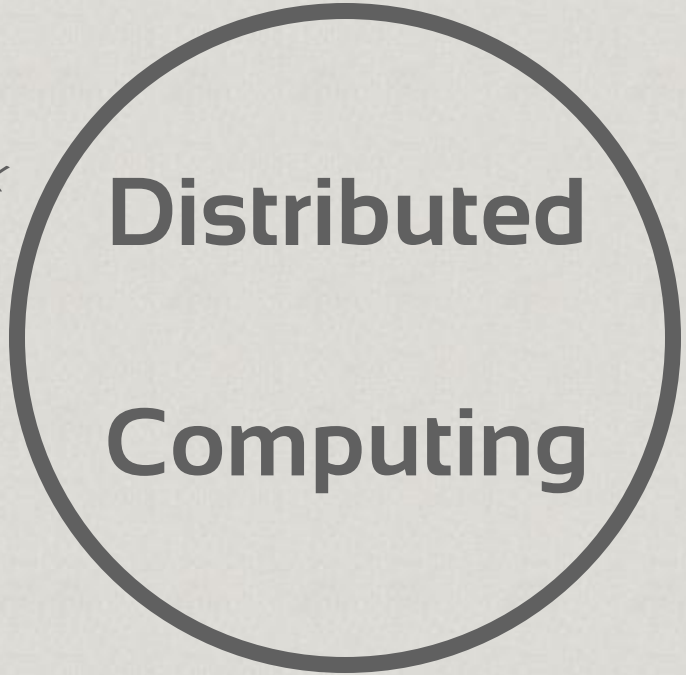
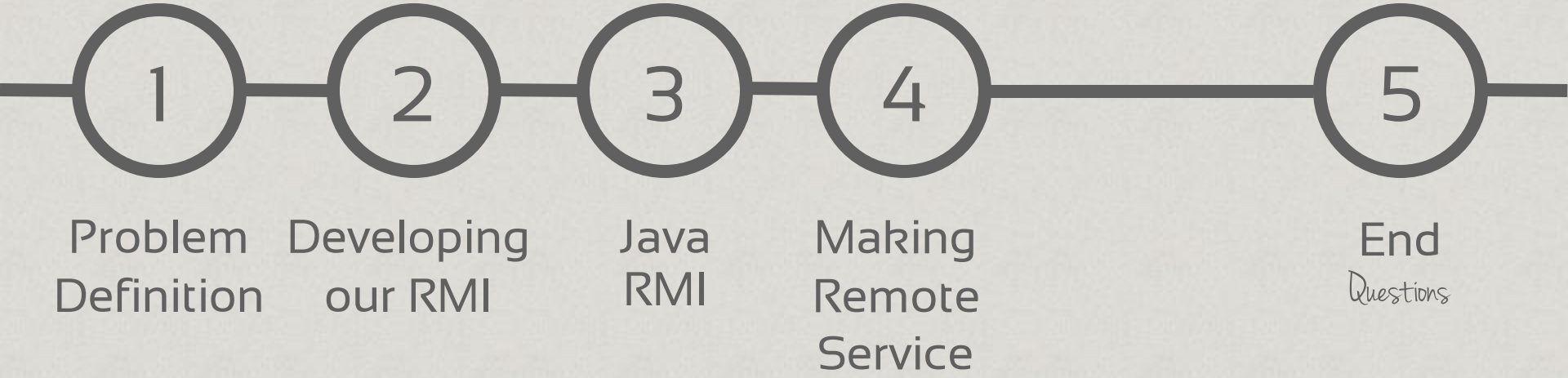
From Head first JAVA book



Java RMI

Mahmoud Badry, mma18@fayoum.edu.eg

AGENDA



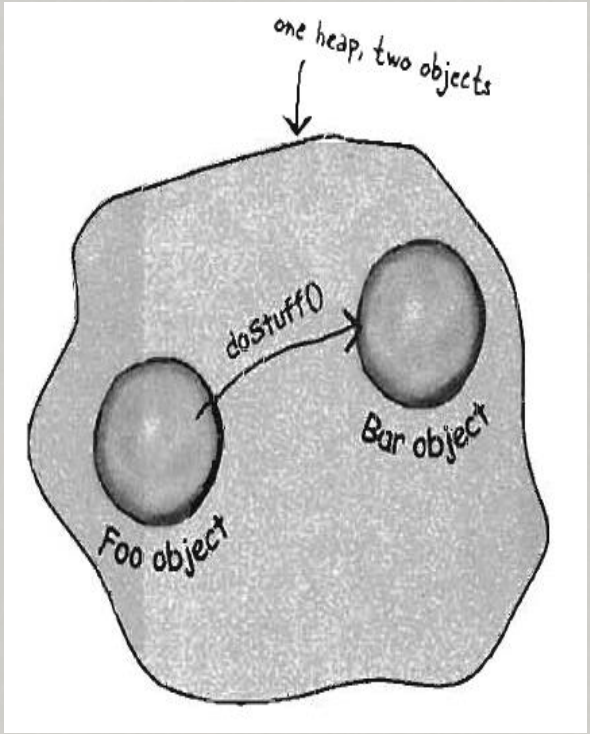
Mahmoud Badry, mma18@fayoum.edu.eg

How many heaps?

