

Memoria Práctica 3: RMI

Primera parte: Ejecución de ejemplos

Ejemplo 1

Para ejecutar el primer ejemplo, he utilizado el script proporcionado en el guión de la práctica pero modificando algunas cosas:

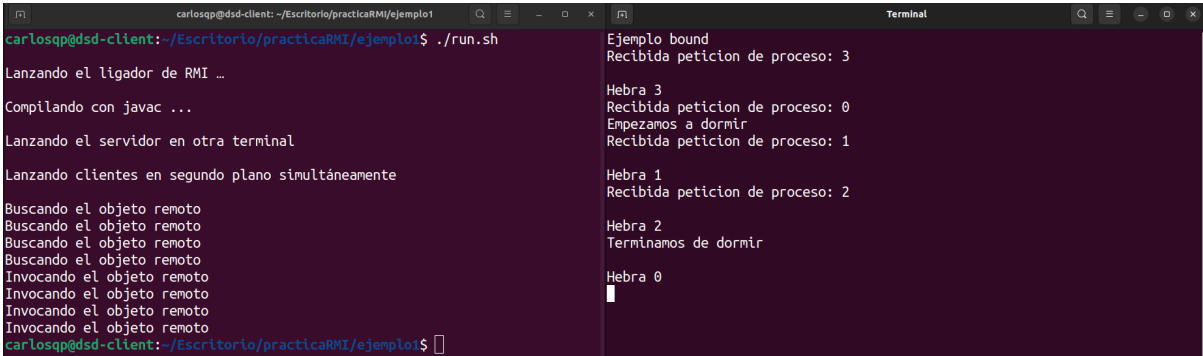
- He lanzado el proceso servidor en otra terminal distinta, en segundo plano, para poder observar las salidas al mismo tiempo que se ejecutan los clientes.

```
gnome-terminal --working-directory=$PWD -- java -cp .  
-Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost  
-Djava.security.policy=server.policy Ejemplo &
```

- He lanzado varios clientes simultáneamente (en segundo plano) para observar cómo recibía las peticiones el servidor

```
java -cp . -Djava.security.policy=server.policy Cliente_Ejemplo localhost 0 &  
java -cp . -Djava.security.policy=server.policy Cliente_Ejemplo localhost 1 &  
java -cp . -Djava.security.policy=server.policy Cliente_Ejemplo localhost 2 &  
java -cp . -Djava.security.policy=server.policy Cliente_Ejemplo localhost 3 &
```

Los resultados de la ejecución son los siguientes:



The screenshot shows two terminal windows. The left window, titled 'carlosqp@dss-client: ~/Escritorio/practicaRMI/ejemplo1', shows the execution of a script 'run.sh'. The script performs the following actions: 'Lanzando el ligador de RMI ...', 'Compilando con javac ...', 'Lanzando el servidor en otra terminal', and 'Lanzando clientes en segundo plano simultáneamente'. It then shows a series of 'Buscando el objeto remoto' and 'Invocando el objeto remoto' messages. The right window, titled 'Terminal', shows the output of the server and clients. The server output includes: 'Ejemplo bound', 'Recibida petición de proceso: 3', 'Hebra 3', 'Recibida petición de proceso: 0', 'Empezamos a dormir', 'Recibida petición de proceso: 1', 'Hebra 1', 'Recibida petición de proceso: 2', 'Hebra 2', 'Terminamos de dormir', and 'Hebra 0'. The client outputs are not visible in the screenshot.

A continuación, y tras analizar el código así como la salida, respondo a las cuestiones expuestas en clase:

¿Qué ocurre con las hebras cuyo nombre acaba en 0? Cuando se envía el número 0, la hebra se va a dormir; imprime: “Empezamos a dormir”. Cuando despierta, indica que se despierta e imprime el mensaje habitual “Hebra 0”

¿Qué hacen las demás hebras? Cada petición recibida por el servidor es tratada por una hebra distinta, por lo que aunque la hebra 0 mande a dormir al servidor, realmente solo manda a dormir a la hebra que lo ha recibido. Si se lanzan otros clientes durante ese

periodo de tiempo, el servidor va a seguir recibiendo peticiones e imprimiendo mensajes, puesto que las trataría con otras hebras.

¿Se entrelazan los mensajes? Sí; como se puede observar los mensajes se entrelazan. Esto ocurre por el mismo motivo expuesto anteriormente (una hebra lanzada por cada petición recibida).

