Information System Architecture

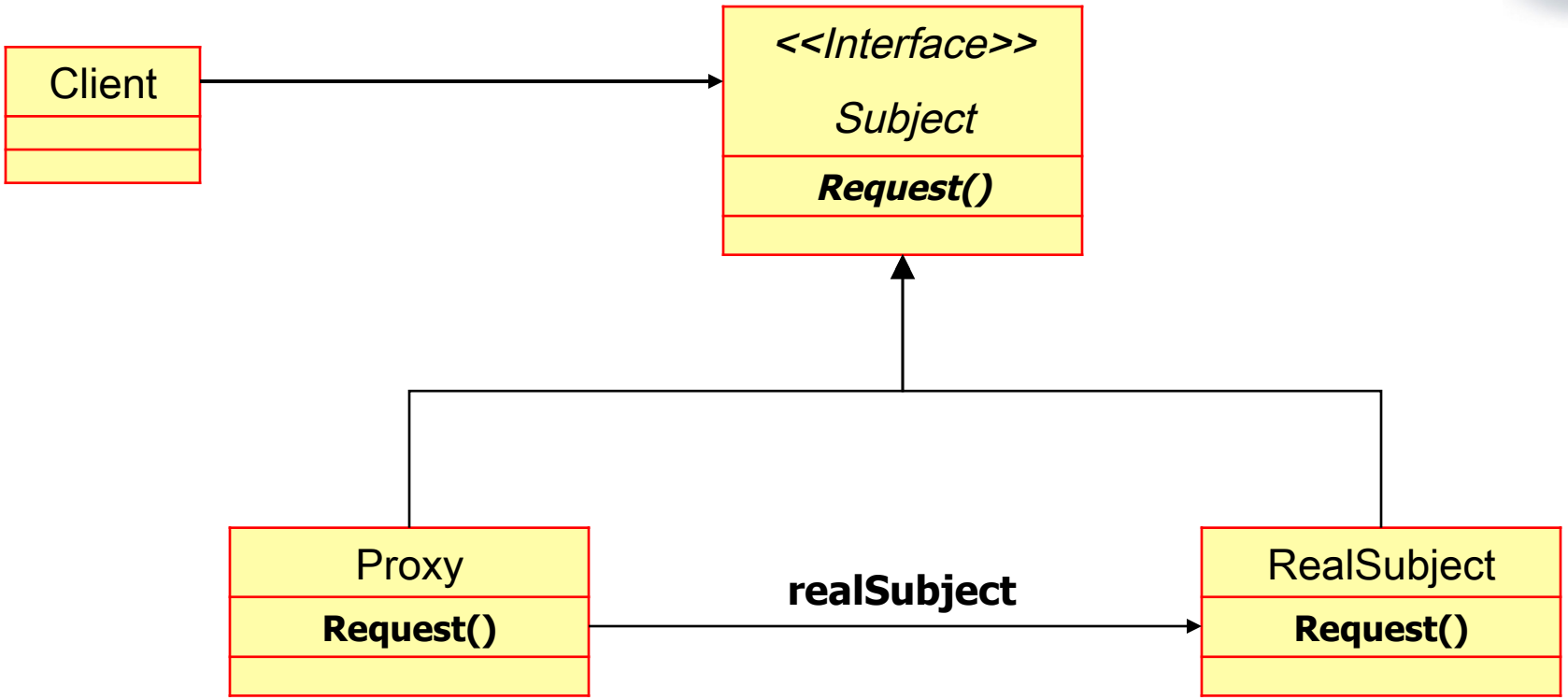
Java RMI

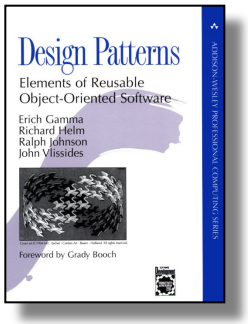
Exemple Add()

