TRƯỜNG ĐẠI HỌC FPT

Lecture 03
Custom Networking
Part 3

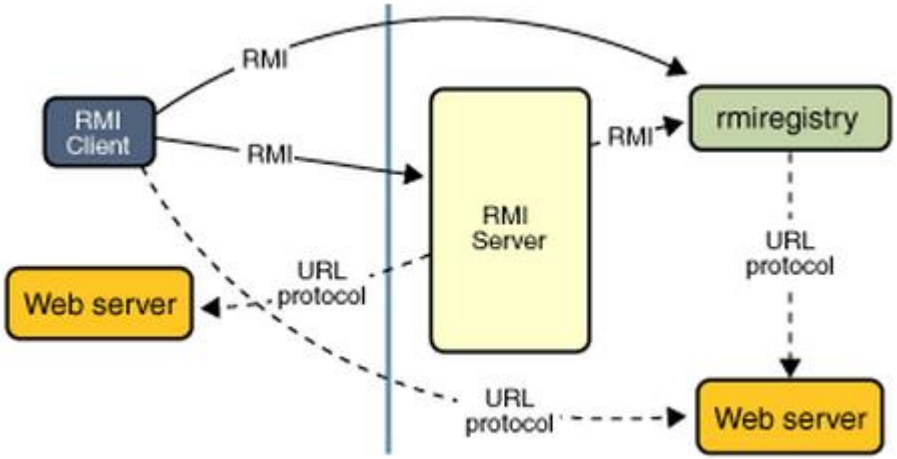
Remote Method Invocation

Book: Chapter 13- Object Streams and RMI

(The java.rmi package)

Why should we study this lecture?

- Nowadays, distributed applications are popular. People need large applications, running based on a computer network (local area networks-LANs- or wide area network-WAN), including many sites working concurrently. Do you want to create such applications?
- How do Java distributed applications work?



From Java8-Tutorial

Contents

- Object Streams and Serialization
- Java Remote Method Invocation (RMI)
- Demonstrations

1- Object Streams & Serialization

- Serialization: a process that converts object's state to a byte stream .
- Do you want to make yourself the way of serialization instead of the Java default one?
- java.io.**Serializable** : **no method is declared**
  - java.io.**Externalizable**:

Methods are declared in the Externalizable Interface

void **readExternal**(ObjectInput in)

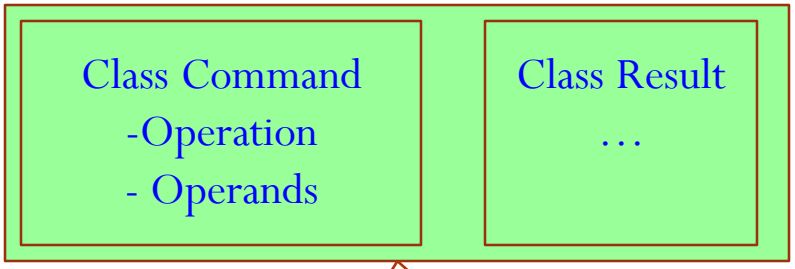
The object implements the readExternal method to restore its contents by calling the methods of DataInput for primitive types and readObject for objects, strings and arrays.

void **writeExternal**(ObjectOutput out)

The object implements the writeExternal method to save its contents by calling the methods of DataOutput for its primitive values or calling the writeObject method of ObjectOutput for objects, strings, and arrays.

A demonstration: Refer the book, page 446.

