INTRODUCTION TO JAVA RMI

ELOI GABALDON

ELOIGABAL@GMAIL.COM

INTRODUCTION

- Java RMI is a library to implement a distributed system using objects
- In this library, the services and resources are implemented as objects that execute their methods remotely from another system
- The library provides the communication at a high level programming language, allowing the communication as simple as executing a method in a local object.
- It uses the ideas of Object Oriented Programming Languages

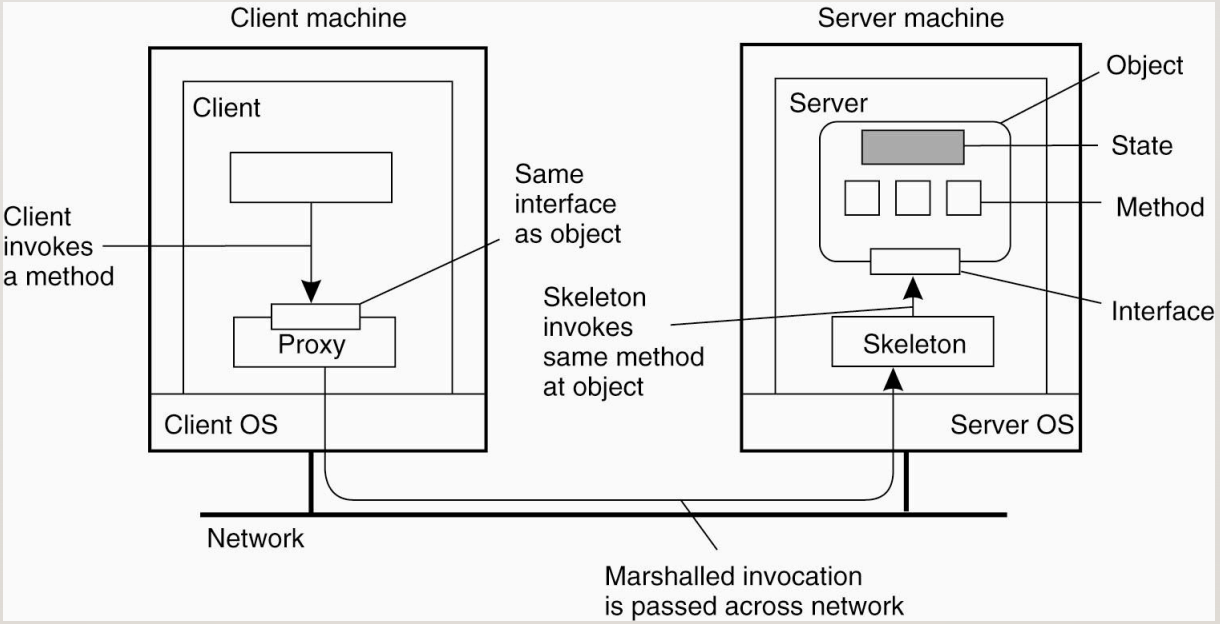
DISTRIBUTED OBJECTS

- An object contains:
  - **State**: attributes
  - **Operations**: methods
- The remote methods are available by means of interfaces
- An object can implement different interfaces
- An interface can be implemented by many objects

DISTRIBUTED OBJECTS

- RMI works using two main components:
  - Stub (proxy): loaded in the client side
    - Encodes the invocation of methods in the remote objects and decodes the response from them
  - Skeleton: loaded in the server side
    - Decodes the invocation of methods from the client and encodes the response.

