Imagen que contiene Icono

El contenido generado por IA puede ser incorrecto.Dibujo en blanco y negro

El contenido generado por IA puede ser incorrecto.

Instituto Politécnico Nacional

Escuela Superior de Cómputo

Ingeniería en Sistemas Computacionales

Aplicaciones para comunicaciones en red

Academia “Sistemas Distribuidos”

Plan 2020

Práctica 25:

“**Java RMI**”

2015090269

González González Armando Omar

Profesor: Ojeda Santillan Rodrigo

Contenido

[Objetivo 3](#_Toc201734807)

[Introducción 3](#_Toc201734808)

[Desarrollo 4](#_Toc201734809)

[3.1 Interfaz Remota (Calculadora.java) 4](#_Toc201734810)

[3.2 Implementación del Servidor (Server.java) 4](#_Toc201734811)

[3.3 Implementación del Cliente (Client.java) 6](#_Toc201734812)

[Conclusión 9](#_Toc201734813)

[Pregunta 9](#_Toc201734814)

[¿Cómo podrían generar un negocio a través de lo visto en la práctica? 9](#_Toc201734815)

[Bibliografía 9](#_Toc201734816)

## Objetivo

Implementar una aplicación cliente-servidor distribuida utilizando la API de Java RMI (Remote Method Invocation). Se desarrollará una calculadora básica para demostrar cómo un cliente puede invocar métodos en un objeto que se ejecuta en una máquina virtual Java remota, comprendiendo el rol del registro RMI (rmiregistry) como intermediario.

## Introducción

Java RMI (Remote Method Invocation) es una API de Java que permite a un objeto invocar métodos de otro objeto que se encuentra en una Máquina Virtual de Java (JVM) distinta. Esta tecnología es un pilar en el desarrollo de aplicaciones distribuidas, ya que permite que componentes de software se comuniquen a través de una red de manera transparente.

La arquitectura de RMI se compone de varios elementos clave:

* **Interfaz Remota:** Define los métodos que un cliente puede invocar remotamente. Debe extender la interfaz java.rmi.Remote.
* **Implementación del Servidor:** Es la clase que contiene la lógica de los métodos declarados en la interfaz remota.
* **Stub (Proxy del cliente):** Un objeto del lado del cliente que actúa como representante del objeto remoto. Para el cliente, invocar un método en el stub es como hacerlo en el objeto local.
* **Registro RMI (rmiregistry):** Un servicio de nombres donde el servidor registra sus objetos remotos para que los clientes puedan buscarlos y utilizarlos.
* **Cliente RMI:** El programa que busca el objeto remoto en el registro, obtiene el stub y ejecuta sus métodos.

El flujo de operación consiste en que el servidor crea una instancia de la implementación y la registra con un nombre único en el rmiregistry. Posteriormente, el cliente busca ese objeto por su nombre en el registro y recibe el stub. A partir de ese momento, todas las llamadas a métodos que el cliente realiza sobre el stub son enviadas a través de la red hacia el objeto remoto en el servidor, el cual procesa la petición y devuelve un resultado.

## Desarrollo

Para la práctica, se desarrolló una calculadora capaz de realizar operaciones de suma, resta, multiplicación y división de forma remota. Para ello, se crearon tres clases principales en Java.

### **3.1 Interfaz Remota (**Calculadora.java**)**

Se define la interfaz Calculadora, que especifica los métodos que estarán disponibles para ser invocados remotamente. Esta interfaz extiende de java.rmi.Remote, un requisito indispensable para que sus métodos puedan ser llamados a través de la red. Cada método declara que puede lanzar una RemoteException para manejar posibles errores de comunicación.

import java.rmi.Remote;

import java.rmi.RemoteException;

public interface Calculadora extends Remote {

int sumar(int a, int b) throws RemoteException;

int restar(int a, int b) throws RemoteException;

int multiplicar(int a, int b) throws RemoteException;

double dividir(int a, int b) throws RemoteException;

}

### **3.2 Implementación del Servidor (**Server.java**)**

La clase Server contiene la lógica de las operaciones. Extiende de UnicastRemoteObject para poder funcionar como un objeto remoto que puede recibir llamadas. Además, implementa la interfaz Calculadora que definimos previamente.