Ejemplo 2: Cliente multihebra

Este ejemplo es similar al anterior, salvo que ahora en lugar de lanzar varias veces el programa cliente, es el programa cliente el que lanza un número N de hebras que se encargarán de realizar su petición de escribir mensaje al servidor. De nuevo he utilizado el mismo script pero con ligeras modificaciones (las mismas que para el primer ejemplo; lanzo el programa servidor en una terminal diferente, y esta vez sólo lanzo una vez el programa cliente).

```
gnome-terminal --working-directory=$PWD -- java -cp .
-Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost
-Djava.security.policy=server.policy Ejemplo &
```

```
java -cp . -Djava.security.policy=server.policy
Cliente_Ejemplo_Multi_Threaded localhost 5
```

Los resultados de la ejecución son los siguientes (para N=5 hebras):

```
carlosqp@dssd-client: ~/Escritorio/practicaRMI/ejemplo2
carlosqp@dssd-client:~/Escritorio/practicaRMI/ejemplo2$ ./run.sh
Lanzando el ligador de RMI ...
Compilando con javac ...
Lanzando el servidor en otra terminal
Lanzando el programa cliente multihebrado
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
Invocando el objeto remoto
Invocando el objeto remoto
Invocando el objeto remoto
Invocando el objeto remoto
carlosqp@dssd-client:~/Escritorio/practicaRMI/ejemplo2$

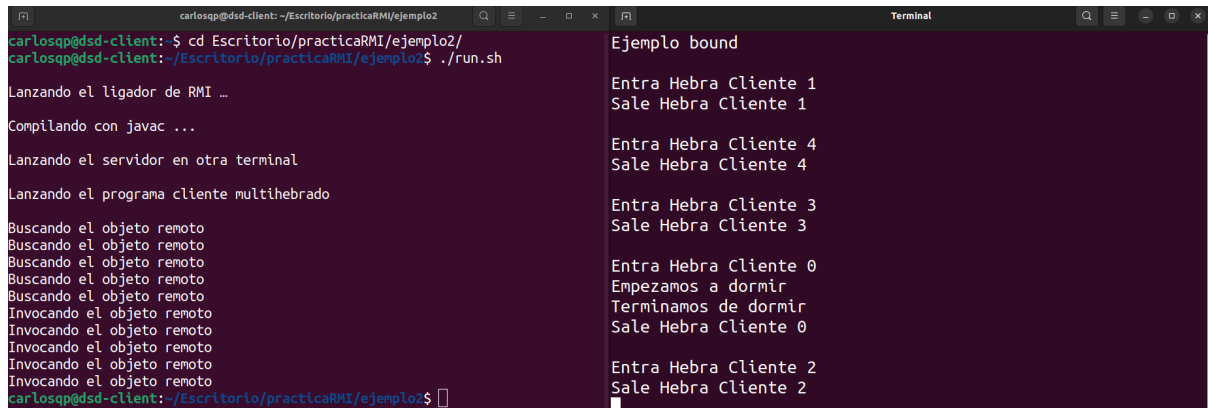
Ejemplo bound
Entra Hebra Cliente 2
Entra Hebra Cliente 4
Sale Hebra Cliente 2
Sale Hebra Cliente 4
Entra Hebra Cliente 0
Empezamos a dormir
Entra Hebra Cliente 1
Sale Hebra Cliente 1
Entra Hebra Cliente 3
Sale Hebra Cliente 3
Terminamos de dormir
Sale Hebra Cliente 0
```

Las respuestas a las tres preguntas expuestas en clase son las mismas que en el ejercicio anterior, salvo que esta vez, como el programa escribe cadenas en lugar de números, si la cadena termina en 0 (por ejemplo: Cliente 0, Cliente 10, Cliente 20...) entonces la hebra se dormirá 5 segundos. Por lo demás la respuesta es la misma.