JRMP

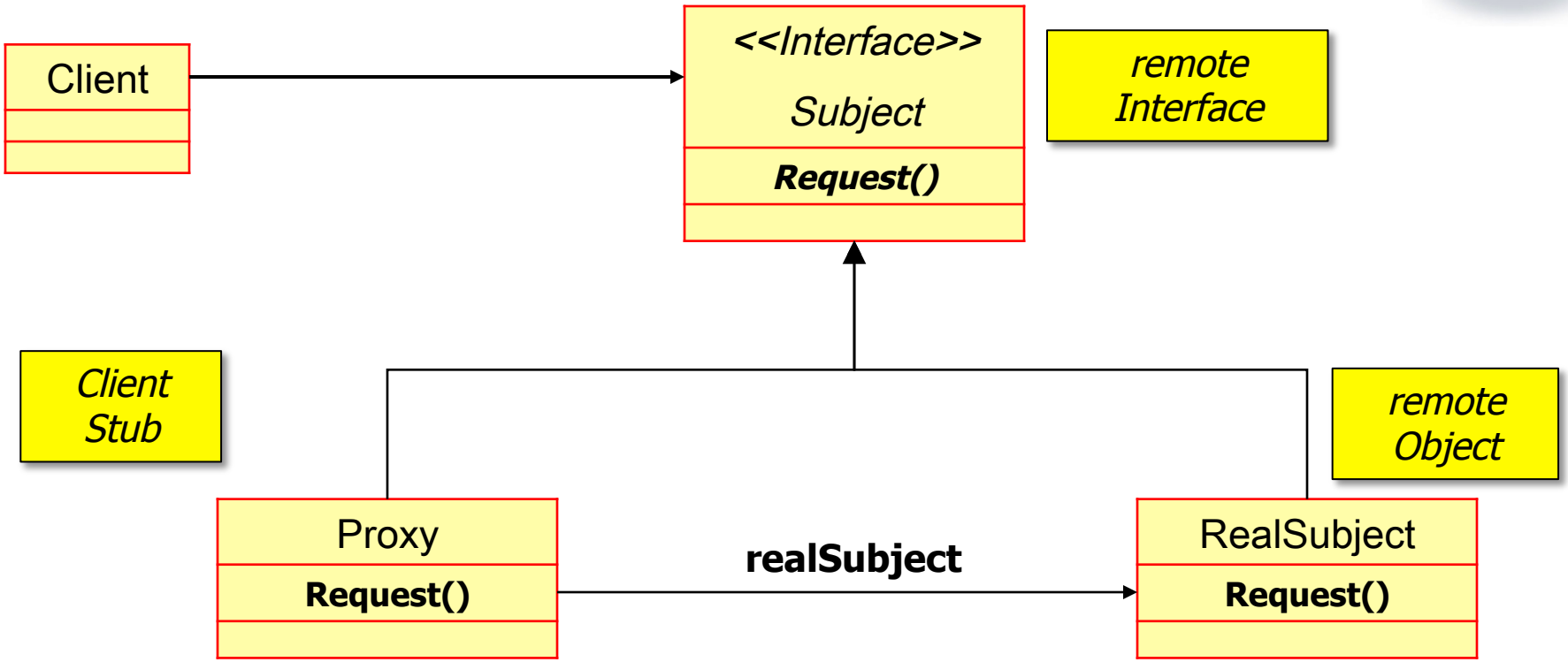
Java RMI Method Protocol

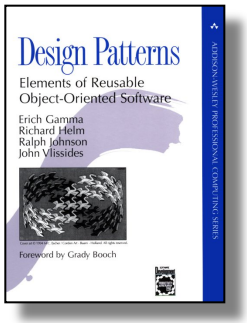
Proxy (UML)


Client
<<Interface>>
Subject
Request()
Proxy
Request()
realSubject
RealSubject
Request()