DISTRIBUTED OBJECTS

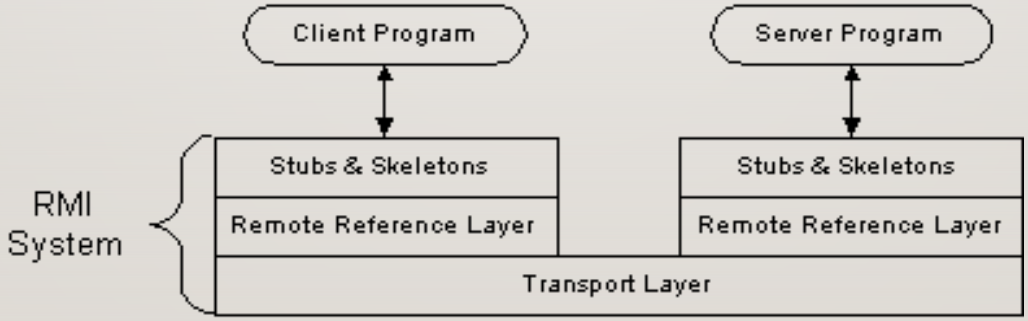


RMI VS LOCAL

	Local object	RMI Object
Object definition	In the object's class or using interfaces	Defined by an instance that must extend the Remote interface. Implemented in the server
Instance creation	Using the new operation with the object	Using the new operation in the server side. Clients never create instances of remote objects.
Access to the object	Using a reference to the instance	Accessed by a reference to the remote object using the Stub
Garbage collection	When no more references are made to the object	When no more local or remote references
Exceptions	Raised locally	Launched in the remote server and sent to the client

RMI WORKFLOW

- The workflow is divided in four layers
  - Developer code: The code that is implemented using Java RMI
  - Stub and skeletons: Intercept remote calls and forward them to the remote RMI service
  - Remote reference layer: Interpret the remote calls and manages the references of the remote objects
  - Transport layer: Provides the communication using TCP/IP protocol



RMI DIRECTORY

- RMI uses a register directory to keep the references to the remote objects
  - Runs in a well know host, so the clients can find the URL
    - Java Naming and Directory Interface (JNDI) can be used, as a complete service
    - RMI offers a simple Directory Registry useful for small application: **rmiregistry**

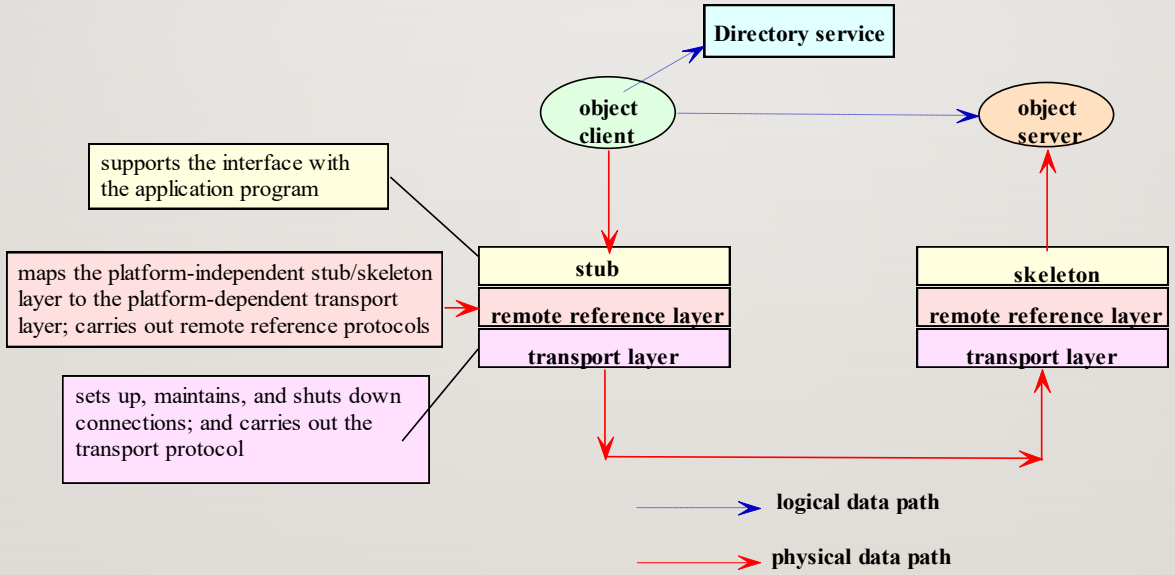
DIFFERENCES BETWEEN JNDI AND RMIREGISTRY

- RMIREGISTRY: Is a simple registry that is able to register server objects in the same machine as it is running (for each server machine, it is necessary to run a rmiregistry)
- JNDI: Is a centralized service that allows to register instances from many different servers.

