



Universidad
Isabel I



Estudio de caso UD5

Java RMI

Alumno Pablo Enrique Guntín Garrido

Asignatura Programación Concurrente y Distribuida

Grado en Ingeniería Informática

curso 2024-2025



Índice

1. Enunciado	3
1.1 Servidor Count.....	3
1.2 Especificación del programa.....	3
1.3 Desarrollo	3
1.4 Cuestiones	3
2. Descripción del problema: contexto del uso de RMI	5
3. Descripción de la funcionalidad.....	5
2.1 Incrementar un contador remoto (CountRMI)	5
2.2 Consultar una web y obtener enlaces (WebRMI).....	6
2.3 Comprobar si un número es primo o no (PrimeRMI)	7
4. Explicación y conclusiones	8
5. Bibliografía	9

1. Enunciado

1.1 Servidor Count

Con el objetivo de evaluar el desempeño de RMI se desea implementar un servidor simple llamado count.

El servidor exporta un método que se llama **increment()** el cual solamente incrementa en 1 el valor de una variable de tipo int llamada **sum** y regresa el resultado al cliente.

1.2 Especificación del programa

A continuación, se mostrarán las especificaciones del programa:

1. Inicializar a cero el valor de **sum**.
2. Invocar 1000 veces el método **increment()**.
3. Desplegar el valor final de **sum** y el tiempo promedio de respuesta.

1.3 Desarrollo

Implementación de la interfaz:

// Interfaz CountRmi

```
public interface CountRMI extends java.rmi.Remote
{
    int sum() throws java.rmi.RemoteException;
    void sum(int _val) throws java.rmi.RemoteException;
    public int increment() throws java.rmi.RemoteException;
}
```

Implementación parcial del servidor:

```
public static void main (String args[])
{
    // Create CountRMIIImpl
    CountRMIIImpl myCount = new CountRMIIImpl("myCountRMI");
    System.out.println("CountRMI Server ready.");
}
```

1.4 Cuestiones

1. Crea una solución RMI que ofrezca la siguiente interfaz e implemente un objeto remoto para dicha funcionalidad:

- *consultarWeb*: Permite obtener el código fuente de una web y guardar el código en un archivo de texto.
- *borrarDatos*: Permite borrar el archivo descargado.
- *consultarWebLinks*: Dada una URL permite obtener todos los enlaces de esa web.

2. Realizar la comprobación de si un número es primo o no mediante Java RMI

Aspectos a tener en cuenta:

- Todas las discusiones y valoraciones personales tienen que estar

justificadas.

- Se elaborará un **informe** que contenga la descripción del problema, y un apartado por cada uno de los puntos expuestos.
- El informe entregable final tiene que estar bajo el formato **pdf**.
- Se debe entregar el código fuente.
- El archivo entregable debe ser .zip/.rar
- La extensión máxima es de 10 folios.

2. Descripción del problema: contexto del uso de RMI

Muchas aplicaciones informáticas requieren acceso remoto a recursos distribuidos, sobre todo cuando se manejan grandes volúmenes de datos o se trata con miles de usuarios al mismo tiempo, como hemos visto en esta unidad.

En sistemas distribuidos es normal realizar operaciones de una manera eficiente en distintas máquinas o servidores. Para satisfacer estas necesidades se utilizan tecnologías distribuidas y, una de las más comunes e importantes en Java, es Java Remote Method Invocation (RMI), ya que permite a los objetos de una aplicación invocar métodos en otros objetos ubicados en diferentes máquinas dentro de una red, lo que facilita la creación de sistemas distribuidos sin tener que lidiar con la complejidad de la red.

Para este estudio de caso se implemente un sistema distribuido básico utilizando Java RMI. Un cliente invoca métodos remotos que estos son ejecutados por un servidor que procesa las solicitudes y devuelve los resultados. El servidor ofrece métodos simples como *increment* (para contar), *consultarWeb* (para obtener el código fuente de una web) y *isPrime* (saber si un número es primo o no). Para esto, hay tres tipos de clientes: *CountRMIClient*, *WebRMIClient*, *PrimeRMIClient*.