Remote Control Using Object Streams



Client

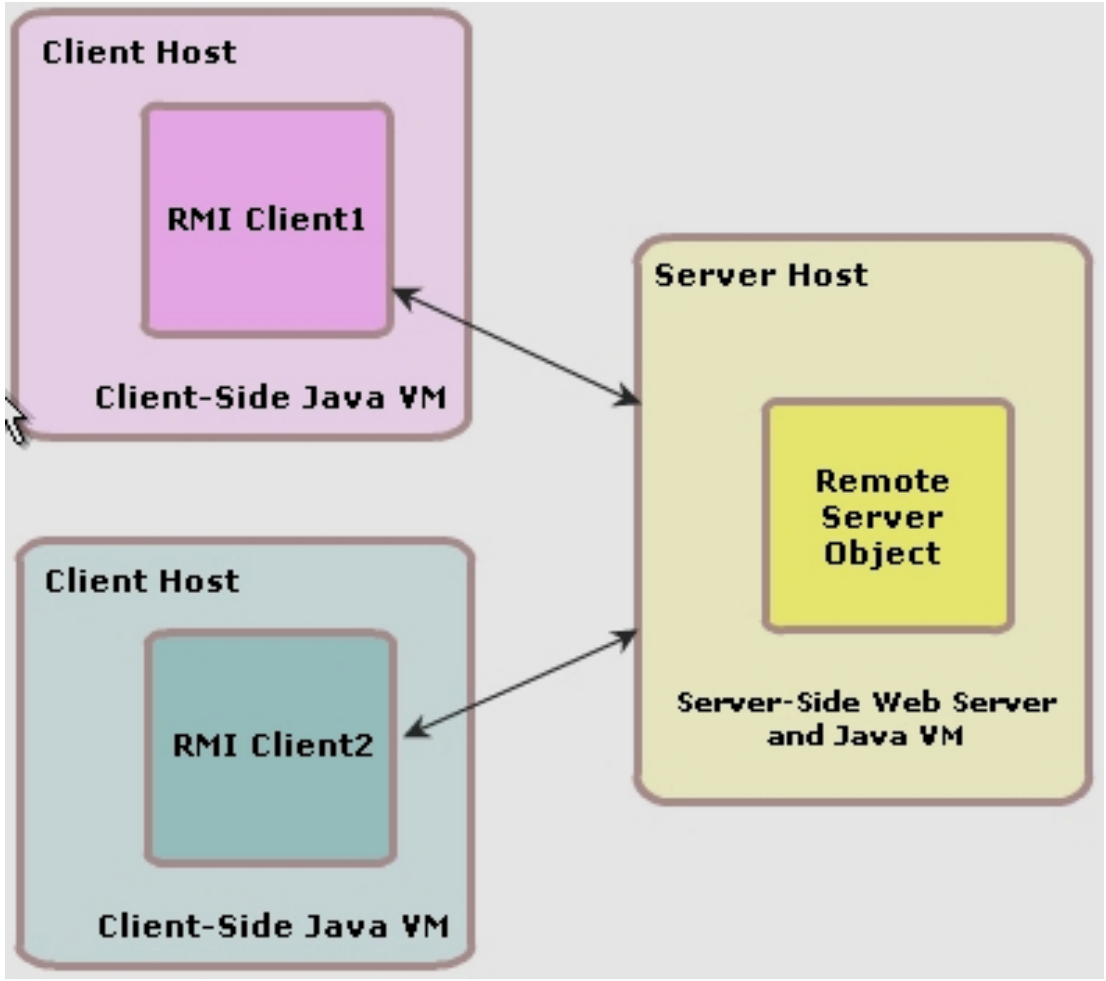
- Create a Command object;
- Connect to server
- Write it to output stream of the socket
- Wait for then get the result object from socket's input stream.
- Show the result.

Server

- Wait for a connect from client
- Get Command object from input stream of client socket.
- Execute the command received to create a Result object
- Write the result object to output stream of the client object

