

## Memoria Práctica 3

---

Mario López González

3 de mayo de 2022



# Índice

<b>1</b>	<b>Practica 3: RMI - Ejemplos</b>	<b>3</b>
1.1	Ejemplo 1: Monohebra . . . . .	3
1.2	Ejemplo 2: Multihebra . . . . .	3
1.3	Ejemplo 3: Contador . . . . .	3
<b>2</b>	<b>Practica 3: RMI - Donaciones</b>	<b>3</b>
2.1	Clase cliente . . . . .	4
2.2	Clase servidor . . . . .	5
2.3	Clase replica . . . . .	5
2.3.1	Interfaz interfazClienteServidor . . . . .	6
2.3.2	Interfaz interfazReplicaReplica . . . . .	7
2.3.3	Operaciones de la clase replica . . . . .	8
2.4	Clase usuario . . . . .	8
2.5	Scripts de ejecución . . . . .	9
2.5.1	Script server.sh . . . . .	9
2.5.2	Script client.sh . . . . .	10
2.6	Ejemplos de ejecución . . . . .	11

## 1. Practica 3: RMI - Ejemplos

### 1.1. Ejemplo 1: Monohebra

Es un pequeño ejemplo de programa cliente servidor al que se le pasa como argumento al cliente un id de proceso. El cliente envía una petición de escribir mensaje pasando por parámetro el id del proceso. Si el id es cero, el servidor se duerme durante 5s, tras ese tiempo se despierta e imprime por pantalla "Hebra + id\_proceso".

Los mensajes no se entrelazan ya que al ser una sola hebra, el cliente espera a que el servidor pueda servir las demandas.

### 1.2. Ejemplo 2: Multihebra

Es el mismo ejemplo anterior pero con multihebra. Los mensajes se entrelazan debido a que no hay espera por parte de las hebras. Añadiendo `synchronized` a la implementación del método `escribir_mensaje` las hebras no se pisan las unas a las otras, de este modo, los mensajes no se entrelazan.

### 1.3. Ejemplo 3: Contador

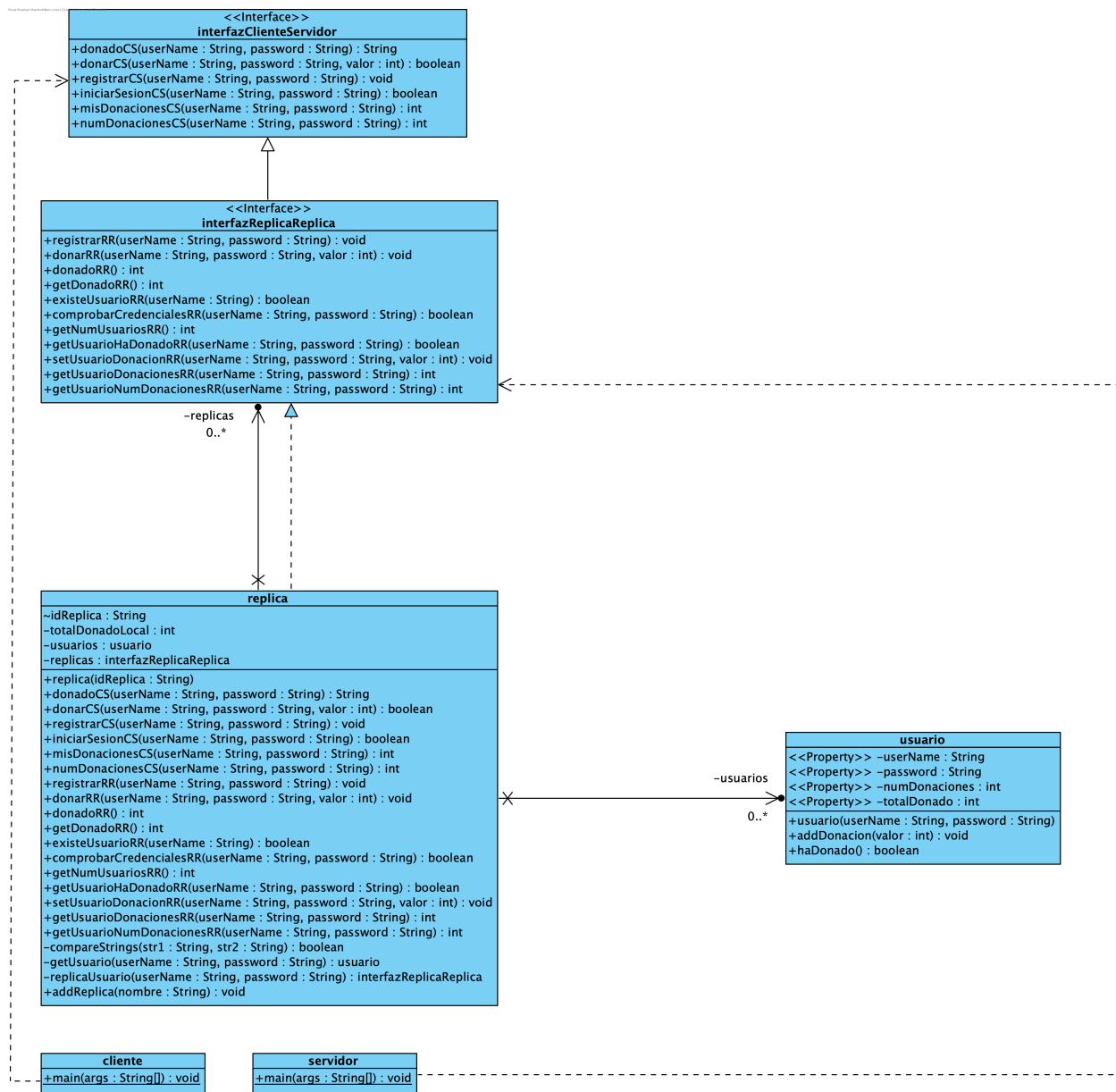
Se crea un objeto remoto instanciado como `micontador` que sólo tiene acceso a los métodos de la interfaz `icontador.java`.

