RMI (Remote Method Invocation)

Eunmi Choi
Kookmin University

RMI Introduction

- **RMI**

  외부에 있는 메소드를 불러주는것. (네트웤을 통해). library를 통해 제공.

  - “**Remote Method Invocation**”
  - communicating the object across the network.
  - allows an object running in one Java virtual machine (Client) to invoke methods on an object running in another Java virtual machine (Server).

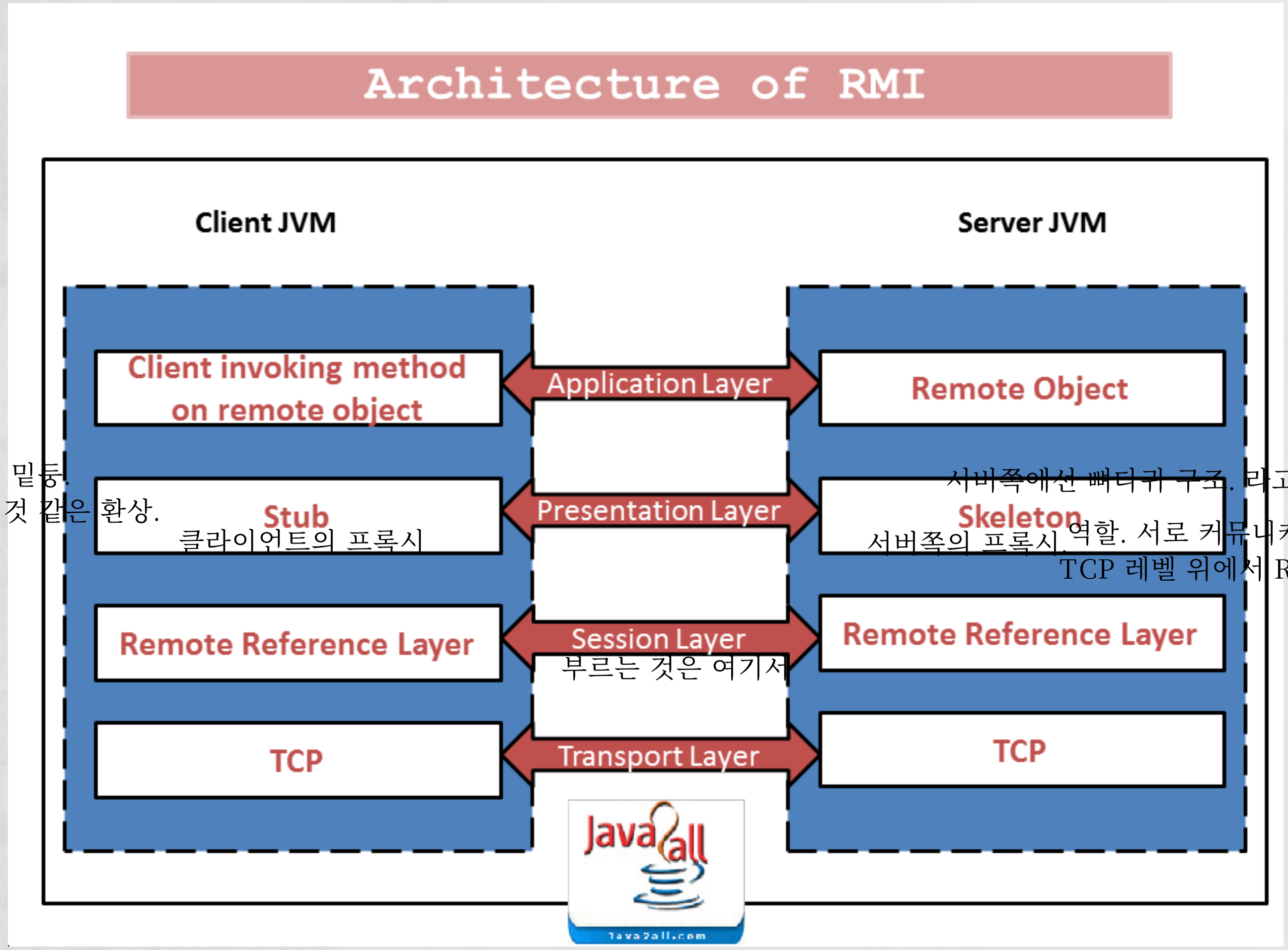
    읽으심.
  - also called **RMI Distributed** Application.

    분산적인어플리케이션을 사용할 수 있음.
  - remote communication between programs written in the JAVA.

