



Práctica #2

Implementación de RMI

Benemérita Universidad Autónoma de Puebla

Alumnos:

Luis Angel Avendaño Avalos – 201933893

Christian Amauri Amador Ortega – 201927821

Programación Distribuida

Otoño 2023

Practica 2.

Introducción.

En esta práctica, exploramos la implementación de un sistema RMI (Remote Method Invocation) en Java, que permite la comunicación entre aplicaciones distribuidas en red a través de objetos remotos. Durante nuestra práctica, abordamos la configuración de un servidor RMI y la implementación de un cliente que se comunica con el servidor. Además, implementamos la forma de pasar un mensaje desde el cliente al servidor.

Las cuatro clases proporcionadas son esenciales para el funcionamiento de este sistema RMI:

HelloInterface: Una interfaz que extiende Remote y define los métodos remotos que pueden ser invocados desde el cliente.

```
import java.rmi.*;

public interface HelloInterface extends Remote {
    public String say() throws RemoteException;
}
```

Hello: La clase que implementa la interfaz HelloInterface. Contiene el mensaje de saludo y puede proporcionar una respuesta al cliente.

```
import java.rmi.*;
import java.rmi.server.*;

public class Hello extends UnicastRemoteObject implements HelloInterface {
    private String message;

    public Hello (String msg) throws RemoteException {
        message = msg;
    }

    public String say() throws RemoteException {
        return message;
    }
}
```

HelloClient: El programa cliente que se comunica con el servidor a través de RMI. El cliente se conecta al servidor y puede enviar un mensaje al servidor para recibir una respuesta.

```
import java.rmi.*;
import java.io.*;

public class HelloClient{

    public static void main (String[] argv) {
        try {
            HelloInterface hello = (HelloInterface) Naming.lookup ("//localhost/Hello");
            System.out.println (hello.say());
        } catch (Exception e) {
            System.out.println ("HelloClient exception: " + e);
        }
    }
}
```

HelloServer: El programa servidor que registra un objeto remoto y escucha las solicitudes de los clientes. Cuando se conecta un cliente, el servidor puede proporcionar una respuesta al cliente.

```
import java.io.*;
import java.rmi.*;

public class HelloServer{
    public static void main (String[] argv) {
        try {
            Naming.rebind ("Hello", new Hello("Hello, world!")); //( "Hello, world!");
            System.out.println ("Hello Server is ready.");
        } catch (Exception e) {
            System.out.println ("Hello Server failed: " + e);
        }
    }
}
```

En una de las fases de nuestra práctica, también agregamos la funcionalidad para contar las vocales en un mensaje proporcionado por el cliente en el servidor. Esta característica amplió la funcionalidad de nuestro sistema RMI.

Sustento Teórico

En esta práctica, exploramos y aplicamos la tecnología de RMI (Remote Method Invocation) en el lenguaje de programación Java. RMI es una tecnología que permite la comunicación y la invocación de métodos en objetos remotos a través de la red. Para comprender esta práctica, es importante abordar varios conceptos clave:

1. RMI (Remote Method Invocation): RMI es un mecanismo que permite a los objetos Java interactuar y comunicarse a través de la red, como si estuvieran en la misma máquina virtual. Esto se logra mediante la invocación remota de métodos en objetos distribuidos.

2. Objetos Remotos: Los objetos remotos son objetos Java que pueden ser referenciados y utilizados en una máquina virtual diferente. Estos objetos se registran y exponen a través de un servidor RMI y se acceden desde un cliente RMI.

3. Interfaz Remota: Las interfaces remotas en RMI definen los métodos que pueden ser llamados de forma remota. Los objetos remotos deben implementar estas interfaces para exponer sus métodos a través de la red. La interfaz remota actúa como un contrato entre el cliente y el servidor, especificando qué métodos pueden ser invocados de forma remota.

4. Servidor RMI: El servidor RMI es una aplicación que registra objetos remotos y los expone para su invocación a través de la red. Los clientes se comunican con el servidor para invocar métodos en objetos remotos registrados en él.

5. Cliente RMI: El cliente RMI es una aplicación que se conecta al servidor RMI para invocar métodos en objetos remotos. El cliente utiliza la interfaz remota para acceder a los métodos proporcionados por los objetos remotos registrados en el servidor.