RMIREGISTRY DIRECTORY

- Server Side:
  - The default port is 1099, but it can be changed with a parameter.
  - When a remote object is created in the server, it has to be registered in the registry directory
  - The registry is done by binding the object instance to a name.
- Client Side:
  - The client has to connect to the server's registry to get the object references.
  - The client can invoke the **lookup** method with the registered name to get the remote instance stub.
  - The client can use the object through the stub obtained after lookup.

RMI WORKFLOW OVERVIEW



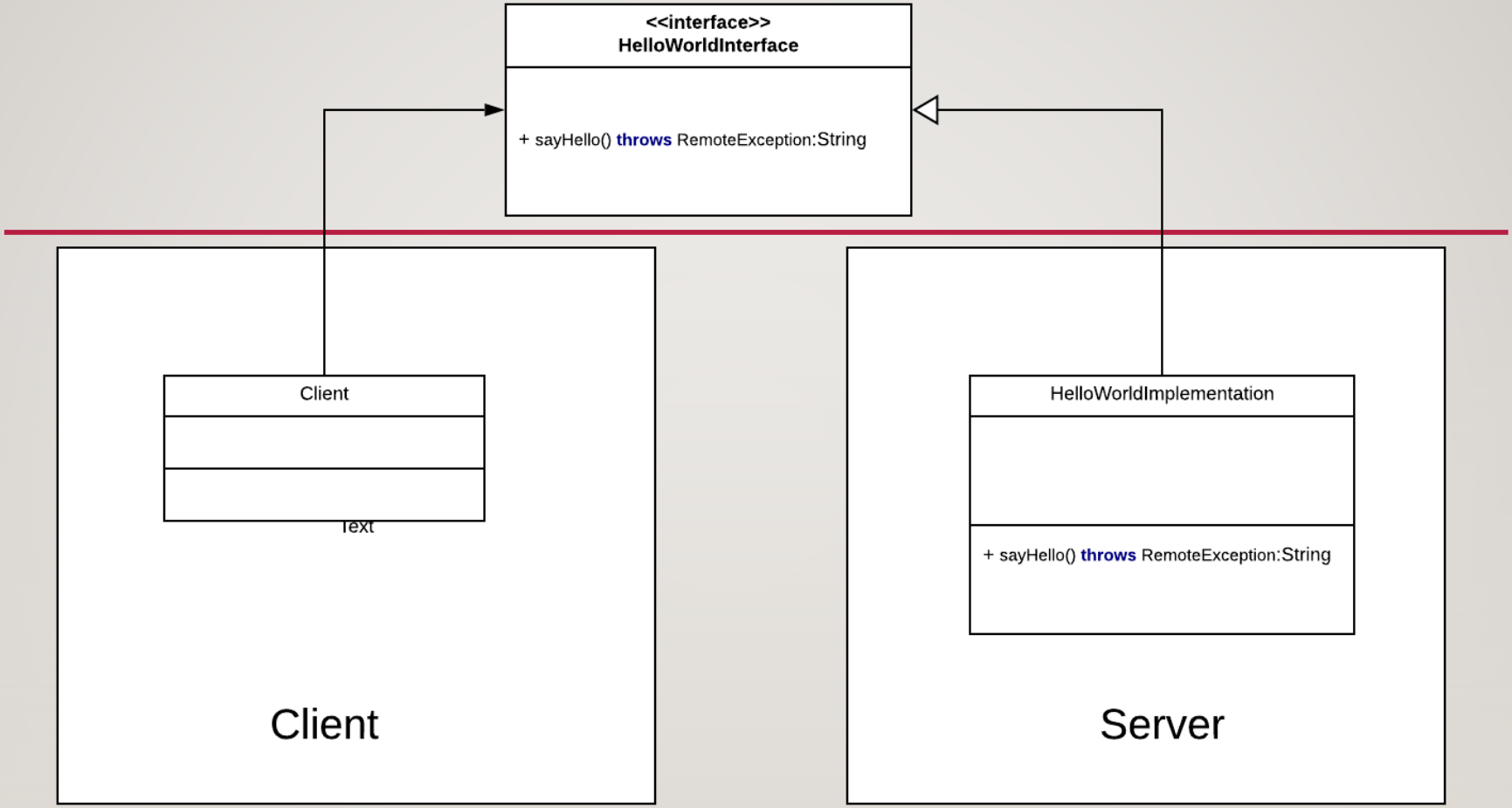
IMPLEMENT A DISTRIBUTED PROGRAM USING RMI

