



ETSIIT

Escuela Técnica Superior
de Ingenierías Informática
y de Telecomunicación



**UNIVERSIDAD
DE GRANADA**

Desarrollo de Sistemas Distribuidos

AUTOR: Jaime Parra Jimenéz

ÍNDICE

ÍNDICE	2
1. INTRODUCCIÓN	3
2. EJEMPLOS	3
2.1 Ejemplo 1	3
2.2 Ejemplo 2	4
2.3 Ejemplo 3	5
3. EJERCICIO	7

1.INTRODUCCIÓN

Esta práctica consiste en hacer un programa basado en la iteración cliente-servidor utilizando RMI. RMI es una característica de Java que permite que los programas distribuidos se comuniquen entre sí a través de la red. Primero he lanzado tres ejemplos propuestos y un ejercicio a resolver. A continuación se explicará en qué consisten.

2.EJEMPLOS

2.1 Ejemplo 1

Este ejemplo consiste en una aplicación cliente-servidor. El programa recibe una petición de un cliente y lo imprime por pantalla. Si el argumento es 0 deberá esperar antes de volver a imprimir el mensaje.

Para la ejecución del programa he usado un script donde compilo los archivos, lanzo el servidor y llamo a un cliente con un id 0 y a otro cliente con un id 1.

```
Lanzando el ligador de RMI...
Compilando con javac...

Lanzando el servidor...
Ejemplo bound

Lanzando el primer cliente

Buscando el objeto remoto
Invocando el objeto remoto
Recibida peticion de proceso: 0
Empezamos a dormir
Terminamos de dormir

Hebra 0

Lanzando el segundo cliente

Buscando el objeto remoto
Invocando el objeto remoto
Recibida peticion de proceso: 1

Hebra 1
```

Como podemos ver si pasamos una id de 0 el programa se duerme durante 5 segundos mientras que si se pasa otro id se imprime la hebra al instante.

2.2 Ejemplo 2

Este ejemplo es similar al anterior, pero en lugar de haber varios clientes, se crean múltiples hebras que realizan la misma tarea de imprimir un mensaje.

Para la ejecución del programa he usado un script donde compilo los archivos, lanzo el servidor y llamo al cliente.

Primero el programa será lanzado sin usar el modificador synchronized por lo que las hebras se imprimirán desordenadamente.

```
Lanzando el ligador de RMI...
Compilando con javac...

Lanzando el servidor...
Ejemplo bound

Lanzando el cliente

Buscando el objeto remoto
Buscando el objeto remoto
Buscando el objeto remoto
Buscando el objeto remoto
Buscando el objeto remoto
Invocando el objeto remoto
Invocando el objeto remoto
Invocando el objeto remoto
Invocando el objeto remoto

Entra Hebra Cliente 3

Entra Hebra Cliente 1

Entra Hebra Cliente 2

Entra Hebra Cliente 0
Sale Hebra Cliente 3
Sale Hebra Cliente 1
Sale Hebra Cliente 2
Empezamos a dormir
Invocando el objeto remoto

Entra Hebra Cliente 4
Sale Hebra Cliente 4
Terminamos de dormir
Sale Hebra Cliente 0
```

Como podemos no se sigue un orden, sino que entran y salen varios a la vez. También podemos observar que si el id es 0 la hebra se duerme durante 5 segundos.

A continuación se va a usar el modificador `synchronized` lo que permite que si entre la hebra 4 hasta que esta no salga no podrá entrar otra hebra.

```
Lanzando el ligador de RMI...
Compilando con javac...

Lanzando el servidor...
Ejemplo bound

Lanzando el cliente

Buscando el objeto remoto
Buscando el objeto remoto
Buscando el objeto remoto
Buscando el objeto remoto
Buscando el objeto remoto
Invocando el objeto remoto
Invocando el objeto remoto
Invocando el objeto remoto
Invocando el objeto remoto

Entra Hebra Cliente 1
Sale Hebra Cliente 1

Entra Hebra Cliente 3
Sale Hebra Cliente 3

Entra Hebra Cliente 4
Sale Hebra Cliente 4

Entra Hebra Cliente 2
Sale Hebra Cliente 2
Invocando el objeto remoto

Entra Hebra Cliente 0
Empezamos a dormir
Terminamos de dormir
Sale Hebra Cliente 0
```

Como podemos observar al usar `synchronized` se sigue un orden en la entrada y salida de las hebras. De esta manera podemos observar que los mensajes no se entrelazan proporcionando más limpieza a la hora de imprimir los mensajes.

2.3 Ejemplo 3

En este ejemplo se crea por una parte el objeto remoto y por otra el servidor. El objeto remoto consta con las funciones a las que se accede remotamente y el servidor exporta los métodos contenidos en la interfaz.

