

# 5.1 Java RMI

---

- Garcia Morales Rebeca
- Jimenez Anaya Erick
- Ramirez Vega Gerardo



# Introducción

---

- Nosotros el equipo 3 conformado por los integrantes García Morales Rebecca, Jiménez Anaya Erick y Ramirez Vega Gerardo les presentaremos el tema 5.1 de JAVA RMI referente a la unidad 3 de la materia de Aplicaciones para Comunicaciones de Red en el grupo 6CV1.
- En esta exposición enseñaremos de manera visual una presentación sobre qué es JAVA RMI.



# Java y RMI



Java, introducido por Sun Microsystems en 1995

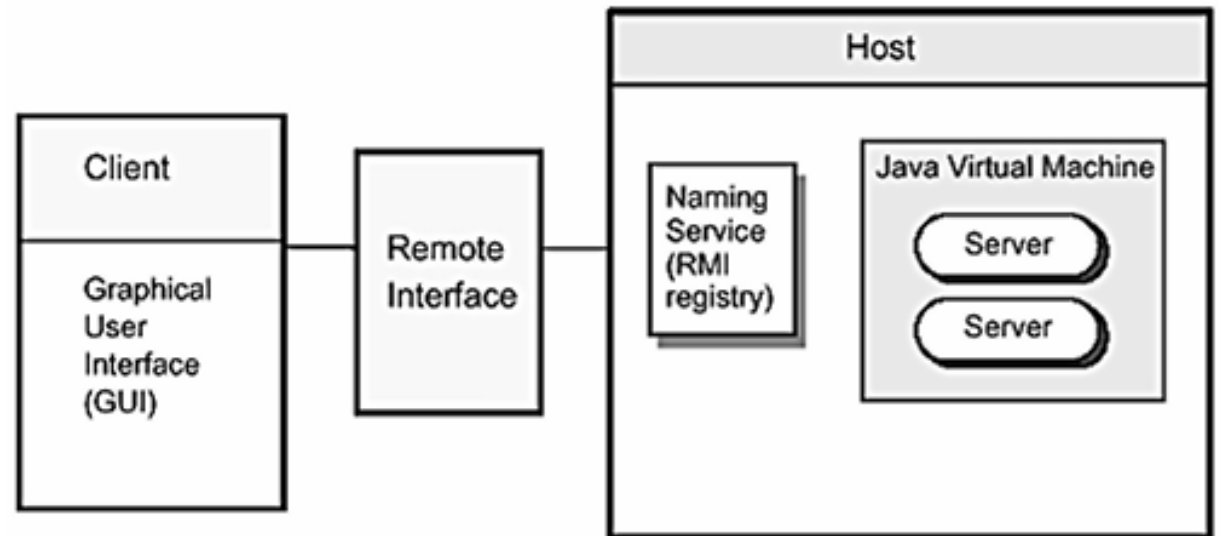


RMI añade a Java la capacidad de llamadas a procedimientos remotos (RPC) manteniendo su orientación a objetos, permitiendo la interacción entre objetos en diferentes JVMs, y se construye sobre la serialización de objetos y el soporte de red TCP/IP de Java.

# Arquitectura de Sistemas RMI

- Sistema RMI Simple:  
Configuración básica con  
una interfaz remota,  
cliente y uno o más  
servidores remotos.

