2 : Remote Method Invocation

IT4206 – Enterprise Application Development

Level II - Semester 4

Overview

- This topic will discuss what is remote method invocation (RMI) and advantages of using RMI. It will guide to develop a simple program using RMI.

Intended Learning Outcomes

- At the end of this lesson, you will be able to;
  - Explain Remote Method Invocation (RMI)
  - Discuss the importance of having RMI in distributed computing
  - Develop a simple program using RMI
  - Explain how the RMI works

List of sub topics

2.1 Describing remote method invocation

2.2 Parsing behavior of remote method invocation

2.3 Using RMI API to establish client and server communications

Remote Method Invocation

- It is easy when all the parts of the application is in one place, in one single heap, in one JVM.
- BUT, it is not possible all the time.
- What if the application is handling powerful computation which is impossible in a little Java enabled device?
- What if the application need to access the database, but for security reasons only the only the application in the server allowed to access the database?

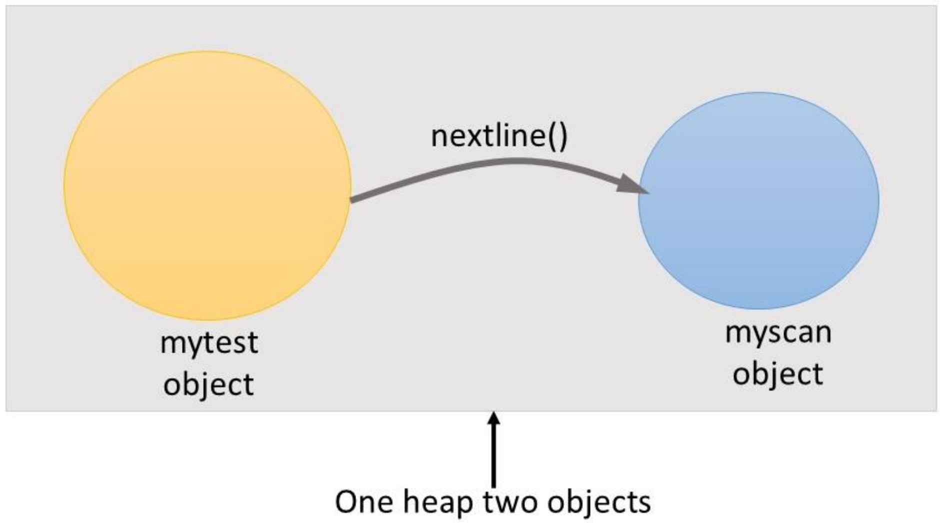
Remote Method Invocation

- Consider the following code segment

```
public class test{

  public void myinput(){
      Scanner myscan = new Scanner(System.in);
      System.out.print("Enter your name :");
      String name = myscan.nextLine();
      System.out.print("Name is "+name);
  }

  public static void main(String [] args){
    test mytest = new test();
    mytest.myinput();
  }
}
```

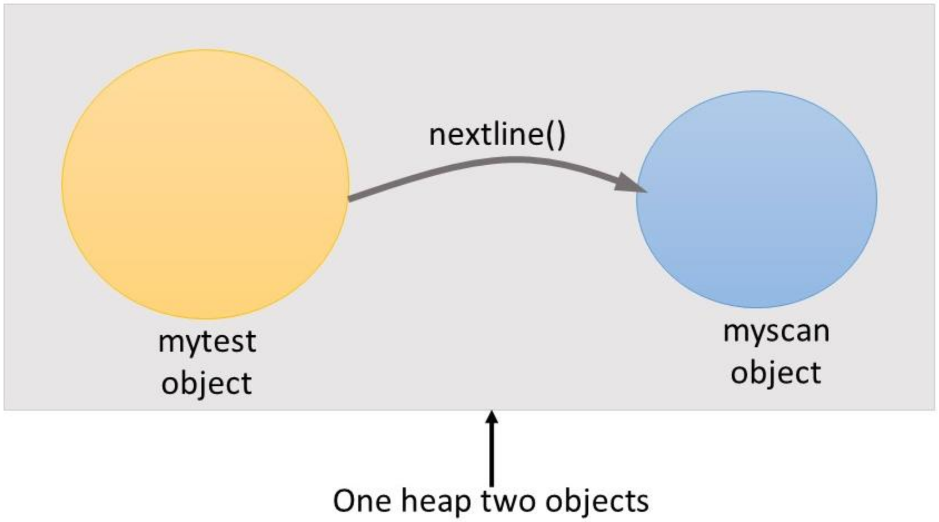
Remote Method Invocation

- Test instance is referred by mytest and Scanner object is referred by myscan are both in the same heap run by the same JVM.
- The JVM knows where the objects are located and how to communicate with them.
- But it is only valid for the objects with the heap of JVM.



