

Práctica 3 DSD: RMI

Alumno: Raúl Rodríguez Pérez

Grupo: dsd2

Esta práctica tiene como objetivo entender el funcionamiento de RMI (Remote Method Invocation) mediante ejemplos prácticos otorgados en la documentación de dicha práctica, y mediante la realización de un ejercicio sobre réplica de servidores. En primer lugar, empezaré explicando los 3 ejemplos que se nos otorgan, hablando en detalle, sobre qué ocurre en cada uno y cuales son las diferencias entre ellos. Tras esto, pasaré a explicar la implementación que he realizado para la realización del ejercicio sobre el servidor replicado. En dicha explicación hablaré de la técnica que he seguido para replicar el servidor, explicaré los métodos utilizados, y expondré un caso de ejecución.

1. Implementación de los ejemplo

- Ejemplo1

Tal y como nos explican en el pdf, el programa ejemplo que se muestra es un pequeño ejemplo de programa cliente-servidor. En dicho programa, nos encontramos con diversos ficheros. En primer lugar, tenemos al servidor (Ejemplo.java), el cual, exporta los métodos contenidos en la interfaz “Ejemplo_I.java” del objeto remoto instanciado como “Ejemplo_I”. La funcionalidad del servidor se basa en que cuando recibe una petición de un cliente imprime el argumento enviado en la llamada. Destacando que si el argumento recibido es un “0”, el programa espera un tiempo antes de volver a imprimir el mensaje.

Por otro lado tenemos el fichero “Cliente_Ejemplo”, el cual realiza las acciones de activar el gestor de seguridad, buscar el objeto remoto, y llamar al método escribir_mensaje.

Si ejecutamos el programa podemos ver que tal y como expliqué anteriormente, si el cliente pasa como parámetro un ‘0’, el servidor espera un momento dormido, mientras que si es cualquier otro número, simplemente recibe la petición y lo imprime por pantalla.

```
raulrguez@pop-os:~/Escritorio/DSD-master/P3/S1/Ejemplo1$ java -
cp . -Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostn
ame=localhost -Djava.security.policy=server.policy Ejemplo
Ejemplo bound
Recibida petición de proceso: 0
Empezamos a dormir
Terminamos de dormir

Hebra 0
Recibida petición de proceso: 3

Hebra 3
[]

raulrguez@pop-os:~/Escritorio/DSD-master/P3/S1/Ejemplo1$ java -
cp . -Djava.security.policy=server.policy Cliente_Ejemplo local
host 0
Buscando el objeto remoto
Invocando el objeto remoto
raulrguez@pop-os:~/Escritorio/DSD-master/P3/S1/Ejemplo1$ java -
cp . -Djava.security.policy=server.policy Cliente_Ejemplo local
host 3
Buscando el objeto remoto
Invocando el objeto remoto
raulrguez@pop-os:~/Escritorio/DSD-master/P3/S1/Ejemplo1$
```

- Ejemplo2

El segundo ejemplo es muy similar al anterior, pero en lugar de enviar varios clientes, se crean varias hebras que realizan la misma tarea de imprimir un mensaje remoto accediendo al stub de un objeto remoto. Esto nos permite ver la gestión de la concurrencia que tiene RMI, ya que al ser multihebrado, permite la gestión concurrente de las peticiones de los clientes. Para este ejemplo, en el cliente se le pasará el nombre del host del servidor seguido del número de hebras que queremos crear.

```

raulrguez@pop-os:~/Escritorio/DSD-master/P3/S1/Ejemplo2$ ps
    PID TTY          TIME CMD
  16978 pts/1    00:00:00 bash
  17036 pts/1    00:00:00 ps
raulrguez@pop-os:~/Escritorio/DSD-master/P3/S1/Ejemplo2$ rmireg
istry &
[1] 17038
raulrguez@pop-os:~/Escritorio/DSD-master/P3/S1/Ejemplo2$ javac
*.java
raulrguez@pop-os:~/Escritorio/DSD-master/P3/S1/Ejemplo2$ jav
a -cp . -Djava.rmi.server.codebase=file:./ -Djava.rmi.server.ho
stname=localhost -Djava.security.policy=server.policy Ejemplo
Ejemplo bound

Entra Hebra Cliente 18
Sale Hebra Cliente 18

Entra Hebra Cliente 15
Sale Hebra Cliente 15

Entra Hebra Cliente 3
Sale Hebra Cliente 3

Entra Hebra Cliente 5
Sale Hebra Cliente 5

Entra Hebra Cliente 7
Sale Hebra Cliente 7

Entra Hebra Cliente 8
Sale Hebra Cliente 8

Entra Hebra Cliente 0
Empezamos a dormir

Entra Hebra Cliente 6
Sale Hebra Cliente 6
Sale Hebra Cliente 26
Sale Hebra Cliente 12
Terminamos de dormir
Sale Hebra Cliente 0
Terminamos de dormir
Sale Hebra Cliente 20
Terminamos de dormir
Sale Hebra Cliente 10

```

Podemos observar como todas las hebras entran y salen seguido, excepto las que acaban en 0. Dichas hebras empiezan a dormir y finalizan cuando las demás ya han entrado y salido correctamente. Si añadimos el modificador `synchronized` lo que logramos es que cuando entran las hebras que terminan en 0, el programa debe esperar a que salgan antes de dar paso a la siguiente hebra.