El programa cliente pone un valor inicial en el contador del servidor y lo incrementa 1000 veces. Al final imprime el tiempo medio de respuesta que ha tardado en incrementar el contador 1000 veces y el número de llamadas realizadas.

## 2. Practica 3: RMI - Donaciones

Para la implementación de este ejercicio he creado cuatro clases: cliente, servidor, replica y usuario.

- **cliente:** utilizada para que los clientes puedan conectarse a una réplica y realizar distintas operaciones.
- **servidor:** crea y lanza el número de réplicas deseado, por defecto crea dos réplicas.
- **replica:** utilizada para la comunicación entre réplicas y entre cliente y servidor.
- **usuario:** utilizada por la clase réplica para registrar el nombre de usuario, contraseña, donaciones de dicho usuario y el número de donaciones realizadas por el mismo.



## 2.1. Clase cliente

La clase cliente permite al usuario decidir a qué réplica conectarse, pasando como argumento el id de la réplica a la que conectarse (`donaciones_repX`), si no se indica nada se conectará a la réplica 1. Además, contiene un main interactivo en el que se puede elegir qué operación realizar:

- **iniciar sesión:** permite iniciar sesión con un usuario y contraseña para tener acceso al resto de operaciones.
- **registro:** permite crear un usuario y contraseña siempre y cuando el usuario no haya sido utilizado previamente.
- **donar:** permite a un cliente identificado realizar una operación si es positiva.
- **total donaciones:** permite consultar al cliente la cantidad total de donativos realizados en todas las réplicas, si ha donado al menos una vez.

- **mis donaciones:** permite al cliente consultar el total donado por el usuario con el que ha iniciado sesión.
- **mi número de donaciones:** permite al cliente consultar cuántas veces ha donado con el usuario con el que ha iniciado sesión.
- **cerrar sesión:** cierra la sesión del usuario que ha iniciado sesión.
- **salir:** cierra el programa.

La conexión a la réplica se realiza por medio de una interfaz que proporciona las operaciones previamente descritas excepto cerrar sesión y salir.

## 2.2. Clase servidor

La clase servidor es la encargada de crear el número de réplicas pasado como argumento, si no se indica ninguno, creará dos réplicas por defecto. Se crean réplicas con id donaciones\_repX con

