



Desarrollo de Aplicaciones Distribuidas

AUTORES:

Alejandro Calderón Mateos

Javier García Blas

David Expósito Singh

Laura Prada Camacho

Departamento de Informática
Universidad Carlos III de Madrid
Julio de 2012

JAVA RMI

Contenidos

1. Introducción:
 1. Paradigma de invocación remota de método
 2. Entorno de programación Java

2. Java RMI
 1. Introducción a RMI
 2. Arquitectura de RMI
 3. Desarrollo de aplicaciones distribuidas
 1. Interfaz y despliegue

Contenidos

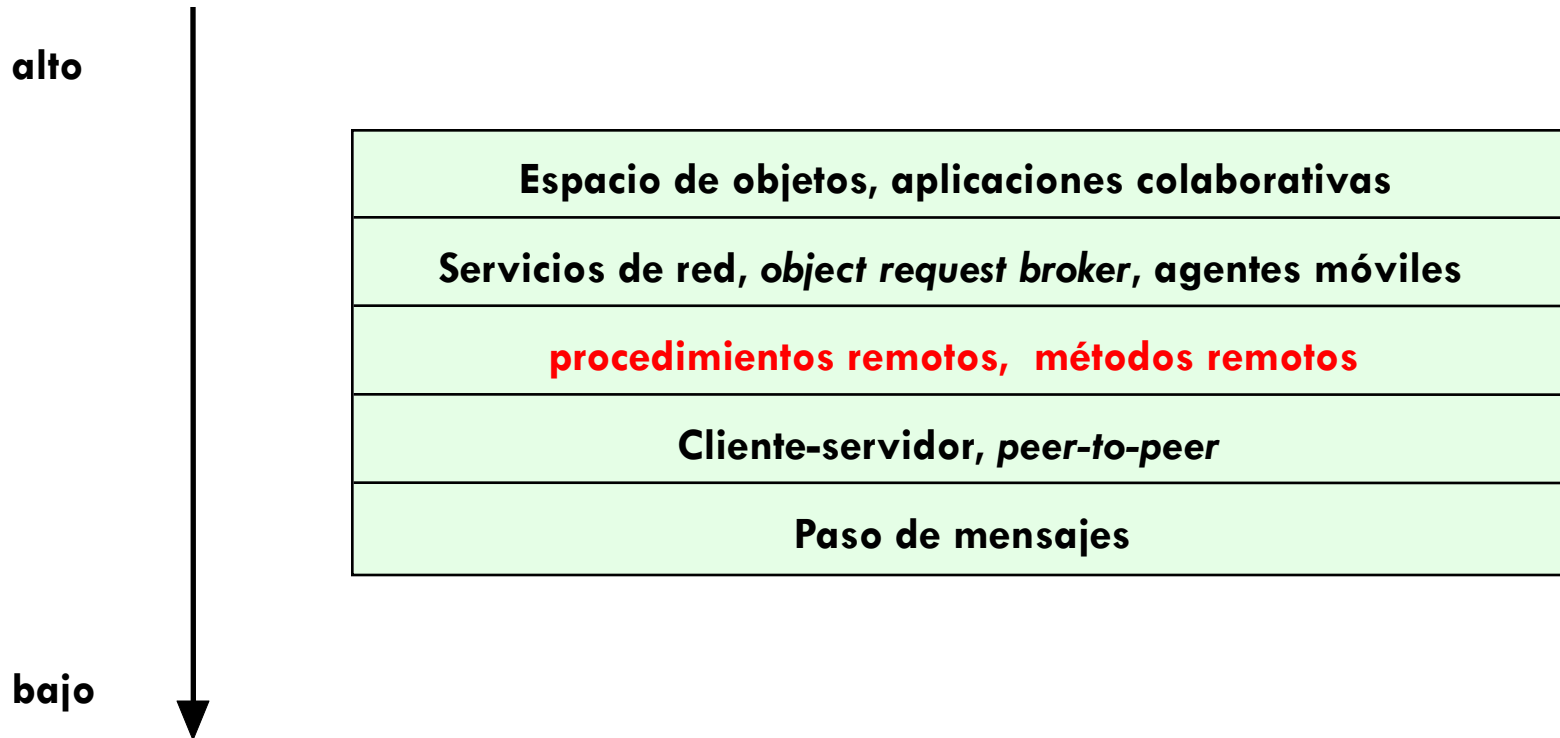
1. Introducción:

1. Paradigma de invocación remota de método
2. Entorno de programación Java

2. Java RMI

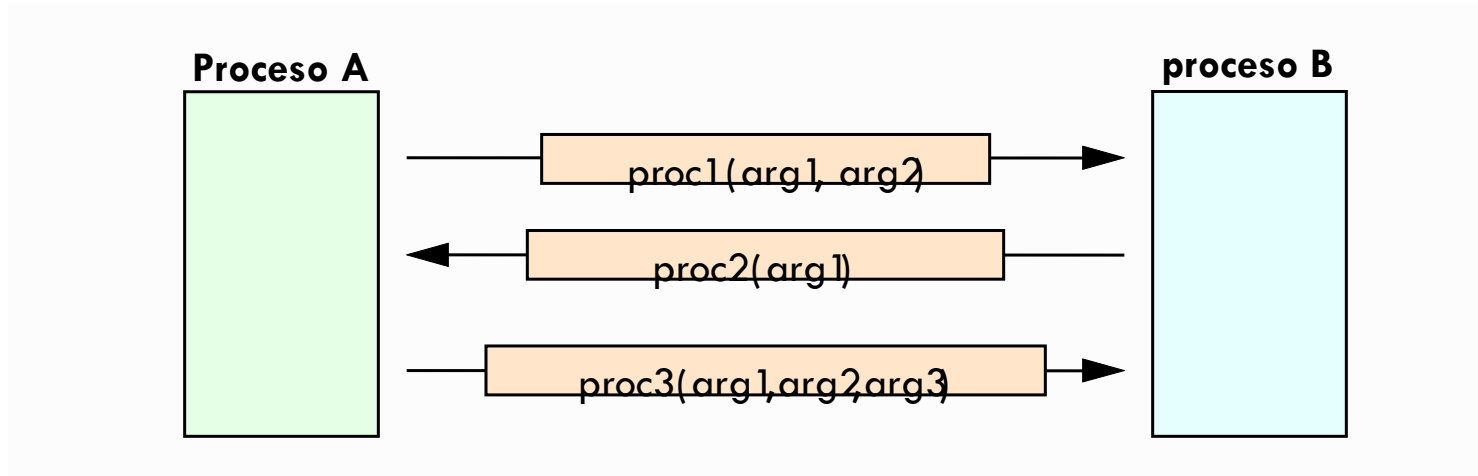
1. Introducción a RMI
2. Arquitectura de RMI
3. Desarrollo de aplicaciones distribuidas
 1. Interfaz y despliegue

Paradigmas de procedimientos/métodos remotos



Llamadas a procedimientos remotos

- ▶ **Objetivo:** hacer que el software distribuido se programe igual que una aplicación no distribuida.
- ▶ Mediante el modelo RPC la comunicación se realiza conceptualmente igual que la invocación de un procedimiento local.



Llamadas a procedimientos remotos

▶ Pasos:

- ▶ **A** llama al procedimiento remoto de **B**.
- ▶ La llamada dispara una acción de un procedimiento de **B**.
- ▶ Al finalizar el procedimiento, **B** devuelve el valor a **A**.

▶ Simplifica la comunicación entre procesos y la sincronización de eventos.