**Figure 1.1. Simple RMI system**

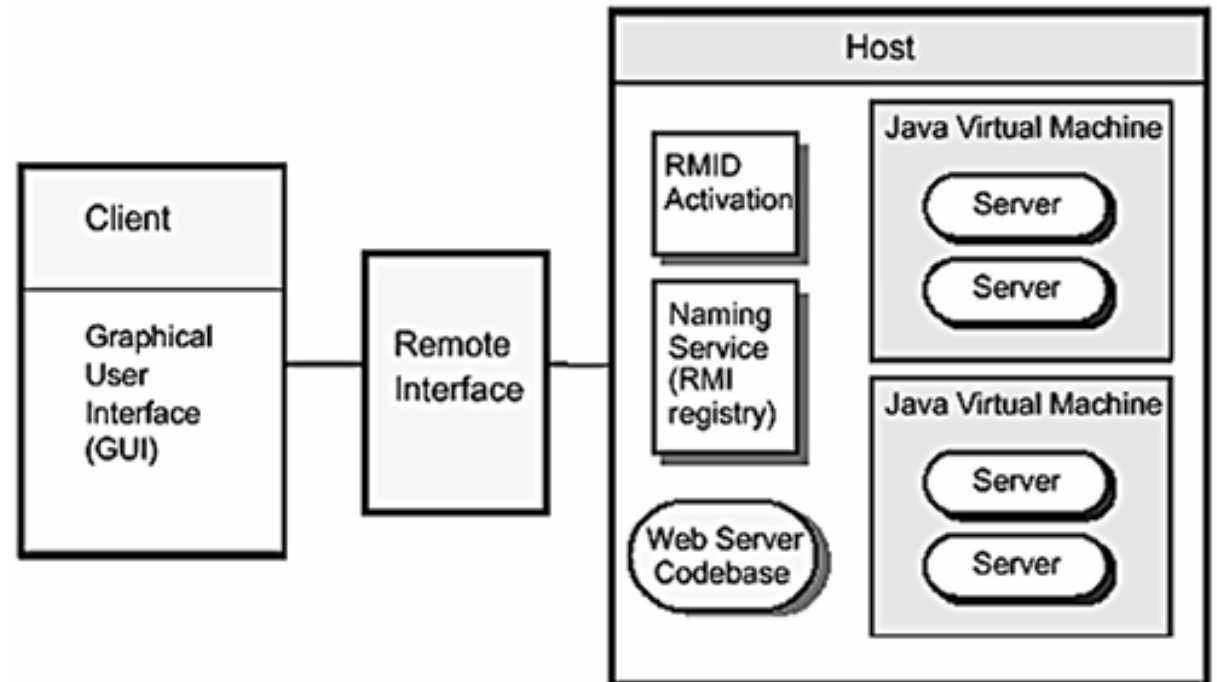


# Arquitectura de Sistemas RMI

- Sistema RMI Avanzado

Incluye activación RMI, que permite activar servidores bajo demanda, y un servidor web que proporciona un servicio de base de código RMI, una ubicación global para archivos de clase Java y archivos JAR accesibles para clientes y servidores RMI.