Utilizar RMI proporciona ventajas como la simplicidad en la comunicación entre aplicaciones distribuidas, transparencia, escalabilidad, facilidad para la implementación de sistemas tolerantes a fallos y la reducción de latencia. [1][2]

3. Descripción de la funcionalidad

2.1 Incrementar un contador remoto (CountRMI)

El servidor implementa un contador remoto que puede ser incrementado por un cliente. Su funcionalidad es permitir que el cliente llame al método *increment()* de manera remota para incrementar un valor entero *sum* en el server. La función *sum()* permite recuperar el valor actual de *sum*.

Su interfaz define los métodos que pueden ser llamados por el cliente remotamente: *int sum()* (devuelve el valor actual de *sum*), *void sum(int _val)* (establece el valor de *sum* a un valor) y *int increment()* (+1 a *sum* y devuelve el valor).

```
public interface CountRMI extends Remote {  
    int sum() throws RemoteException;  
    void sum(int _val) throws RemoteException;  
    int increment() throws RemoteException;  
}
```

Para la implementación se utiliza la clase *CountRMIImpI* que implementa la interfaz anterior y proporciona la lógica para poder incrementar *sum*. Como antes, el método *increment()* aumenta el valor de *sum* en +1.

El cliente, *CountRMIClient*, llama al método *increment()* 1000 veces y mide el tiempo que tarda en realizar todas las llamadas.

Se muestra el valor de *sum* y el tiempo promedio por operación:

```

1 package client;
2
3 import java.rmi.registry.LocateRegistry;
4
5
6
7
8
9 public class CountRMIClient {
10     public static void main(String[] args) {
11         try {
12             //Para localizar el registro RMI

```

Valor final de sum: 2000
Tiempo total: 83 ms
Tiempo promedio por operacion: 0.083 ms

Imagen 1. Resultado de Count.

2.2 Consultar una web y obtener enlaces (WebRMI)

El servidor proporciona metadatos para interactuar con las webs. El cliente solicita el código fuente de la página web, guarda su contenido en un archivo y así obtiene los enlaces de la página. Una vez finalizado, elimina los archivos descargados.

En la interfaz, *WebRMI*, se definen tres métodos con los que se interactúa con el server y así manipular las páginas web: *String consultarWeb(String url)* (coge el contenido HTML y lo guarda en un archivo de texto), *boolean borrarDatos(String fileName)* (elimina el archivo) y *tList<String> consultarWebLinks(String url)* (coge los enlaces de la web).

```

public interface WebRMI extends Remote {

    String consultarWeb(String url) throws RemoteException;

    boolean borrarDatos(String fileName) throws RemoteException;

    List<String> consultarWebLinks(String url) throws RemoteException;

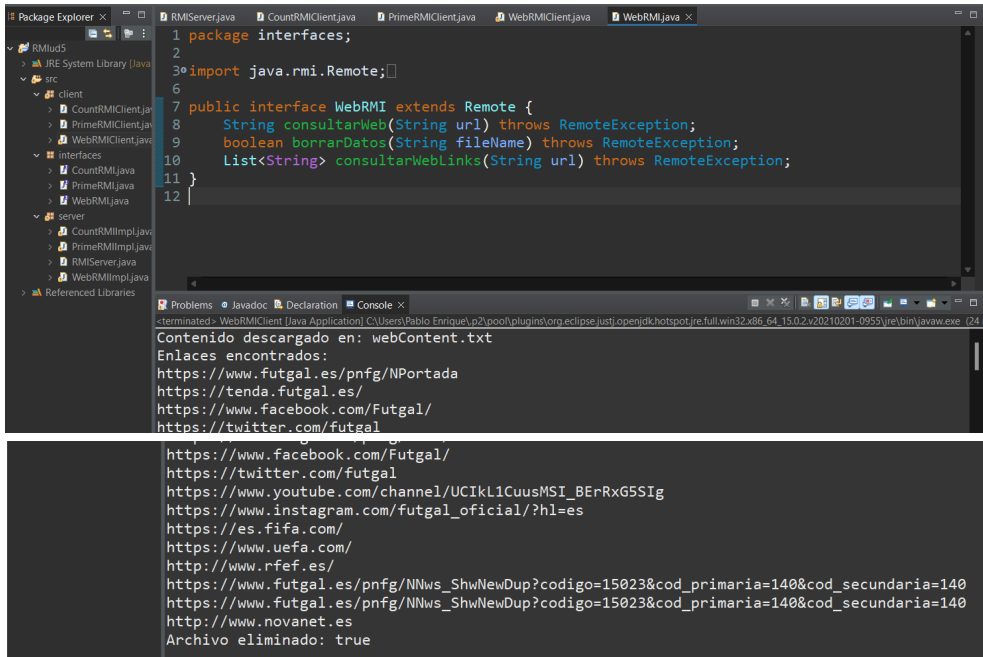
}