▶ Ejemplos:

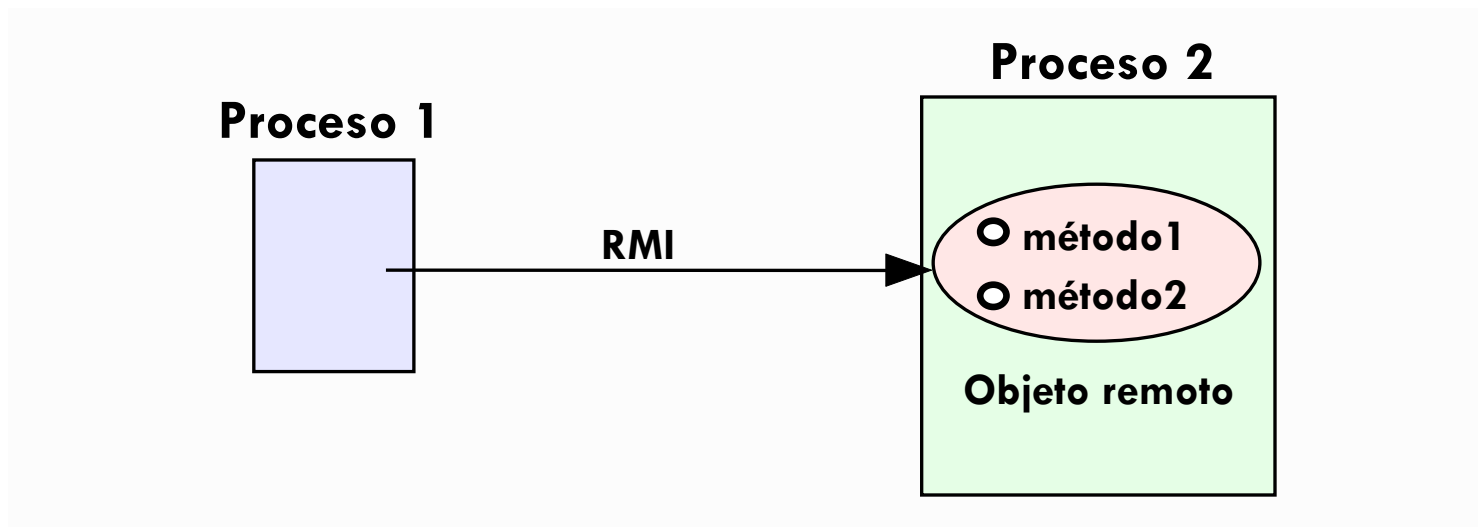
- ▶ *Open Network Computing Remote Procedure Call*, desarrollada a partir del API RPC de *Sun Microsystems* a comienzo de los años 80
- ▶ *Distributed Computing Environment (DCE) RPC* de *Open Group*
- ▶ *Simple objeto Access Protocol (SOAP)*

Llamada a métodos remotos

- ▶ Primera aproximación al uso de un modelo orientado a objetos sobre aplicaciones distribuidas
- ▶ **Objetos distribuidos dentro de una red**
 - ▶ Los **objetos proporcionan métodos**, los cuales **dan acceso a los servicios**
- ▶ **Ejemplo:**
 - ▶ *Remote method invocation* (RMI) de Java

Remote method invocation

- ▶ Modelo equivalente a las llamadas a procedimientos remotos
- ▶ Proceso invoca un método local de otro proceso
- ▶ Se envían tanto los argumentos del método como el valor devuelto por el mismo



Contenidos

1. Introducción:

1. Paradigma de invocación remota de método
2. Entorno de programación Java

2. Java RMI

1. Introducción a RMI
2. Arquitectura de RMI
3. Desarrollo de aplicaciones distribuidas
 1. Interfaz y despliegue

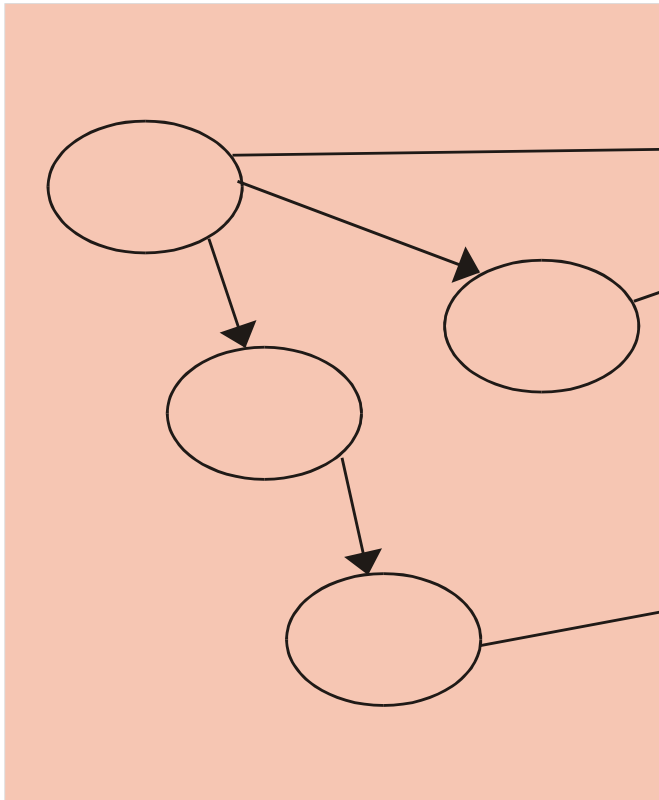
Contenidos

1. **Introducción:**
 1. Paradigma de invocación remota de método
 2. Entorno de programación Java

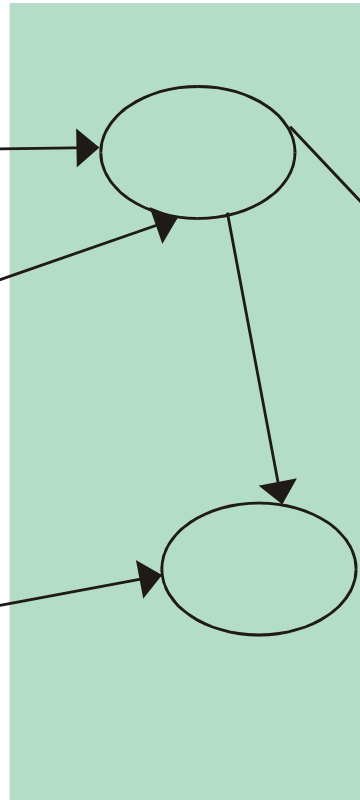
2. **Java RMI**
 1. **Introducción a RMI**
 2. Arquitectura de RMI
 3. Desarrollo de aplicaciones distribuidas
 1. Interfaz y despliegue

Modelo de objetos en sistemas distribuidos

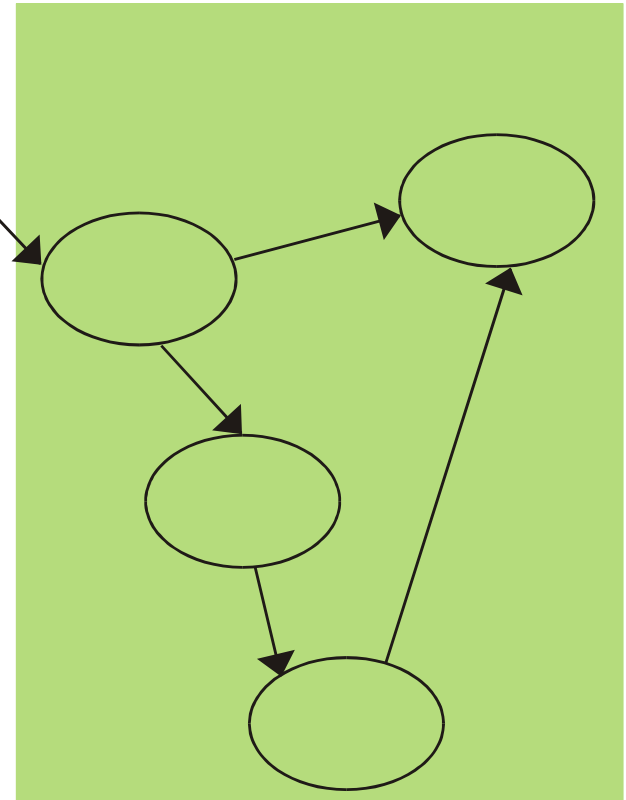
Máquina A



Máquina B



Máquina C



Java RMI (*Remote Method Invocation*)

- ▶ El soporte para RMI en Java está basado en las interfaces y clases definidas en los paquetes *java.rmi* y *java.rmi.server*.
- ▶ RMI ofrece:
 - ▶ Mecanismos para crear servidores y objetos cuyos métodos se puedan invocar remotamente.
 - ▶ Mecanismos que permiten a los clientes localizar los objetos remotos.
 - ▶ Servicio de directorios:
 - ▶ ***rmiregistry***, servicio de directorios de Java
 - ▶ Se ejecuta en la máquina servidor objeto

Comparación RMI y sockets

▶ **Ventajas:**

- ▶ Los programas RMI son más sencillos de diseñar.
- ▶ Servidor RMI concurrente.