RMI System Architecture View

- The complete RMI system has a **FOUR** 네개 layer,
  - (1) Application Layer
  - (2) Proxy Layer
  - (3) Remote Reference Layer
  - (4) Transport Layer

RMI System Architecture View



몽땅연필. 나무를 자르면 밑등!
나한테 오브젝트가 있는 것 같은 환상.
하지만 서버쪽에 있음.

클라이언트의 프록시

서버쪽에선 뼈다귀 구조. 라고 함.

서버쪽의 프록시 역할. 서로 커뮤니케이션을 홀딩해줌.
TCP 레벨 위에서 RMI가 돌아감.

부르는 것은 여기서

RMI System Architecture View

컨텐트가 들어가 있는 쪽

(1) Application Layer:

- Contains actual logic (implementation) of the client and server applications.
- Generally at the server side class contain implementation logic and also apply the reference to the appropriate object as per the requirement of the logic in application.

(2) Proxy Layer:

스터브 : 클라이언트 쪽의 프록시. 외부에 요청을 해줌.
스켈레톤 : 리얼한 오브젝을 해줄 수 있음.

- also called the **"Stub/Skeleton layer"**.
- A Stub class is a client side proxy handles the remote objects which are getting from the reference.
- A Skeleton class is a server side proxy that set the reference to the objects which are communicates with the Stub.

RMI System Architecture View

레퍼런스 레이어.

(3) Remote Reference Layer (RRL):

- manage the references made by the client to the remote object on the server so it is available on both JVM (Client and Server).
- The Client side RRL 오브젝이 있으면 바이트 단위로 시리얼라이제이션 하는 마셜링을 함
  - receives the request for methods from the Stub that is transferred into byte stream process called serialization (Marshaling) and then these data are send to the Server side RRL.
- The Server side RRL 바이트 레벨을 받아 오브젝화 시켜(디시리얼라이즈 해서 언마셜링) - 오브젝 핸들링 서버측에 요청..
  - doing reverse process and convert the binary data into object.
  - This process called deserialization or unmarshaling and then sent to the Skeleton class.

(4) Transport Layer: TCP/IP라고 보면 됨. 네트워크 커넥션이 됨.

- also called the "**Connection layer**".
- managing the existing connection and also setting up new connections.
- a link between the RRL on the Client side and the RRL on the Server side.

RMI Components

- The RMI application contains the THREE components

읽으심.