Mahmoud Badry, mma18@fayoum.edu.eg

Method calls are always between two objects in the same heap

```
class Foo {

    void go() {

        Bar b = new Bar();

        b.doStuff();

    }

    public static void main (String[] args) {

       Foo f = new Foo();

       f.go();

    }

}
```

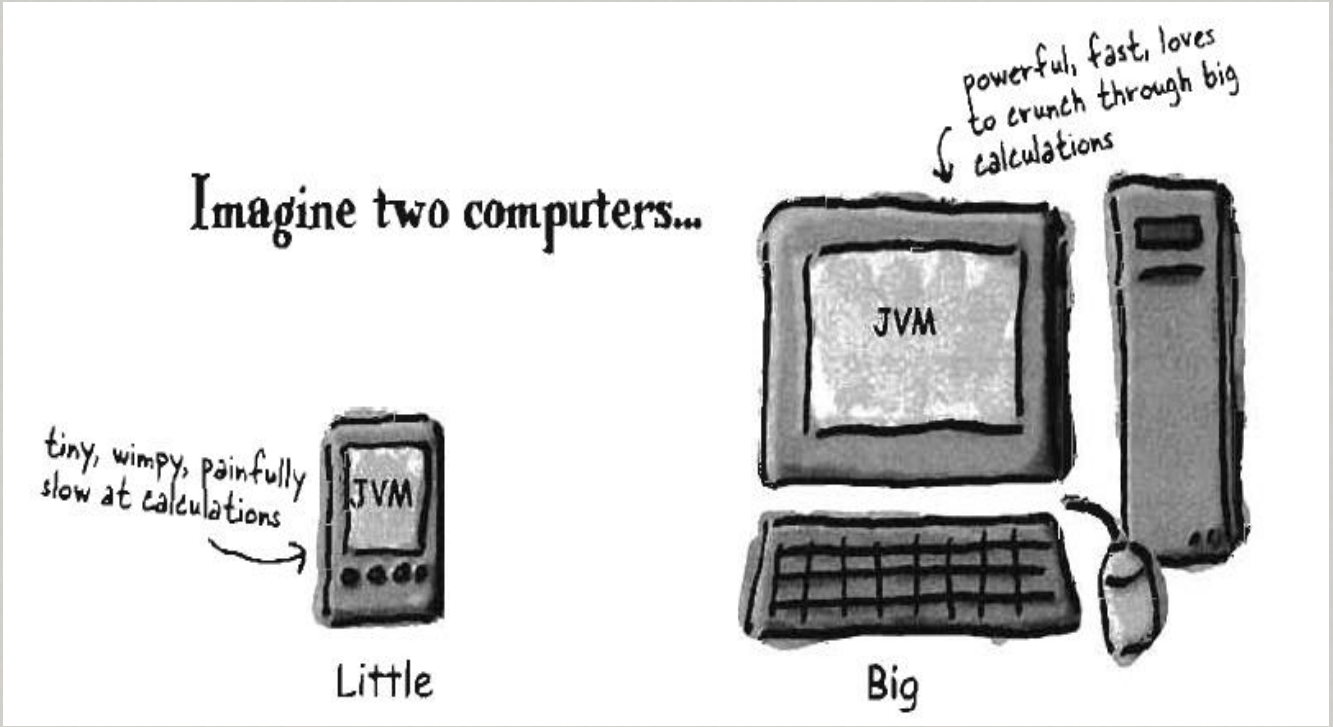


So far every method we have invoked runs on the same JVM.

The JVM Can know about references on only its own heap! You can't, for example, have a JVM running on one machine knowing about the heap space of a JVM running on a different machine. Even if there was two JVM in the same machine.

