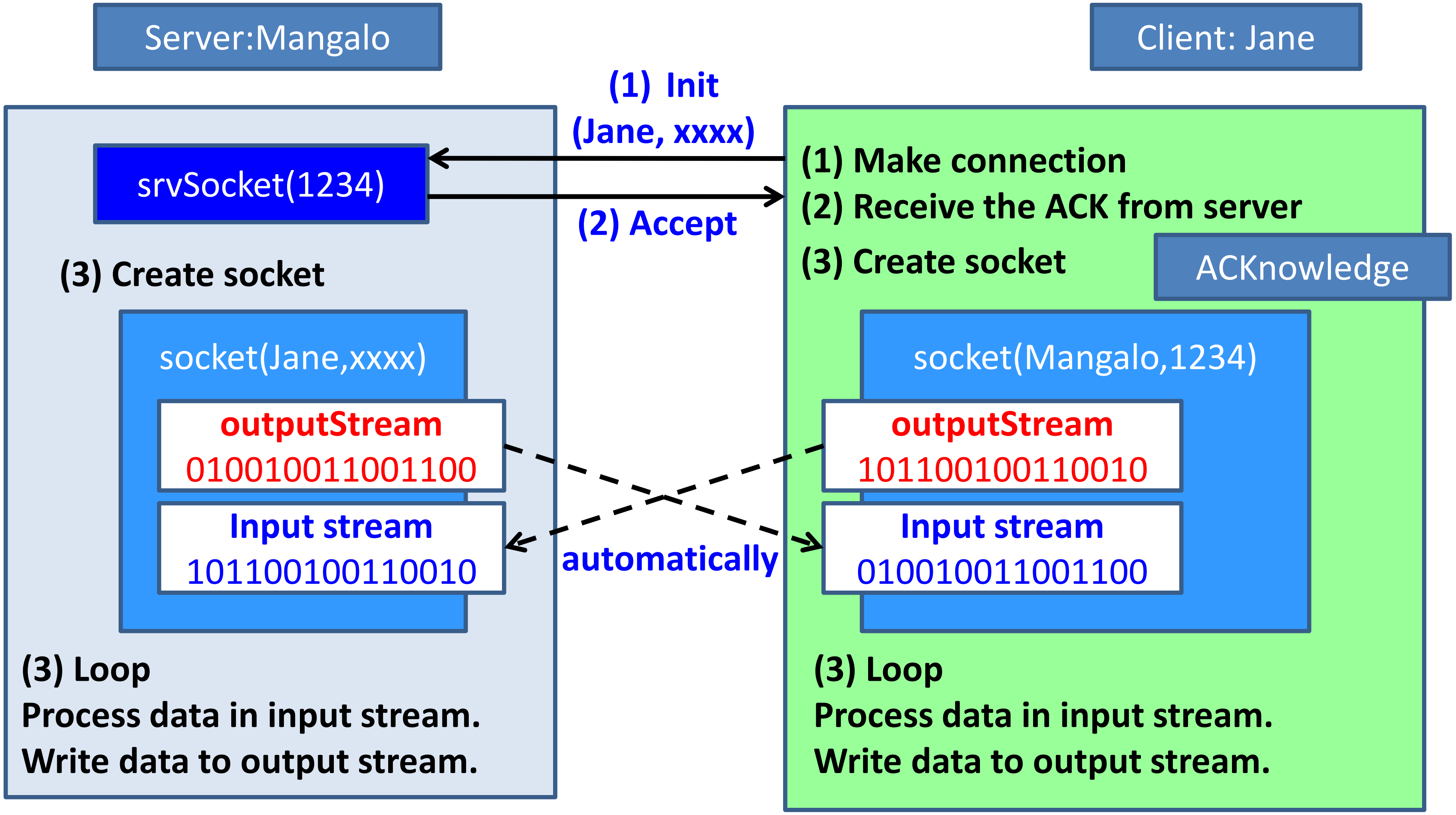
- A *socket* is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent.
- Main members:

  Socket = IP + Port + IO Streams + Methods

Socket and ServerSocket Classes

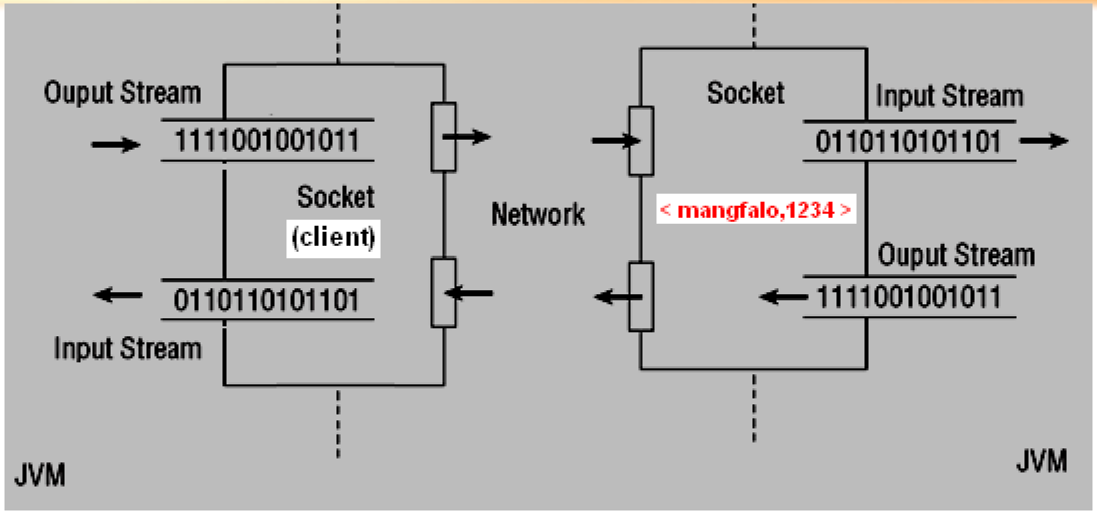
- The java.net.Socket class implements one side of a two-way connection between your Java program and another program on the network
- The java.net.ServerSocket implements server sockets. A server socket waits for requests to come in over the network. It performs some operation based on that request, and then possibly returns a result to the requester.

Sockets: How do they works?



Sockets: How do we code?

Server Program Name: mangfalo or IP



Client Program

```
ServerSocket ss= new
ServerSocket(1234);
Socket clientSoctket=ss.accept();
```

```
Socket  srvSocket= new
Socket("mangfalo",1234);
```

```
// Receive data
bf= new BufferedReader( new InputStreamReader(socket.getInputStream()));
aString= bf.readLine();
// Send data
os= new DataOutputStream (socket.getOutputStream());
os.writeBytes( aString); os.write(13); os.write(10); os.flush();
```

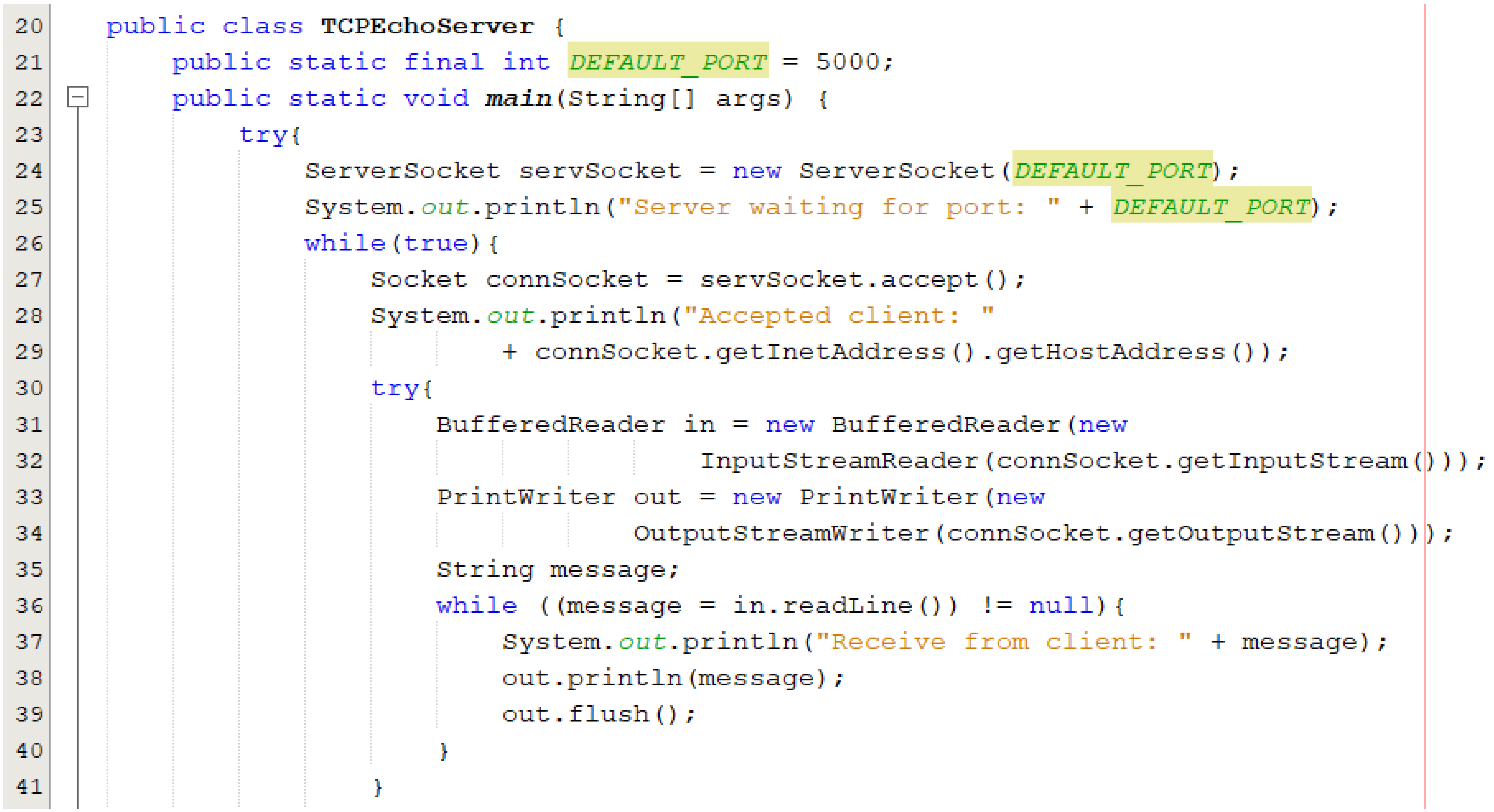
Các bước xây dựng ứng dụng server

- **B1**: Khởi tạo đối tượng ServerSocket để lắng nghe yêu cầu kết nối từ Client.

Sử dụng một vòng lặp để thực hiện các bước từ B2 đến B5:

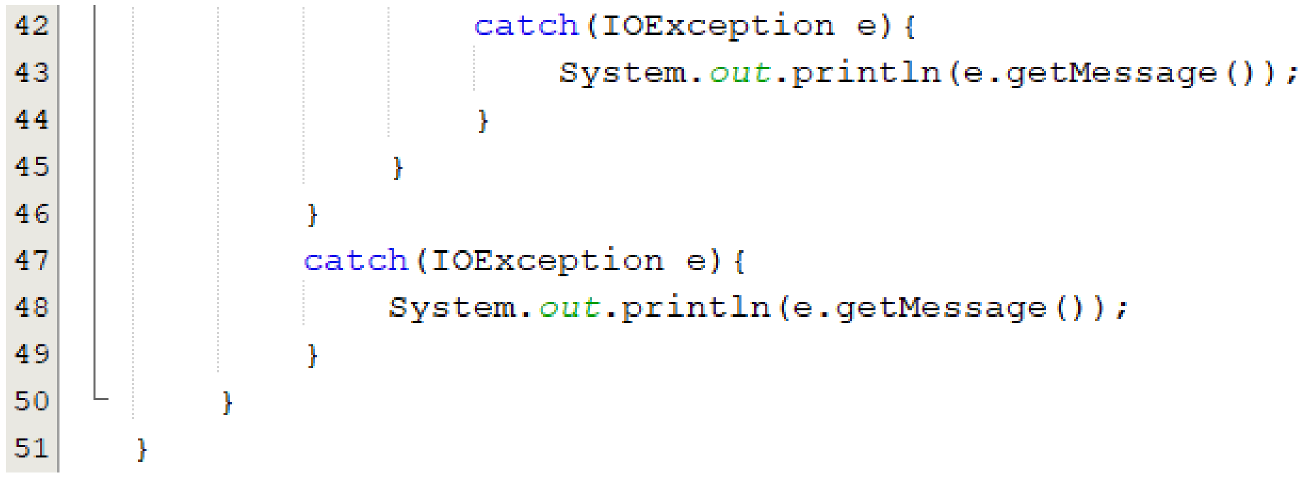
- **B2**: Gọi phương thức accept() để chấp nhận. Phương thức này trả về một đối tượng Socket để trao đổi dữ liệu với client.
- **B3**: Sử dụng phương thức getInputStream() và getOutputStream() lấy các luồng vào-ra để trao đổi dữ liệu với client.
- **B4**: Trao đổi dữ liệu với client.
- **B5**: Đóng Socket kết nối với client.
- **B6**: Đóng ServerSocket.

Các bước xây dựng ứng dụng server



```java
public class TCPEchoServer {
    public static final int DEFAULT_PORT = 5000;
    public static void main(String[] args) {
        try{
            ServerSocket servSocket = new ServerSocket(DEFAULT_PORT);
            System.out.println("Server waiting for port: " + DEFAULT_PORT);
            while(true){
                Socket connSocket = servSocket.accept();
                System.out.println("Accepted client: "
                        + connSocket.getInetAddress().getHostAddress());
                try{
                    BufferedReader in = new BufferedReader(new
                                    InputStreamReader(connSocket.getInputStream()));
                    PrintWriter out = new PrintWriter(new
                                OutputStreamWriter(connSocket.getOutputStream()));
                    String message;
                    while ((message = in.readLine()) != null){
                        System.out.println("Receive from client: " + message);
                        out.println(message);
                        out.flush();
                    }
                }
```

Các bước xây dựng ứng dụng server

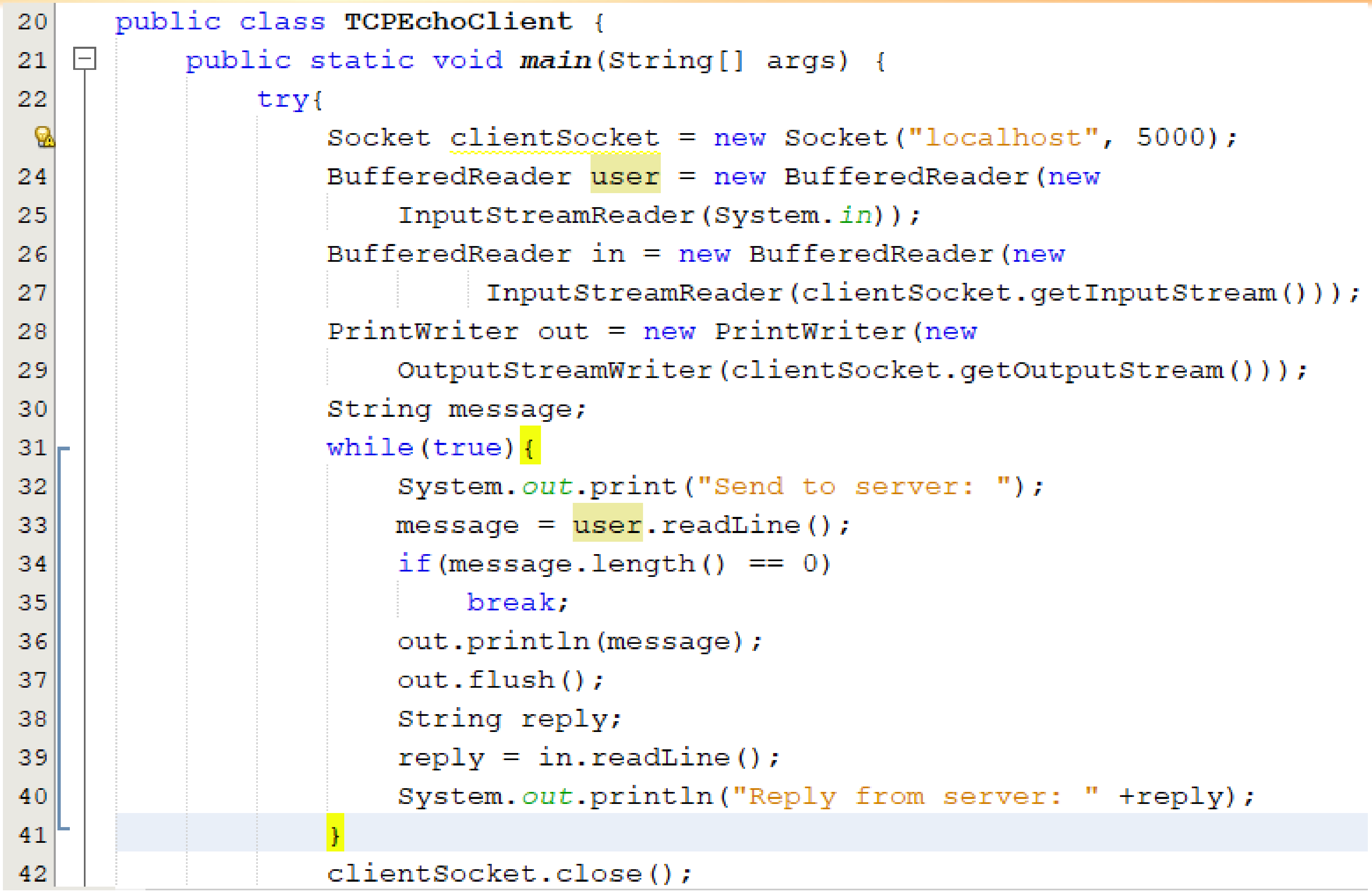


```
                        catch(IOException e){
                            System.out.println(e.getMessage());
                        }
                    }
                }
                catch(IOException e){
                    System.out.println(e.getMessage());
                }
            }
        }
```

Các bước xây dựng ứng dụng client

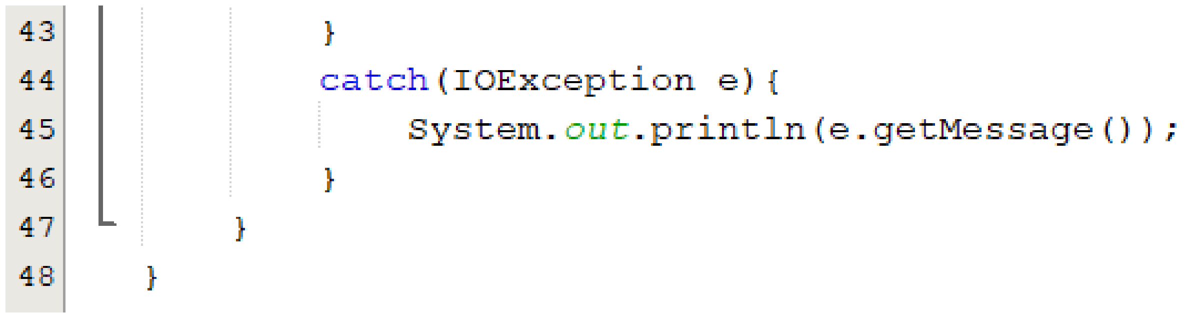
- **B1**: Khởi tạo đối tượng Socket để kết nối đến Server.
- **B2**: Sử dụng phương thức getInputStream() và getOutputStream() lấy các luồng vào-ra để trao đổi dữ liệu với server.
  - Gửi-nhận dữ liệu với Socket giống vào-ra trên file.
  - Cần sử dụng linh hoạt các loại luồng vào-ra trong Java.
- **B4**: Trao đổi dữ liệu với server.
- **B5**: Đóng Socket.

Các bước xây dựng ứng dụng client



```
public class TCPEchoClient {
    public static void main(String[] args) {
        try{
            Socket clientSocket = new Socket("localhost", 5000);
            BufferedReader user = new BufferedReader(new
                InputStreamReader(System.in));
            BufferedReader in = new BufferedReader(new
                     InputStreamReader(clientSocket.getInputStream()));
            PrintWriter out = new PrintWriter(new
                OutputStreamWriter(clientSocket.getOutputStream()));
            String message;
            while(true){
                System.out.print("Send to server: ");
                message = user.readLine();
                if(message.length() == 0)
                    break;
                out.println(message);
                out.flush();
                String reply;
                reply = in.readLine();
                System.out.println("Reply from server: " +reply);
            }
            clientSocket.close();
```

Các bước xây dựng ứng dụng client



```
            }
            catch(IOException e){
                System.out.println(e.getMessage());
            }
        }
    }
```

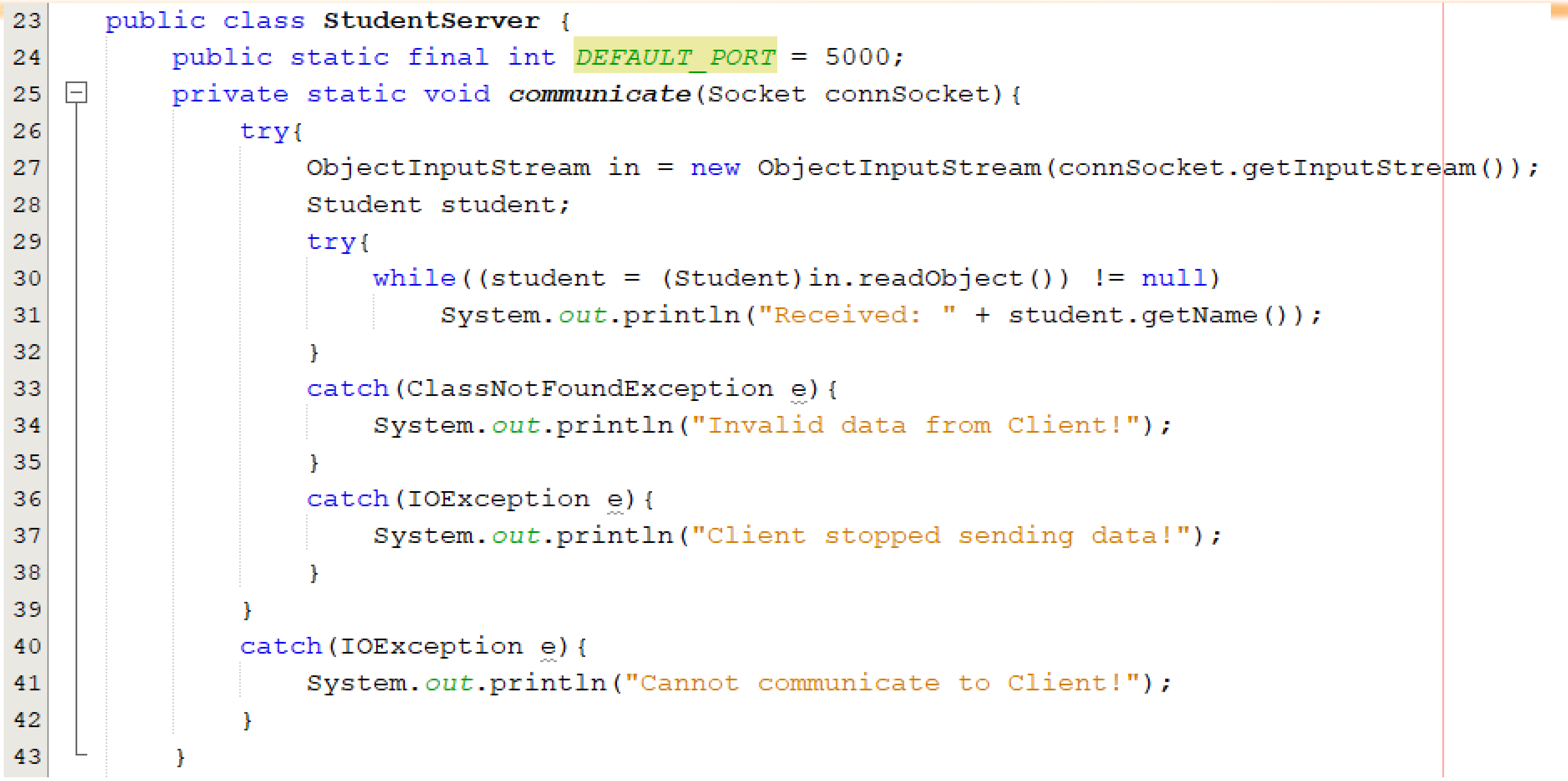
Truyền các đối tượng qua Socket

- Sử dụng luồng ObjectOutputStream để gửi đối tượng qua Socket
  - Phương thức Object writeObject(Object o)
- Sử dụng luồng ObjectInputStream để nhận đối tượng từ socket
  - Phương thức Object readObject()
- Đối tượng truyền trên socket phải thuộc lớp được thực thi từ interface Serializable.