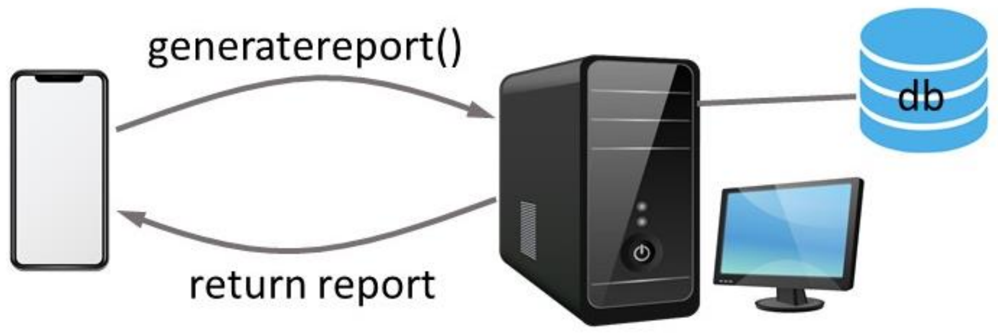
Remote Method Invocation

- But, the JVM can know about references on only its own heap.
- JVM doesn’t know about the heap of of JVM running on another machine or about a different JVM running on same machine.



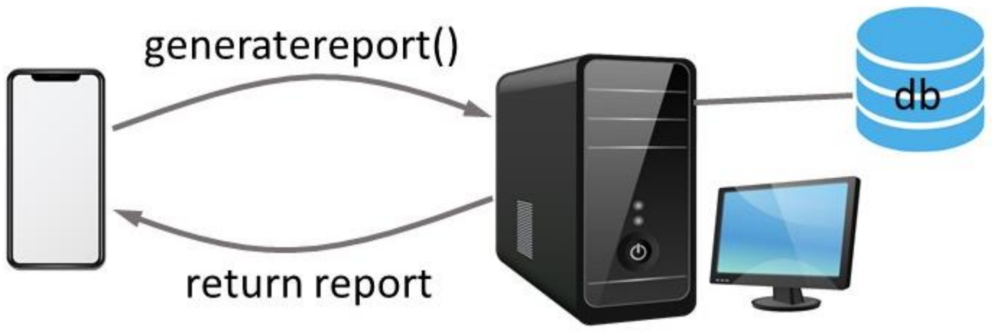
Remote Method Invocation

- Consider the following scenario.
- A Java application is running on a Java enabled device with less computational power. But it needs to perform a calculation including database which need high computational power.



Remote Method Invocation

- In this scenario, this is difficult. Because it is impossible to get the reference to an object in a separate heap.



Remote Method Invocation

- Imagine if it is possible to write a code in a similar way that invoking a method on a local object and invoke a method on remote object (Object in a different heap).
- This is where Remote Method Invocation (RMI) is needed.
- How to perform remote method calls
- It needs,
    - Server
    - Client
    - Server helper
    - Client helper

