

GRADO EN INGENIERÍA INFORMÁTICA
DESARROLLO DE SISTEMAS DISTRIBUIDOS



**UNIVERSIDAD
DE GRANADA**

P3 : RMI

David Serrano Domínguez

7 de mayo de 2024

Índice

1	Parte 1	III
1.1	Ejemplo 1	III
1.2	Ejemplo 2	IV
1.3	Ejemplo 3	VI
2	Ejercicio de Servidor replicado para donaciones	VII
2.1	Interfaz para Servidor/Cliente	VII
2.2	Interfaz para Servidor/Servidor	VII
2.3	Implementación de ambas interfaces en ServidorCliente	VIII
2.4	Programa Cliente	XI
3	Ejemplo de uso con un cliente que ha donado y otro que no	XIII

1. Parte 1

En esta primera parte vamos a tener que ejecutar diferentes ejemplos que se nos han proporcionado y responder a distintas preguntas tras ejecutar dichos ejemplos. Dichas preguntas son: ¿Que ocurre con las hebras cuyo nombre acaba en 0? ¿Que hacen las demás hebras? ¿Se entrelazan los mensajes?

1.1. Ejemplo 1

En el caso de este primer ejemplo el código del servidor esta implementado de manera que si el número proceso/hebra es cero entonces el servidor hace un sleep de cinco y segundos y después imprime por la consola Hebra más el número de la misma, sin embargo en caso de que el nombre de la hebra sea distinto de cero entonces no duerme cinco segundos y muestra directamente el nombre de la hebra que ha hecho la petición.

Aquí una prueba de ejecución de dicho ejemplo:

```
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
```

Figura 1: Ejecución Ejemplo 1.

1.2. Ejemplo 2

En este segundo ejemplo vamos a tener algo parecido, solo que esta vez tenemos varias hebras y como la hebra de nombre cero va a dormir cinco segundos durante ese tiempo van a ejecutarse el resto de hebras y por tanto la ultima hebra cliente que saldrá va a ser la número cero. El resto de hebras lo que haran será entrar se imprime que han entrado y seguidamente imprimen que han salido, ya que no van a tener un sleep.

Ahora los mensajes no se van a entrelazar, ya que las hebras distintas a la cero mostraran su mensaje de entrada y salida seguido y la hebra número cero mostrara su mensaje de entrada y el de salida siempre será el último. Ahora si usamos synchronized lo que va a suceder es que las hebras se van a ejecutar en orden de manera que si una hebra cliente ha entrada hasta que no salga esta no se va a ejecutar la siguiente, de manera que si entra la dos hasta que no salga la dos no pueden entrar la hebra cero o uno, en esta caso los mensajes tampoco se entrelazan y siguen un orden de manera que:

- Entra hebra 2
- Sale hebra 2
- Entra hebra 0
- Sale hebra 0
- Entra hebra 1
- Sale hebra 1

Aquí una prueba de ejecución de dicho ejemplo sin synchronized:

```
Lanzando el servidor
Ejemplo bound

Lanzando el primer cliente

Lanzando el segundo cliente

Buscando el objeto remoto
Buscando el objeto remoto
Buscando el objeto remoto
Invocando el objeto remoto
Invocando el objeto remoto

Entra Hebra Cliente 0

Entra Hebra Cliente 2
Empezamos a dormir
Sale Hebra Cliente 2
Invocando el objeto remoto

Entra Hebra Cliente 1
Sale Hebra Cliente 1
Terminamos de dormir
Sale Hebra Cliente 0
```

Figura 2: Ejecución Ejemplo 2 sin sincronización.

Aquí una prueba de ejecución de dicho ejemplo con synchronized:

```
Lanzando el servidor
Ejemplo bound

Lanzando el primer cliente

Lanzando el segundo cliente

Buscando el objeto remoto
Buscando el objeto remoto
Buscando el objeto remoto
Invocando el objeto remoto
Invocando el objeto remoto

Entra Hebra Cliente 1
Sale Hebra Cliente 1

Entra Hebra Cliente 0
Empezamos a dormir
Invocando el objeto remoto
Terminamos de dormir
Sale Hebra Cliente 0

Entra Hebra Cliente 2
Sale Hebra Cliente 2
```

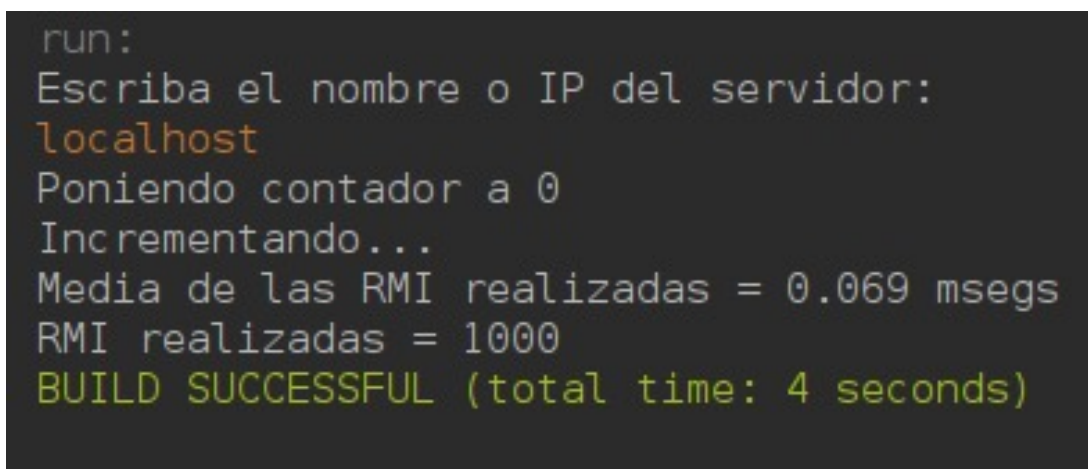
Figura 3: Ejecución Ejemplo 2 con sincronización.

1.3. Ejemplo 3

En este tercer y último ejemplo vamos a ver explicar que realiza dicho programa. Primero tenemos un objeto remoto de la clase contador que implementa la interfaz `icontador` la cual tiene principalmente tres métodos que son `sumar` el cual devuelve el valor del atributo privado `suma`, luego tenemos `sumar` que asigna a `suma` el valor del parámetro pasado a dicha función y por último `incrementar`, que lo que hace es incrementar el valor del atributo `suma` en uno.

Por tanto cuando ejecutamos el cliente este llama al método `incrementar` del objeto remoto mil veces y cuando termina muestra el número de llamadas del objeto remoto que se ha hecho en dicho por `msg` y posteriormente la cantidad de llamadas al objeto remoto que se han realizado usando el método `sumar` de dicho objeto.

Aquí una prueba de ejecución de dicho ejemplo:



```
run:
Escriba el nombre o IP del servidor:
localhost
Poniendo contador a 0
Incrementando...
Media de las RMI realizadas = 0.069 msecs
RMI realizadas = 1000
BUILD SUCCESSFUL (total time: 4 seconds)
```

Figura 4: Ejecución Ejemplo 3.

2. Ejercicio de Servidor replicado para donaciones

2.1. Interfaz para Servidor/Cliente

Primero vamos a ver como funciona la interfaz para comunicar al servidor con el cliente para que este pueda acceder a las siguientes operaciones:

```

1 import java.rmi.Remote;
2 import java.rmi.RemoteException;
3 import java.util.ArrayList;
4
5 public interface ServidorCliente_I extends Remote{
6     boolean registrar(String user, String password) throws RemoteException;
7     boolean iniciarSesion(String user, String password) throws RemoteException;
8     boolean donar(String user, double cantidad) throws RemoteException;
9     String getMiServidor(String user) throws RemoteException;
10    ArrayList<String> listaDonantes(String user) throws RemoteException;
11    ArrayList<String> listaDonantesTotal(String user) throws RemoteException;
12    double totalDonadoUsuario(String user) throws RemoteException;
13    double totalDonado(String user) throws RemoteException;
14    double subtotal(String user) throws RemoteException;
15    int getUsuarios() throws RemoteException;
16    int getTotalUsuarios() throws RemoteException;
17 }

```

Listing 1: Interfaz Servidor/Cliente.

Como vemos tenemos las operaciones obligatorias que se nos exigían que eran: Registrar(), Donar(), TotalDonado() y ListaDonantes(). He decidido añadir también operaciones extra:

- iniciarSesion()**: Devuelve true si el usuario está registrado.
- listaDonantes()**: Devuelve la lista donantes del servidor del usuario.
- getUsuarios()**: Devuelve los registrados del servidor del usuario.
- getMiServidor()**: Devuelve el nombre del objeto remoto del servidor en el que está registrado el usuario.
- getUsuariosTotal()**: Devuelve todos los registrados en el sistema.
- subtotal()**: Devuelve la cantidad donada en el servidor del usuario.
- totalDonadoUsuario()**: Devuelve la cantidad total de dinero donada por el usuario.