Ví dụ - Student

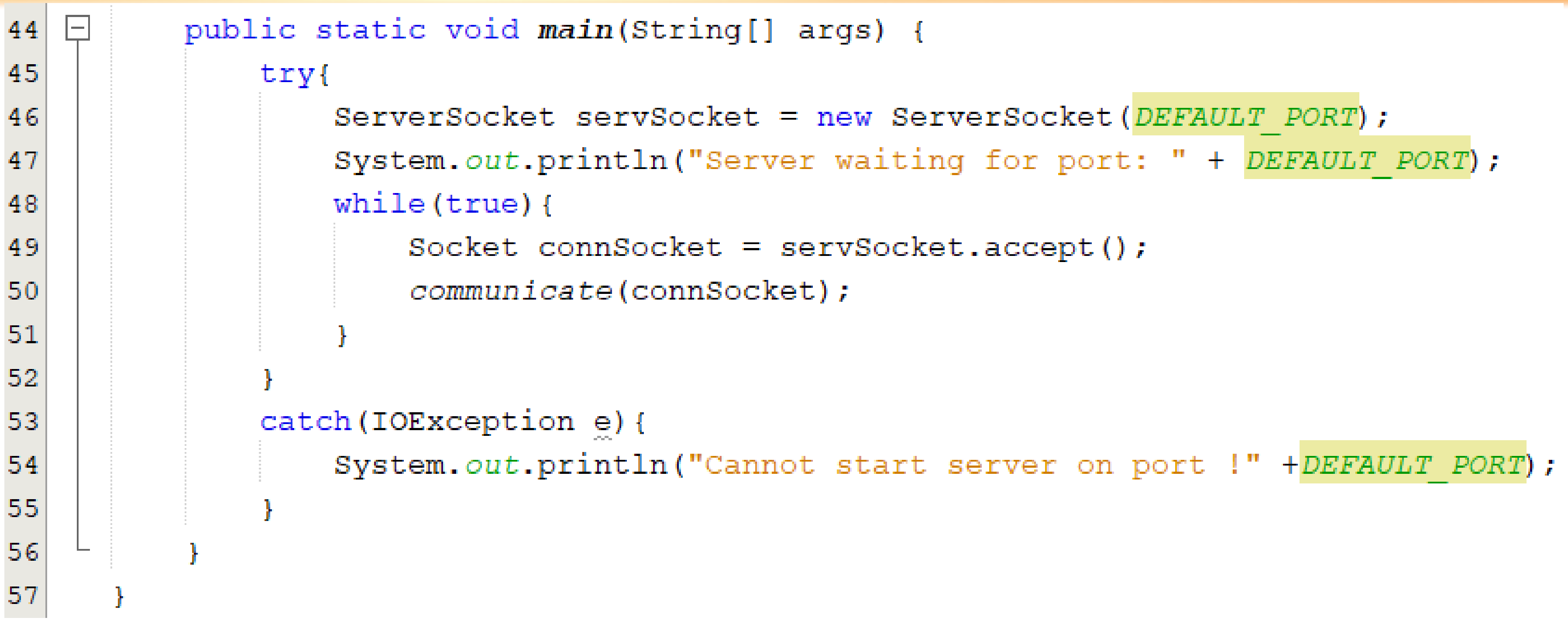
```java
public class Student implements Serializable{
    private String id;
    private String name;
    public Student(String id, String name) {
        this.id = id;
        this.name = name;
    }
    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

Ví dụ - Server



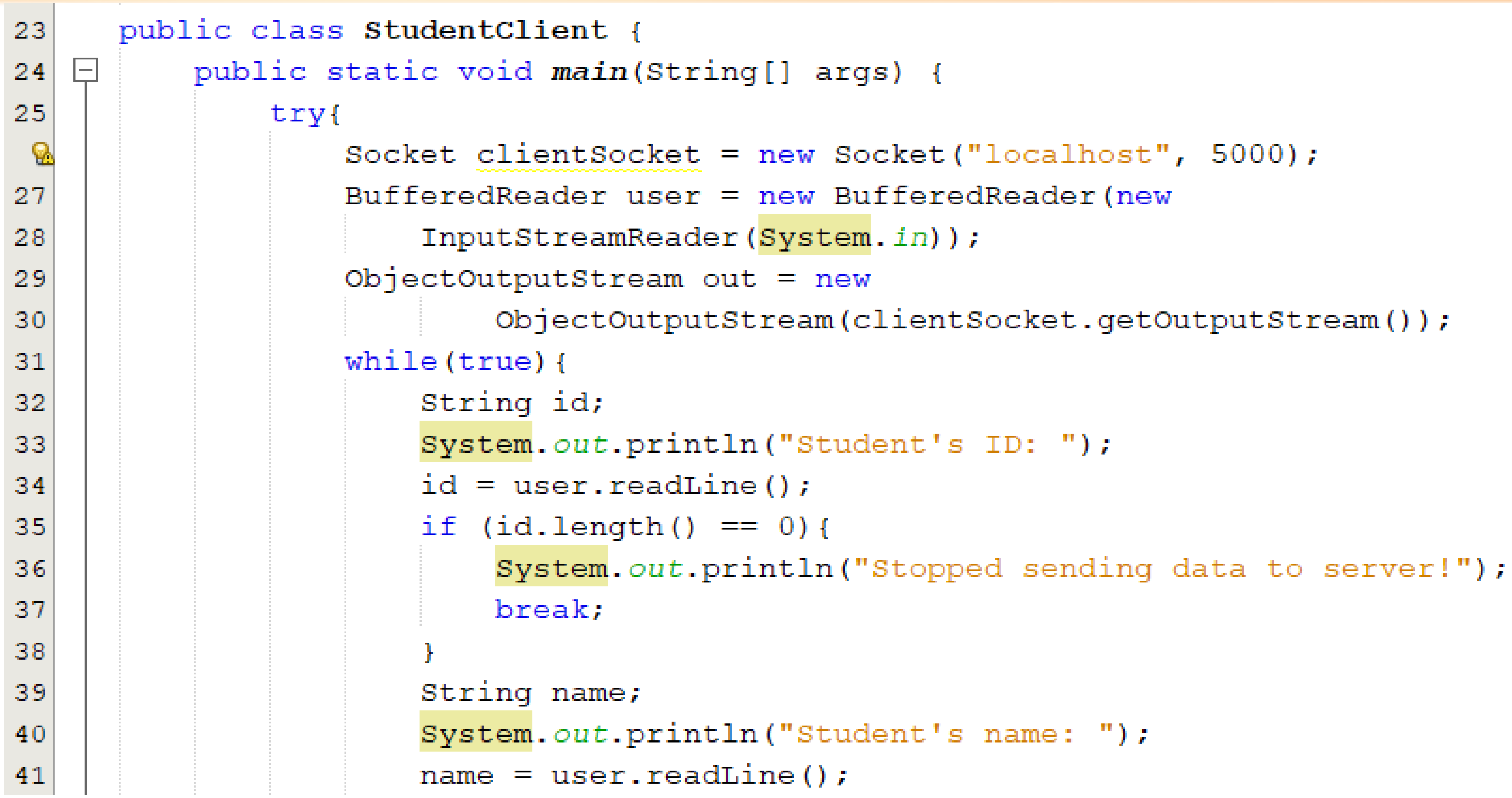
```java
public class StudentServer {
    public static final int DEFAULT_PORT = 5000;
    private static void communicate(Socket connSocket){
        try{
            ObjectInputStream in = new ObjectInputStream(connSocket.getInputStream());
            Student student;
            try{
                while((student = (Student)in.readObject()) != null)
                    System.out.println("Received: " + student.getName());
            }
            catch(ClassNotFoundException e){
                System.out.println("Invalid data from Client!");
            }
            catch(IOException e){
                System.out.println("Client stopped sending data!");
            }
        }
        catch(IOException e){
            System.out.println("Cannot communicate to Client!");
        }
    }
```

Ví dụ - Server



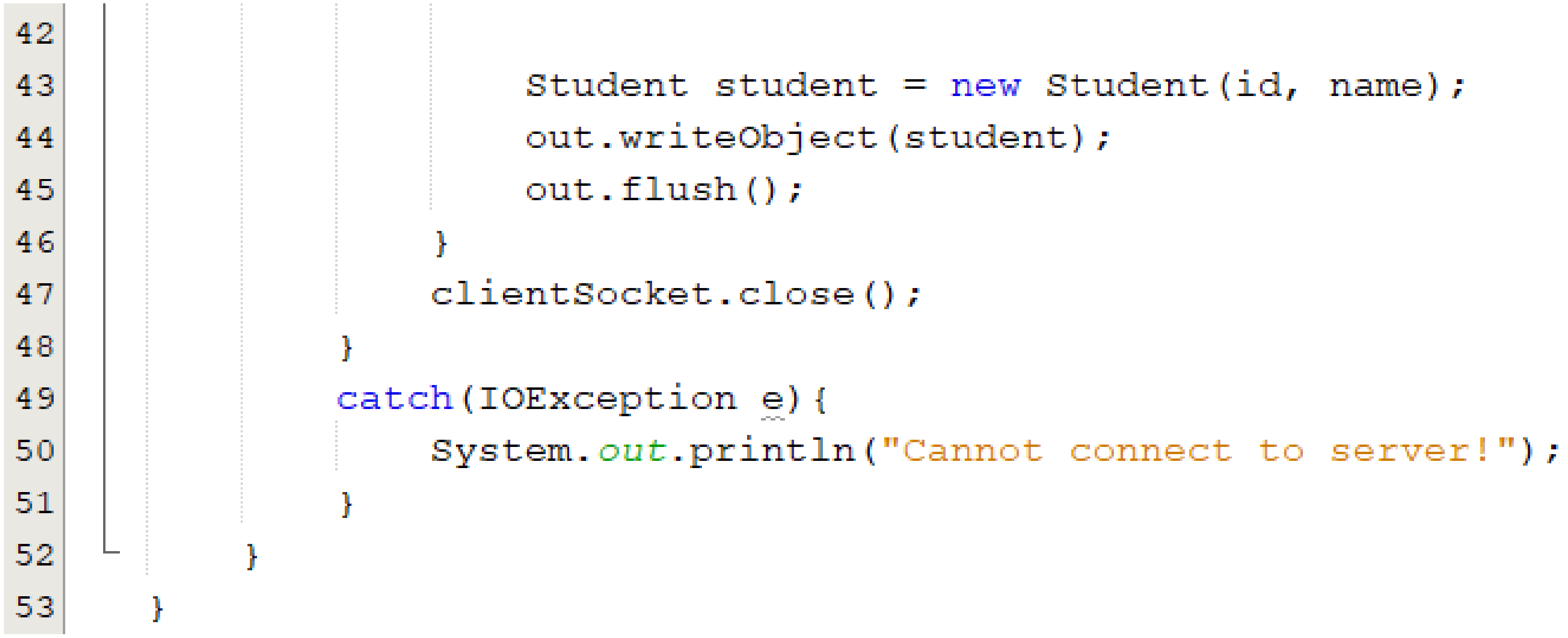
```java
    public static void main(String[] args) {
        try{
            ServerSocket servSocket = new ServerSocket(DEFAULT_PORT);
            System.out.println("Server waiting for port: " + DEFAULT_PORT);
            while(true){
                Socket connSocket = servSocket.accept();
                communicate(connSocket);
            }
        }
        catch(IOException e){
            System.out.println("Cannot start server on port !" +DEFAULT_PORT);
        }
    }
}
```

Ví dụ - Client



```
public class StudentClient {
    public static void main(String[] args) {
        try{
            Socket clientSocket = new Socket("localhost", 5000);
            BufferedReader user = new BufferedReader(new
                InputStreamReader(System.in));
            ObjectOutputStream out = new
                    ObjectOutputStream(clientSocket.getOutputStream());
            while(true){
                String id;
                System.out.println("Student's ID: ");
                id = user.readLine();
                if (id.length() == 0){
                    System.out.println("Stopped sending data to server!");
                    break;
                }
                String name;
                System.out.println("Student's name: ");
                name = user.readLine();
```

Ví dụ - Client



```java

                Student student = new Student(id, name);
                out.writeObject(student);
                out.flush();
            }
            clientSocket.close();
        }
        catch(IOException e){
            System.out.println("Cannot connect to server!");
        }
    }
}
```

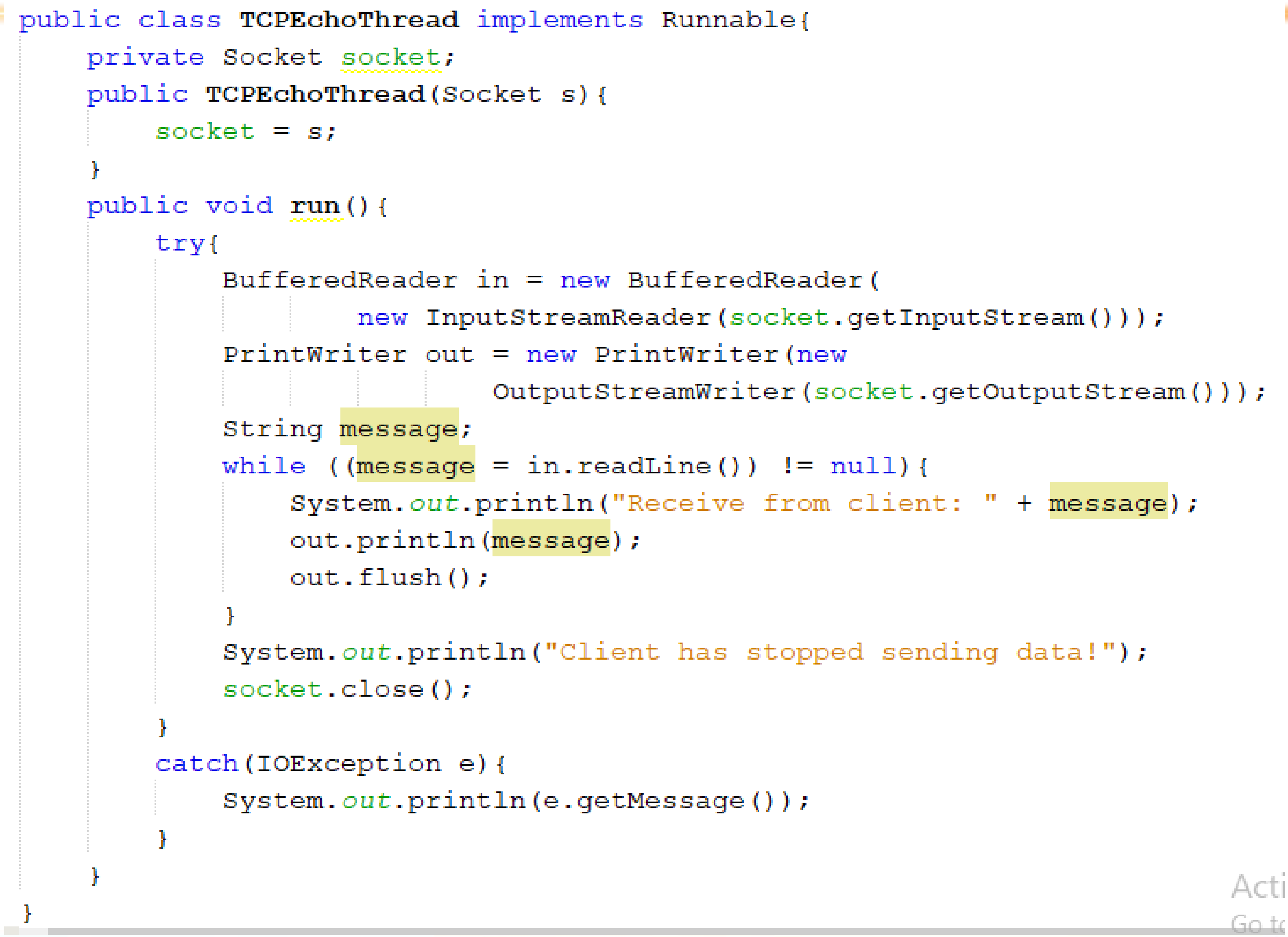
Đa luồng trong lập trình socket

- Với các ví dụ trên, một server tại một thời điểm chỉ xử lý một client

  → Không thể đáp ứng nhiều yêu cầu cùng một lúc.

- Sử dụng đa luồng (***multithread***) để khắc phục nhược điểm trên
  - Khi có một client kết nối đến server (accept)
  - Tạo ra một luồng để xử lý công việc với client đó.

Ví dụ - TCPEchoMultiThread



```java
public class TCPEchoThread implements Runnable{
    private Socket socket;
    public TCPEchoThread(Socket s){
        socket = s;
    }
    public void run(){
        try{
            BufferedReader in = new BufferedReader(
                    new InputStreamReader(socket.getInputStream()));
            PrintWriter out = new PrintWriter(new
                        OutputStreamWriter(socket.getOutputStream()));
            String message;
            while ((message = in.readLine()) != null){
                System.out.println("Receive from client: " + message);
                out.println(message);
                out.flush();
            }
            System.out.println("Client has stopped sending data!");
            socket.close();
        }
        catch(IOException e){
            System.out.println(e.getMessage());
        }
    }
}
```

Ví dụ - TCPEchoServerMultiThread