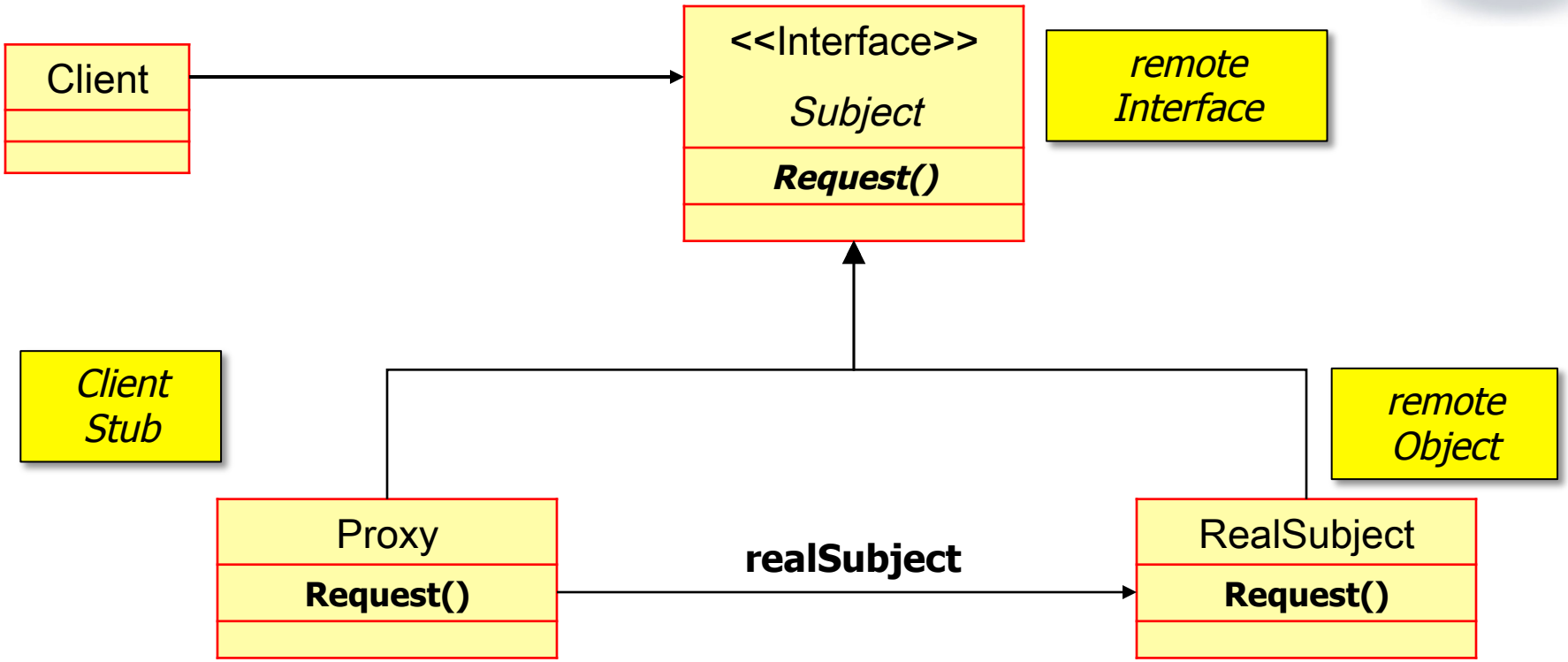

Design Patterns
Elements of Reusable
Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch

Proxy (UML)


Client
<<Interface>>
Subject
Request()
remote
Interface
Client
Stub
remote
Object
Proxy
Request()
realSubject
RealSubject
Request()


Design Patterns
Elements of Reusable
Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch

Proxy (UML)



