Remote Method Invocation (RMI)

Instructor: Louise Moser
TA: Yung-Ting Chuang
Quarter: Spring 2012

Project2 Addendum

- The goal for a patient to connect to multiple doctors is that he/she keep all his/her connections all the time.
- All the doctors connected to this patient "SHOULDN'T" drop their connections until when this patient exit his/her GUI

Algorithm1

- Algorithm1.

1. When a patient (Pat1) connects to both Doc1 and Doc2.
2. 2. Pat1 authenticate with Doc1 and Doc2.
3. Pat1 "broadcast" time requests to Doc1 and Doc2
4. Doc1 and Doc2 "both" send him back the available time.
5. Pat1 receive both times from Doc1 and Doc2, and display these times on GUI
6. Pat1 decides to make 9am with Doc1, and "broadcast" this appointment requests to both Doc1 and Doc2.
7. Doc1 receives that and see this message is for him, make appointment and reply back to Pat1. Doc2 knows that appointment request is not for him, ignore it and drop the lock.

Algorithm2

1. When a patient (Pat1) connects to both Doc1 and Doc2.
2. Pat1 authenticate with Doc1 and Doc2.
3. PatientGUI first ask Pat1 which doctor he/she should send time requests, and send it (For Example Pat1 chose Doc1)
4. Then Doc1 sends his time availability...
5. Pat1 receive available time from Doc1, and display these times on GUI
6. Pat1 decides to make 9am with Doc1, so send appointment request to Doc1
7. Doc1 receives that and see this message is for him, make appointment and reply back to Pat1.

Algorithm3

1. When a patient (Pat1) connects to both Doc1 and Doc2.
2. Pat1 authenticate with Doc1 and Doc2.
3. PatientGUI first ask Pat1 which doctor he/she wants to send request message to (For example Pat1 selects Doc1). Pat1 broadcast time request to both Doc1 and Doc2, along with XML "<to>Doc1</to>" in its message.
4. Doc1 and Doc2 both receive this XML messages, Doc1 sees this message is for him, so reply back to Pat1 with his/her time slots. Doc2 basically would just ignore it
5. Pat1 decides to make 9am with Doc1. So he/she sends appointment request to both Doc1 and Doc2, but with "<to>Doc1</to>" in its XML message.
   Doc1 receives that and see this message is for him, make appointment and reply back to Pat1.
   Doc2 would simply ignore it since this message is not for him...

Administrative Stuff…

- This is how I grade your projects: total100 pt
  - Report: 10 pt
  - Demo: (regarding to grading sheet) 10 pt
  - Submission: 10 pt
  - Programming: 70 pt
  - Extra Credit: (extra feature you built for project) – Max 5 pt
- This is the break down for the programming part:
  - First see everything works: 10 pt
  - I ask random questions to help me determine your overall efforts toward your projects: 45 pt
    - If you did your projects by yourself and attend discussions & lectures regularly, you will be ok.
    - You can't expect what I will ask, so it will be fair to everyone.
  - Then I compile & test your work at home: 5 pt
  - Examine your codes: 10 pt

RMI In General

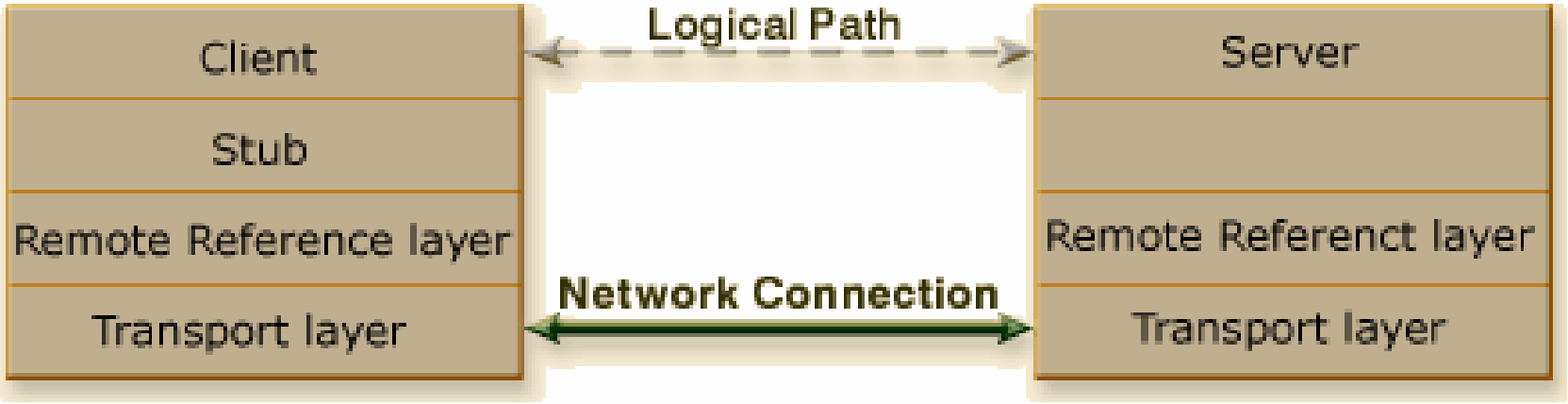
- Java objects can be invoked from other Java virtual machines, or possibly on different hosts.
- To the client and server, the program operates much like it is local to their environment and within a single memory space
- It is a Java application programming interface that performs the object-oriented equivalent of remote procedure calls.