$$X \in [2, args[0]],$$

siendo args[0] el número de réplicas pasadas como argumento.

```

1 public class servidor {
2     public static void main(String[] args) {
3         int numReplicas = 2;
4         if(args.length == 1 && Integer.parseInt(args[0]) > 2)
5             numReplicas = Integer.parseInt(args[0]);
6         // Crea e instala el gestor de seguridad
7         if (System.getSecurityManager() == null) {
8             System.setSecurityManager(new SecurityManager());
9         }
10        try {
11            Registry reg = LocateRegistry.createRegistry(1099);
12            ArrayList<replica> replicas = new ArrayList<replica>();
13            ArrayList<String> idReplicas = new ArrayList<String>();
14
15
16            for(int i = 0; i < numReplicas; i++){
17                idReplicas.add("donaciones_rep" + (i+1));
18                replica replica = new replica(idReplicas.get(i));
19                Naming.rebind(idReplicas.get(i), replica);
20                replicas.add(replica);
21            }
22
23            for(int i = 0; i < replicas.size(); i++)
24                for(int j = 0; j < replicas.size(); j++)
25                    if(i != j)
26                        replicas.get(i).addReplica(idReplicas.get(j));
27
28            System.out.println("Servidor RemoteException | MalformedURLException o preparado");
29        } catch (RemoteException | MalformedURLException e) {
30            System.out.println("Exception: " + e.getMessage());
31        }
32    }
33 }

```

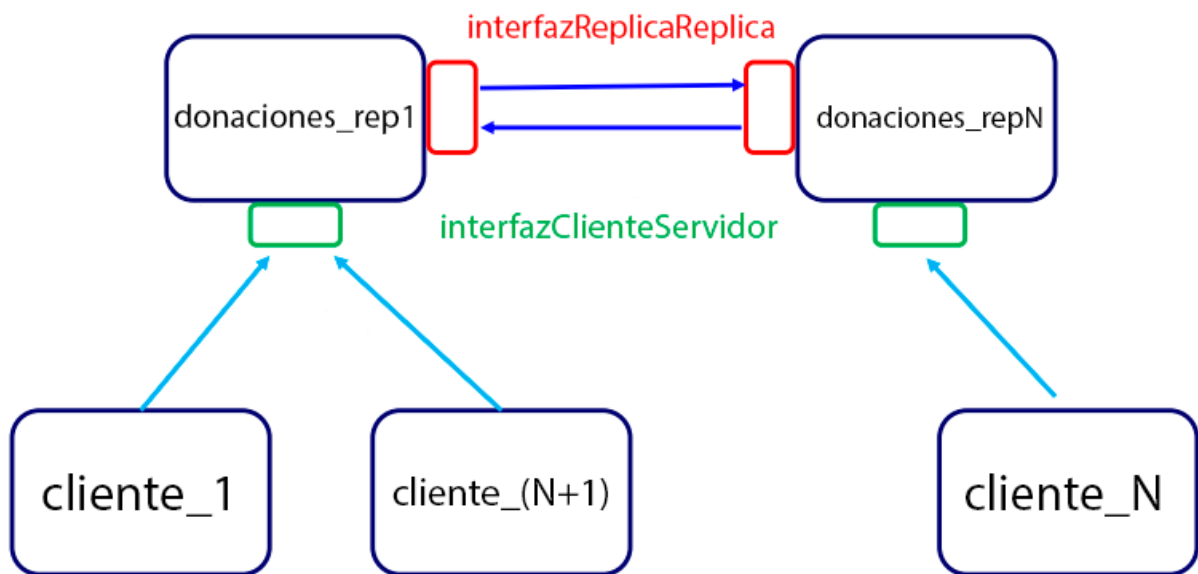
## 2.3. Clase replica

Implementa las interfaces interfazClienteServidor e interfazReplicaReplica. Estas clases son utilizadas para permitir acceso sólo a las operaciones deseadas. Por ejemplo, en la clase cliente se utiliza

la interfaz `interfazClienteServidor` para tener acceso sólo a las operaciones registro, donar, etc.

Para llevar a cabo el seguimiento de los clientes, tiene una lista de usuarios que se incrementa cuando se registra un usuario nuevo. Para saber a qué réplicas puede acceder hace uso de una lista de réplicas que se añaden desde el servidor. El total de donaciones acumulado se guarda en la variable `totalDonadoLocal` que se incrementa cuando un usuario identificado dona.

En general, cuando una réplica recibe una operación de un cliente, comprueba si el cliente está registrado en la propia réplica. En caso contrario, pregunta a las demás réplicas si alguna tiene registrado a dicho cliente. Si el cliente está registrado, se llama a la operación correspondiente de la réplica que contiene al cliente.



### 2.3.1. Interfaz `interfazClienteServidor`

Únicas operaciones accesibles por los clientes.

```
1 public interface interfazClienteServidor extends Remote {
2     public String donadoCS(String userName, String password) throws RemoteException;
3
4     public boolean donarCS(String userName, String password, int valor) throws RemoteException;
5
6     public void registrarCS(String userName, String password) throws RemoteException;
7
8     public boolean iniciarSesionCS(String userName, String password) throws RemoteException;
9
10    public int misDonacionesCS(String userName, String password) throws RemoteException;
11
12    public int numDonacionesCS(String userName, String password) throws RemoteException;
13 }
```

### 2.3.2. Interfaz interfazReplicaReplica

Extiende de la interfaz anterior ya que utilizará dichas operaciones para la comunicación entre réplicas además de tener otras más.