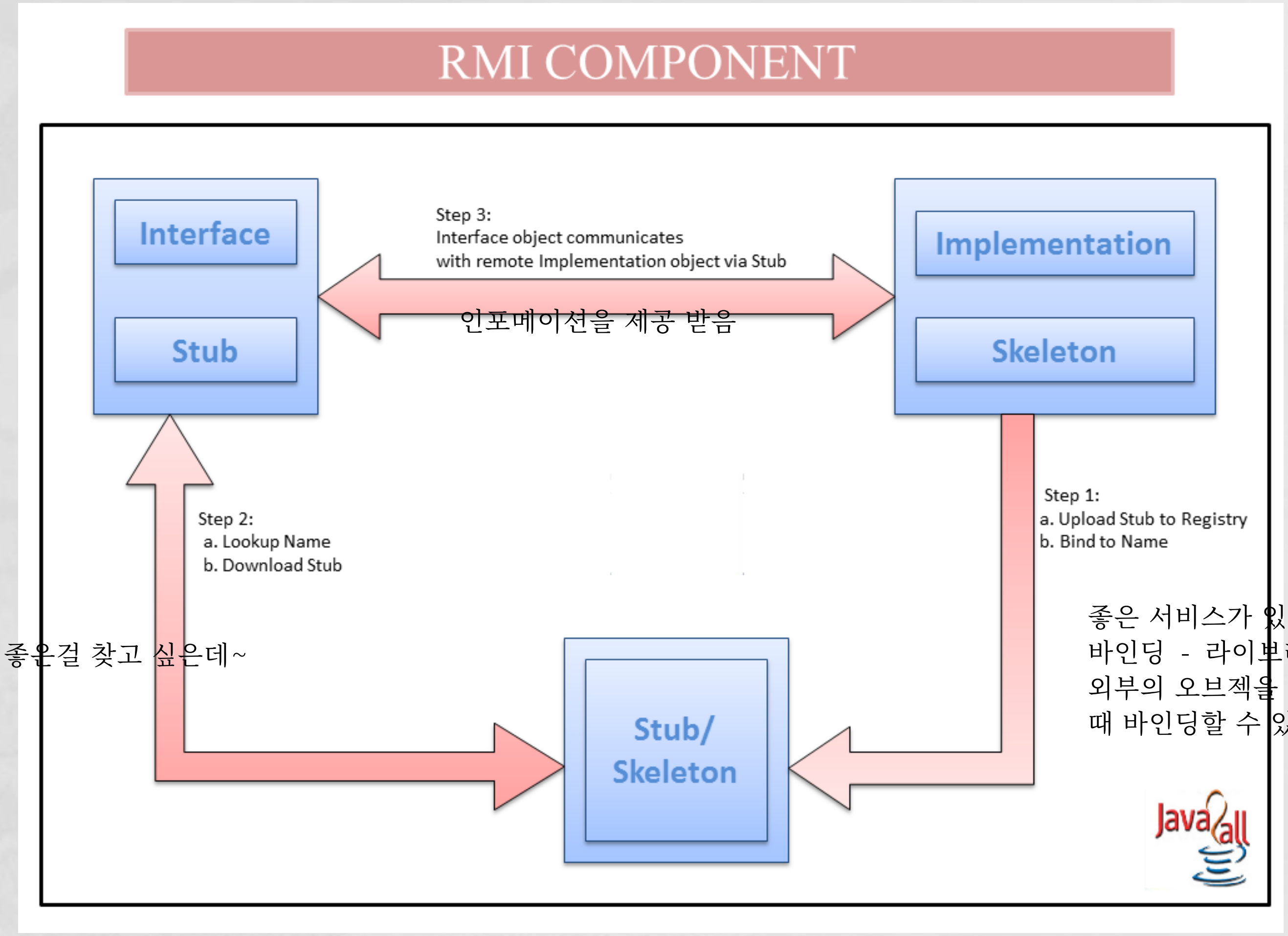
(1) RMI Server

(2) RMI Client

(3) RMI Registry

등록해주는 자
서버와 클라이언트의 외부에 있음.

RMI Components



인포메이션을 제공 받음

소팅알고리즘을 좋은걸 찾고 싶은데~

좋은 서비스가 있어요 등록할게요.
바인딩 - 라이브러리 바인딩
외부의 오브젝을 다른 시스템에서 사용할 때 바인딩할 수 있는 여건을 해줌.

RMI Components

실제오브젝을 갖고 있음. 헤비한 워크를 할 수 있음. 좋은 오브젝이 있으면 레지스트리에 등록을 함.

(1) RMI Server:

- contains objects whose methods are to be called remotely.
- It creates remote objects and applies the reference to these objects in the Registry,
  - after that the Registry registers these objects who are going to be called by client remotely.

중간에 프로세싱을 하고 싶으면 외부에 대한 것을 요청할 수 있는 RRL 구조를 가짐.

(2) RMI Client:

- The RMI Client gets the reference of one or more remote objects from Registry with the help of object name.
- Once the client gets the reference of remote object, the methods in the remote object are invoked just like as the methods of a local object.

(3) RMI Registry: 서버쪽에서 등록을 할 수 있음. 클라이언트 쪽에서 레지스트리에 등록된 정보를 가져갈 수 있음.

- In the Server side the reference of the object (which is invoked remotely) is applied and after that this reference is set in the RMI registry.
- When the Client call the method on this object,
  - first get the object from this reference which is available at RMI Registry
  - then calls the methods as per the requirement of logic in RMI application.

RMI Registry

레지스트리는 네이밍으로 작업을 함.
그 서버에 대한 identify를 찾아서 매핑해서 보내줄 수 있음.

- The RMI Registry is a naming service.
  - RMI server programs use this service to bind the remote java object with the names.
  - Clients executing on local or remote machines retrieve the remote objects by their name registered with the RMI registry and then execute methods on the objects.
  - RMI creates a remote proxy for that object and sent it to clients.
  - An object proxy contains the reference to an object.

- Command

```
> start rmiregistry
```

이러케 시작할 수 있음.

포트 1099　　　　얘가 제공해주는 리모트 오브젝을 바인딩 해서 실행함.

- By default the port 1099 is used by RMI registry to look up the remote objects. After the RMI registry starts objects can bind to it.

레지스트리를 띄우고 - > 서버를 띄우고 - > 클라이언트를 띄움.

- The next step
  - bind the remote object with the RMI registry, execute the server program.
  - execute the client program.