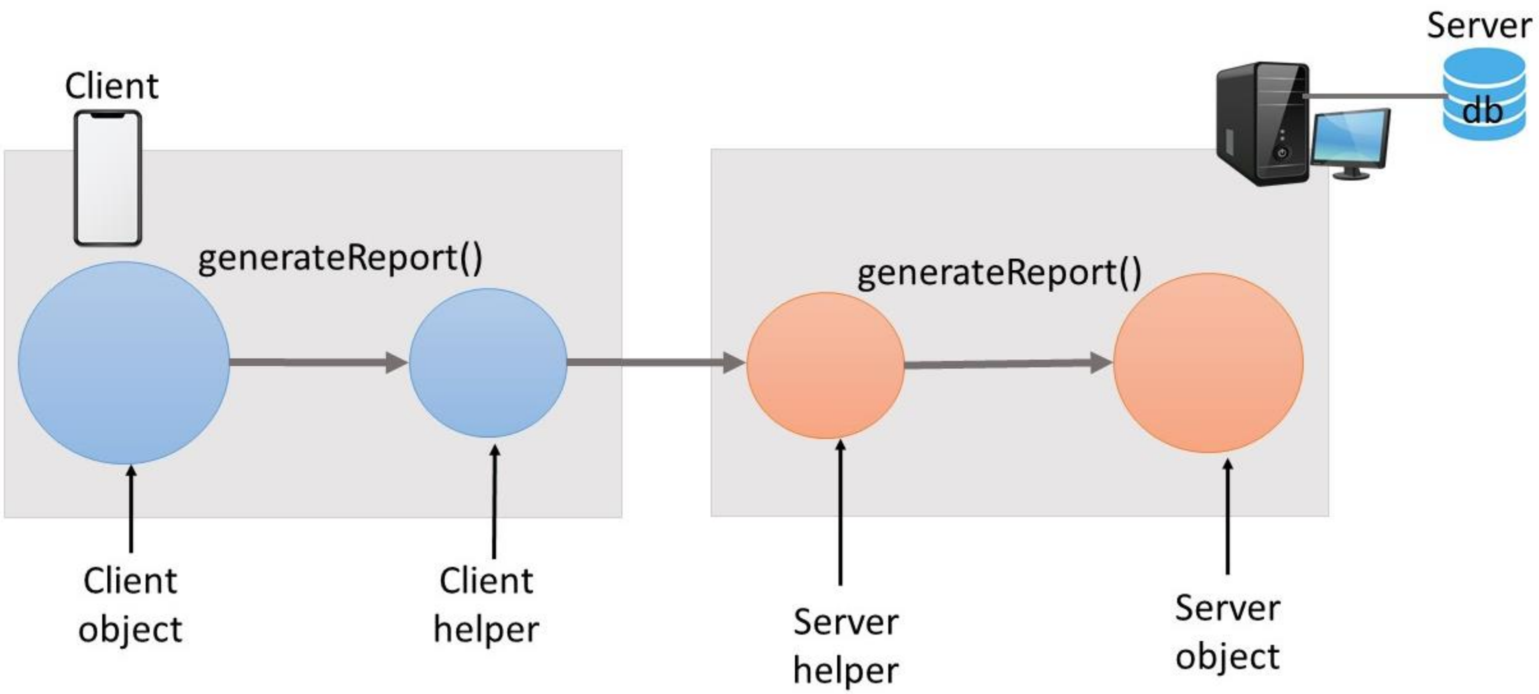
Remote Method Invocation

- What are helpers?
- They are the objects which do the communication.
- Client helper is within the same heap of client app.
- Client calls a method on the client helper as if the client helper were the actual service. Client helper pretend to be the service object.
- But client helper doesn’t have any actual method logic.
- So, the client helper contact the server and transfer the information about the method call.

Remote Method Invocation

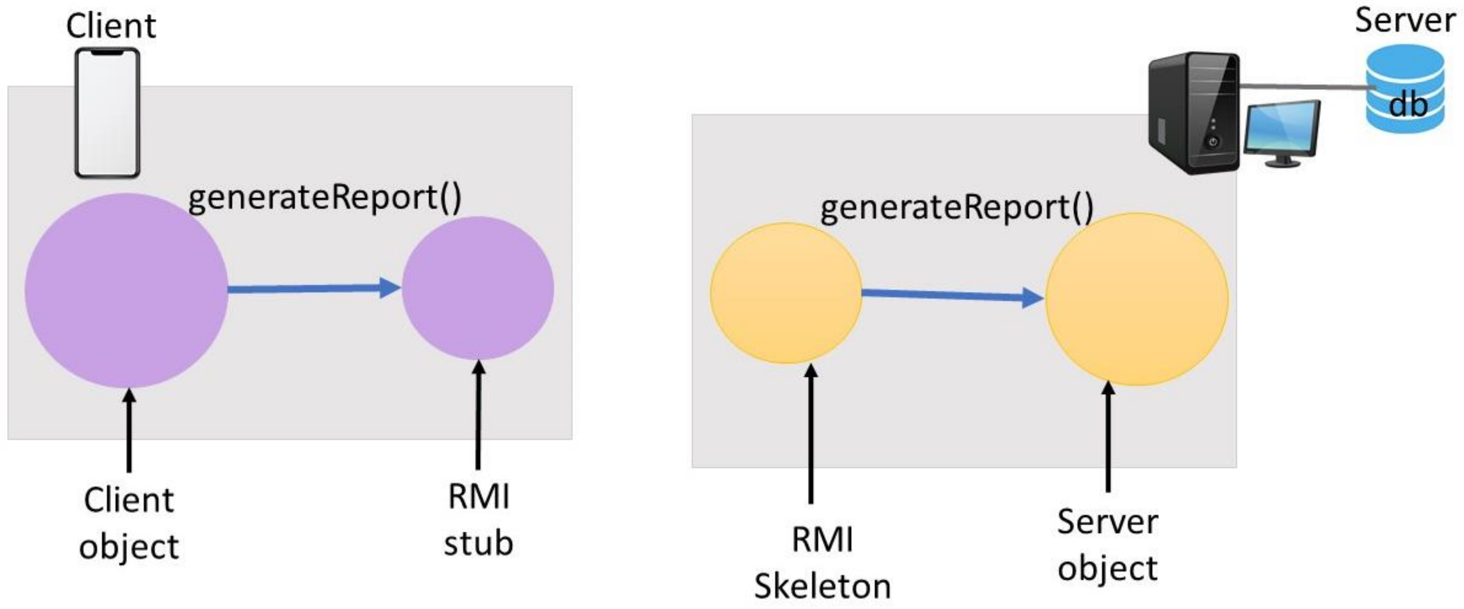
- What are helpers?
- On the server side, there is a server helper which receives the request from the client helper.
- It unpack the request, invoke the real method on the real service object, because it is available within the heap of sever helper.
- Server helper get the return value from the service, pack it and send back to the client helper.
- Client helper will return the value to the client object.