Demonstration: Book, page 447 .. 452

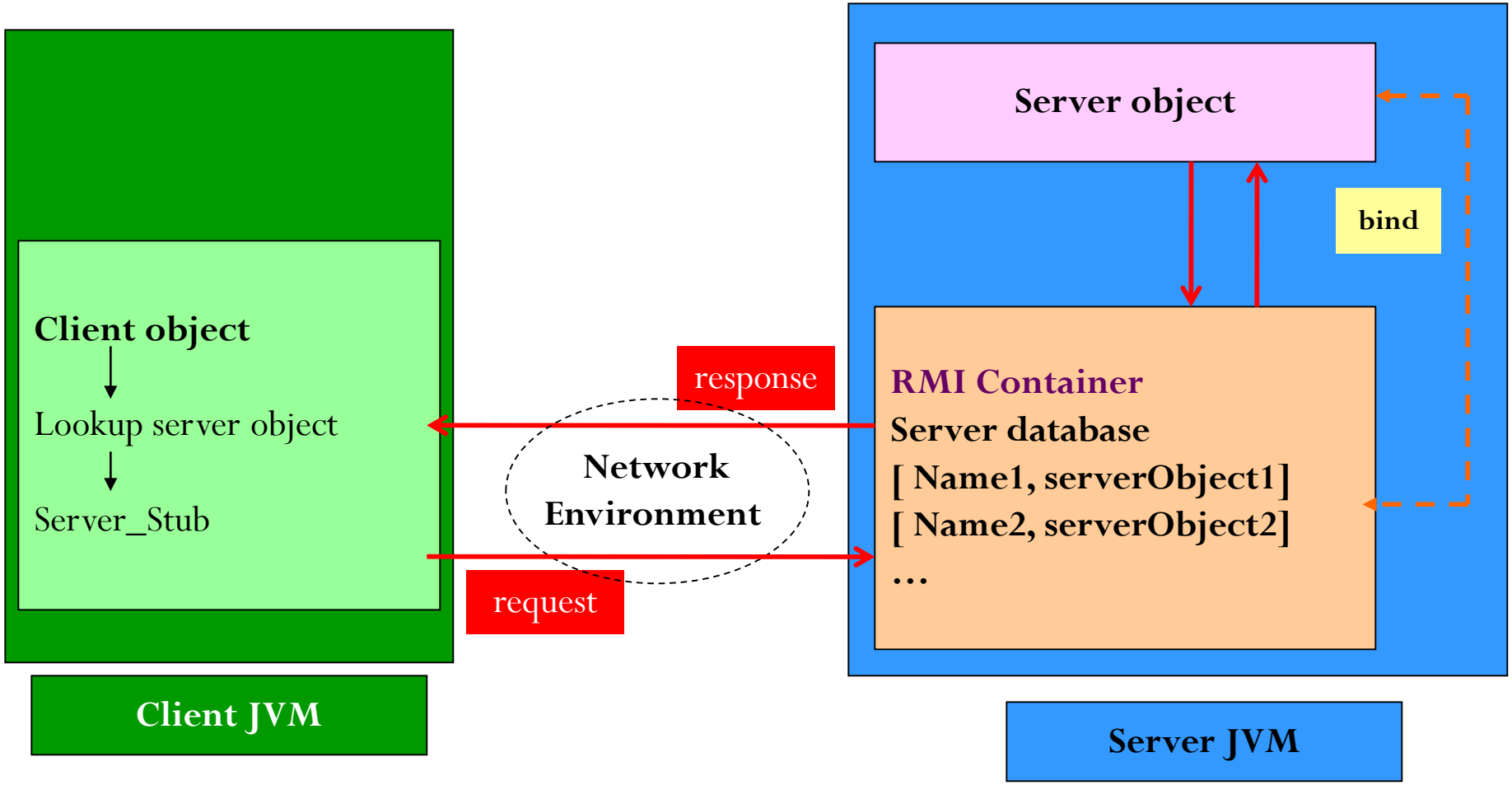
2- Remote Method Invocation (RMI)



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 Protocol (JRMP).