▶ **Inconvenientes:**

- ▶ *Sockets* tienen menos sobrecarga.
- ▶ RMI sólo para plataformas Java.

Contenidos

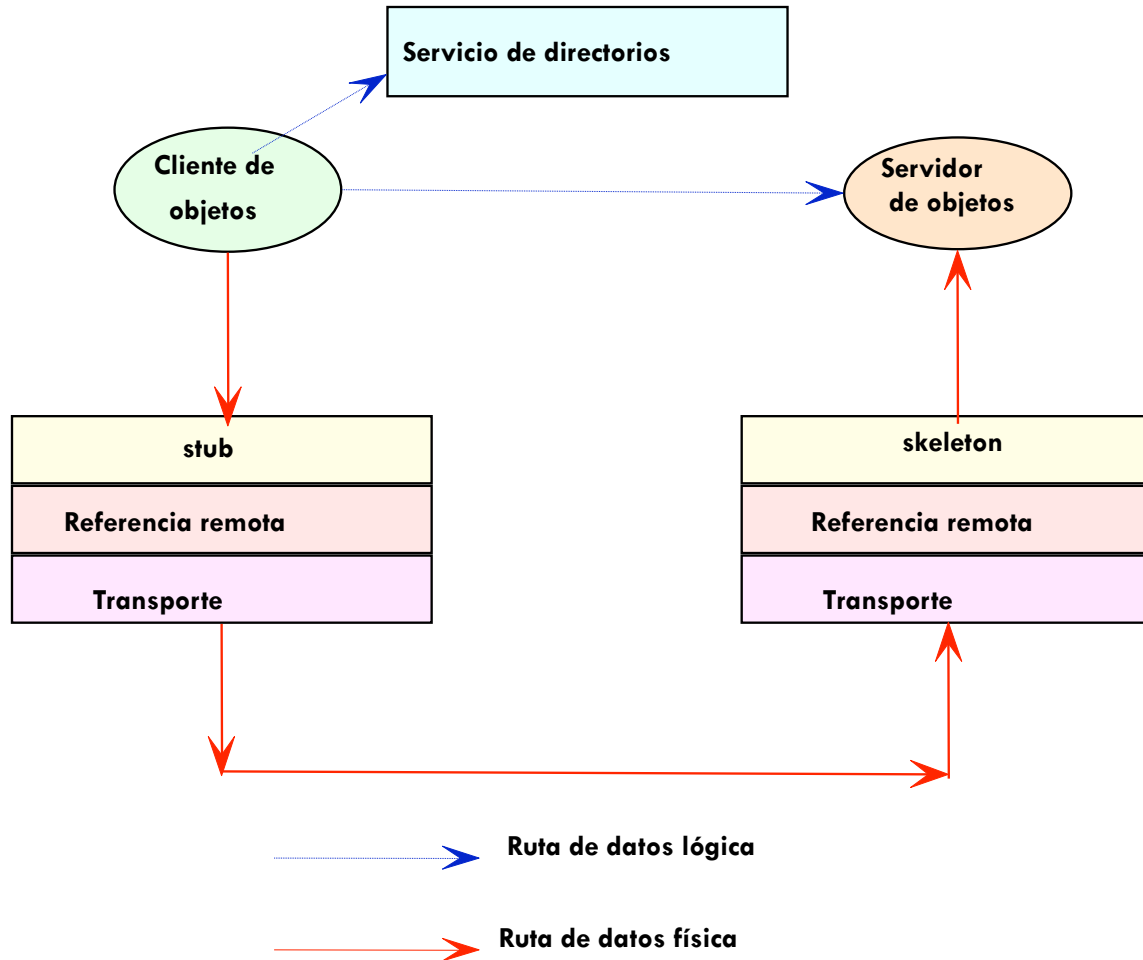
1. Introducción:

1. Paradigma de invocación remota de método
2. Entorno de programación Java

2. **Java RMI**

1. Introducción a RMI
2. **Arquitectura de RMI**
3. Desarrollo de aplicaciones distribuidas
 1. Interfaz y despliegue

Arquitectura de RMI



Arquitectura de RMI

▶ Nivel de resguardo o stub

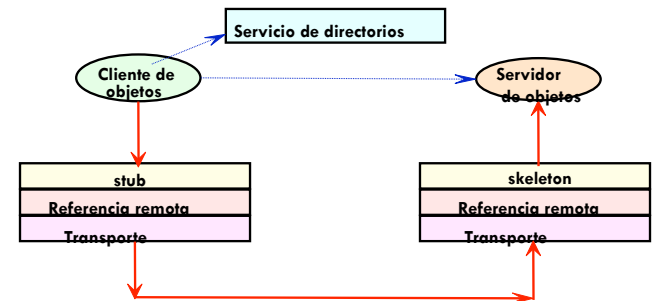
- ▶ Se encarga del aplanamiento de los parámetros.
- ▶ **Stub**: resguardo local. Cuando un cliente realiza una invocación remota, en realidad hace una invocación de un método del resguardo local.

▶ Nivel de gestión de referencias remotas

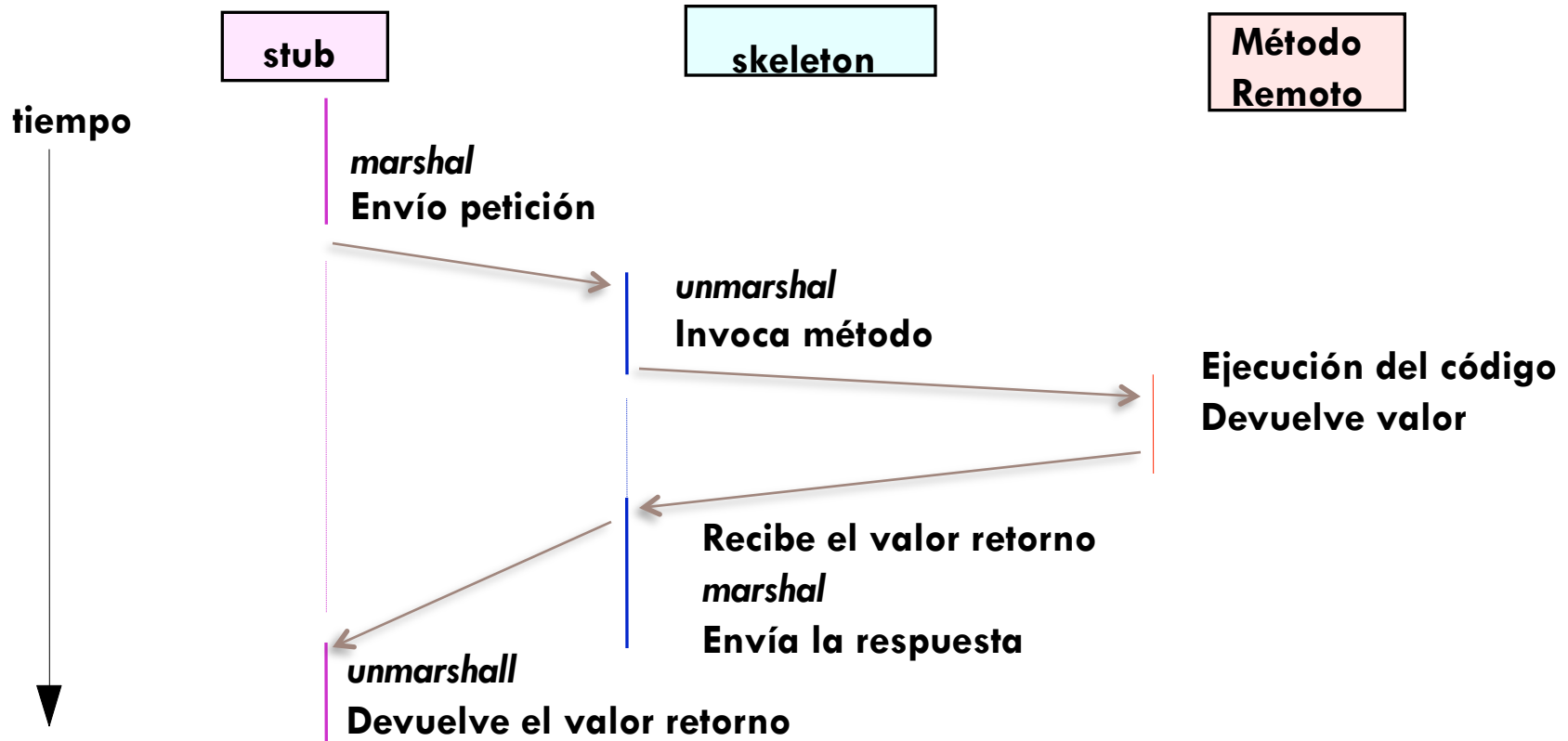
- ▶ Interpreta y gestiona las referencias a objetos remotos.
- ▶ Invoca operaciones de la capa de transporte.

