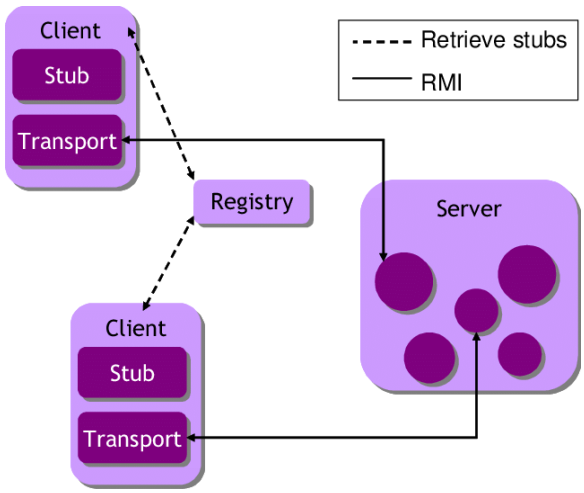
JAVA RMI

Java RMI (Remote Method Invocation) es una API de Java que permite a un programa invocar métodos de objetos que están en diferentes máquinas virtuales Java, ya sea en la misma computadora o en computadoras distintas, como si fueran locales. Esto es especialmente útil en aplicaciones distribuidas donde se requiere que diferentes componentes de software se comuniquen entre sí a través de la red.



Composición de Java RMI

- **Interfaz Remota:** Define los métodos que se pueden invocar de manera remota. Esta interfaz debe extender java.rmi.Remote.
- **Implementación del Servidor:** Es una clase que implementa la interfaz remota. En esta clase, se definen los métodos que se pueden invocar desde un cliente.
- **Cliente RMI:** Es un programa que usa el servicio remoto. Se conecta al objeto remoto y llama a sus métodos.
- **Registro RMI:** Es un servicio que permite a los servidores registrar sus objetos remotos, de modo que los clientes puedan encontrarlos. Esto se hace a través de la clase java.rmi.registry.LocateRegistry.
- **Proxies:** Java RMI utiliza proxies para permitir que los clientes invoquen métodos en objetos remotos como si fueran locales.

Ciclo de Vida de Java RMI

- **Definición de la Interfaz:** Se define la interfaz remota que contiene los métodos que se pueden invocar de manera remota.
- **Implementación del Servidor:** Se crea la implementación del servidor que define cómo se realizan esos métodos.
- **Registro del Objeto Remoto:** El servidor registra el objeto remoto en el registro RMI.
- **Creación del Cliente:** Se desarrolla un cliente que busca el objeto remoto en el registro RMI y realiza invocaciones a sus métodos.