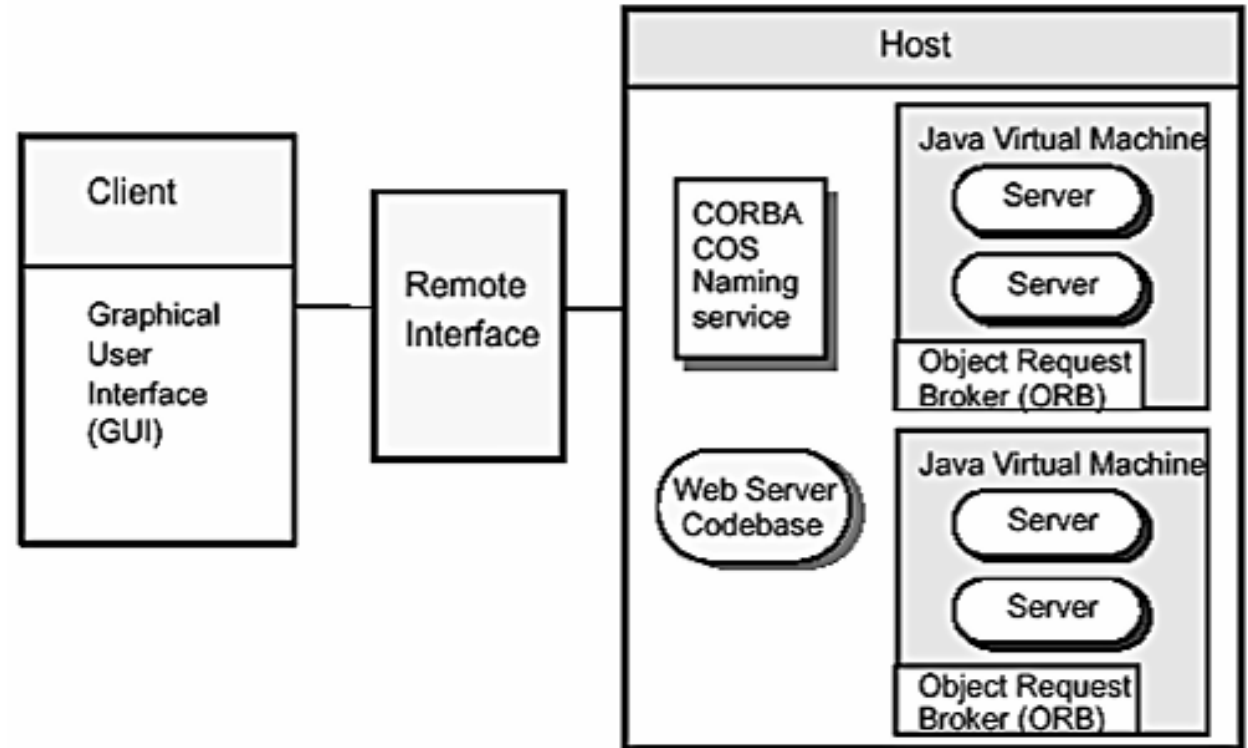
**Figure 1.2. Advanced RMI system showing codebase and activation**



# Arquitectura de Sistemas RMI

- Sistema RMI sobre IIOP
- Utiliza el servicio de nombres COS, accesible a través de la Interfaz de Directorios y Nombres de Java (JNDI), en lugar del registro RMI. No admite la activación RMI, solo soportada bajo el protocolo de métodos remotos de Java (JRMP). RMI simplifica las complejidades de la computación distribuida a una simple llamada de método y un manejador de excepciones en el cliente.

**Figure 1.3. RMI over IIOP**





# Invocación de métodos remotos y Excepciones remotas

La invocación de métodos remotos permite que un método en un objeto remoto sea invocado desde otra JVM a través de una interfaz remota que extiende `java.rmi.Remote` y debe lanzar `RemoteException`. Los objetos remotos deben implementar una interfaz remota y ser exportados al sistema RMI.

Las excepciones remotas se utilizan en RMI para manejar errores que ocurren durante la comunicación entre objetos en diferentes JVMs. Estos errores pueden incluir fallos de red, problemas de conexión, o problemas de serialización/deserialización de objetos.

- Sintaxis

La sintaxis es el conjunto de reglas que gobiernan la disposición de palabras.

- Semántica

La semántica es el conjunto de reglas que definen (a) reglas adicionales de compilación que no forman parte de la sintaxis y (b) cómo se ejecuta una declaración.

---

## Características de RMI



Local: Argumentos y resultados de tipo objeto se pasan por referencia, los de tipo primitivo por valor. Excepciones se manejan localmente. Objetos locales se recolectan mediante detección de ciclos de basura.

Remota: Argumentos y resultados de tipo objeto se pasan por copia profunda. Métodos remotos deben declarar RemoteException. Clientes manejan excepciones remotas. Objetos remotos se recolectan mediante conteo de referencias.

---

## Semántica

Lógica de negocio:  
Código que realiza  
tareas útiles.

Interfaz de usuario:  
Código para la  
aplicación cliente.

Marshalling y  
demarshalling:  
Conversión de datos y  
ejecución de métodos  
en otro proceso.