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

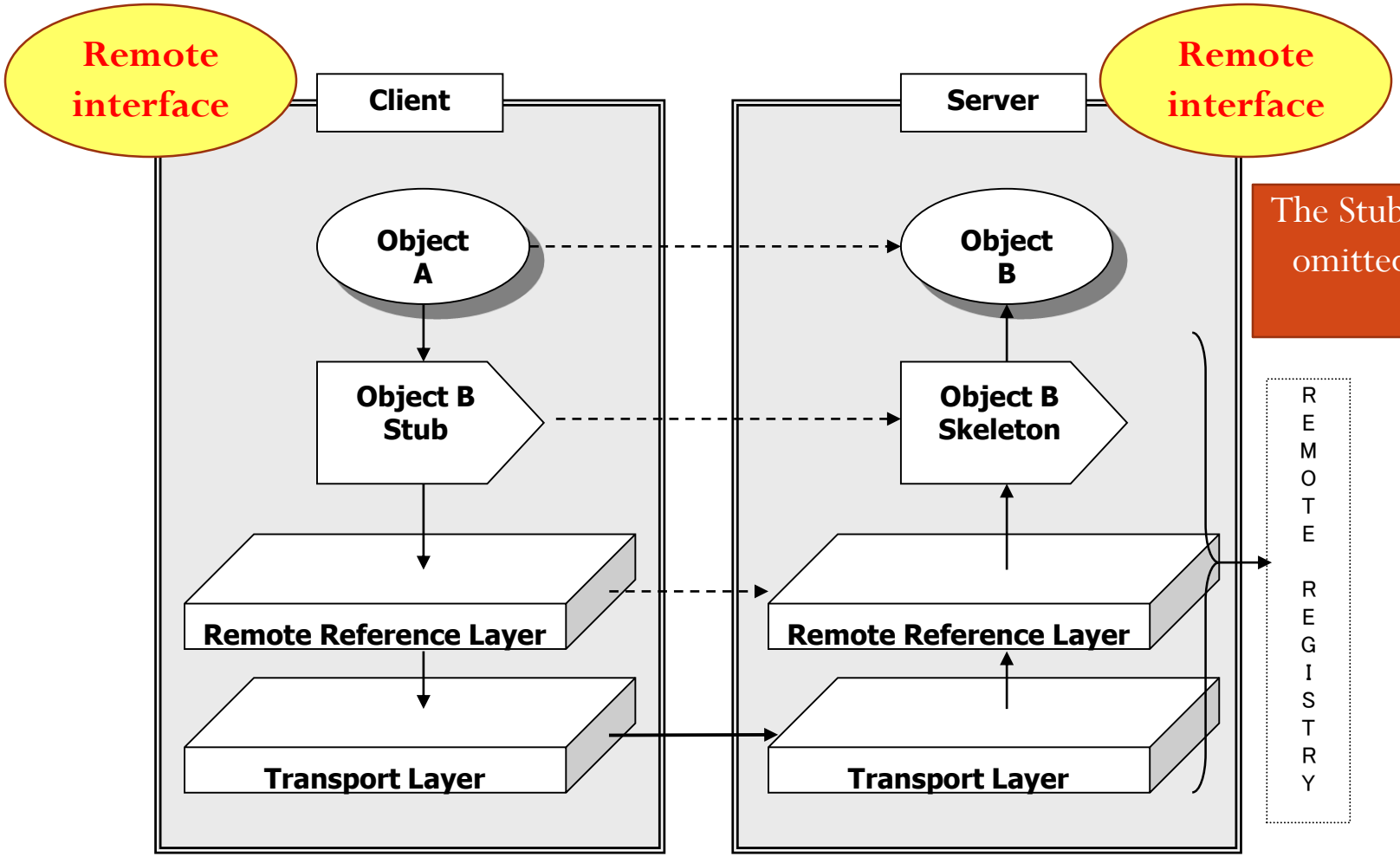
RMI…



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.

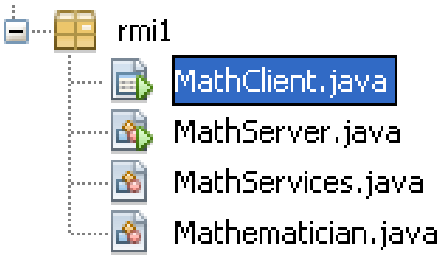
3- Create Server program using server object

4- Create the client program

5- Run apps: Start server program first then the client program.