<pre> raulrguez@pop-os:~/Escritorio/DSD-master/P3/S1/Ejemplo2\$ javac *.java raulrguez@pop-os:~/Escritorio/DSD-master/P3/S1/Ejemplo2\$ java -cp . -Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy Ejemplo Ejemplo bound Entra Hebra Cliente 17 Sale Hebra Cliente 17 Entra Hebra Cliente 0 Empezamos a dormir Terminamos de dormir Sale Hebra Cliente 0 Entra Hebra Cliente 27 Sale Hebra Cliente 27 </pre>	<pre> raulrguez@pop-os:~/Escritorio/DSD-master/P3/S1/Ejemplo2\$ java -cp . -Djava.security.policy=server.policy Cliente Ejemplo_Multi Threaded localhost 30 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 Buscando el objeto remoto Buscando el objeto remoto </pre>
--	--

- Ejemplo3

Por último el programa contador, es un pequeño ejemplo de programa cliente-servidor. El cual se crea por un lado el objeto remoto que consta de varias funciones accesibles remotamente, y por el otro el servidor que exporta los métodos contenidos en la interfaz “icontador” del objeto remoto instanciado como micontador. La funcionalidad del programa es poner un valor inicial al contador del servidor, después, incrementar dicho valor 1.000 veces e imprimir el valor final del contador junto con el tiempo de respuesta medio calculado a partir de las invocaciones remotas al método incrementar. En este ejemplo se nos da una estructura más completa de la que tomaré referencia para realizar el ejercicio a continuación.

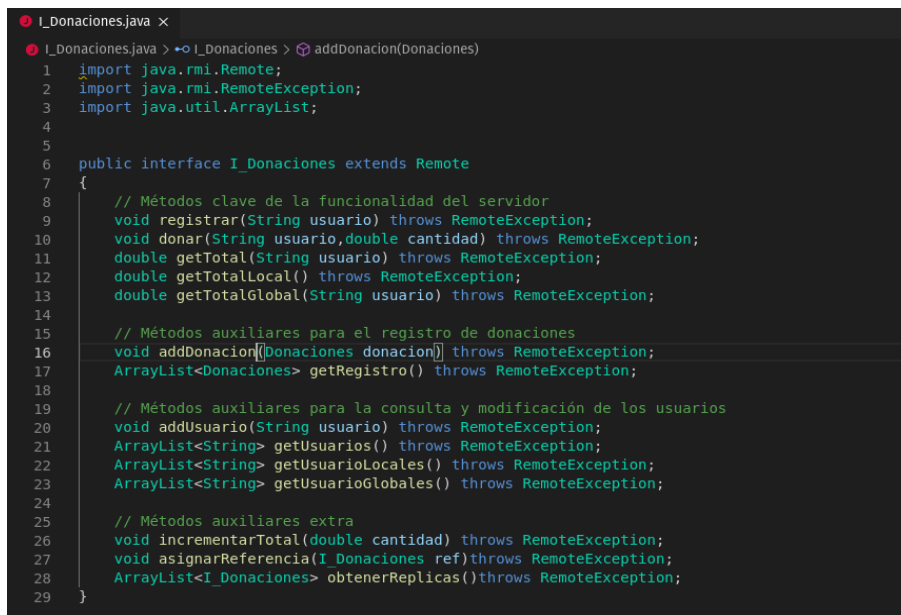
<pre> raulrguez@pop-os:~/Escritorio/DSD-master/P3/S1/Ejemplo3\$ rmiregistry & [1] 18315 raulrguez@pop-os:~/Escritorio/DSD-master/P3/S1/Ejemplo3\$ javac *.java raulrguez@pop-os:~/Escritorio/DSD-master/P3/S1/Ejemplo3\$ java -cp . -Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy servidor Servidor RemoteException MalformedURLExceptiononr preparado </pre>	<pre> raulrguez@pop-os:~/Escritorio/DSD-master/P3/S1/Ejemplo3\$ java -cp . -Djava.security.policy=server.policy cliente localhost 10 Poniendo contador a 0 Incrementando... Media de las RMI realizadas = 0.218 msecs RMI realizadas = 1000 raulrguez@pop-os:~/Escritorio/DSD-master/P3/S1/Ejemplo3\$ </pre>
--	--

2. Servidor Replicado

La tarea a realizar consiste en desarrollar en RMI un sistema cliente-servidor teniendo en cuenta una serie de requisitos. Entre ellos cabe destacar que el servidor será un servidor replicado (con exactamente 2 réplicas). Estas dos réplicas se encargarán de recibir donaciones de diferentes usuarios/clientes.

Para llevar a cabo la implementación del ejercicio, seguí al pie de la letra el apartado 1.2.3 de la memoria sobre la práctica. En dicho apartado se describe el proceso para la creación, desarrollo y ejecución de las clases RMI. Por lo que, el primer paso que realicé fue “definir una interfaz remota”, en donde el objeto remoto tenga declarados sus servicios. En mi implementación, la interfaz tiene como nombre “I_Donaciones.java”. Esta clase es muy sencilla, los únicos puntos a destacar son por un lado, el uso del import de la interfaz java.rmi.Remote, sobre la cual extiende nuestra clase. Y por otro lado, la declaración de todos los métodos/servicios que podrá emplear el objeto remoto seguido de una excepción que deben lanzar (java.rmi.RemoteException).