Lanzamiento y  
configuración:  
Inicialización y  
configuración de la  
aplicación.

Robustez y  
escalabilidad: Mejora la  
robustez y escalabilidad  
de la aplicación.

---

## Estructura Básica de RMI

# NOTA

- Las categorías tres y cuatro son diferentes ya que la mayoría de este código puede generarse automáticamente sin mucho esfuerzo del programador. RMI ya contiene o puede generar automáticamente la mayoría del código en estas categorías, lo cual es una razón convincente para usar RMI.



# Serialización

---

La serialización es el proceso de convertir un conjunto de instancias de objetos que contienen referencias entre sí en un flujo lineal de bytes. Este flujo puede enviarse a través de un socket, almacenarse en un archivo o manipularse como un flujo de datos.

RMI utiliza la serialización para pasar objetos entre JVMs, ya sea como argumentos en una invocación de método de un cliente a un servidor o como valores de retorno de una invocación de método.

## Marshalling y Unmarshalling.

- Para ejecutar un método remoto, RMI debe transmitir los argumentos del método del cliente al servidor y el resultado en dirección inversa. Este proceso se denomina marshalling (codificación) y unmarshalling (decodificación). RMI realiza estas tareas mediante la serialización de Java.

# Elementos Esenciales de la Serialización

Los objetos que se transmiten mediante marshalling y unmarshalling por RMI deben ser serializables.

Implementar la interfaz Serializable o Externalizable.

Tener un constructor accesible sin argumentos.

Cumplir con las reglas de serialización recursivamente o ser nulos, o ser declarados como estáticos o transitorios.


La interfaz Serializable no declara funciones miembro; es solo una interfaz de marcado.



# Registro de RMI

Figure 6.1. RMI registry

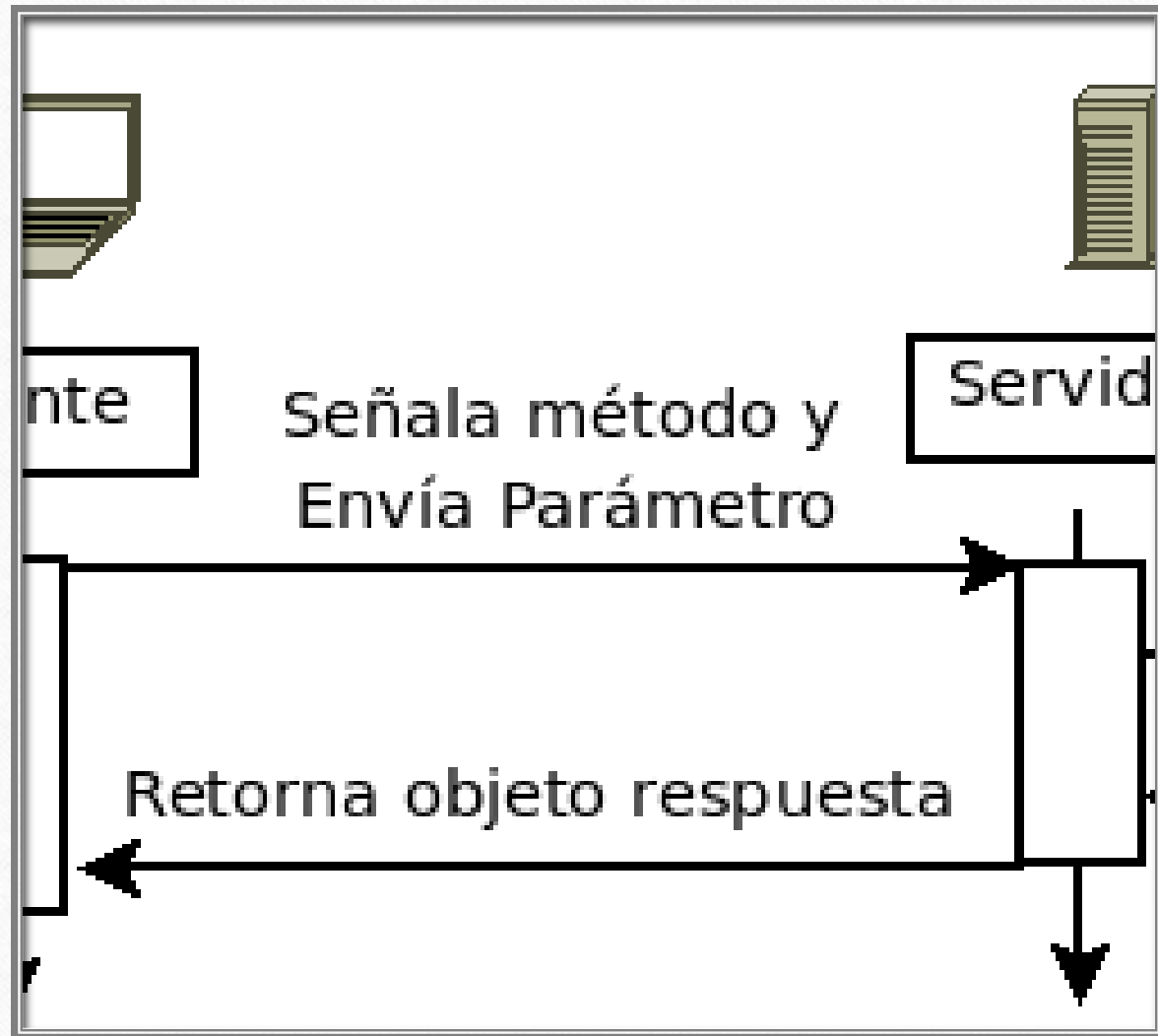
Name	Reference
Echo	●
DateTime	●
Login	●