Para completar el ejemplo, vamos a probar a ejecutarlo pero esta vez introduciendo el modificador `synchronized` en el método de la implementación remota:

```
public class Ejemplo implements Ejemplo_I {
    // ...
    public synchronized void escribir_mensaje (String mensaje) {
```

Esta vez los resultados son diferentes:



```

carlosqp@dss-client: ~/Escritorio/practicaRMI/ejemplo2
carlosqp@dss-client:~/Escritorio/practicaRMI/ejemplo2$ ./run.sh
Lanzando el ligador de RMI ...
Compilando con javac ...
Lanzando el servidor en otra terminal
Lanzando el programa cliente multihebrado
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
Invocando el objeto remoto
carlosqp@dss-client:~/Escritorio/practicaRMI/ejemplo2$

Ejemplo bound
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 0
Empezamos a dormir
Terminamos de dormir
Sale Hebra Cliente 0

Entra Hebra Cliente 2
Sale Hebra Cliente 2

```

Se puede observar que al añadir *synchronized*, ya no se entrelazan los mensajes y que cada petición se trata en exclusión mutua.

La palabra clave *synchronized* asegura que solo un hilo pueda acceder al método a la vez. Cuando un hilo invoca un método *synchronized* en un objeto remoto, adquiere un bloqueo en el objeto remoto y ningún otro hilo puede acceder a ese método *synchronized* hasta que el bloqueo es liberado por el hilo actual que lo posee.

Ejemplo 3

El tercer ejemplo es algo más complejo puesto que el cliente realiza 1000 llamadas a método remoto, y el programa servidor tiene el main en un archivo diferente a la clase que implementa la interfaz remota (en los dos ejemplos anteriores no era así). Además, en este caso es el propio servidor desde el programa el que lanza el registro RMI en lugar de tener nosotros que lanzarlo por terminal (para ello, ver la siguiente orden):

```
Registry reg=LocateRegistry.createRegistry(1099);
```

En este caso, especifica el puerto 1099, que es el puerto por defecto. Esa línea de código es equivalente a ejecutar *rmiregistry* por terminal.

Para ejecutar el ejemplo, he modificado el script que se proporcionaba en el guión, eliminando la parte de lanzar el registro RMI puesto que de esto se encarga el programa servidor. Para lanzar el servidor y el cliente lo he hecho con las siguientes órdenes:

```

gnome-terminal --working-directory=$PWD -- java -cp .
-Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost
-Djava.security.policy=server.policy servidor &

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

```

El resultado de la ejecución del programa es el siguiente:

```

carlosqp@dssd-client: ~/Escritorio/practicaRMI/ejemplo3
carlosqp@dssd-client:~/Escritorio/practicaRMI/ejemplo3$ ./run.sh

Compilando con javac ...
Lanzando el servidor en otra terminal
Lanzando el programa cliente

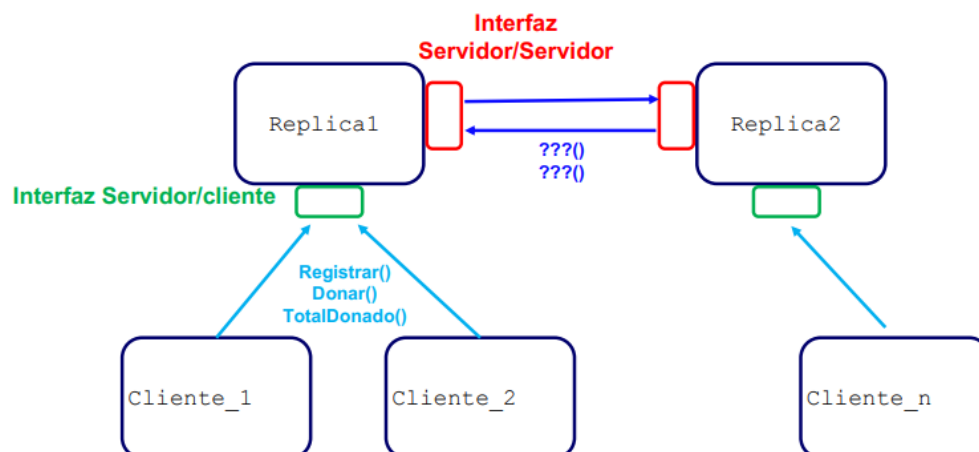
Poniendo contador a 0
Incrementando...
Media de las RMI realizadas = 0.385 msecs
RMI realizadas = 1000
carlosqp@dssd-client:~/Escritorio/practicaRMI/ejemplo3$

```

Segunda parte: Servidor replicado de donaciones

El ejercicio propuesto consistía en implementar un servidor replicado de donaciones, de forma que el cliente se conectaba a uno de los servidores, y este se encargaría de relegar las operaciones correspondientes de ese cliente a la réplica del servidor que tuviera registrado a ese cliente.

