


TEMA 5.3 IMPLEMENTACIÓN DE STUB

Integrantes

Hernández Hernández Roberto Isaac

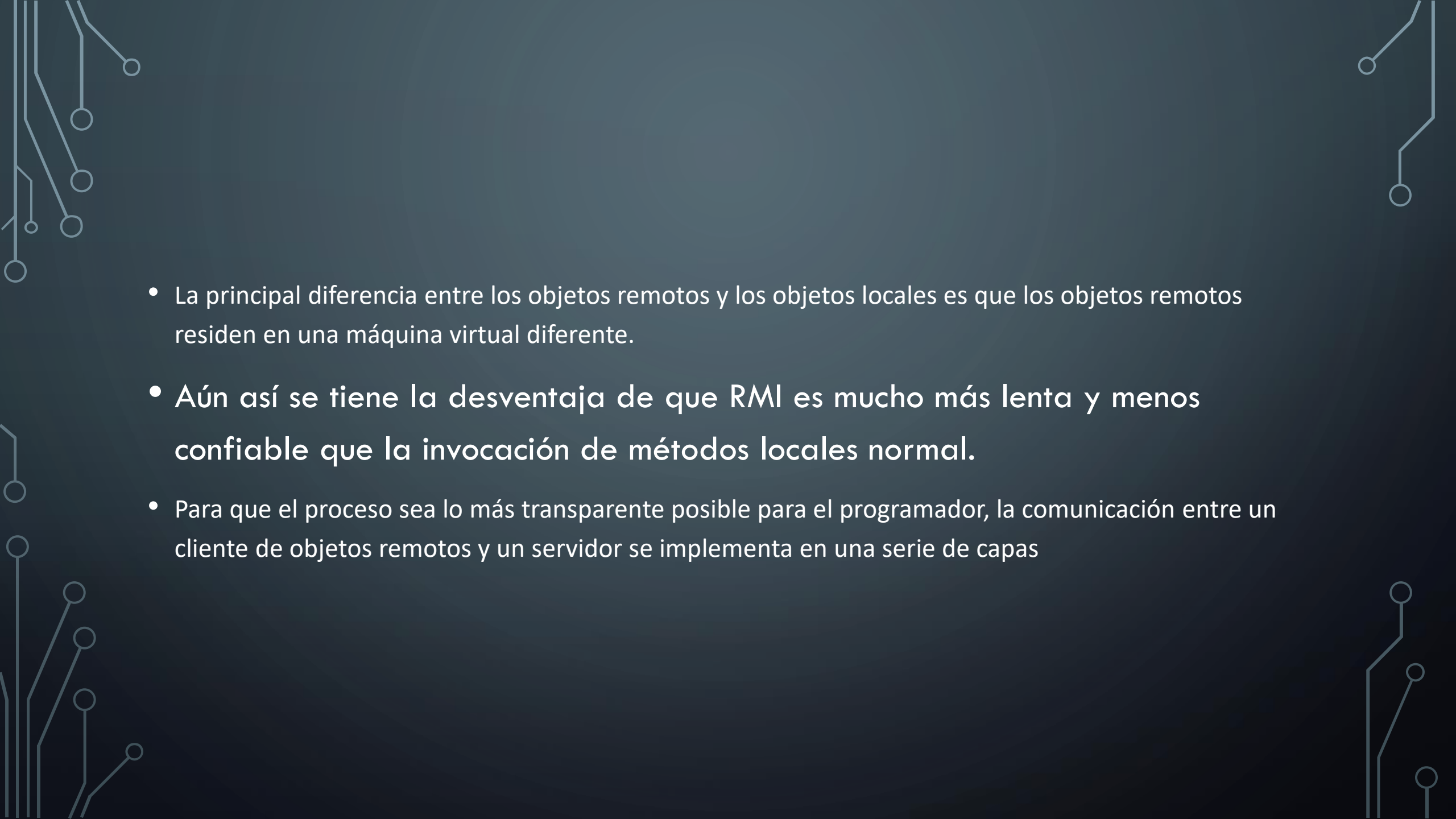
Pérez Ortiz Saúl

Ramirez Godinez Jesus Ernesto

The slide features a dark blue background with white decorative circuit-like lines. These lines, consisting of straight segments and small circles at junctions, are positioned along the left and right edges of the slide, framing the central text.

