

# Memoria P3

Raúl Durán Racero

Mayo 2022

## 1. Introducción

En esta práctica aprenderemos como funciona RMI, una de las formas que ofrece Java para trabajar con objetos distribuidos. Una de las características de RMI es su facilidad de uso al estar diseñado específicamente en Java. Para poder invocar métodos de manera remota a través de RMI, un programa Java exporta un objeto, el cual estará accesible a través de la red, permaneciendo el programa a la espera de peticiones en un puerto TCP. Desde ese momento, un cliente puede conectarse e invocar los métodos proporcionados por el objeto. Nuestro objetivo en esta práctica es aprender a diseñar y programar aplicaciones Cliente-Servidor desarrollando en RMI un sistema cliente-servidor teniendo en cuenta ciertos requisitos.

## 2. Ejemplos

Antes de realizar el ejercicio, se nos ofrecen varios ejemplos para entender mejor como trabajar con RMI. En los 3 ejemplos, veremos que se repite el proceso de desarrollo RMI:

- Definimos e implementamos una interfaz remota.
- Compilamos la clase del objeto remoto con *javac*.
- Implementamos el gestor de seguridad.
- Lanzamos el ligador RMI en el servidor con *rmiregistry*.
- Lanzamos el objeto remoto.
- Registramos el objeto remoto en el ligador.
- Escribimos el código del cliente.
- Compilamos el código del cliente con *javac*.
- Lanzamos el cliente, cargando las clases del cliente y sus *stubs*.

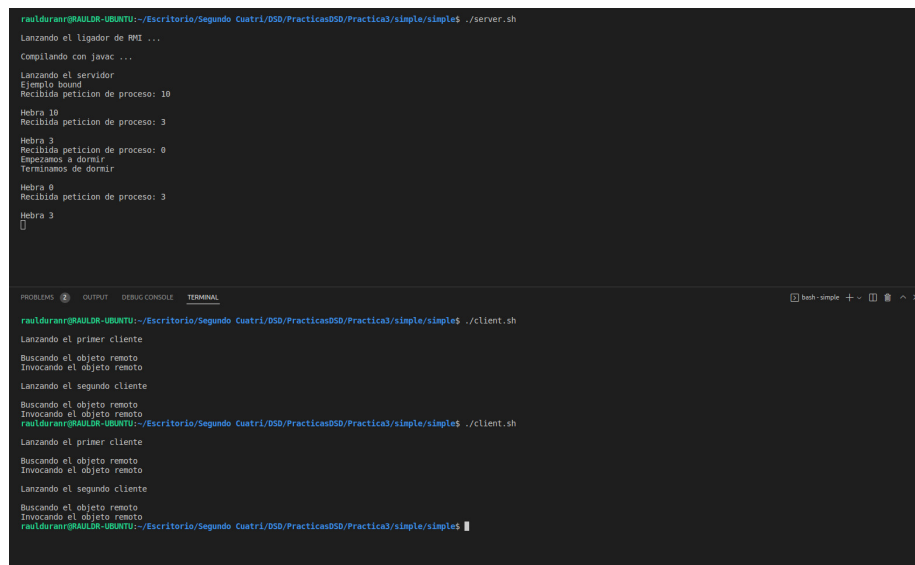
Disponemos además de una macro que nos permite compilar y ejecutar en una única máquina los ejemplos, siempre y cuando todos los archivos \*.java y el archivo .policy (que contiene todas las políticas de seguridad necesarias) estén en el mismo directorio que la macro. Ya solo nos falta implementar cada ejemplo y comprobar su funcionamiento.

## 2.1. Ejemplo 1

En este primer ejemplo el servidor *Ejemplo.java* exporta los métodos contenidos en la interfaz *Ejemplo\_I.java* del objeto remoto instanciado como *Ejemplo\_I.java*. Cuando el programa servidor recibe una petición de un cliente, lo que hace es imprimir el argumento que recibe de la llamada. Si dicho argumento es un "0", entonces se espera un tiempo antes de volver a imprimir el mensaje.