```java
public class TCPEchoServerMultiThread {
    public static final int DEFAULT_PORT = 5000;
    public static void main(String[] args) {
        try{
            ServerSocket servSocket = new ServerSocket(DEFAULT_PORT);
            System.out.println("Server waiting for port: " + DEFAULT_PORT);
            while(true){
                Runnable t = new TCPEchoThread(servSocket.accept());
                new Thread(t).start();
            }
        }
        catch(IOException e){
            System.out.println(e.getMessage());
        }
    }
}
```

Java TCP Socket Demo

Problem

- ➢ A manager wants to communicate with his/her staffs. Each communication is distinct from others.
- ➢ Write Java programs which allow this manager and his/her staffs carrying out their communications.

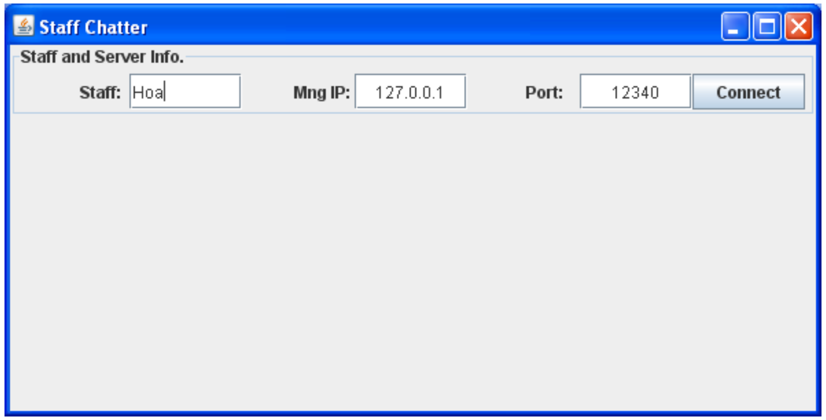
➔Each client program is used by a staff.

➔The program used by the manager has some threads, each thread supports a communication between the manager and a staff.

➔Manager site is specified by IP and the port 12340.

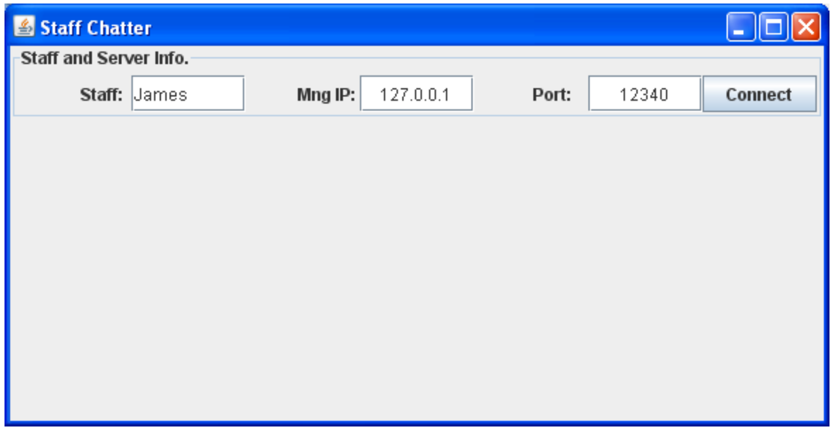
Java TCP Socket Demo.: GUIs

2 staffs and manager – when no connection is established



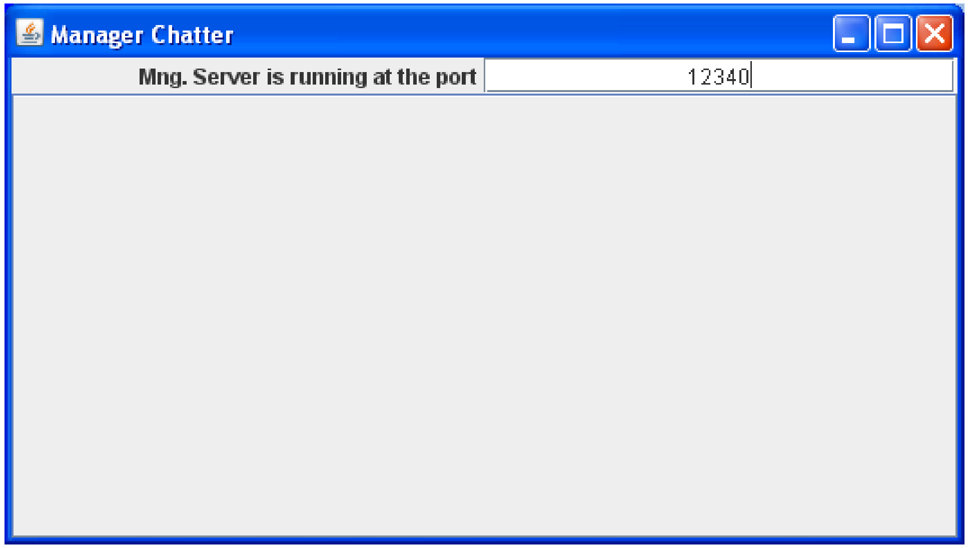
Staff and Server Info.

Staff: Hoa   Mng IP: 127.0.0.1   Port: 12340



Staff and Server Info.

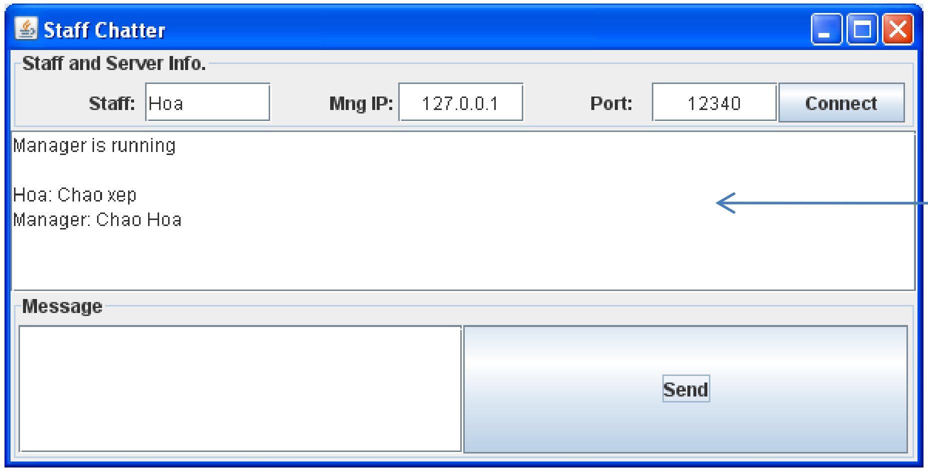
Staff: James   Mng IP: 127.0.0.1   Port: 12340



Mng. Server is running at the port 12340

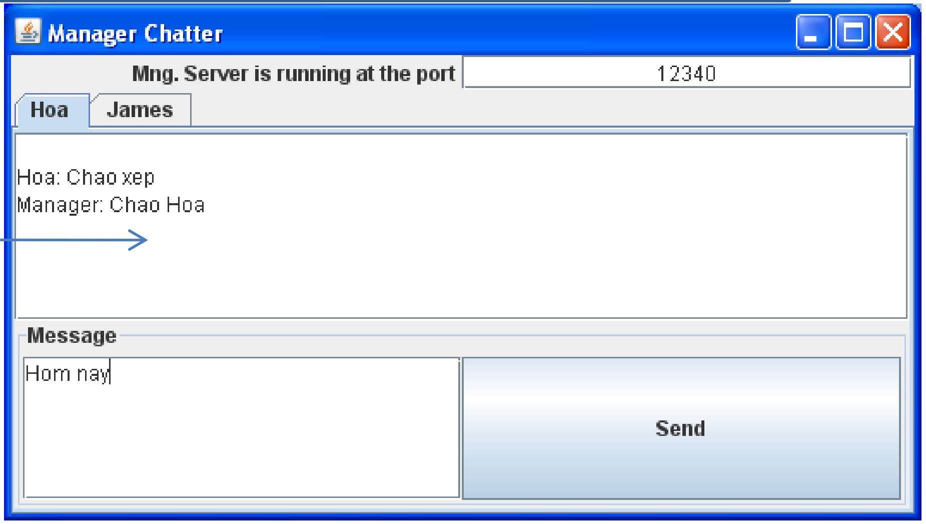
Java TCP Socket Demo.: GUIs

2 staffs and manager – when 2 connections is established



Hoa 127.0.0.1 12340

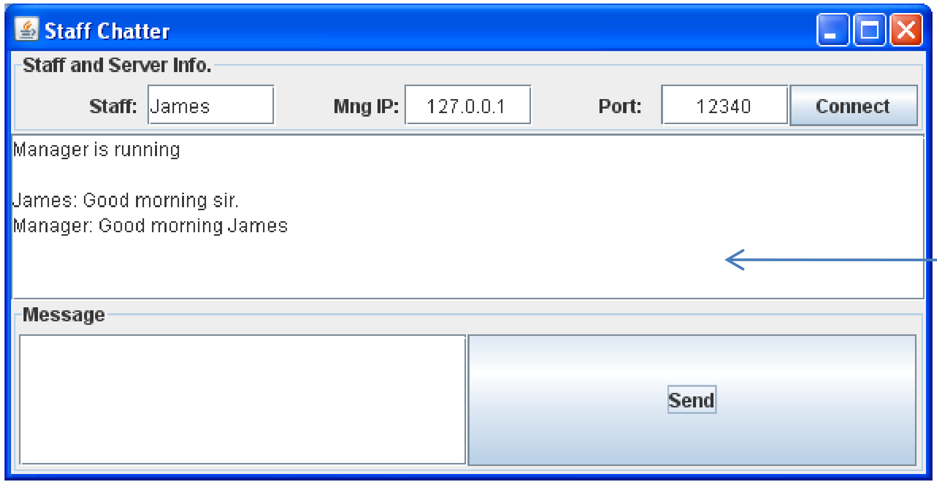
```
Manager is running

Hoa: Chao xep
Manager: Chao Hoa
```



12340

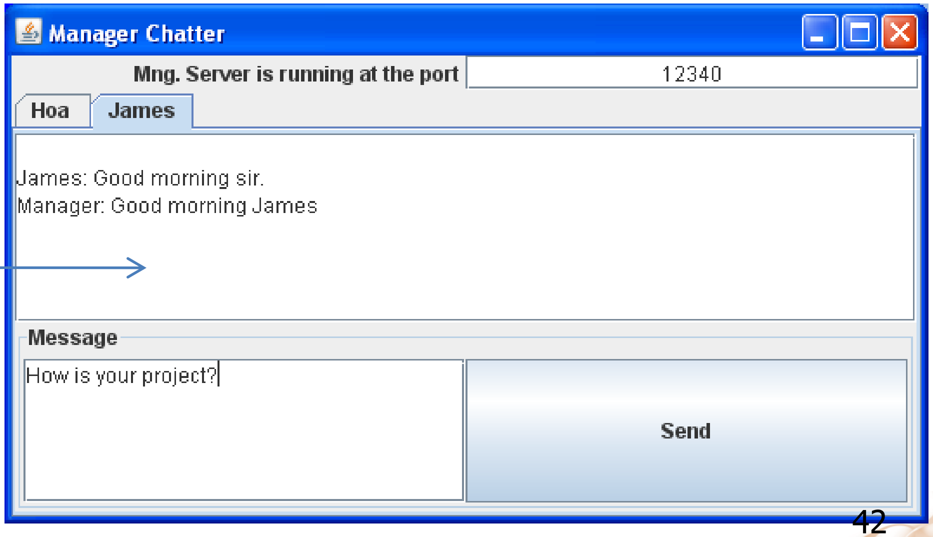
```
Hoa: Chao xep
Manager: Chao Hoa
```

Hom nay



James 127.0.0.1 12340

```
Manager is running

James: Good morning sir.
Manager: Good morning James
```

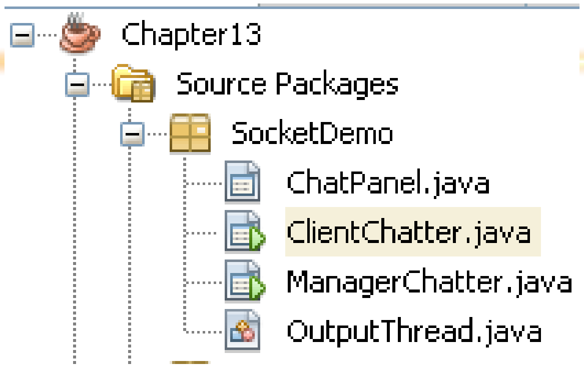


12340

```
James: Good morning sir.
Manager: Good morning James
```

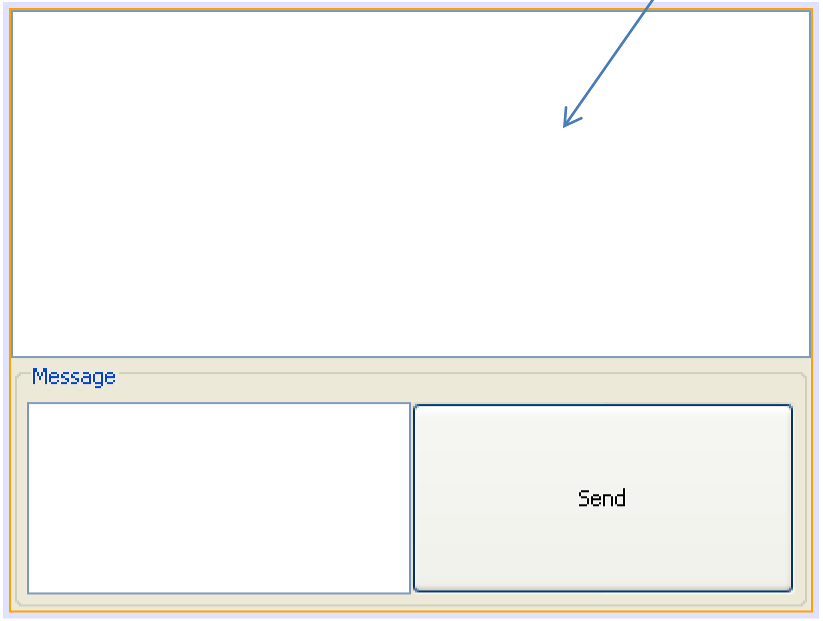
How is your project?

Java TCP Socket Demo.: Project architecture



```
Chapter13
  Source Packages
    SocketDemo
      ChatPanel.java
      ClientChatter.java
      ManagerChatter.java
      OutputThread.java
```

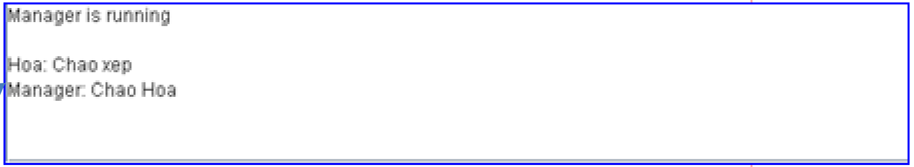
ChatPanel: Panel for chatting, it is used in client and server.
ClientChatter: GUI client program for staffs
ManagerChatter: GUI Server program for manager
OutputThread: a thread helps presenting received data (1 time/second)



TCP Socket Demo...

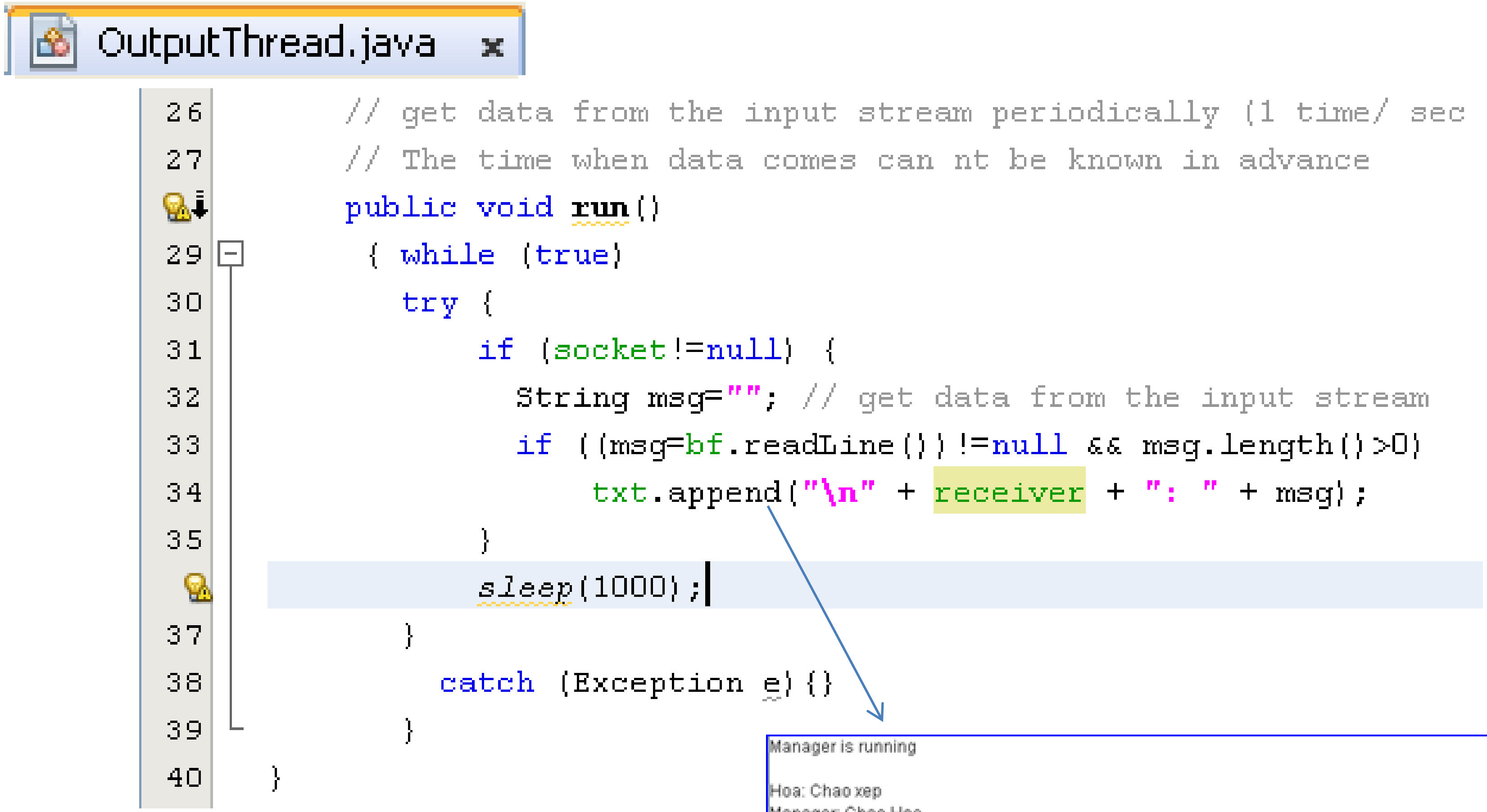
OutputThread.java

```
/* Thread presents received messages automatically */
package SocketDemo;
import javax.swing.JTextArea;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.Socket;
import javax.swing.JOptionPane;
public class OutputThread extends Thread {
    Socket socket;      // socket is joining to the communication
    JTextArea txt;   // text-area contains communicated message
    BufferedReader bf; // in put buffer of the socket
    String sender;         // sender, a site of the communication
    String receiver;       // receiver, other site of the communication

    public OutputThread ( Socket s, JTextArea txt, String sender, String receiver){
        super();
        this.socket =s; this.txt=txt; this.sender=sender; this.receiver=receiver;
        try{
         bf= new BufferedReader( new InputStreamReader(socket.getInputStream()));
        }
        catch (Exception e){
            JOptionPane.showMessageDialog(null, "Network Error!");
            System.exit(0);
        }
    }
```

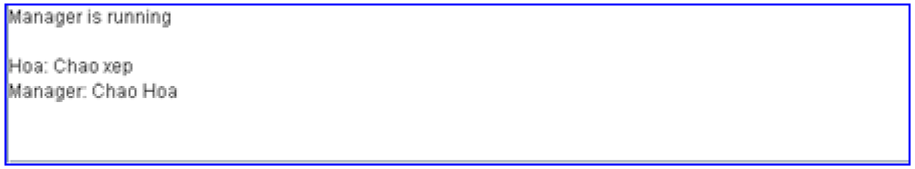


```
Manager is running

Hoa: Chao xep
Manager: Chao Hoa
```

TCP Socket Demo...

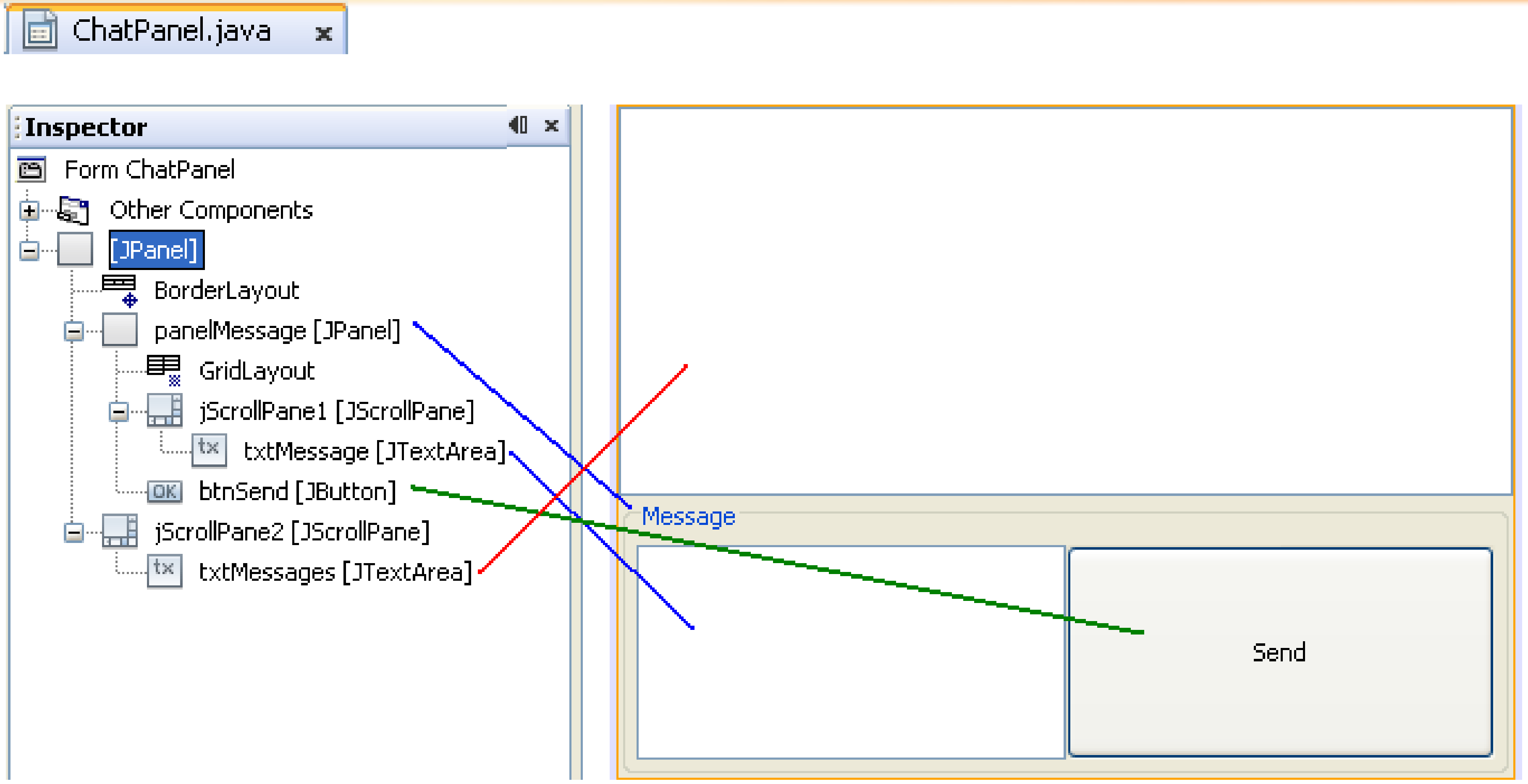

OutputThread.java

```java
        // get data from the input stream periodically (1 time/ sec
        // The time when data comes can nt be known in advance
        public void run()
         { while (true)
           try {
               if (socket!=null) {
                 String msg=""; // get data from the input stream
                 if ((msg=bf.readLine())!=null && msg.length()>0)
                     txt.append("\n" + receiver + ": " + msg);
               }
               sleep(1000);
           }
             catch (Exception e){}
           }
}
```



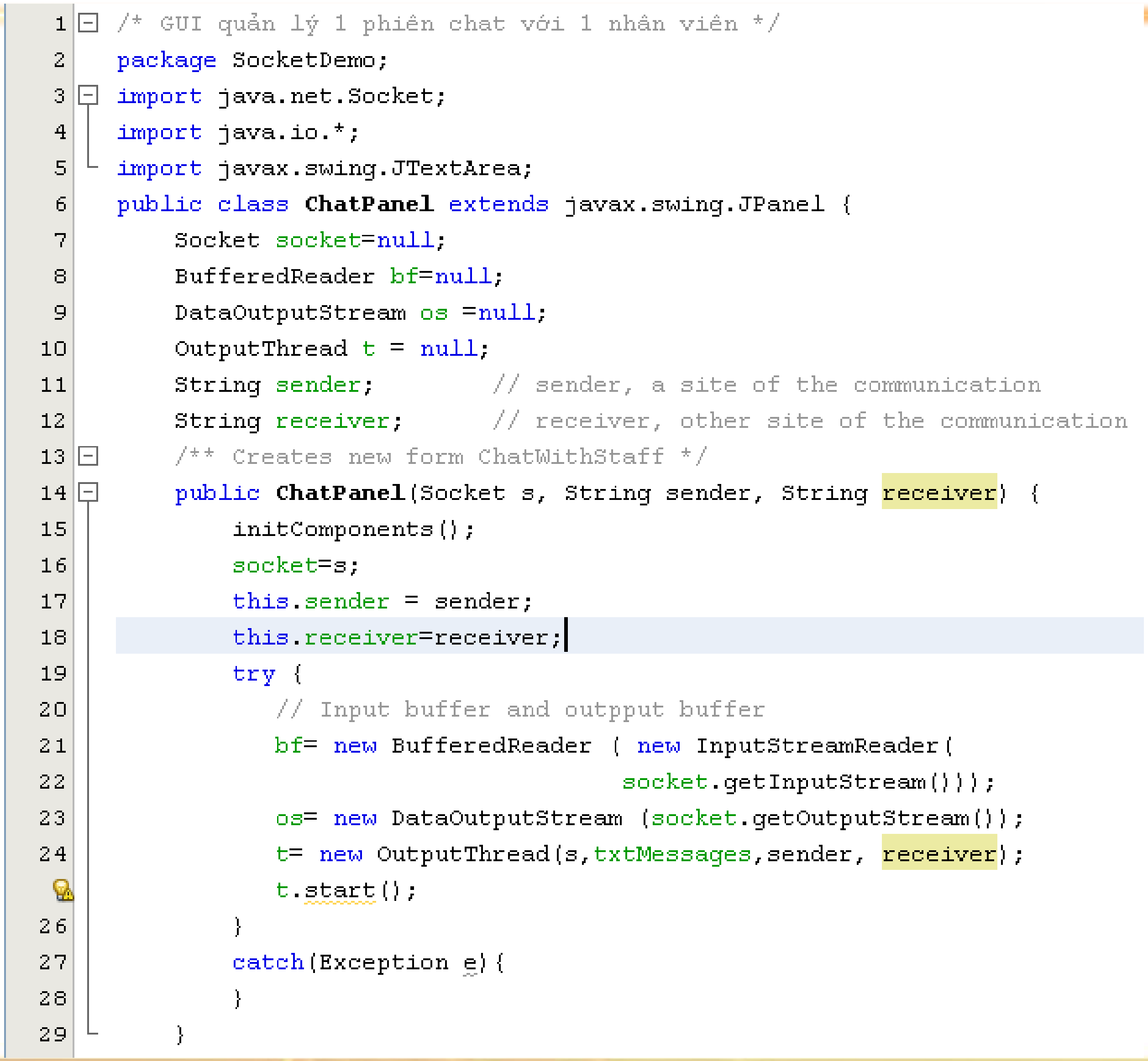
```
Manager is running