Para ello se necesitaba una interfaz remota no sólo para la comunicación cliente-servidor sino también para la comunicación servidor-servidor. El esquema de interacción de elementos es el siguiente (sacado de las diapositivas de la práctica):



En mi solución al ejercicio, he diseñado dos interfaces remotas (*interfazClienteServidor* e *interfazServidorServidor*) cada una con distintas operaciones. Ambas interfaces son implementadas por la clase *Servidor*, que es la clase que será instanciada y registrada en el registro RMI. La definición de las interfaces son las siguientes:

```

public interface interfazClienteServidor extends Remote {
    public Boolean registrar(int id) throws RemoteException;
    public Boolean donar (int id, double cantidad) throws RemoteException;
    public double totalDonaciones(int id) throws RemoteException;
}

public interface interfazServidorServidor extends Remote {
    public Boolean isRegistered(int id) throws RemoteException;
    public int getNumClientes() throws RemoteException;
}

```

```

    public void addCliente(int id) throws RemoteException;
    public void makeDonation(int id, double cantidad) throws
    RemoteException;
    public double getSubtotalDonaciones() throws RemoteException;
    public double getTotalDonaciones(int id) throws RemoteException;
}

```

La *interfazServidorServidor* tiene algunas operaciones adicionales necesarias para comprobaciones, como por ejemplo obtener el subtotal de donaciones de la réplica o comprobar si el cliente está registrado en la réplica. Más abajo se comentan las diferencias entre los métodos de las interfaces (pero básicamente, los métodos de la *interfazClienteServidor* se encargan de invocar al método de la *interfazServidorServidor* de la réplica responsable de realizar la operación; por así decirlo, delegan la operación a otro servidor).

A continuación resalto los detalles más importantes de la implementación. No obstante, para más información, ver la implementación en Java así como los comentarios en el código.

- En el lado del cliente, este se conecta a una de las réplicas (la que sea, no es relevante a cuál se conecte) y realiza las operaciones que escoja. La lógica del programa hace que los servidores se comuniquen entre sí para que la réplica que alberga al cliente sea la que realice las operaciones, en lugar de la réplica a la que está conectado. En el programa cliente proporcionado se conecta a la réplica 0, pero si se quisiera cambiar, basta con modificar la variable *server_name* por el nombre de la réplica deseada.

```

// Crea el stub para el cliente especificando el nombre del servidor
Registry mireg = LocateRegistry.getRegistry("127.0.0.1", 1099);

// Se conecta a la replica0
String server_name = "replica0";
// ...
interfazClienteServidor server =
(interfazClienteServidor)mireg.lookup(server_name);

```

- La clase *Servidor* implementa ambas interfaces remotas y almacena la estructura de datos de cliente y donación de forma local. Por tanto, cada servidor tiene un registro local de sus datos, y no puede acceder a los datos del resto de servidores.

```

public class Servidor extends UnicastRemoteObject implements
interfazClienteServidor, interfazServidorServidor{...}

```

- La clase *Servidor* almacena el registro del que obtener las referencias a objetos remotos, así como un vector de *interfazServidorServidor* en el que almacenará todas las referencias a objetos remotos para poder interaccionar con ellos.

```

// Atributos

```

```

private static int num_servidores = 0; // Para asignar un identificador
único a cada nombre de servidor

public String name; // nombre del servidor replicado (es el mismo que en el
registro RMI)

private Registry registro; // Registro RMI del que se obtendrán las
referencias al resto de replicas remotas

private ArrayList<interfazServidorServidor> replicas; // referencias al
resto de servidores replicados

private Map<Integer, Double> donaciones; // Estructura que almacena los
clientes y sus aportaciones

```

- Para añadir una nueva referencia remota al servidor se llama al método *addReplica(String name)*, siendo name el nombre con el que ha sido registrado en el registro RMI el objeto remoto.

```

public void addReplica(String name) {
    try {
        if(name!=this.name) {
            // Obtiene una referencia al objeto remoto desde el registro y lo
añade a su estructura
            replicas.add((interfazServidorServidor) registro.lookup(name));
        }
    } catch (Exception e) {
        System.err.println("El objeto remoto con el nombre " + name + " no se
ha podido añadir.");
        e.printStackTrace();
    }
}