El cliente se encarga de activar el gestor de seguridad, buscar el objeto remoto e invocar el método que imprimirá el argumento por pantalla.

Podemos ver como funciona:



```
rauldurant@RAULDR-UBUNTU:~/Escritorio/Segundo Cuatri/OSD/PracticasOSD/Practicas/simple/simple$ ./server.sh
Lanzando el ligador de RMI ...
Compilando con javac ...
Lanzando el servidor
Ejemplo bound
Recibida peticion de proceso: 10
Hebra 10
Recibida peticion de proceso: 3
Hebra 3
Recibida peticion de proceso: 0
Esperamos a dormir
Terminamos de dormir
Hebra 0
Recibida peticion de proceso: 3
Hebra 3
[]

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
rauldurant@RAULDR-UBUNTU:~/Escritorio/Segundo Cuatri/OSD/PracticasOSD/Practicas/simple/simple$ ./client.sh
Lanzando el primer cliente
Buscando el objeto remoto
Invocando el objeto remoto
Lanzando el segundo cliente
Buscando el objeto remoto
Invocando el objeto remoto
rauldurant@RAULDR-UBUNTU:~/Escritorio/Segundo Cuatri/OSD/PracticasOSD/Practicas/simple/simple$ ./client.sh
Lanzando el primer cliente
Buscando el objeto remoto
Invocando el objeto remoto
Lanzando el segundo cliente
Buscando el objeto remoto
Invocando el objeto remoto
rauldurant@RAULDR-UBUNTU:~/Escritorio/Segundo Cuatri/OSD/PracticasOSD/Practicas/simple/simple$
```

Figura 1: Ejemplo 1

Como puede verse en la salida, una vez lanzado el servidor, cuando recibe un cliente imprime el número que pasa como argumento dicho cliente. Vemos como, si el argumento es 0, el servidor se pone a dormir 5 segundos, y al terminar de dormir vuelve a funcionar con normalidad.

## 2.2. Ejemplo 2

En este ejemplo en vez de lanzar varios clientes, crearemos varias hebras que realizan la misma tarea, imprimir un mensaje remoto accediendo al stub de un objeto remoto. Ahora, al lanzar el cliente el primer parámetro que se reciba será el nombre del host del servidor, y el segundo el número de hebras que queremos crear.

Si el nombre de una hebra acaba en 0, se pondrá a dormir 5 segundos. Si el número es 10, también dormirá una sola vez, ya que realmente ejecuta las hebras de la 0 a la 9, por lo que si queremos que duerma 2 veces, en la 0 y en la 10, tendrán que ser 11 hebras.

Además, las demás hebras entran sin seguir el orden lógico (podemos ver esto

en la Figura 2), y sin esperar a que haya salido la anterior, entrelazándose los mensajes.

```
raulduran@RAULDR-UBUNTU:~/Escritorio/Segundo Cuatri/DSO/PracticasDSO/Practica3/multihebras$ ./server.sh
Lanzando el ligador de PMI ...
Compilando con javac ...
Lanzando el servidor
Ejemplo bound
Entra Hebra Cliente 2
Entra Hebra Cliente 1
Sale Hebra Cliente 2
Sale Hebra Cliente 1
Entra Hebra Cliente 0
Empezamos a dormir
Terminamos de dormir
Sale Hebra Cliente 0

raulduran@RAULDR-UBUNTU:~/Escritorio/Segundo Cuatri/DSO/PracticasDSO/Practica3/multihebras$ ./client.sh
Lanzando el primer cliente

Lanzando el segundo cliente
Buscando el objeto remoto
Buscando el objeto remoto
Buscando el objeto remoto
Invocando el objeto remoto
Invocando el objeto remoto
Invocando el objeto remoto
raulduran@RAULDR-UBUNTU:~/Escritorio/Segundo Cuatri/DSO/PracticasDSO/Practica3/multihebras$
```