6. Invocación Remota de Métodos: La invocación de métodos remotos implica que un cliente llama a un método en un objeto remoto a través de la red. El cliente envía parámetros al servidor, que realiza la invocación y devuelve una respuesta al cliente.

7. Configuración del Entorno: La configuración del entorno de desarrollo RMI implica la instalación de Java, la definición de clases que implementan interfaces remotas y la configuración de la clase del servidor y del cliente para que puedan utilizar el sistema RMI.

8. Comunicación Cliente-Servidor: La comunicación entre el cliente y el servidor RMI se basa en la invocación de métodos definidos en la interfaz remota. El cliente llama a estos métodos en el servidor y recibe respuestas.

Objetivo.

El objetivo principal de esta práctica es familiarizarse y experimentar con la tecnología de RMI (Remote Method Invocation) en el contexto del lenguaje de programación Java. Los objetivos específicos de esta práctica son los siguientes:

- Comprender los conceptos fundamentales de RMI: Adquirir conocimientos sobre cómo RMI permite la comunicación entre aplicaciones distribuidas y cómo se invocan métodos de forma remota.
- Configurar un entorno de desarrollo RMI: Configurar un entorno de desarrollo adecuado para trabajar con RMI, lo que incluye la instalación de Java y la preparación de las máquinas que actuarán como el servidor y el cliente.
- Diseñar e implementar las clases RMI: Crear y definir las clases esenciales para la implementación de RMI, incluyendo la interfaz remota, la clase del servidor y la clase del cliente.
- Lograr la comunicación Cliente-Servidor: Establecer la comunicación efectiva entre el cliente y el servidor utilizando RMI. El cliente debe ser capaz de enviar mensajes al servidor y recibir respuestas.
- Ampliar las capacidades del sistema: Agregar funcionalidades adicionales al sistema RMI, como el recuento de vocales en un mensaje enviado por el cliente al servidor, para ilustrar cómo se pueden extender las capacidades de una aplicación RMI.

Al lograr estos objetivos, adquiriremos una comprensión práctica de la tecnología RMI en el contexto de aplicaciones distribuidas en Java. Esto nos permitirá aplicar estos conocimientos en el diseño y desarrollo de sistemas distribuidos y aplicaciones en red que requieran comunicación entre componentes remotos.

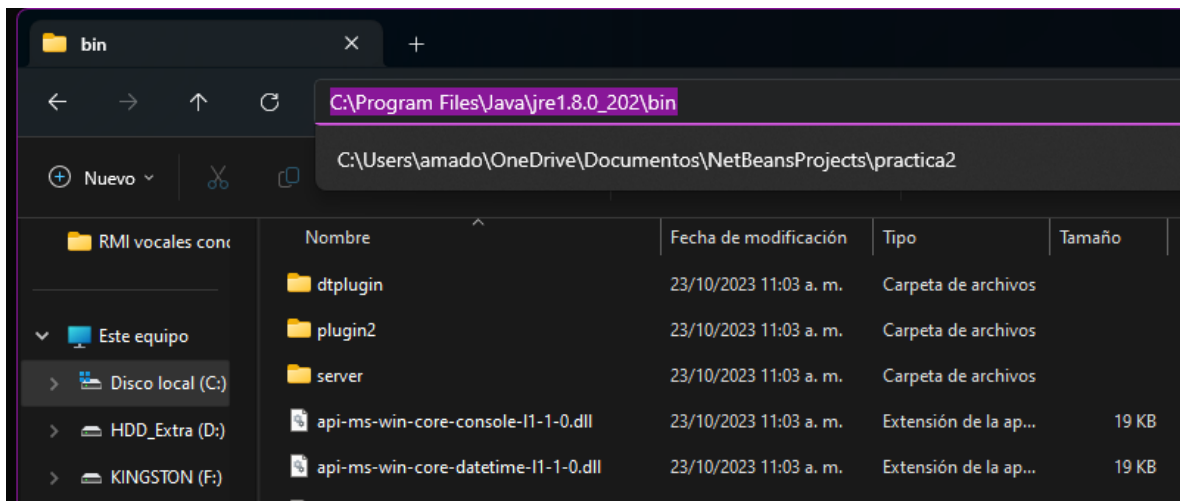
Desarrollo.