Dichos métodos serán empleados para lograr la correcta funcionalidad del servidor, por lo cual, la clase servidor implementará la interfaz “I_Donaciones.java”. A continuación voy a proceder a explicar en concreto cada uno de estos métodos:



```
I_Donaciones.java x
I_Donaciones.java > -> I_Donaciones > addDonacion(Donaciones)
1  import java.rmi.Remote;
2  import java.rmi.RemoteException;
3  import java.util.ArrayList;
4
5
6  public interface I_Donaciones extends Remote
7  {
8      // Métodos clave de la funcionalidad del servidor
9      void registrar(String usuario) throws RemoteException;
10     void donar(String usuario,double cantidad) throws RemoteException;
11     double getTotal(String usuario) throws RemoteException;
12     double getTotalLocal() throws RemoteException;
13     double getTotalGlobal(String usuario) throws RemoteException;
14
15     // Métodos auxiliares para el registro de donaciones
16     void addDonacion([Donaciones donacion]) throws RemoteException;
17     ArrayList<Donaciones> getRegistro() throws RemoteException;
18
19     // Métodos auxiliares para la consulta y modificación de los usuarios
20     void addUsuario(String usuario) throws RemoteException;
21     ArrayList<String> getUsuarios() throws RemoteException;
22     ArrayList<String> getUsuarioLocales() throws RemoteException;
23     ArrayList<String> getUsuarioGlobales() throws RemoteException;
24
25     // Métodos auxiliares extra
26     void incrementarTotal(double cantidad) throws RemoteException;
27     void asignarReferencia(I Donaciones ref)throws RemoteException;
28     ArrayList<I_Donaciones> obtenerReplicas()throws RemoteException;
29 }
```

Como se puede observar en la foto, tengo hecha una división de los métodos según su logística. En el primer grupo, se encuentran los métodos que son empleados para realizar la funcionalidad del servidor. Dentro de este grupo se encuentran los tres métodos principales del programa (registrar, donar, getTotal). En el enunciado sobre el ejercicio, se pedía que al implementar dichos métodos, se tuvieran en cuenta una serie variada de requisitos. Por lo que a la hora de realizar su implementación, en una primera versión que realicé, me quedaron métodos muy engorrosos, con diversas líneas de código que pueden, en un primer vistazo, confundir al lector. Por ello, decidí crear un método auxiliar “obtenerReplica”, el cual consiste en encapsular en un switch la funcionalidad de estos 3 métodos. Es decir, en primer lugar para el método de registrar, se comprueba en todas las réplicas si existe el usuario cuyo nombre se pasa por parámetro. Si se encuentra, notifica el error de que el usuario ya se encuentra registrado en una de las réplicas. Pero, si no lo encuentra, busca la réplica que posee menor número de registros y almacena la referencia de la misma en una variable denominada “replicaSelec”, de contraer “replica seleccionada”. Esta variable será usada para almacenar la referencia apropiada a una réplica y aplicar, a la misma, las operaciones oportunas.

```

private void obtenerReplica(int opcion, String usuario )throws RemoteException{
    Boolean encontrado = false;
    switch(opcion){
        //Registrar
        case 0:
            this.replicaSelec = null;
            for(int i = 0; i < this.replicas.size() && !encontrado; i++){
                if(this.replicas.get(i).getUsuarios().contains(usuario)){
                    this.replicaSelec = this.replicas.get(i);
                    encontrado = true;
                }
            }

            if(encontrado){
                System.err.println("Error usuario que quiere registrar ya se encuentra en alguna réplica");
                this.replicaSelec = null;
            }
            else{
                this.replicaSelec = this;
                for(int i = 0; i < this.replicas.size(); i++){
                    if(this.replicaSelec.getUsuarios().size() >= this.replicas.get(i).getUsuarios().size() ){
                        this.replicaSelec = this.replicas.get(i);
                    }
                }
            }
        }
    }
    break;
}

```

Gracias al encapsulamiento de la funcionalidad, el método registrar queda sencillo, legible y poco engorroso. En él se llama al método “obtenerReplica”, pasándole como argumentos el ‘0’ (registrar) y el nombre de usuario del cliente. Tras esto se comprueba de que la referencia a “replicaSelec” sea distinta de null, si lo es, se añade el usuario a la lista de usuarios de la referencia de la “replicaSelec”.

```

/*****
/* Métodos clave de la funcionalidad del servidor */
*****/
@Override
public void registrar(String usuario) throws RemoteException{
    obtenerReplica(0,usuario);
    if(this.replicaSelec != null){
        this.replicaSelec.addUsuario(usuario);
        System.out.println("El usuario " + usuario + " ha sido registrado con éxito");
    }
}

```

En segundo lugar, en la funcionalidad del método donar, simplemente se busca el nombre del usuario pasado como argumento en todas las listas de usuarios de las diversas réplicas. Si no se encuentra, se notificará que el usuario no está registrado y por lo tanto no puede donar. Pero si se encuentra, se incrementará el precio total local, referente a la referencia de la réplica seleccionada. Y, tal y como se pedía en el enunciado, se almacenará un registro de donaciones por parte de los usuarios. En mi caso he creado la clase “Donaciones.java”, la cual tiene como atributos el nombre del usuario, y la cantidad donada. Tras esto he creado un ArrayList de donaciones al que he nombrado registro. Añadiendo, cada vez que se done, una nueva donación al array.

```
//Donar
case 1:
    encontrado = this.usuarios.contains(usuario);
    for(int i = 0; i < this.replicas.size() && !encontrado; i++){
        if(this.replicas.get(i).getUsuarios().contains(usuario)){
            this.replicaSelec = this.replicas.get(i);
            encontrado = true;
        }
    }
    if(!encontrado){
        System.err.println("Error el usuario no está registrado");
        this.replicaSelec = null;
    }
    break;
```

```
import java.io.Serializable;

public class Donaciones implements Serializable{
    public String usuario;
    public double donacion;

    Donaciones(String user, double cant){
        this.usuario = user;
        this.donacion = cant;
    }

    public String getUser()
    {
        return this.usuario;
    }

    public void setUser(String u)
    {
        this.usuario = u;
    }

    public double getDonacion()
    {
        return this.donacion;
    }

    public void setDonacion(double d)
    {
        this.donacion = d;
    }
}
```

```
@Override
public void donar(String usuario,double cantidad) throws RemoteException{

    if(cantidad > 0){
        obtenerReplica(1,usuario);
        if(this.replicaSelec != null)
        {
            this.replicaSelec.incrementarTotal(cantidad);
            Donaciones aport = new Donaciones(usuario, cantidad);
            this.replicaSelec.addDonacion(apor);
            System.out.println("El usuario " + usuario + " ha donado " + cantidad + " euros");
        }
    }else
    {
        System.out.println("Se ha intentado donar una cantidad no válida");
    }
}
```

Ese array de donaciones lo emplearemos en el último método que nos queda por explicar de los principales, el `getTotal`. En el enunciado del problema nos plantean que, solo puede observar el total donado, aquellos usuarios que estén registrados, y hayan donado por lo menos una vez. Por ello, en el método “`obtenerReplica`”, para el case 2, lo que se hace es buscar en todos los registros de las diversas réplicas, para buscar si se encuentra el nombre del usuario pasado por parámetro. Ya que si se encuentra, significa que dicho usuario ha donado, mínimo una vez.

```
//Consultar
case 2:
    boolean valido = false;
    for(int j = 0; j < this.getRegistro().size() && !valido; j++){
        if( this.getRegistro().get(j).getUser().equals(usuario)){
            valido = true;
            this.replicaSelec = this;
        }
    }

    for(int i = 0; i < this.replicas.size() && !valido; i++){
        for(int j = 0; j < this.replicas.get(i).getRegistro().size() && !valido; j++){
            if( this.replicas.get(i).getRegistro().get(j).getUser().equals(usuario)){
                valido = true;
                this.replicaSelec = this.replicas.get(i);
            }
        }
    }

    if(!valido)
        this.replicaSelec = null;

    break;
```

```
@Override
public double getTotal(String usuario) throws RemoteException{
    obtenerReplica(2,usuario);
    if(this.replicaSelec != null)
    {
        double total = this.replicaSelec.getTotalLocal();
        return total;
    }
    else{
        System.out.println("Es necesario realizar una donación para poder consultar el total local aportado");
        return 0.0;
    }
}
```

Con esto, ya estaría explicada la funcionalidad básica del programa. Aún así, como se puede apreciar existen diversos métodos que aún no he explicado. Para no extenderme en exceso, la gran cantidad de métodos restantes se basan en métodos `get` (como por ejemplo `getRegistro()`, que devuelve una referencia al array de registros, o `get usuarios` que devuelve una referencia al array de usuarios), métodos `add` (como por ejemplo `addDonacion()`, que añade una donación al registro), un método para incrementar el aporte donado al `totalLocal` (“`incrementarTotal()`”). En definitiva, me quedan por destacar 4 métodos que son llamados desde el cliente, los cuales son, `getUsuariosLocales()` y `...Globales()`; y `getTotalLocal()` y `...Global()`. Estos métodos sirven de ayuda para diferencias, a la hora de la ejecución del programa, que cosas están en donde. Es decir, por un lado están los métodos para mostrar los usuarios que se encuentran registrados en la réplica local, desde donde se están realizando las operaciones. Y por otro lado tienes la opción de mostrar

todos los usuarios, ya que aunque realices la operación de registrar sobre la réplica 0, por ejemplo, puede darse el caso de que el usuario se registre en la réplica 1. Del mismo modo, tenemos esa opción para el total donado. En donde un usuario registrado puede observar lo que se ha donado tanto local, como de manera global.

```
public double getTotalLocal() throws RemoteException{
    return this.totalLocal;
}

public double getTotalGlobal(String usuario) throws RemoteException{
    double totalGlobal = 0.0;
    obtenerReplica(2,usuario);
    if(this.replicaSelec != null){
        for(int i = 0; i < this.replicas.size(); i++){
            totalGlobal += this.replicas.get(i).getTotalLocal();
        }
        return totalGlobal;
    }
    else{
        System.out.println("Es necesario realizar una donación para poder consultar el total global aportado");
        return 0.0;
    }
}
```

```
public ArrayList<String> getUsuarioLocales() throws RemoteException{
    ArrayList<String> usuariosLocales = new ArrayList<String>();
    for(int i = 0; i < this.usuarios.size(); i++){
        usuariosLocales.add(this.usuarios.get(i));
    }
    return usuariosLocales;
}

public ArrayList<String> getUsuarioGlobales() throws RemoteException{
    ArrayList<String> usuariosGlobales= new ArrayList<String>();
    for(int i = 0; i < this.replicas.size(); i++){
        for(int j = 0; j < this.replicas.get(i).getUsuarios().size(); j++){
            usuariosGlobales.add(this.replicas.get(i).getUsuarios().get(j));
        }
    }
    return usuariosGlobales;
}
```

Por último, pero no menos importante, queda explicar el método en el cual enlazamos las 2 réplicas del servidor. De primeras estuve un poco perdido sobre cómo podía enlazar las réplicas, pero, leyendo nuevamente los apuntes sobre RMI, y buscando algunos ejemplos en google encontré una manera de hacerlo. En dichos ejemplos usaban la interfaz Registry de rmi, la cual, es una interfaz remota para el registro de objetos remotos simples, que proporciona métodos para almacenar y recuperar referencias de objetos remotos enlazadas con nombres de cadenas arbitrarios. Buscando más acerca de esta interfaz, encontré los métodos necesarios para lograr mi objetivo. Con los métodos LocateRegistry.getRegistry(), y registry.lookup(), se puede conseguir extraer las referencias concretas de una réplica. Con la referencia de una réplica puedes realizar cualquier operación que requieras hacer, como por ejemplo registrar un usuario en esa réplica, observar la lista de usuarios, observar el totalLocal de las donaciones, y todo referente a la réplica que se referencia. De esta forma, para conseguir enlazar dos réplicas, solo

tenemos que almacenar las referencias de una, por ejemplo, guardar la referencia de la réplica original en la réplica y viceversa.

```
public void enlazarServidores(String ubi, String nombre, int nReplicas){
    try {
        if(nReplicas == 2){
            Registry registry = LocateRegistry.getRegistry(ubi);
            //En el registro localhost miro el nombre
            I_Donaciones aux = (I_Donaciones) registry.lookup(nombre);

            // Añado referencias a la lista de replicas
            this.replicas.add(aux);
            this.replicas.add(this);

            // Añado las referencias al servidor original
            aux.asignarReferencia(this);
            aux.asignarReferencia(aux);

            System.out.println("2 Servidores enlazados correctamente");
        }
    }
}
```

Tras esto, en el servidor solo faltaría por explicar el main, el cual no tiene mucha ciencia. En dicho método según el argumento numérico que le pasemos se creará el servidor original, o una réplica llamando al método que hemos explicado anteriormente.

```
public static void main(String[] args) {
    if(args.length < 2){
        System.out.println("Error: uso correcto -> java Servidor <IP> <0> (primer servidor) | <1|2|...> si ya existe el primer servidor");
        System.exit(0);
    }

    if (System.getSecurityManager() == null) {
        System.setSecurityManager(new SecurityManager());
    }

    try {
        String nombre_objeto_remoto = "";
        switch(args[1]){
            case "0":
                nombre_objeto_remoto = "Original";
                break;

            case "1":
                nombre_objeto_remoto = "Replica";
                break;
        }

        I_Donaciones servidor= new Servidor();
        I_Donaciones stub = (I_Donaciones ) UnicastRemoteObject.exportObject(servidor, 0);
        Registry registry = LocateRegistry.getRegistry();
        registry.rebind(nombre_objeto_remoto, stub);

        if(Integer.parseInt(args[1]) == 1){
            ((Servidor) servidor).enlazarServidores(args[0], "Original", 2);
        }
    }
}
```

Para finalizar con la explicación sobre las clases y sus métodos, nos queda explicar el funcionamiento de la clase “Cliente.java”. En dicha clase, básicamente se hace un pequeño menú con un switch, en donde ofrece las opciones disponibles a los diversos usuarios/clientes. En este menú puedes realizar las siguientes acciones:

- a) Registrar: Pide que escribamos el nombre de usuario por teclado, y luego mediante una instancia/referencia al servidor que hemos elegido por medio del paso de argumentos de la clase, se llama al método encargado de registrar que se encuentra en el servidor.
- b) Donar: Pide que escribamos el nombre de usuario y la cantidad a donar, tras esto se llama al método encargado de realizar la donación en el servidor

- c) Consultar usuarios locales: Permite observar una lista con los nombres de usuarios de los clientes que se han registrado localmente.
- d) Consultar usuarios globales: Permite observar una lista con los nombres de usuarios de los clientes que se han registrado globalmente.
- e) Consultar total local: Tras escribir el nombre de usuario del cliente, permite observar el total donado por los usuarios localmente.
- f) Consultar total global: Tras escribir el nombre de usuario del cliente, permite observar el total donado por los usuarios globalmente.
- g) Salir: salir del menú.

```
try {
    switch(args[1]){
        case "0":
            nombre_objeto_remoto = "Original";
            break;

        case "1":
            nombre_objeto_remoto = "Replica";
            break;
    }

    System.out.println("Buscando el servidor...");
    Registry registry = LocateRegistry.getRegistry(args[0]);
    I_Donaciones instancia_local = (I_Donaciones) registry.lookup(nombre_objeto_remoto);
    System.out.println("Invocando el servidor");

    while(!salir){
        System.out.println("\nMenú sistema RMI cliente-servidor:");
        System.out.println("[1] Registrar" + "\n[2] Donar" + "\n[3] Consultar usuarios locales" +
            "\n[4] Consultar usuarios globales" + "\n[5] Consultar total local" + "\n[6] Consultar total global" + "\n[7] Salir");
        System.out.println("\nElige la opción que quieras realizar: ");
        opc = Integer.parseInt(teclado.nextLine());
        switch(opc){
            case 1:
                System.out.println("Introduzca el nombre de usuario: ");
                nombre = teclado.nextLine();
                instancia_local.registrar(nombre);
                break;

            case 2:
                System.out.println("Introduzca el nombre de usuario: ");
                nombre = teclado.nextLine();
                System.out.println("Introduzca la cantidad a donar por "+ nombre + " : ");
                dinero = teclado.nextLine();
                instancia_local.donar(nombre, Double.parseDouble(dinero));
                break;
        }
    }
}
```

```
        case 3:
            ArrayList<String> usuariosLocales = new ArrayList<String>();
            usuariosLocales = instancia_local.getUsuarioLocales();

            for(int i = 0; i < usuariosLocales.size(); i++){
                System.out.print(usuariosLocales.get(i) + " ");
            }
            System.out.println("");
            break;

        case 4:
            ArrayList<String> usuariosGlobales = new ArrayList<String>();
            usuariosGlobales = instancia_local.getUsuarioGlobales();

            for(int i = 0; i < usuariosGlobales.size(); i++){
                System.out.print(usuariosGlobales.get(i) + " ");
            }
            System.out.println("");
            break;

        case 5:
            System.out.println("Introduzca el nombre de usuario: ");
            nombre = teclado.nextLine();
            System.out.println("El total local donado por los usuarios es "+instancia_local.getTotal(nombre)+" euros");
            break;

        case 6:
            System.out.println("Introduzca el nombre de usuario: ");
            nombre = teclado.nextLine();
            System.out.println("El total global donado por los usuarios es "+instancia_local.getTotalGlobal(nombre)+" euros");
            break;

        case 7:
            System.out.println("Hasta luego!");
            salir = true;
            break;
    }
}
```

A modo de pequeño punto rápido, he incluido también el archivo `server.policy`, el cual tiene el contenido que se explica en la memoria de la práctica. Es decir, por parte de la implementación de un gestor de seguridad he realizado los pasos que se nos explican, en primer lugar he activado el gestor de seguridad en el servidor llamando al método “`System.getSecurityManager()`”. A continuación he creado el archivo `server.policy` para modificar la política de seguridad de java. Finalmente, a la hora de ejecutar nuestro programa, tanto para el servidor como para el cliente, especifico la ruta del fichero que he creado anteriormente.

```
server.policy
1 // server.policy
2 grant {
3     permission java.security.AllPermission;
4 };
```

- Caso de ejecución

Ahora voy a dar paso a la explicación de un caso de ejecución en donde intentaré mostrar cómo mi implementación proporciona todos los requisitos pedidos en el ejercicio. En primer lugar, para realizar la ejecución correcta del ejemplo debemos en primer lugar, lanzar el ligador de RMI -> `$rmiregistry &`, luego compilar las clases con -> `javac *.java`. Tras esto lanzamos dos servidores (un original y su réplica) con la sentencia que se explica en la memoria de la práctica, y a su vez, lanzamos el cliente. Todo en diferentes terminales:

// Original

a) `java -cp . -Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy localhost Servidor 0`

// Réplica

b) `java -cp . -Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy localhost Servidor 1`

//Cliente

`java -cp . -Djava.security.policy=server.policy Cliente localhost 0`

Una vez lanzado los servidores y el cliente, vamos a proceder a explicar los posibles casos a la hora de registrar un usuario. En primer lugar elegimos la opción 1, y escribimos el nombre de usuario “pepe”. Observamos que en el servidor “original” aparece la sentencia que nos confirma el correcto registro del usuario.

```
raulrguez@pop-os: ~/Escritorio/DSD-master/P3/S2$ java -cp . -Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy Cliente localhost 0
Buscando el objeto servidor...
Invocando el objeto servidor

Menú sistema RMI cliente-servidor:
[1] Registrar
[2] Donar
[3] Consultar usuarios locales
[4] Consultar usuarios globales
[5] Consultar totallLocal
[6] Consultar total global
[7] Salir

Elige la opción que quieras realizar:
1
Introduzca el nombre de usuario:
pepe

Menú sistema RMI cliente-servidor:
[1] Registrar
[2] Donar
[3] Consultar usuarios locales
[4] Consultar usuarios globales
[5] Consultar totallLocal
[6] Consultar total global
[7] Salir

Elige la opción que quieras realizar:
1

raulrguez@pop-os: ~/Escritorio/DSD-master/P3/S2$ java -cp . -Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy Servidor localhost 0
Servidor iniciado
El usuario pepe ha sido registrado con éxito

raulrguez@pop-os: ~/Escritorio/DSD-master/P3/S2$ java -cp . -Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy Servidor localhost 1
2 Servidores enlazados correctamente
Servidor iniciado
```

Ahora si procedemos a intentar registrar de nuevo al usuario “pepe”, el servidor nos diría que ha surgido un error, ya que el usuario ya se encuentra registrado en alguna de las réplicas existentes.

```
raulrguez@pop-os: ~/Escritorio/DSD-master/P3/S2$ java -cp . -Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy Cliente localhost 0
Buscando el objeto servidor...
Invocando el objeto servidor

Menú sistema RMI cliente-servidor:
[1] Registrar
[2] Donar
[3] Consultar usuarios locales
[4] Consultar usuarios globales
[5] Consultar totallLocal
[6] Consultar total global
[7] Salir

Elige la opción que quieras realizar:
1
Introduzca el nombre de usuario:
pepe

Menú sistema RMI cliente-servidor:
[1] Registrar
[2] Donar
[3] Consultar usuarios locales
[4] Consultar usuarios globales
[5] Consultar totallLocal
[6] Consultar total global
[7] Salir

Elige la opción que quieras realizar:
1

raulrguez@pop-os: ~/Escritorio/DSD-master/P3/S2$ javac *.java
raulrguez@pop-os: ~/Escritorio/DSD-master/P3/S2$ java -cp . -Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy Servidor localhost 0
Servidor iniciado
El usuario pepe ha sido registrado con éxito
Error al registrar, el usuario ya se encuentra registrado en alguna réplica

raulrguez@pop-os: ~/Escritorio/DSD-master/P3/S2$ java -cp . -Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy Servidor localhost 1
2 Servidores enlazados correctamente
Servidor iniciado
```

A continuación vamos a proceder a registrar un nuevo usuario, esto va a provocar que se añada en la réplica del servidor, no en el original. Esto se debe a que al registrar a “antonio”, el servidor réplica tiene el menor número de registros (0), comparando con el número de registros del servidor original (1). Esto podemos comprobarlo llamando la opción número 3 y 4. Esto nos muestra que en la réplica local solo se encuentra el usuario pepe, mientras que los usuarios globales se encuentran pepe y antonio.

```
raulrguez@pop-os: ~/Escritorio/DSD-master/P3/S2
[4] Consultar usuarios globales
[5] Consultar totallLocal
[6] Consultar total global
[7] Salir

Elige la opción que quieras realizar:
3
pepe

Menú sistema RMI cliente-servidor:
[1] Registrar
[2] Donar
[3] Consultar usuarios locales
[4] Consultar usuarios globales
[5] Consultar totallLocal
[6] Consultar total global
[7] Salir

Elige la opción que quieras realizar:
4
Antonio pepe

Menú sistema RMI cliente-servidor:
[1] Registrar
[2] Donar
[3] Consultar usuarios locales
[4] Consultar usuarios globales
[5] Consultar totallLocal
[6] Consultar total global
[7] Salir

Elige la opción que quieras realizar:
1
```

```
raulrguez@pop-os:~/Escritorio/DSD-master/P3/S2$ javac *.java
raulrguez@pop-os:~/Escritorio/DSD-master/P3/S2$ java -cp . -Djava
.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost
-Djava.security.policy=server.policy Servidor localhost 0
Servidor iniciado
El usuario pepe ha sido registrado con éxito
Error al registrar, el usuario ya se encuentra registrado en algu
na réplica
El usuario Antonio ha sido registrado con éxito
```

```
raulrguez@pop-os:~/Escritorio/DSD-master/P3/S2$ java -cp . -Djava
.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost
-Djava.security.policy=server.policy Servidor localhost 1
2 Servidores enlazados correctamente
Servidor iniciado
```

Solo nos faltaría comprobar que, ejecutando un nuevo cliente que actúe en el servidor réplica, aunque intentemos registrarnos de nuevo con el usuario “pepe”, no muestre el mismo error que al intentar registrar a pepe en el servidor original.

```
raulrguez@pop-os:~/Escritorio/DSD-master/P3/S2$ java -cp . -Djava
.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost
-Djava.security.policy=server.policy Cliente localhost 1
Buscando el objeto servidor...
Invocando el objeto servidor

Menú sistema RMI cliente-servidor:
[1] Registrar
[2] Donar
[3] Consultar usuarios locales
[4] Consultar usuarios globales
[5] Consultar totallLocal
[6] Consultar total global
[7] Salir

Elige la opción que quieras realizar:
1
Introduzca el nombre de usuario:
pepe

Menú sistema RMI cliente-servidor:
[1] Registrar
[2] Donar
[3] Consultar usuarios locales
[4] Consultar usuarios globales
[5] Consultar totallLocal
[6] Consultar total global
[7] Salir

Elige la opción que quieras realizar:
1
```

```
raulrguez@pop-os:~/Escritorio/DSD-master/P3/S2$ javac *.java
raulrguez@pop-os:~/Escritorio/DSD-master/P3/S2$ java -cp . -Djava
.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost
-Djava.security.policy=server.policy Servidor localhost 0
Servidor iniciado
El usuario pepe ha sido registrado con éxito
Error al registrar, el usuario ya se encuentra registrado en algu
na réplica
El usuario Antonio ha sido registrado con éxito
```

```
raulrguez@pop-os:~/Escritorio/DSD-master/P3/S2$ java -cp . -Djava
.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost
-Djava.security.policy=server.policy Servidor localhost 1
2 Servidores enlazados correctamente
Servidor iniciado
Error al registrar, el usuario ya se encuentra registrado en algu
na réplica
```

Ahora vamos a comprobar la funcionalidad de donar, para ello una vez que pepe se ha registrado, hacemos una donación y en el servidor nos muestra que se ha realizado correctamente.

```
raulrguez@pop-os: ~/Escritorio/DSD-master/P3/S2
Elige la opción que quieras realizar:
1
Introduzca el nombre de usuario:
pepe

Menú sistema RMI cliente-servidor:
[1] Registrar
[2] Donar
[3] Consultar usuarios locales
[4] Consultar usuarios globales
[5] Consultar totallLocal
[6] Consultar total global
[7] Salir

Elige la opción que quieras realizar:
2
Introduzca el nombre de usuario:
pepe
Introduzca la cantidad a donar por pepe :
20.0

Menú sistema RMI cliente-servidor:
[1] Registrar
[2] Donar
[3] Consultar usuarios locales
[4] Consultar usuarios globales
[5] Consultar totallLocal
[6] Consultar total global
[7] Salir

Elige la opción que quieras realizar:
1

raulrguez@pop-os: ~/Escritorio/DSD-master/P3/S2$ java -cp . -Djava
.rmi.server.codebase=file:/// -Djava.rmi.server.hostname=localhost
-Djava.security.policy=server.policy Servidor localhost 0
Servidor iniciado
El usuario pepe ha sido registrado con éxito
El usuario pepe ha donado 20.0 euros
```

Probamos ahora a donar con un usuario que no está registrado, y como se puede apreciar aparece el error oportuno en el servidor.

```
raulrguez@pop-os: ~/Escritorio/DSD-master/P3/S2
2
Introduzca el nombre de usuario:
pepe
Introduzca la cantidad a donar por pepe :
20.0

Menú sistema RMI cliente-servidor:
[1] Registrar
[2] Donar
[3] Consultar usuarios locales
[4] Consultar usuarios globales
[5] Consultar totallLocal
[6] Consultar total global
[7] Salir

Elige la opción que quieras realizar:
2
Introduzca el nombre de usuario:
Antonio
Introduzca la cantidad a donar por Antonio :
30.0

Menú sistema RMI cliente-servidor:
[1] Registrar
[2] Donar
[3] Consultar usuarios locales
[4] Consultar usuarios globales
[5] Consultar totallLocal
[6] Consultar total global
[7] Salir

Elige la opción que quieras realizar:
1

raulrguez@pop-os: ~/Escritorio/DSD-master/P3/S2$ java -cp . -Djava
.rmi.server.codebase=file:/// -Djava.rmi.server.hostname=localhost
-Djava.security.policy=server.policy Servidor localhost 0
Servidor iniciado
El usuario pepe ha sido registrado con éxito
El usuario pepe ha donado 20.0 euros
Error usuario no registrado
```

Para finalizar, voy a simular que dos usuarios registrados donan al programa. Tras esto se observará la funcionalidad de observar el total local, en el cual aparecerá el total donado por el usuario local. Y luego mostraré el total global, siendo una suma de los totales locales donados.

```
raulrguez@pop-os: ~/Escritorio/DSD-master/P3/S2
Elige la opción que quieras realizar:
1
Introduzca el nombre de usuario:
pepe

Menú sistema RMI cliente-servidor:
[1] Registrar
[2] Donar
[3] Consultar usuarios locales
[4] Consultar usuarios globales
[5] Consultar total local
[6] Consultar total global
[7] Salir

Elige la opción que quieras realizar:
2
Introduzca el nombre de usuario:
pepe
Introduzca la cantidad a donar por pepe :
20

Menú sistema RMI cliente-servidor:
[1] Registrar
[2] Donar
[3] Consultar usuarios locales
[4] Consultar usuarios globales
[5] Consultar total local
[6] Consultar total global
[7] Salir

Elige la opción que quieras realizar:
1

raulrguez@pop-os: ~/Escritorio/DSD-master/P3/S2$ java -cp . -Djava
.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost
-Djava.security.policy=server.policy Servidor localhost 0
Servidor iniciado
El usuario pepe ha sido registrado con éxito
El usuario pepe ha donado 20.0 euros

raulrguez@pop-os: ~/Escritorio/DSD-master/P3/S2$ java -cp . -Djava
.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost
-Djava.security.policy=server.policy Servidor localhost 1
2 Servidores enlazados correctamente
Servidor iniciado
```

```
Elige la opción que quieras realizar:
5
Introduzca el nombre de usuario:
pepe
El total local donado por los usuarios es 20.0 euros

Menú sistema RMI cliente-servidor:
[1] Registrar
[2] Donar
[3] Consultar usuarios locales
[4] Consultar usuarios globales
[5] Consultar total local
[6] Consultar total global
[7] Salir
```

```
raulrguez@pop-os: ~/Escritorio/DSD-master/P3/S2
Elige la opción que quieras realizar:
1
Introduzca el nombre de usuario:
antonio

Menú sistema RMI cliente-servidor:
[1] Registrar
[2] Donar
[3] Consultar usuarios locales
[4] Consultar usuarios globales
[5] Consultar total local
[6] Consultar total global
[7] Salir

Elige la opción que quieras realizar:
2
Introduzca el nombre de usuario:
antonio
Introduzca la cantidad a donar por antonio :
30

Menú sistema RMI cliente-servidor:
[1] Registrar
[2] Donar
[3] Consultar usuarios locales
[4] Consultar usuarios globales
[5] Consultar total local
[6] Consultar total global
[7] Salir

Elige la opción que quieras realizar:

raulrguez@pop-os: ~/Escritorio/DSD-master/P3/S2$ java -cp . -Djava
.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost
-Djava.security.policy=server.policy Servidor localhost 0
Servidor iniciado
El usuario pepe ha sido registrado con éxito
El usuario pepe ha donado 20.0 euros
El usuario antonio ha sido registrado con éxito
El usuario antonio ha donado 30.0 euros

raulrguez@pop-os: ~/Escritorio/DSD-master/P3/S2$ java -cp . -Djava
.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost
-Djava.security.policy=server.policy Servidor localhost 1
2 Servidores enlazados correctamente
Servidor iniciado
```



```
Elige la opción que quieras realizar:
5
Introduzca el nombre de usuario:
antonio
El total local donado por los usuarios es 30.0 euros

Menú sistema RMI cliente-servidor:
[1] Registrar
[2] Donar
[3] Consultar usuarios locales
[4] Consultar usuarios globales
[5] Consultar total local
[6] Consultar total global
[7] Salir
```

```
Elige la opción que quieras realizar:
6
Introduzca el nombre de usuario:
pepe
El total global donado por los usuarios es 50.0 euros
```

Como último punto traté de añadir una nueva réplica y ver si seguía funcionando sin problemas el programa. Al principio tuve problemas y no conseguí hacer que funcionara. Pero al final conseguí que funcionara con tres servidores. Al final me di cuenta de que se podría escalar para n servidores, ya que solo tendría que recorrer las réplicas del original e ir asignando una a una las nuevas réplicas y viceversa. Aún así, esta forma no la he implementado por falta de tiempo. Pero si he dejado la opción de poner 3 servidores.

```
}
if(nReplicas == 3){
    Registry registry = LocateRegistry.getRegistry(ubi);
    I_Donaciones aux = (I_Donaciones) registry.lookup("Original");
    I_Donaciones aux3 = (I_Donaciones) registry.lookup("Replica2");

    aux.asignarReferencia(aux3);
    aux.asignarReferencia(aux);

    aux3.asignarReferencia(aux3);
    aux3.asignarReferencia(aux);

    System.out.println("3 Servidores enlazados correctamente");
}
```

En la siguiente foto se puede ver un caso de ejecución en el que he registrado al usuario pepe en el servidor original, y he intentado registrarlo en la réplica 2, sin éxito, porque me aparece el error de que ya ha sido registrado en alguna réplicas.

```
raulrguez@pop-os: ~/Escritorio/DSD_P3
raulrguez@pop-os:~/Escritorio/DSD_P3$ java -cp . -Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy Servidor localhost 0
Servidor iniciado
El usuario pepe ha sido registrado con éxito

```

```
raulrguez@pop-os:~/Escritorio/DSD_P3$ cd Escritorio/DSD_P3/
raulrguez@pop-os:~/Escritorio/DSD_P3$ java -cp . -Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy Servidor localhost 2
3 Servidores enlazados correctamente
Servidor iniciado
Error usuario que quiere registrar ya se encuentra en alguna réplica

```

```
raulrguez@pop-os:~/Escritorio/DSD_P3$ cd Escritorio/DSD_P3/
raulrguez@pop-os:~/Escritorio/DSD_P3$ java -cp . -Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy Servidor localhost 1
2 Servidores enlazados correctamente
Servidor iniciado

```

```
1
Introduzca el nombre de usuario:
pepe

Menú sistema RMI cliente-servidor:
[1] Registrar
[2] Donar
[3] Consultar usuarios locales
[4] Consultar usuarios globales
[5] Consultar total local
[6] Consultar total global
[7] Salir

Elige la opción que quieras realizar:

```