More on RMI

- Resources can be physically located where they make the most sense
- Dedicated machines can be configured optimally to perform specialized operations
- Heavy processing can be allocated to the best suited hardware, data processing can be handled by centralized data servers, and web servers can just serve web pages
- RMI is a proven framework to design and implement a distributed application in a Java environment

Architecture of RMI

- The client is defined as the application which uses a remote object
- The server is defined as the machine which hosts the remote object
- Communication is two way. Below is the standard simple diagram showing the relationship



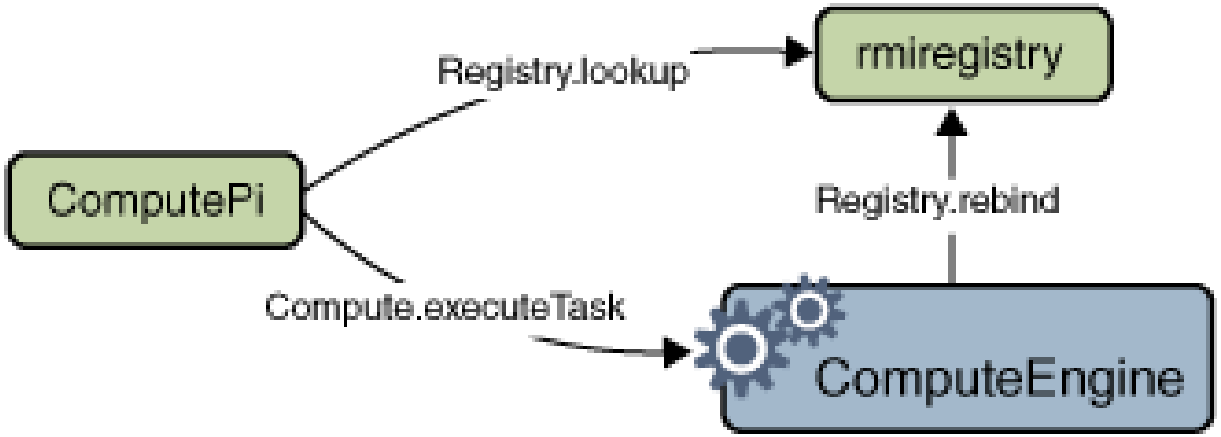
What's passes back and forth

- To both the client and server, the application appears to be local
- This is achieved by implementing interfaces which define the methods which can be invoked
  - The interfaces define the contract between the objects
- What is actually passed back and forth are
  - **primitive data types** (*passed by value*),
  - **remote references** (*objects which implement Remote*)
  - **copies of local objects** (*objects which implement* ***Serializable***).

RMI Registry

- Remote objects are listed in the **rmiregistry** of the server
- Client will query the rmiregistry for an object by its handle and will receive a reference to the remote object
- Client then invokes methods on the remote object

Example of RMI Diagram



Example Project

- It is best to first understand the objective
- Create a specification and plan of attack BEFORE writing any code
- RMI can become quite complex so this approach is essential

Project Description

- Create a distributed time server
- Supply a client which invokes methods on a remote object located on the server
- The server will compute and return a response which will be displayed by the client

Project Spec

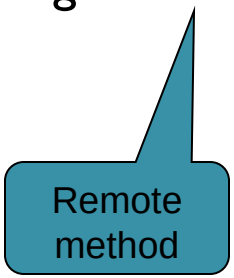
- The client will provide simple user interaction and display time of server machine
- The client is responsible for displaying the content

What's needed

- The components that we will need to create are:
  - **Client and Server** interfaces
  - **RMIServer Application**
  - **RMIClient Application**
  - **Stub** (automatically generated by **rmic**)

Server Interface

```
public interface RMIServerIntf extends Remote{
    public String GetTime() throws RemoteException;
}
```