Parte I.

En la primera fase de la práctica, nos centramos en la configuración inicial y la ejecución local del sistema RMI. Para lograr esto, se llevaron a cabo los siguientes pasos:

1. Configuración del Entorno:

Se verificó que Java estuviera correctamente instalado en las máquinas que actuarían como el servidor y el cliente. Se confirmó que el entorno de desarrollo estuviera configurado adecuadamente para trabajar con RMI. (La versión del JDK compatible con esta tecnología es la versión 1.8.0 o inferiores, ya que esta es una tecnología vieja) Para garantizar que se esté trabajando con la versión del JDK apropiada para la práctica, primero debemos descargar dicha versión. Luego de eso, debemos añadir la ruta de su carpeta bin, al path de las variables de entorno de nuestro sistema operativo.



Parte II.

En la segunda fase del desarrollo, se implementó el método `cuentaVocales` para contar las vocales en un mensaje proporcionado por el cliente al servidor. Además, se configuró la ejecución remota para permitir que el cliente y el servidor se comunicaran a través de la red. Aquí están los detalles de esta parte del desarrollo:

1. Implementación del Método "cuentaVocales":

Se agregó un nuevo método, `cuentaVocales`, a la interfaz remota `HelloInterface`, sacado del método `cuenta vocales` de la práctica 1 de este curso. Este método acepta una cadena de texto como argumento y devuelve el número de vocales en el mensaje, siguiendo las reglas especificadas. En la clase `Hello`, se implementó el método `cuentaVocales` de acuerdo con las reglas proporcionadas en el enunciado. El método realiza un recorrido de la cadena de texto, identifica las vocales y devuelve su recuento.

```
//Devuelve la cantidad de vocales de la frase
private static int numeroDeVocales(String frase) {
    int res = 0;
    String fraseMin = frase.toLowerCase();

    for (int i = 0; i < fraseMin.length(); ++i) {
        switch(fraseMin.charAt(i)) {
            case 'a': case 'á':
            case 'e': case 'é':
            case 'i': case 'í':
            case 'o': case 'ó':
            case 'u': case 'ú':
                res++;
                break;
            default:
                // se ignoran las demás letras
        }
    }
    return res;
}
```

2. Configuración de la Ejecución Remota:

La configuración se modificó para permitir que el cliente y el servidor se comunicaran a través de la red en lugar de localmente. (esto se logra reemplazando el valor 'localhost' por la IP del servidor a la que te tratas de conectar, en la clase `HelloClient.java`)

3. Registro y Ejecución Remota:

El servidor RMI se ejecutó en una máquina, y el cliente se ejecutó en otra máquina dentro de la misma red o en una red accesible. (explicación en la sección de resultados)

Resultados.

Parte I.

1. Configuración de la Ejecución Local:

Se compiló el código Java utilizando el comando `javac` desde el terminal de Windows (CMD). Esto generó los archivos de clase necesarios para ejecutar el servidor y el cliente. Es importante destacar que si tienes más de una versión del JDK instalada, aunque tengas la 1.8.0 agregada al path, esto puede causar problemas de compilación, por la coexistencia de versiones, es por eso que en esta parte es importante especificarle al comando `javac`, la versión sobre la que estamos trabajando, usando el comando de la siguiente manera:

```
Javac -source 8 -target 8 Hello*.java
```

2. Registro y Ejecución Local:

Se registró el objeto remoto en el servidor utilizando el método `Naming.rebind()`. Esto permitió que el servidor estuviera disponible localmente en la dirección `//localhost/Hello`. El servidor RMI se ejecutó en una máquina y el cliente en otra, pero ambos dentro de la misma red local. La comunicación se realizó localmente a través de `localhost`. Esto lo logramos mediante el uso del comando:

```
rmic Hello
```

A continuación, iniciamos `"rmiregistry"` para luego poder ejecutar el server, con los siguientes comandos:

```
Start rmiregistry
```

```
Java HelloServer
```