Mahmoud Badry, mma18@fayoum.edu.eg

What if you want to invoke a method On an object running on another machine



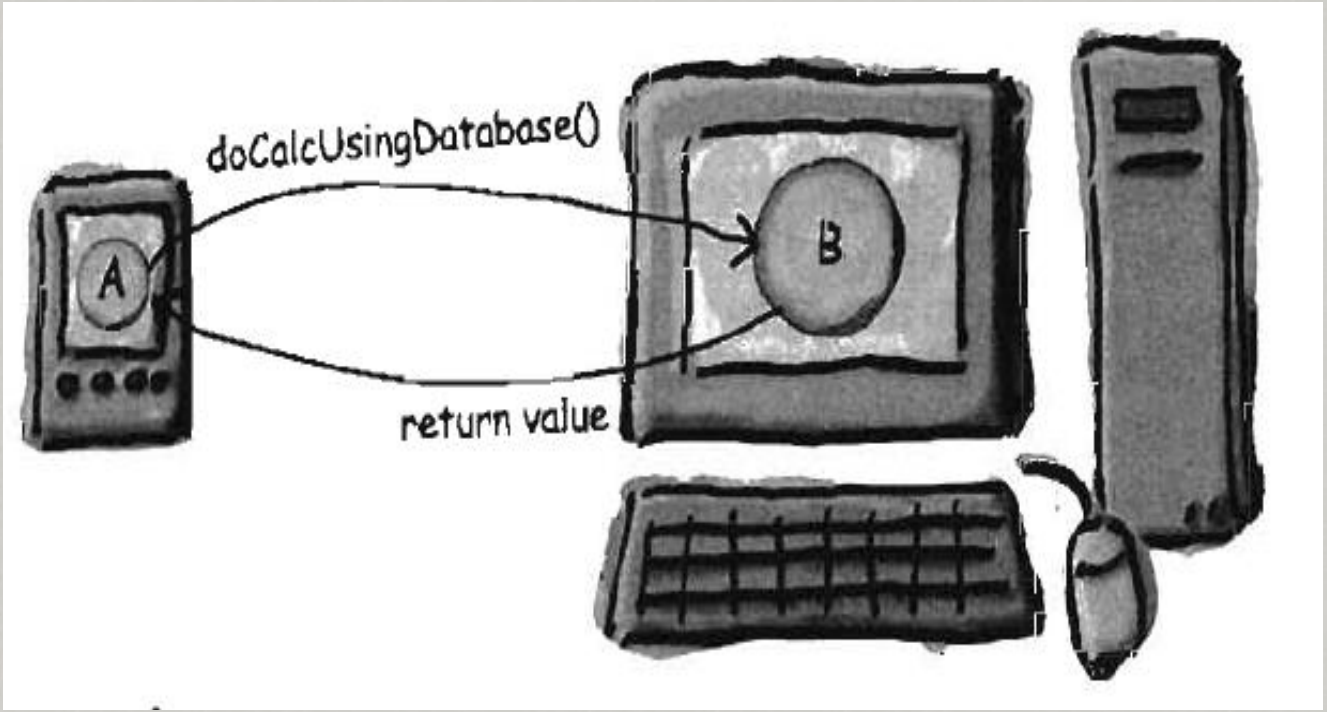
Little wants simply to call a method and get the results

We can user sockets but it will be so hard.

You could send data to a ServerSocket the server could parse it, figure out what you meant, do the work, and send back the result on another stream. What a pain, though.

Mahmoud Badry, mma18@fayoum.edu.eg

Object A, running on little, wants to call a method on object B, running on Big



The question is, how do we get an object on one machine(which means a different heap/VM) to call a method on another machine?