▶ Nivel de transporte

- ▶ Se encarga de las comunicaciones y de establecer las conexiones necesarias.
- ▶ Basada en protocolo TCP.



Arquitectura de RMI



Contenidos

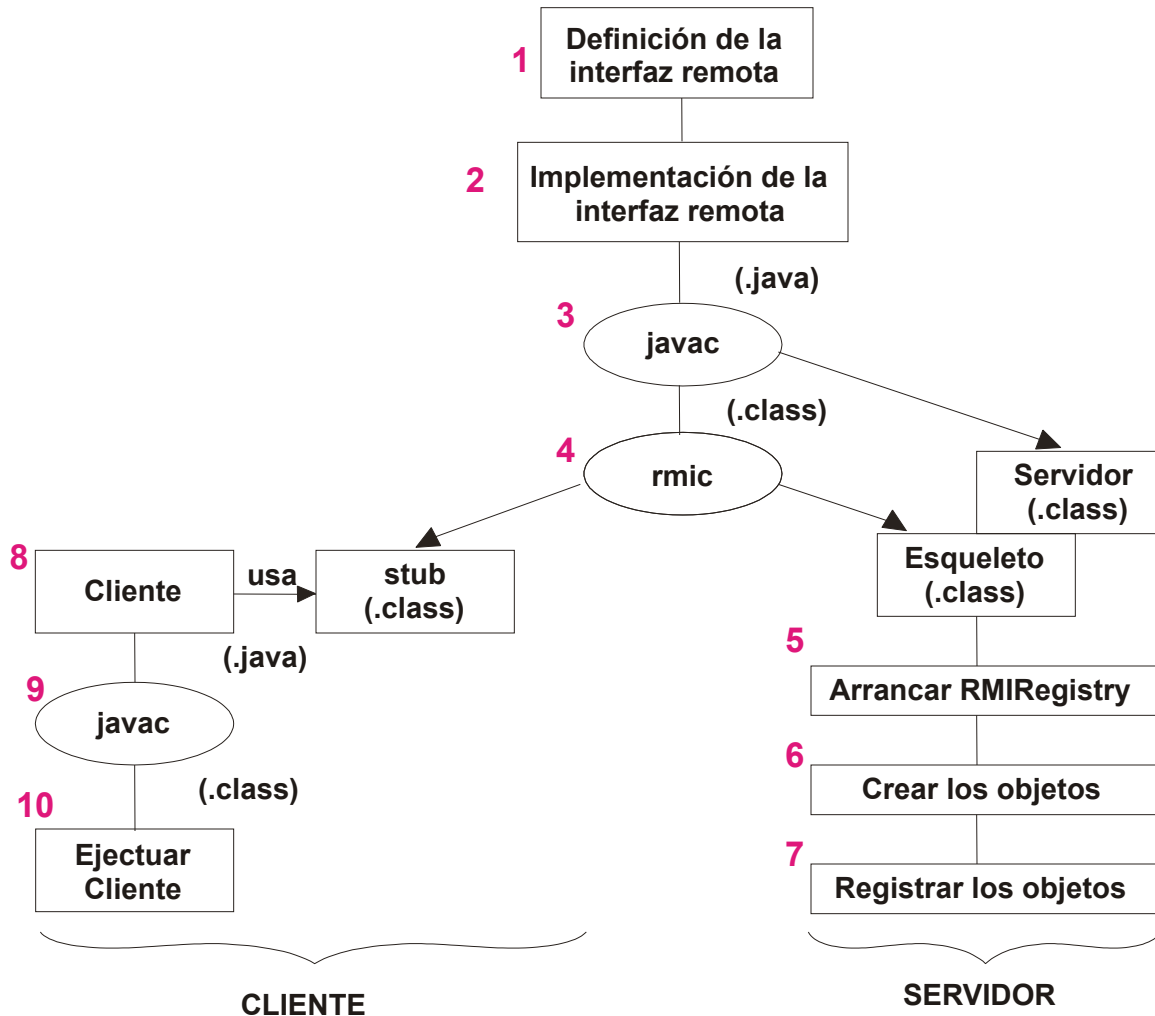
1. Introducción:

1. Paradigma de invocación remota de método
2. Entorno de programación Java

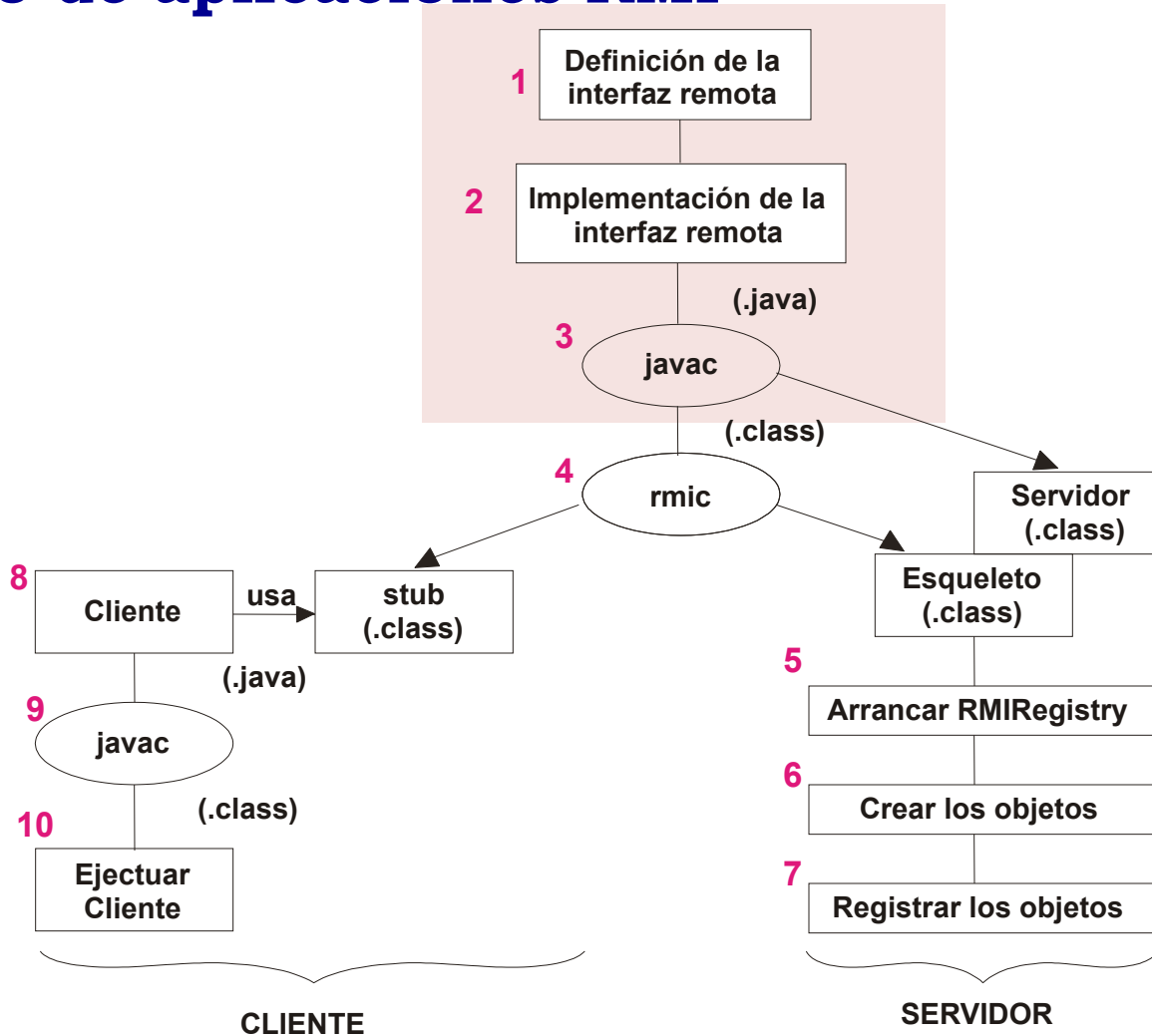
2. **Java RMI**

1. Introducción a RMI
2. Arquitectura de RMI
3. **Desarrollo de aplicaciones distribuidas**
 1. **Interfaz y despliegue**