INTRODUCCIÓN: MÉTODO DE INVOCACIÓN REMOTA (RMI)

Con esta implementación, algunas partes de un programa en Java se ejecutan en una computadora local mientras que algunas otras se ejecutan en un host remoto. RMI crea la ilusión de que este programa distribuido se ejecuta en un sistema con un espacio de memoria que contiene todos los códigos y datos utilizados a ambos lados de la conexión física real.

- 
- The slide features a dark blue background with decorative white circuit-like lines in the corners. These lines consist of straight segments and small circles, resembling a stylized electronic circuit or network diagram.
- La principal diferencia entre los objetos remotos y los objetos locales es que los objetos remotos residen en una máquina virtual diferente.
 - Aún así se tiene la desventaja de que RMI es mucho más lenta y menos confiable que la invocación de métodos locales normal.
 - Para que el proceso sea lo más transparente posible para el programador, la comunicación entre un cliente de objetos remotos y un servidor se implementa en una serie de capas

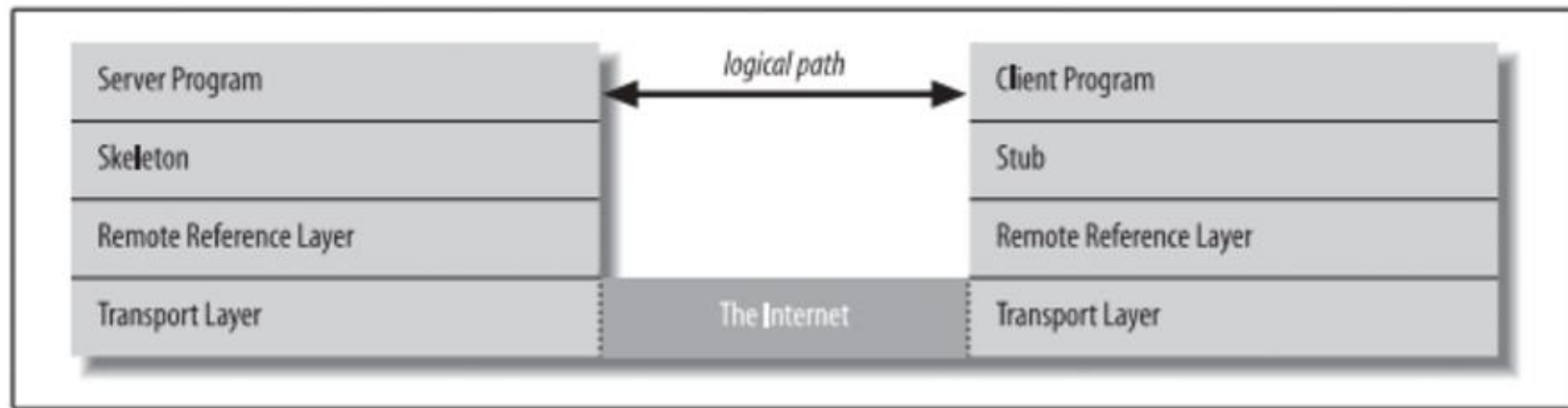
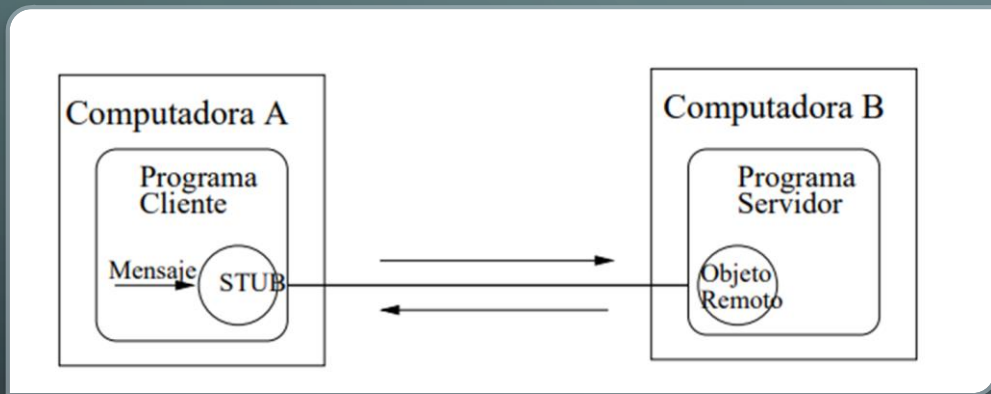


Figure 18-1. *The RMI layer model*

¿QUÉ ES STUB?