Demo 1: Simple RMI

Step 1: Create a remote interface



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

```
/* 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;
}
```

Demo 1: Simple RMI…

Step 2: Create server class implementing remote interface

```
1  /* This class implements the MathServices interface */
2  package rmi1;
3  import java.rmi.server.UnicastRemoteObject;
4  import java.rmi.RemoteException;
5  public class Mathematician extends UnicastRemoteObject
6                                 implements MathServices  {
7    public  Mathematician() throws RemoteException {}
8    public double add(double x, double y) throws RemoteException{
9        return x+y;
10   }
11   public double subtract(double x, double y) throws RemoteException {
12       return x-y;
13   }
14 }
```

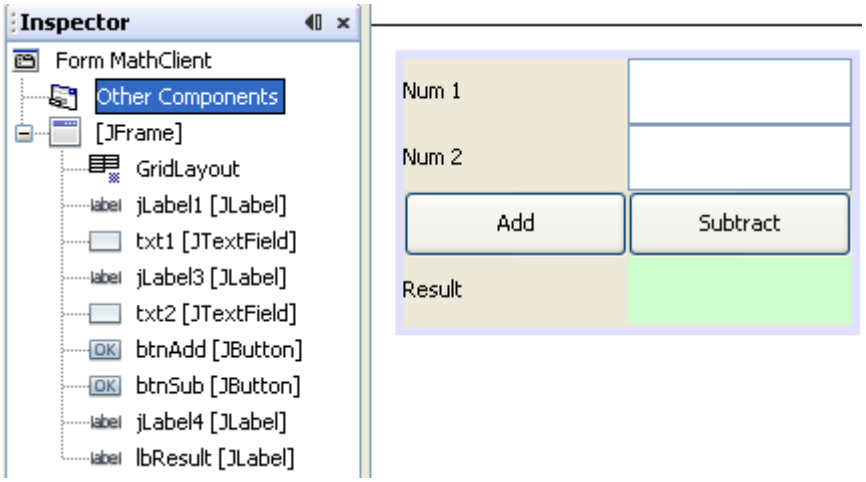
Demo 1: Simple RMI…

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

```
import java.rmi.Naming;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

/**
 *
 * @author quocv
 */
public class MathServer {
    public static void main(String[] args) {
        String serviceName="rmi://127.0.0.1:3000/Math1";
        MathServices server;

        try {
            server = new Mathmatician();
            Registry registry=LocateRegistry.createRegistry(3000);

            Naming.rebind(serviceName, server);
            System.out.println("Sercice " + serviceName + "is running ");
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }

}
```

Demo 1: Simple RMI…

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




```
import java.rmi.Naming;
import javax.swing.JOptionPane;

public class MathClient extends javax.swing.JFrame {

    String serviceName = "rmi://127.0.0.1:3000/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…

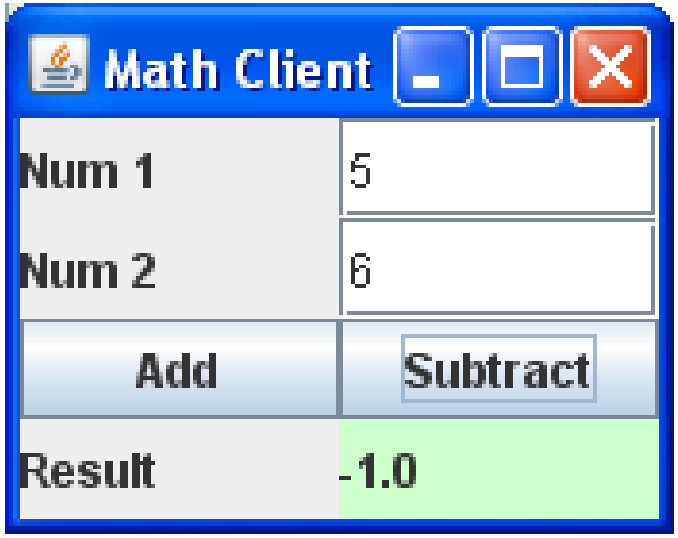
```
private void btnAddActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    if (stub!=null) {
        double x= Double.parseDouble(txt1.getText());
        double y= Double.parseDouble(txt2.getText());
        try {
            double result= stub.add(x, y);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