2.2. Interfaz para Servidor/Servidor

Ahora vamos a ver las operaciones que incluye la interfaz Servidor/Servidor para que se puedan comunicar ambos servidores.

```

1 import java.rmi.Remote;
2 import java.rmi.RemoteException;
3 import java.util.ArrayList;
4
5 public interface ServidorServidor_I extends Remote{
6     boolean existeUsuario(String User) throws RemoteException;
7     boolean confirmarSesion(String user, String password) throws RemoteException;
8     boolean donarReplica(String user, double cantidad) throws RemoteException;
9     String identificarUsuario(String user) throws RemoteException;
10    int getUsuariosReplica() throws RemoteException;
11    double getSubtotal() throws RemoteException;
12    ServidorServidor_I getReplica(String host, String nombre) throws RemoteException;

```

```
13 ArrayList<String> getDonantesReplica(String user) throws RemoteException;  
14 void asignarServidor(String user, String password) throws RemoteException;  
15 }
```

Listing 2: Interfaz Servidor/Servidor.

Tenemos distintos tipos de operaciones como son:

- existeUsuario()**: Devuelve true si ya existe ese usuario y false en caso contrario.
- confirmarSesion()**: Comprueba si el usuario ha llegado al máximo de intentos al iniciar sesión que son 3 y en dicho caso no le deja iniciar sesión.
- donarReplica()**: Manda una donación a la replica en caso de que un cliente se conecte al servidor que no esta registrado y en dicho caso manda la donación a la replica a la que pertenece.
- identificarCliente()**: Devuelve el nombre del servidor en el que esta registrado el usuario.
- getUsuariosReplica()**: Devuelve la cantidad de usuarios registrados en la replica.
- getSubtotal()**: Devuelve el total de donaciones hechas en el servidor.
- getReplica()**: Obtiene el objeto remoto para que el servidor pueda obtener el objeto remoto de la replica.
- getDonantesReplica()**: Devuelve la lista de usuarios que han donado en la replica.
- asignarServidor()**: Registra al usuario en la replica()

2.3. Implementación de ambas interfaces en ServidorCliente

Primero vamos a ver que tipo de datos hemos usado para almacenar los usuarios con sus respectivas contraseñas y para almacenar la cantidad donada por un usuario, ver si ha donado alguna vez, etc...

```
1 private static int INTENTOS = 3;  
2 private static String HOST = "localhost";  
3  
4 private Map<String,String> usuariosRegistrados;  
5 private Map<String,Double> donacionesPorUsuario;  
6 private Map<String,Boolean> usuarioDonado;  
7 private Map<String,Integer> intentos;  
8 private double subtotal;  
9 private String replica,nombre;
```

Listing 3: Atributos de ServidorCliente.

Tenemos tres variables de clase que son intentos que representa la cantidad máxima de intentos al iniciar sesión, luego N que representa cuantas replicas tenemos y HOST con el nombre del host.

Ahora vamos a ver como representamos los datos de los usuarios. Pare ello principalmente tenemos 4 map los cuales tienen todos como clave el usuario ya que este es único en todo el sistema.

Ahora el primer map contiene para cada usuario su contraseña, el segundo para cada usuario la cantidad que ha donado, el tercero un booleano que nos sirve para comprobar si cada usuario ha donado alguna vez o no y por último el número de intentos de inicio sesión de cada usuario para bloquearle dicho inicio o no.

Luego tenemos subtotal que contiene la cantidad donada hasta ahora en el servidor y por último dos string uno con el nombre de la replica y otro con el mio.

Ahora el método registrar único que hacen es añadir el usuario al map con su contraseña si no existe y si tiene menos usuarios que la replica ya que en dicho caso se mandara el usuario a la replica para que lo registre en su map dicha replica.


```

1  @Override
2  public boolean registrar(String user, String password) throws RemoteException {
3      ServidorServidor_I replica = this.getReplica(ServidorCliente.HOST, this.replica);
4      if (!this.usuariosRegistrados.containsKey(user) && !replica.existeUsuario(user)) {
5          if (this.getUsuarios() <= replica.getUsuariosReplica()) {
6              this.asignarServidor(user, password);
7          } else {
8              replica.asignarServidor(user, password);
9          }
10         return true; // Se ha registrado correctamente el usuario
11     } else {
12         return false; // Ya existe un usuario registrado con ese nombre
13     }
14 }

```

Listing 4: Registrar.

Luego el método iniciar sesión comprueba en el map si existe dicho usuario y en caso de que no, comprueba que exista en la replica y en caso de que existe en alguna de las dos devuelve true y en caso de que no o de que la contraseña sea incorrecta devuelve false y en este ultimo caso incrementa el número de intentos del usuario actual.

```

1  public boolean iniciarSesion(String user, String password) throws RemoteException {
2      if(this.usuariosRegistrados.containsKey(user)){
3          String possiblePassword = this.usuariosRegistrados.get(user);
4          if(intentos.get(user)<ServidorCliente.INTENTOS){
5              if(password.equals(possiblePassword)){
6                  return true;
7              }else{
8                  intentos.put(user, intentos.get(user)+1);
9                  return false;
10             }
11         }else{
12             return false;
13         }
14     }else if (this.getReplica(ServidorCliente.HOST, this.replica).existeUsuario(user)) {
15         return this.getReplica(ServidorCliente.HOST, this.replica).confirmarSesion(user,
16         password);
17     }else{
18         return false;
19     }
20 }

```

Listing 5: Iniciar Sesión.

Ahora el método donar simplemente añade la donación al servidor actual y suma la cantidad al map de total donado por cada usuario.

Si el usuario no existe en el servidor actual se hace la donación en la replica ya que entonces debe pertenecer a esta.

```

1  @Override
2  public boolean donar(String user, double cantidad) throws RemoteException {
3      if(this.existeUsuario(user){
4          double actual = this.donacionesPorUsuario.get(user);
5          this.donacionesPorUsuario.put(user, actual+cantidad);
6          this.usuarioDonado.put(user, true);
7          this.subtotal+=cantidad;
8          return true;
9      }else if(this.getReplica(ServidorCliente.HOST, this.replica).existeUsuario(user)){
10         double actual = this.donacionesPorUsuario.get(user);
11         return this.getReplica(ServidorCliente.HOST, this.replica).donarReplica(user,
12         actual+cantidad);
13     }else{
14         return false; //No esta registrado en el sistema el usuario y no puede donar
15     }
16 }

```

Listing 6: Donar.

El siguiente método es el de la lista total de donantes. Lo primero que comprobamos es si tiene alguna donación el usuario ya que en caso de que no, no se le permite mostrar la lista de donantes a usuario que no hayan donado nunca, en dicho caso se le devuelve una lista de donantes vacía y en caso contrario una lista con los donantes del servidor.

```

1  @Override
2  public ArrayList<String> listaDonantes(String user) throws RemoteException {
3      ArrayList<String> donantes = new ArrayList<>();
4      if(this.usuarioDonado.get(user)){
5          for(Map.Entry<String,Double> entry : donacionesPorUsuario.entrySet()){
6              String usuario = entry.getKey();
7              Double donaciones = entry.getValue();
8              if(donaciones>0){
9                  donantes.add(usuario);
10             }
11         }
12         return donantes;
13     }else{
14         return donantes;
15     }

```

Listing 7: Lista donantes.

La lista de donantes total sería parecido pero sumándole la lista de donantes de la réplica y devolviéndosela al cliente.

Por último se va a explicar cómo funciona el último método obligatorio que será el de total de dinero donado a sistema el cual devolverá al usuario la cantidad total donada y al igual que lista de donantes lo hará cuando el usuario haya donado mínimo una vez.

```

1  @Override
2  public double totalDonado(String user) throws RemoteException {
3      if(this.usuarioDonado.get(user)){
4          ServidorServidor_I replica = this.getReplica(ServidorCliente.HOST, this.replica);
5          return this.subtotal + replica.getSubtotal();
6      }
7      return -1;
8  }

```

Listing 8: Total donado.