```

En la clase *WebRMIImpl* se implementan los métodos que se definieron en *WebRMI*, con el uso de Jsoup (<https://jsoup.org/download>) para conseguir el contenido de la página web.

Jsoup es una biblioteca de análisis de HTML y extracción de datos para Java. Permite parsear y manipular documentos HTML, extrayendo información específica y modificando el contenido.

El cliente, *WebRMIClient*, consulta una página web, coge el código fuente con los enlaces y después elimina el archivo descargado:



Imágenes 2 y 3. Resultados de Web.

2.3 Comprobar si un número es primo o no (PrimeRMI)

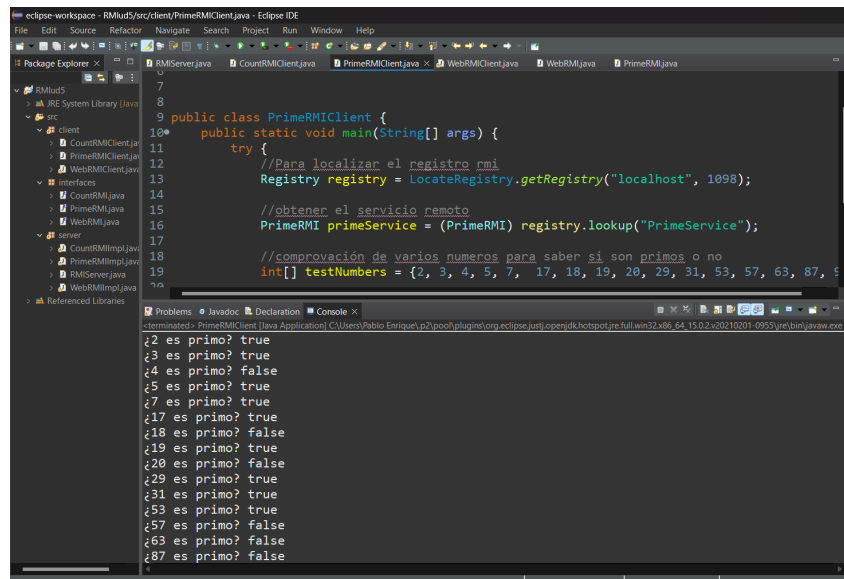
El servidor permite al cliente comprobar si un número es primo o no. El cliente llama al método *isPrime(int number)* y el servidor devuelve un valor booleano para decir si es primo o no.

Su interfaz, *PrimeRMI*, define un método remoto para verificar si es primo: *boolean isPrime(int number)* verifica si un número es primo o no.

```
public interface PrimeRMI extends Remote {
    boolean isPrime(int number) throws RemoteException;
}
```

La clase *PrimeRMIImp* implementa el método *isPrime* para realizar la comprobación de si es primo o no.