Call methods of remote object

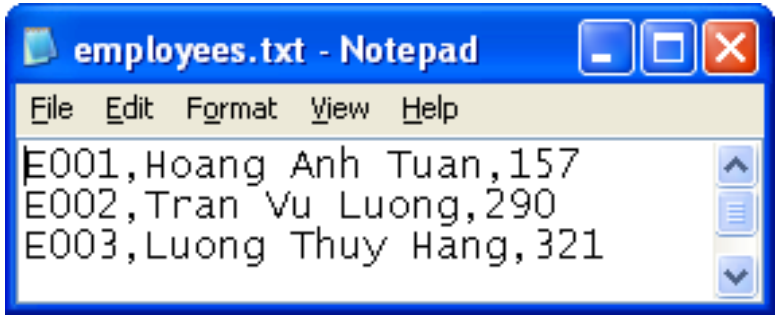
```
private void btnSubActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
            if (stub!=null) {
        double x= Double.parseDouble(txt1.getText());
        double y= Double.parseDouble(txt2.getText());
        try {
            double result= stub.subtract(x,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

}
```

Demo 1: Simple RMI…

Step 6: Run server program first then client program



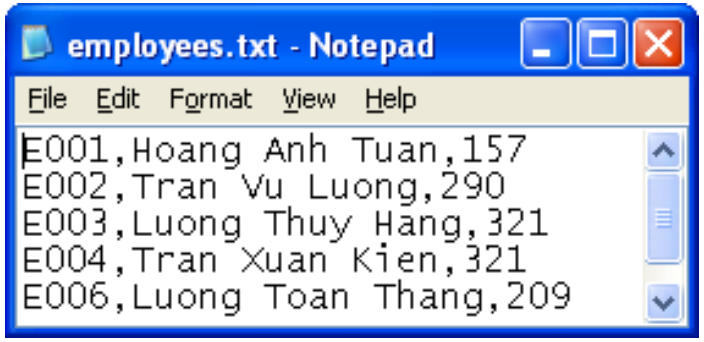
Demo 2: Data are stored in server



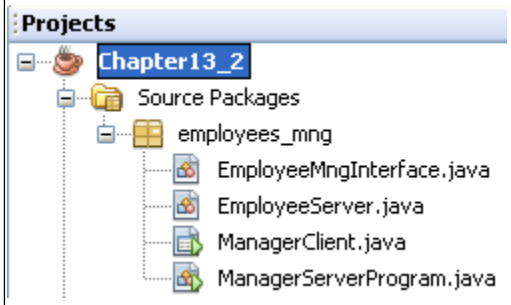
- At server side
  - An initial list of employees is stored in the **employees.txt** file (a line for an employee with the format: code, Name, salary).
  - A program running in GUI mode in which a remote server can support two operations:
    - Supply initial list of employees to a client program.
 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



Demo 2: Remote Interface and Server Object

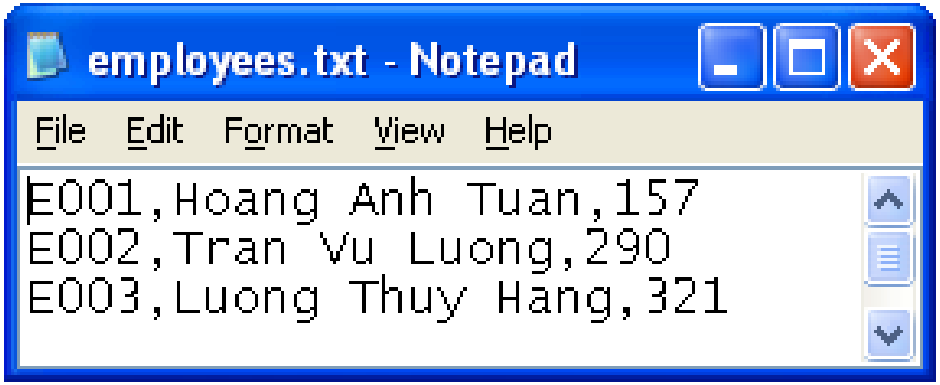



```java
package employees_mng;
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.Vector;
public interface EmployeeMngInterface extends Remote {
    // Retunr a set of element. So, this method return a vector
    Vector getInitialData() throws RemoteException;
    // This operation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