```

- Es muy importante que en el main del servidor, una vez instanciados todos los servidores y registrados en el registro RMI, a cada servidor se le añadan todos los demás servidores. El orden debe de ser así, puesto que si a un servidor se le añade otro que aún no está registrado en el registro RMI, entonces la búsqueda del recurso fallará y la réplica no se añadirá, quedando así sin conexión los servidores.

```

System.out.println("Instanciando " + n_replicas + " replicas...");
// Crear los servidores y pasarles el registro
for (int i=0;i<n_replicas;i++) {
    // Creamos una réplica
    v_server[i] = new Servidor(registry0);
    // La hacemos accesible
    Naming.rebind(v_server[i].getName(), v_server[i]);
    System.out.println("\treplica "+i+" lista");
}

// Ahora, a cada réplica le pasamos el resto de réplicas para
// que almacenen referencias a ellas

```

```
for (int i=0;i<n_replicas;i++) {
    for (int j=0;j<n_replicas;j++) {
        if (i!=j) v_server[i].addReplica(v_server[j].getName());
    }
}
```

- También es importante que los clientes se lancen una vez se muestre el mensaje en el servidor “¡Servidores replicados listos para responder peticiones de clientes!” puesto que es entonces cuando las réplicas han sido inicializadas y conectadas entre sí. Si se lanzan antes los clientes puede ocurrir que: las réplicas aún no hayan sido instanciadas o que hayan sido instanciadas pero no se hayan conectado aún entre sí (ver implementación del main del servidor). Para que esto no ocurra, en el script de ejecución se ha añadido un retraso de 5 segundos entre el lanzamiento del programa servidor y el del cliente, no obstante, si se ejecuta sin el script se deben de tener en cuenta estos posibles casos de error.

```
echo
echo "Lanzando el servidor en otra terminal"
gnome-terminal --working-directory=$PWD -- java -cp .
-Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost
-Djava.security.policy=server.policy Servidor 3 &
```

```
# Damos tiempo a que se lance el programa
# y se conecten entre sí los servidores replicados
sleep 5
```

```
echo
echo "Lanzando el programa cliente"
echo
```

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

- Por último comentar que, los métodos de la interfazServidorServidor son los que realizan realmente la operación deseada (por ejemplo, añadir un cliente, realizar una donación, etc.). Los métodos de la interfazClienteServidor lo único que hacen es llamar a la réplica encargada (la que tenga el menor número de clientes en el caso de registro, o la que tenga registrada al cliente en el resto de casos) de realizar la operación para que la efectúe (invocando al método de la interfazServidorServidor). Por ejemplo, la operación de obtener el total de donaciones:

```
// Método de la interfaz cliente servidor
public double totalDonaciones(int id) throws RemoteException {
    // Busca al servidor donde está alojado el cliente y llama a la operación
    de ese servidor
    if(isRegistered(id))
        return getTotalDonaciones(id);
    else {
        for(int i=0; i<replicas.size(); ++i) {
            if(replicas.get(i).isRegistered(id))
                return replicas.get(i).getTotalDonaciones(id);
        }
    }
}
```

```

    }
}
// Si llega hasta este punto es porque no está registrado
return -1;
}

// método de la interfaz servidor servidor; el que realmente realiza la
//operación
public double getTotalDonaciones(int id) throws RemoteException {
    if(isRegistered(id) && donaciones.get(id).doubleValue()>0) {
        System.out.println("\t" + getName() + ": recopilando información de
subtotales de otras replicas para responder al cliente " + id);
        double total = getSubtotalDonaciones();
        for(int i=0; i<replicas.size(); ++i)
            total += replicas.get(i).getSubtotalDonaciones();
        return total;
    } else
        return -1;
}

```

A continuación se muestran algunas capturas de ejecución. Las salidas del servidor ayudan a comprender el programa y las operaciones que se realizan aunque no son relevantes. En el script de ejecución proporcionado se lanzan tres servidores. Esto se puede cambiar en el script de ejecución proporcionado simplemente cambiando el primer parámetro del programa servidor.

```

gnome-terminal --working-directory=$PWD -- java -cp .
-Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost
-Djava.security.policy=server.policy Servidor 3 &

```

Para lanzar el script, simplemente ejecutar/:

```
./run.sh
```

Si se quisiera ejecutar sin script, lanzar primero el servidor con la siguiente orden:

```

java -cp . -Djava.rmi.server.codebase=file:./
-Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy
Servidor 3