Diseño de aplicaciones RMI



Diseño de aplicaciones RMI



Diseño de aplicaciones RMI

- ▶ Interfaz remota:
 - ▶ Clase que sirve de plantilla para otras clases.

```
import java.rmi.*;
public interface SomeInterface extends Remote {
    // Cabecera del primer método remoto
    public String someMethod1( )
        throws java.rmi.RemoteException;

    // Cabecera del segundo método remoto
    public int someMethod2( float parameter) throws
        java.rmi.RemoteException;
}
```

Diseño de aplicaciones RMI

- ▶ Implementación de la interfaz remota
 - ▶ Realizado por el **servidor**

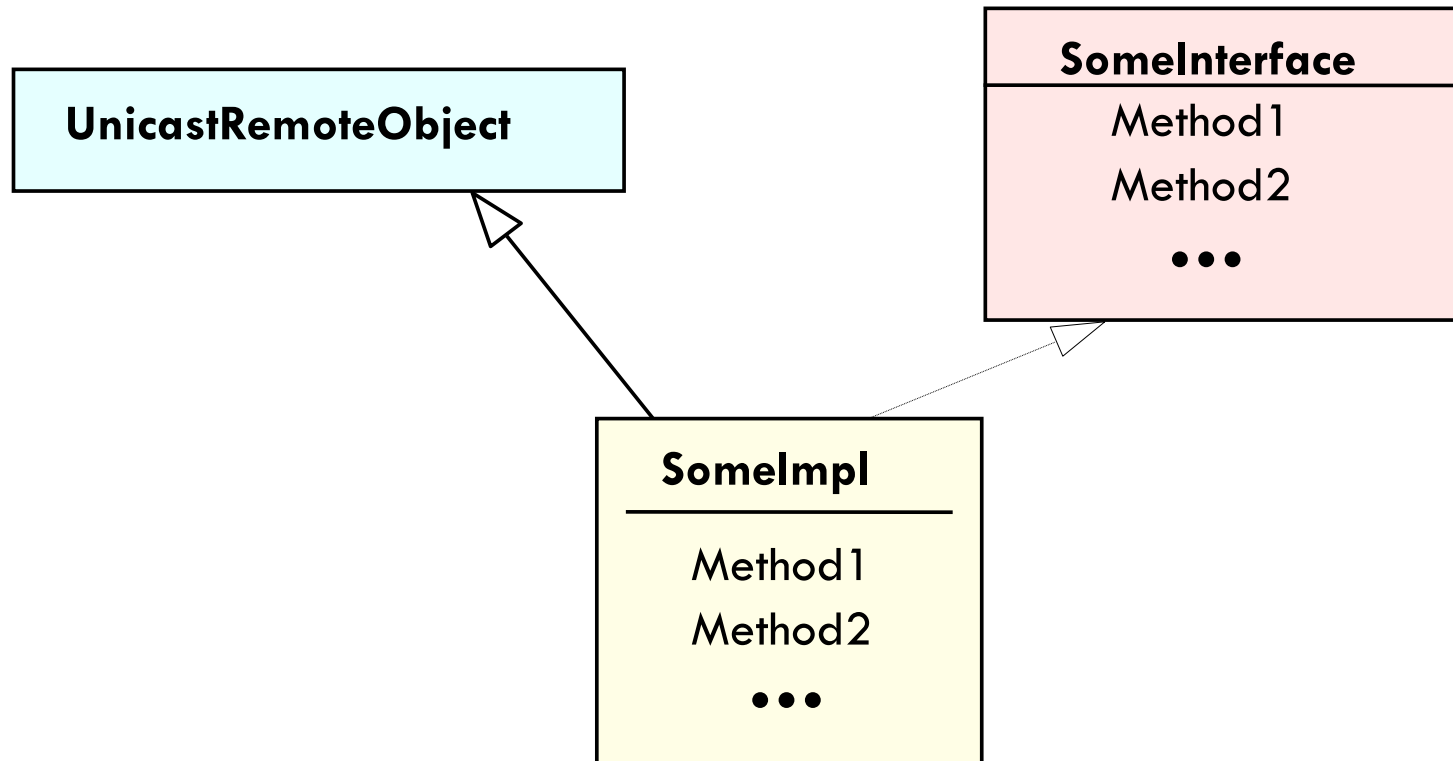
```
import java.rmi.*;
import java.rmi.server.*;

public class SomeImpl extends UnicastRemoteObject
    implements SomeInterface
{
    public SomeImpl() throws RemoteException { super( ); }

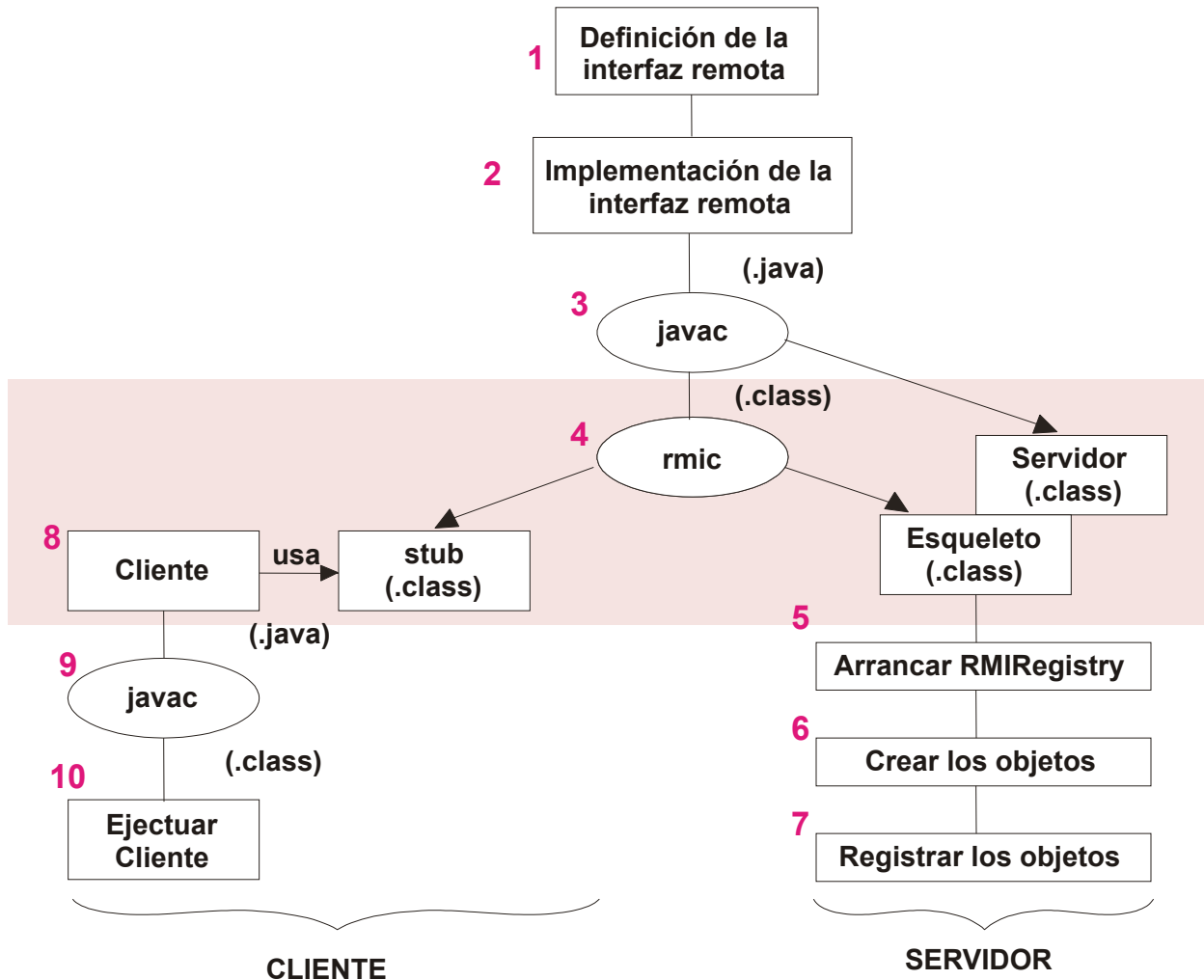
    public String someMethod1( ) throws RemoteException
    { /* Código fuente */ }

    public int someMethod2(float a) throws RemoteException
    { /* Código fuente */ }
}
```