Lets begin to think on doing something like that without RMI

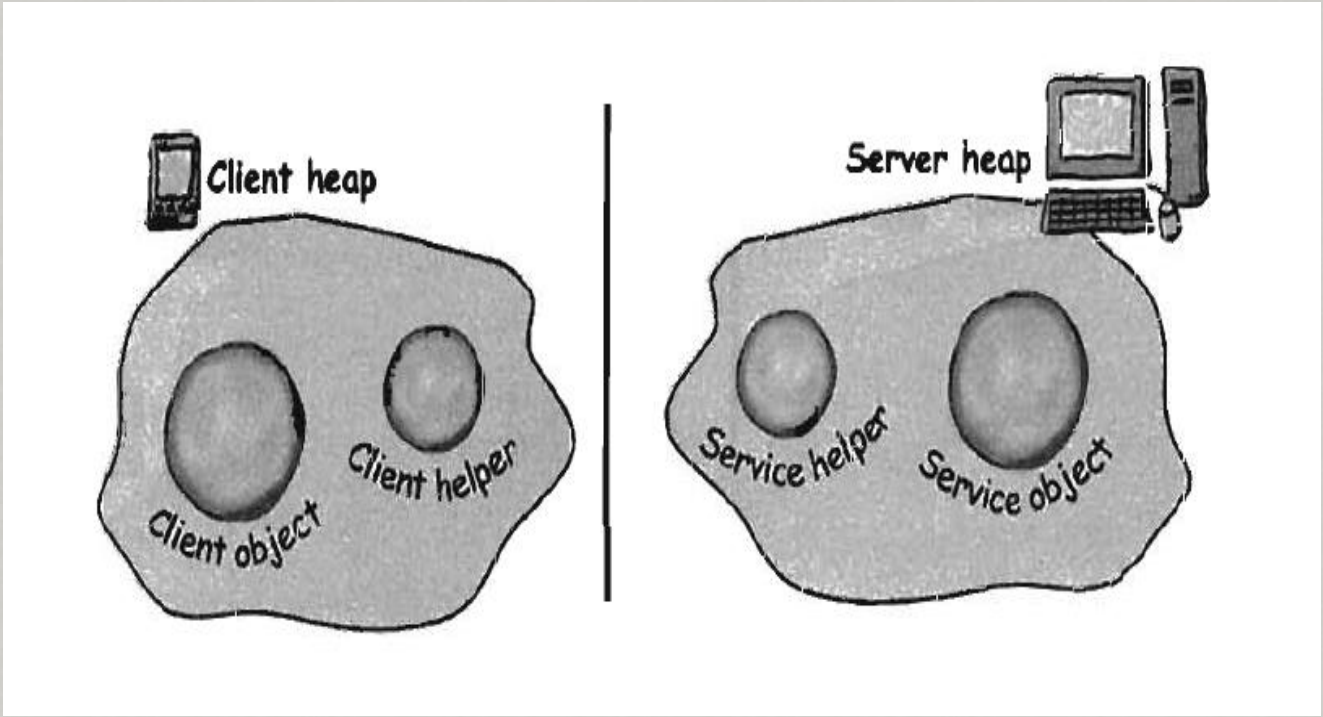
Mahmoud Badry, mma18@fayoum.edu.eg

Developing our RMI

Lets create our own Remote method invoking

Mahmoud Badry, mma18@fayoum.edu.eg

Programs architecture



We will create two applications on different machines, one for the server and one for the client.

Suppose that client object wants to access a method on service object.

We'll create two other helper classes that will handle all the low-level networking and I/O details so your client and service can pretend like they're in the same heap.