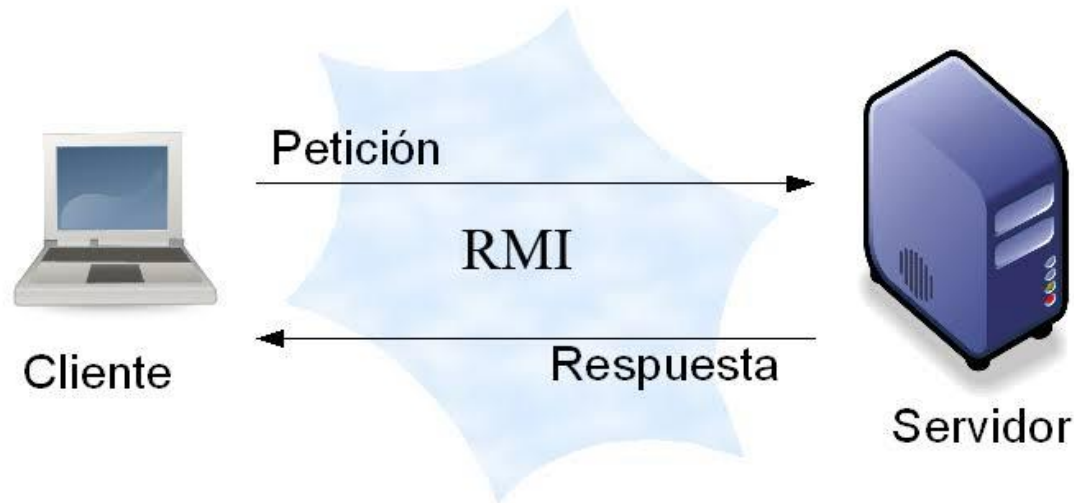
El registro RMI es un servicio de nomenclatura que proporciona a los clientes un mecanismo para encontrar uno o más servidores RMI iniciales.

# Como funciona:

- Un cliente primero busca en el registro RMI para encontrar uno o más servidores RMI iniciales.
- - Luego, obtiene servidores RMI subsiguientes a partir del conjunto de servidores RMI ya encontrados.