Diseño de aplicaciones RMI



Diseño de aplicaciones RMI



Diseño de aplicaciones RMI

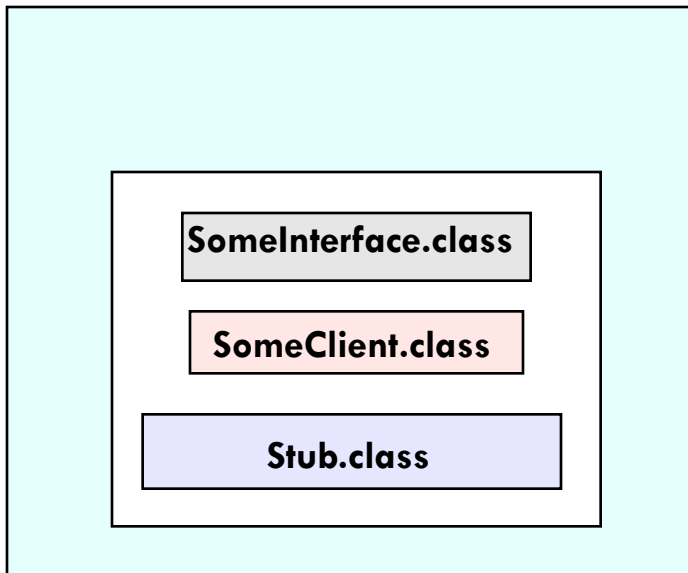
- ▶ Diseño por parte del servidor:
 - ▶ Implementación de la interfaz remota
 - ▶ Generar el resguardo y el esqueleto

nombre de la clase de la implementación de la interface remota

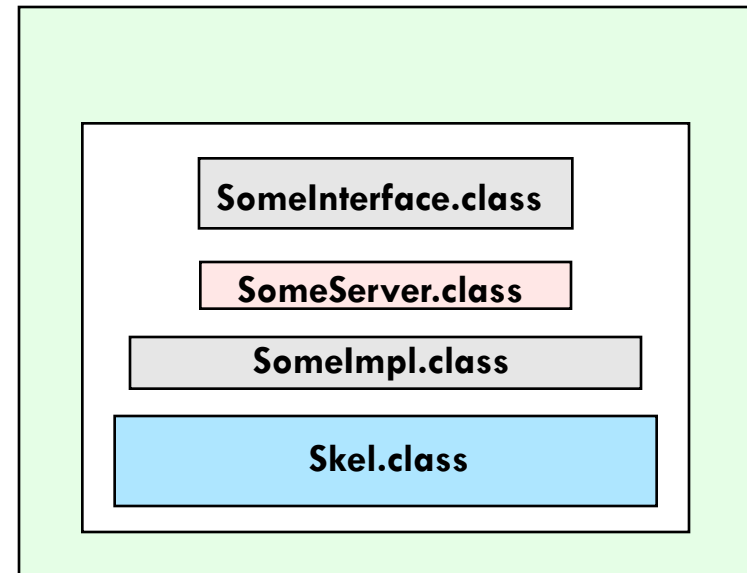
```
# rmic SomeImpl
# ls SomeImpl*.class
...
SomeImpl_skel.class
SomeImpl_stub.class
```

Invocación remota

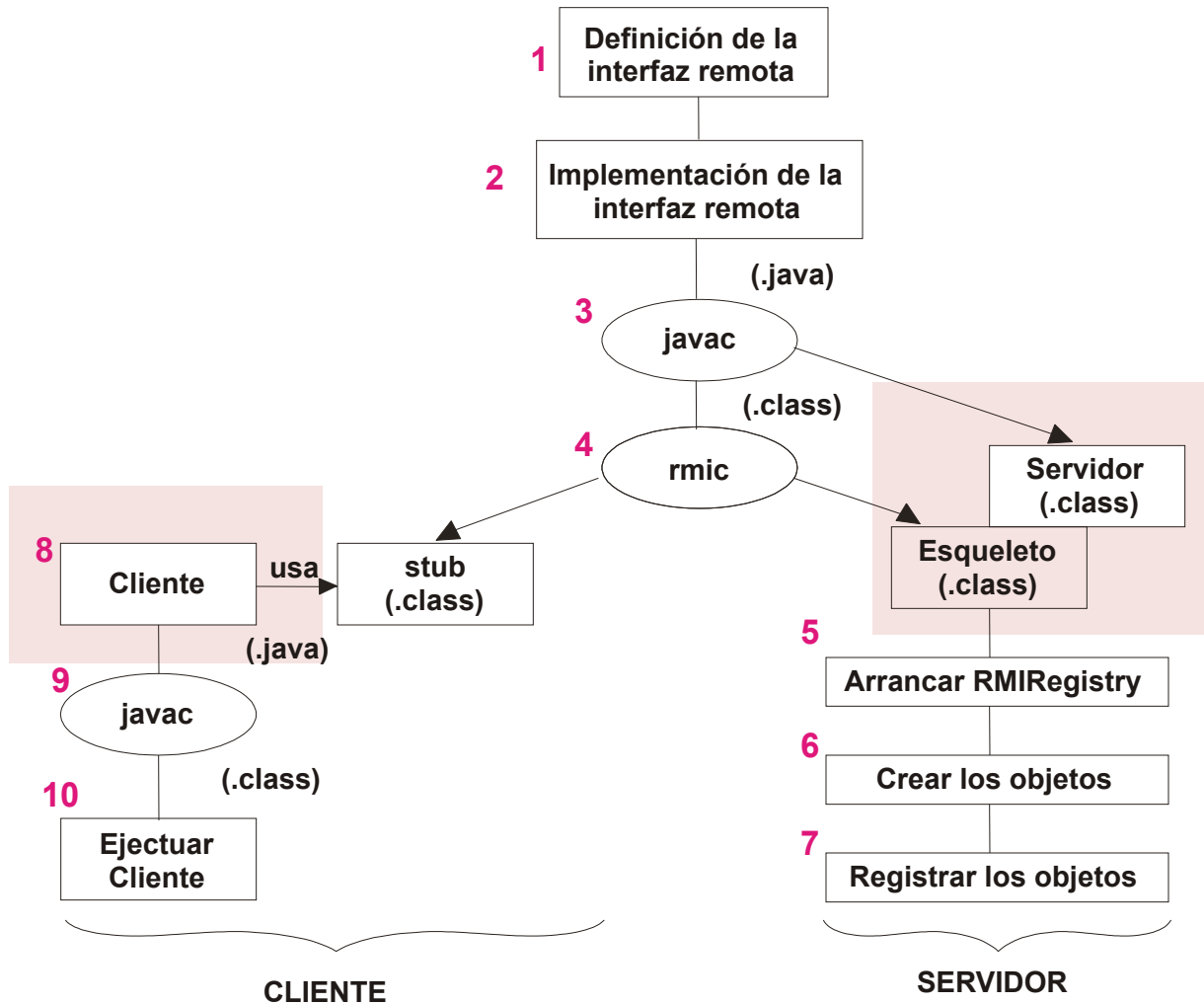
Cliente de objetos



Servidor de objetos



Diseño de aplicaciones RMI



Plantilla de clase de servidor de objeto

```
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
public class SomeServer {
    public static void main(String args[]) {
        try{
            SomeImpl exportedObj = new SomeImpl();

            int portNum=1099;
            startRegistry(portNum);

            registryURL = "rmi://localhost:"+portNum+"/some";
            Naming.rebind(registryURL, exportedObj);
            System.out.println("Some Server ready.");
        }
    }
    catch (Exception ex) {
        System.out.println("Exception: "+ex);
    }
}
```