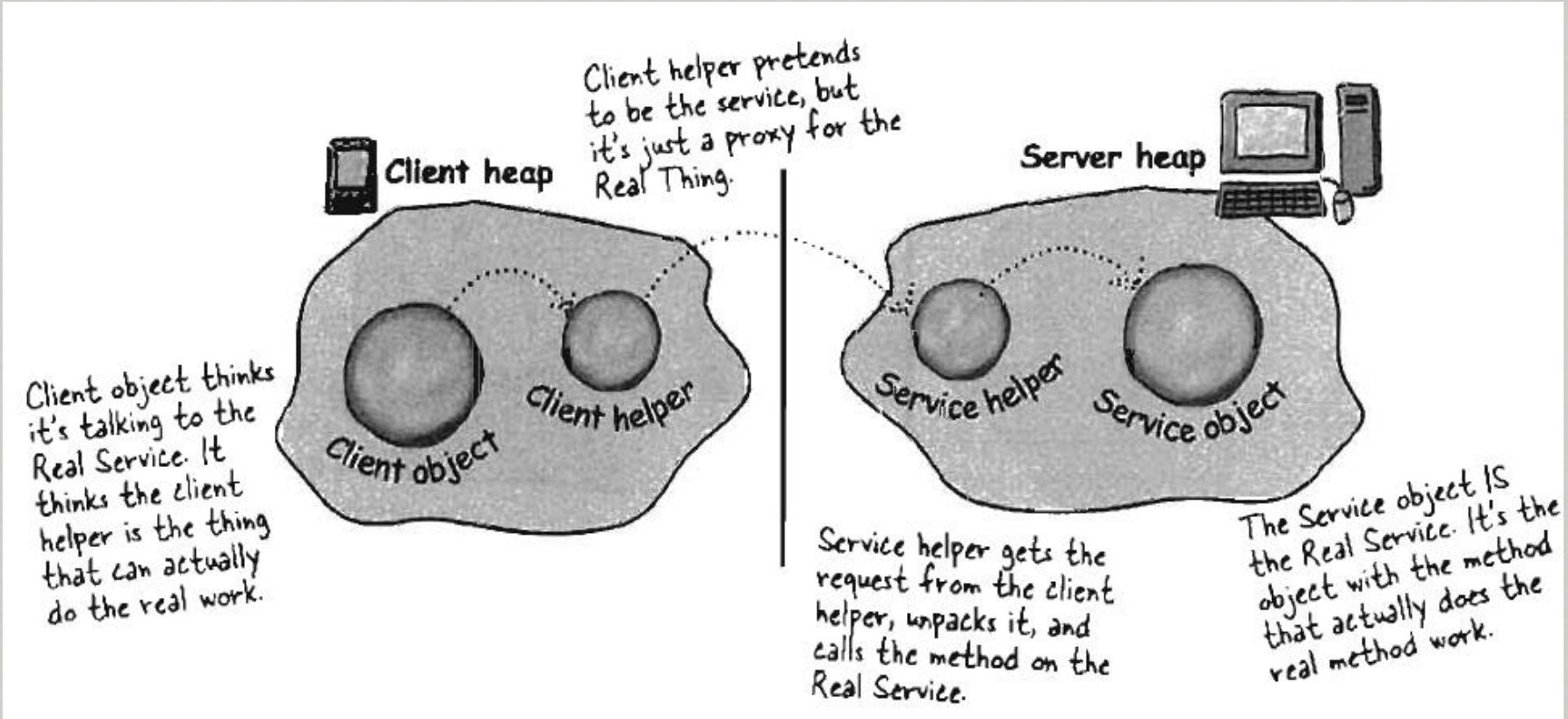
Mahmoud Badry, mma18@fayoum.edu.eg

The role of helpers

- The 'helpers' are the objects that actually do the communicating.
- The client object thinks it's calling a method on the remote service, because the client helper is pretending to be the service object.
- On the server side, the service helper receives the request from the client helper (through a Socket connection), unpacks the information about the call, and then invokes the real method on the real service object.
- The service helper gets the return value from the service. packs it up, and ships it back (over a Socket's output stream) to the client helper.