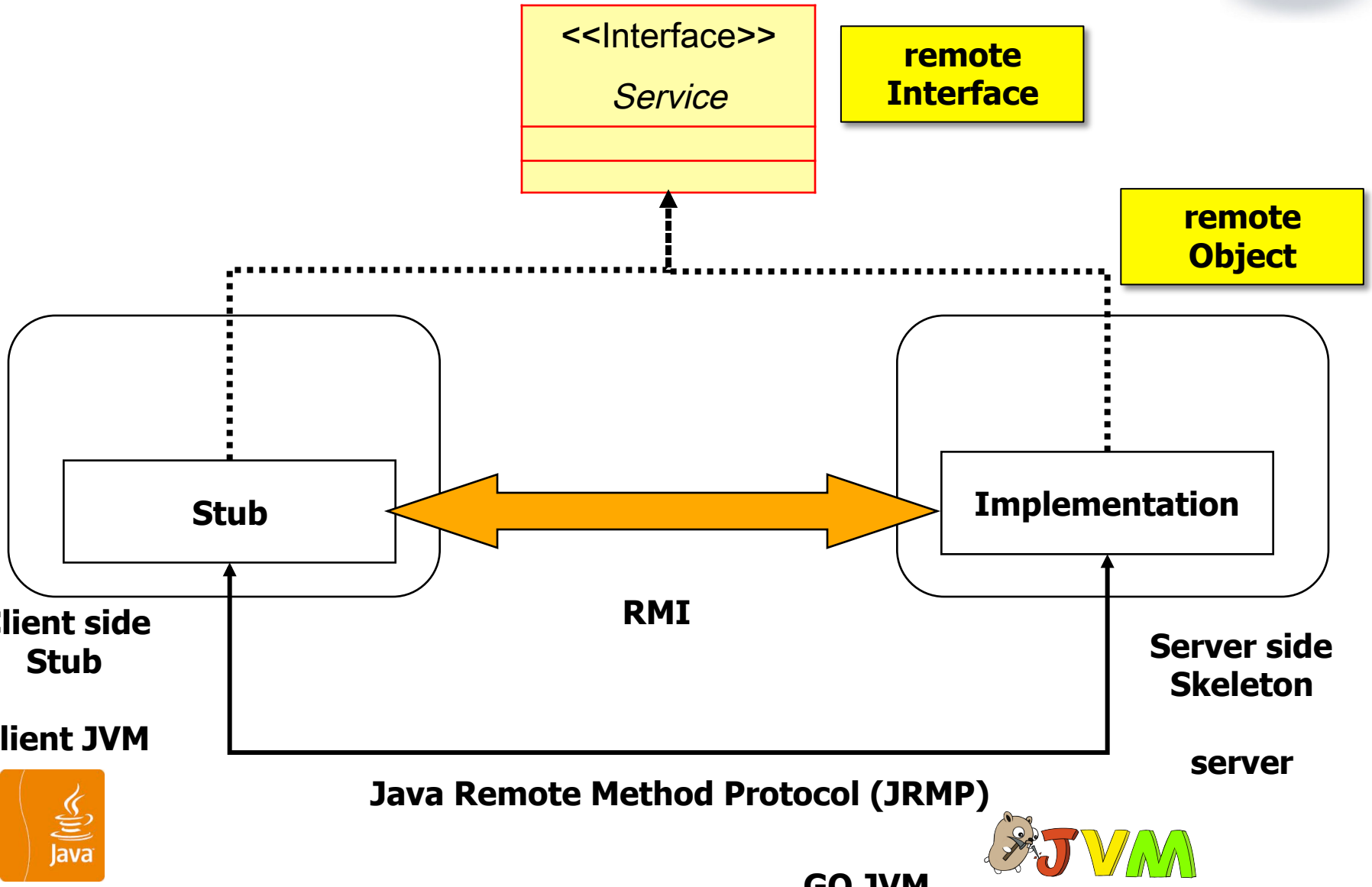
Client side Stub

Client JVM

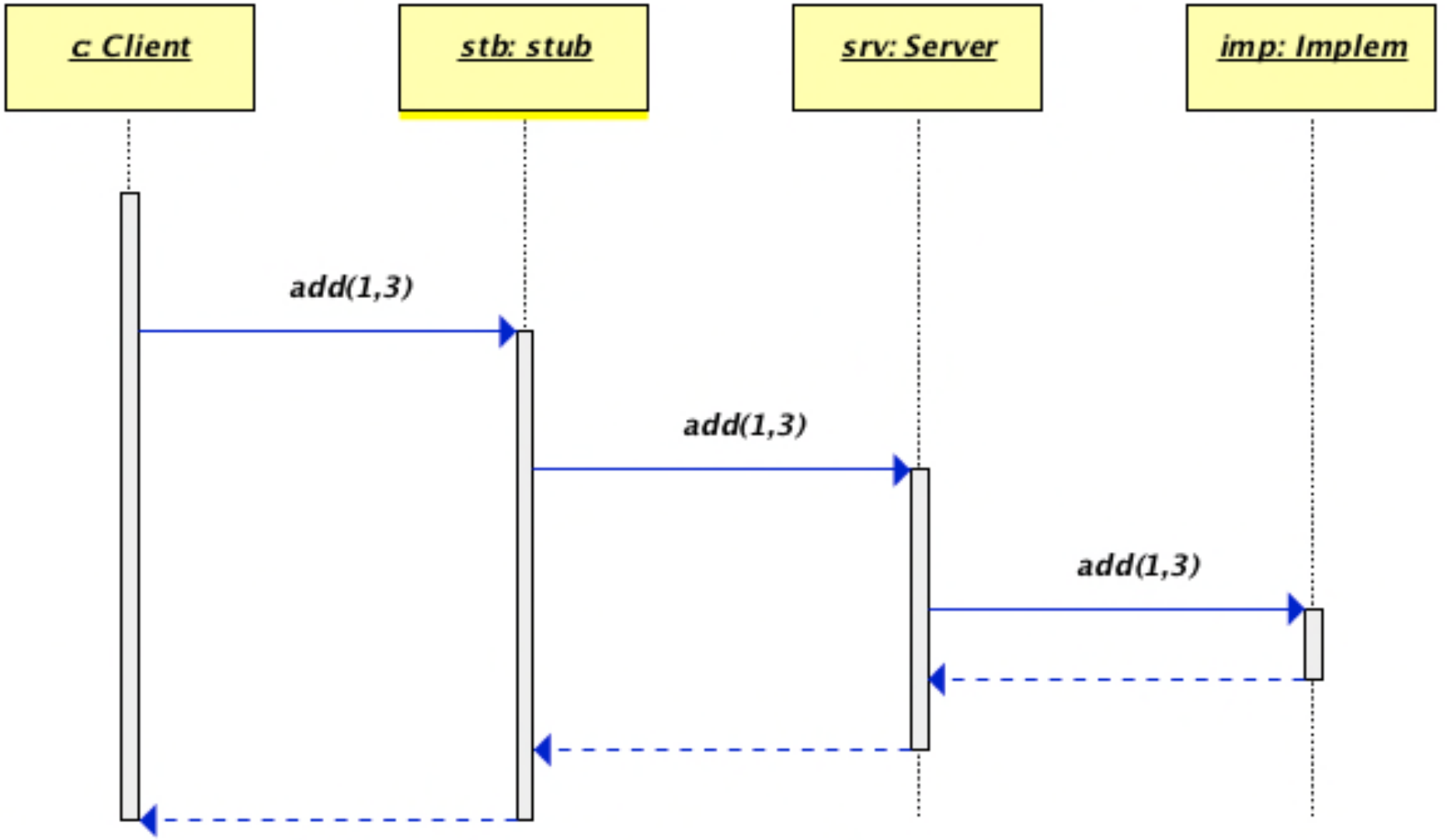
Server side Skeleton