```java
/* 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

```
18      // Get initial employees from the text file. Return a vector
19      // Format: Code,Name,Salary
        public Vector getInitialData() throws RemoteException {
21          Vector data= new Vector(0);
22          try {
23              FileReader f= new FileReader(filename);
24              BufferedReader br= new BufferedReader(f);
25              String line;
26              StringTokenizer stk;
27              String code, name; int salary;
28              while ((line=br.readLine())!=null){
29                  stk= new StringTokenizer(line, ",");
30                  Vector v= new Vector();
31                  v.add(stk.nextToken()); // code
32                  v.add(stk.nextToken()); // name
33                  v.add(Integer.parseInt(stk.nextToken()));// salary
34                  data.add(v);
35              }
36              br.close();f.close();
37          }
38          catch (Exception e) {}
39          return data;
40      }
```



Demo 2: Server Object…

```
41      // Write a vector of employees to the text file
        public boolean saveList(Vector data) throws RemoteException {
43          try {
44              FileWriter f= new FileWriter(filename);
45              PrintWriter pw= new PrintWriter(f);
46              for (int i=0; i<data.size(); i++)
47              { Vector v = ((Vector)(data.get(i)));
48                String S=""; //Format:  Code,Name,Salary
49                S += v.get(0) + "," + v.get(1)+ "," + v.get(2);
50                // write a line to the file
51                pw.println(S);
52              }
53              pw.close();f.close();
54              return true;
55          }
56          catch(Exception e) {}
57          return false;
58      }
59  }
```

employees.txt - Notepad

File Edit Format View Help

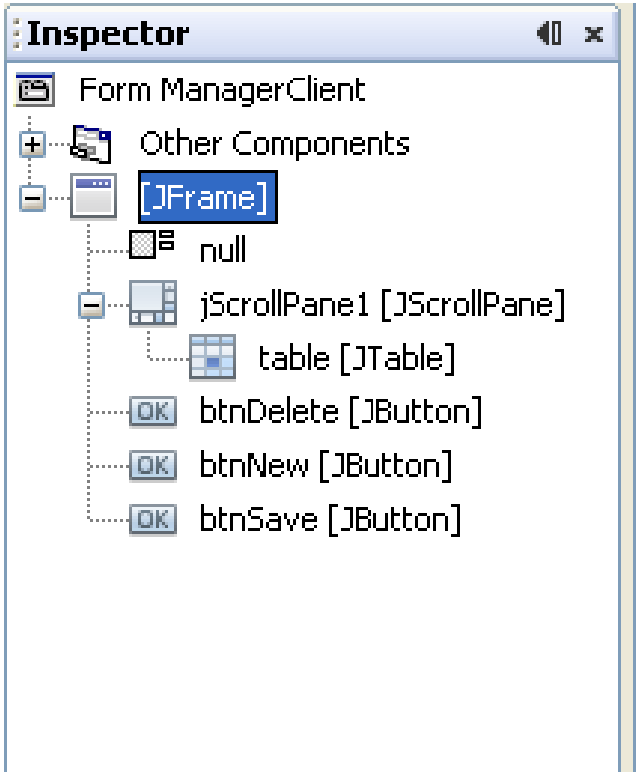
```
E001,Hoang Anh Tuan,157
E002,Tran Vu Luong,290
E003,Luong Thuy Hang,321
E004,Tran Xuan Kien,321
E006,Luong Toan Thang,209
```