La operación más relevante es la de registrar un cliente, primero se comprueba que el cliente no está ya registrado con un nombre de usuario pasado como argumento en ninguna réplica, en caso de que no lo esté se envía una solicitud de registro a la réplica con menor número de clientes registrados.

```
1 public interface interfazReplicaReplica extends interfazClienteServidor{
2
3     public void registrarRR(String userName, String password) throws RemoteException;
4
5     public void donarRR(String userName, String password, int valor) throws RemoteException;
6
7     public int donadoRR() throws RemoteException;
8
9     public int getDonadoRR() throws RemoteException;
10
11     public boolean existeUsuarioRR(String userName) throws RemoteException;
12
13     public boolean comprobarCredencialesRR(String userName, String password) throws RemoteException;
14
15     public int getNumUsuariosRR() throws RemoteException;
16
17     public boolean getUsuarioHaDonadoRR(String userName, String password) throws RemoteException;
18
19     public void setUsuarioDonacionRR(String userName, String password, int valor) throws RemoteException;
20
21     public int getUsuarioDonacionesRR(String userName, String password) throws RemoteException;
22
23     public int getUsuarioNumDonacionesRR(String userName, String password) throws RemoteException;
24 }
```

Las operaciones más relevantes para la comunicación entre réplicas son las siguientes:

- **existeUsuarioRR**: comprueba si existe o no un usuario con el nombre pasado como argumento.
- **comprobarCredencialesRR**: comprueba si existe o no un usuario con el nombre y contraseña pasados como argumentos.
- **getNumUsuariosRR**: devuelve el número de usuarios registrados en cada réplica.
- **getUsuarioHaDonadoRR**: comprueba si ha donado o no un usuario en concreto.
- **setUsuarioDonacionRR**: incrementa el total donado de un cierto usuario.
- **getUsuarioDonacionRR**: devuelve el total donado por un usuario concreto.
- **getUsuarioNumDonacionesRR**: devuelve el número total de donaciones realizadas por un usuario.

### 2.3.3. Operaciones de la clase replica

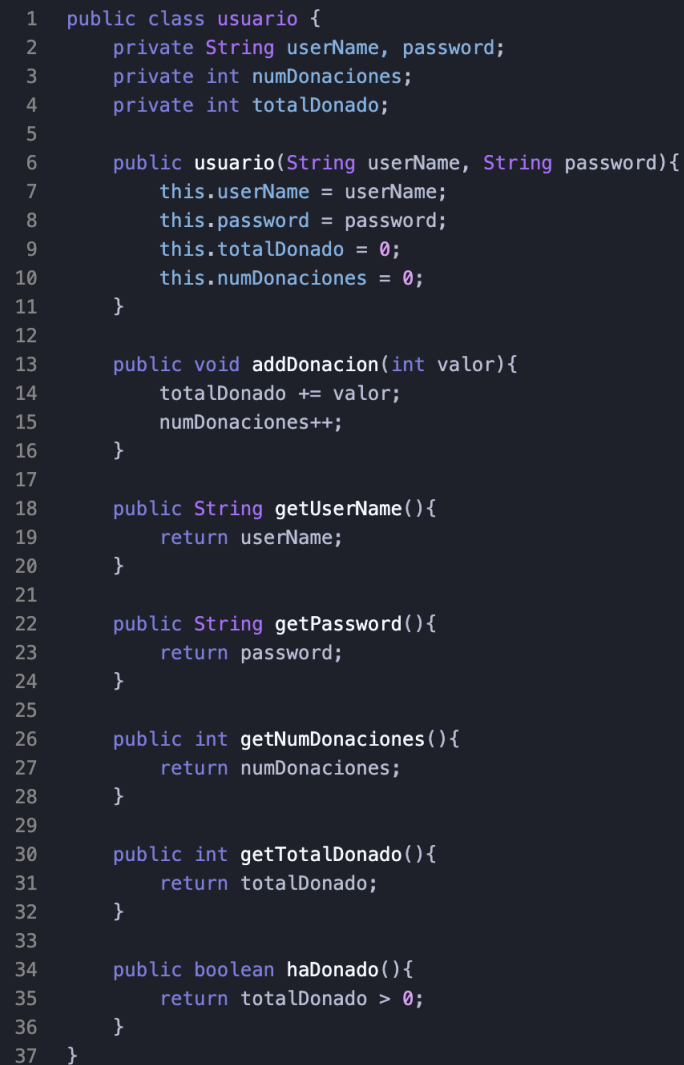
```
1 private boolean compareStrings(String str1, String str2){
2     if(str1.compareTo(str2) == 0)
3         return true;
4     return false;
5 }
6
7 private usuario getUsuario(String userName, String password){
8     for(int i = 0; i < usuarios.size(); i++)
9         if(compareStrings(userName, usuarios.get(i).getUserName()) && compareStrings(password, usuarios.get(i).getPassword()))
10            return usuarios.get(i);
11
12     return null;
13 }
14
15 private interfazReplicaReplica replicaUsuario(String userName, String password) throws RemoteException{
16     boolean existeUsuario = comprobarCredencialesRR(userName, password);
17     if(existeUsuario)
18         return this;
19
20     for(int i = 0; i < replicas.size() && !existeUsuario; i++){
21         existeUsuario = replicas.get(i).comprobarCredencialesRR(userName, password);
22         if(existeUsuario)
23             return replicas.get(i);
24     }
25
26     return null;
27 }
28
29 public void addReplica(String nombre){
30     try {
31         Registry mireg = LocateRegistry.getRegistry("127.0.0.1", 1099);
32         interfazReplicaReplica replica;
33         replica = (interfazReplicaReplica) mireg.lookup(nombre);
34         replicas.add(replica);
35     } catch (RemoteException | NotBoundException e) {
36         e.printStackTrace();
37     }
38 }
```