Hoa: Chao xep
Manager: Chao Hoa
```

TCP Socket Demo...



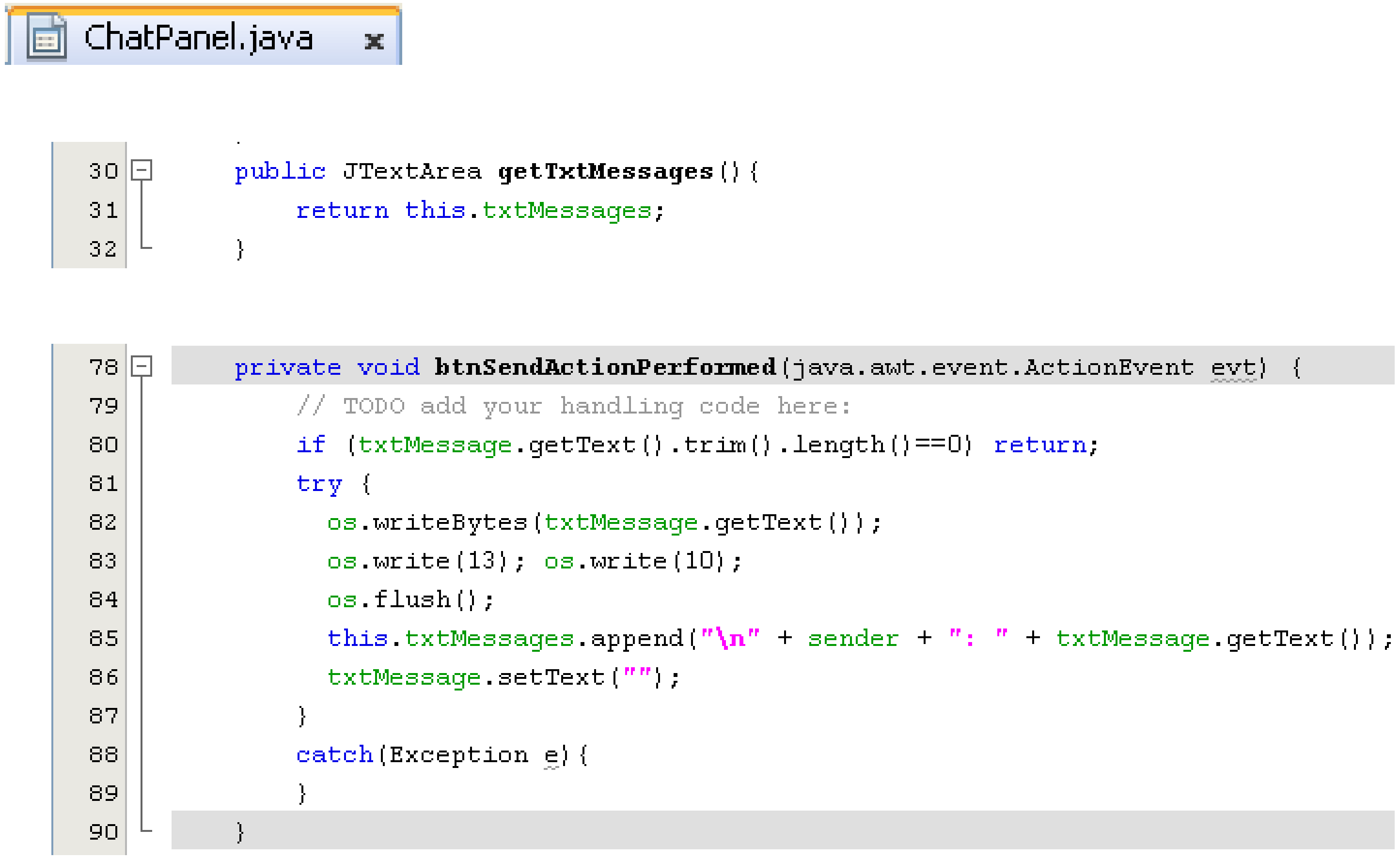
TCP Socket Demo....

ChatPanel.java



```java
/* GUI quản lý 1 phiên chat với 1 nhân viên */
package SocketDemo;
import java.net.Socket;
import java.io.*;
import javax.swing.JTextArea;
public class ChatPanel extends javax.swing.JPanel {
    Socket socket=null;
    BufferedReader bf=null;
    DataOutputStream os =null;
    OutputThread t = null;
    String sender;        // sender, a site of the communication
    String receiver;      // receiver, other site of the communication
    /** Creates new form ChatWithStaff */
    public ChatPanel(Socket s, String sender, String receiver) {
        initComponents();
        socket=s;
        this.sender = sender;
        this.receiver=receiver;
        try {
            // Input buffer and outpput buffer
            bf= new BufferedReader ( new InputStreamReader(
                                   socket.getInputStream()));
            os= new DataOutputStream (socket.getOutputStream());
            t= new OutputThread(s,txtMessages,sender, receiver);
            t.start();
        }
        catch(Exception e){
        }
    }
```

TCP Socket Demo….

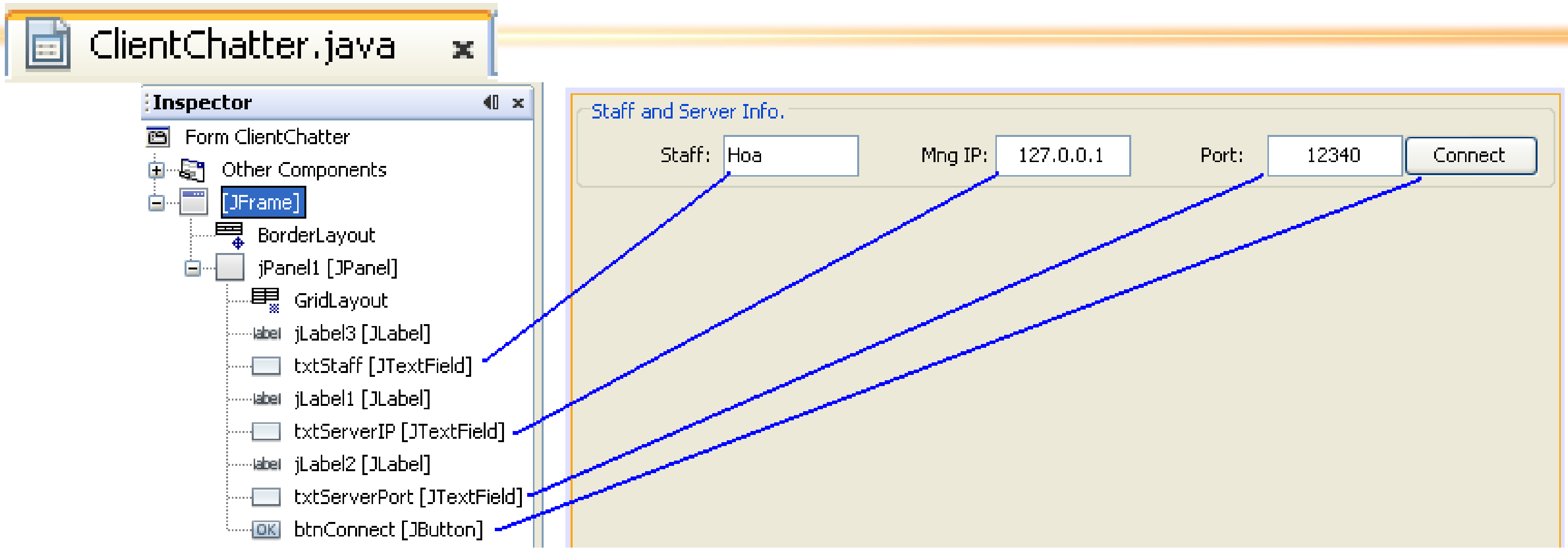


ChatPanel.java

```java
    public JTextArea getTxtMessages(){
        return this.txtMessages;
    }
```

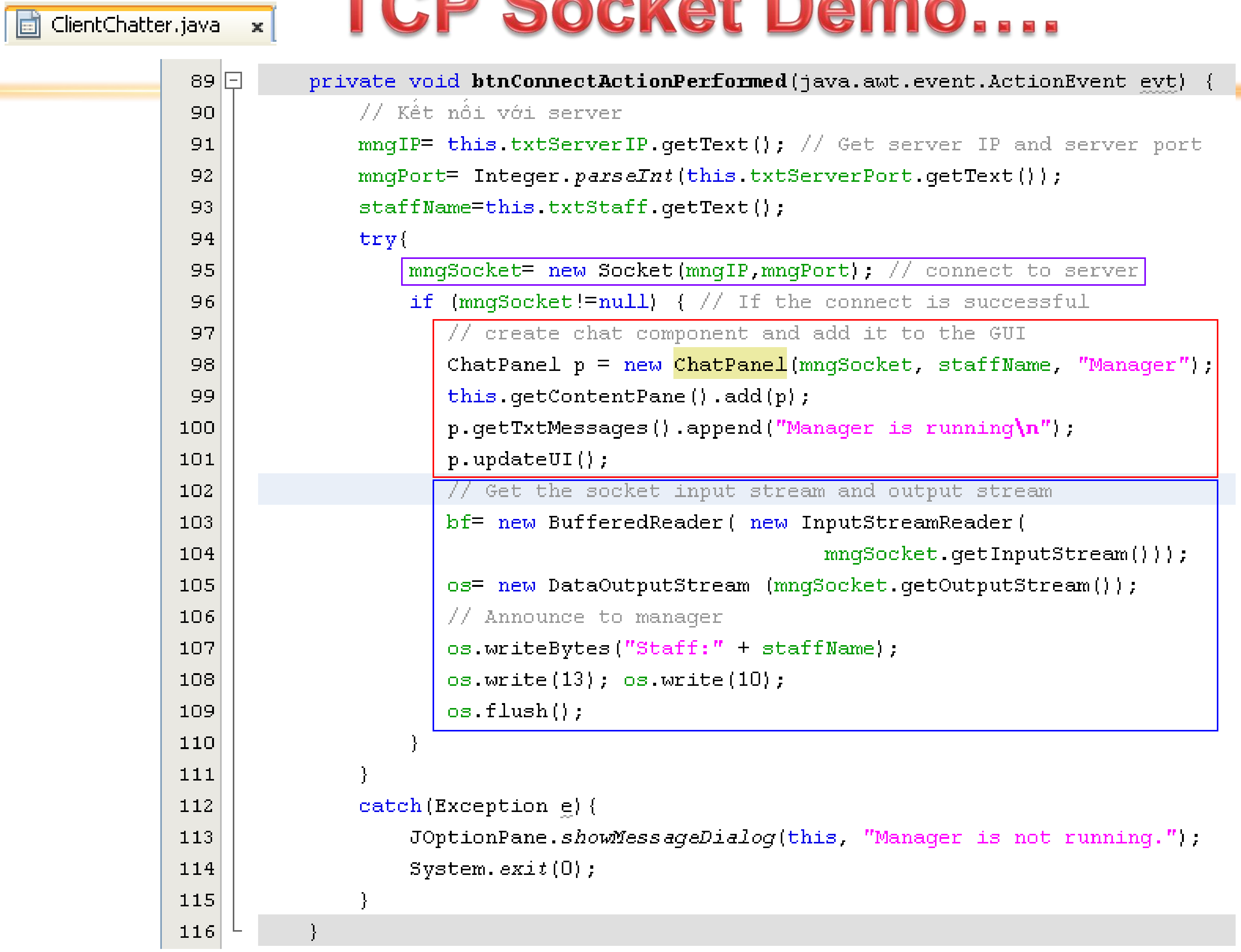
```java
    private void btnSendActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        if (txtMessage.getText().trim().length()==0) return;
        try {
          os.writeBytes(txtMessage.getText());
          os.write(13); os.write(10);
          os.flush();
          this.txtMessages.append("\n" + sender + ": " + txtMessage.getText());
          txtMessage.setText("");
        }
        catch(Exception e){
        }
    }
```

TCP Socket Demo....



```java
/* GUI for staffs */
package SocketDemo;
import java.net.Socket;
import javax.swing.JOptionPane;
import java.io.*;
public class ClientChatter extends javax.swing.JFrame {
    Socket mngSocket=null ; // Socket of the manager program
    String mngIP="";        // Manager IP
    int mngPort=0;          // Manager port
    String staffName="";    // name of the staff
    BufferedReader bf=null;       // Input buffer
    DataOutputStream os =null;    // output buffer
    // Thread allows presenting received data automatically
    OutputThread t=null;
```

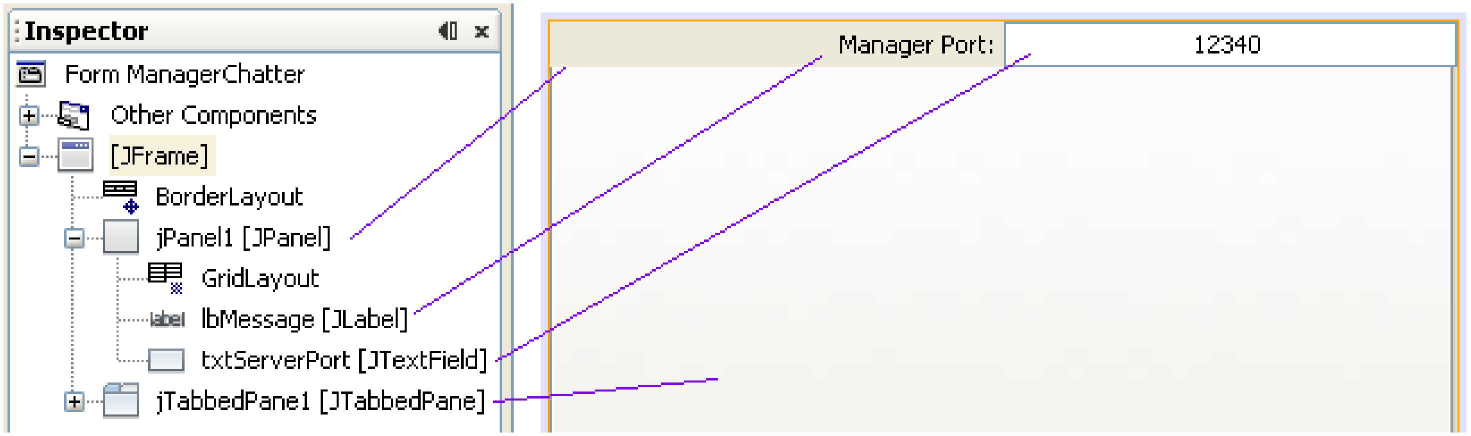
TCP Socket Demo….


ClientChatter.java

```java
private void btnConnectActionPerformed(java.awt.event.ActionEvent evt) {
    // Kết nối với server
    mngIP= this.txtServerIP.getText(); // Get server IP and server port
    mngPort= Integer.parseInt(this.txtServerPort.getText());
    staffName=this.txtStaff.getText();
    try{
        mngSocket= new Socket(mngIP,mngPort); // connect to server
        if (mngSocket!=null) { // If the connect is successful
           // create chat component and add it to the GUI
           ChatPanel p = new ChatPanel(mngSocket, staffName, "Manager");
           this.getContentPane().add(p);
           p.getTxtMessages().append("Manager is running\n");
           p.updateUI();
           // Get the socket input stream and output stream
           bf= new BufferedReader( new InputStreamReader(
                                         mngSocket.getInputStream()));
           os= new DataOutputStream (mngSocket.getOutputStream());
           // Announce to manager
           os.writeBytes("Staff:" + staffName);
           os.write(13); os.write(10);
           os.flush();
        }
    }
    catch(Exception e){
        JOptionPane.showMessageDialog(this, "Manager is not running.");
        System.exit(0);
    }
}
```

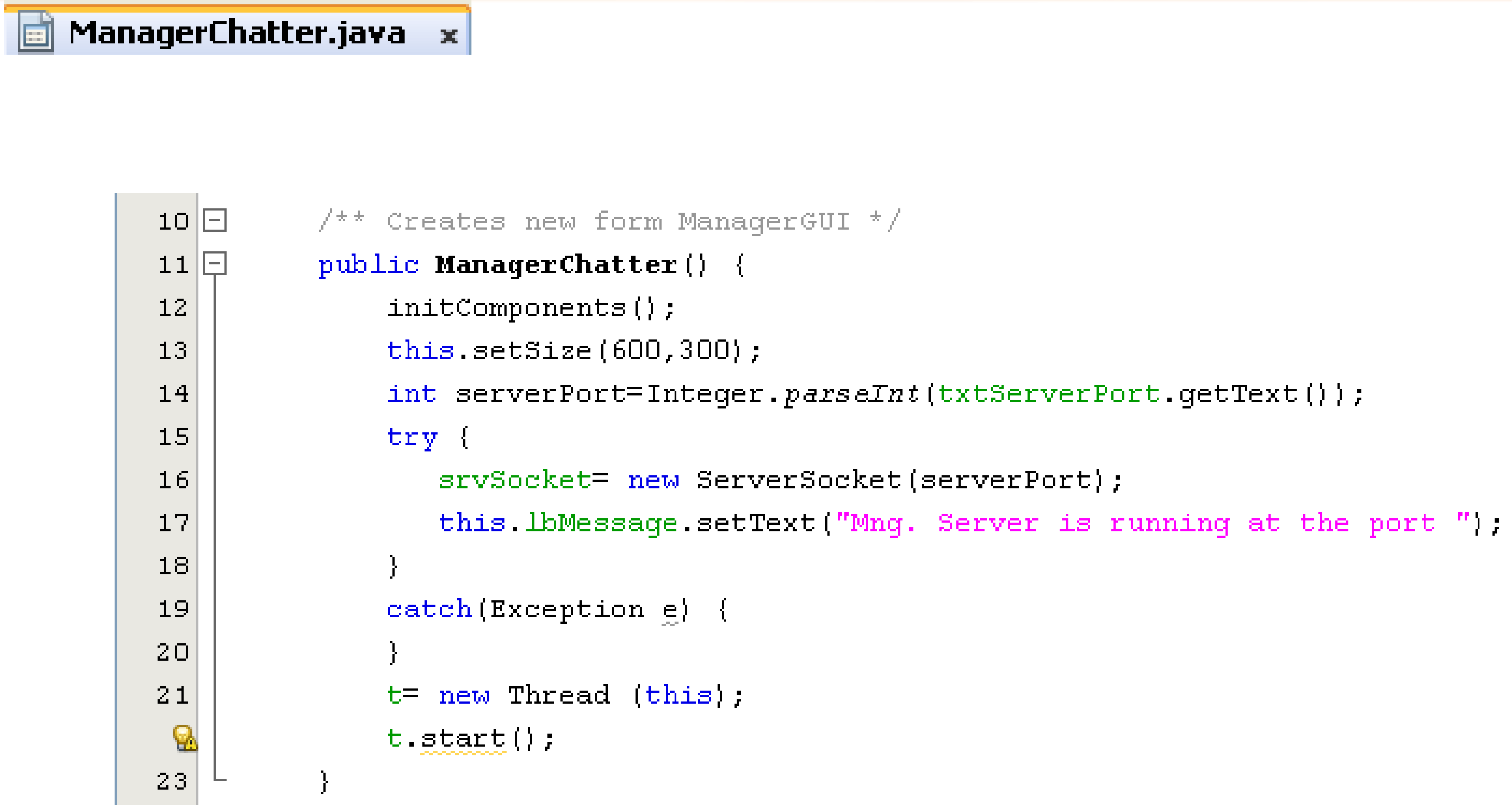
TCP Socket Demo....

ManagerChatter.java



```
package SocketDemo;
/* @author SuTV */
import java.net.Socket;
import java.net.ServerSocket;
import java.io.*;
public class ManagerChatter extends javax.swing.JFrame implements Runnable {
    ServerSocket srvSocket=null;
    BufferedReader br=null;
    Thread t; // thread for exploring connections from staffs
```

TCP Socket Demo....



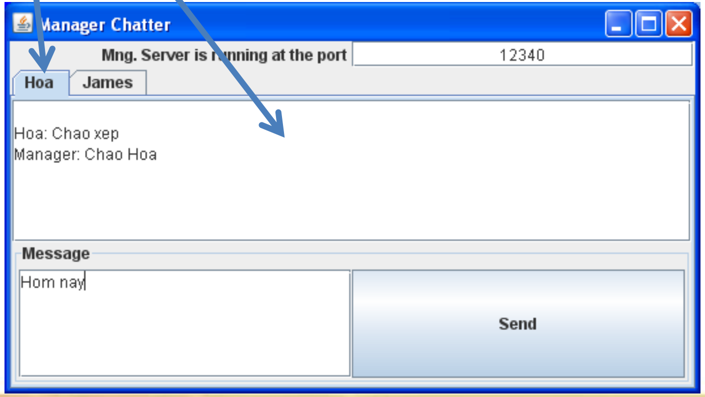
ManagerChatter.java

```
    /** Creates new form ManagerGUI */
    public ManagerChatter() {
        initComponents();
        this.setSize(600,300);
        int serverPort=Integer.parseInt(txtServerPort.getText());
        try {
            srvSocket= new ServerSocket(serverPort);
            this.lbMessage.setText("Mng. Server is running at the port ");
        }
        catch(Exception e) {
        }
        t= new Thread (this);
        t.start();
    }
```

TCP Socket Demo….

ManagerChatter.java

