



Instituto Politécnico Nacional
Escuela Superior de Cómputo



Aplicaciones para Comunicaciones en Red

Práctica 6: Calculadora RMI

Alumnos:

Malagón Baeza Alan Adrian

Martínez Chávez Jorge Alexis

Profesor:

Moreno Cervantes Axel Ernesto

Grupo: 6CM1

1. Introducción

RMI (Java Remote Method Invocation) es un mecanismo ofrecido por Java para invocar un método de manera remota. Forma parte del entorno estándar de ejecución de Java y proporciona un mecanismo simple para la comunicación de servidores en aplicaciones distribuidas basadas exclusivamente en Java. Si se requiere comunicación entre otras tecnologías debe utilizarse CORBA o SOAP en lugar de RMI.

RMI se caracteriza por la facilidad de su uso en la programación por estar específicamente diseñado para Java; proporciona paso de objetos por referencia (no permitido por SOAP), recolección de basura distribuida (Garbage Collector distribuido) y paso de tipos arbitrarios (funcionalidad no provista por CORBA).

A través de RMI, un programa Java puede exportar un objeto, con lo que dicho objeto estará accesible a través de la red y el programa permanece a la espera de peticiones en un puerto TCP. A partir de ese momento, un cliente puede conectarse e invocar los métodos proporcionados por el objeto.

La invocación se compone de los siguientes pasos:

- Encapsulado (marshalling) de los parámetros (utilizando la funcionalidad de serialización de Java).
- Invocación del método (del cliente sobre el servidor). El invocador se queda esperando una respuesta.
- Al terminar la ejecución, el servidor serializa el valor de retorno (si lo hay) y lo envía al cliente.
- El código cliente recibe la respuesta y continúa como si la invocación hubiera sido local.

2. Desarrollo

El proyecto CalculadoraRMI cuenta con 8 clases:

Clases Servidor

1. AritmeticasInterface
2. AritmeticasServer
3. TrigonometricasInterface
4. TrigonometricasServer
5. StartServers

Clases Cliente

1. RMIClient
2. Main

Clase de constantes:

- Constants

Se procederá a dar una breve explicación de estas clases y las partes importantes del código para el funcionamiento de la aplicación.

2.1 Clase AritmeticasInterface

La interfaz "AritmeticasInterface" es una interfaz pública en Java que define un conjunto de métodos relacionados con operaciones aritméticas. Esta interfaz extiende la interfaz "Remote", lo que significa que los métodos definidos en esta interfaz podrán ser invocados de forma remota.

```
public interface AritmeticasInterface extends Remote {  
  
    double suma(double a, double b) throws RemoteException;  
  
    double resta(double a, double b) throws RemoteException;  
  
    double multiplicacion(double a, double b) throws RemoteException;  
  
    double division(double a, double b) throws RemoteException;  
}
```

```
double exponenciacion(double a, double b) throws RemoteException;
double raiz(double a) throws RemoteException;
}
```

2.2 Clase AritmeticasServer

La clase AritmeticasServer es una implementación de la interfaz AritmeticasInterface y también extiende la clase Thread, lo que significa que puede ser ejecutada como un hilo independiente. Esta clase sirve como el servidor de operaciones aritméticas.

La implementación de AritmeticasServer incluye el método run(), que se ejecuta cuando el hilo se inicia. En este método, se crea un registro RMI (Remote Method Invocation) en el puerto especificado en Constants.RMIA_PORT utilizando java.rmi.registry.LocateRegistry.createRegistry(). Este registro RMI es necesario para que los clientes puedan encontrar y comunicarse con el servidor.

A continuación, se crea una instancia de AritmeticasServer y se exporta como un objeto remoto utilizando UnicastRemoteObject.exportObject(). Luego, se obtiene una referencia al registro RMI utilizando LocateRegistry.getRegistry() y se vincula el objeto remoto al registro con el nombre "Aritmeticas" utilizando registry.bind().

La clase AritmeticasServer implementa todos los métodos definidos en la interfaz AritmeticasInterface. Por lo tanto, proporciona la funcionalidad real para realizar operaciones aritméticas, como la suma, resta, multiplicación, división, exponenciación y raíz cuadrada.

En la implementación de cada método, simplemente se realiza la operación correspondiente y se devuelve el resultado. Por ejemplo, el método suma(double a, double b) devuelve la suma de a y b, el método resta(double a, double b) devuelve la resta de a y b, y así sucesivamente.