Primero se define `IContador` donde se define la interfaz que especifica los métodos que pueden ser invocados de manera remota. A continuación tenemos

Contador el cual implementa la interfaz IContador y donde implementamos los métodos de la interfaz.

Después tenemos el servidor donde se crea una instancia del objeto Contador y cliente el cual interactúa con el servidor. En el cliente se invocará el método incrementar donde se incrementa el contador 1000 veces.

Para la ejecución de este ejemplo primero debemos compilar los archivos de java. Para compilarlos ponemos en la terminal el siguiente comando:

```
javac *.java
```

A continuación lanzamos el servidor con el siguiente comando:

```
java -cp . -Djava.rmi.server.codebase=file:./  
-Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy  
servidor
```

Y por último lanzamos el cliente en otra terminal con el siguiente comando:

```
java -cp . -Djava.security.policy=server.policy cliente
```

Cuando lanzamos el servidor en la terminal nos aparece lo siguiente:

```
jaine@jaine-MSI:~/Escritorio/3AÑO/2cuatri/DSD/P3-DSD/Ejemplo3$ java -cp . -Djava.rmi.server.codebase=  
file:./ -Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy servidor  
Servidor RemoteException | MalformedURLExceptionondor preparado  
□
```

Y cuando lanzamos el cliente nos aparece esto:

```
jaine@jaine-MSI:~/Escritorio/3AÑO/2cuatri/DSD/P3-DSD/Ejemplo3$ java -cp . -Djava  
.security.policy=server.policy cliente  
Poniendo contador a 0  
Incrementando...  
Media de las RMI realizadas = 0.07 msecs  
RMI realizadas = 1000  
jaine@jaine-MSI:~/Escritorio/3AÑO/2cuatri/DSD/P3-DSD/Ejemplo3$
```

Como podemos observar en el cliente se realiza la incrementación correctamente.

3.EJERCICIO

Este ejercicio consiste en desarrollar un sistema cliente-servidor utilizando RMI. Para empezar he implementado una Interface donde declaro los métodos que puede hacer el servidor. A continuación implemento la clase ServerReplica donde implemento la interface. En esta clase implemento todos los métodos declarados en la interface.

Estos métodos son los siguientes:

- void Registrar(String clientId): Este método permite al usuario registrarse en un servidor.

```
@Override
public synchronized void Registrar(String clientId) throws RemoteException{
    if (!clientesRegistrados.contains(clientId)) {
        clientesRegistrados.add(clientId);
        System.out.println("Cliente " + clientId + " registrado en el servidor.");
    } else {
        System.out.println("El cliente " + clientId + " ya está registrado en el servidor.");
    }
}
```

- void Donar(String clientId, double cantidad): Este método permite al usuario donar la cantidad que él quiera.

```
@Override
public synchronized void Donar(String clientId, double cantidad) throws RemoteException{
    if (clientesRegistrados.contains(clientId) && cantidad>0) {
        donaciones.put(clientId, donaciones.getDefault(clientId, 0.0) + cantidad);
        System.out.println("Depósito de " + cantidad + " realizado por el cliente " + clientId + ".");
    } else {
        System.out.println("El cliente " + clientId + " no está registrado en el servidor.");
    }
}
```

- String getDonantes(): Muestra los donantes registrados en el servidor.

```
@Override
public synchronized String getDonantes() throws RemoteException{
    StringBuilder donantes = new StringBuilder("Lista de donantes en este servidor:\n");
    for (String clientId : donaciones.keySet()) {
        donantes.append(clientId).append("\n");
    }
    return donantes.toString();
}
```

- double TotalDonado(): Muestra la cantidad de lo donado en todos los servidores.

```
@Override
public synchronized double getTotalDonado() throws RemoteException{
    double Donado = 0.0;
    for (double cantidad : donaciones.values()) {
        Donado += cantidad;
    }
    return Donado;
}
```

- String getDonacionesCliente(String clientId): Este método permite ver la cantidad total donada de un usuario.

```
@Override
public synchronized String getDonacionesCliente(String clientId) throws RemoteException{
    StringBuilder donacionesCliente = new StringBuilder("Donaciones del cliente " + clientId + ":\n");
    if (donaciones.containsKey(clientId)) {
        donacionesCliente.append(donaciones.get(clientId)).append("\n");
    } else {
        donacionesCliente.append("Este cliente no tiene donaciones registradas.\n");
    }
    return donacionesCliente.toString();
}
```

- boolean EstaRegistrado(String clientId): Muestra si un cliente está registrado en el servidor.

```
@Override
public synchronized boolean EstaRegistrado(String clientId) throws RemoteException {
    return clientesRegistrados.contains(clientId);
}
```

Una vez implementada la clase del servidor réplica implemento la clase del servidor principal. En ella declaro dos servidores réplica y creo un registro RMI en el puerto 1099. A continuación, registro los objetos remotos de Replica1 y Replica2 en el registro RMI para que el cliente tenga acceso al registro RMI.