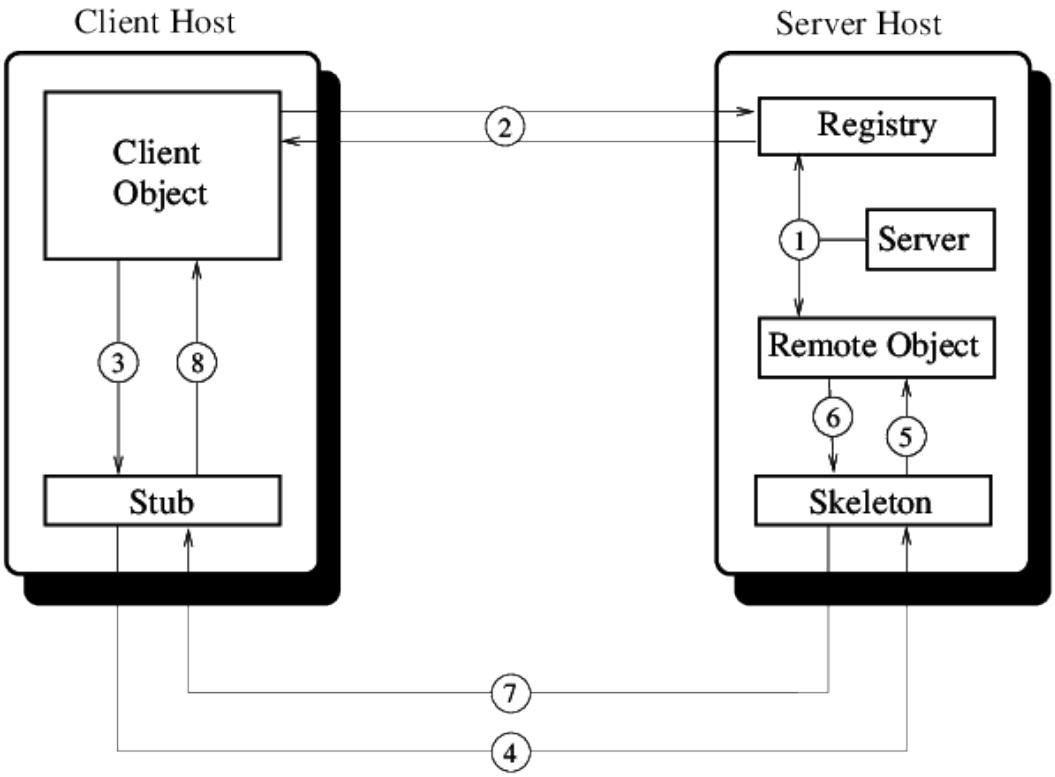
- **Invocación de Métodos:** El cliente invoca métodos en el objeto remoto, y la llamada se envía a través de la red.
- **Respuesta del Servidor:** El servidor procesa la invocación y devuelve la respuesta al cliente.

Aplicaciones de Java RMI

Aplicaciones Distribuidas: Facilita la creación de aplicaciones que requieren la comunicación entre diferentes componentes distribuidos en distintas ubicaciones.

Sistemas de Información: Permite que diferentes módulos de un sistema interactúen, como en sistemas de gestión de bases de datos.

Servicios Web: Aunque no es su propósito principal, puede integrarse en servicios web que requieren comunicación entre clientes y servidores.



Desarrollo:

Para la presente práctica se va a ver una breve aplicación de lo que se puede realizar con java RMI y como se puede utilizar para ejecutar de forma remota una aplicación.

Para ello se realizará una calculadora básica donde se hagan las operaciones de Suma, resta, multiplicación y división, con apoyo del rmiregistry se van a poder intercambiar estas transacciones el cliente al stub y por ese medio al servidor.

Para ello crearemos las clases:

- Calculadora.java
- ClienteCalculadora.java
- ServidorCalculadora.java

Comenzaremos estableciendo la lógica para la clase Calculadora.java como se muestra a continuación:

```
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;
}
```

Se define una interfaz remota llamada Calculadora en Java utilizando la API de Java RMI (Remote Method Invocation). La interfaz extiende Remote, lo que indica que sus métodos pueden ser invocados de manera remota desde clientes que se ejecutan en otras máquinas. Esta interfaz incluye cuatro métodos: sumar, restar, multiplicar, y dividir, cada uno de los cuales recibe dos enteros como parámetros y devuelve un resultado. Todos los métodos declaran que pueden lanzar RemoteException, lo que indica que pueden producir errores relacionados con la red durante la invocación remota. Para implementar esta interfaz, un servidor debe crear una clase que la implemente, proporcionando la lógica para cada operación, y registrar el objeto remoto en el registro RMI para que los clientes puedan acceder a él.