En su método main, el servidor crea una instancia de sí mismo y la registra en el rmiregistry en el puerto 5555 bajo el nombre "ServicioCalculadora", quedando a la espera de peticiones de los clientes.

import java.rmi.RemoteException;

import java.rmi.registry.LocateRegistry;

import java.rmi.registry.Registry;

import java.rmi.server.UnicastRemoteObject;

public class Server extends UnicastRemoteObject implements Calculadora {

public Server() throws RemoteException {

super(); // Calls UnicastRemoteObject constructor which may throw RemoteException

}

public static void main(String[] args) {

try {

Server servidor = new Server();

Registry registry = LocateRegistry.createRegistry(5555);

registry.rebind("ServicioCalculadora", servidor);

System.out.println("Servidor de Calculadora en funcionamiento...");

} catch (Exception e) {

e.printStackTrace();

}

}

@Override

public int sumar(int a, int b) throws RemoteException {

return a + b;

}

@Override

public int restar(int a, int b) throws RemoteException {

return a - b;

}

@Override

public int multiplicar(int a, int b) throws RemoteException {

return a \* b;

}

@Override

public double dividir(int a, int b) throws RemoteException {

if (b == 0) {

throw new RemoteException("No se puede dividir entre cero");

}

return (double) a / b;

}

}

### **3.3 Implementación del Cliente (**Client.java**)**

El cliente se encarga de buscar el servicio remoto en el rmiregistry, interactuar con el usuario para obtener los números y, finalmente, invocar los métodos remotos para mostrar los resultados.

Utiliza LocateRegistry.getRegistry("127.0.0.1", 5555) para conectarse al registro y registry.lookup("ServicioCalculadora") para obtener una referencia (el stub) al objeto remoto.

import java.rmi.registry.LocateRegistry;

import java.rmi.registry.Registry;

import java.util.Scanner;

public class Client {

public static void main(String[] args) {

try {

Registry registry = LocateRegistry.getRegistry("127.0.0.1", 5555);

Calculadora calculadora = (Calculadora) registry.lookup("ServicioCalculadora");

Scanner scanner = new Scanner(System.in);

System.out.print("Ingrese el primer número: ");

int a = scanner.nextInt();

System.out.print("Ingrese el segundo número: ");

int b = scanner.nextInt();

System.out.println("Suma: " + calculadora.sumar(a, b));

System.out.println("Resta: " + calculadora.restar(a, b));

System.out.println("Multiplicación: " + calculadora.multiplicar(a, b));

System.out.println("División: " + calculadora.dividir(a, b));

scanner.close();

} catch (Exception e) {

e.printStackTrace();

}

}

}

A screenshot of a computer

AI-generated content may be incorrect.  
Figura 1. Ejecución de rmiregistry.

A computer screen shot of a program

AI-generated content may be incorrect.  
Figura 2. Ejecución del Server

A screenshot of a computer program

AI-generated content may be incorrect.  
Figura 3. Ejecución del Client

## Conclusión

La práctica se completó con éxito, cumpliendo el objetivo de implementar una aplicación distribuida con Java RMI. Se pudo comprobar el flujo de comunicación completo: el servidor registrando un objeto, y el cliente buscándolo e invocando sus métodos como si fueran locales.

Se concluye que Java RMI es una herramienta poderosa y relativamente sencilla para la creación de sistemas distribuidos en el ecosistema de Java. Abstrae gran parte de la complejidad de la comunicación en red, permitiendo al desarrollador enfocarse en la lógica de negocio. La clara separación entre la interfaz y la implementación facilita el mantenimiento y la escalabilidad de las aplicaciones, un principio fundamental en la ingeniería de software moderna.

## Pregunta

### ¿Cómo podrían generar un negocio a través de lo visto en la práctica?

El modelo RMI es la base del paradigma de "Software como Servicio" (SaaS). En lugar de una calculadora, se podría desarrollar un servidor con capacidades de cómputo mucho más complejas y especializadas (ej. análisis financiero, renderizado de video, modelos de inteligencia artificial). Se podría comercializar el acceso a esta API, cobrando a otras empresas por cada llamada (pago por uso) o mediante una suscripción mensual. Esto les permitiría a los clientes acceder a poder de cómputo avanzado sin tener que invertir en la costosa infraestructura y el desarrollo del software subyacente.

## Bibliografía

[1] H. Schildt, Java: The Complete Reference, Twelfth Edition. New York: McGraw-Hill Education, 2021.

[2] G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair, Distributed Systems: Concepts and Design, 5th Edition. Boston: Addison-Wesley, 2011.

[3] Oracle, "An Overview of RMI Applications," Java Platform, Standard Edition Documentation. [En línea]. Disponible: <https://docs.oracle.com/javase/tutorial/rmi/overview.html>. [Consultado: 25-jun-2025].