```

Y posteriormente lanzar el cliente en otra terminal con la siguiente orden:

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



```

carlosqp@dssd-client: ~/Escritorio/practicaRMI/ServidoresReplicados
carlosqp@dssd-client:~/Escritorio/practicaRMI/ServidoresReplicados$ ./run.sh

Compilando con javac ...

Lanzando el servidor en otra terminal

Lanzando el programa cliente

BIENVENIDO AL SERVIDOR DE DONACIONES

Realizando conexión con un servidor de donaciones (replica0)

Antes de continuar, introduzca su identificador de cliente:
0
1. Registrarme
2. Realizar donacion
3. Consultar total donaciones
4. Cerrar sesion (vuelve al paso de introduzca su id de cliente)
1
Realizando solicitud de registro al servidor...
... registro de cliente realizado con éxito
1. Registrarme
2. Realizar donacion
3. Consultar total donaciones
4. Cerrar sesion (vuelve al paso de introduzca su id de cliente)
1
Realizando solicitud de registro al servidor...
... error en el registro, puede que ya se haya registrado anteriormente.

creando registro RMI ...
Instanciando 3 replicas...
replica 0 lista
replica 1 lista
replica 2 lista
¡Todas las replicas se han conectado entre si!
¡Servidores replicados listos para responder peticiones de clientes!

Servidor RemoteException | MalformedURLExceptionor preparado

replica0: recibida petición de registro del cliente 0
replica0: actualmente tengo 0 clientes registrados
replica1: actualmente tengo 0 clientes registrados
replica2: actualmente tengo 0 clientes registrados
replica0: actualmente tengo 0 clientes registrados
replica0: registrado nuevo cliente (id=0)
replica0: recibida petición de registro del cliente 0
replica0: actualmente tengo 1 clientes registrados
replica1: actualmente tengo 0 clientes registrados
replica1: actualmente tengo 0 clientes registrados
replica2: actualmente tengo 0 clientes registrados
replica0: actualmente tengo 1 clientes registrados
replica0: Error, cliente ya registrado

```

Salida 1: se lanza el script, se registra el usuario 0 dos veces, la primera sí puede, y la segunda no. En este caso, como se acaba de lanzar el servidor no hay clientes registrados en ninguna réplica y por tanto se registra en la 0.

```

carlosqp@dssd-client: ~/Escritorio/practicaRMI/ServidoresReplicados
carlosqp@dssd-client:~/Escritorio/practicaRMI/ServidoresReplicados$ java -cp . -Djava.secur
ity.policy=server.policy Cliente
BIENVENIDO AL SERVIDOR DE DONACIONES

Realizando conexión con un servidor de donaciones (replica0)

Antes de continuar, introduzca su identificador de cliente:
1
1. Registrarme
2. Realizar donacion
3. Consultar total donaciones
4. Cerrar sesion (vuelve al paso de introduzca su id de cliente)
1
Realizando solicitud de registro al servidor...
... registro de cliente realizado con éxito
1. Registrarme
2. Realizar donacion
3. Consultar total donaciones
4. Cerrar sesion (vuelve al paso de introduzca su id de cliente)

creando registro RMI ...
Instanciando 3 replicas...
replica 0 lista
replica 1 lista
replica 2 lista
¡Todas las replicas se han conectado entre si!
¡Servidores replicados listos para responder peticiones de clientes!

Servidor RemoteException | MalformedURLExceptionor preparado

replica0: recibida petición de registro del cliente 0
replica0: actualmente tengo 0 clientes registrados
replica1: actualmente tengo 0 clientes registrados
replica2: actualmente tengo 0 clientes registrados
replica0: actualmente tengo 0 clientes registrados
replica0: registrado nuevo cliente (id=0)
replica0: recibida petición de registro del cliente 0
replica0: actualmente tengo 1 clientes registrados
replica1: actualmente tengo 0 clientes registrados
replica1: actualmente tengo 0 clientes registrados
replica2: actualmente tengo 0 clientes registrados
replica0: actualmente tengo 1 clientes registrados
replica0: Error, cliente ya registrado
replica0: recibida petición de registro del cliente 1
replica0: actualmente tengo 1 clientes registrados
replica1: actualmente tengo 0 clientes registrados
replica1: actualmente tengo 0 clientes registrados
replica2: actualmente tengo 0 clientes registrados
replica0: actualmente tengo 1 clientes registrados
replica1: registrado nuevo cliente (id=1)