Un "stub" es un objeto local que actúa como una representación del objeto remoto en el cliente. Cuando el cliente invoca un método en el objeto remoto, en realidad está llamando a un método equivalente en el stub local. El stub se encarga de pasar la llamada al objeto remoto a través de la red

PROCESO (STUB)

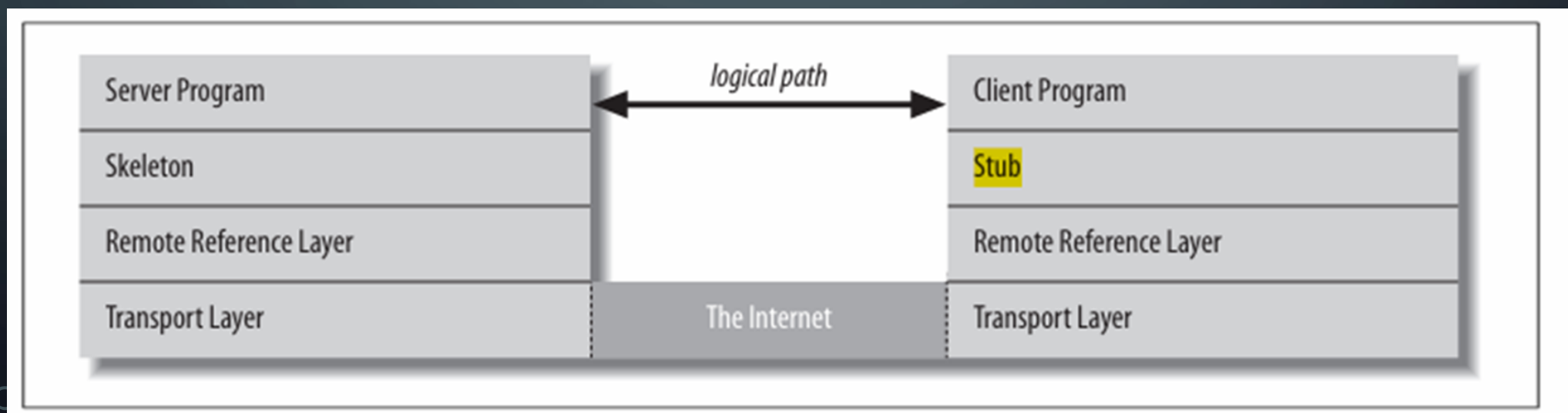


Cuando la aplicación cliente envía un mensaje al stub local del objeto remoto, la petición se transmite a la máquina que contiene al objeto real, donde el método es invocado y cualquier resultado retornado al stub local, de modo que la aplicación cliente puede obtener la respuesta apropiada

PROCESO (STUB)

En esencia, en lugar de crear un objeto, el programador liga el objeto remoto con un representante local, conocido como stub. Los mensajes dirigidos al objeto remoto se envían al stub local, como si fuera el objeto real. El stub acepta los mensajes que se le envían, y a su vez, los envía al objeto remoto, el cual invoca sus métodos apropiados. El resultado de la invocación de los métodos en el objeto remoto se envía de regreso al stub local, que los remite al emisor original de la llamada

El cliente cree que está comunicándose directamente con el servidor. Sin embargo, la comunicación real se lleva a cabo de manera más compleja: el cliente se comunica con el stub local, que a su vez interactúa con la capa de referencia remota. Esta capa se encarga de manejar la comunicación con el servidor remoto a través de la capa de transporte. Los datos viajan verticalmente desde el cliente hasta el servidor y viceversa, a pesar de que conceptualmente se perciba como un flujo horizontal