Una vez teniendo la clase Calculadora podremos comenzar con la clase ClienteCalculadora.java:

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.Scanner;

public class ClienteCalculadora {
    public static void main(String[] args) {
        try {
            Registry registry = LocateRegistry.getRegistry("tu_ip", 9090);
            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();
        }
    }
}
```

Se define una clase llamada ClienteCalculadora, que actúa como cliente para invocar métodos en un servicio de cálculo remoto a través de Java RMI. La clase importa las bibliotecas necesarias, incluyendo LocateRegistry y Registry para conectarse al registro RMI y Scanner para la entrada de datos del usuario. En el método main, se establece una conexión con el registro RMI usando la dirección IP del servidor y el puerto 9090, donde se espera que esté registrado el servicio de calculadora bajo el nombre "ServicioCalculadora". Luego, se crea una instancia de Scanner para leer los números ingresados por el usuario. A continuación, el cliente invoca los métodos remotos sumar, restar, multiplicar, y dividir en el objeto Calculadora, pasando los números leídos, y finalmente imprime los resultados en la consola.

Finalmente se tiene la clase ServidorCalculadora.java donde su lógica se explica a continuación:

```
public static void main(String[] args) {
    try {
        ServidorCalculadora servidor = new ServidorCalculadora();
        Registry registry = LocateRegistry.createRegistry(9090);
        registry.rebind("ServicioCalculadora", servidor);
        System.out.println("Servidor de Calculadora en funcionamiento...");
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

El método main es el punto de entrada para el servidor. Aquí se crea una instancia de ServidorCalculadora, lo que también inicializa el objeto remoto. Luego, se crea un registro RMI en el puerto 9090. Con registry.rebind("ServicioCalculadora", servidor), el objeto servidor se registra en el registro RMI bajo el nombre "ServicioCalculadora", lo que permite que los clientes lo busquen e interactúen con él.

```
@Override
public int restar(int a, int b) throws RemoteException {
    return a - b;
}

@Override
```

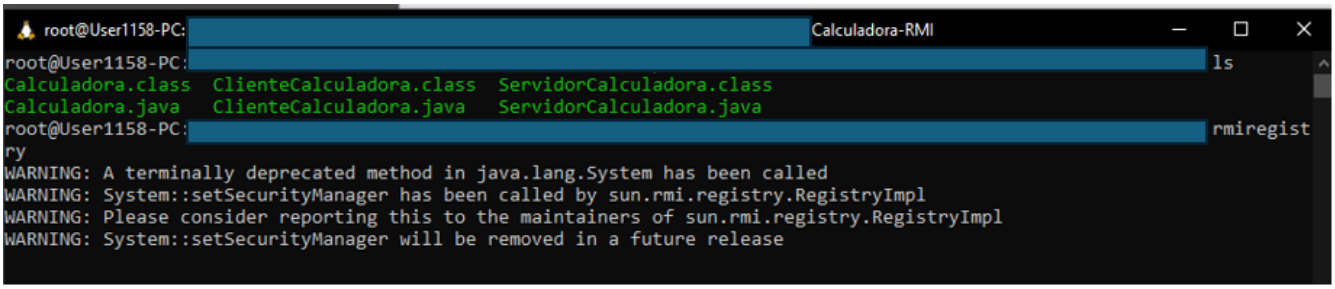
```
public int multiplicar(int a, int b) throws RemoteException {
    return a * b;
}

@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;
}
```

Cada uno de estos métodos realiza la operación matemática correspondiente y lanza RemoteException en caso de errores. En el caso de dividir, se agrega un manejo especial para la división por cero, lanzando una RemoteException con un mensaje específico.

```
public class ServidorCalculadora extends UnicastRemoteObject implements
Calculadora {
```

La clase ServidorCalculadora extiende UnicastRemoteObject. Al hacerlo, el servidor se convierte en un objeto remoto capaz de recibir llamadas a métodos desde clientes. Además, la clase implementa la interfaz Calculadora, que contiene los métodos remotos que se pueden invocar.

Ahora se procede a probar el programa con tres terminales después de compilar como se muestra a continuación donde inicialmente se requiere ejecutar el servicio rmiregistry para que sirva como un intermediario entre las consultas y como distribuye las mismas al servidor:



Después en dos terminales distintas, de preferencia que se encuentren en un sistema operativo distinto o en una red distinta: