GRADO EN INGENIERÍA INFORMÁTICA

DESARROLLO DE SISTEMAS DISTRIBUIDOS

UNIVERSIDAD
DE GRANADA

P3 : RMI. Desarrollo de Sistemas Distribuidos

Juan Miguel Acosta Ortega (*acostaojuanmi@correo.ugr.es*)

2 de mayo de 2024

Índice

Objetivos

La primera parte consiste en seguir los pasos para implementar y probar los ejemplos del guión. Se ha de demostrar que funcionan y las diferencias hay entre ellos. Además se responden a las cuestiones planteadas sobre el ejemplo 2.

La segunda parte consiste en implementar el servidor replicado que se especifica en el guión. Sin embargo, habrá que extender el servidor con funcionalidad adicional para optar a mejor nota.

1. Primera parte de la práctica : Revisión de ejemplos

1.1. Primer ejemplo

Para comenzar a analizar el primer ejemplo, se han de generar en primer lugar tanto los archivos necesarios (interfaz de servicio, el servicio y el cliente), y además el fichero "server.policy"que otorga todos los permisos a la aplicación para evitar fallos a la hora de acceder a un puerto específico en "localhost".

Una vez todo preparado compilamos con javac y levantamos los servicios con la macro preparada en el guión. Esto ejecuta el código cuya salida es la siguiente:

```
bash macro.sh

Lanzando el ligador de RMI

Compilando con javac ...

Lanzando el servidor
Ejemplo bound

Lanzando el primer cliente

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

Hebra 0

Lanzando el segundo cliente

Buscando el objeto remoto
Invocando el objeto remoto
Recibida peticion de proceso: 3

Hebra 3
```

Listing 1: Lanzamiento del ejemplo 1

A continuación pasamos a explicar paso a paso qué es lo que sucede internamente:

En primer lugar explicamos la implementación de la interfaz remota:

```
//Fichero: Ejemplo.java
//Implementa la Interfaz remota
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;

import java.lang.Thread;
public class Ejemplo implements Ejemplo_I {

    public Ejemplo() {
        super();
    }

    public void escribir_mensaje(int id_proceso) {
        System.out.println("Recibida peticion de proceso: " + id_proceso);
        if (id_proceso == 0) {
            try {
                System.out.println("Empezamos a dormir");
```

```
                    Thread.sleep(5000);
                    System.out.println("Terminamos de dormir");
                } catch (Exception e) {
                    System.err.println("Ejemplo exception:");
                    e.printStackTrace();
                }
            }
            System.out.println("\nHebra " + id_proceso);
        }

        public static void main(String[] args) {
            if (System.getSecurityManager() == null) {
                System.setSecurityManager(new SecurityManager());
            }
            try {
                String nombre_objeto_remoto = "Ejemplo_I";
                Ejemplo_I prueba = new Ejemplo();
                Ejemplo_I stub
                        = (Ejemplo_I) UnicastRemoteObject.exportObject(prueba, 0);
                Registry registry = LocateRegistry.getRegistry();
                registry.rebind(nombre_objeto_remoto, stub);
                System.out.println("Ejemplo bound");
            } catch (Exception e) {
                System.err.println("Ejemplo exception:");
                e.printStackTrace();
            }
        }
    }
```

Listing 2: Implementación de interfaz remota

En este fichero se implementa el método definido en la interfaz "Ejemplo-I"(escribir-mensaje) , en este simplemente se imprime un mensaje indicando que se recibió una petición de un proceso con un determinado ID. Si el ID del proceso es 0, espera 5 segundos (simulado con Thread.sleep(5000)) antes de continuar. Luego, imprime información sobre la hebra (hilo) que se ejecuta.

En el main, y como se explica en las transparencias se crea o utiliza (en función de si existe ya o no) un Security Manager (se encarga de aplicar políticas de seguridad para controlar el acceso a recursos críticos del sistema, como archivos, red y memoria. Su función principal es proteger al sistema de código malicioso y garantizar que las aplicaciones se ejecuten de forma segura.). Luego instancia el objeto remoto "Ejemplo", lo exporta como un objeto remoto usando "UnicastRemoteObject.exportObject", y la vincula al registro RMI utilizando "registry.rebind".

```

    //Fichero: Cliente_Ejemplo.java
    //C digo del cliente
    import java.rmi.registry.LocateRegistry;

    import java.rmi.registry.Registry;
    public class Cliente_Ejemplo {

        public static void main(String args[]) {
            if (System.getSecurityManager() == null) {
                System.setSecurityManager(new SecurityManager());
            }
            try {
                String nombre_objeto_remoto = "Ejemplo_I";
                System.out.println("Buscando el objeto remoto");
                Registry registry = LocateRegistry.getRegistry(args[0]);
                Ejemplo_I instancia_local = (Ejemplo_I) registry.lookup(nombre_objeto_remoto);
                System.out.println("Invocando el objeto remoto");
                instancia_local.escribir_mensaje(Integer.parseInt(args[1]));
            } catch (Exception e) {
                System.err.println("Ejemplo_I exception:");
                e.printStackTrace();
            }
        }
```

```
25      }
```

Listing 3: Código del cliente

Por otro lado en el cliente también se instancia un SecurityManager y se hace una búsqueda de objeto remoto en el registro RMI con "LocateRegistry.getRegistry". En primer lugar se le pasa la dirección del servidor como primer argumento (localhost), a continuación se instancia un objeto local con la información del objeto remoto, y este se busca con "(tipo-objeto)registry.lookup(nombre-objeto-remoto)". Por último se llama al método remoto con el objeto local (realmente el procedimiento será remoto) con el id pasado como segundo argumento por línea de comando.

Para concluir el análisis vemos que tiene sentido la salida, pues el primer id es 0, y esto hace que se haga print de "Empezamos a dormir"y "terminamos de dormir"con una diferencia de 5 segundos, mientras que la segunda petición al tener id diferente de 0, no lo hace.

Esta sería la arquitectura resultante de la implementación de RMI:

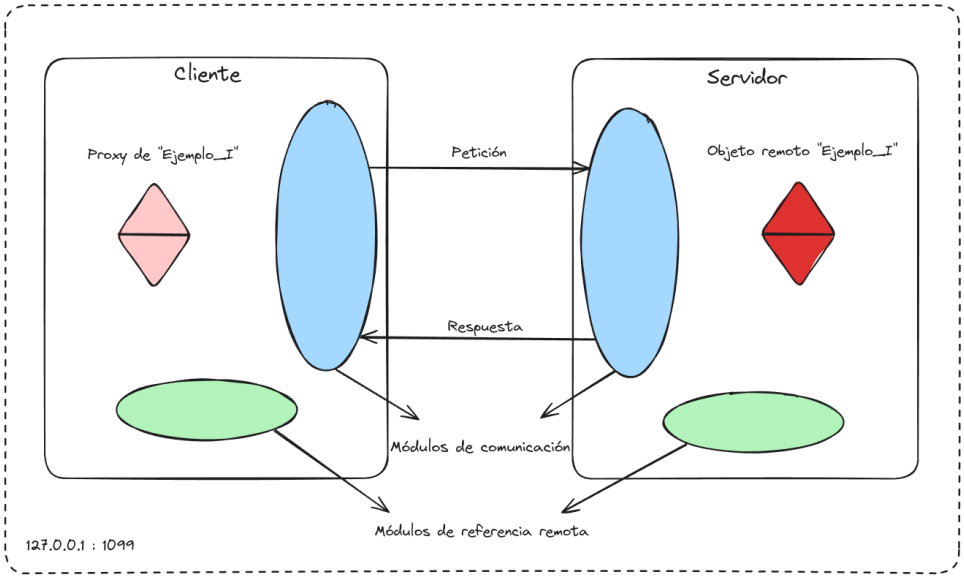

Figura 1: Arquitectura RMI Cliente/Servidor.

1.2. Segundo ejemplo

En este segundo ejemplo la arquitectura es similar. El cambio significativo es que el método "escribir-mensaje"no recibe un entero id-petición , si no que recibe un String con la información de la hebra que lo ha llamado. Se hará tanto un vector de clientes como uno de hebras de tamaño correspondiente a número de hebras a crear (argumento por línea de comandos). Al igual que antes si la petición es hecha por la hebra (cliente) 0 el servidor hace una espera de 5 segundos además de notificar qué hebra entró, cosa que también hace con el resto. Podemos ver un ejemplo de salida con 5 hebras:

```
1
2     Lanzando el servidor
3     Ejemplo bound
4
5     Lanzando cliente multi-threaded ASINCRONO ...
6
7     Buscando el objeto remoto
8     Buscando el objeto remoto
9     Buscando el objeto remoto
10    Buscando el objeto remoto
11    Buscando el objeto remoto
12    Invocando el objeto remoto
13    Invocando el objeto remoto
14    Invocando el objeto remoto
15    Invocando el objeto remoto
16
17    Entra Hebra Cliente 0
18
19    Entra Hebra Cliente 1
20
21    Entra Hebra Cliente 3
22    Empezamos a dormir
23    Sale Hebra Cliente 1
24    Sale Hebra Cliente 3
25
26    Entra Hebra Cliente 4
27    Sale Hebra Cliente 4
28    Invocando el objeto remoto
29
30    Entra Hebra Cliente 2
31    Sale Hebra Cliente 2
32    Terminamos de dormir
33    Sale Hebra Cliente 0
```

Listing 4: Ejemplo con 5 hebras

En esta salida vemos que se ha buscado e invocado el objeto remoto n veces (una por cada hebra/cliente que ha comenzado a ejecutar el método run. Además vemos que lo hacen de manera asíncrona, esto se ve reflejado en la salida estándar.

En este ejemplo multihebrado se generan n clientes que solicitarán el uso del objeto remoto.

1.2.1. ¿Qué ocurre con las hebras cuyo nombre acaba en 0?¿Qué hacen las demás hebras?¿Se entrelazan los mensajes?

La hebra 0, como ya he comentado anteriormente, es la que duerme durante 5 segundos, y esto es debido a la línea 18 del servidor "mensaje.endsWith("0")". Las demás no duermen e indican, como también lo hace la 0, cuando "entran"y cuando "salen". Los mensajes se entrelazan debido a diversos factores. En RMI (Remote Method Invocation), la ejecución de las hebras es asíncrona debido a la naturaleza de la comunicación remota.

Cuando un cliente invoca un método en un objeto remoto a través de RMI, normalmente se establece una conexión de red entre el cliente y el servidor. Esta conexión introduce latencia en la comunicación debido a factores

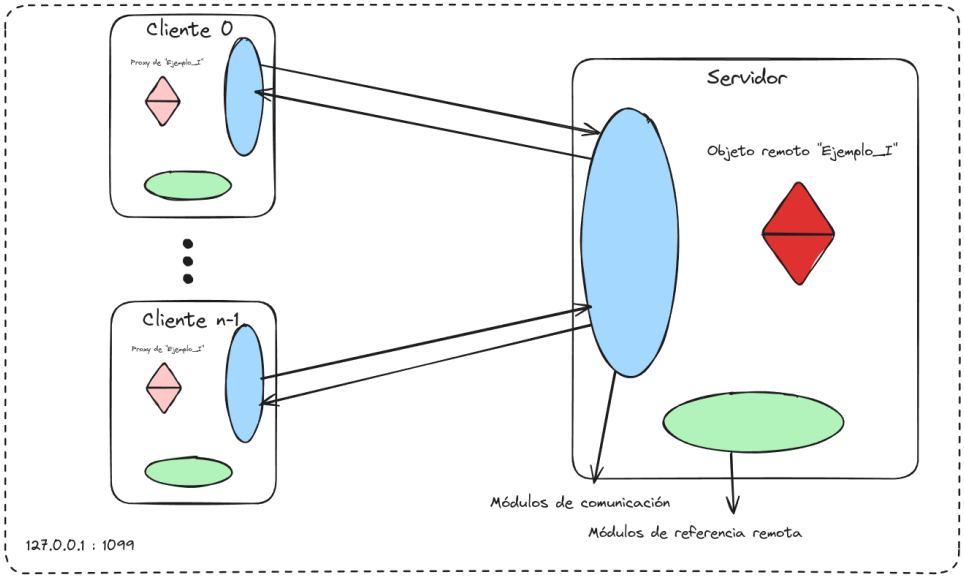

Figura 2: Arquitectura RMI Cliente/Servidor Multi-hebrado.

como la velocidad de la red y la carga del servidor.

Esta asincronía permite que el cliente siga siendo receptivo y pueda realizar otras tareas mientras espera la respuesta del servidor, lo que mejora la eficiencia y la capacidad de respuesta del sistema en general.

Esto también implica que las implementaciones de los objetos remotos han de ser seguras (thread-safe).

De esta manera no se garantiza la exclusión mutua, ya que más de una hebra podría acceder concurrentemente al mismo recurso.

1.2.2. Prueba a introducir el modificador synchronized en el método de la implementación remota y trata de entender las diferencias de ejecución de los dos programas

```
Lanzando cliente multi-threaded SINCRONO ...

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

Entra Hebra Cliente 4
Sale Hebra Cliente 4

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

```

```
    Entra Hebra Cliente 2
    Sale Hebra Cliente 2

    Entra Hebra Cliente 3
    Sale Hebra Cliente 3

    Entra Hebra Cliente 1
    Sale Hebra Cliente 1
```

Listing 5: Ejemplo con 5 hebras, modificador synchronized

La diferencia principal en esta ejecución es que todas las hebras esperan a que se complete la invocación del método remoto antes de terminar.

Esto garantiza que las hebras no se entrelacen durante la invocación del método remoto y que todas las operaciones se realicen de manera ordenada y sincronizada.

1.3. Tercer ejemplo

En este ejemplo de contador distribuido el objeto remoto se instancia y controla de manera externa al servidor, es decir, se crea por una parte el objeto y por otra el servicio (que importa el paquete contador para poder crear el objeto). El servidor accede a varios métodos externos para interactuar con este.

Por otro lado el cliente se encarga de recoger la dirección del servidor (en este caso será localhost o 127.0.0.1), de crear una instancia local de contador (objeto sustituto "stub"), y lo incrementa mil veces llamando al método del objeto remoto "incrementar()".

Para ejecutar este ejemplo he utilizado el guión para usar NetBeans y un ejemplo de salida sería el siguiente:

```
run:
Escriba el nombre o IP del servidor:
localhost
Poniendo contador a 0
Incrementando...
Media de las RMI realizadas = 0.067 msegs
RMI realizadas = 1000
BUILD SUCCESSFUL (total time: 25 seconds)
```

Listing 6: Salida del ejemplo 3

La diferenciación más significativa de este ejemplo como ya se ha comentado es que separa el modelo del objeto remoto del servidor, y que tanto este como el cliente se lanzan sin argumentos, se pide la dirección del servidor desde consola.

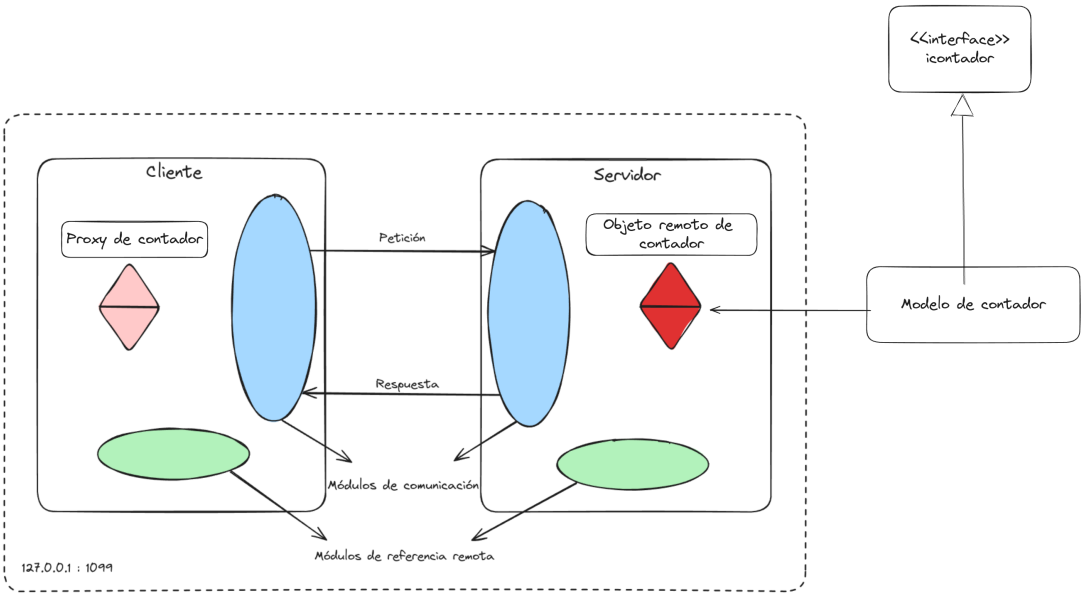

Figura 3: Cliente / Servidor RMI con modelo de objeto remoto separado del servidor.

Tras haber comparado los tres ejemplos también podemos observar que hay diversas maneras de registrar el stub al nombre del registro, pues en unos ejemplos utilizamos "Runtime.getRuntime().exec(rmiregistry"); ", y en otros "LocateRegistry.createRegistry(int port); ".
Sucede lo mismo con la asociación de un stub a un registro desde Java, se puede hacer de diferentes maneras, con java.rmi.registry.Registry "Registry registro=LocateRegistry.getRegistry(); registro.rebind(nombre,stub); "o mediante java.rmi.Naming "Naming.rebind(nombre, stub); "

2. Sistema de donaciones con RMI

El sistema a desarrollar como se detalla en la práctica es un sistema de donaciones que permita a usuarios registrarse, iniciar sesión y donar, además de otras funcionalidades y restricciones.

En concreto en esta primera versión he desarrollado dos réplicas (ambas con una instancia del objeto remoto "Donación del que hablaremos más adelante) ". Una se identifica con el nombre de ServerEUW, y la réplica con ServerNA.

En el lado de servidor lanzaremos tanto el servidor como la réplica para que el cliente se comunique con estos. Como vimos en los ejemplo anteriores asociamos un nombre a un objeto remoto y lo vinculamos al registro de RMI en el servidor:

```
1
2     String servidor = "ServerEUW";
3     String replica = "ServerNA";
4
5     System.setProperty("java.rmi.server.hostname","localhost");
6     Registry reg=LocateRegistry.createRegistry(1099);
7     Donacion donacion = new Donacion(servidor,replica);
8     Naming.rebind(servidor, donacion);
```

Listing 7: Servidor donaciones

La réplica tiene un código similar, simplemente cambia de orden los nombres , es decir, el atributo "Servidor"del objeto "Donacion"de la Réplica será "ServerNAza que hace referencia al servidsor actual, y la réplica en su caso será "ServerEUW".

El servidor ha de proveer las operaciones registrarsez "donar"(operación que no se puede realizar sin estar registrado). Estas operaciones conllevan a su vez más suboperaciones que se han de tener en cuenta.
Por ejemplo, a la hora de registrarse se ha desarrolado el siguiente código (código que se encuentra en la clase del objeto remoto Donación):

```
1
2     @Override
3     public boolean registrar(String usuario, String contrasenia) throws RemoteException {
4         boolean registro =false;
5
6         if (this.usuarioEnSistema(usuario)){
7             return registro;
8         }else{
9             idonacion_replica replica = this.getReplica(this.replica1);
10            if (replica.usuarioEnSistema(usuario)){
11                return registro;
12            }
13            else{
14                registro = true;
15                if (this.usuariosEnReplica() <= replica.usuariosEnReplica()){
16                    this.almacenarUsuario(usuario,contrasenia);
17                }else{
18                    replica.almacenarUsuario(usuario,contrasenia);
19                }
20            }
21        }
22
23        return registro;
24    }
```

Listing 8: Registrarse en el sistema

En este método en primer lugar comprobamos que el usuario no se encuentre ya registrado previamente en el servidor principal ni en la réplica. Para esto usamos el método üsuarioEnSistema(usuario).ªl que le pasamos el String usuario para ver si se encuentra en el atributo üsuarios"que es un Map<String,Usuario>para almacenar tanto

el nombre de los usuarios registrados como su información (Clase Usuario que contiene número de donaciones, total donado, contraseña ...).
A continuación si no se encuentra registrado vemos en qué réplica hay menos usuarios registrados, y en esa será en la que se registre el cliente.

Para trabajar con la réplica y sus métodos (Comunicación entre las dos réplicas) se han implementado dos interfaces, una ïdonacionçon los métodos necesarios en el sistema, y otra ïdonacion replica"que será la que contenga los métodos que se han de utilizar desde la réplica del servidor actual.

Para simular un inicio de sesión he implementado un método replicaUsuario(String nombre, String contraseña)"que verifica que el usuario esté registrado en alguna réplica, y en caso afirmativo comprueba que lo esté con la contraseña que inserta el cliente. Llegados a este punto devuelvo la réplica en la que está registrado para así simplificar las operaciones que realice más adelante el cliente (donar, listado de donantes ...).

```
1
2       @Override
3       public String replicaUsuario(String usuario, String contrasenia) throws RemoteException {
4           String id_replica = "";
5
6           if (this.usuarioEnSistema(usuario)){
7
8               if (contrasenia.equals(this.usuarios.get(usuario).getContrasenia())){
9
10                  id_replica = this.servidor;
11
12              }
13
14          } else {
15
16              idonacion_replica replica = this.getReplica(this.replica1);
17
18              if (replica.usuarioEnSistema(usuario)){
19
20                  if (contrasenia.equals(replica.getUsuariosEnReplica(this.replica1).get(usuario
    ).getContrasenia())){
21
22                      id_replica = this.replica1;
23
24                  }
25              }
26          }
27
28          return id_replica;
29      }
```

Listing 9: Iniciar sesión (devolviendo la réplica en la que se encuentra el usuario)

Una vez el cliente se haya registrado e iniciado sesión en el sistema, puede hacer uso del resto de funcionalidades del sistema, en este caso puede :

- Donar
- Consultar el total que ha donado en el sistema
- Listado de usuarios en el sistema
- Donaciones realizadas por mí.
- Comprobar el podio de donantes en el sistema
- Comprobar el listado de donantes en el sistema

Aunque realmente hasta que no done (se convierte en donante) no se le permite hacer uso de ninguna acción excepto donar.
Para ello se implementa un método "haDonadoEnElSistema(Sting usuario, String r)":

```
@Override
public boolean haDonadoEnSistema(String usuario, String r) throws RemoteException {

    boolean donante= false;

    if (r.equals(this.servidor)){
        float donado = this.obtenerTotalDonadoDelCliente(usuario);
        if (donado == 0){
            return donante;
        }
        donante=true;

    }else{
        idonacion_replica replica = this.getReplica(this.replica1);
        float donado = replica.obtenerTotalDonadoDelCliente(usuario);
        if (donado == 0){
            return donante;
        }
        donante=true;

    }

    return donante;
}
```

Listing 10: Comprobar si el usuario es donante

Tras esto el cliente podrá hacer uso del resto de funcionalidades.

Para hacer la acción de "donar"por parte del usuario se implementa el método donar en el objeto remoto al que se le pasa el nombre de usuario y la cuantía a donar. En este se comprueba en qué réplica se encuentra el usuario y se incrementa tanto su información personal (en el objeto de tipo Usuario se incrementa en uno el número de donaciones y totalDonado += cuantia) y el subtotal de la réplica. En otro método se obtiene el total donado al sistema sumando el subtotal de cada réplica.

```
@Override
public void donar(String usu, float cuantia) throws RemoteException {

    System.out.println("Se dona "+ cuantia + " euros por parte del usuario "+ usu +"\n");

    if (this.usuarioEnSistema(usu)){
        this.actualizarDonaciones(cuantia, usu);
        this.incDonadoReplica(cuantia);
    }else{
       idonacion_replica replica = this.getReplica(this.replica1);
       replica.actualizarDonaciones(cuantia, usu);
       replica.incDonadoReplica(cuantia);
    }
}
```

Listing 11: Método donar en objeto remoto

```
@Override
public float totalDonadoServidor() throws RemoteException {
    float total=0;

    total += this.getDonado();

    System.out.println("Total donado en el servidor: "+total);

```

```
        idonacion_replica replica = this.getReplica(this.replica1);

        total += replica.getDonado();

        System.out.println("Total donado en el sistema: "+total);

        return total;
    }
```

Listing 12: Obtener el total donado en el sistema

Además de estas operaciones se puede obtener tanto el listado de registrados en el sistema como el listado de donantes. El primer método devuelve una lista con la síntesis de los registrados en una réplica y la otra, y el método que devuelve a los donantes devuelve una síntesis de los donantes de una réplica y otra. Los donantes de cada réplica se van actualizando mediante cada donación.

```

    @Override
    public ArrayList<String> getDonantesEnSistema() throws RemoteException {

        ArrayList<String> donantes = new ArrayList<>();

        donantes.addAll(this.getDonantes());

        idonacion_replica replica = this.getReplica(this.replica1);

        donantes.addAll(replica.getDonantes());

        return donantes;

    }

    @Override
    public ArrayList<String> getDonantesEnReplica(String r) throws RemoteException {

        ArrayList<String> donantes = new ArrayList<>();

        if (r.equals(this.servidor)){
            donantes = this.getDonantes();
        }else{
            idonacion_replica replica = this.getReplica(this.replica1);
            donantes = replica.getDonantes();
        }

        return donantes;
    }
```

Listing 13: Métodos para la obtención de los donantes del sistema completo

Para comprobar las donaciones realizadas por un usuario (número y total) usamos los métodos de la clase Usuario (al actualizar en donar también al usuario sólo hemos de usar los getters):

```

    @Override
    public float obtenerTotalDonadoDelCliente(String usuario) throws RemoteException {
        Usuario usu = this.usuarios.get(usuario);
        return usu.getTotalDonaciones();
    }

    @Override
    public int obtenerNumeroDonaciones(String usuario) throws RemoteException {
        Usuario usu = this.usuarios.get(usuario);
        return usu.getNumDonaciones();
    }
```

Listing 14: Métodos para la obtención de la información de donaciones de un usuario en concreto

Además se ha implementado otro método adicional que muestra un podio de donantes por réplica que devuelve un Map<String nombre, Float cuantía>. En este se ordenan de mayor a menor los usuarios por total donado:

```
    @Override
    public Map<String, Float> getMayoresDonantesReplica(String r) throws RemoteException {
        Map<String,Float> podio_replica = new LinkedHashMap<>();

        if (r.equals(this.servidor)){

            List<Usuario> lista = new ArrayList<>(this.usuarios.values());
            Collections.sort(lista, Comparator.comparingDouble(Usuario::getTotalDonaciones).
    reversed());

            for (int i = 0; i < Math.min(3, lista.size()); i++) {
                Usuario usuario = lista.get(i);
                podio_replica.put(usuario.getUsuario(), (float) usuario.getTotalDonaciones());
            }

        }else{
            idonacion_replica replica = this.getReplica(this.replica1);
            List<Usuario> lista = new ArrayList<>(replica.getUsuariosEnReplica(this.replica1).
    values());
            Collections.sort(lista, Comparator.comparingDouble(Usuario::getTotalDonaciones).
    reversed());

            for (int i = 0; i < Math.min(3, lista.size()); i++) {
                Usuario usuario = lista.get(i);
                podio_replica.put(usuario.getUsuario(), (float) usuario.getTotalDonaciones());
            }
        }

        return podio_replica;
    }
```

Listing 15: Método que recoge podio de donantes en réplica

Todos los métodos realizados para la práctica y sus suboperaciones quedan reflejados y explicados en la interfaz principal del objeto remoto "idonacion "(Adjuntado con el resto del código de la práctica).

3. Última versión del ejercicio (N réplicas)

Para lograr esta última versión con N réplicas se hace un cambio importante sobre todo en los parámetros para construir el objeto "Donacion ", y es que ahora en lugar de pasar un String Servidor y un String replica 1 pasamos un Set<String>"LinkedHashSet "para pasarle de forma ordenada los ids del servidor actual (primer id) y sus réplicas.

De esta forma la funcionalidad es similar, pero cambia la forma en la que recogemos la réplica con la que queremos trabajar, es decir, antes siempre recogíamos la réplica por "this.replica1 ". Ahora se ha de recorrer el Set de ids en busca del deseado. Para mejor comprensión dejo a continuación el método registrar de esta última versión:

```
    @Override
    public boolean registrar(String usuario, String contrasenia) throws RemoteException {
        boolean registro =false;
        Map<String,Integer> menorUsuarios = new HashMap<>();
        menorUsuarios.put(this.servidor, this.usuariosEnReplica());

        if (this.usuarioEnSistema(usuario)){
            return registro;
        }else{
            for (String id : this.conjuntoIds){
                    idonacion_replica replica = this.getReplica(id);
```

```
                    menorUsuarios.put(id, replica.usuariosEnReplica());
                    if (replica.usuarioEnSistema(usuario)){
                        return registro;
                    }
            }

            registro = true;
            //BUSCAR LA REPLICA CON MENOS USUARIOS
            int menor =0;
            String idReplica="ServerEUW";
            for (Map.Entry<String, Integer> entry : menorUsuarios.entrySet()) {
                int numRegistrados = entry.getValue();
                if (numRegistrados <= menor){
                    menor = numRegistrados;
                    idReplica = entry.getKey();
                }
            }

            if (idReplica != "ServerEUW"){
                idonacion_replica replica = this.getReplica(idReplica);
                replica.almacenarUsuario(usuario,contrasenia);
            }else{
                this.almacenarUsuario(usuario,contrasenia);
            }

        }

        return registro;
    }
```

Listing 16: Método registrar de la última versión

A continuación y por hacer más fácil la corrección y visualización de un ejemplo capturaré la salida de una entrada de tres posibles usuarios que interactúan con el sistema. En el caso siguiente se levantan "Servidor "y "Replica1 "pero se podrían levantar N réplicas y el funcionamiento (de cara al cliente) sería el mismo.

4. Ejemplo de ejecución de programa

Al componerse de distintos bucles controlados por centinela (while) el cliente puede manejar la acción de varios usuarios:

```
Escriba el nombre o IP del servidor:
localhost
MENU:

1. Registrar usuario

2. Iniciar sesi n

3. Salir

Seleccione una opci n:
1
Ingrese nombre de usuario:
juan
Ingrese contrase a:
123
Usuario registrado correctamente.

MENU:

1. Registrar usuario
```

```
2. Iniciar sesi n

3. Salir

Seleccione una opci n:
1
Ingrese nombre de usuario:
jorge
Ingrese contrase a:
123
Usuario registrado correctamente.

MENU:

1. Registrar usuario

2. Iniciar sesi n

3. Salir

Seleccione una opci n:
1
Ingrese nombre de usuario:
miguel
Ingrese contrase a:
123
Usuario registrado correctamente.

MENU:

1. Registrar usuario

2. Iniciar sesi n

3. Salir

Seleccione una opci n:
2
Ingrese nombre de usuario:
juan
Ingrese contrase a:
123
Inicio de sesi n exitoso para el usuario juan en su servidor ServerEUW

MENU:

1. Donar

2. Consultar total donado al sistema

3. Listado de usuarios en sistema

4. Comprobar las donaciones realizadas por  m

5. Comprobar el podio de donantes

6. Obtener listado de donantes en el sistema

7. Cerrar sesi n

Seleccione una opci n:
1
Ingrese la cantidad a donar:
40
Donaci n realizada con  xito . juan don  a su servidor ServerEUW: 40.0 euros.

MENU:

1. Donar
```

```
2. Consultar total donado al sistema

3. Listado de usuarios en sistema

4. Comprobar las donaciones realizadas por m

5. Comprobar el podio de donantes

6. Obtener listado de donantes en el sistema

7. Cerrar sesi n

Seleccione una opci n:
2
El total donado al sistema asciende a 40.0 euros.

En concreto en su servidor ServerEUW hay un subtotal de 40.0

MENU:

1. Donar

2. Consultar total donado al sistema

3. Listado de usuarios en sistema

4. Comprobar las donaciones realizadas por m

5. Comprobar el podio de donantes

6. Obtener listado de donantes en el sistema

7. Cerrar sesi n

Seleccione una opci n:
3
El listado de usuarios en su servidor "ServerEUW" es el siguiente:

 - juan
 - miguel
El listado de usuarios en el sistema completo es el siguiente:

 - jorge
 - juan
 - miguel
MENU:

1. Donar

2. Consultar total donado al sistema

3. Listado de usuarios en sistema

4. Comprobar las donaciones realizadas por m

5. Comprobar el podio de donantes

6. Obtener listado de donantes en el sistema

7. Cerrar sesi n

Seleccione una opci n:
6
A continaci n se listan los donantes del sistema completo

 - Donante 0: juan

MENU:
```

```
1. Donar

2. Consultar total donado al sistema

3. Listado de usuarios en sistema

4. Comprobar las donaciones realizadas por m

5. Comprobar el podio de donantes

6. Obtener listado de donantes en el sistema

7. Cerrar sesi n

Seleccione una opci n:
7
Cierre de sesi n exitoso para el usuario juan

MENU:

1. Registrar usuario

2. Iniciar sesi n

3. Salir

Seleccione una opci n:
2
Ingrese nombre de usuario:
jorge
Ingrese contrase a:
123
Inicio de sesi n exitoso para el usuario jorge en su servidor ServerNA

MENU:

1. Donar

2. Consultar total donado al sistema

3. Listado de usuarios en sistema

4. Comprobar las donaciones realizadas por m

5. Comprobar el podio de donantes

6. Obtener listado de donantes en el sistema

7. Cerrar sesi n

Seleccione una opci n:
1
Ingrese la cantidad a donar:
100
Donaci n realizada con  xito . jorge don  a su servidor ServerNA: 100.0 euros.

MENU:

1. Donar

2. Consultar total donado al sistema

3. Listado de usuarios en sistema

4. Comprobar las donaciones realizadas por m

5. Comprobar el podio de donantes
```

```
6. Obtener listado de donantes en el sistema

7. Cerrar sesi n

Seleccione una opci n:
7
Cierre de sesi n exitoso para el usuario jorge

MENU:

1. Registrar usuario

2. Iniciar sesi n

3. Salir

Seleccione una opci n:
2
Ingrese nombre de usuario:
miguel
Ingrese contrase a:
123
Inicio de sesi n exitoso para el usuario miguel en su servidor ServerEUW

MENU:

1. Donar

2. Consultar total donado al sistema

3. Listado de usuarios en sistema

4. Comprobar las donaciones realizadas por  m

5. Comprobar el podio de donantes

6. Obtener listado de donantes en el sistema

7. Cerrar sesi n

Seleccione una opci n:
1
Ingrese la cantidad a donar:
10
Donaci n realizada con  xito . miguel don  a su servidor ServerEUW: 10.0 euros.

MENU:

1. Donar

2. Consultar total donado al sistema

3. Listado de usuarios en sistema

4. Comprobar las donaciones realizadas por  m

5. Comprobar el podio de donantes

6. Obtener listado de donantes en el sistema

7. Cerrar sesi n

Seleccione una opci n:
5
El podio del servidor en el que usted est  registrado "ServerEUW" es el siguiente :

 - juan: 40.0

 - miguel: 10.0
```

```
El podio del sistema general es el siguiente :

- jorge: 100.0

 - juan: 40.0

 - miguel: 10.0

MENU:

1. Donar

2. Consultar total donado al sistema

3. Listado de usuarios en sistema

4. Comprobar las donaciones realizadas por  m

5. Comprobar el podio de donantes

6. Obtener listado de donantes en el sistema

7. Cerrar sesi n

Seleccione una opci n:
4
El n mero de donaciones que ha realizado es : 1

El total donado por usted asciende a : 10.0

MENU:

1. Donar

2. Consultar total donado al sistema

3. Listado de usuarios en sistema

4. Comprobar las donaciones realizadas por  m

5. Comprobar el podio de donantes

6. Obtener listado de donantes en el sistema

7. Cerrar sesi n

Seleccione una opci n:
7
Cierre de sesi n exitoso para el usuario miguel

MENU:

1. Registrar usuario

2. Iniciar sesi n

3. Salir

Seleccione una opci n:
3
Saliendo del sistema.
```

Listing 17: Ejemplo de ejecución del programa