RMI Program Code

- First program is for declare a method in an interface. 인터페이스 관련
- Second Program is for implementing this method and logic. 메소드랑 로직
- Third program is for server side. 서버쪽
- And last one is for client side. 클라이언트쪽

Build an Interface

- **Calculator.java**

```
import java.rmi.Remote;
import java.rmi.RemoteException;

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

Implementing the methods

- **CalculatorImpl.java**

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class CalculatorImpl extends UnicastRemoteObject implements
   Calculator
{
  protected CalculatorImpl() throws RemoteException
  {
    super();
  }
  public long add(long a, long b) throws RemoteException
  {
    return a+b;
  }
}
```

Server Side Program

서버쪽 코드

- **CalculatorServer.java**

```
import java.rmi.Naming;

public class CalculatorServer
{
   CalculatorServer()
   {
      try
      {
         Calculator c = new CalculatorImpl();
         Naming.rebind("rmi://127.0.0.1:1099/CalculatorService", c);
      }
      catch (Exception e)
      {
         e.printStackTrace();
      }
   }
   public static void main(String[] args)
   {
      new CalculatorServer();
   }
}
```

오브젝 하나 만들어서 바인딩~

Client Side Program

- **CalculatorClient.java**

```
import java.rmi.Naming;

public class CalculatorClient
{
   public static void main(String[] args)
   {
      try
      {
         Calculator c = (Calculator) Naming.lookup("//127.0.0.1:1099/CalculatorService");
         System.out.println("addition : "+c.add(10, 15));
      }
      catch (Exception e)
      {
         System.out.println(e);
      }
   }
}
```

How to Run the example

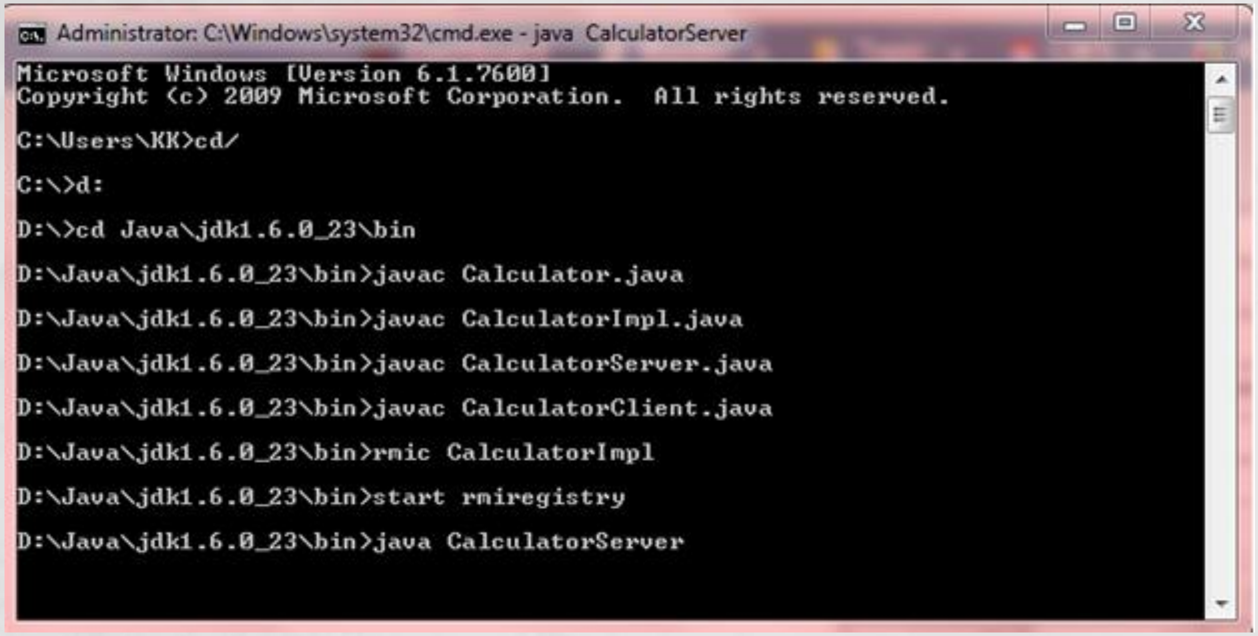
- javac Calculator.java
- javac CalculatorImpl.java
- javac CalculatorServer.java
- javac CalculatorClient.java
- rmic CalculatorImpl