La clase *PrimeRMIClient* prueba si varios números enteros son clientes o no:



```

7
8
9 public class PrimeRMIClient {
10     public static void main(String[] args) {
11         try {
12             //Para localizar el registro rmi
13             Registry registry = LocateRegistry.getRegistry("localhost", 1098);
14
15             //obtener el servicio remoto
16             PrimeRMI primeService = (PrimeRMI) registry.lookup("PrimeService");
17
18             //comprovación de varios numeros para saber si son primos o no
19             int[] testNumbers = {2, 3, 4, 5, 7, 17, 18, 19, 20, 29, 31, 53, 57, 63, 87, 9
20
21
22 es primo? true
23 es primo? true
24 es primo? false
25 es primo? true
26 es primo? true
27 es primo? true
28 es primo? true
29 es primo? false
30 es primo? true
31 es primo? true
32 es primo? true
33 es primo? true
34 es primo? false
35 es primo? false
36 es primo? false
37 es primo? false
38 es primo? false
39 es primo? false
40 es primo? false
41 es primo? false
42 es primo? false
43 es primo? false
44 es primo? false
45 es primo? false
46 es primo? false
47 es primo? false
48 es primo? false
49 es primo? false
50 es primo? false
51 es primo? false
52 es primo? false
53 es primo? true
54 es primo? true
55 es primo? true
56 es primo? true
57 es primo? true
58 es primo? true
59 es primo? true
60 es primo? true
61 es primo? true
62 es primo? true
63 es primo? true
64 es primo? true
65 es primo? true
66 es primo? true
67 es primo? true
68 es primo? true
69 es primo? true
70 es primo? true
71 es primo? true
72 es primo? true
73 es primo? true
74 es primo? true
75 es primo? true
76 es primo? true
77 es primo? true
78 es primo? true
79 es primo? true
80 es primo? true
81 es primo? true
82 es primo? true
83 es primo? true
84 es primo? true
85 es primo? true
86 es primo? true
87 es primo? true
88 es primo? true
89 es primo? true
90 es primo? true
91 es primo? true
92 es primo? true
93 es primo? true
94 es primo? true
95 es primo? true
96 es primo? true
97 es primo? true
98 es primo? true
99 es primo? true
100 es primo? true

```

Imagen 4. Resultados de número primo.

4. Explicación y conclusiones

Para poder ejecutar todo esto, a parte de realizar el código, previamente tuve que añadirle Jsoup, que ya expliqué su funcionalidad en el apartado 2.2 de forma resumida. Para ello, descargué el el archivo .jar de su página principal (<https://jsoup.org/download>) y lo añadí a mi proyecto de la siguiente forma: click derecho en mi proyecto → Build Path → Add External Archives... y ahí seleccioné el archivo .jar para añadirlo. Con esto es suficiente.

Para su ejecución, para poder ejecutar el programa, ejecuté los siguientes comandos desde el archivo bin de mi proyecto:

```

PS C:\Users\Pablo Enrique\eclipse-workspace\RMiud5> cd bin
PS C:\Users\Pablo Enrique\eclipse-workspace\RMiud5\bin> rmiregistry

```

El comando *rmiregistry* es una herramienta de JDK para inciar el registro RMI. Actúa como una especie de directorio y su función principal es asociar nombres con objetos remotos.

Una vez hechos los pasos previos, desde Eclipse ejecuto el servidor RMI y ya puedo ejecutar uno a uno los clientes para obtener los resultados solicitados de la práctica.

5. Bibliografía

- [1] "RMI: Remote Method Incocation (Invocacion Remota de Métodos)", *Profesores UTFSM*. Online. Disponible:
<http://profesores.elo.utfsm.cl/~agv/elo330/2s09/lectures/RMI/RMI.html>
- [2] R. Santamaría, "Java RMI", *USAL*. Online. Disponible:
<http://vis.usal.es/rodrigo/documentos/sisdis/seminarios/JavaRMI.pdf>
- [3] "Ejemplo simple de RMI". Online. Disponible: <https://old.chuidiang.org/java/rmi/rmi.php>