# Operaciones esenciales

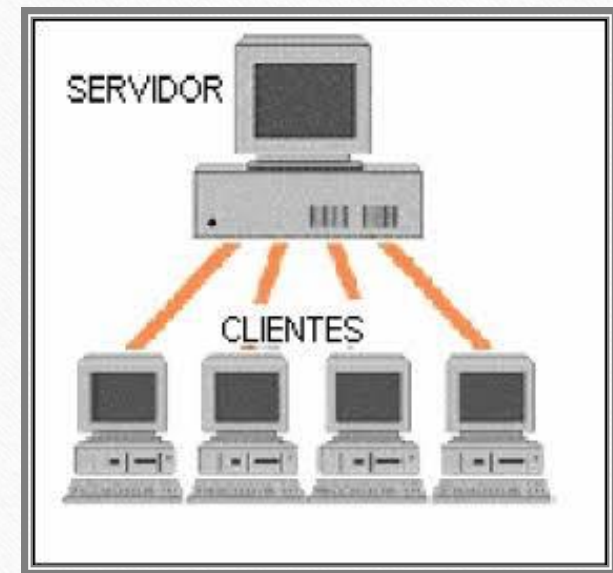


- Bind: Añade una entrada (nombre de servicio/dirección) al registro.
- - Unbind: Elimina la entrada de un servicio del registro por nombre.
- - Lookup: Permite a cualquiera usar el nombre del servicio para encontrar la dirección del servicio.



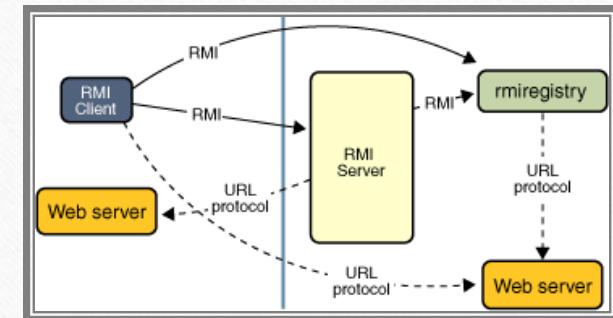
# Operaciones esenciales

- - Rebind: Similar a bind, pero sobrescribe el enlace si el nombre especificado ya está en uso.
- - List: Lista todos los nombres actualmente vinculados en el registro.

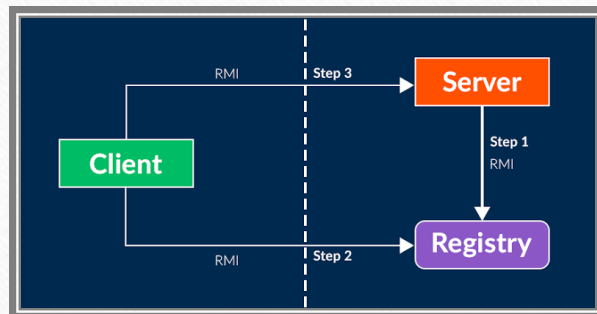


# Nombres en el registro

- - Los nombres a los que se vincula un servidor RMI son cadenas arbitrarias y pueden contener cualquier carácter sin interpretación jerárquica.
- - Se recomienda utilizar un nombre único, como un nombre cualificado de Java, para evitar conflictos con otros proveedores de software.