Server JVM

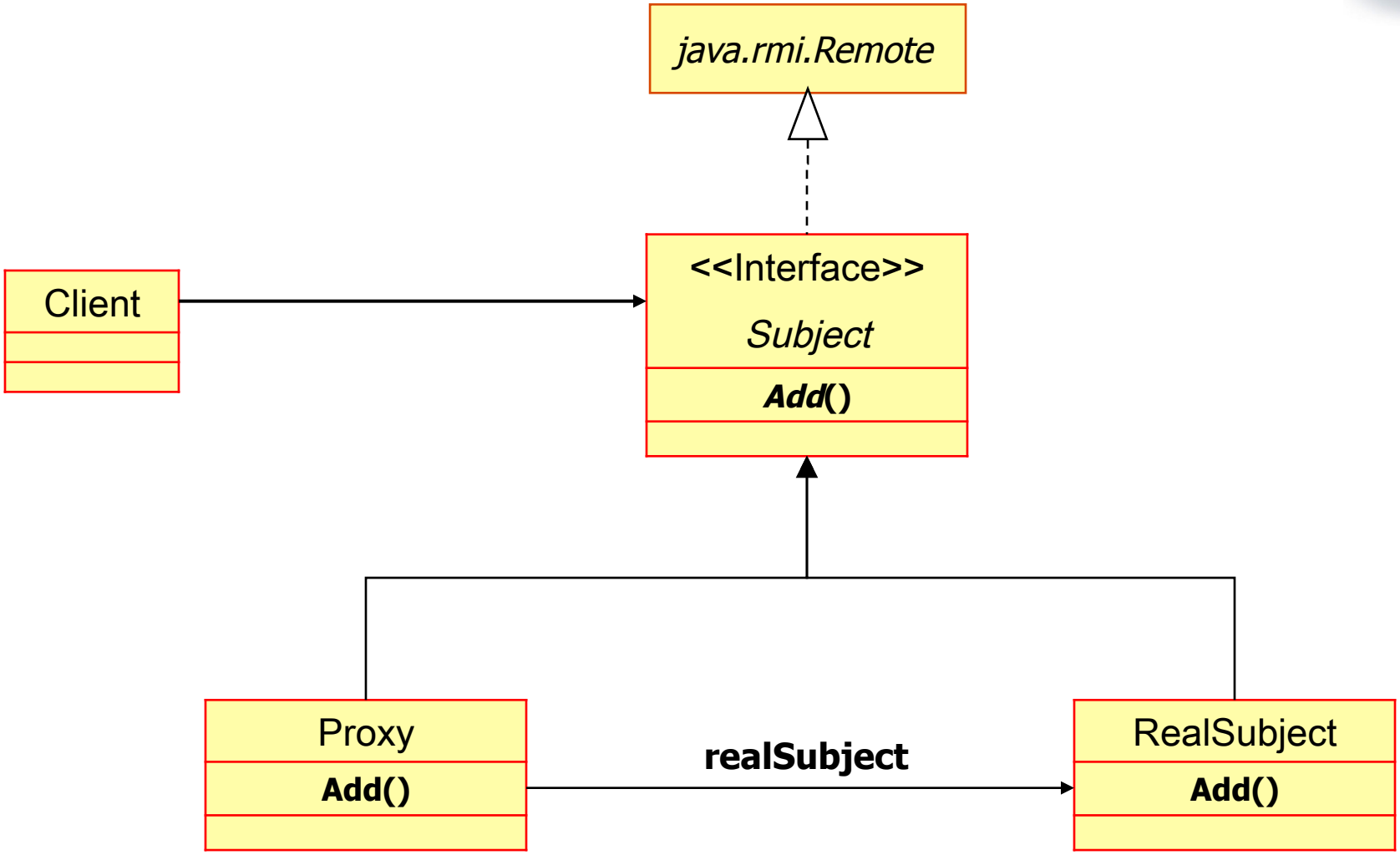
RMI proxy stub skeleton



RMI : Add server