3. Ejecución del Cliente:

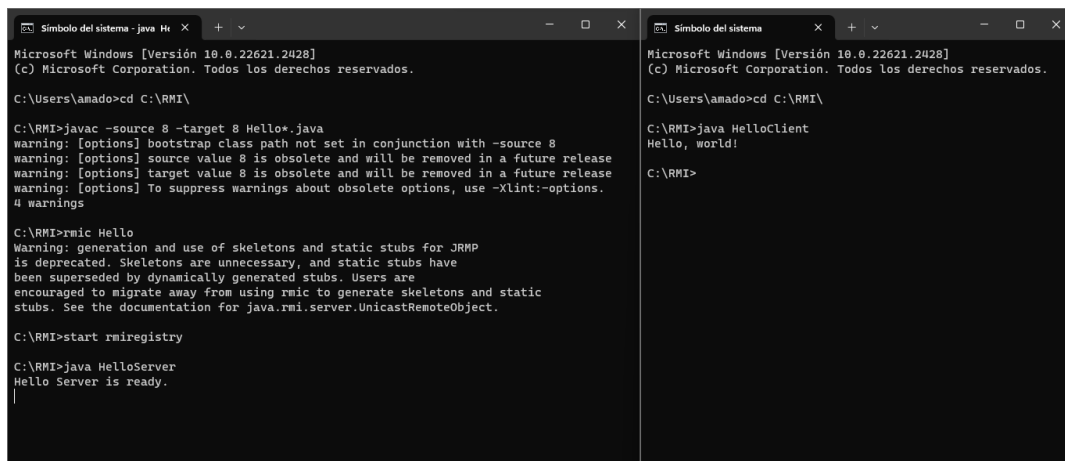
El cliente se ejecutó desde el terminal de Windows proporcionando un mensaje como argumento de línea de comandos, permitiendo al cliente enviar un mensaje al servidor y recibir una respuesta. (obviamente primero tenemos que haber tenido éxito con todos los pasos anteriormente descritos). Para ejecutar el cliente, debemos abrir otra terminal, regresar a la misma ubicación donde ejecutamos todos los comandos anteriores, y escribir el comando:

Java HelloClient

4. Resultados de la Ejecución Local:

Se observó que la configuración y la comunicación local se llevaron a cabo con éxito, lo que permitió al cliente y al servidor interactuar a través de RMI en un entorno controlado.

En esta primera parte del desarrollo, se logró la configuración inicial del entorno y la ejecución local del sistema RMI. Esto proporcionó una base sólida para continuar con la expansión de la funcionalidad del sistema RMI en las siguientes fases de la práctica.



```
Microsoft Windows [Versión 10.0.22621.2428]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\amado>cd C:\RMI\

C:\RMI>javac -source 8 -target 8 Hello*.java
warning: [options] bootstrap class path not set in conjunction with -source 8
warning: [options] source value 8 is obsolete and will be removed in a future release
warning: [options] target value 8 is obsolete and will be removed in a future release
warning: [options] To suppress warnings about obsolete options, use -Xlint:-options.
4 warnings

C:\RMI>rmic Hello
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.

C:\RMI>start rmiregistry

C:\RMI>java HelloServer
Hello Server is ready.
|

Microsoft Windows [Versión 10.0.22621.2428]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\amado>cd C:\RMI\

C:\RMI>java HelloClient
Hello, world!

C:\RMI>
```

Parte II.

1. Ejecución del Cliente Remoto:

El cliente se ejecutó desde la máquina remota proporcionando un mensaje como argumento de línea de comandos. El cliente se conectó al servidor a través de la red para enviar un mensaje y solicitar el recuento de vocales. Para esto se debe verificar que el valor 'localhost' de la clase HelloClient.java haya sido reemplazada por la IP del servidor. Esto a veces puede ser conflictuoso, pues al parecer el sistema puede tener problemas con la resolución de las IP's, por el caché DNS del sistema operativo. Para intentar corregir este problema, debemos escribir el comando:

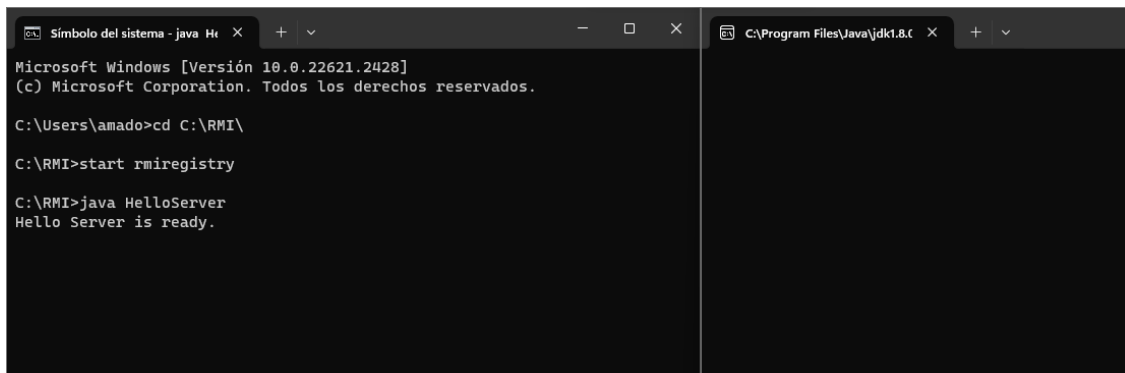
Ipconfig/flushdns

Desde el cmd, de ambas máquinas (cliente y servidor) y luego ejecutar el código correspondiente en cada máquina, siguiendo en cada máquina, los pasos descritos en la ejecución local.

2. Resultados de la Ejecución Remota:

Se confirmó que la configuración y la comunicación remota se llevaron a cabo con éxito. El cliente pudo enviar mensajes al servidor a través de la red y recibir respuestas que incluían el mensaje de saludo y el recuento de vocales.

En esta segunda parte del desarrollo, se amplió la funcionalidad del sistema RMI al agregar el método cuentaVocales y se configuró la ejecución remota para permitir la comunicación a través de la red. Esto demostró la capacidad de RMI para facilitar la comunicación entre componentes distribuidos en una red, ampliando las aplicaciones y las funcionalidades que se pueden lograr mediante esta tecnología.



```
Símbolo del sistema - java H... x + v
Microsoft Windows [Versión 10.0.22621.2428]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\amado>cd C:\RMI\

C:\RMI>start rmiregistry

C:\RMI>java HelloServer
Hello Server is ready.
```

```
Selecciónar Símbolo del sistema
Microsoft Windows [Versión 10.0.19045.2728]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\diosd>cd C:\RMI\

C:\RMI>javac -source 8 -target 8 Hello*.java
warning: [options] bootstrap class path not set in conjunction with -source 8
warning: [options] source value 8 is obsolete and will be removed in a future release
warning: [options] target value 8 is obsolete and will be removed in a future release
warning: [options] To suppress warnings about obsolete options, use -Xlint:-options.
4 warnings

C:\RMI>rmic Hello
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.

C:\RMI>java HelloClient 'Hola mundo ¿cómo están? espero que bien!'
Hello, world!
La frase "Hola" tiene: 2 vocales.

C:\RMI>java HelloClient "Hola mundo ¿cómo están? espero que bien!"
Hello, world!
La frase "Hola mundo ¿cómo están? espero que bien!" tiene: 15 vocales.

C:\RMI>
```

(Nótese que las rutas de los archivos, difieren en el usuario, porque las máquinas son diferentes)

Conclusión.

En este proyecto, se implementó un sistema de producción concurrente con dos brazos de ensamblaje y dos contenedores de piezas. Se aplicó la exclusión mutua para garantizar un acceso seguro a los recursos compartidos. El sistema demostró coordinación efectiva entre los brazos, evitando condiciones de carrera y asegurando la integridad de los recursos compartidos. Este proyecto ilustra la aplicación práctica de conceptos clave de programación concurrente en un entorno de producción.

Referencias:

- [1] ... [Blackboard Learn](#) / contenido / practica 2 HelloInterface.java
- [2] ... [Blackboard Learn](#) / contenido / practica 2 Hello.java
- [3] ... [Blackboard Learn](#) / contenido / practica 2 HelloClient.java
- [4] ... [Blackboard Learn](#) / contenido / practica 2 HelloServer.java
- [5] ... [Blackboard Learn](#)