RMIServer.java

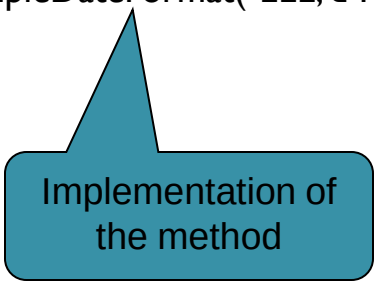
```
public RMIServerImpl() throws RemoteException {
    try {
        System.setSecurityManager( new RMISecurityManager() );
        UnicastRemoteObject.exportObject(this);
        Registry registry = LocateRegistry.createRegistry(8080);
        registry.bind("RMIServer", this);
        System.out.println("Server ready, running");
    } catch (Exception ex) {
        System.out.println("Server NOT ready");
        System.out.println(ex);
    }
}
```

Export Object

Create Registry

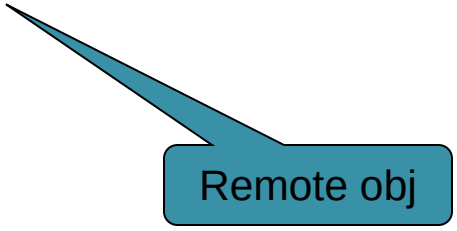
Bind it

```
    /* Define Remotely allowed calls */
    public String GetTime() throws RemoteException {
            System.out.println("Client ["+(++counter)+"] made call");
            return "Time at server .. "+
                    new SimpleDateFormat("EEE, d MMM yyyy HH:mm:ss").
            format(new Date());
    }
}
```



Discussion

- The **RMIServer** implements our interface which means it extends Remote
- **RMIServer** also opens RMI Registry by call:
  - `LocateRegistry.createRegistry(8080);`
  - You don't need to call `rmiregistry`
    - `Some tutorials show command line option to start registry, it is better practice to do it in the code`
- **Binding** is done with call:
  - registry.bind("RMIServer", this);



RMIClient.java

```
public class RMIClient {
    RMIServerIntf remoteObj;

    public RMIClient() {
        try {
            System.setSecurityManager( new RMISecurityManager() );
            remoteObj =(RMIServerIntf) Naming.lookup("//localhost:8080/RMIServer");
            System.out.println("Calling GetTime()");
            System.out.println(remoteObj.GetTime());
        } catch (Exception ex) {
            System.out.println(ex);
        }
    }
}
```

Reference to Remote Object

//URL:PORT/ ObjectName

You get errors here if
- codebase is not well-defined (including spaces in path).. Stub file will not be found, therefore client application fails.
- Host name cannot be resolved
- Remote object name does not match

Path problem…

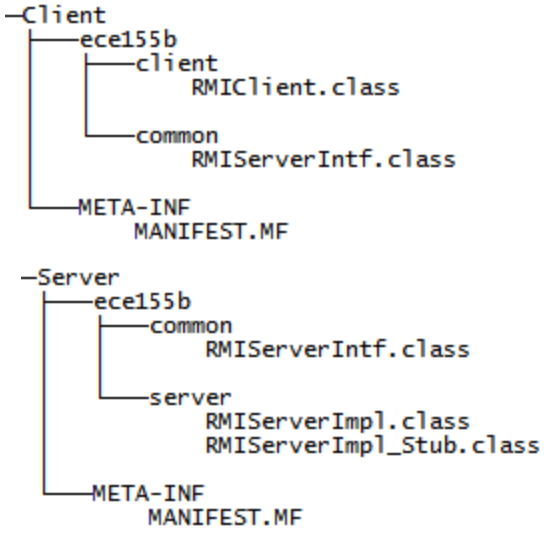
- You get errors here if codebase includes spaces in path, you will see compiler complains:
  - Codebase is not well-defined(path problem)
  - Stub file will not be found, therefore client application fails
  - Host name cannot be resolved
  - Remote object name does not match.
  - For Example:
    - remoteObj =(RMIServerIntf)Naming.lookup("//localhost:8080/RMIServer"); //works
    - remoteObj =(RMIServerIntf)Naming.lookup("//localhost:8080/RMI Server"); //error

What do we have?

- Compile:
  - Ant compile
- Create Stub
  - *This should be called for classes implementing Remote class*
  - Ant rmic
- Create jar files to run
  - Ant jar
- Run patient, doctor and pharmacy
  - Ant patient
  - Ant doctor
  - Ant pharmacy

Important

- **2 ways to organize Stub files**
  - **Include in client jar file (not likely)**
    - **Stub** file needs to be packed with client jar file
  - **Set the server code base**
    - Client loads stub file from codebase of the server
- **Interface** class for RMI server needs to be packed with both Server and Client
- This is to say:
  - Client.jar content →
  - Server.jar content



Project3

- Before you start project3, please do the addendum part of project2 (it's required!)
- So you should first create pharmacyGUI (Believe me, this is the last GUI for the class ^^)
- Then changed the project3 template so instead of writing prescriptions on the command prompt, save these info into pharmacy.xml

  (That easy?! Yeah believe me ☺)
- Please refer to sample pharmacy.xml on web.
- Once you are done, try the extra point ;)