Figura 2: Ejemplo 2 asíncrono-1

```
raulduran@RAULDR-UBUNTU:~/Escritorio/Segundo Cuatri/DSO/PracticasDSO/Practica3/multihebras$ ./server.sh
Lanzando el ligador de PMI ...
Compilando con javac ...
Lanzando el servidor
Ejemplo bound
Entra Hebra Cliente 0
Empezamos a dormir
Entra Hebra Cliente 7
Sale Hebra Cliente 7
Entra Hebra Cliente 5
Sale Hebra Cliente 5
Entra Hebra Cliente 8
Sale Hebra Cliente 8
Entra Hebra Cliente 1
Sale Hebra Cliente 1
Entra Hebra Cliente 4
Sale Hebra Cliente 4
Entra Hebra Cliente 3
Sale Hebra Cliente 3
Entra Hebra Cliente 6
Entra Hebra Cliente 9
Sale Hebra Cliente 6
Sale Hebra Cliente 9
Entra Hebra Cliente 10
Empezamos a dormir
Entra Hebra Cliente 2
Sale Hebra Cliente 2

raulduran@RAULDR-UBUNTU:~/Escritorio/Segundo Cuatri/DSO/PracticasDSO/Practica3/multihebras$ ./client.sh
Lanzando el primer cliente
Buscando el objeto remoto
Buscando el objeto remoto
Buscando el objeto remoto
Buscando el objeto remoto
Buscando el objeto remoto
Buscando el objeto remoto
Buscando el objeto remoto
Buscando el objeto remoto
Buscando el objeto remoto
Buscando el objeto remoto
Buscando el objeto remoto
Invocando el objeto remoto
```

Figura 3: Ejemplo 2 asíncrono-2

Si añadimos el modificador **synchronized** en el método de la implementación remota, las diferencias en la ejecución del programa son que los mensajes no se entrelazan, respetando su turno, aunque siguen entrando sin seguir el orden lógico. Al respetar el turno, cuando una hebra que acaba en 0 entra y se pone a

dormir, el resto espera a que despierte para seguir entrando, como puede verse en las siguientes imágenes.

```
raulduram@RAULDR-UBUNTU:~/Escritorio/Segundo Cuatri/DSD/PracticasDSD/Practica3/multihebras$ ./server.sh
Lanzando el ligador de PMI ...

Compilando con javac ...

Lanzando el servidor
Ejemplo bound

Entra Hebra Cliente 2
Sale Hebra Cliente 2

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

Entra Hebra Cliente 1
Sale Hebra Cliente 1
█
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

bash - multihebras

raulduram@RAULDR-UBUNTU:~/Escritorio/Segundo Cuatri/DSD/PracticasDSD/Practica3/multihebras\$ ./client.sh

Lanzando el primer cliente

Lanzando el segundo cliente

Buscando el objeto remoto

Buscando el objeto remoto

Buscando el objeto remoto

Invocando el objeto remoto

Invocando el objeto remoto

Invocando el objeto remoto

raulduram@RAULDR-UBUNTU:~/Escritorio/Segundo Cuatri/DSD/PracticasDSD/Practica3/multihebras\$

Figura 4: Ejemplo 2 synchronized-1

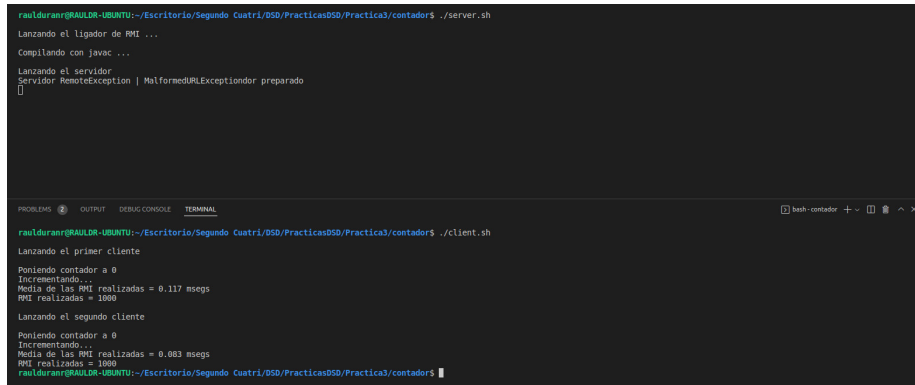
[illegible]

Figura 5: Ejemplo 2 synchronized-2

### 2.3. Ejemplo 3

En este ejemplo se crea por un lado el objeto remoto y por otro el servidor. El servidor exporta los métodos de la interfaz *icontador.java* del objeto remoto instanciado como *micontador* de la clase *contador.java*. El cliente se encarga de poner un valor inicial al contador, invoca al método para incrementarlo 1000 veces, e imprimir el valor final del contador junto al tiempo medio de respuesta.

que se calcula a partir de las invocaciones remotas del método.