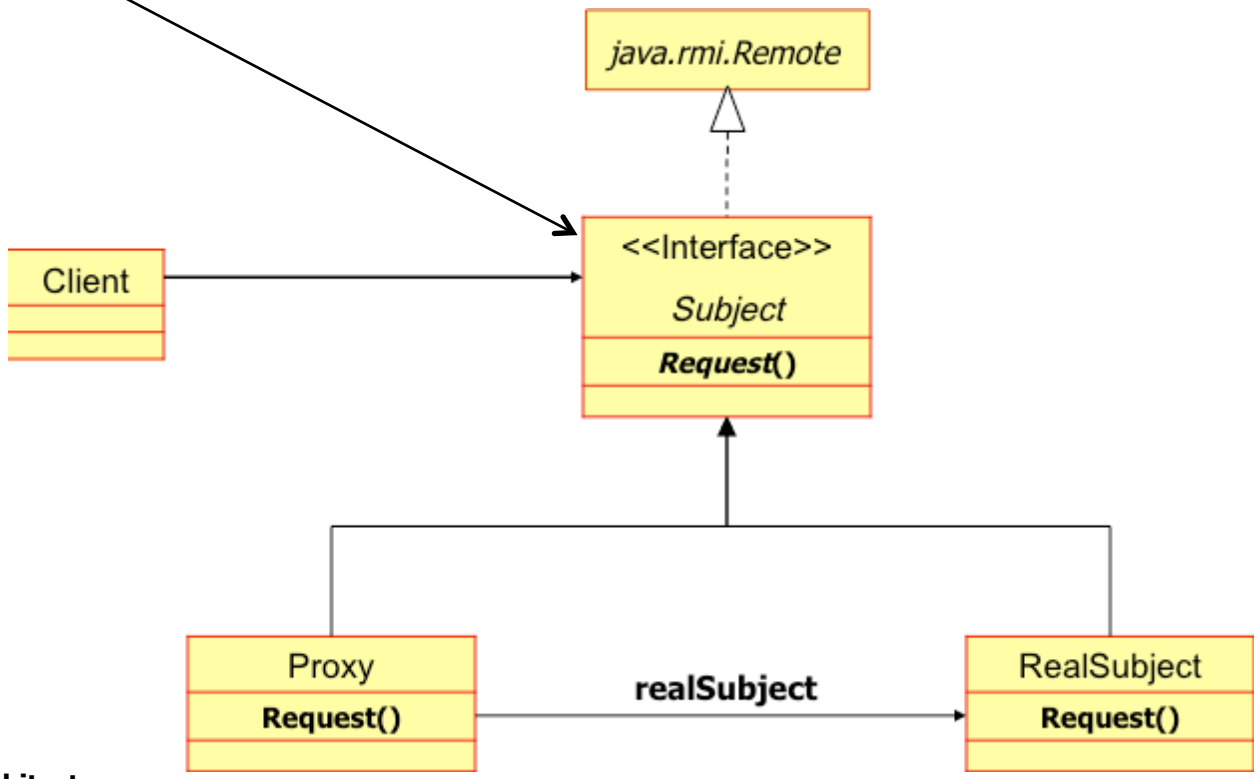
RMI


java.rmi.Remote
Client
<<Interface>>
Subject
Add()
Proxy
Add()
realSubject
RealSubject
Add()