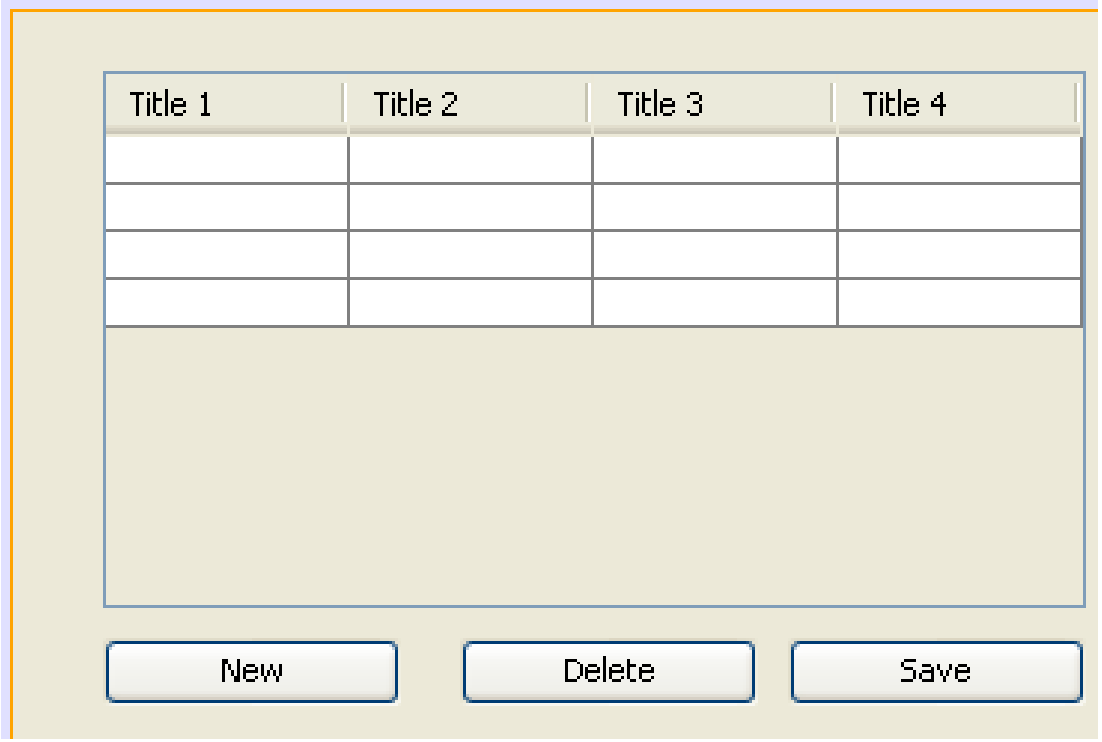
Demo 2: Server Program

```java
import java.rmi.Naming;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class ManagerServerProgram {
    public static void main(String[] args) {
        String serviceName="rmi://127.0.0.1:1098/EmployeeService";
        String filename="employees.txt";
        EmployeeServer server = null;
        try {
            server = new EmployeeServer(filename);

            Registry r = LocateRegistry.createRegistry(1098);
            Naming.rebind(serviceName, server);
            System.out.println("Service " + serviceName + " is running.");
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

RMI Demo 2.

- Client Program





RMI Demo 2.

```
import java.rmi.Naming;
import java.util.Vector;
import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;


public class ManagerClient extends javax.swing.JFrame {
    String serviceName ="rmi://localhost:1098/EmployeeService";
    EmployeeMngInterface stub = null;
    Vector header = new Vector();
    Vector data = null;

    public ManagerClient() {
        initComponents();
        this.setSize(500, 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);
    }
```

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);
    }
}
```

Result:

Step 1- Run server program

Output - PRJ_WS6 (run)

```
run:
Service rmi://127.0.0.1:1098/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

Summary

- Remote Control using Object Streams
- Remote Method Invocation
  - Remote interface, Class for Server object
  - Server Program, Client Program

Thank You