Remote Method Invocation



Remote Method Invocation

- In RMI, the client helper is known as ‘stub’ and server helper is known as ‘skeleton’



Marshalling and Unmarshalling in Remote Method Invocation

- When calling a remote method, necessary information/parameters need to be transferred to the server from the client and return values need to be transferred to the client from the server.
- The process of collecting information/parameters, convert and a standard format before transmitting over the network is known as the **Marshalling.**
- **Unmarshalling** is the opposite process of marshalling. The return values from the remote method invocation are unmarshalled, to the format which can be execute.

Remote Method Invocation

- Java Remote Method Invocation (RMI) allow to write distributed objects using Java.
- RMI provides a simple and direct model for distributed computation with Java objects.
- The objects can be two types

1. Java Objects
2. Simple Java wrappers around an existing API

Remote Method Invocation

- RMI is centred around Java.
- RMI has the Java safety and the portability.
- Behaviors, such as agents and business logic, can be moved to the part of the network where it makes the most sense.
- RMI connects to existing and legacy systems using the standard Java native method interface JNI.
- RMI connects to existing relational database using the standard JDBC package.

Advantages of RMI

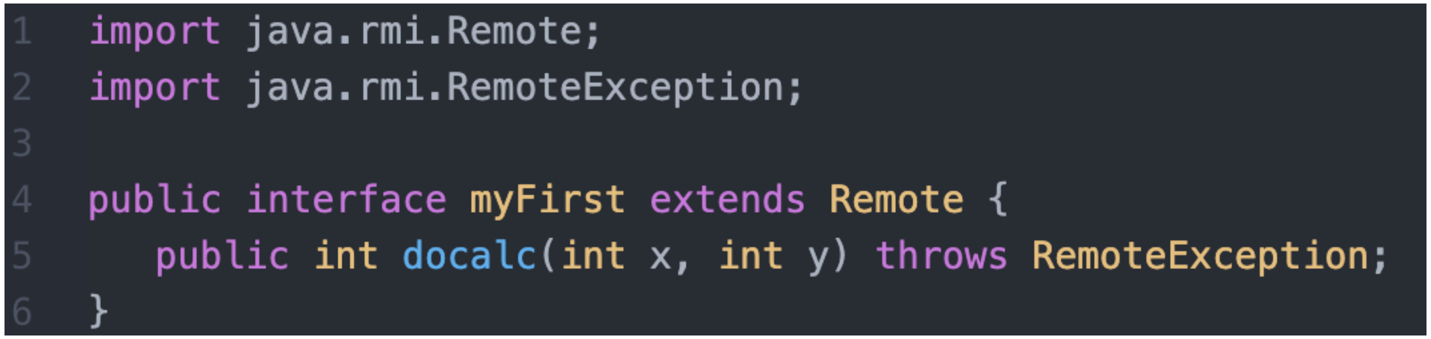
- Object Oriented - RMI can pass full objects as arguments and return values, not just predefined data types.
- Mobile Behavior - RMI can move behavior from client to server and server to client.
- Design Patterns - can use object oriented design patterns.
- Safe and Secure - RMI uses built-in Java security mechanisms that allows the system to be safe when users downloading implementations.
- Easy to Write/Easy to Use - RMI makes it simple to write remote Java servers and Java clients that access those servers.

Advantages of RMI

- Connects to Existing/Legacy Systems - RMI interacts with existing systems through Java's native method interface JNI.
- Write Once, Run Anywhere - RMI is part of Java's "Write Once, Run Anywhere" approach. Any RMI based system is 100% portable to any Java Virtual Machine *, as is an RMI/JDBC system.
- Distributed Garbage Collection - RMI uses its distributed garbage collection feature to collect remote server objects that are no longer referenced by any clients in the network.
- Parallel Computing - RMI is multi-threaded, allowing the servers to exploit Java threads for better concurrent processing of client requests.
- The Java Distributed Computing Solution - RMI is part of the core Java platform starting with JDK 1.1, so it exists on every 1.1 Java Virtual Machine.