- **getUsuario**: devuelve una referencia al usuario identificado por nombre y contraseña en la lista privada de usuarios de la réplica.
- **replicaUsuario**: devuelve una referencia a la réplica que contiene el usuario identificado por nombre y contraseña.
- **addReplica**: añade una réplica identificada por el nombre pasado como argumento a la lista de réplicas.

### 2.4. Clase usuario

Clase utilizada para llevar un seguimiento de los usuarios registrados por parte de las réplicas. Se compone de un nombre de usuario, contraseña, total donado y el número de donaciones realizadas.





```

1  public class usuario {
2      private String userName, password;
3      private int numDonaciones;
4      private int totalDonado;
5
6      public usuario(String userName, String password){
7          this.userName = userName;
8          this.password = password;
9          this.totalDonado = 0;
10         this.numDonaciones = 0;
11     }
12
13     public void addDonacion(int valor){
14         totalDonado += valor;
15         numDonaciones++;
16     }
17
18     public String getUserNombre(){
19         return userName;
20     }
21
22     public String getPassword(){
23         return password;
24     }
25
26     public int getNumDonaciones(){
27         return numDonaciones;
28     }
29
30     public int getTotalDonado(){
31         return totalDonado;
32     }
33
34     public boolean haDonado(){
35         return totalDonado > 0;
36     }
37 }

```

## 2.5. Scripts de ejecución

### 2.5.1. Script server.sh


Lanza el servidor, admite como argumento un parámetro, siendo este el número de réplicas a realizar.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The terminal displays a shell script with 10 lines of code, numbered 1 through 10 on the left margin. The script includes comments in Spanish and Java commands for compilation and execution with specific JVM options.

```
1 #!/bin/sh -e
2 echo
3 echo "Compilando con javac ..."
4 javac *.java
5 sleep 2
6
7 echo
8 echo "Lanzando el servidor"
9 java -cp . -Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy servidor $1
10 sleep 2
```

### 2.5.2. Script client.sh

Lanza el cliente, admite como argumento un parámetro, siendo este el id de la réplica a la que conectarse.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The terminal displays a shell script with 5 lines of code, numbered 1 through 5 on the left margin. The script includes comments in Spanish and Java commands for compilation and execution with specific JVM options.

```
1 javac *.java
2 echo
3 echo "Lanzando el cliente"
4 echo
5 java -cp . -Djava.security.policy=server.policy cliente $1
```