근데 별로안중요해보임 ,,,

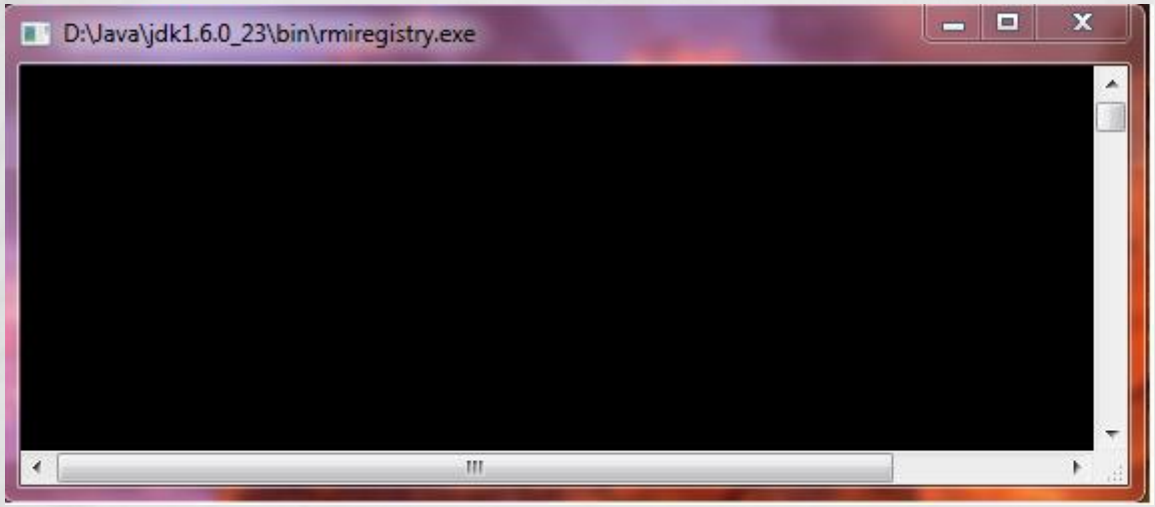
레지스트리는 정보만 주는 것
프록시가 있으면 필요 없음.
RMI 안에 이미 프록시가 있기 때문에.

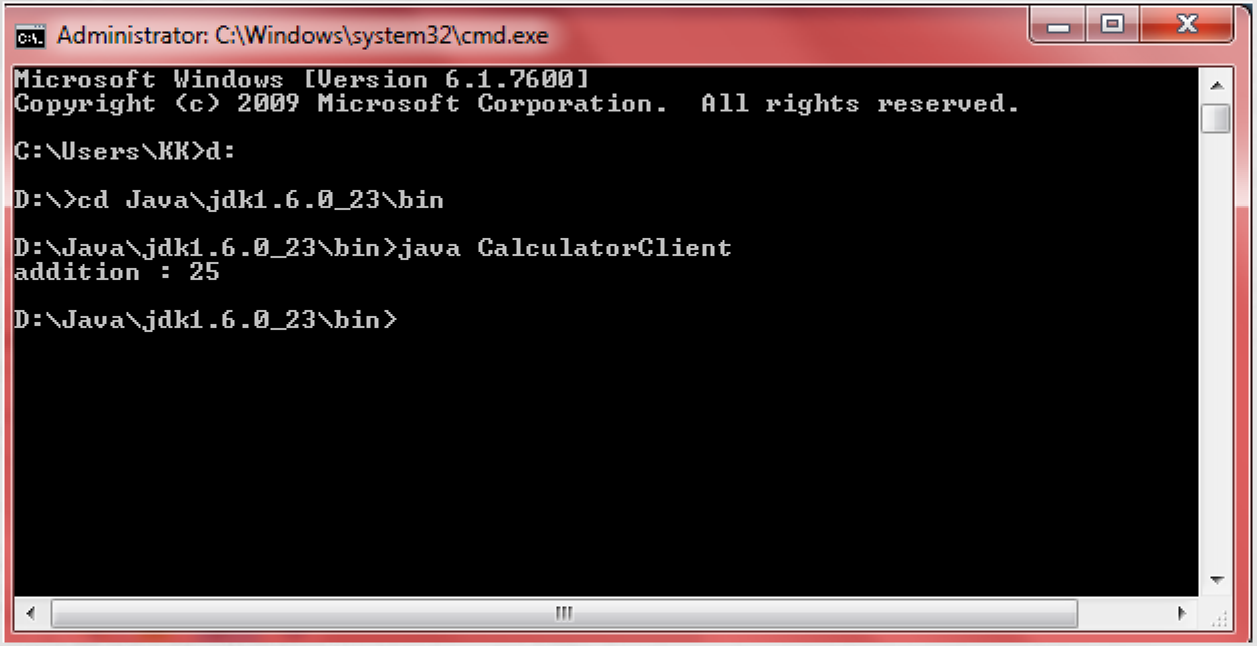
- start rmiregistry rmi를 띄움. 멍하니 있음.
- java CalculatorServer