PARA CREAR UN NUEVO OBJETO REMOTO

- Define una interfaz que extienda la interfaz `java.rmi.Remote`.
- La interfaz `Remote` es una interfaz marcadora sin métodos propios, utilizada para etiquetar objetos remotos.
- Un objeto remoto es una instancia de una clase que implementa la interfaz `Remote` o cualquier interfaz que extienda `Remote`.
- La subinterfaz de `Remote` determina qué métodos del objeto remoto pueden ser llamados por los clientes.
- Solo los métodos declarados en la interfaz remota pueden ser invocados de forma remota; los demás métodos solo pueden ser invocados localmente.
- Cada método en la subinterfaz debe declarar que lanza `RemoteException`.

ADDERIMPL



Es una clase con un constructor y el método add.



El constructor se utiliza en el lado del servidor pero no está disponible para el cliente.



Llama al constructor de la superclase que exporta el objeto, creando un UnicastRemoteObject en un puerto y poniéndolo a la escucha de conexiones.



El constructor está declarado para lanzar una excepción RemoteException.

CADA OBJETO REMOTO EN EL SERVIDOR ESTÁ REPRESENTADO POR UNA CLASE DE STUB EN EL CLIENTE.



El stub contiene la información de la interfaz Remote.



En Java 1.5, los stubs se generan automáticamente según sea necesario, pero en Java 1.4 y versiones anteriores, debes compilar manualmente los stubs para cada clase remota.

EJEMPLO

```
import java.rmi.Remote;  
import java.rmi.RemoteException;  
  
// Interfaz remota que extiende Remote  
public interface Sum extends Remote { no usages 1 implementation  
    // Método remoto que suma tres números y lanza RemoteException  
    int add(int a, int b, int c) throws RemoteException; 1 implementation  
}
```

EJEMPLO

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

// Implementación de la interfaz remota Sum
public class SumImpl extends UnicastRemoteObject implements Sum { no usages

    // Constructor que lanza RemoteException
    protected SumImpl() throws RemoteException { no usages
    {
        super();
    }

    // Implementación del método add que suma tres números
    @Override
    public int add(int a, int b, int c) throws RemoteException {
        return a + b + c;
    }
}
```

EJEMPLO

```
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.net.MalformedURLException;

// Clase del servidor
public class SumServer {
    public static void main(String[] args) {
        try {
            // Crear una instancia de SumImpl
            Sum sum = new SumImpl();

            // Iniciar el registro RMI en el puerto 1099
            LocateRegistry.createRegistry(port: 1099);

            // Publicar el objeto remoto en el registro RMI con el nombre "SumService"
            Naming.rebind(name: "SumService", sum);

            System.out.println("Servidor RMI listo.");
        } catch (RemoteException | MalformedURLException e) {
            e.printStackTrace();
        }
    }
}
```


EJEMPLO

```
import java.rmi.Naming;

// Clase del cliente
public class SumClient {
    public static void main(String[] args) {
        try {
            // Buscar el objeto remoto en el registro RMI
            Sum sum = (Sum) Naming.lookup("name: rmi://localhost/SumService");

            // Llamar al método remoto y mostrar el resultado
            int result = sum.add(a: 5, b: 10, c: 15);
            System.out.println("La suma de 5, 10 y 15 es: " + result);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

CONCLUSIÓN

La implementación de stubs en Java RMI ejemplifica cómo se puede simplificar y abstraer la comunicación entre el cliente y el servidor en un sistema distribuido. Los stubs permiten a los clientes invocar métodos remotos como si fueran locales, manejando de manera transparente la transmisión de datos y la ejecución de métodos. Esto reduce la complejidad del desarrollo de aplicaciones distribuidas, permitiendo a los desarrolladores centrarse en la lógica de la aplicación en lugar de los detalles de la comunicación subyacente.

REFERENCIAS

- Harold, E. R. (204). Java Network Programming. En E. R. Harold, *Java Network Programming* (3ra edicion ed., págs. 616,618-620). O'Reilly Media Obtenido de <https://lib.fbtuit.uz/assets/files/Java-NetworkProgrammin.pdf>