```
public void run(){
    while (true){
      try {  // Wait for a client
         Socket aStaffSocket = srvSocket.accept();
         if (aStaffSocket!=null) { // If there is a connection
            // Get staffname
            // When a staff inits a connection, he/she sends his/her name first
            br= new BufferedReader (new InputStreamReader(
                                        aStaffSocket.getInputStream()));
            String S= br.readLine();
            int pos = S.indexOf(":"); // Fortmat:   Staff:Hoa
            String staffName = S.substring(pos+1); // Get name
            // Crate a tab for this connection
            ChatPanel p= new ChatPanel(aStaffSocket, "Manager",staffName);
            jTabbedPane1.add(staffName,p);
            p.updateUI();
         }
         Thread.sleep(1000);
      }
      catch(Exception e)  {
      }
    }
}
```



Manager Chatter

Mng. Server is running at the port 12340

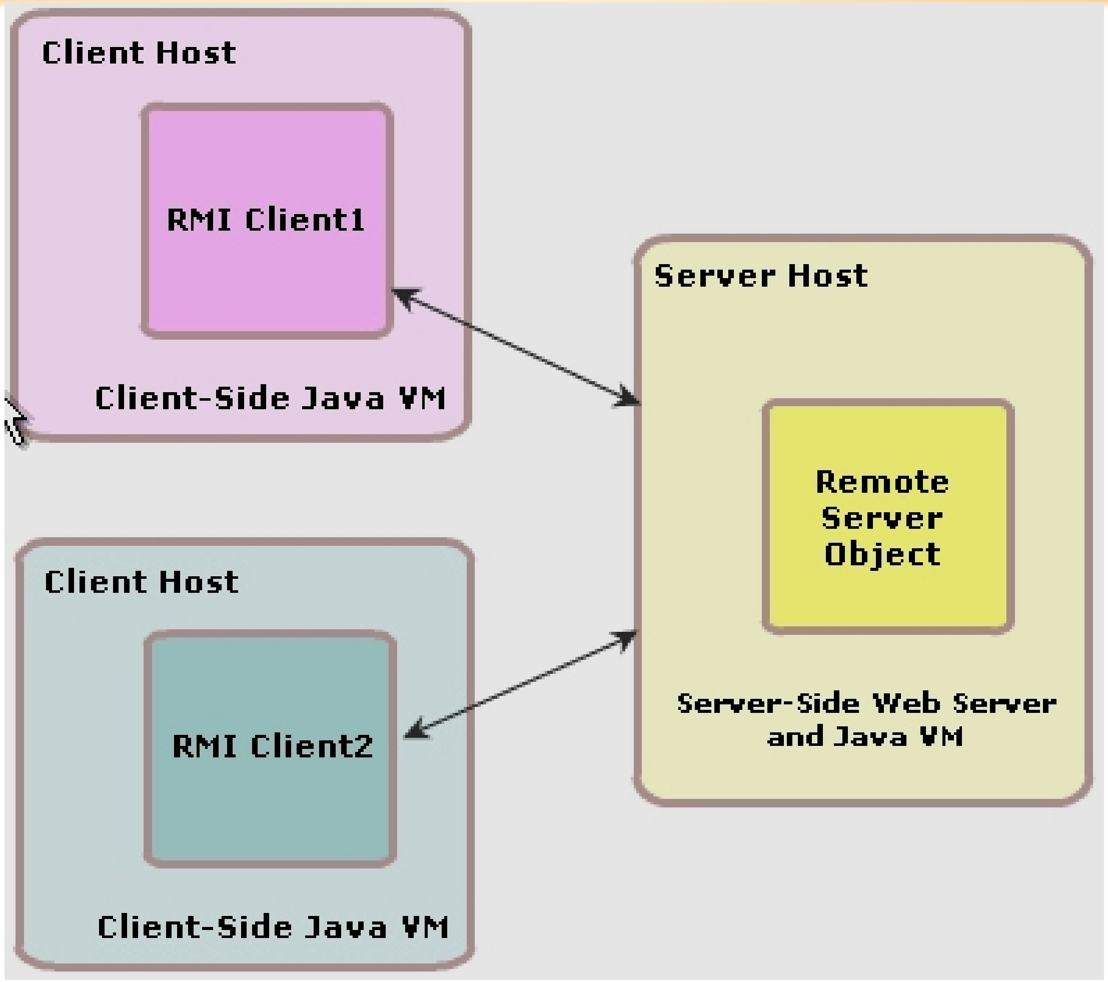
Hoa | James

```
Hoa: Chao xep
Manager: Chao Hoa
```

Message

Hom nay

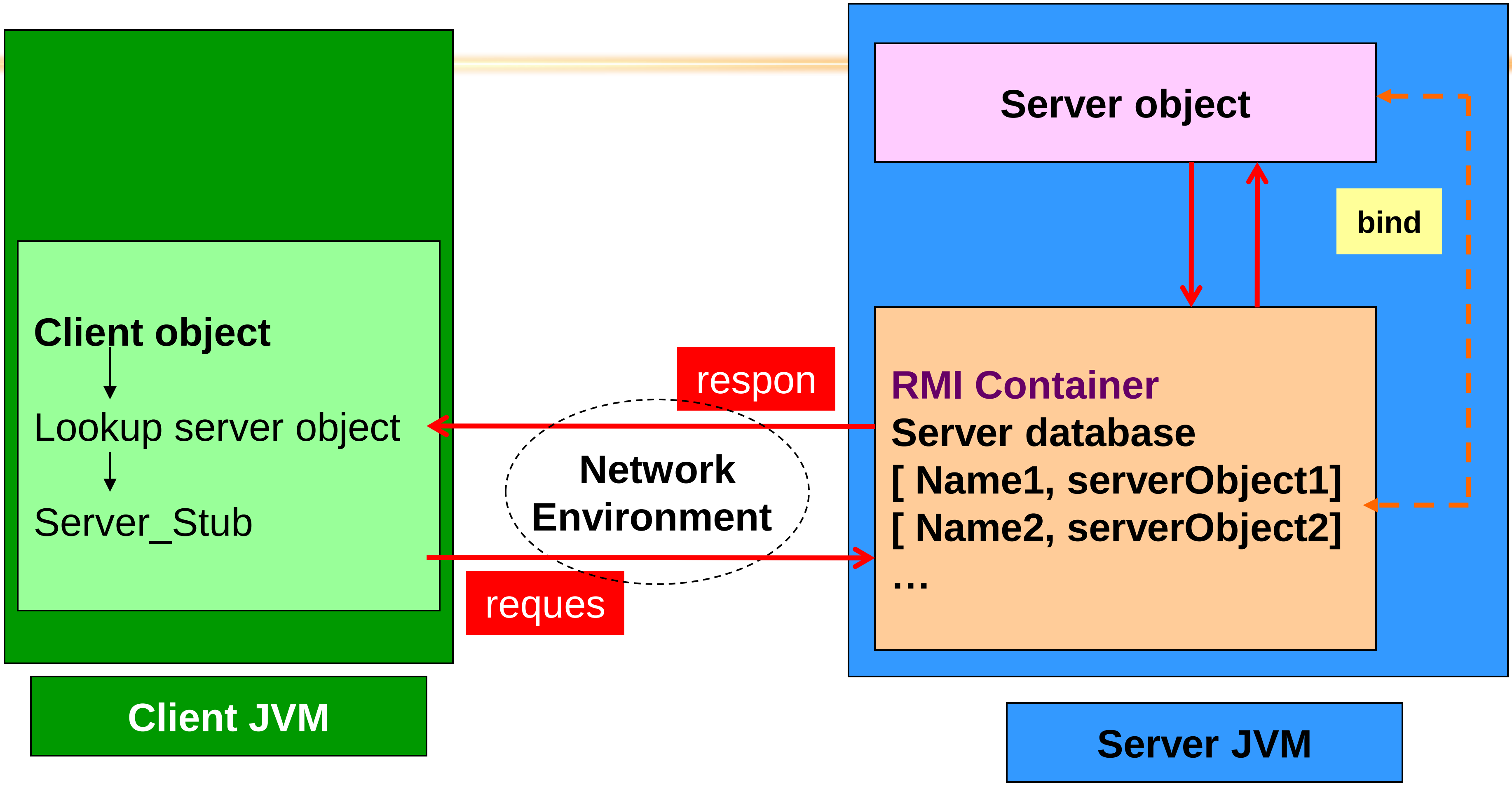
4- Remote Method Invocation (RMI)



It is the basic for protocols used in Java application server, JBoss for example.

The **Java Remote Method Invocation** (**Java RMI**) is a Java API that performs the object-oriented equivalent of remote procedure calls (RPC), with support for direct transfer of serialized Java classes and distributed garbage collection.
The original implementation depends on JVM class representation mechanisms and it thus only supports making calls from one JVM to another. The protocol underlying this Java-only implementation is known as Java Remote Method

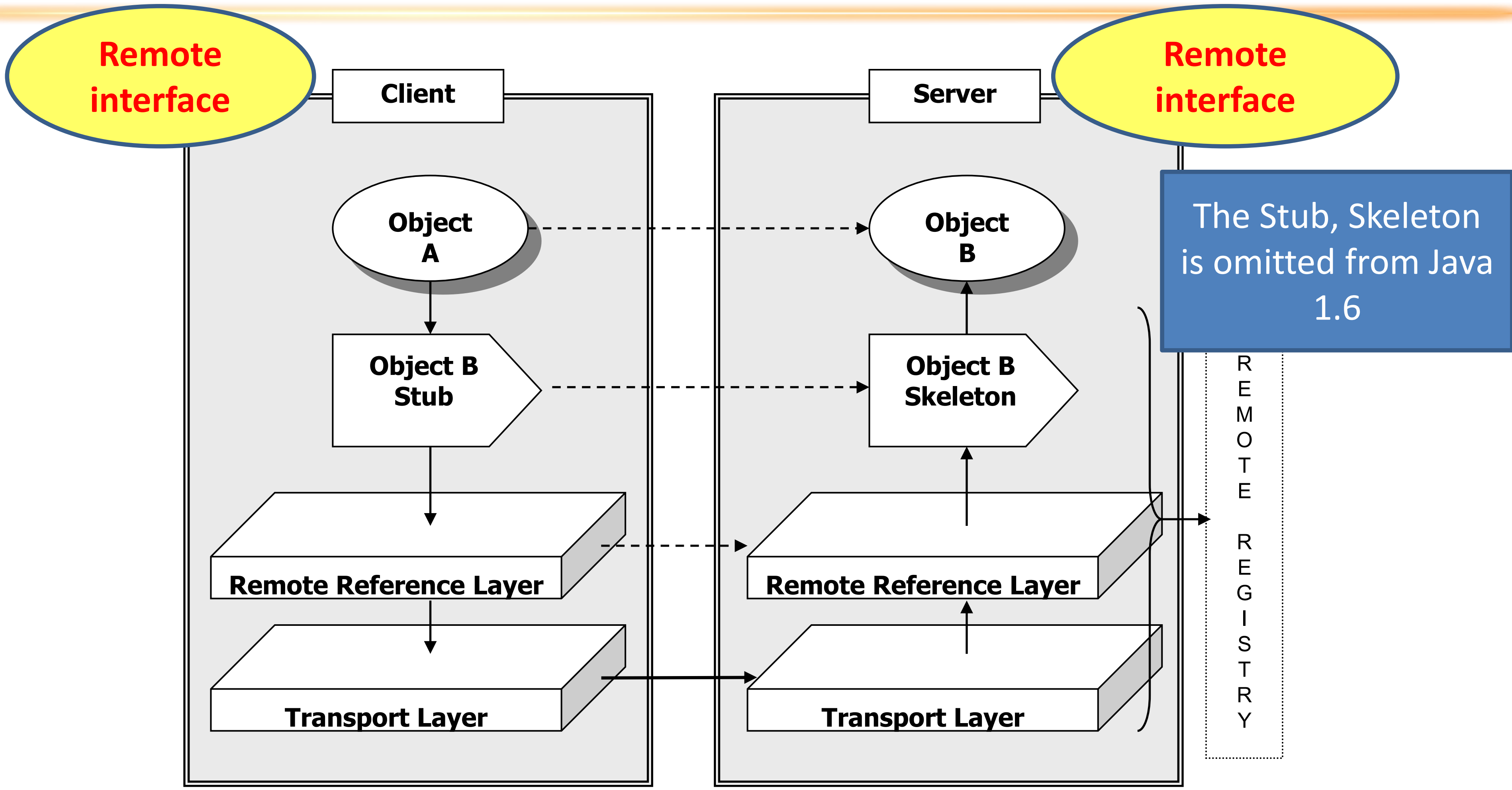
RMI…



In Windows, RMI container, pre-defined in JDK, is the program **rmiregistry.exe**

We can create a RMI container by an Java object. See demo.

RMI...



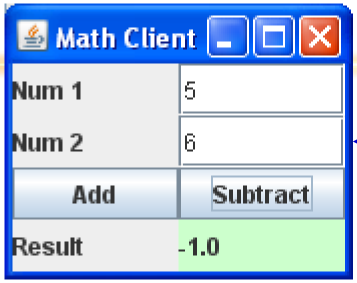
The Stub, Skeleton is omitted from Java 1.6

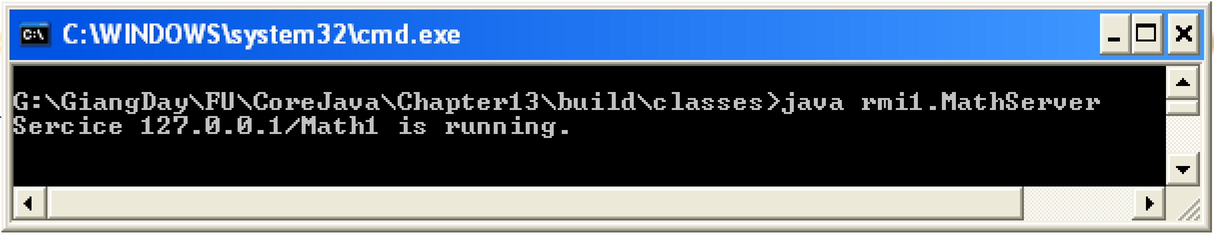
From Java 1.6, code for network communicating is implemented automatically

RMI…: 5 Steps

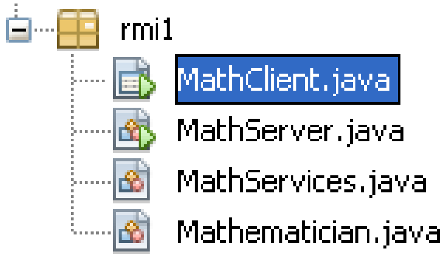
1. Create the remote interface
2. Create the remote class (server) implementing the remote interface.
3. Create Server program using server object
4. Create the client program
5. Run apps: Start server program first then the client program.

Demo 1: Simple RMI

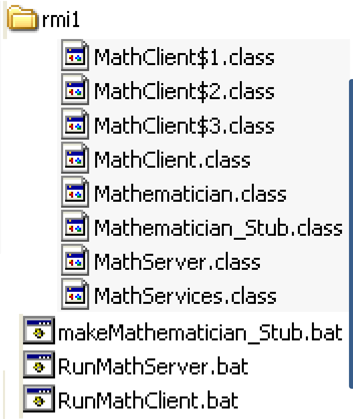




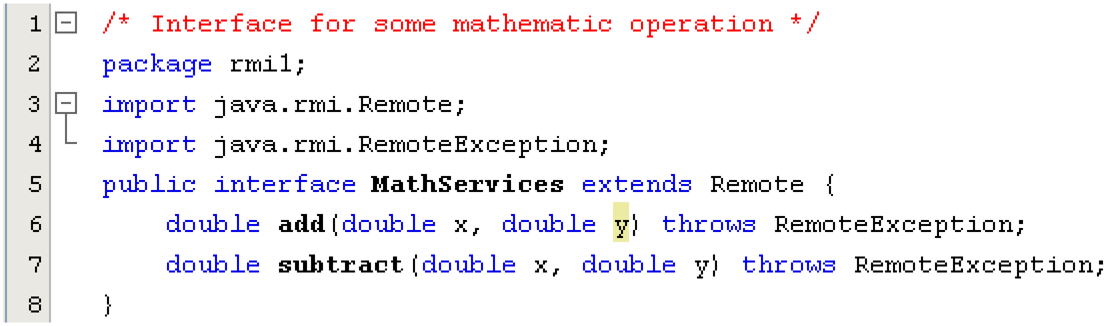
```
G:\GiangDay\FU\CoreJava\Chapter13\build\classes>java rmi1.MathServer
Sercice 127.0.0.1/Math1 is running.
```



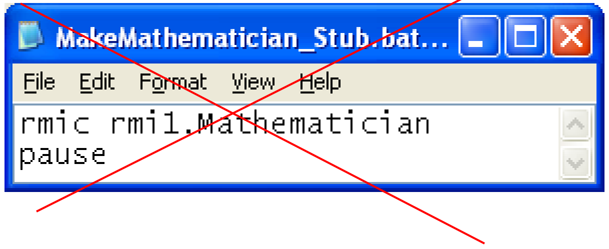
Client program
Server program (using server class)
Remote interface
Server class implements the interface



The Stub, Skeleton is omitted from Java 1.6

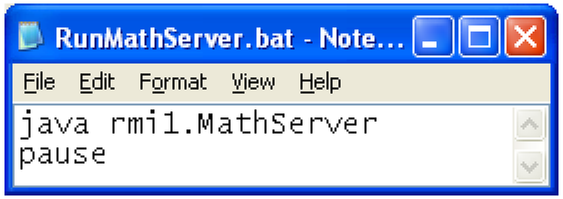


```java
/* Interface for some mathematic operation */
package rmi1;
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface MathServices extends Remote {
    double add(double x, double y) throws RemoteException;
    double subtract(double x, double y) throws RemoteException;
}
```

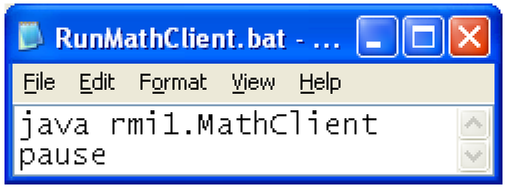


```
~~rmic rmi1.Mathematician~~
~~pause~~
```

Step 1: Create a remote interface



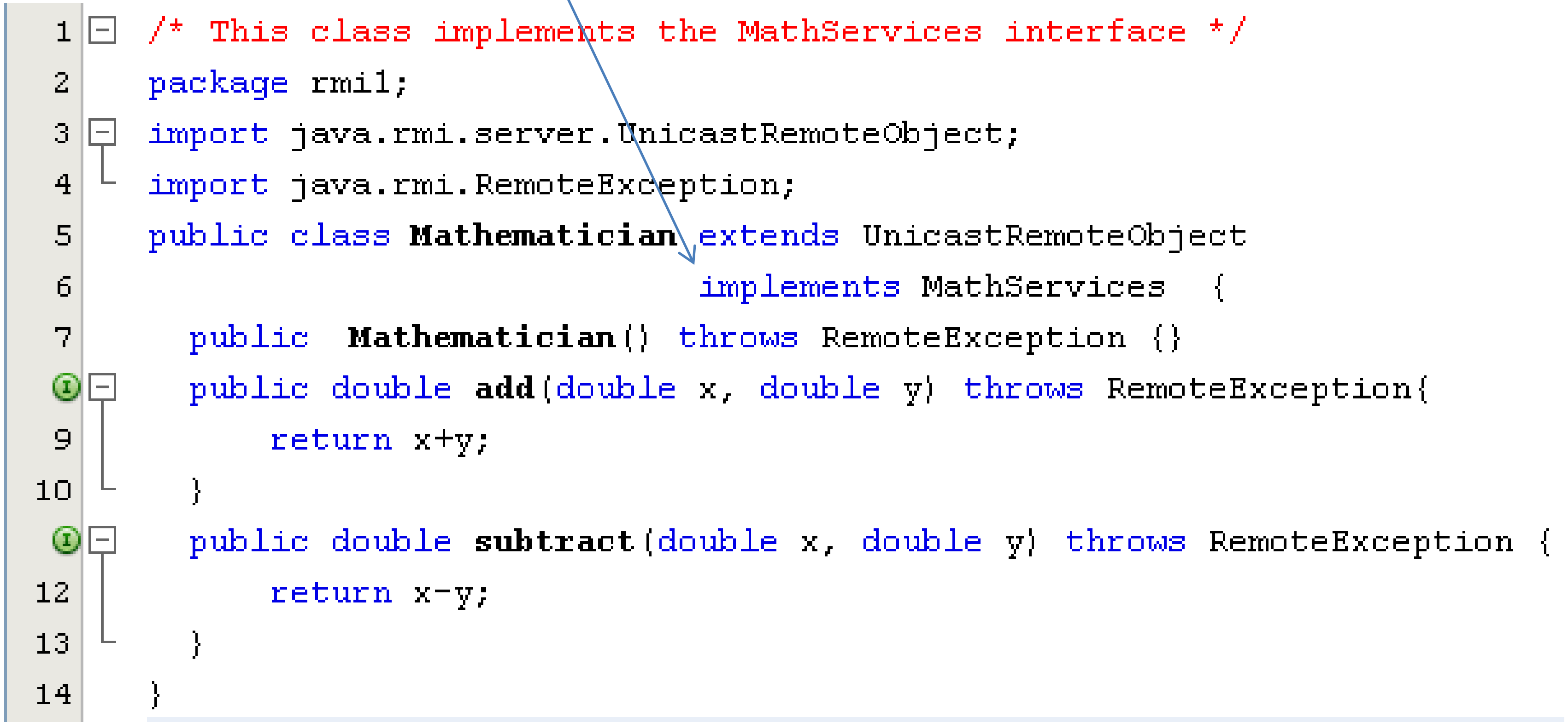
```
java rmi1.MathServer
pause
```



```
java rmi1.MathClient
pause
```

Demo 1: Simple RMI...

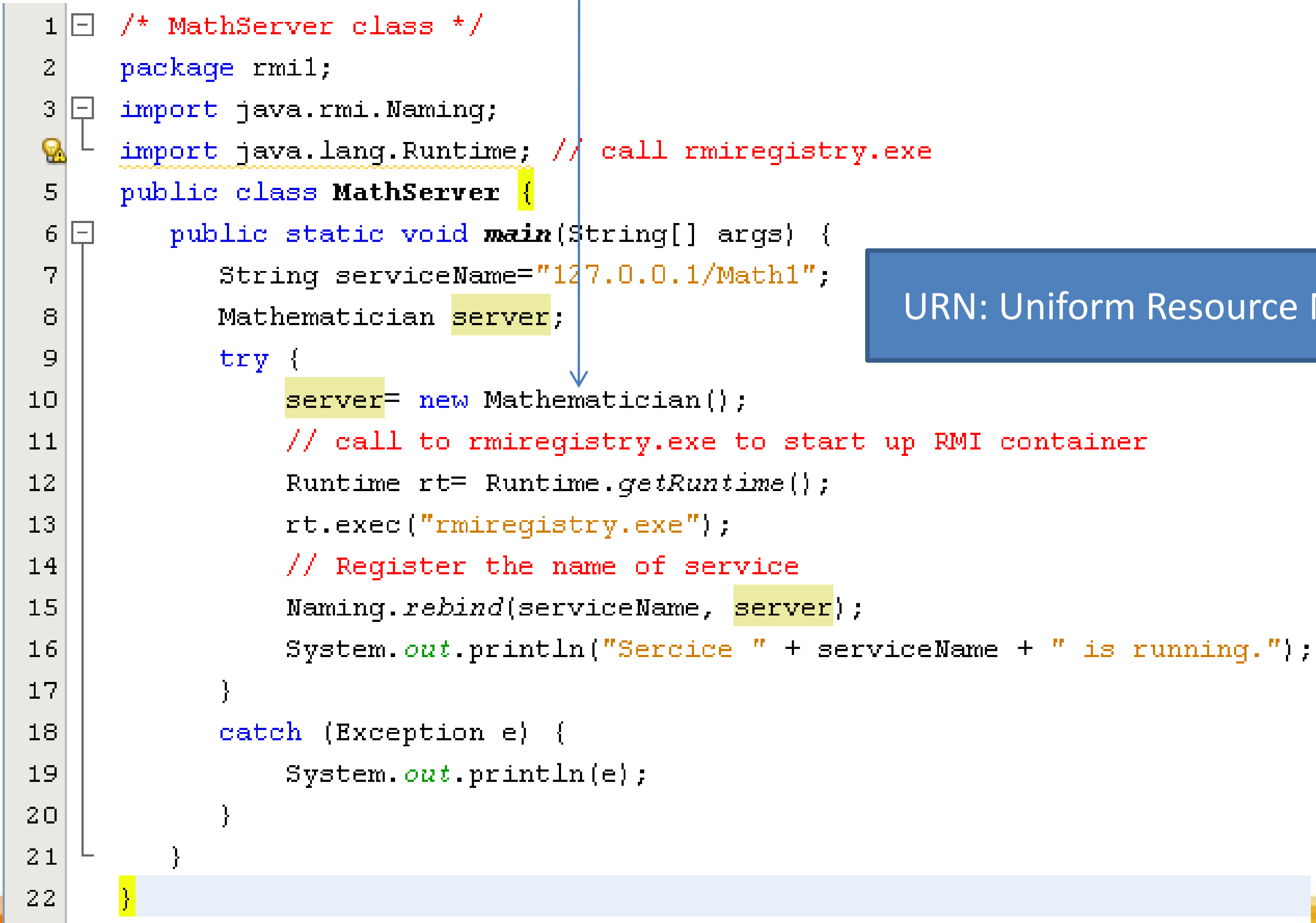
Step 2: Create server class implementing remote interface