Cada método declara que puede lanzar una excepción de tipo RemoteException, lo cual es necesario cuando se trabaja con RMI, ya que puede ocurrir algún error durante la comunicación remota.

```
public class AritmeticasServer extends Thread implements
AritmeticasInterface {

    public void run() {
        try {

java.rmi.registry.LocateRegistry.createRegistry(Constants.RMIA_PORT);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```

    }
    try {

//java.rmi.registry.LocateRegistry.createRegistry(Constants.RMI_PORT);
        AritmeticasServer obj = new AritmeticasServer();
        AritmeticasInterface stub = (AritmeticasInterface)
UnicastRemoteObject.exportObject(obj, 0);
        Registry registry =
LocateRegistry.getRegistry(Constants.RMIA_PORT);
        registry.bind("Aritmeticas", stub);
    }catch (Exception e){
        e.printStackTrace();
    }
}

    public double suma(double a, double b) throws RemoteException {
        return a + b;
    }

    public double resta(double a, double b) throws RemoteException {
        return a - b;
    }

    public double multiplicacion(double a, double b) throws
RemoteException {
        return a * b;
    }

    public double division(double a, double b) throws RemoteException{
        return a / b;
    }

    @Override
    public double exponenciacion(double a, double b) throws
RemoteException {
        return Math.pow(a,b);
    }

    @Override
    public double raiz(double a) throws RemoteException {
        return Math.sqrt(a);
    }
}

```

2.3 Clase TrigonometricasInterface

Misma aplicación que AritmeticasInterface pero para las operaciones trigonométricas.

```

public interface TrigonometricasInterface extends Remote {

    double cos(double angle) throws RemoteException;

    double sin(double angle) throws RemoteException;
}

```

```

    double tan(double angle) throws RemoteException;
    double cot(double angle) throws RemoteException;
    double csc(double angle) throws RemoteException;
    double sec(double angle) throws RemoteException;
    double log(double angle) throws RemoteException;
}

```

2.4 Clase TrigonometricasServer

Misma aplicación que AritmeticasServer pero para las operaciones trigonométricas.

```

public class TrigonometricasServer extends Thread implements
TrigonometricasInterface {

    public void run() {
        try {

java.rmi.registry.LocateRegistry.createRegistry(Constants.RMIT_PORT);
        } catch (Exception e) {
            e.printStackTrace();
        }
        try {

//java.rmi.registry.LocateRegistry.createRegistry(Constants.RMI_PORT);
            TrigonometricasServer obj = new TrigonometricasServer();
            TrigonometricasInterface stub = (TrigonometricasInterface)
UnicastRemoteObject.exportObject(obj, 0);

Registry registry =
LocateRegistry.getRegistry(Constants.RMIT_PORT);
            registry.bind("Trigonometricas", stub);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public double cos(double angle) throws RemoteException {
        return Math.cos(Math.toRadians(angle));
    }

    public double sin(double angle) throws RemoteException {
        return Math.sin(Math.toRadians(angle));
    }

    public double tan(double angle) throws RemoteException {
        return Math.tan(Math.toRadians(angle));
    }

    @Override
    public double cot(double angle) throws RemoteException {
        return 1/Math.tan(Math.toRadians(angle));
    }

    @Override
    public double csc(double angle) throws RemoteException {

```

```

        return 1/Math.sin(Math.toRadians(angle));
    }

    @Override
    public double sec(double angle) throws RemoteException {
        return 1/Math.cos(Math.toRadians(angle));
    }

    @Override
    public double log(double angle) throws RemoteException {
        return Math.log(angle);
    }
}

```

2.5 Clase StartServers

Permite iniciar los servidores: AritmeticasServer y TrigonometricasServer

```

public class StartServers extends Thread{

    AritmeticasServer aritmeticasServer = new AritmeticasServer();
    TrigonometricasServer trigonometricasServer = new
TrigonometricasServer();
    public StartServers(){
        System.out.println(Constants.SERVER_INIT_MSG);
        aritmeticasServer.start();
        trigonometricasServer.start();
    }

    public static void main(String[] args) {
        try {
            StartServers startServers = new StartServers();
            startServers.start();
        }catch (Exception e){
            e.printStackTrace();
        }
    }
}

```

2.6 Clase RMIClient

La clase `RMIClient` es un cliente RMI que se encarga de realizar cálculos aritméticos y trigonométricos utilizando un servidor remoto que implementa las interfaces `AritmeticasInterface` y `TrigonometricasInterface`.