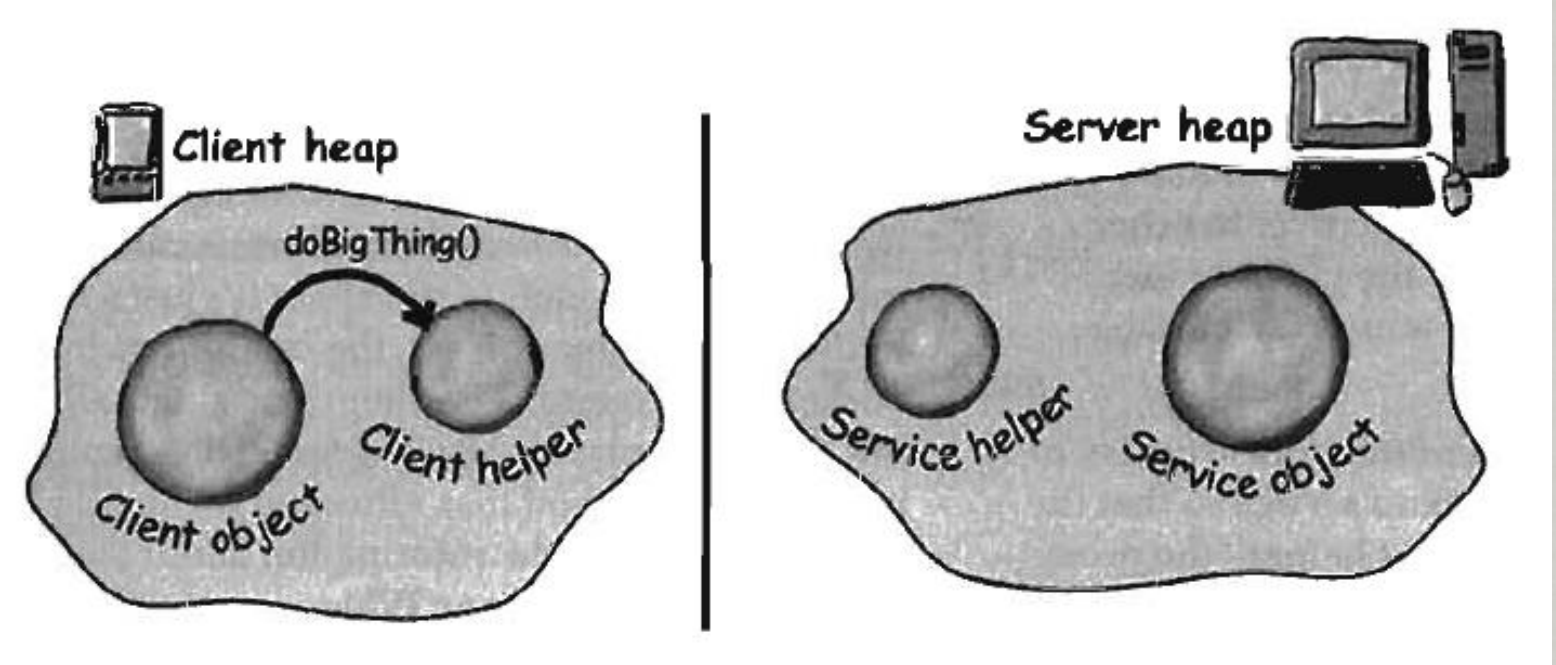
Mahmoud Badry, mma18@fayoum.edu.eg

Helpers Role



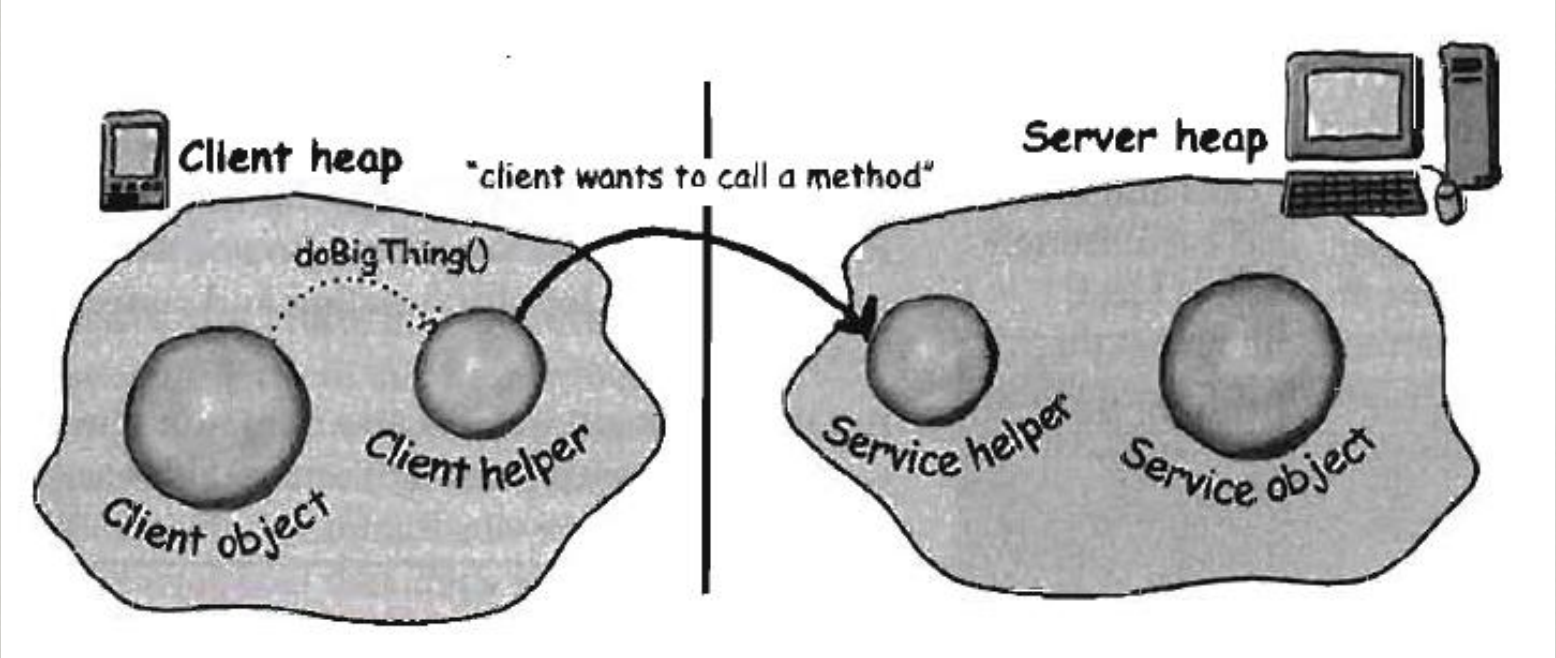
Mahmoud Badry, mma18@fayoum.edu.eg

doBigThing() example



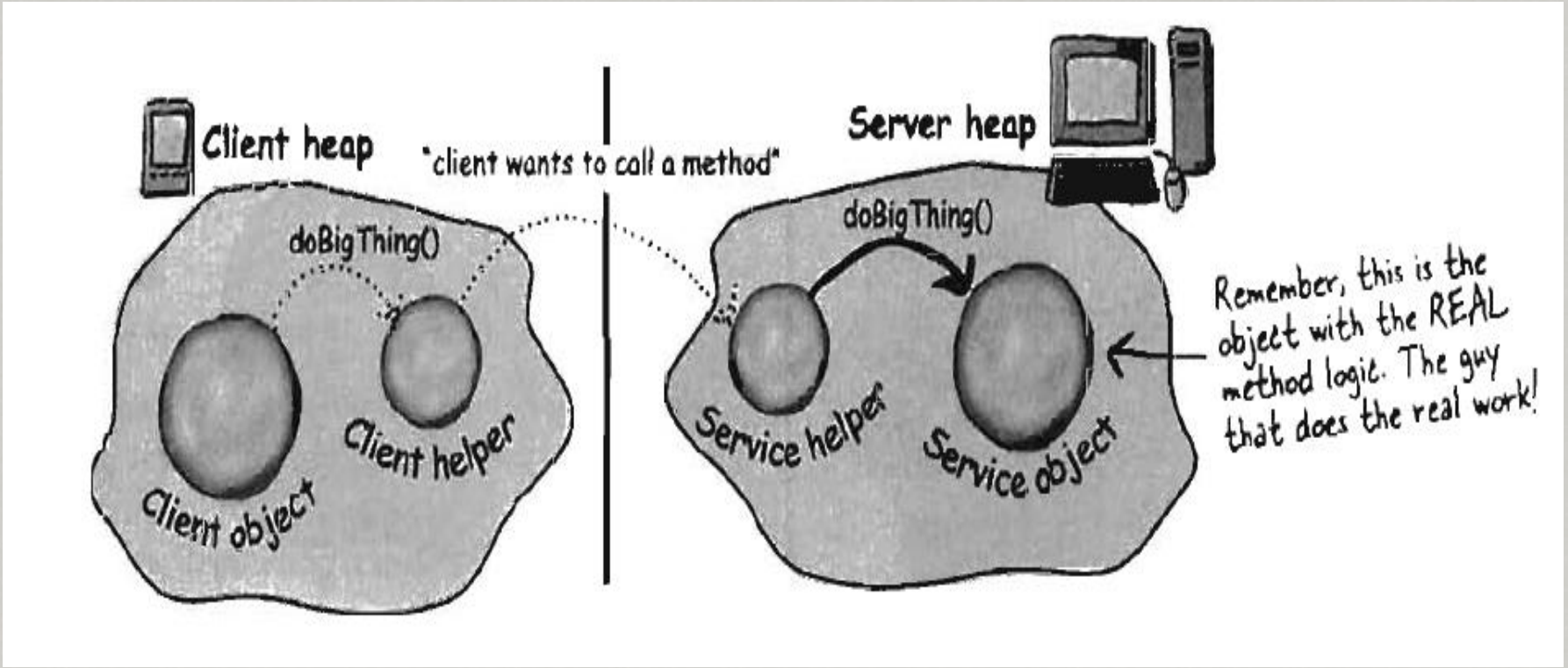
Mahmoud Badry, mma18@fayoum.edu.eg

doBigThing() example cont.



Mahmoud Badry, mma18@fayoum.edu.eg

doBigThing() example cont.



Mahmoud Badry, mma18@fayoum.edu.eg

Explaining JAVA RMI

Mahmoud Badry, mma18@fayoum.edu.eg

Java RMI gives you the client and service helper objects!

- It even knows how to make the **client helper** look like the **real Service**.
- With Java RMI. you don't write **any** of the **networking** or **I/O code** yourself. The client gets to call **remote methods**.
- But you have to know that there are **actually** networking and I/O code is running and you have to **handle exceptions** that are **raised** from calling such methods.

Mahmoud Badry, mma18@fayoum.edu.eg

Java RMI protocols

- A **remote call** means a method that's invoked on an object on another **JVM**. How the **information** about that call gets **transferred** from one **JVM** to **another** depends on the **protocol** used by the helper objects.
- RMI has two protocols, JRMP and IIOP.
- JRMP is RMI's 'native' protocol, the one made just for **Java-ta-Java** remote calls.
- IIOP, is the protocol for **CORBA** (Common Object Request Broker Architecture) and works on objects that are no necessary java.
- But thankfully, all we care about is Java-to-Java, so we're sticking with plain old. remarkably easy RMI.

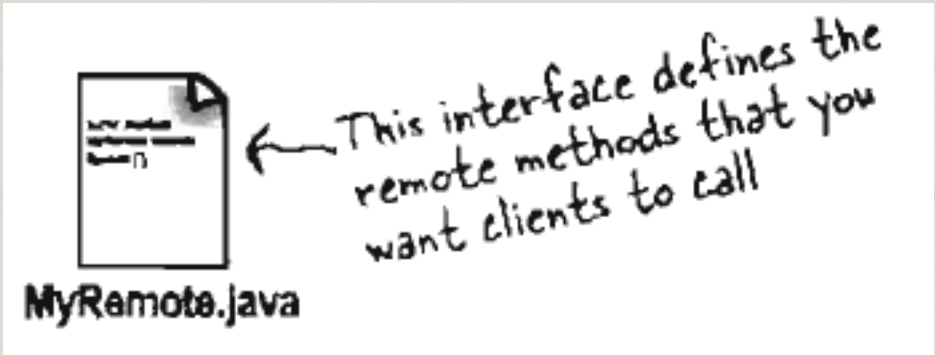
Making Remote service

Making the server remote service in five steps

Mahmoud Badry, mma18@fayoum.edu.eg

Step One: Make a Remote Interface

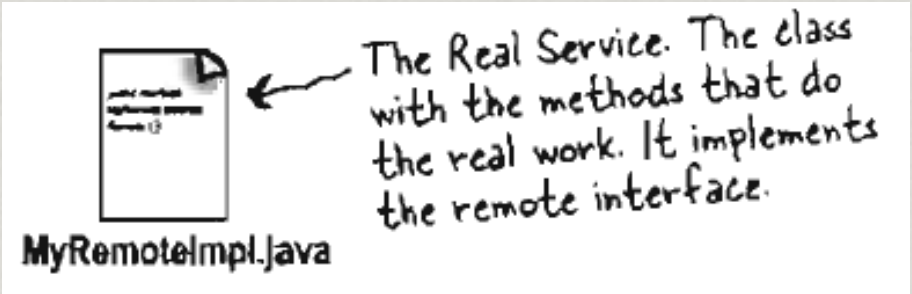
- Create "MyRemote.java" class that extends java.rmi.Remote. This interface will contain all the **methods** that a **client** can call.
- **Declare** all the **methods** you want and make them **throw** RemoteException.
- Be sure arguments and return values are primitives or Serializable.



Mahmoud Badry, mma18@fayoum.edu.eg

Step Two: Make a Remote Implementation

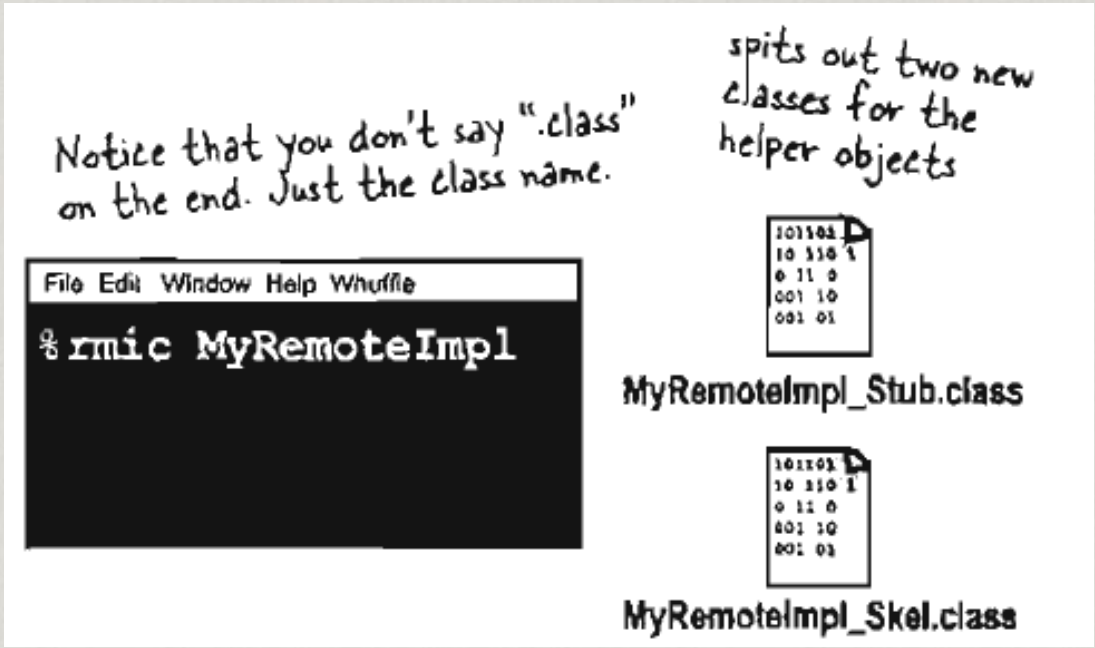
- Create "MyRemoteImp.java" class that extends UnicastRemoteObject and implements the interface you have created.
- Create an empty constructor that throws RemoteException.
- Implement Methods you have added into your interface.
- Create a main method inside the class. Inside that main create a new instance of the class you've created and using Naming.Rebind give your service a name.



Mahmoud Badry, mma18@fayoum.edu.eg

Step Three: Generate stubs and skeletons (Helpers)

- Run **rmic** on the remote implementation **class** (not the remote: Interface)



Mahmoud Badry, mma18@fayoum.edu.eg

Step Four: run rmiregistry

1 **Bring up a terminal and start the rmiregistry.**

Be sure you start it from a directory that has access to your classes. The simplest way is to start it from your 'classes' directory.

Mahmoud Badry, mma18@fayoum.edu.eg

Step Five: Start the service

1 Bring up another terminal and start your service

This might be from a main() method in your remote implementation class, or from a separate launcher class. In this simple example, we put the starter code in the implementation class, in a main method that instantiates the object and registers it with RMI registry.

Running a simple RMI Server and client

Mahmoud Badry, mma18@fayoum.edu.eg

QUESTIONS

Mahmoud Badry, mma18@fayoum.edu.eg

THANK YOU!

Mahmoud Badry, mma18@fayoum.edu.eg