```
/* This class implements the MathServices interface */
package rmi1;
import java.rmi.server.UnicastRemoteObject;
import java.rmi.RemoteException;
public class Mathematician extends UnicastRemoteObject
                                           implements MathServices  {
  public  Mathematician() throws RemoteException {}
  public double add(double x, double y) throws RemoteException{
      return x+y;
  }
  public double subtract(double x, double y) throws RemoteException {
      return x-y;
  }
}
```

Demo 1: Simple RMI...

Step 3: Create server program in which a server object is used

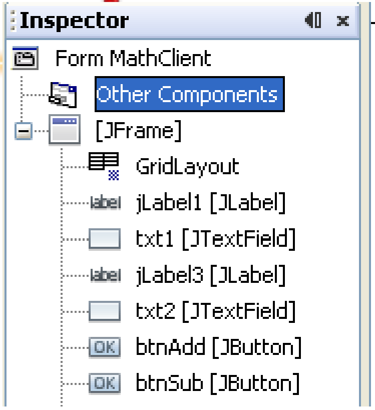


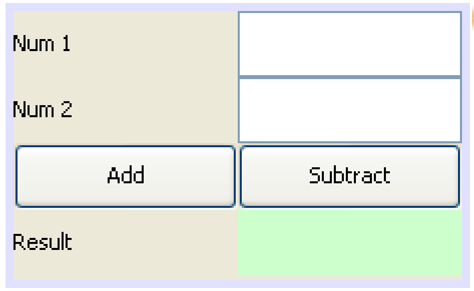
```
/* MathServer class */
package rmi1;
import java.rmi.Naming;
import java.lang.Runtime; // call rmiregistry.exe
public class MathServer {
    public static void main(String[] args) {
        String serviceName="127.0.0.1/Math1";
        Mathematician server;
        try {
            server= new Mathematician();
            // call to rmiregistry.exe to start up RMI container
            Runtime rt= Runtime.getRuntime();
            rt.exec("rmiregistry.exe");
            // Register the name of service
            Naming.rebind(serviceName, server);
            System.out.println("Sercice " + serviceName + " is running.");
        }
        catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

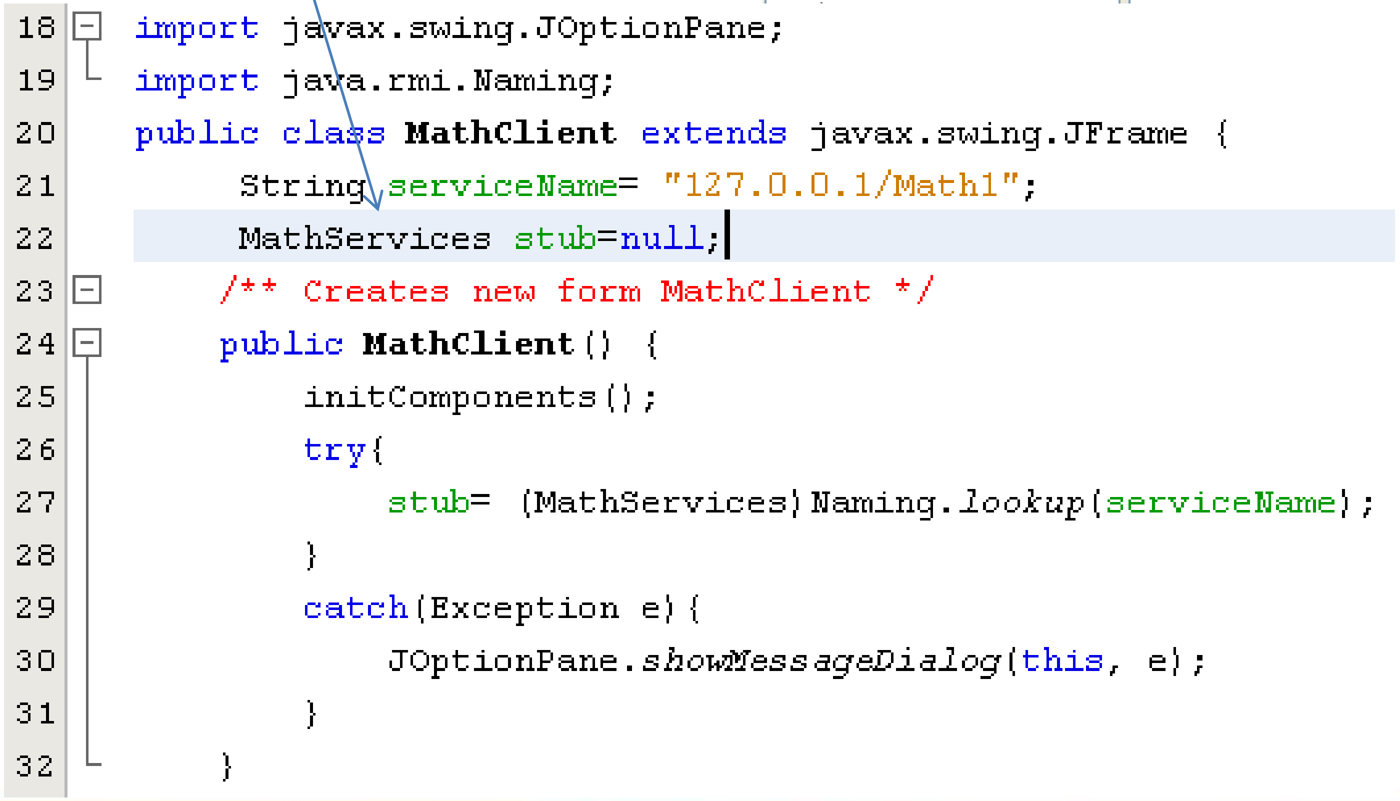
URN: Uniform Resource Name

Demo 1: Simple RMI...

Step 4: Create client program in which the remote interface is used

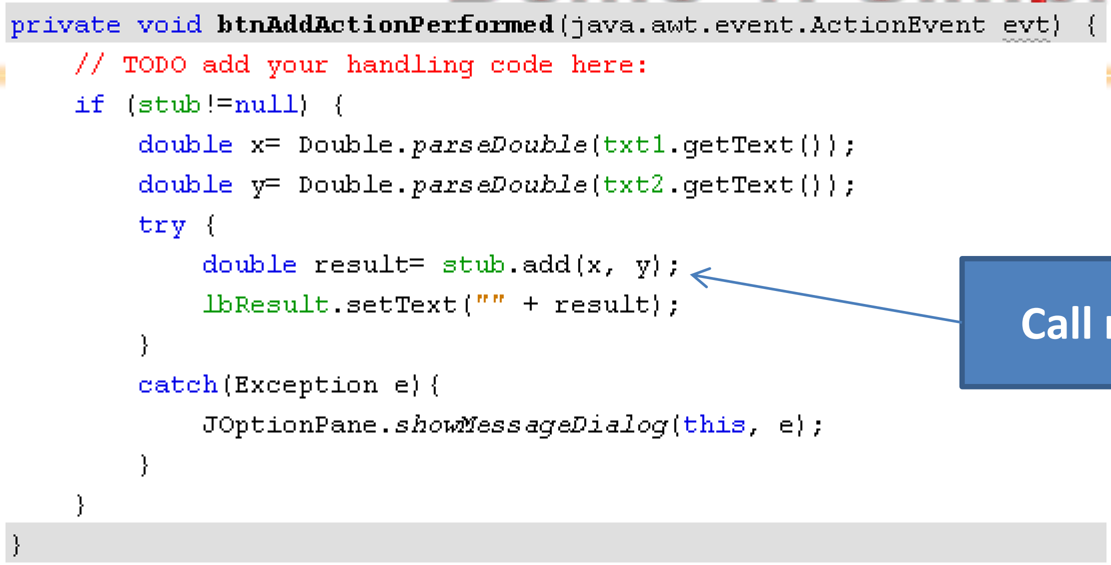

Inspector
Form MathClient
Other Components
[JFrame]
GridLayout
jLabel1 [JLabel]
txt1 [JTextField]
jLabel3 [JLabel]
txt2 [JTextField]
btnAdd [JButton]
btnSub [JButton]





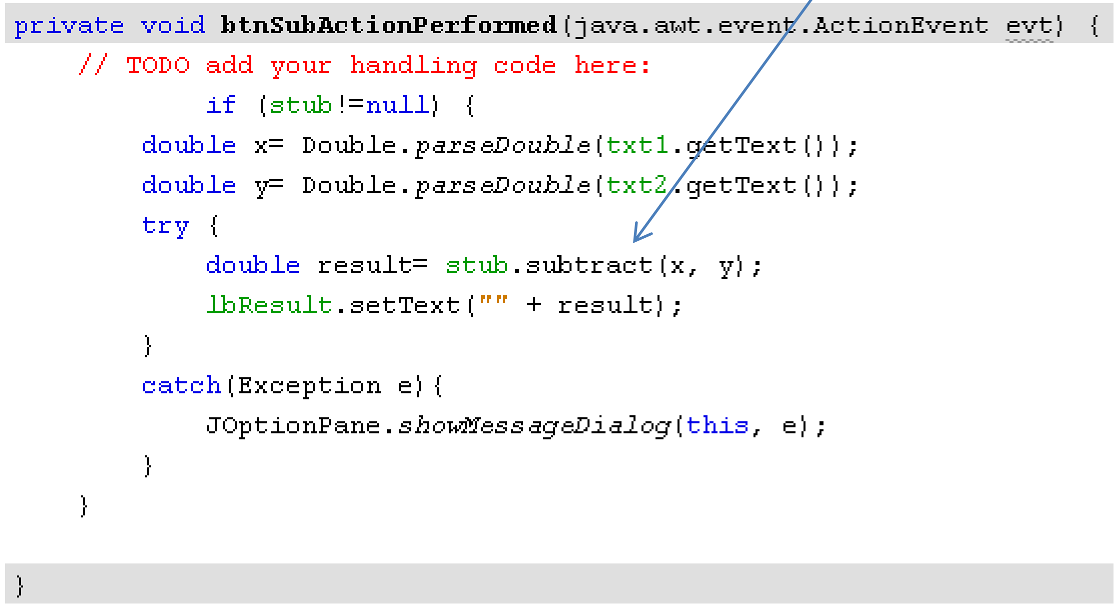
```
import javax.swing.JOptionPane;
import java.rmi.Naming;
public class MathClient extends javax.swing.JFrame {
     String serviceName= "127.0.0.1/Math1";
     MathServices stub=null;
    /** Creates new form MathClient */
    public MathClient() {
        initComponents();
        try{
            stub= (MathServices)Naming.lookup(serviceName);
        }
        catch(Exception e){
            JOptionPane.showMessageDialog(this, e);
        }
    }
```

Demo 1: Simple RMI...



```
private void btnAddActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    if (stub!=null) {
        double x= Double.parseDouble(txt1.getText());
        double y= Double.parseDouble(txt2.getText());
        try {
            double result= stub.add(x, y);
            lbResult.setText("" + result);
        }
        catch(Exception e){
            JOptionPane.showMessageDialog(this, e);
        }
    }
}
```

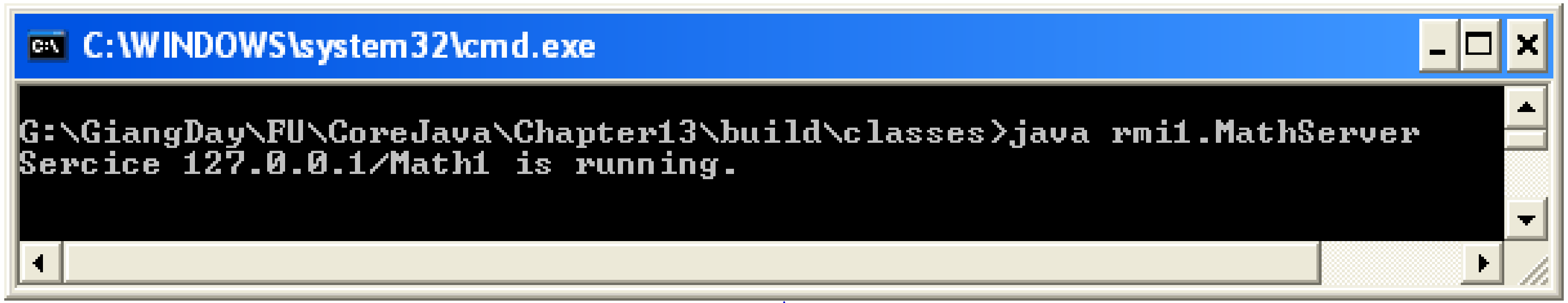
Call methods of remote object



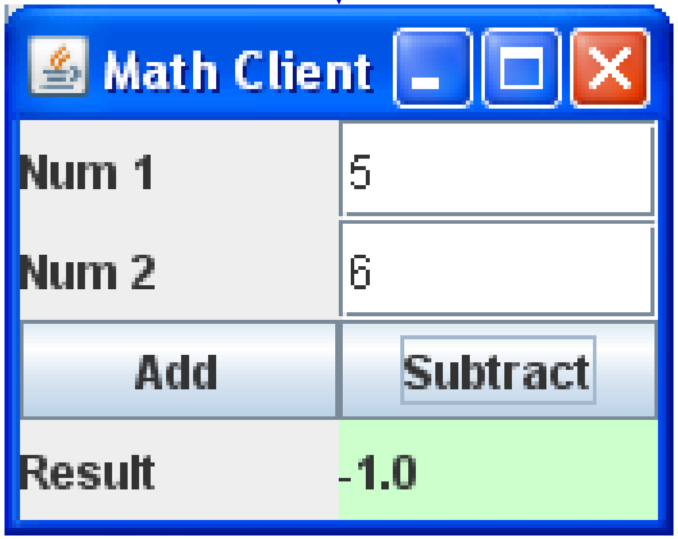
```
private void btnSubActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
            if (stub!=null) {
        double x= Double.parseDouble(txt1.getText());
        double y= Double.parseDouble(txt2.getText());
        try {
            double result= stub.subtract(x, y);
            lbResult.setText("" + result);
        }
        catch(Exception e){
            JOptionPane.showMessageDialog(this, e);
        }
    }

}
```

Demo 1: Simple RMI…

Step 6: Run server program first then client program

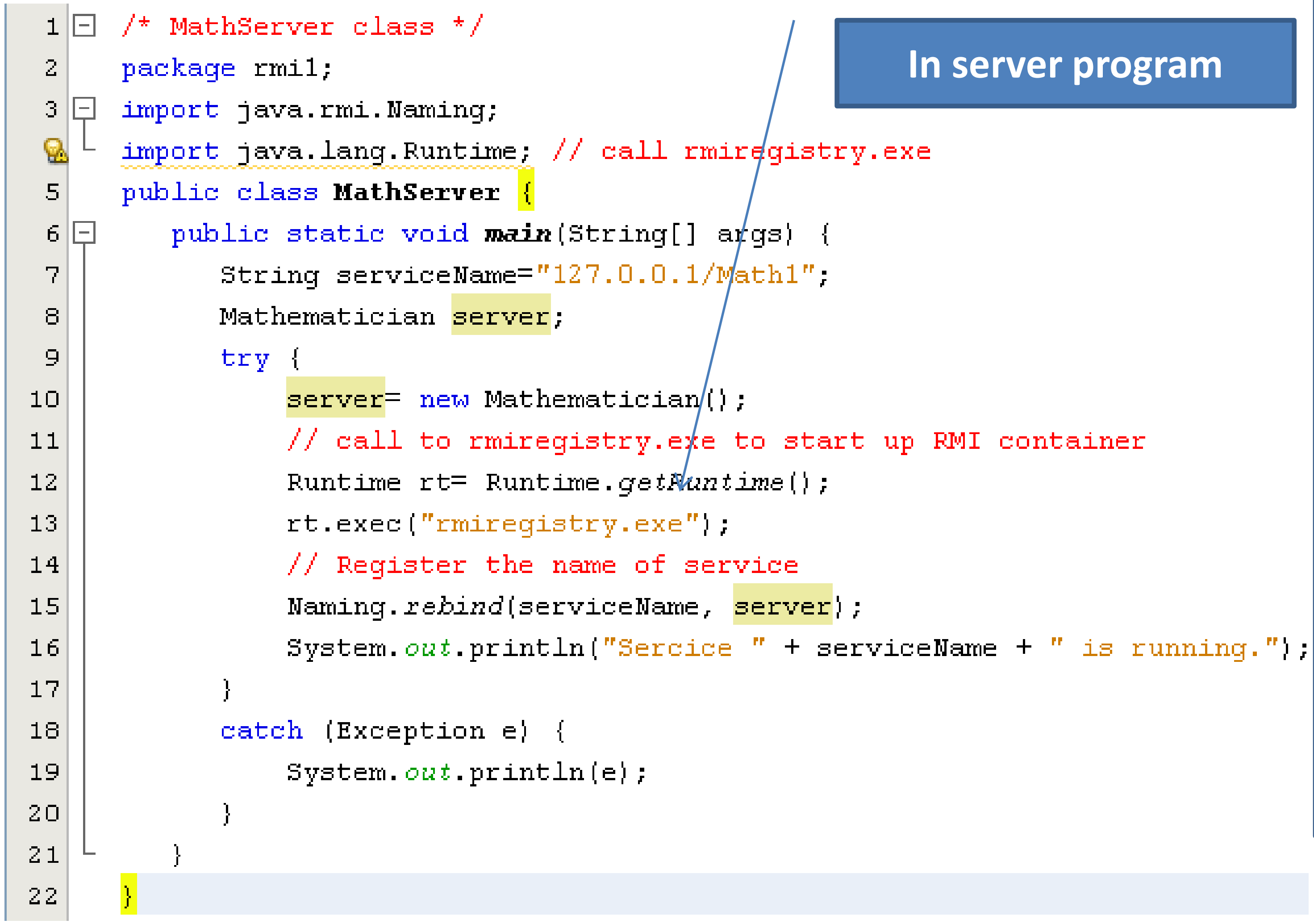


```
G:\GiangDay\FU\CoreJava\Chapter13\build\classes>java rmi1.MathServer
Sercice 127.0.0.1/Math1 is running.
```