```

Salida 2: Se intenta registrar el cliente 1. La operación la realiza el servidor con menos clientes (en este caso cualquiera menos la replica0).

```

carlosqp@dssd-client: ~/Escritorio/practicaRMI/ServidoresReplicados
carlosqp@dssd-client:~/Escritorio/practicaRMI/ServidoresReplicados$ java -cp . -Djava.secur
ity.policy=server.policy Cliente
BIENVENIDO AL SERVIDOR DE DONACIONES

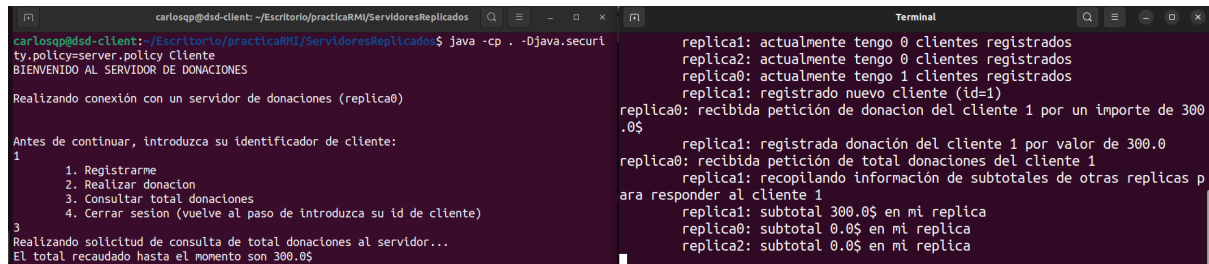
Realizando conexión con un servidor de donaciones (replica0)

Antes de continuar, introduzca su identificador de cliente:
1
1. Registrarme
2. Realizar donacion
3. Consultar total donaciones
4. Cerrar sesion (vuelve al paso de introduzca su id de cliente)
2
Introduzca la cantidad a donar:
300
Realizando solicitud de donacion al servidor...
... donacion realizada con éxito

replica0: actualmente tengo 1 clientes registrados
replica0: Error, cliente ya registrado
replica0: recibida petición de registro del cliente 1
replica0: actualmente tengo 1 clientes registrados
replica1: actualmente tengo 0 clientes registrados
replica2: actualmente tengo 0 clientes registrados
replica0: actualmente tengo 0 clientes registrados
replica0: actualmente tengo 1 clientes registrados
replica1: registrado nuevo cliente (id=1)
replica0: recibida petición de donacion del cliente 1 por un importe de 300.0$
replica1: registrada donación del cliente 1 por valor de 300.0

```

Salida 3: el cliente 1 realiza una donación. La operación la realiza el servidor en el que está registrado (replica1) en lugar del servidor al que está conectado (replica0).



```
carlosqp@dssd-client: ~/Escritorio/practicaRMI/ServidoresReplicados
carlosqp@dssd-client:~/Escritorio/practicaRMI/ServidoresReplicados$ java -cp . -Djava.security.policy=server.policy Cliente
BIENVENIDO AL SERVIDOR DE DONACIONES

Realizando conexión con un servidor de donaciones (replica0)

Antes de continuar, introduzca su identificador de cliente:
1
  1. Registrarme
  2. Realizar donacion
  3. Consultar total donaciones
  4. Cerrar sesion (vuelve al paso de introduzca su id de cliente)
3
Realizando solicitud de consulta de total donaciones al servidor...
El total recaudado hasta el momento son 300.0$

replica1: actualmente tengo 0 clientes registrados
replica2: actualmente tengo 0 clientes registrados
replica0: actualmente tengo 1 clientes registrados
replica1: registrado nuevo cliente (id=1)
replica0: recibida petición de donacion del cliente 1 por un importe de 300.0$
replica1: registrada donación del cliente 1 por valor de 300.0
replica0: recibida petición de total donaciones del cliente 1
replica1: recopilando información de subtotales de otras replicas para responder al cliente 1
replica1: subtotal 300.0$ en mi replica
replica0: subtotal 0.0$ en mi replica
replica2: subtotal 0.0$ en mi replica
```

Salida 4: el cliente 1 realiza una consulta de total donado. De nuevo, la operación la realiza el servidor en el que está registrado (replica1) en lugar del servidor al que está conectado (replica0).