## 2.6. Ejemplos de ejecución

```
mariolopg donaciones (P3) $ ./client.sh donaciones_rep3

Lanzando el cliente

Conectado a réplica con id: donaciones_rep3

*****
*   Seleccionar operación   *
*****
*   i -> iniciar sesion    *
*   r -> registro          *
*   d -> donar              *
*   t -> total donaciones  *
*   m -> mis donaciones    *
*   n -> mi numero donaciones *
*   c -> cerrar sesion     *
*   s -> salir              *
*****
d

Operacion seleccionada -> donar
Cantidad a donar: 40

ERROR -> registro necesario para donar

*****
*   Seleccionar operación   *
*****
*   i -> iniciar sesion    *
*   r -> registro          *
*   d -> donar              *
*   t -> total donaciones  *
*   m -> mis donaciones    *
*   n -> mi numero donaciones *
*   c -> cerrar sesion     *
*   s -> salir              *
*****
r

Operacion seleccionada -> registro
Nombre de usuario: mario
Contraseña: mario

Sesión iniciada correctamente

*****
*   Seleccionar operación   *
*****
*   i -> iniciar sesion    *
*   r -> registro          *
*   d -> donar              *
*   t -> total donaciones  *
*****
```

```
mariolopg donaciones (P3) $ ./server.sh 4

Compilando con javac ...

Lanzando el servidor
Servidor RemoteException | MalformedURLExceptionor preparado
Usuario no registrado
Registro realizado en donaciones_rep3
```

```
mariolopg donaciones (P3) $ ./client.sh donaciones_rep3

*****
*   Seleccionar operación   *
*****
*   i -> iniciar sesion    *
*   r -> registro          *
*   d -> donar              *
*   t -> total donaciones  *
*   m -> mis donaciones    *
*   n -> mi numero donaciones *
*   c -> cerrar sesion     *
*   s -> salir              *
*****
t

Operacion seleccionada -> total donaciones
Total donado: ???€

*****
*   Seleccionar operación   *
*****
*   i -> iniciar sesion    *
*   r -> registro          *
*   d -> donar              *
*   t -> total donaciones  *
*   m -> mis donaciones    *
*   n -> mi numero donaciones *
*   c -> cerrar sesion     *
*   s -> salir              *
*****
d

Operacion seleccionada -> donar
Cantidad a donar: 10
Donación de 10€ realizada correctamente

*****
*   Seleccionar operación   *
*****
*   i -> iniciar sesion    *
*   r -> registro          *
*   d -> donar              *
*   t -> total donaciones  *
*   m -> mis donaciones    *
*   n -> mi numero donaciones *
*   c -> cerrar sesion     *
*   s -> salir              *
*****
n

Has donado 1 veces
```

```
mariolopg donaciones (P3) $ ./server.sh 4

Compilando con javac ...

Lanzando el servidor
Servidor RemoteException | MalformedURLExceptionor preparado
Usuario no registrado
Registro realizado en donaciones_rep3
Donación recibida de 10€
Donacion realizada en donaciones_rep3
```

```
~/Desktop/UGR/Tercero/Segundo Cuatrimestre/DSD/P3/donaciones
*      r -> registro      *
*      d -> donar        *
*      t -> total donaciones *
*      m -> mis donaciones *
*      n -> mi numero donaciones *
*      c -> cerrar sesion *
*      s -> salir        *
*****
r
Operacion seleccionada -> registro
Nombre de usuario: mesa
Contraseña: mesa
Sesión iniciada correctamente

*****
*      Seleccionar operación      *
*****
*      i -> iniciar sesion      *
*      r -> registro            *
*      d -> donar                *
*      t -> total donaciones     *
*      m -> mis donaciones       *
*      n -> mi numero donaciones *
*      c -> cerrar sesion        *
*      s -> salir                *
*****
d
Operacion seleccionada -> donar
Cantidad a donar: 70
Donación de 70€ realizada correctamente

*****
*      Seleccionar operación      *
*****
*      i -> iniciar sesion      *
*      r -> registro            *
*      d -> donar                *
*      t -> total donaciones     *
*      m -> mis donaciones       *
*      n -> mi numero donaciones *
*      c -> cerrar sesion        *
*      s -> salir                *
*****
t
Operacion seleccionada -> total donaciones
Total donado: 80€

~/Desktop/UGR/Tercero/Segundo Cuatrimestre/DSD/P3/donaciones
mariolopg donaciones (P3) $ ./server.sh 4

Compilando con javac ...

Lanzando el servidor
Servidor RemoteException | MalformedURLException preparado
Usuario no registrado
Registro realizado en donaciones_rep3
Donación recibida de 10€
Donación realizada en donaciones_rep3
Registro realizado en donaciones_rep1
Donación recibida de 70€
Donación realizada en donaciones_rep1
Solicitud total donado en donaciones_rep1
[]
```