Interface : Add

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

public interface AddInterface extends Remote {
    public Integer add(Integer nb1, Integer nb2) throws RemoteException;
}
```

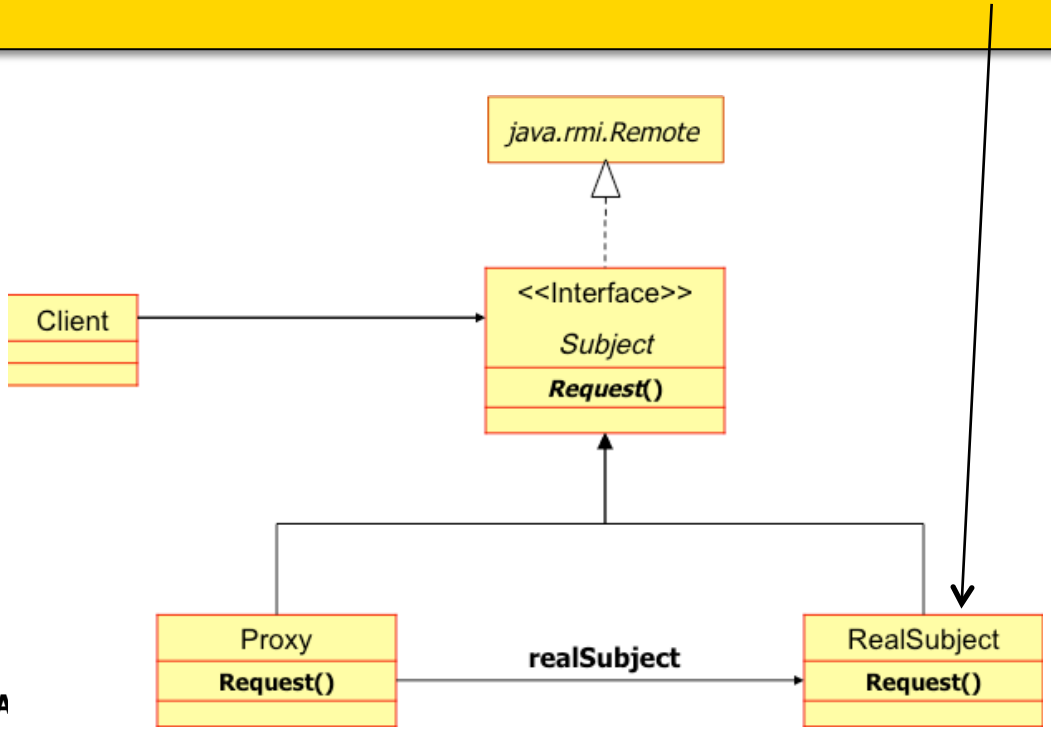


Implementation of the remote interface

```
import java.rmi.RemoteException;

public class AddImpl implements AddInterface {

    public Integer add(Integer nb1, Integer nb2)
                            throws RemoteException
         {
        return nb1 + nb2;
         }
}
```



The Server

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;

public class Server {
    public static void main(String[] argv) {
        try {

         AddInterface skeleton =
         (AddInterface) UnicastRemoteObject.exportObject(new AddImpl(), 0)

         Registry registry = LocateRegistry.createRegistry(0);

        registry.rebind("Add", skeleton);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Export a remote object

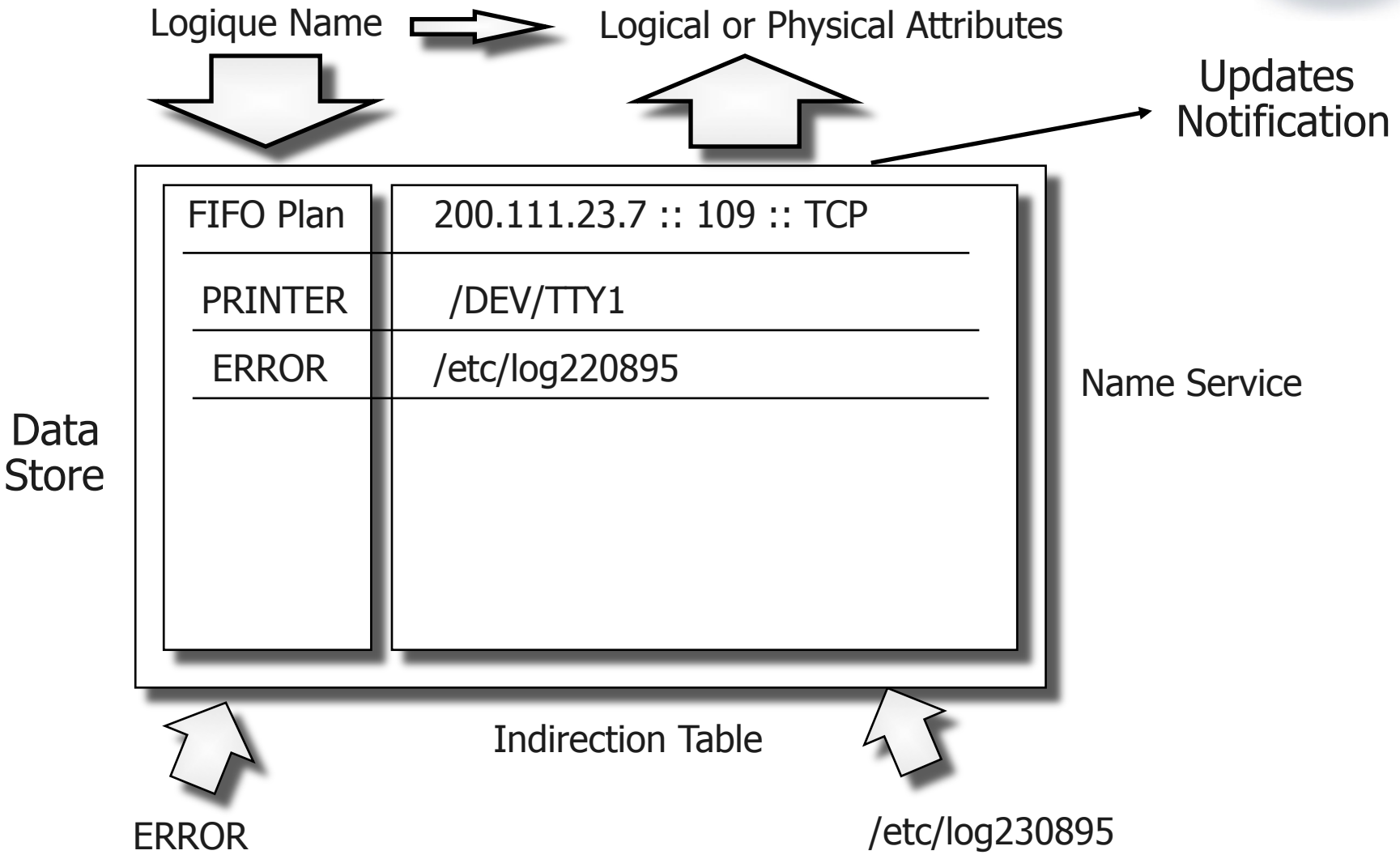
- Export a remote object :
  - to make it available to accept incoming calls from clients.
- The exportObject() method takes 2 parameters :
  - 1) Instance of the remote object,
  - 2) TCP port number.
- Port number can accept incoming calls for more than one remote objects.
- If TCP port number = 0, the default RMI port number 1099 is used.
- UnicastRemoteObject is used for exporting a remote object with JRMP and obtaining a stub that communicates to the remote object.

Java RMI registry

- To be able to invoke a method on a remote object, caller must first obtain a stub for the remote object.

- For bootstrapping, Java RMI provides a registry API for:
  - Server applications to bind a name to a remote object's stub.
  - clients to look up remote objects by name in order to obtain their stubs.

Registry model



Java RMI registry

- A Java RMI registry is a simplified name service that allows clients to get a reference (a stub) to a remote object.
- The Java RMI registry is a remote object that maps names to remote objects.
- LocateRegistry is a bootstrap to the registry.
- The method bind() or rebind() binds a unique name to the reference of the remote object
- the remote object name "Add" is bound to the stub that is returned from the exportObject() method.

The client

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

public class Client {
    public static void main(String[] argv) {
        try {

            Registry registry =
                LocateRegistry.getRegistry(0);

            AddInterface stub = (AddInterface)
                registry.lookup("Add");

            System.out.println(stub.add(1, 2));

        } catch (Exception e) { e.printStackTrace(); }
    }
}
```

Client registry

- The client program obtains a stub for the registry on the server's host.
- Looks up the remote object's stub by name in the registry.
- Then invokes the remote method on the remote object using the stub.