open another cmd
- java CalculatorClient


Administrator: C:\Windows\system32\cmd.exe - java CalculatorServer

```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\KK>cd/

C:\>d:

D:\>cd Java\jdk1.6.0_23\bin

D:\Java\jdk1.6.0_23\bin>javac Calculator.java

D:\Java\jdk1.6.0_23\bin>javac CalculatorImpl.java

D:\Java\jdk1.6.0_23\bin>javac CalculatorServer.java

D:\Java\jdk1.6.0_23\bin>javac CalculatorClient.java

D:\Java\jdk1.6.0_23\bin>rmic CalculatorImpl

D:\Java\jdk1.6.0_23\bin>start rmiregistry

D:\Java\jdk1.6.0_23\bin>java CalculatorServer
```


D:\Java\jdk1.6.0_23\bin\rmiregistry.exe



```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\KK>d:

D:\>cd Java\jdk1.6.0_23\bin

D:\Java\jdk1.6.0_23\bin>java CalculatorClient
addition : 25

D:\Java\jdk1.6.0_23\bin>
```

More Java RMI Examples

- Interface File
  - The client uses the interface file.
  - At the Server and Client parts
  - Hello.java
- Interface Implementation file
  - At the Server part
  - HelloImpl.java
- Binding step
  - At the Server part
  - HelloBind.java
- Invocation from the client
  - At the Client part
  - Build an application
  - HelloTest.java

```
//  Hello.java

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Hello extends Remote {
    public String sayWelcome() throws
    RemoteException;
    public String sayHello() throws RemoteException;
}
```

```
// HelloImpl.java

import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class HelloImpl extends UnicastRemoteObject implements Hello {
    public HelloImpl() throws RemoteException {
        super();
    }                                         이런저런 하는 일들

    public String sayWelcome() {
        return "Welcome to the Obecjt Oriented Programming Application";
    }

    public String sayHello() {
        return "Hello World, Buddy!";
    }
}
```

```
// HelloBind.java

import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.RemoteException;

public class HelloBind {
    public static void main(String[] args) {
        try {
            HelloImpl hello = new HelloImpl();
            Naming.rebind("rmi://localhost:1099/hello", hello);        서버쪽으로 바인딩
            System.out.println("HelloImpl이 등록되었습니다.");
        } catch (RemoteException e) {
            e.printStackTrace();
        } catch (MalformedURLException e) {
            e.printStackTrace();
        }
    }
}
```

```
// HelloTest.java

import java.net.MalformedURLException;
import java.rmi.Naming;                    rmi로 Naming을 불렀던 것이구나
import java.rmi.NotBoundException;
import java.rmi.RemoteException;

public class HelloTest {
  public static void main(String[] args) {
    try {          리모트 헬로로 연결이 됨.
      Hello hello = (Hello) Naming.lookup("rmi://localhost:1099/hello");   헬로를 찾음.
      String msg = hello.sayWelcome();
      System.out.println("Call RMI method sayWelcome() : " + msg);
      msg = hello.sayHello();                                    계속해서 불러나갈 수 있음.
      System.out.println("Call RMI method sayHello() : " + msg);
    } catch (MalformedURLException e) {
      e.printStackTrace();
    } catch (RemoteException e) {
      e.printStackTrace();
    } catch (NotBoundException e) {
      e.printStackTrace();
    }
  }
}
```

Steps to activate the RMI example

- Apply registry
  ```
  > start rmiregistry
  ```
- At the server
  ```
  > javac HelloBind.java
  > java HelloBind
  ```
- At the client
  ```
  > javac HelloTest.java
  > java HelloTest
  ```

Reference

- http://www.java2all.com
- http://zkakira.tistory.com/entry/JAVA-RMI-%EC%98%88%EC%A0%9C
- http://blog.naver.com/PostView.nhn?blogId=eunicon&logNo=100043860463
- http://blog.naver.com/PostView.nhn?blogId=kkpa1002&logNo=20121297089&redirect=Dlog&widgetTypeCall=true