El método getReplica del apartado anterior es el encargado de devolver el objeto remoto de la interfaz Servidor/Servidor y el cual funciona de la siguiente manera:

```

1  @Override
2  public ServidorServidor_I getReplica(String host, String nombre) throws RemoteException {
3      ServidorServidor_I replica = null;
4
5      try {
6          Registry mireg = LocateRegistry.getRegistry(host, 1099);
7          replica = (ServidorServidor_I)mireg.lookup(nombre);
8      } catch (NotBoundException | RemoteException e) {
9          System.err.println("Exception del sistema: " + e);
10     }
11     return replica;
12 }

```

Listing 9: Get replica.

Faltan varios métodos por explicar pero los cuales su implementación es bastante sencilla e intuitiva por lo mismo no se considera necesario explicarlos.

2.4. Programa Cliente

El programa cliente tendrá principalmente dos menús con los que podrá interactuar dicho cliente, el primero para antes de iniciar sesión y otro para cuando haya iniciado dicha sesión.

```

1 public static int menuSesion() {
2     Scanner scannerSesion = new Scanner(System.in);
3     int opcion=-1;
4     while (opcion<0 || opcion>3) {
5         System.out.print("\nSelecciona una de las siguientes operaciones:\n"+
6             "1.Registrarse\n"+
7             "2.Iniciar Sesion\n"+
8             "3.Salir\n");
9         opcion = scannerSesion.nextInt();
10    }
11    return opcion;
12 }

```

Listing 10: Menú antes de iniciar sesión.

```

1 public static int menuSesionIniciada() {
2     Scanner scannerSesion = new Scanner(System.in);
3     int opcion=-1;
4     while (opcion<0 || opcion>9) {
5         System.out.print("\nSelecciona una de las siguientes operaciones:\n"+
6             "\n1.Donar\n"+
7             "\n2.Obtener Donantes de mi servidor\n"+
8             "\n3.Obtener Donantes totales\n"+
9             "\n4.Obtener cantidad de registrados en mi servidor\n"+
10            "\n5.Obtener cantidad total de registrados\n"+
11            "\n6.Obtener cantidad donada en de mi servidor\n"+
12            "\n7.Obtener cantidad total de donaciones\n"+
13            "\n8.Obtener cantidad total donada por mi\n"+
14            "\n9.Salir\n");
15        opcion = scannerSesion.nextInt();
16    }
17    return opcion;
18 }

```

Listing 11: Menú sesión iniciada.

Ambos menús están metidos en dos bucles anidados while en el que cada uno controla un menú y se sale de dicho bucle cuando el cliente lo solicita mediante la opción del menú. Como ejemplo de código se muestra a continuación como se gestionaría el registro.

```

1 String nombre = "servidor";
2 Registry registry = LocateRegistry.getRegistry(args[0], 1099);
3 ServidorCliente_I stub = (ServidorCliente_I) registry.lookup(nombre);
4 while (true) {
5     opcion = menuSesion();
6     switch (opcion) {
7         case 1:
8             valor=false;
9             while (!valor) {
10                System.out.print("\nIntroduzca el nombre de usuario: ");
11                user=scanner.next();
12                char[] passwordArray = console.readPassword("Introduzca la
13                contrasenia: ");
14                password=new String(passwordArray);
15                valor=stub.registrar(user, password);
16                if (!valor) {
17                    System.out.print("\n\tYa existe un usuario con ese nombre,
18                    porfavor introduzca otro");
19                    break;
20                }else{
21                    System.out.print("\n\tRegistrado correctamente\n");
22                }

```

```

21         }
22         break;

```

Listing 12: Opción registro del menú.

Por defecto nos conectamos al servidor principal y usamos dicho stub para registrarnos, pero una vez registrados mediante la función de `getMiServidor()` obtenemos el nombre del objeto remoto del servidor al que pertenece e interactúa directamente con dicho servidor mediante su objeto remoto, como se muestra a continuación al iniciar sesión.

```

1 System.out.print("\nIntroduzca el nombre de usuario: ");
2 user=scanner.next();
3 char[] passwordArray = console.readPassword("\nIntroduzca la contraseña: ");
4 password=new String(passwordArray);
5 valor=stub.iniciarSesion(user, password);
6 if (!valor) {
7     System.out.print("\n\tError al iniciar sesion\n");
8 }else{
9     System.out.println("\n\tSesion Iniciada\n");
10    ServidorCliente_I stub2 = (ServidorCliente_I) registry.lookup(stub.getMiServidor(user)
11    );
12    stub = stub2;
13    System.out.println("\n\t-Mi servidor es: "+stub2.getMiServidor(user)+"\n");

```

Listing 13: Opción registro del menú.

Por último lo que vamos a ver en el cliente es como puede donar y si no se ha donado no se puede ver la lista de donantes.

```

1 case 1:
2     System.out.print("\nIntroduzca la cantidad que desea donar: \n");
3     cantidad=scanner.nextDouble();
4     if(stub.donar(user,cantidad) && cantidad>0.0){
5         System.out.print("\n\t-Ha donado correctamente\n");
6     }else{
7         System.out.print("\n\t-No ha podido donar\n");
8     }
9     break;

```

Listing 14: Opción donar del menu inicio sesión.

```

1 case 3:
2     ArrayList<String> donantesTotales = stub.listaDonantesTotal(user);
3     System.out.print("\nDonantes totales:\n ");
4     if(donantesTotales.size()>0){
5         for(int i=0;i<donantesTotales.size();i++){
6             System.out.print("\n\t-Donante numero "+i+": "+donantesTotales.get(i)+"\n");
7         }
8     }else{
9         System.out.print("\n\t-No puede ver los donantes hasta que no haya donado\n");
10    }
11    break;

```

Listing 15: Opción obtener donantes totales del menu inicio sesión.

Como vemos si la lista donantes es 0 significa que se ha devuelto una lista vacía ya que el usuario no ha donado aún y por tanto no va a poder ver la lista total de donantes hasta que haya donado mínimo 1 vez.

3. Ejemplo de uso con un cliente que ha donado y otro que no

Vamos a mostrar un ejemplo con un cliente que ha donado y con otro que no. Primero vamos a registrar a ambos clientes.

Primero registro a dos usuarios llamados David y Pepe. Posteriormente inicio sesión con David hago que done mil y veo la lista de donantes. Luego me salgo e inicio sesión con Pepe y hago que intente ver la lista de donantes y no le dejara, por tanto hago que done cien y vuelva a intentarlo y ya si le deja ver todos los donantes que son tanto David como Pepe y por último veo el total donado que será mil cien.

```
Selecciona una de las siguientes operaciones:
1.Registrarse
2.Iniciar Sesion
3.Salir
1

Introduzca el nombre de usuario: David
Introduzca la contraseña:

Registrado correctamente
```

Figura 5: Registro David.

```
Selecciona una de las siguientes operaciones:
1.Registrarse
2.Iniciar Sesion
3.Salir
1

Introduzca el nombre de usuario: Pepe
Introduzca la contraseña:

Registrado correctamente
```

Figura 6: Registro Pepe.

Luego inicio sesión con David y hago que done 1000.

```
Introduzca el nombre de usuario: David
Introduzca la contraseña:
    ¡¡¡Sesion Iniciada!!!

    -Mi servidor es: servidor

Selecciona una de las siguientes operaciones:
1.Donar
2.Obtener Donantes de mi servidor
3.Obtener Donantes totales
4.Obtener cantidad de registrados en mi servidor
5.Obtener cantidad total de registrados
6.Obtener cantidad donada en de mi servidor
7.Obtener cantidad total de donaciones
8.Obtener cantidad total donada por mi
9.Salir
```

Figura 7: Inicio Sesión David.

```
Introduzca la cantidad que desea donar:
1000

-Ha donado correctamente
```

Figura 8: Donación de David.

```
Donantes totales:

-Donante número 0: David
```

Figura 9: Lista donantes totales para David.

```
!!!Sesion Iniciada!!!

-Mi servidor es: replica

Selecciona una de las siguientes operaciones:
1.Donar
2.Obtener Donantes de mi servidor
3.Obtener Donantes totales
4.Obtener cantidad de registrados en mi servidor
5.Obtener cantidad total de registrados
6.Obtener cantidad donada en de mi servidor
7.Obtener cantidad total de donaciones
8.Obtener cantidad total donada por mi
9.Salir
█
```

Figura 10: Inicio sesión Pepe.

```
Donantes totales:  
-No puede ver los donantes hasta que no haya donado
```

Figura 11: Lista donantes totales para Pepe.

```
Introduzca la cantidad que desea donar:  
100  
  
-Ha donado correctamente
```

Figura 12: Donación de Pepe.

```
Donantes totales:  
  
-Donante número 0: Pepe  
  
-Donante número 1: David
```

Figura 13: Lista donantes totales para Pepe.

```
-El total donado es: 1100.0
```

Figura 14: Total Donado.