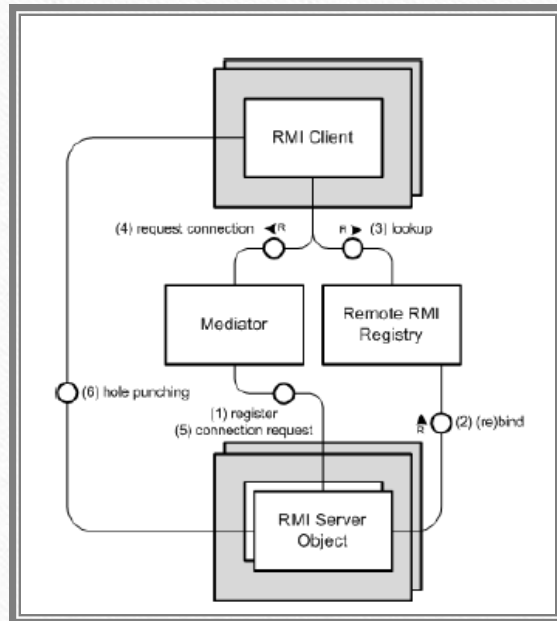
# La clase de nomenclatura



- La clase `java.rmi.Naming` proporciona la forma más simple de acceder a un registro remoto utilizando URLs.
- - La interfaz `java.rmi.registry.Registry` proporciona un control de nivel más bajo, permitiendo la obtención de una instancia del registro y la ejecución de operaciones bind, lookup, list, rebind, y unbind.



# Creación de un registro local



- Es posible crear una instancia del registro en la misma máquina virtual que su aplicación mediante `LocateRegistry.createRegistry`.
- Esto puede ser útil para configuraciones específicas de la aplicación, compartir el entorno Java de los servidores RMI, o tener el registro cerrado junto con la aplicación.

```
¿Está registrado en la Tienda Online? (s/n): s
*****
          INICIO DE SESIÓN
*****
Introduzca su usuario: a
Introduzca su contraseña: 
Bienvenid@ a
*****
          TIENDA ONLINE
*****
1 Gestión de productos
2 Gestión de usuarios
0 Salir

Introduzca una opción y pulse INTRO: █
```

# Despliegue

---

- - Se requiere al menos un registro RMI por host en el que se ejecuten servidores RMI.
- - El registro solo permite llamadas bind, rebind y unbind que provengan del mismo host, por razones de seguridad.

**Example 6.1. Program to list the registry**

Code View: Scroll / [Show All](#)

```
import java.rmi.*;
import java.util.*;
/**
 ** List the names and objects bound in RMI registries.
 ** Invoke with zero or more registry URLs as command-line arguments,
 ** e.g. "rmi://localhost", "rmi://localhost:1099".
 */
public class ListRegistry
{
    public static void main(String[] args)
    {
        System.setSecurityManager(new RMISecurityManager());
        for (int i = 0; i < args.length; i++)
        {
            try
            {
                String[]list = Naming.list(args[i]);
                System.out.println("Contents of registry at "+args[i]);
                for (int j = 0; j < list.length; j++)
                {
                    Remote remote = Naming.lookup(list[j]);
                    System.out.println((j+1)+".\tname="+list[j]
                                     +"\n\tremote="+remote);
                }
            }
            catch (java.net.MalformedURLException e)
            {
                System.err.println(e); // bad argument
            }
            catch (NotBoundException e)
            {
                // name vanished between list and lookup - ignore
            }
            catch (RemoteException e)
            {
                System.err.println(e); // General RMI exception
            }
        }
    }
}
```





# Conclusión

---

- Como conclusión general, se debe tener en cuenta el uso de RMI para aplicaciones tipo cliente/servidor a mediana escala, en donde los clientes están muy distribuidos y sea difícil la diversificación de los códigos o el trabajo en equipo.
- Además de la posibilidad de separar el trabajo, encapsulando en aplicaciones clientes servidor, en donde el cliente y el servidor se desligan a tal grado que no deben saber nada del trabajo del uno del otro. Esto es sumamente útil en los tiempos actuales en donde las aplicaciones cliente servidor son cada vez más abundantes y la complejidad de ellas hace fundamental la separación del código en módulos independientes

# Referencias

---

- JAVa RMI: Remote Method invocation (First Edition). (1998). John Wiley & Sons Inc.
- Java RMI (First Edition). (2001). Publisher: O'Reilly.