Num 1	5
Num 2	6
Result	-1.0

Demo 1- Evaluation



```
/* MathServer class */
package rmi1;
import java.rmi.Naming;
import java.lang.Runtime; // call rmiregistry.exe
public class MathServer {
    public static void main(String[] args) {
        String serviceName="127.0.0.1/Math1";
        Mathematician server;
        try {
            server= new Mathematician();
            // call to rmiregistry.exe to start up RMI container
            Runtime rt= Runtime.getRuntime();
            rt.exec("rmiregistry.exe");
            // Register the name of service
            Naming.rebind(serviceName, server);
            System.out.println("Sercice " + serviceName + " is running.");
        }
        catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

In server program

Disadvantages

- Platform dependent

- An exception is thrown when we run the server program again because rmiregistry.exe must be terminated after each run (use task manager/ Processes)

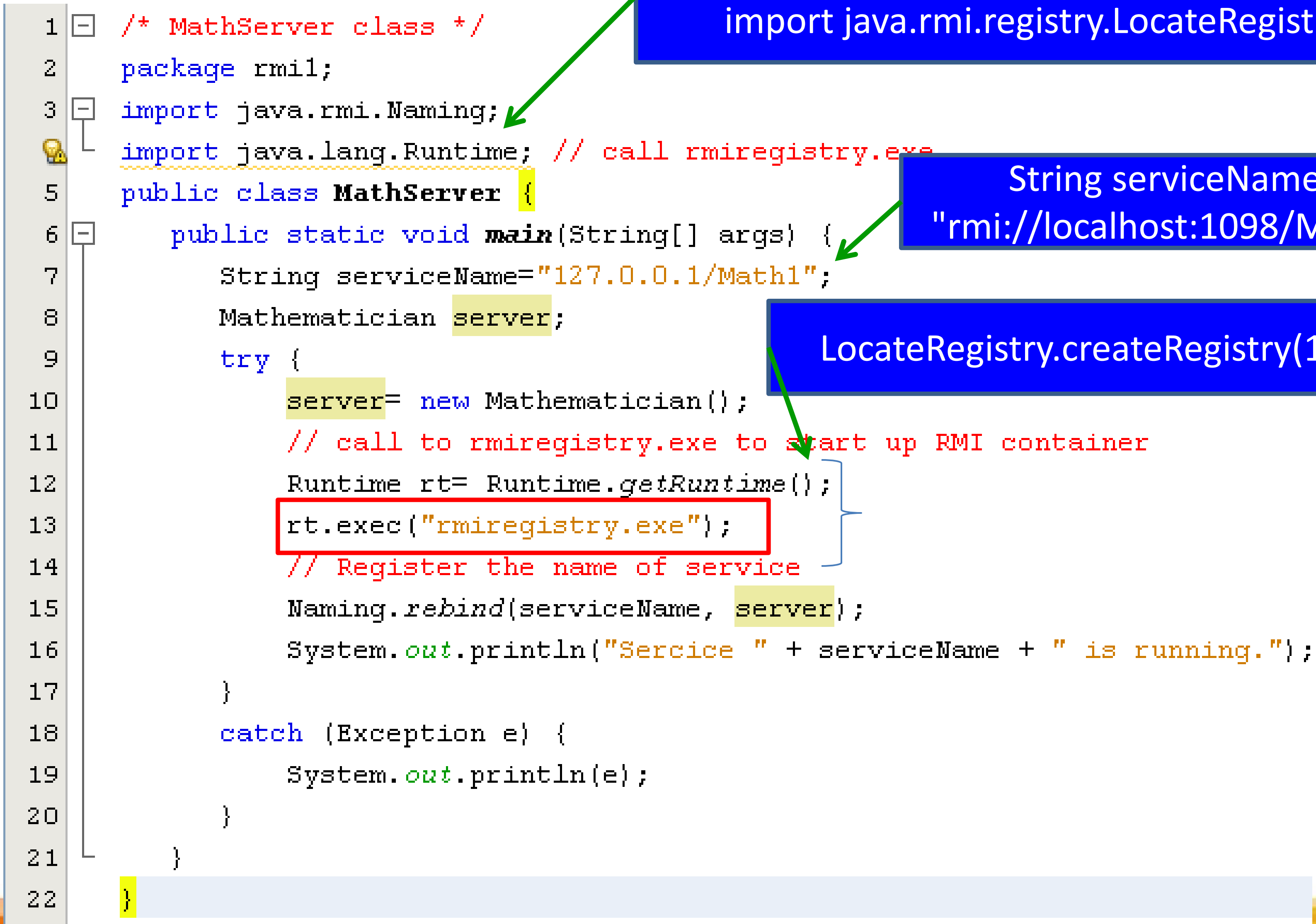
Demo 1- Overcome
Use the default RMI container in JVM

In server program

// Using default RMI container in JVM
import java.rmi.registry.LocateRegistry;

String serviceName =
"rmi://localhost:1098/Math1";

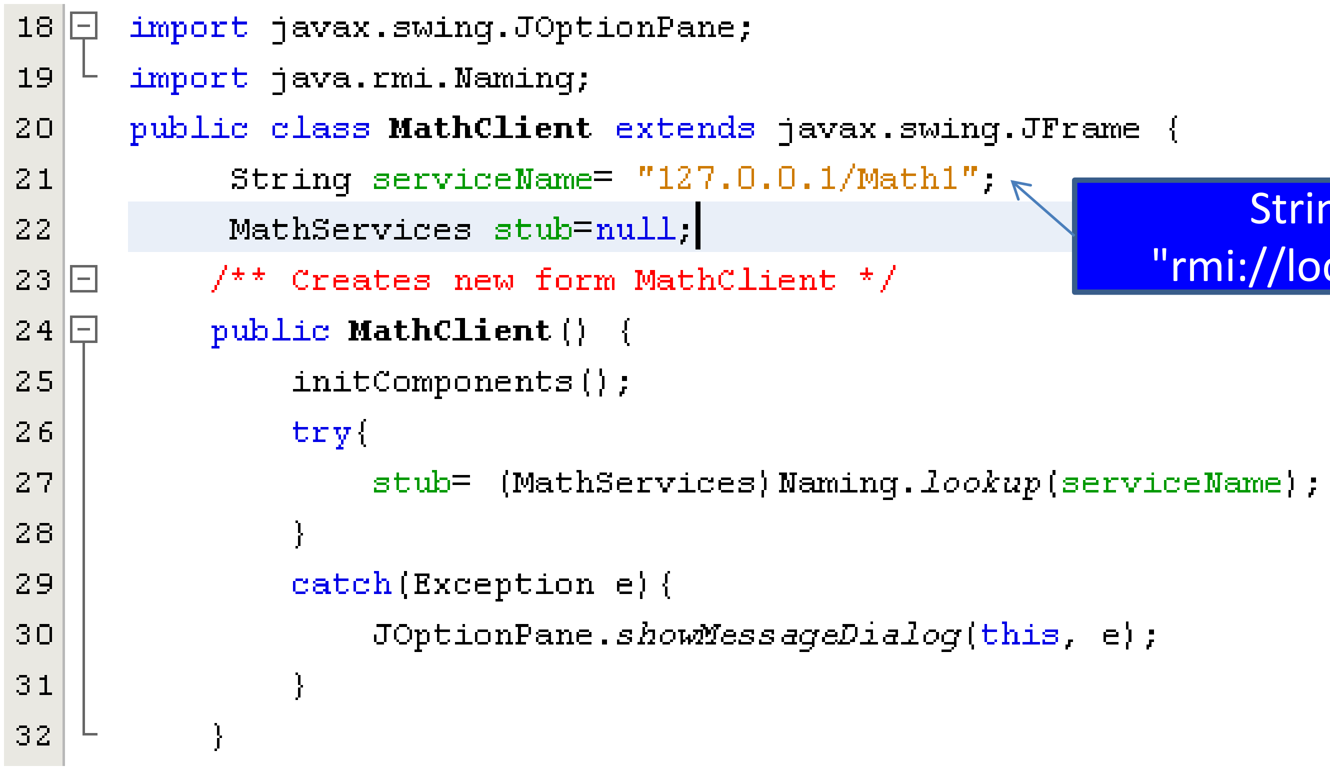
LocateRegistry.createRegistry(1098);



```
/* MathServer class */
package rmi1;
import java.rmi.Naming;
import java.lang.Runtime; // call rmiregistry.exe
public class MathServer {
    public static void main(String[] args) {
        String serviceName="127.0.0.1/Math1";
        Mathematician server;
        try {
            server= new Mathematician();
            // call to rmiregistry.exe to start up RMI container
            Runtime rt= Runtime.getRuntime();
            rt.exec("rmiregistry.exe");
            // Register the name of service
            Naming.rebind(serviceName, server);
            System.out.println("Sercice " + serviceName + " is running.");
        }
        catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

Demo 1- Overcome
Use the default RMI container in JVM

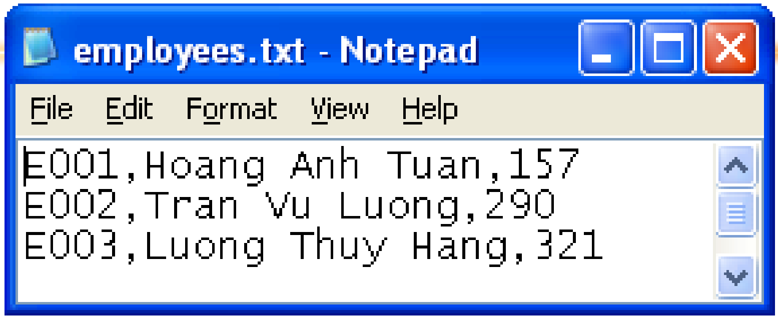
In client program



```
import javax.swing.JOptionPane;
import java.rmi.Naming;
public class MathClient extends javax.swing.JFrame {
     String serviceName= "127.0.0.1/Math1";
     MathServices stub=null;
    /** Creates new form MathClient */
    public MathClient() {
        initComponents();
        try{
            stub= (MathServices)Naming.lookup(serviceName);
        }
        catch(Exception e){
            JOptionPane.showMessageDialog(this, e);
        }
    }
```

String serviceName= "rmi://localhost:1098/Math1";

Demo 2: Data are stored in server


employees.txt - Notepad

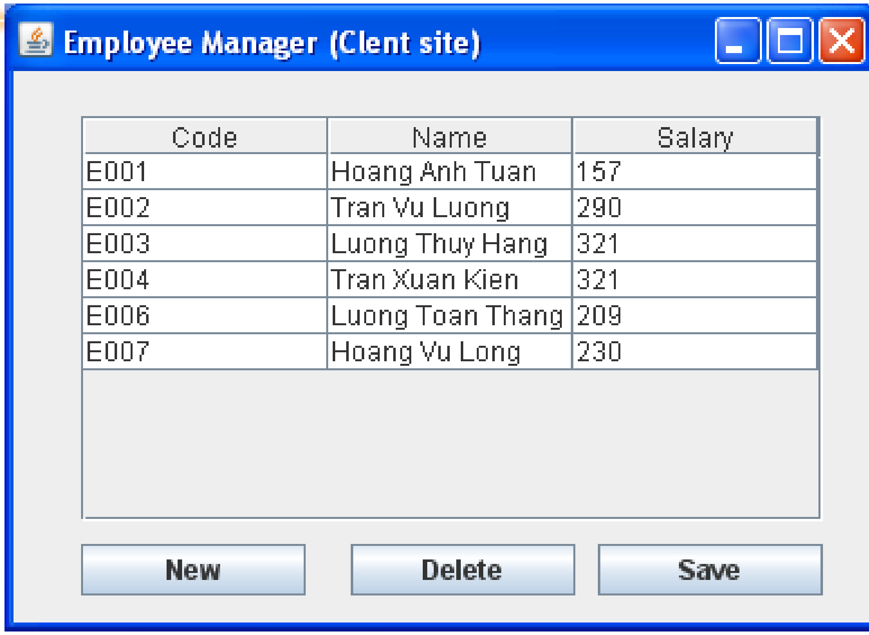
File Edit Format View Help

```
E001,Hoang Anh Tuan,157
E002,Tran Vu Luong,290
E003,Luong Thuy Hang,321
```

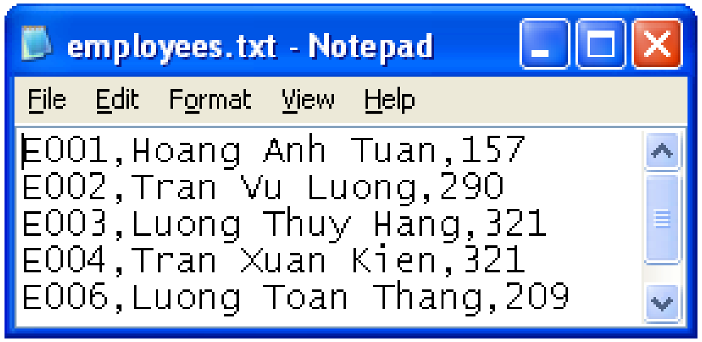
- At server side
  - An initial list of employees is stored in the **employees.txt** file (a line for an employee with the format: code, Name, salary).
  - A program running in console mode in which a remote server can support two operations:
    - Supply initial list of employees to a client program.
    - Save using override mode a list of employees transferred from a client program.

Demo 2...

- At client side:
  - Initially, a list of employees is supplied from server will be presented on a table of the GUI.
  - User can
    - Add new employee (the employee's code must have the format E000 and it is not duplicated with existing employee codes.
    - Remove an employee.
    - Update employee details.
    - Save the list on server.

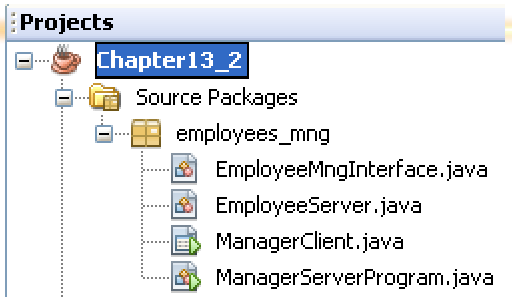


Code	Name	Salary
E001	Hoang Anh Tuan	157
E002	Tran Vu Luong	290
E003	Luong Thuy Hang	321
E004	Tran Xuan Kien	321
E006	Luong Toan Thang	209
E007	Hoang Vu Long	230

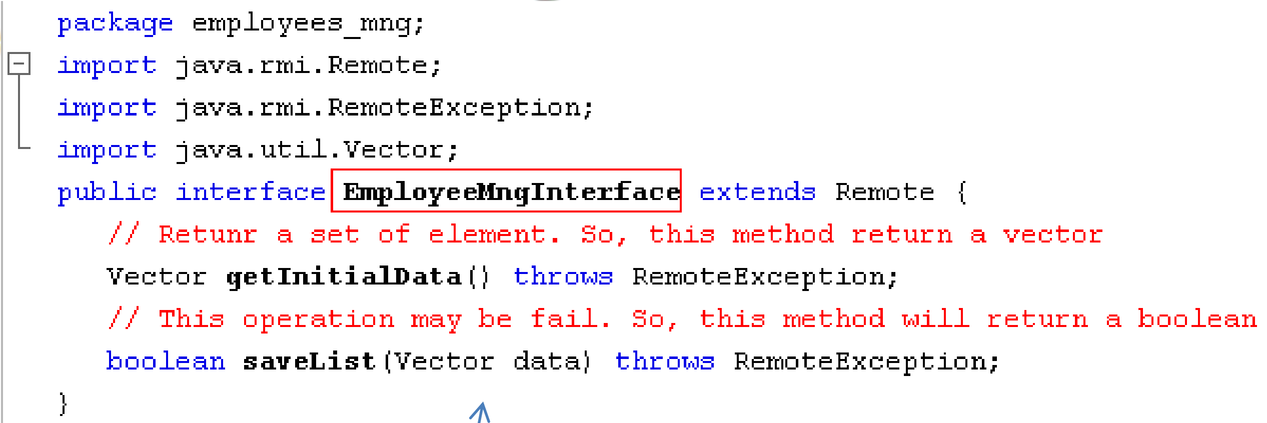


```
E001,Hoang Anh Tuan,157
E002,Tran Vu Luong,290
E003,Luong Thuy Hang,321
E004,Tran Xuan Kien,321
E006,Luong Toan Thang,209
```

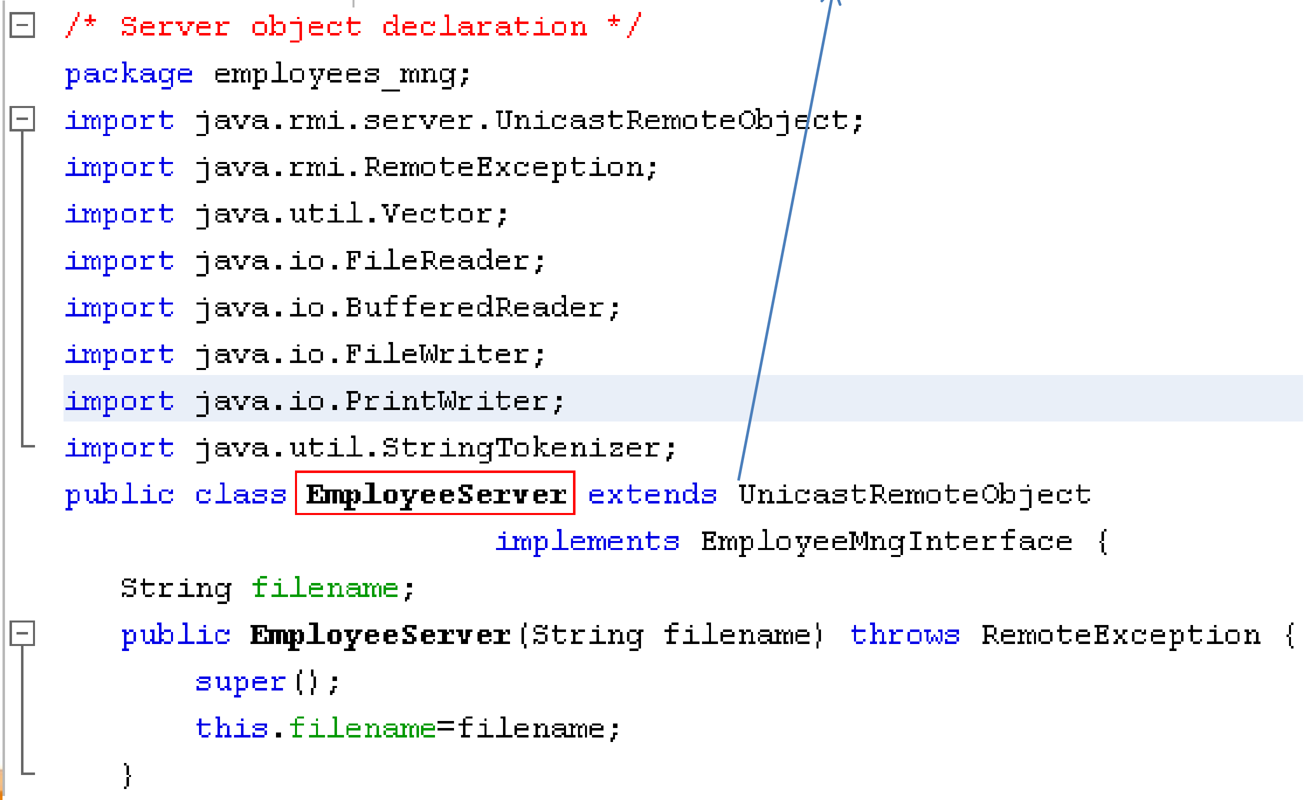
Demo 2: Remote Interface and Server Object



```
Projects
Chapter13_2
  Source Packages
    employees_mng
      EmployeeMngInterface.java
      EmployeeServer.java
      ManagerClient.java
      ManagerServerProgram.java
```



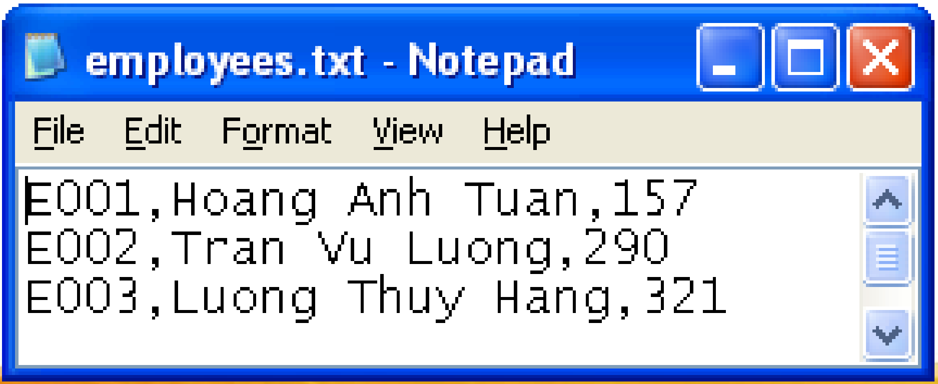
```
package employees_mng;
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.Vector;
public interface EmployeeMngInterface extends Remote {
   // Retunr a set of element. So, this method return a vector
   Vector getInitialData() throws RemoteException;
   // This operation may be fail. So, this method will return a boolean
   boolean saveList(Vector data) throws RemoteException;
}
```



```
/* Server object declaration */
package employees_mng;
import java.rmi.server.UnicastRemoteObject;
import java.rmi.RemoteException;
import java.util.Vector;
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.util.StringTokenizer;
public class EmployeeServer extends UnicastRemoteObject
                              implements EmployeeMngInterface {
   String filename;
   public EmployeeServer(String filename) throws RemoteException {
      super();
      this.filename=filename;
   }
```

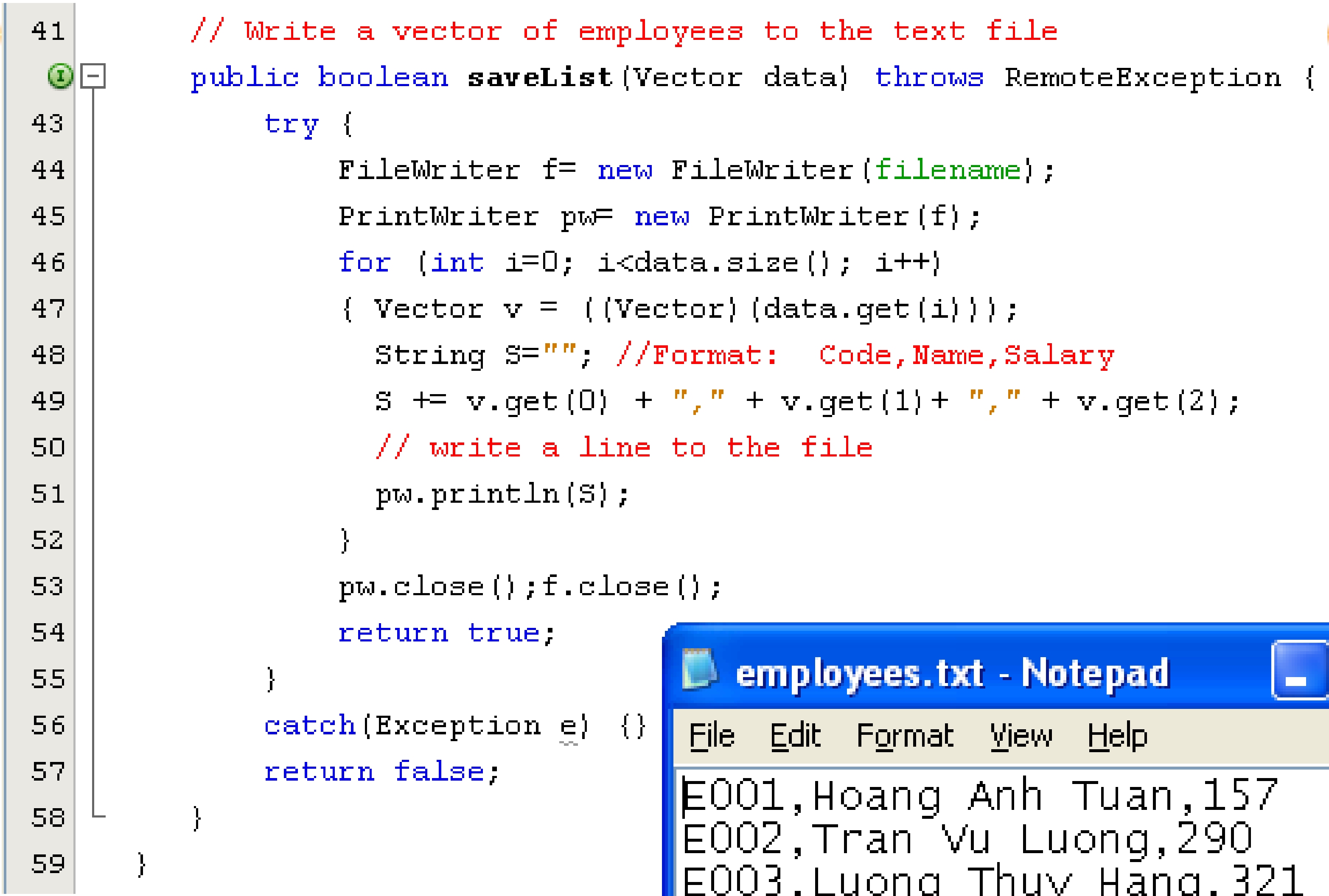
Demo 2: Server side

```java
    // Get initial employees from the text file. Return a vector
    // Format: Code,Name,Salary
    public Vector getInitialData() throws RemoteException {
        Vector data= new Vector(0);
        try {
            FileReader f= new FileReader(filename);
            BufferedReader br= new BufferedReader(f);
            String line;
            StringTokenizer stk;
            String code, name; int salary;
            while ((line=br.readLine())!=null){
               stk= new StringTokenizer(line, ",");
               Vector v= new Vector();
               v.add(stk.nextToken()); // code
               v.add(stk.nextToken()); // name
               v.add(Integer.parseInt(stk.nextToken()));// salary
               data.add(v);
            }
            br.close();f.close();
        }
        catch (Exception e) {}
        return data;
    }
```

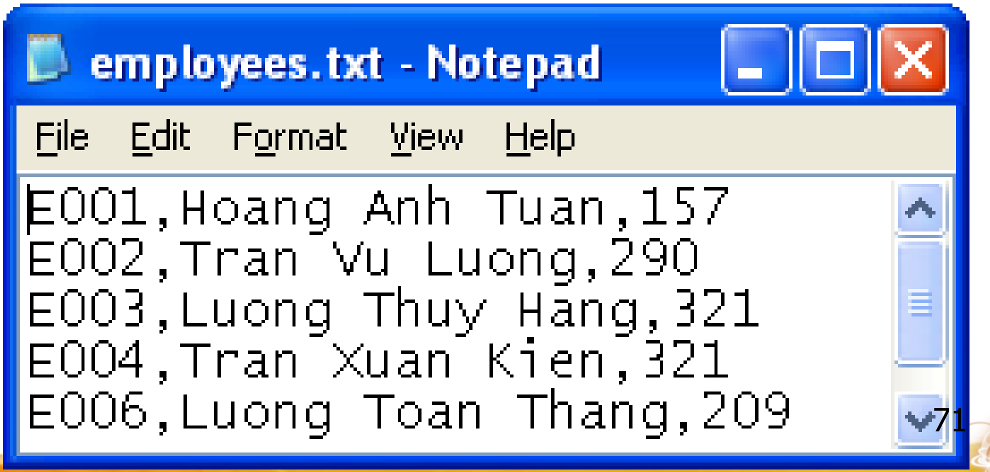


```
E001,Hoang Anh Tuan,157
E002,Tran Vu Luong,290
E003,Luong Thuy Hang,321
```

Demo 2: Server Object...