La clase extiende `Thread`, lo que permite que se ejecute como un hilo independiente. En su método `run()`, que se ejecuta cuando el hilo se inicia, no tiene ninguna implementación específica.

El método principal de esta clase es `calcular(String expresion)`, que recibe una expresión matemática como argumento. Dentro de este método, se establece la conexión con el servidor remoto utilizando la clase `Registry` y los métodos `LocateRegistry.getRegistry()` para obtener las referencias a los registros RMI.

Se obtiene la referencia al servidor remoto de operaciones aritméticas utilizando `aritmeticasRegistry.lookup("Aritmeticas")`, y la referencia al servidor remoto de operaciones trigonométricas utilizando `trigonometricasRegistry.lookup("Trigonometricas")`.

A continuación, se llama al método `evaluateExpression()` para evaluar la expresión matemática utilizando los servidores remotos. Este método utiliza pilas (`Stack`) para evaluar la expresión en notación infija. Se recorre cada carácter de la expresión y se realiza una serie de operaciones dependiendo del tipo de carácter.

Si se encuentra un dígito, se agrega a la pila de números (`numbers`). Si se encuentra un paréntesis de apertura, se agrega a la pila de operadores (`operators`). Si se encuentra un paréntesis de cierre, se realizan operaciones hasta que se encuentra un paréntesis de apertura, aplicando los operadores a los números correspondientes. Si se encuentra un operador, se realizan operaciones hasta que se encuentra un operador de menor precedencia en la pila.

Si se encuentra una letra, se trata como una función trigonométrica y se agrega a la pila de operadores. Al finalizar el recorrido de la expresión, se realizan las operaciones restantes en las pilas.

Los métodos `applyOperator()` se utilizan para aplicar las operaciones correspondientes a los números utilizando los servidores remotos. Dependiendo del tipo de operador, se invoca el método correspondiente en el servidor remoto para obtener el resultado de la operación.

La clase también define métodos auxiliares `isOperator()`, `hasPrecedence()` y `evaluateExpression()` que ayudan en la evaluación de la expresión y el orden de las operaciones.

```
public class RMIClient extends Thread{

    public void run(){}
    public void calcular(String expresion) {
        try {
            Registry aritmeticasRegistry =
LocateRegistry.getRegistry(Constants.host, Constants.RMIA_PORT);
            AritmeticasInterface aritmeticasServer =
(AritmeticasInterface) aritmeticasRegistry.lookup("Aritmeticas");
```



```

        Registry trigonometricasRegistry =
LocateRegistry.getRegistry(Constants.host, Constants.RMIT_PORT);
        TrigonometricasInterface trigonometricasServer =
        (TrigonometricasInterface)
trigonometricasRegistry.lookup("Trigonometricas");

        //String expression = "(2+3)*sin(90)";
        double result = evaluateExpression(expression,
aritmeticasServer, trigonometricasServer);

        System.out.println("Resultado: " + result);

    } catch (Exception e) {
        e.printStackTrace();
    }
}

public double evaluateExpression(String expression,
AritmeticasInterface aritmeticasServer, TrigonometricasInterface
trigonometricasServer) throws RemoteException {
    Stack<Double> numbers = new Stack<>();
    Stack<String> operators = new Stack<>();

    for (int i = 0; i < expression.length(); i++) {
        char c = expression.charAt(i);

        if (Character.isDigit(c)) {
            StringBuilder numBuilder = new StringBuilder();
            numBuilder.append(c);

            while (i + 1 < expression.length() &&
Character.isDigit(expression.charAt(i + 1))) {
                numBuilder.append(expression.charAt(i + 1));
                i++;
            }

            double num = Double.parseDouble(numBuilder.toString());
            numbers.push(num);
        } else if (c == '(') {
            operators.push("(");
        } else if (c == ')') {
            while (!operators.isEmpty() &&
!operators.peek().equals("(")) {
                String operator = operators.pop();
                if (operator.equals("sin") || operator.equals("cos")
|| operator.equals("tan") ||
                    operator.equals("cot") || operator.equals("csc")
|| operator.equals("sec") ||
                    operator.equals("log") || operator.equals("sqrt"))
                {
                    double num = numbers.pop();
                    double result;
                    if(operator.equals("sqrt")) result =
applyOperator(num,2, operator, aritmeticasServer);
                    else result = applyOperator(num, operator,
trigonometricasServer);