Plantilla de clase de servidor de objeto

```
private static void startRegistry(int RMIPortNum)
    throws RemoteException
{
    try {
        Registry registry= LocateRegistry.getRegistry (RMIPortNum) ;
        registry.list( ) ;
    }
    catch (RemoteException ex)
    {
        System.out.println("RMI registry cannot be located at port" + RMIPortNum);
        Registry registry= LocateRegistry.createRegistry (RMIPortNum) ;
        System.out.println("RMI registry created at port " + RMIPortNum);
    }
}
```

Alternativa: activar el registro manualmente con
`rmiregistry <número de puerto>`

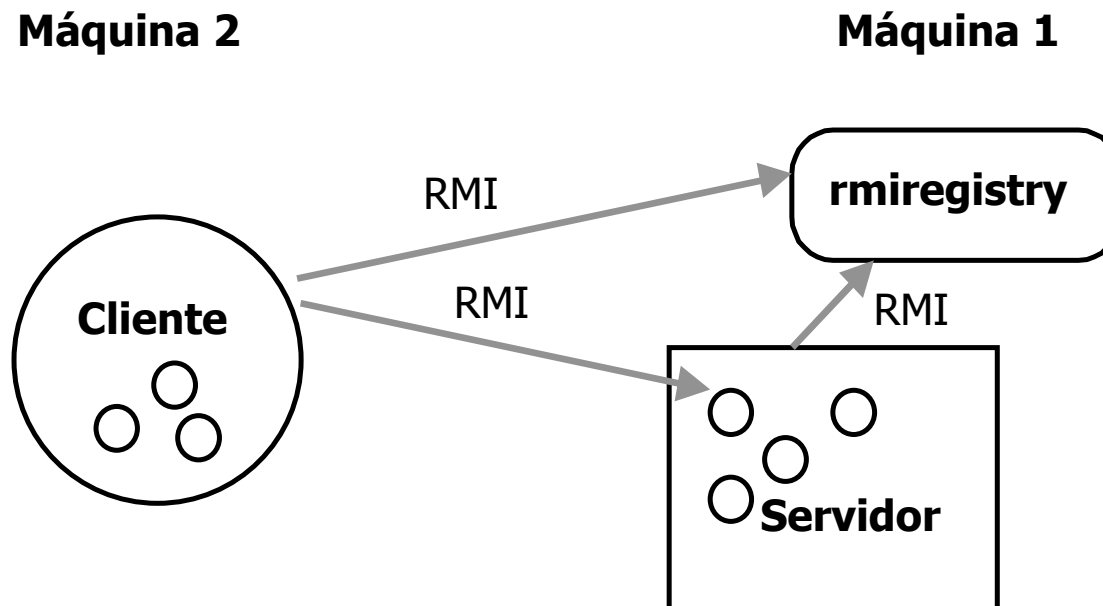
Plantilla de clase de cliente de objeto

```
import java.rmi.*;

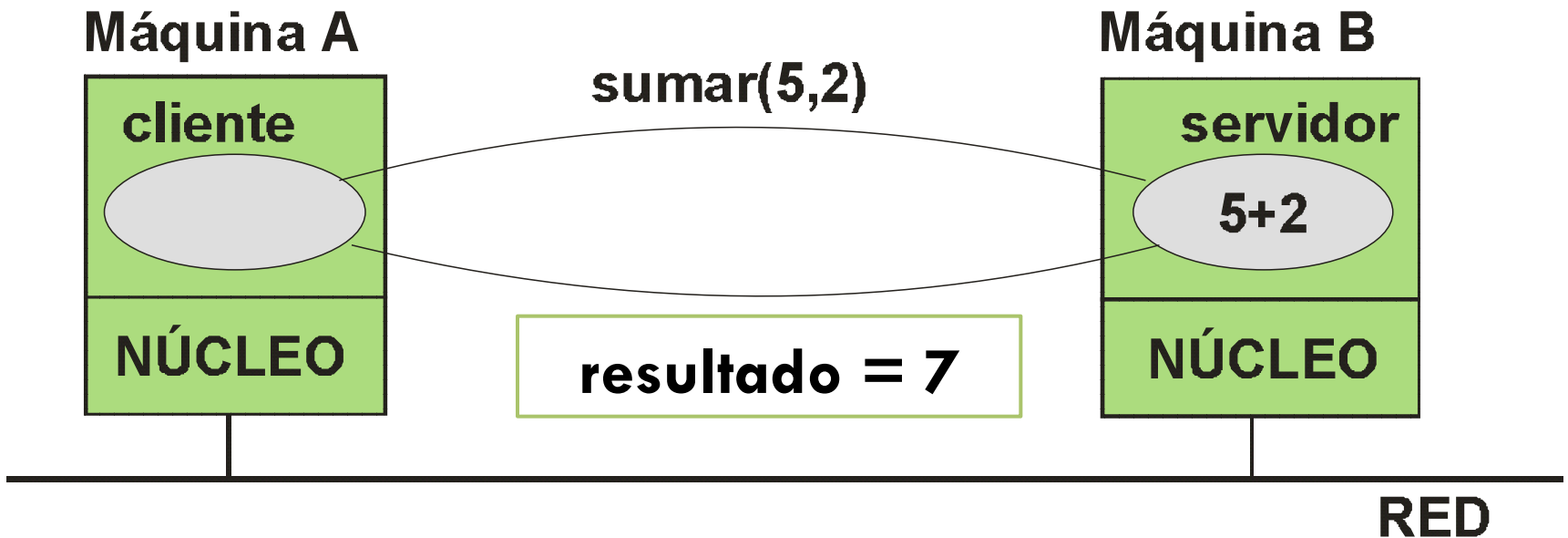
public class SomeClient
{
    public static void main(String args[])
    {
        try {
            int portNum=1099;
            String registryURL ="rmi://serverhost:" + portNum + "/some";
            SomeInterface h = (SomeInterface)Naming.lookup(registryURL);

            String message = h.someMethod1();
            System.out.println(message);
        }
        catch (Exception e) {
            System.out.println("Exception in SomeClient: " + e);
        }
    }
}
```

Invocación remota



Ejemplo (RMI)



Modelización de la interfaz remota (Sumador)

```
public interface Sumador
    extends java.rmi.Remote
{
    public int sumar(int a, int b)
        throws java.rmi.RemoteException;
}
```

Clase que implementa la interfaz (SumadorImpl)

```
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
public class SumadorImpl
    extends UnicastRemoteObject implements Sumador {
    public SumadorImpl(String name) throws RemoteException {
        super();
        try {
            System.out.println("Rebind objeto " + name);
            Naming.rebind(name, this);
        } catch (Exception e) {
            System.out.println("Exception: " + e.getMessage());
            e.printStackTrace();
        }
    }
    public int sumar (int a, int b) throws RemoteException
    { return a + b; }
}
```

Registro del servicio

- ▶ Cualquier programa que quiera instanciar un objeto de esta clase debe realizar el registro con el servicio de nombrado. Ejemplo:

Sumador misuma =

```
(Sumador) Naming.lookup("rmi://" + args[0] + "/" + "MiSumador");
```

- ▶ Antes de arrancar el cliente y el servidor, se debe arrancar el programa *rmiregistry* en el servidor para el servicio de nombres.



Código del servidor (SumadorServer)

```
import java.rmi.*;  
import java.rmi.server.*;
```

```
public class SumadorServer {  
    public static void main (String args[]) {  
        try{  
            SumadorImpl misuma = new  
                SumadorImpl ("rmi://localhost/MiSumador");  
        } catch(Exception excr) {  
            System.out.println("Excepcion: "+excr);  
        }  
    }  
}
```

```
import java.rmi.*;  
import java.rmi.server.UnicastRemoteobjeto;  
public class SumadorImpl  
    extends UnicastRemoteobjeto implements Sumador {  
    public SumadorImpl(String name)  
        throws RemoteException {  
        super();  
        try {  
            System.out.println("Rebind objeto " + name);  
            Naming.rebind(name, this);  
        }  
    }  
}
```

Código en el cliente (SumadorCliente)

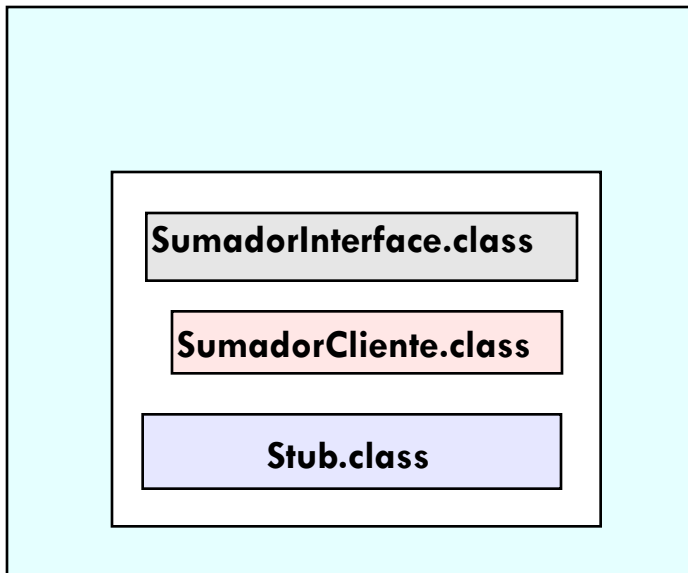
```
import java.rmi.*;
```

```
public class SumadorClient {  
    public static void main(String args[]) {  
        int res = 0;  
        try {  
            System.out.println("Buscando Objeto ");  
            Sumador misuma = (Sumador)Naming.lookup(  
                "rmi://" + args[0] + "/" + "MiSumador");  
            res = misuma.sumar(5, 2);  
            System.out.println("5 + 2 = " + res);  
        } catch (Exception e) {  
            System.err.println(" System exception");  
        }  
        System.exit(0);  
    }  
}
```

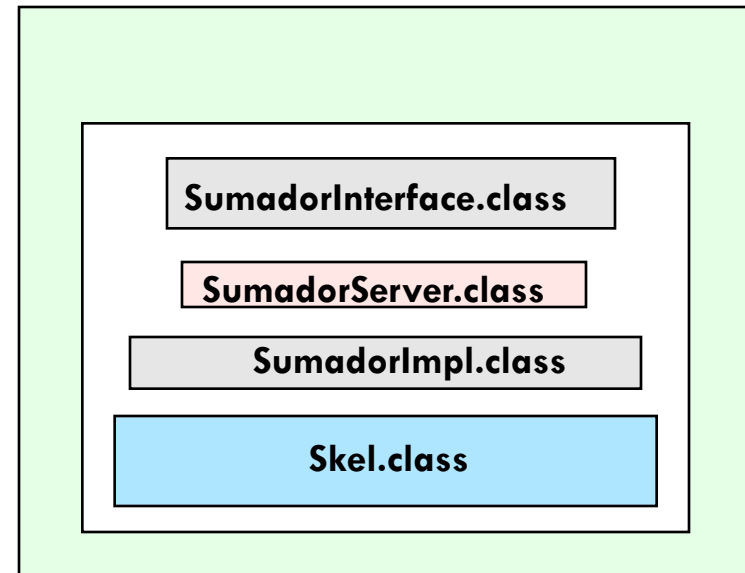
```
public interface Sumador  
    extends java.rmi.Remote  
{  
    public int sumar(int a, int b)  
        throws java.rmi.RemoteException;  
}
```

Invocación remota

Cliente de objetos



Servidor de objetos



¿Cómo se ejecuta?

1. Compilación

- ▶ javac Sumador.java
- ▶ javac SumadorImpl.java
- ▶ javac SumadorClient.java
- ▶ javac Sumador Server.java

2. Generación de los esqueletos

- ▶ rmic SumadorImpl

3. Copiar *SumadorImpl_Stub.class* e *interfaz remota* a clientes

4. Ejecución del programa de registro de RMI

- ▶ rmiregistry

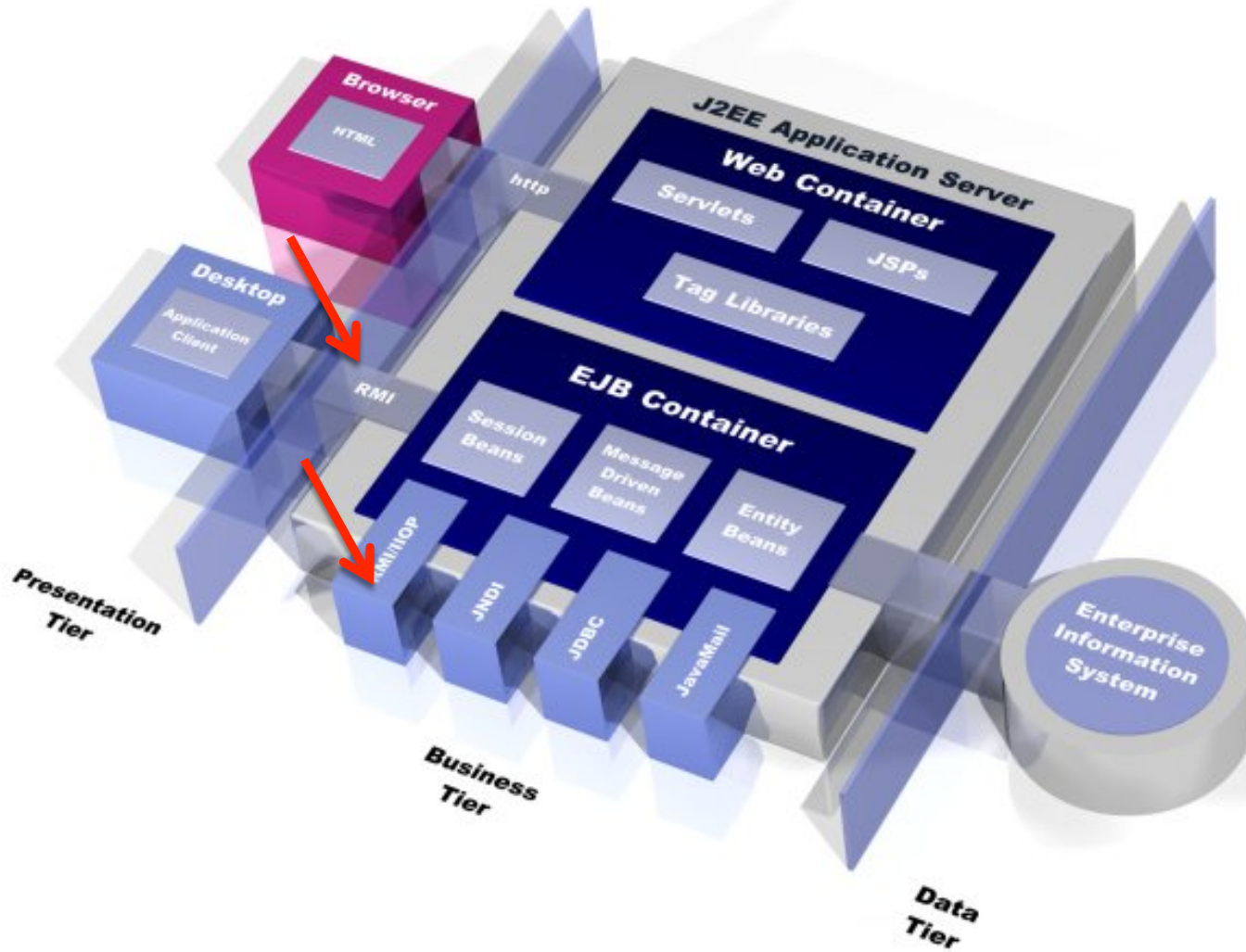
5. Ejecución del servidor

- ▶ java SumadorServer

6. Ejecución del cliente

- ▶ java SumadorCliente <host-del-servidor>

Entornos donde se usa Java RMI



Entornos donde se usa Java RMI