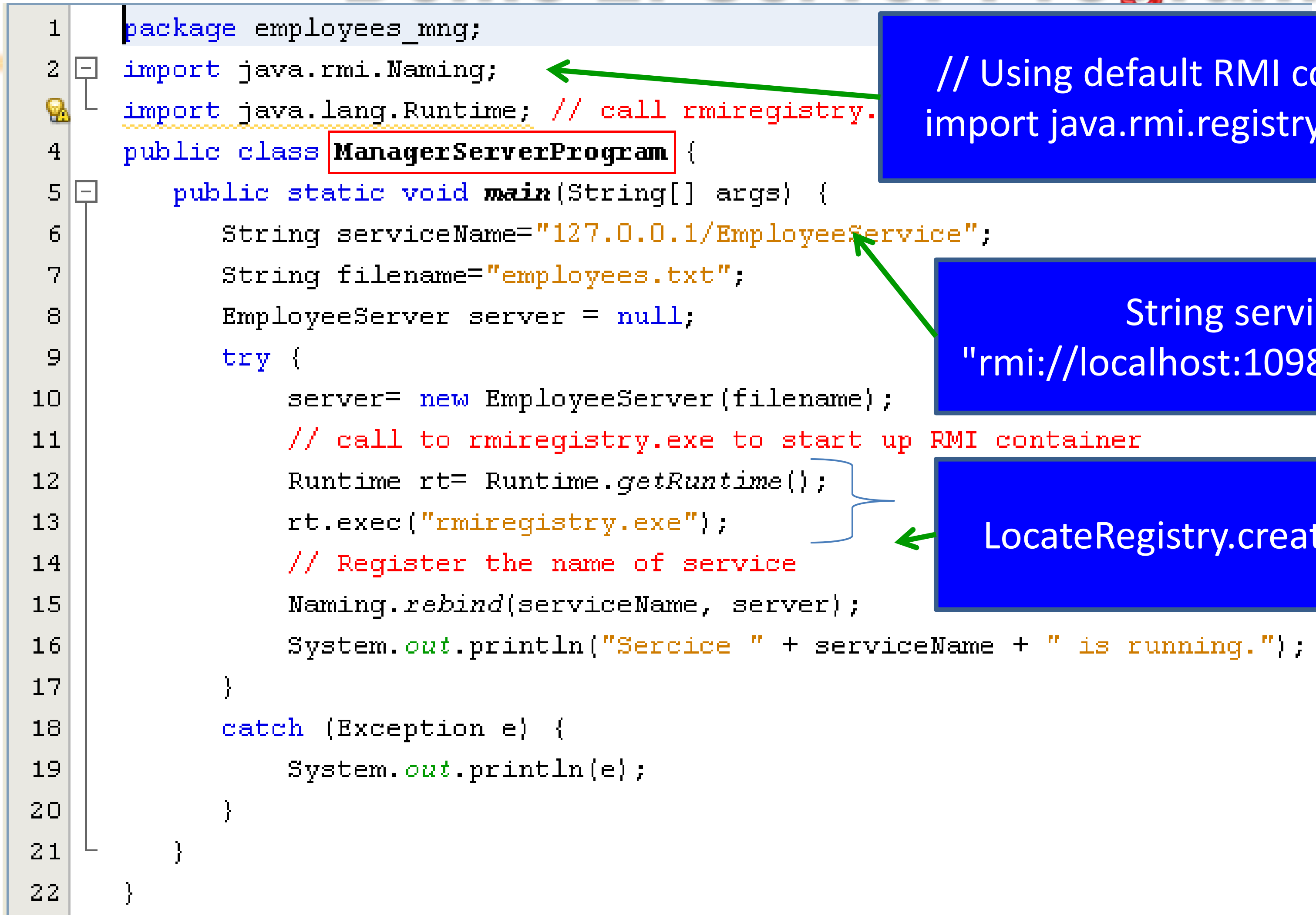


```java
   // Write a vector of employees to the text file
   public boolean saveList(Vector data) throws RemoteException {
       try {
           FileWriter f= new FileWriter(filename);
           PrintWriter pw= new PrintWriter(f);
           for (int i=0; i<data.size(); i++)
           { Vector v = ((Vector)(data.get(i)));
             String S=""; //Format:  Code,Name,Salary
             S += v.get(0) + "," + v.get(1)+ "," + v.get(2);
             // write a line to the file
             pw.println(S);
           }
           pw.close();f.close();
           return true;
       }
       catch(Exception e) {}
       return false;
   }
}
```



```
E001,Hoang Anh Tuan,157
E002,Tran Vu Luong,290
E003,Luong Thuy Hang,321
E004,Tran Xuan Kien,321
E006,Luong Toan Thang,209
```

Demo 2: Server Program



```java
package employees_mng;
import java.rmi.Naming;
import java.lang.Runtime; // call rmiregistry.
public class ManagerServerProgram {
    public static void main(String[] args) {
        String serviceName="127.0.0.1/EmployeeService";
        String filename="employees.txt";
        EmployeeServer server = null;
        try {
            server= new EmployeeServer(filename);
            // call to rmiregistry.exe to start up RMI container
            Runtime rt= Runtime.getRuntime();
            rt.exec("rmiregistry.exe");
            // Register the name of service
            Naming.rebind(serviceName, server);
            System.out.println("Sercice " + serviceName + " is running.");
        }
        catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

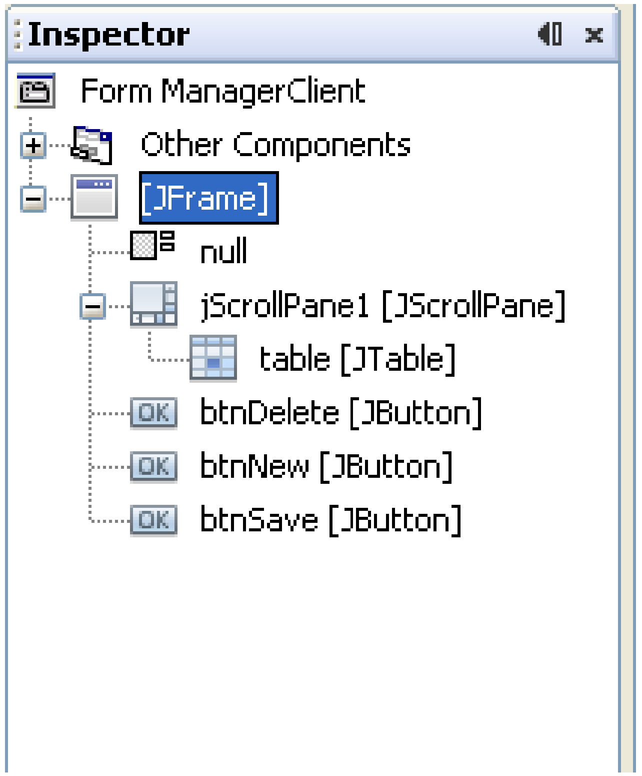
// Using default RMI container in JVM
import java.rmi.registry.LocateRegistry;

String serviceName =
"rmi://localhost:1098/EmployeeService”

LocateRegistry.createRegistry(1098);

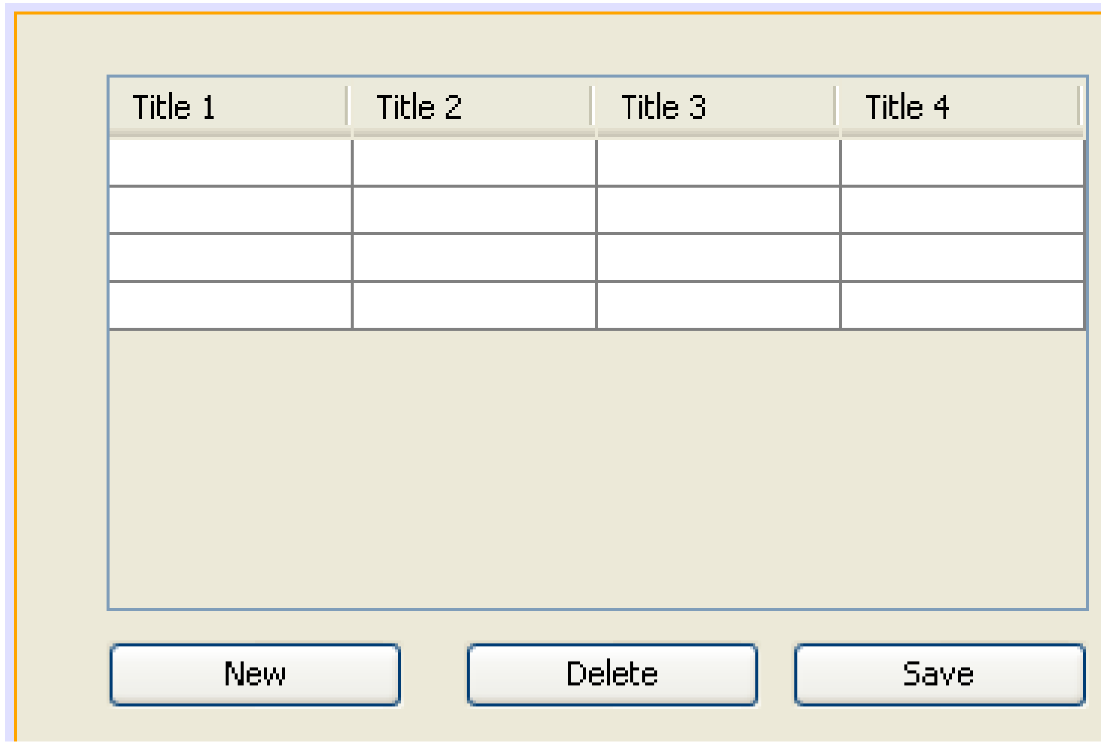
RMI Demo 2.

➢ Client Program



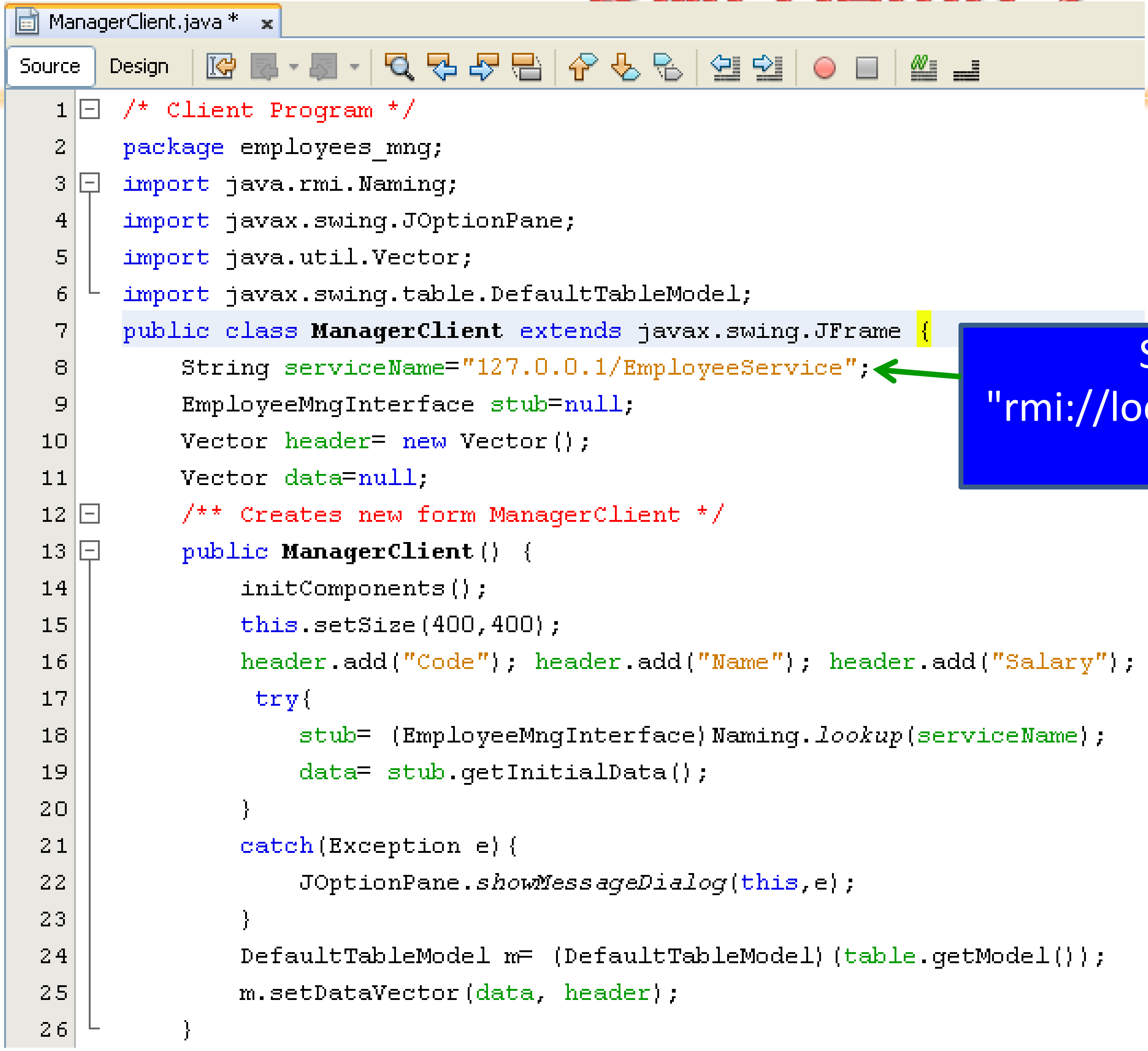
Inspector

Form ManagerClient
- Other Components
- [JFrame]
  - null
  - jScrollPane1 [JScrollPane]
    - table [JTable]
  - btnDelete [JButton]
  - btnNew [JButton]
  - btnSave [JButton]



Title 1	Title 2	Title 3	Title 4

RMI Demo 2



```
/* Client Program */
package employees_mng;
import java.rmi.Naming;
import javax.swing.JOptionPane;
import java.util.Vector;
import javax.swing.table.DefaultTableModel;
public class ManagerClient extends javax.swing.JFrame {
    String serviceName="127.0.0.1/EmployeeService";
    EmployeeMngInterface stub=null;
    Vector header= new Vector();
    Vector data=null;
    /** Creates new form ManagerClient */
    public ManagerClient() {
        initComponents();
        this.setSize(400,400);
        header.add("Code"); header.add("Name"); header.add("Salary");
         try{
            stub= (EmployeeMngInterface)Naming.lookup(serviceName);
            data= stub.getInitialData();
        }
        catch(Exception e){
            JOptionPane.showMessageDialog(this,e);
        }
        DefaultTableModel m= (DefaultTableModel)(table.getModel());
        m.setDataVector(data, header);
    }
```

String serviceName = "rmi://localhost:1098/EmployeeService”

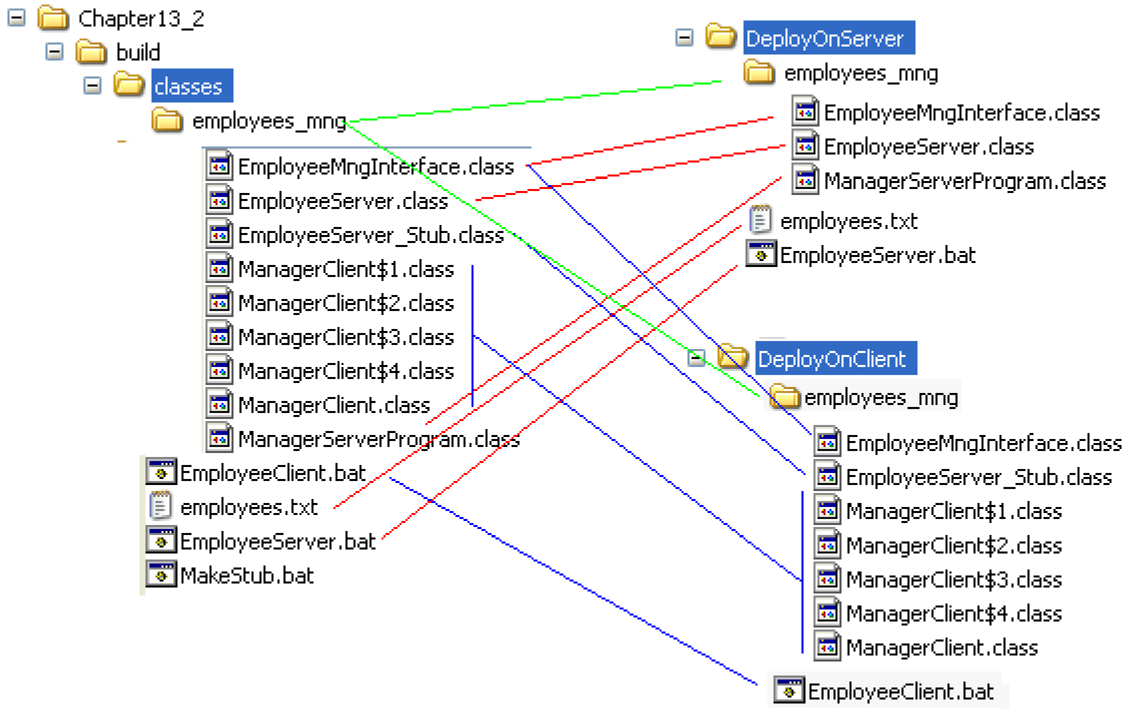
RMI Demo 2.

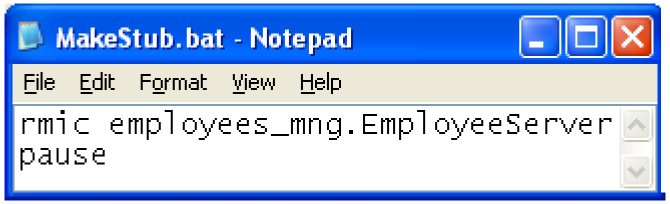
```
  private void btnDeleteActionPerformed(java.awt.event.ActionEvent evt) {
      // TODO add your handling code here:
      int pos= table.getSelectedRow();
      data.remove(pos);
      table.updateUI();
  }
```

```
  private void btnNewActionPerformed(java.awt.event.ActionEvent evt) {
      // TODO add your handling code here:
      Vector v= new Vector ();
      v.add(""); v.add(""); v.add(0);
      data.add(v);
      table.updateUI();
      int lastRow=data.size()-1;
      table.addRowSelectionInterval(lastRow,lastRow);
  }
```

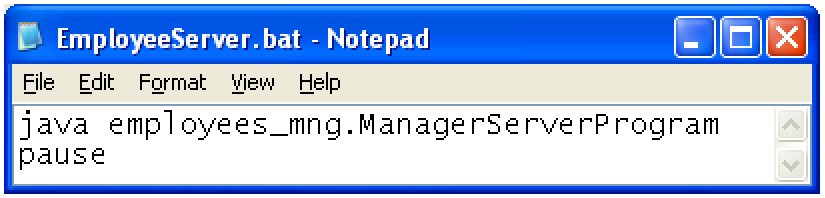
```
private void btnSaveActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    try {
      if(stub.saveList(data)==true)
          JOptionPane.showMessageDialog(this, "Saved.");
      else
          JOptionPane.showMessageDialog(this, "Sorry. Data can not be saved");
    }
    catch(Exception e){
        JOptionPane.showMessageDialog(this, e);
    }
}
```

RMI Demo 2. - Deploying

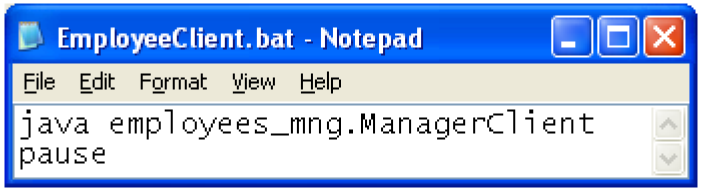




```
rmic employees_mng.EmployeeServer
pause
```



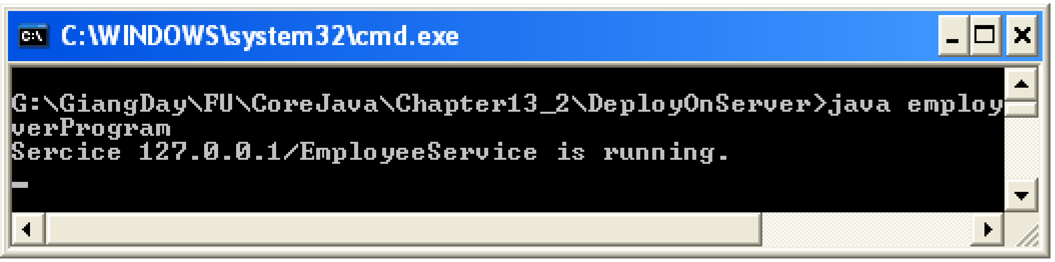
```
java employees_mng.ManagerServerProgram
pause
```



```
java employees_mng.ManagerClient
pause
```

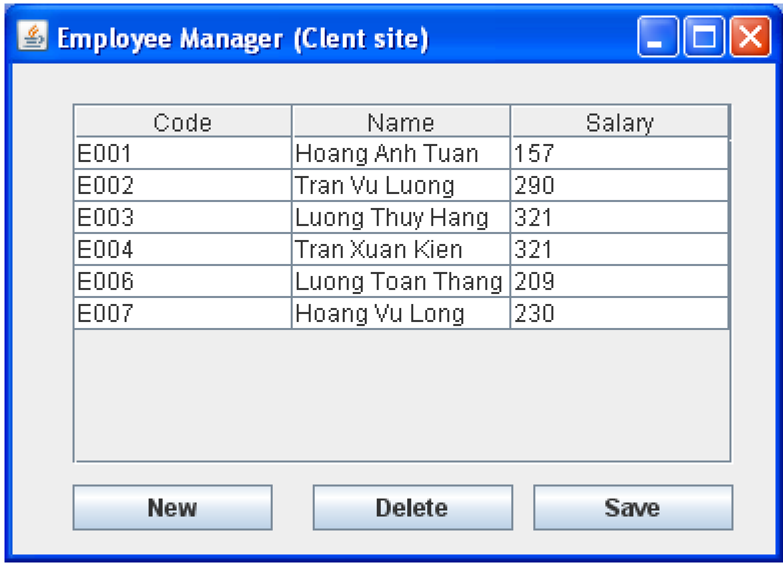
Result:

Step 1- Run server program



```
G:\GiangDay\FU\CoreJava\Chapter13_2\DeployOnServer>java employ
verProgram
Sercice 127.0.0.1/EmployeeService is running.
_
```

Step 2- Run client program



Code	Name	Salary
E001	Hoang Anh Tuan	157
E002	Tran Vu Luong	290
E003	Luong Thuy Hang	321
E004	Tran Xuan Kien	321
E006	Luong Toan Thang	209
E007	Hoang Vu Long	230