Simple Client Server Communication with RMI

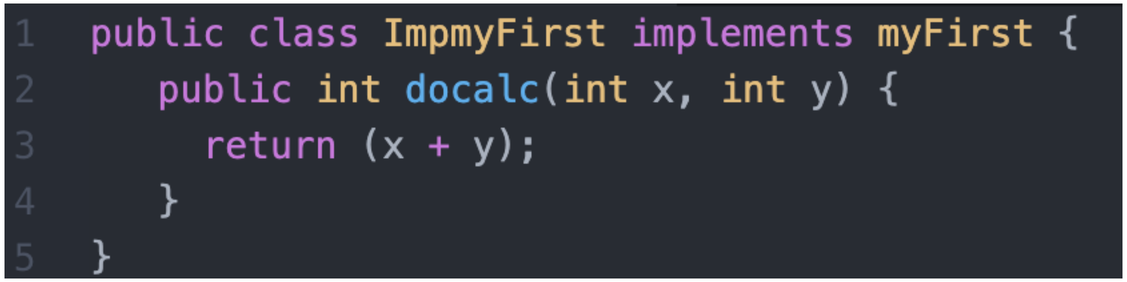
- Make the remote interface (myFirst.java)
  - Remote is a maker interface. It has not implementation



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

public interface myFirst extends Remote {
    public int docalc(int x, int y) throws RemoteException;
}
```

Simple Client Server Communication with RMI

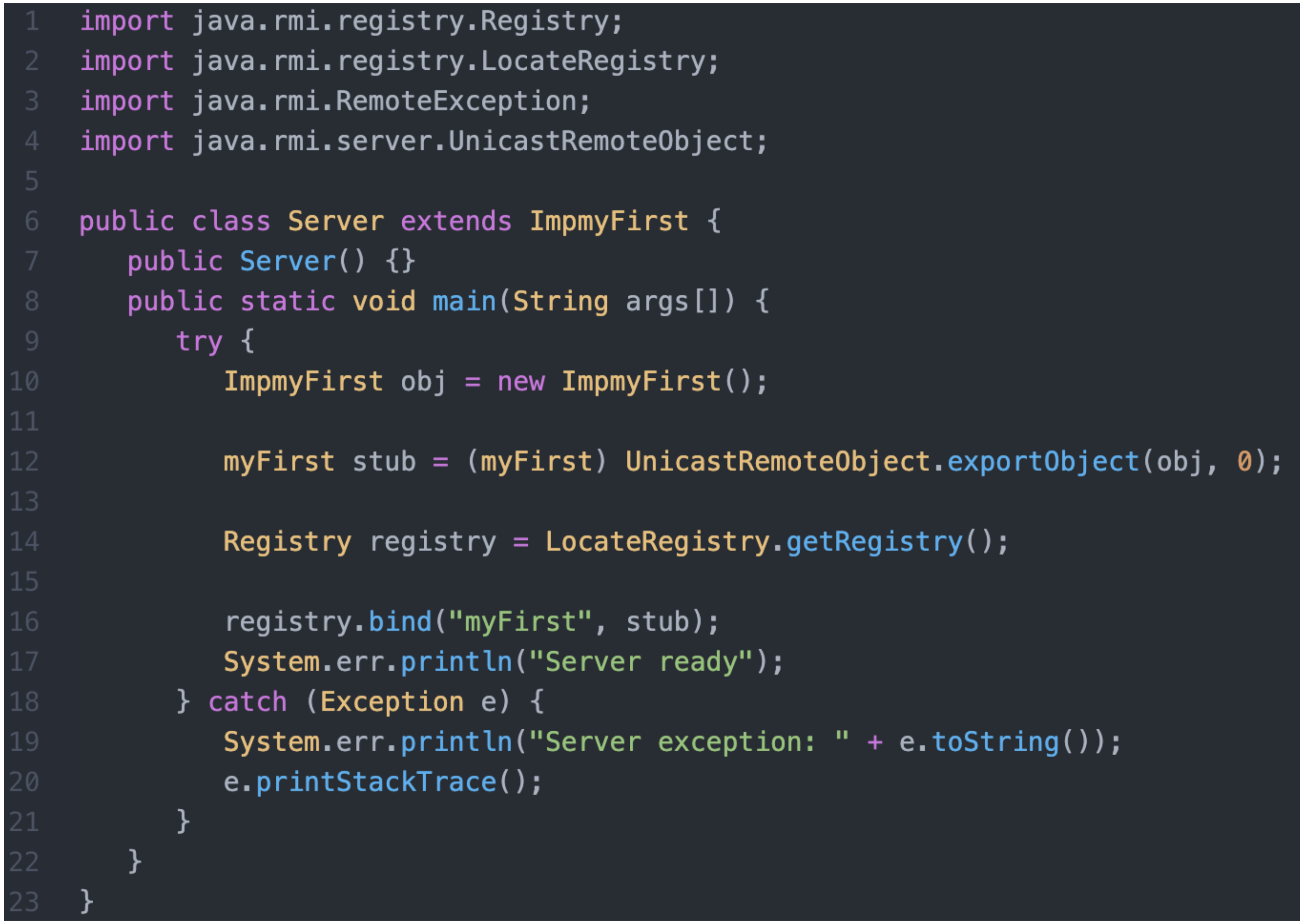
- Make the remote implementation (ImpmyFirst.java)