Por último, implemento la clase Cliente donde busco el registro RMI donde está registrado los objetos remotos de Replica1 y Replica2. Seguidamente, creo un menú de opciones donde se muestra al usuario un conjunto de opciones donde se espera que el usuario seleccione una. Una vez seleccionada, se procesa las opciones donde cada una realiza una acción diferente.

- Opción 1: El usuario puede registrarse en uno de los servidores proporcionando un ID de cliente.
- Opción 2: El usuario puede hacer una donación proporcionando un ID de cliente y una cantidad.
- Opción 3: Se muestra el total donado en ambas réplicas.
- Opción 4: Se muestra una lista de donantes en ambas réplicas.
- Opción 5: Se muestra el total donado de un cliente específico en la réplica donde está registrado.
- Opción 6: Se verifica si un cliente está registrado en una réplica específica.
- Opción 7: Termina el programa.

Para la compilación del programa deberemos poner en la terminal el siguiente comando:

```
javac *.java
```

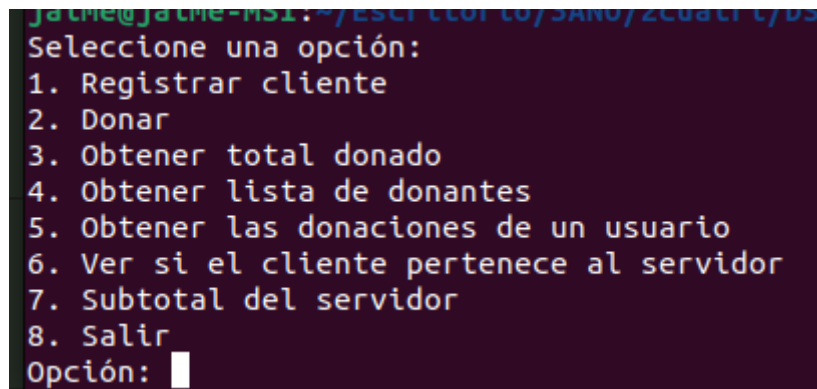
A continuación deberemos lanzar los servidores. Para ello ejecutamos el siguiente comando:

```
java -cp . MainServer
```

Ahora abrimos otra terminal donde se lanzará el cliente:

```
java Cliente
```

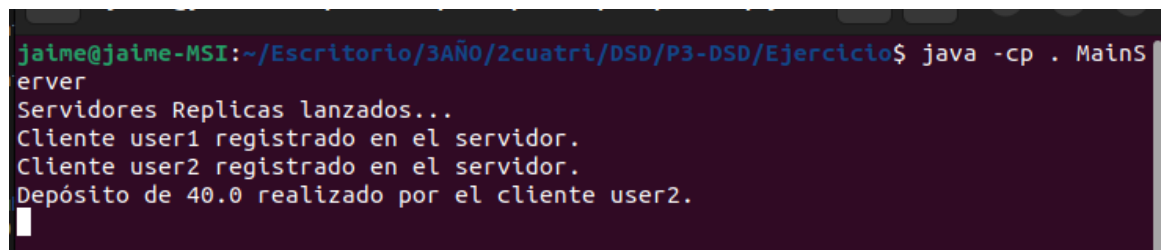
Una vez lanzado el cliente se mostrará en la terminal el siguiente menú:



```
jai@jame-MSI:~/Escritorio/3AÑO/2cuatri/DSD/P3-DSD/Ejercicio$ java Cliente
Seleccione una opción:
1. Registrar cliente
2. Donar
3. Obtener total donado
4. Obtener lista de donantes
5. Obtener las donaciones de un usuario
6. Ver si el cliente pertenece al servidor
7. Subtotal del servidor
8. Salir
Opción: █
```

Lo primero que debemos hacer es registrar un cliente en un servidor para poder donar. Si el cliente no está registrado no será posible donar.

En la terminal donde se ha lanzado el servidor se pueden ver mensajes donde se confirma si el servidor ha sido registrado o no o si ha donado y la cantidad que ha donado.



```
jai@jame-MSI:~/Escritorio/3AÑO/2cuatri/DSD/P3-DSD/Ejercicio$ java -cp . MainServer
Servidores Replicas lanzados...
Cliente user1 registrado en el servidor.
Cliente user2 registrado en el servidor.
Depósito de 40.0 realizado por el cliente user2.
█
```

En esta imagen se puede apreciar cómo se va mostrando en la terminal lo que el cliente va haciendo en el servidor.