```

```

        numbers.push(result);
    } else {
        double num2 = numbers.pop();
        double num1 = numbers.pop();
        double result = applyOperator(num1, num2,
operator, aritmeticasServer);
        numbers.push(result);
    }
}

    if (!operators.isEmpty() && operators.peek().equals("("))
{
        operators.pop(); // Remove '(' from stack
    }
    } else if (isOperator(String.valueOf(c))) {
        String operator = String.valueOf(c);
        while (!operators.isEmpty() && hasPrecedence(operator,
operators.peek())) {
            String topOperator = operators.pop();
            if (topOperator.equals("sin") ||
topOperator.equals("cos") || topOperator.equals("tan") ||
                topOperator.equals("cot") ||
topOperator.equals("csc") || topOperator.equals("sec") ||
                topOperator.equals("log") ||
topOperator.equals("sqrt")) {
                double num = numbers.pop();

                double result;
                if(topOperator.equals("sqrt")) result =
applyOperator(num,2, topOperator, aritmeticasServer);
                else result = applyOperator(num, topOperator,
trigonometricasServer);
                numbers.push(result);
            }
            else {
                double num2 = numbers.pop();
                double num1 = numbers.pop();
                double result = applyOperator(num1, num2,
topOperator, aritmeticasServer);
                numbers.push(result);
            }
        }

        operators.push(operator);
    } else if (Character.isLetter(c)) {
        StringBuilder funcBuilder = new StringBuilder();
        funcBuilder.append(c);

        while (i + 1 < expression.length() &&
Character.isLetter(expression.charAt(i + 1))) {
            funcBuilder.append(expression.charAt(i + 1));
            i++;
        }

        String function = funcBuilder.toString();
        operators.push(function);
    }
}

```

```

    }

    while (!operators.isEmpty()) {
        String operator = operators.pop();
        if (operator.equals("sin") || operator.equals("cos") ||
operator.equals("tan") ||
        operator.equals("cot") || operator.equals("csc") ||
operator.equals("sec") ||
        operator.equals("log") || operator.equals("sqrt")) {
            double num = numbers.pop();
            double result;
            if(operator.equals("sqrt")) result = applyOperator(num,2,
operator, aritmeticasServer);
            else result = applyOperator(num, operator,
trigonometricasServer);
            numbers.push(result);
        } else {
            double num2 = numbers.pop();
            double num1 = numbers.pop();
            double result = applyOperator(num1, num2, operator,
aritmeticasServer);
            numbers.push(result);
        }
    }

    return numbers.pop();
}

private double applyOperator(double num, String operator,
TrigonometricasInterface trigonometricasServer) throws RemoteException {
    if (operator.equals("sin")) {
        return trigonometricasServer.sin(num);
    } else if (operator.equals("cos")) {
        return trigonometricasServer.cos(num);
    } else if (operator.equals("tan")) {
        return trigonometricasServer.tan(num);
    } else if (operator.equals("cot")) {
        return trigonometricasServer.cot(num);
    } else if (operator.equals("sec")) {
        return trigonometricasServer.sec(num);
    } else if (operator.equals("csc")) {
        return trigonometricasServer.csc(num);
    } else if (operator.equals("log")) {
        return trigonometricasServer.log(num);
    }

    return 0.0; // Default case
}

private double applyOperator(double num1, double num2, String
operator, AritmeticasInterface aritmeticasServer) throws RemoteException
{
    if (operator.equals("+")) {
        return aritmeticasServer.suma(num1, num2);
    } else if (operator.equals("-")) {
        return aritmeticasServer.resta(num1, num2);
    } else if (operator.equals("*")) {