```java
public class ImpmyFirst implements myFirst {
    public int docalc(int x, int y) {
      return (x + y);
    }
}
```

Simple Client Server Communication with RMI

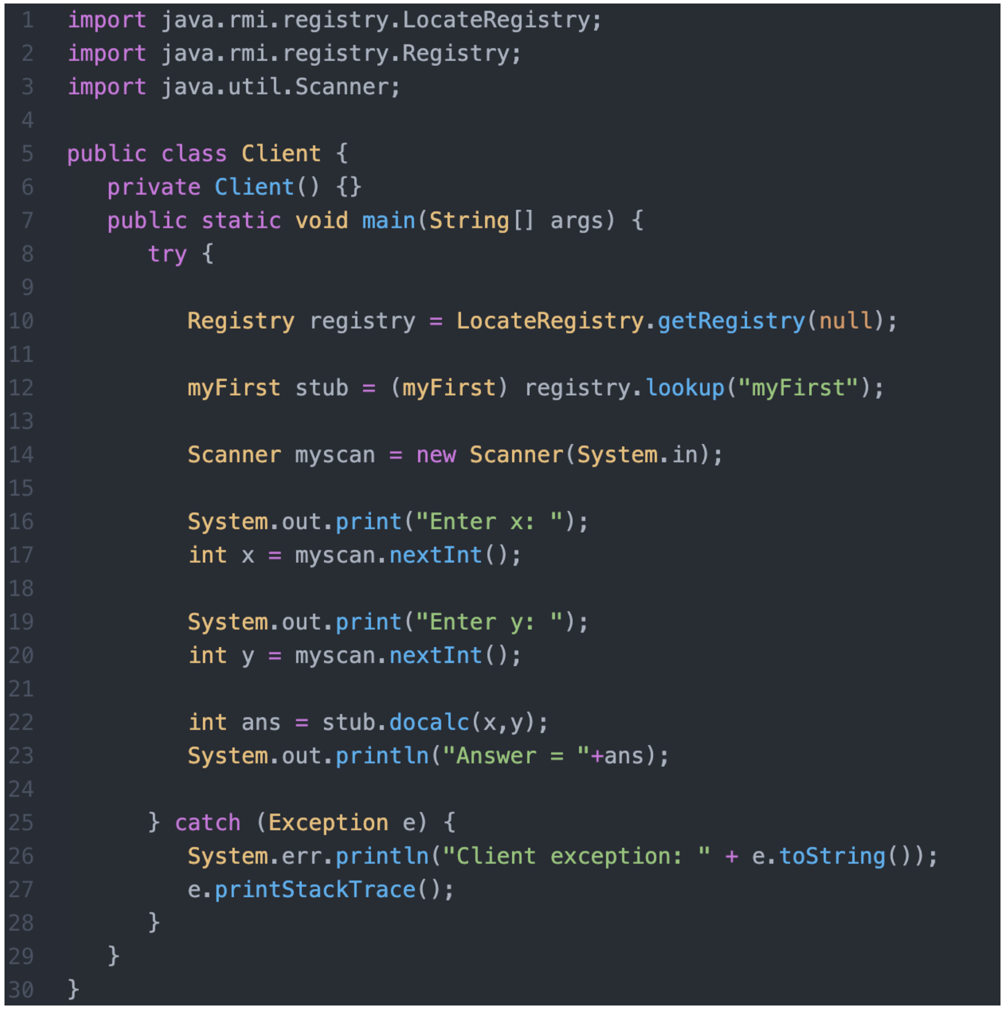
- Implement the Server Program (Server.java)



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

public class Server extends ImpmyFirst {
   public Server() {}
   public static void main(String args[]) {
      try {
         ImpmyFirst obj = new ImpmyFirst();

         myFirst stub = (myFirst) UnicastRemoteObject.exportObject(obj, 0);

         Registry registry = LocateRegistry.getRegistry();

         registry.bind("myFirst", stub);
         System.err.println("Server ready");
      } catch (Exception e) {
         System.err.println("Server exception: " + e.toString());
         e.printStackTrace();
      }
   }
}
```

Simple Client Server Communication with RMI

- Implement the Client Program (Client.java)



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

public class Client {
   private Client() {}
   public static void main(String[] args) {
      try {

         Registry registry = LocateRegistry.getRegistry(null);

         myFirst stub = (myFirst) registry.lookup("myFirst");

         Scanner myscan = new Scanner(System.in);

         System.out.print("Enter x: ");
         int x = myscan.nextInt();

         System.out.print("Enter y: ");
         int y = myscan.nextInt();

         int ans = stub.docalc(x,y);
         System.out.println("Answer = "+ans);

      } catch (Exception e) {
         System.err.println("Client exception: " + e.toString());
         e.printStackTrace();
      }
   }
}
```

Simple Client Sever Communication with RMI

- Compile the Java codes

```
% javac *.java
```

- To make it more interesting lets create two directories are "Server" and "Client", and copy the files to the directories as follows.

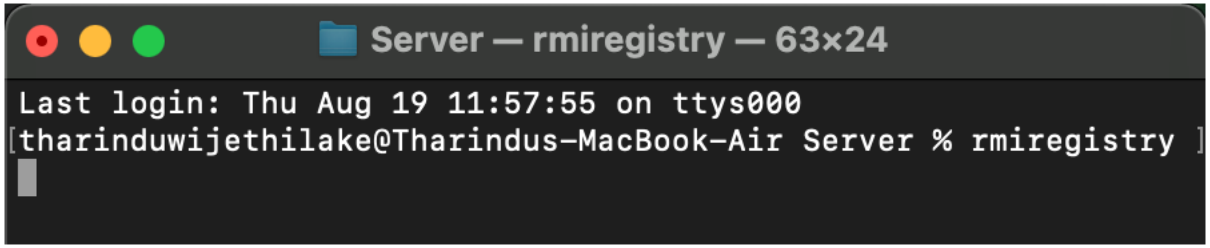
```
Server ->                          Client ->
        ImpmyFirst.class                   myFirst.class
        myFirst.class                      Client.class
        Server.class
```

Simple Client Server Communication with RMI

- Open a new terminal inside the Server directory and run the RMI registry

% rmiregistry (for Linux)

start rmiregistry (for windows)

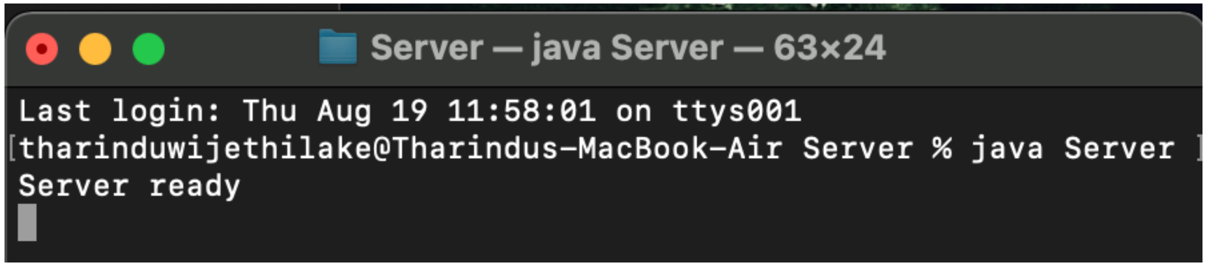


```
Last login: Thu Aug 19 11:57:55 on ttys000
tharinduwijethilake@Tharindus-MacBook-Air Server % rmiregistry
```

Simple Client Server Communication with RMI

- Open a new terminal inside the Server directory and start the Server program

```
% java Server
```

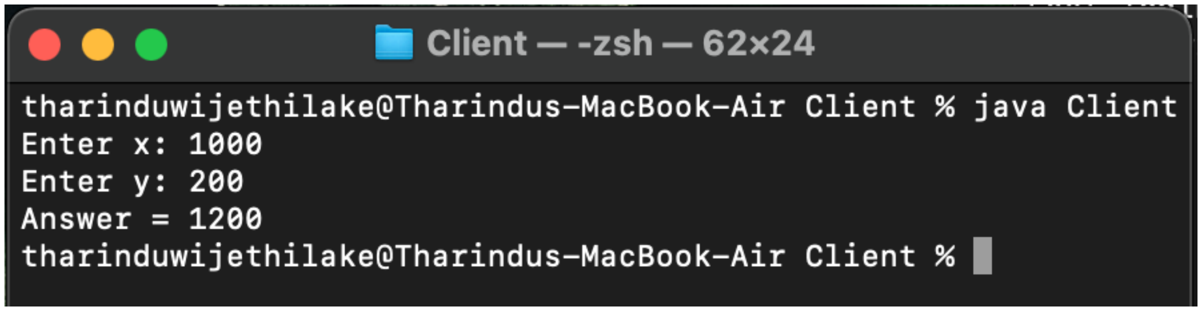


```
Last login: Thu Aug 19 11:58:01 on ttys001
tharinduwijethilake@Tharindus-MacBook-Air Server % java Server
Server ready
```

Simple Client Server Communication with RMI

- Open a new terminal inside the Client directory and start the Client program

```
% java Client
```



```
tharinduwijethilake@Tharindus-MacBook-Air Client % java Client
Enter x: 1000
Enter y: 200
Answer = 1200
tharinduwijethilake@Tharindus-MacBook-Air Client %
```

- Go through the code and understand what has happened.

What is RMI registry ?

- A remote object registry is a bootstrap naming service that is used by RMI servers on the same host to bind remote objects to names. Clients on local and remote hosts can then look up remote objects and make remote method invocations.

- Read more : https://docs.oracle.com/javase/7/docs/technotes/tools/solaris/rmiregistry.html

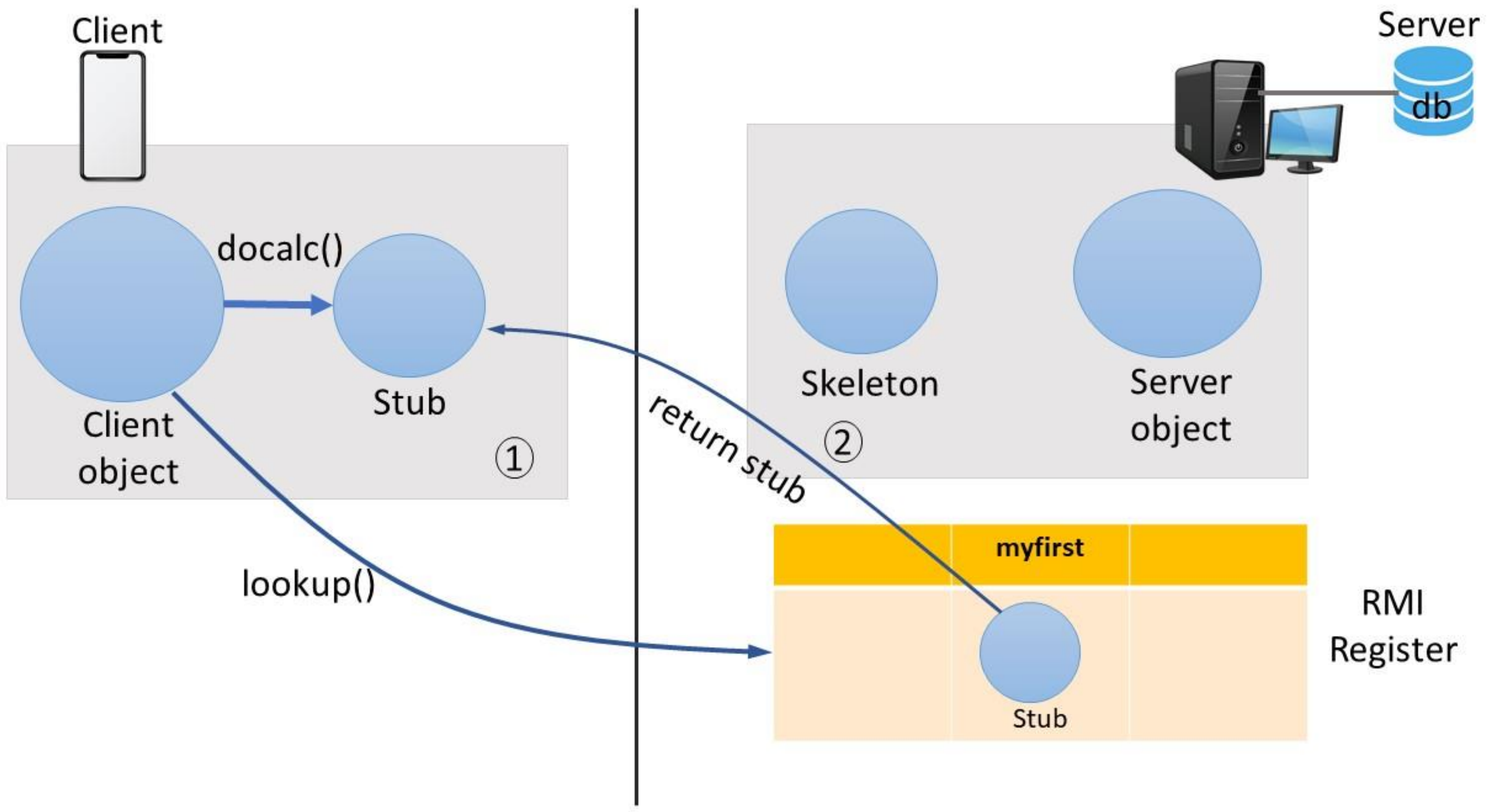
How does the Client get the stub object?

- The following code segment help the client to find the stub object.
- Client do the lookup on RMI registry

```
myFirst stub = (myFirst) registry.lookup("myFirst");
```

- RMI registry return the stub object.
- Client invoke the method on the stub.

How does the Client get the stub object?



Client
docalc()
Stub
Client object
①
lookup()
return stub
Server
db
Skeleton
Server object
②
myfirst
Stub
RMI Register