```
rauldurant@RAULDR-UBUNTU:~/Escritorio/Segundo Cuatri/DSO/PracticasDSO/Practica3/contadors$ ./server.sh
Lanzando el ligador de RMI ...
Compilando con javac ...
Lanzando el servidor
Servidor RemoteException | MalformedURLException preparado
[]

rauldurant@RAULDR-UBUNTU:~/Escritorio/Segundo Cuatri/DSO/PracticasDSO/Practica3/contadors$ ./client.sh
Lanzando el primer cliente
Poniendo contador a 0
Incrementando...
Medio de las RMI realizadas = 0.117 msecs
RMI realizadas = 1000
Lanzando el segundo cliente
Poniendo contador a 0
Incrementando...
Medio de las RMI realizadas = 0.083 msecs
RMI realizadas = 1000
rauldurant@RAULDR-UBUNTU:~/Escritorio/Segundo Cuatri/DSO/PracticasDSO/Practica3/contadors$
```

Figura 6: Ejemplo 3

Hay que mencionar que ya no es necesario lanzar el ligador RMI desde el script del servidor, ya que lo estamos creando dentro de la clase *servidor.java*.

### 3. Ejercicio

En este ejercicio, tendremos que desarrollar en RMI un sistema cliente-servidor con los requisitos indicados en el guión de la práctica. Consiste básicamente en tener un sistema con 2 réplicas de un servidor, las cuales gestionan el registro y otras operaciones que le pidan los clientes.

Para hacer posible esta comunicación, necesitaremos 2 interfaces: una para el cliente y el servidor, y otra para las 2 réplicas, para permitir la comunicación entre ellas.

La estructura sería la siguiente:

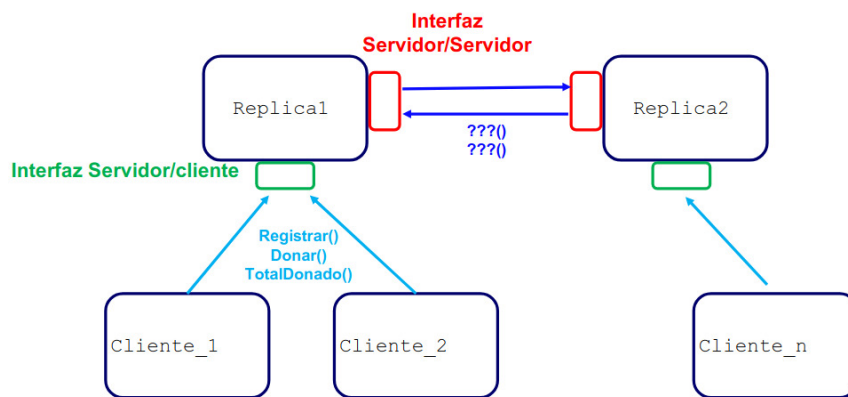


Figura 7: Esquema réplicas

Aquí vemos claramente como las réplicas tienen que implementar 2 interfaces, la que las comunica con los clientes y la que hace que se comuniquen entre ellas.

Una vez está implementada la réplica, hay que crear 2 de ellas desde el servidor, y añadir una a la otra, para que así puedan comunicarse entre ellas, como vemos en el siguiente fragmento de código:

```
replica replica1 = new replica("replica1");
replica replica2 = new replica("replica2");
reg.rebind("replica1", replica1);
reg.rebind("replica2", replica2);
replica1.addReplica(replica2);
replica2.addReplica(replica1);
```

Es momento de explicar los métodos de las réplicas.

Cuando un cliente quiere registrarse, llama al método de registrar de una réplica. Dentro del método es donde se comprueba en qué réplica se registrará al cliente, si no está ya registrado en alguna réplica. Al registrar, lo que hacemos es añadir el cliente en el ArrayList de clientes de la réplica, y añadir una entrada con el mismo índice al array de donaciones, con un valor inicial de 0.

A la hora de donar, la réplica comprueba si el cliente está registrado en ella o no. Si lo está, actualiza el valor del array de donaciones del cliente. Si no está

registrado en ella, llama al mismo método de la otra réplica. Además, se actualiza el valor del totalDonado en la réplica. Un punto importante es comprobar que la donación es mayor que 0, ya que si no el cliente podría sacar dinero del servidor.

Y para las consultas, si el cliente quiere ver su total donado, primero se comprueba que esté registrado en la réplica a la que ha consultado. En caso positivo, devuelve el valor correspondiente en el array de donaciones. Si el cliente o bien no está registrado en esa réplica, o bien aún no ha donado nada, devolverá -1. Para consultar el total donado en la réplica, no importa el cliente que llame al método, simplemente devuelve el valor totalDonado de dicha réplica.

Por último, comprobamos su correcto funcionamiento. Para ello, intentaremos registrar, hacer un par de donaciones (una que se pasa como argumento y otra fija de 100), y comprobar el total donado. En todo momento, a través de la salida de la terminal, comprobamos en qué réplica se ha realizado la operación correspondiente. Además, intentaremos registrar 2 veces a cada cliente, que consulte el total donado en la réplica donde no está registrado, que lo consulte antes de donar, y que done a la réplica donde no se ha registrado.

La salida correcta sería (con el código de las llamadas que hace cada cliente a la izquierda):

```

cliente> java cliente2 Qd mainDonado
// Crea el club para el cliente especificando el nombre del servidor
Registry mireg = LocateRegistry.getRegistry(args[0]);
//Por defecto, solicitarse a la replica 1
IServerClient servidor1 = (IServerClient) mireg.lookup("replica1");
IServerClient servidor2 = (IServerClient) mireg.lookup("replica2");
System.out.println("Registrando al cliente ...");
String cliente = args[1];
servidor1.registrar(cliente);
//Si intentamos que se registre otra vez, te dirá que no.
servidor1.registrar(cliente);
//Si intentamos consultar lo donado antes de donar, mostrará un mensaje de error.
servidor1.getTotalDonadoCliente(cliente);
servidor2.getTotalDonadoCliente(cliente);
//Hace una donacion
int donacion = Integer.parseInt(args[2]);
System.out.println("Añadiendo donación de "+donacion+" euros ...");
servidor1.donar(donacion, cliente);
servidor1.donar(100, cliente);
servidor2.donar(donacion, cliente);
System.out.println("Añadiendo donación en la replica 2");
servidor2.donar(donacion, cliente);
//Consulta el total donado
System.out.println("Total donado por el "+cliente+" en la replica: "+servidor1);
System.out.println("Total donado por el "+cliente+" en la replica: "+servidor2);
//Consulta el total donado en cada replica
System.out.println("Total donado en la replica 1: "+servidor1.getTotalDonadoRepl);
System.out.println("Total donado en la replica 2: "+servidor2.getTotalDonadoRepl);
} catch (NotBoundException | RemoteException e) {
    System.err.println("Excepción del sistema: " + e);
}

```

```

Cliente cliente2 registrado en la replica1
Numero de clientes en la replica1: 1
Error en replica1: El cliente ya está registrado.
Error en replica2: El cliente no ha realizado ninguna donación aún.
Error en replica2: Esta no es la réplica en la que está registrado el cliente
Donacion de 10 añadida en la replica del cliente1
Donacion de 100 añadida en la replica del cliente1
Donacion de 10 añadida en la replica del cliente1
Donacion total del cliente cliente1 en la replica1: 120
Error en replica2: Esta no es la réplica en la que está registrado el cliente1
Cliente cliente2 registrado en la replica2
Numero de clientes en la replica2: 1
Error en replica1: El cliente ya está registrado.
Error en replica2: El cliente2 no ha realizado ninguna donación aún.
Error en replica2: Esta no es la réplica en la que está registrado el cliente2
Donacion de 1000 añadida en la replica2 del cliente2
Donacion de 100 añadida en la replica2 del cliente2
Donacion de 1000 añadida en la replica2 del cliente2
Error en replica1: Esta no es la réplica en la que está registrado el cliente2
Donacion total del cliente cliente2 en la replica2: 2100

```

Figura 8: Salida 1 ejercicio

Podemos probar añadiendo un tercer cliente, y comprobar que se registre y done correctamente:

```
Lanzando el primer cliente

Registrando al cliente ...
Añadiendo donación de 10 euros ...
Añadiendo donación en la replica
Total donado por el cliente1 en la replica1: 120
Total donado por el cliente1 en la replica2: -1
Total donado en la réplica 1: 120
Total donado en la réplica 2: 0

Lanzando el segundo cliente

Registrando al cliente ...
Añadiendo donación de 1000 euros ...
Añadiendo donación en la replica
Total donado por el cliente2 en la replica1: -1
Total donado por el cliente2 en la replica2: 2100
Total donado en la réplica 1: 120
Total donado en la réplica 2: 2100

Lanzando el tercer cliente

Registrando al cliente ...
Añadiendo donación de 100 euros ...
Añadiendo donación en la replica
Total donado por el cliente3 en la replica1: 300
Total donado por el cliente3 en la replica2: -1
Total donado en la réplica 1: 420
Total donado en la réplica 2: 2100
```

Figura 9: Salida con 3 clientes

Una vez hecho de manera simple, pero comprobando que funciona correctamente, podemos hacer que funcione de manera interactiva. Para ello, tendremos que cambiar la clase cliente, y añadir funcionalidades a los métodos del servidor para que nos den retroalimentación por la parte del cliente. Un fragmento de la salida interactiva sería el siguiente:



```

rauldurant@RAULDR-UBUNTU:~/Escritorio/Segundo Cuatri/DSD/PracticasDSD/Practica3/Ejercicio$ ./client.sh
Lanzando el primer cliente
Bienvenido al servicio de donaciones de DSD, cliente1.
Eliga la réplica a la que quiere conectarse:
1.- Réplica 1
2.- Réplica 2
2
Seleccione la operación deseada:
1.- Registrarse
2.- Donar
3.- Mi Total Donado
4.- Total Donado réplica
Otro valor para salir.
1
Cliente cliente1 registrado en la replica2
Eliga la réplica a la que quiere conectarse:
1.- Réplica 1
2.- Réplica 2
1
Seleccione la operación deseada:
1.- Registrarse
2.- Donar
3.- Mi Total Donado
4.- Total Donado réplica
Otro valor para salir.
2
Introduzca la cantidad a donar:
100
cliente1 has donado 100 en la replica2
Eliga la réplica a la que quiere conectarse:
1.- Réplica 1
2.- Réplica 2
1
Seleccione la operación deseada:
1.- Registrarse
2.- Donar
3.- Mi Total Donado
4.- Total Donado réplica
Otro valor para salir.
3
Error en replica1 : Ésta no es la réplica en la que está registrado el cliente1

```

Figura 10: Salida interactiva 1

Hemos conseguido que funcione la parte básica de la práctica. Ahora tenemos libertad para extenderla como consideremos.

En mi caso, añadiré una operación que permita al cliente retirar dinero de sus donaciones (sin afectar a las de otros clientes), modificando el total donado de la réplica.

Podemos ver un ejemplo de la salida de esta operación:

```

2
Seleccione la operación deseada:
1.- Registrarse
2.- Donar
3.- Mi Total Donado
4.- Total Donado réplica
5.- Retirar dinero
Otro valor para salir.
2
Introduzca la cantidad a donar:

10
cliente1 has donado 10 en la replica1
Elija la réplica a la que quiere conectarse:
1.- Réplica 1
2.- Réplica 2
1
Seleccione la operación deseada:
1.- Registrarse
2.- Donar
3.- Mi Total Donado
4.- Total Donado réplica
5.- Retirar dinero
Otro valor para salir.
5
Introduzca la cantidad a sacar:

5
cliente1 has sacado 5 de la replica1
Elija la réplica a la que quiere conectarse:
1.- Réplica 1
2.- Réplica 2
1
Seleccione la operación deseada:
1.- Registrarse
2.- Donar
3.- Mi Total Donado
4.- Total Donado réplica
5.- Retirar dinero
Otro valor para salir.
5
Introduzca la cantidad a sacar:

10
Error: la cantidad que desea sacar es mayor que la que ha donado.
Elija la réplica a la que quiere conectarse:
1.- Réplica 1
2.- Réplica 2
□

```

Figura 11: Salida retirada dinero

Además, añadiré una tercera réplica, que se comunique de forma similar a las anteriores. Para comprobar mejor su correcto funcionamiento, usaré 5 clientes. Para usar 3 réplicas, hay que, de nuevo, hacer que se añadan entre ellas, de tal manera que cada réplica conoce a sus vecinas, no sólo a la siguiente, formando un anillo:

```

        replica replica1 = new replica("replica1");
        replica replica2 = new replica("replica2");
        replica replica3 = new replica("replica3");

        reg.rebind("replica1", replica1);
        reg.rebind("replica2", replica2);
        reg.rebind("replica3", replica3);

        replica1.addReplica(replica2);
        replica1.addReplica(replica3);

        replica2.addReplica(replica1);
        replica2.addReplica(replica3);

        replica3.addReplica(replica1);
        replica3.addReplica(replica2);

```

Como estamos añadiendo más de una réplica a cada una, necesitamos un Array-List de réplicas. Además, habrá que modificar los métodos de registrarse y donar de la réplica para que consulte el array de réplicas para saber si un cliente está registrado o no, y para mandarle la operación correspondiente.

Tal y como está implementada la réplica, podríamos utilizar n réplicas y seguiría funcionando correctamente.

En la parte del servidor también tenemos una salida, para tener retroalimentación por ambas partes en caso de que alguna salida falle al imprimirse por pantalla.

```

rauldurant@RAULDR-UBUNTU:~/Escritorio/Segundo Cuatri/DSD/Practi
Lanzando el ligador de RMI ...
Compilando con javac ...
Lanzando el servidor
Servidor RemoteException preparado
Cliente client1 registrado en la replica1
Número de clientes en la replica1: 1
Error en replica2: El cliente ya está registrado.
Error en replica3: El cliente ya está registrado.
Donacion de 100 añadida en la replica1 por parte del client1
Cliente client2 registrado en la replica2
Número de clientes en la replica2: 1

```

Figura 12: Salida del servidor con 3 replicas

```

Lanzando el primer cliente

Bienvenido al servicio de donaciones de DSD, clientel.

Elija la réplica a la que quiere conectarse:
1.- Réplica 1
2.- Réplica 2
3.- Réplica 3
1
Seleccione la operación deseada:
1.- Registrarse
2.- Donar
3.- Mi Total Donado
4.- Total Donado réplica
5.- Retirar dinero
Otro valor para salir.
1
Cliente clientel registrado en la replica1

Elija la réplica a la que quiere conectarse:
1.- Réplica 1
2.- Réplica 2
3.- Réplica 3
2
Seleccione la operación deseada:
1.- Registrarse
2.- Donar
3.- Mi Total Donado
4.- Total Donado réplica
5.- Retirar dinero
Otro valor para salir.
1
Error en replica2: El cliente ya está registrado.

Elija la réplica a la que quiere conectarse:
1.- Réplica 1
2.- Réplica 2
3.- Réplica 3
3
Seleccione la operación deseada:
1.- Registrarse
2.- Donar
3.- Mi Total Donado
4.- Total Donado réplica
5.- Retirar dinero
Otro valor para salir.
1
Error en replica3: El cliente ya está registrado.

Elija la réplica a la que quiere conectarse:
1.- Réplica 1
2.- Réplica 2
3.- Réplica 3

```

Figura 13: Salida al registrar