```

```

        return aritmeticasServer.multiplicacion(num1, num2);
    } else if (operator.equals("/")) {
        return aritmeticasServer.division(num1, num2);
    } else if (operator.equals("^")) {
        return aritmeticasServer.exponenciacion(num1, num2);
    } else if (operator.equals("sqrt")) {
        return aritmeticasServer.raiz(num1);
    }

    return 0.0; // Default case
}

private boolean isOperator(String operator) {
    return operator.equals("+") || operator.equals("-") ||
operator.equals("*") || operator.equals("/") || operator.equals("^")
    || operator.equals("sin") || operator.equals("cos") ||
operator.equals("tan") ||
    operator.equals("cot") || operator.equals("csc") ||
operator.equals("sec") ||
    operator.equals("log") || operator.equals("sqrt");
}

private boolean hasPrecedence(String op1, String op2) {
    if (op2.equals("(") || op2.equals(")")) {
        return false;
    }

    if ((op1.equals("^") && !op2.equals("^")) || ((op1.equals("*") ||
op1.equals("/")) && (op2.equals("+") || op2.equals("-")))) {
        return false;
    }

    return true;
}
}

```

2.7 Clase Main

La clase `Main` contiene el método principal `main`, que es el punto de entrada del programa. En este caso, el programa utiliza la clase `RMIClient` para realizar cálculos aritméticos y trigonométricos utilizando un servidor remoto.

En el método `main`, se crea una instancia de `RMIClient` llamada `rmiClient` y se inicia como un hilo independiente mediante el método `start()`. Esto permite que el cliente RMI se ejecute en segundo plano mientras el programa principal continúa su ejecución.

A continuación, se crea un objeto `Scanner` para leer las expresiones matemáticas ingresadas por el usuario desde la consola.

El programa entra en un bucle infinito donde se le solicita al usuario ingresar una expresión matemática utilizando el mensaje "Ingrese una expresión (o escriba 'salir' para finalizar): ". El usuario puede ingresar la expresión o escribir "salir" para finalizar el programa.

Dentro del bucle, se lee la expresión ingresada por el usuario utilizando `scanner.nextLine()`. Si la expresión es igual a "salir", se rompe el bucle y se finaliza el programa. De lo contrario, se llama al método `calcular()` de `rmiClient` para evaluar la expresión ingresada.

El resultado de la evaluación de la expresión se imprime en la consola.

Después de que el bucle termina, se cierra el objeto `Scanner` y el programa finaliza su ejecución.

```
public class Main {  
  
    public static void main(String[] args) {  
        RMIClient rmiClient = new RMIClient();  
        rmiClient.start();  
  
        Scanner scanner = new Scanner(System.in);  
        String expresion;  
  
        while (true) {  
            System.out.print("Ingrese una expresión (o escriba 'salir'  
para finalizar): ");  
            expresion = scanner.nextLine();  
  
            if (expresion.equalsIgnoreCase("salir")) {  
                break;  
            }  
  
            rmiClient.calcular(expresion);  
        }  
  
        scanner.close();  
    }  
  
    // (2^3)*(sin(90))  
}
```

2.8 Clase Constants

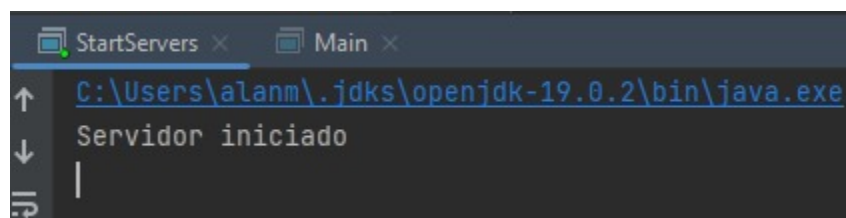
La clase `Constants` define algunas constantes utilizadas en el programa. Estas constantes representan valores fijos que se utilizan en varias partes del código y que no deben cambiar durante la ejecución del programa.

- `SERVER_INIT_MSG`: Es una constante de tipo `String` que almacena el mensaje "Servidor iniciado". Probablemente se utiliza para imprimir un mensaje en la consola o registrar el inicio del servidor.
- `RMIT_PORT`: Es una constante de tipo `int` que representa el puerto utilizado para la comunicación RMI con el servidor de operaciones trigonométricas.
- `RMIA_PORT`: Es una constante de tipo `int` que representa el puerto utilizado para la comunicación RMI con el servidor de operaciones aritméticas.
- `host`: Es una constante de tipo `String` que almacena el nombre del host o dirección IP donde se encuentra el servidor. En este caso, se establece como "localhost", lo que significa que el servidor se encuentra en la misma máquina que el cliente.

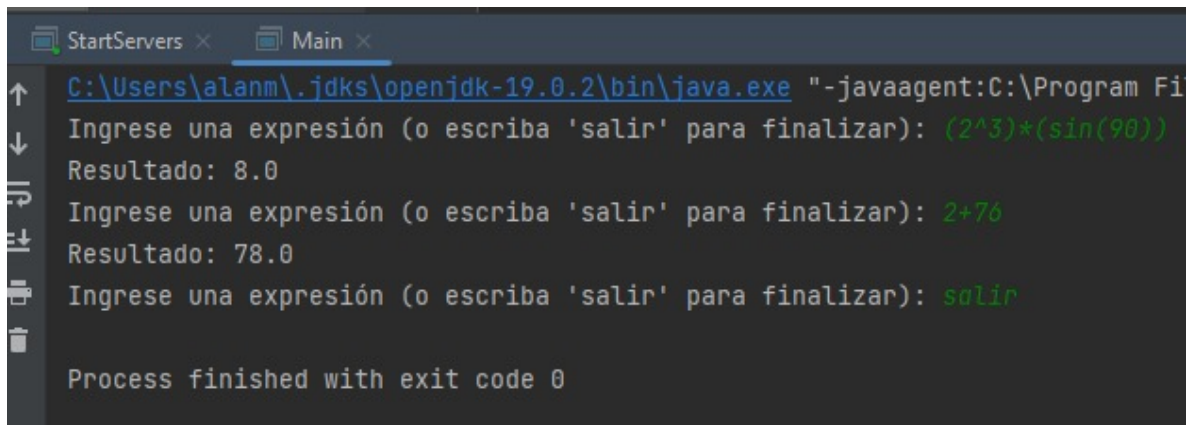
```
public class Constants {  
    public static final String SERVER_INIT_MSG = "Servidor iniciado";  
    public static final int RMIT_PORT = 1099;  
    public static final int RMIA_PORT = 1098;  
    public static String host = "localhost";  
}
```

3. Pruebas

Corremos la clase `StartServers` y nos mandará mensaje de que se inicio el servidor



Corremos la clase `Main` y nos pedirá ingresar la expresión a calcular o ingresar "salir" para terminar el programa.



```
StartServers x Main x
C:\Users\alanm\.jdk\openjdk-19.0.2\bin\java.exe "-javaagent:C:\Program Fi
Ingrese una expresión (o escriba 'salir' para finalizar): (2^3)*(sin(90))
Resultado: 8.0
Ingrese una expresión (o escriba 'salir' para finalizar): 2+76
Resultado: 78.0
Ingrese una expresión (o escriba 'salir' para finalizar): salir
Process finished with exit code 0
```

5. Conclusiones

En conclusión, la práctica realizada ilustra la implementación de un sistema distribuido de cálculos matemáticos utilizando RMI (Remote Method Invocation). El sistema consta de un servidor y uno o varios clientes que se comunican entre sí para realizar operaciones aritméticas y trigonométricas.

El servidor RMI proporciona una interfaz que define los métodos de cálculo y su implementación correspondiente. Utiliza el registro RMI para registrar sus servicios y está disponible para recibir las solicitudes de cálculo de los clientes.

Por otro lado, los clientes RMI utilizan las interfaces definidas en el servidor para invocar los métodos de cálculo de forma remota. Los clientes se comunican con el servidor a través del registro RMI para obtener acceso a los servicios y enviar las solicitudes de cálculo.

La comunicación entre el cliente y el servidor se realiza de manera transparente gracias a la tecnología RMI, lo que permite que los cálculos se realicen en el servidor y los resultados se devuelvan al cliente de forma eficiente.

El uso de constantes en el código ayuda a mantener la legibilidad y la modularidad, permitiendo cambios más fáciles en los valores necesarios, como puertos y direcciones de host.

En resumen, esta práctica ejemplifica la implementación de un sistema distribuido de cálculos matemáticos utilizando RMI. Este enfoque permite una comunicación

fluida y transparente entre el servidor y los clientes, brindando la posibilidad de realizar cálculos complejos de manera remota y compartida.

.

Bibliografía

- Oracle. (2021). Java Remote Method Invocation (RMI). Recuperado de <https://docs.oracle.com/en/java/javase/14/docs/api/java.rmi.html>
- Grosso, W. (2001). Java RMI. Sebastopol, CA: O'Reilly Media