- Define the common interfaces and models in a common package
- Implement the server classes, importing the common project classes
- Implement the client classes, importing the common project classes
- Run the directory registry (in this course rmiregistry) in the same directory where the java server will be executed
- Run the server, that binds the remote objects into the directory registry
- Run the client

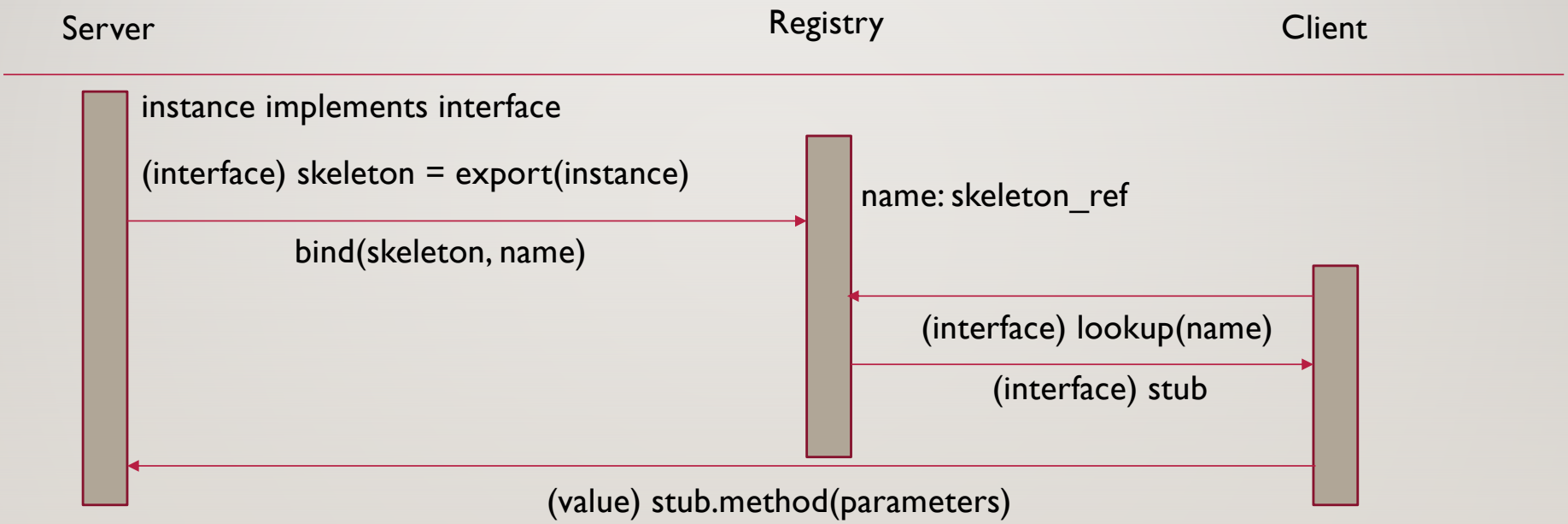
RMI Example

SUMMARY OF RMI

- Start the registry server! (Whether using the 'rmiregistry' or programmatically in the code)
- The remote object definition (interface extending Remote) has to be common in the server and the client.
- The remote object is implemented in the server.
- The server exports and registers the object in the directory.
- The client obtains the stub (object reference) using the lookup function.


<<interface>>
HelloWorldInterface
+ sayHello() throws RemoteException:String
Client
Text
HelloWorldImplementation
+ sayHello() throws RemoteException:String
Client
Server

SEQUENCE DIAGRAM



CLIENT-SERVER EXERSICES

1. Implement a client server application where the client calls the hello_world function with the user name, and the server responds “Hello <username>”.
2. Define the UML of an application to add, substract, multiply and divide two integers passed by parameter.
3. Define the Sequence Diagram, describing the interaction with the directory registry
4. Implement the previous defined application
5. Run the application using different clients
6. If possible, try the connection of clients in a different machine

Ask any questions during the class!