Práctica 3: RMI

Índice

Introducción

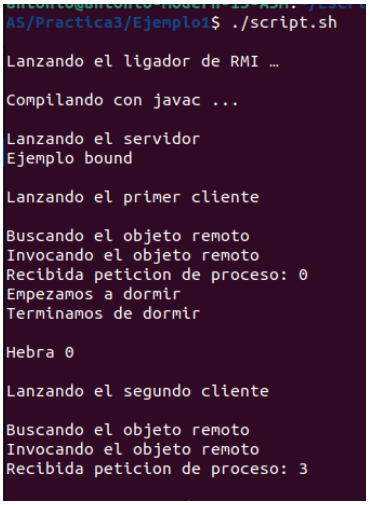
Esta práctica ha consistido en 2 partes:

1º → Ha consistido en ejecutar los códigos de ejemplo para ver su funcionamiento y después responder algunas preguntas sobre ellos.

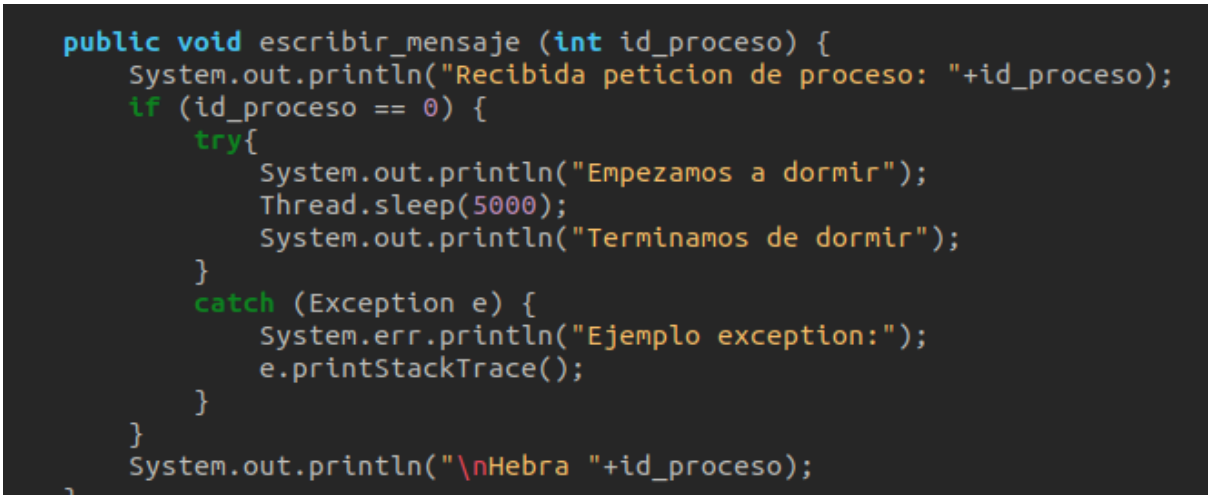
2º → Ha consistido en crear un servidor de donaciones con una réplica donde el cliente se conecta a cualquiera pero se registra y dona en la réplica o el servidor según algunas restricciones.

Parte 1

El ejemplo 1:



Lanzamos el script, creamos un Ejemplo_I como instancia_local y vemos que con esta instancia local usamos escribir_mensaje(argumento introducido). En la primera ejecución introducimos el número 0.



y como vemos en el código en escribir_mensaje si es 0 duerme pero si es otro número no pasa nada, que es lo que hace en la segunda ejecución del script.

Ejemplo 2:

```
Entra Hebra Cliente 0

Entra Hebra Cliente 6

Entra Hebra Cliente 8

Entra Hebra Cliente 1

Entra Hebra Cliente 2

Entra Hebra Cliente 9
Empezamos a dormir
Sale Hebra Cliente 6
Sale Hebra Cliente 1
Sale Hebra Cliente 8

Entra Hebra Cliente 3
Sale Hebra Cliente 3
Sale Hebra Cliente 2
Sale Hebra Cliente 9

Entra Hebra Cliente 5

Entra Hebra Cliente 4
Sale Hebra Cliente 5
Sale Hebra Cliente 4

Entra Hebra Cliente 10
Empezamos a dormir

Entra Hebra Cliente 7
Sale Hebra Cliente 7
Terminamos de dormir
Sale Hebra Cliente 0
Terminamos de dormir
Sale Hebra Cliente 10

Lanzando el segundo cliente

Buscando el objeto remoto
Invocando el objeto remoto

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

```java
public void escribir_mensaje (String mensaje) {
    System.out.println("\nEntra Hebra "+mensaje);

    //Buscamos los procesos 0, 10, 20,...
    if (mensaje.endsWith("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("Sale Hebra "+mensaje);
}
```

Ahora en vez de ejecutar distintos clientes hemos ejecutado con diferentes números de hebras que es el número del argumento introducido en vez del idproceso, He ejecutado primero con 11 hebras para tener 2 hebras terminadas en 0 y la segunda ejecución con solo una hebra. vemos que ahora busca las hebras que terminan en 0 y las hace esperar mientras que las demás no tienen que esperar, así terminando todas las hebras sin 0 antes que las que terminan en 0. Ahora lo ejecutaremos con synchronized.

```
Invocando el objeto remoto

Entra Hebra Cliente 4
Sale Hebra Cliente 4Invocando


Entra Hebra Cliente 7
Sale Hebra Cliente 7

Entra Hebra Cliente 10
Empezamos a dormir
Terminamos de dormir
Sale Hebra Cliente 10

Entra Hebra Cliente 6
Sale Hebra Cliente 6

Entra Hebra Cliente 2
Sale Hebra Cliente 2

Entra Hebra Cliente 9
Sale Hebra Cliente 9

Entra Hebra Cliente 5
Sale Hebra Cliente 5

Entra Hebra Cliente 1
Sale Hebra Cliente 1

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

Entra Hebra Cliente 3
Sale Hebra Cliente 3

Entra Hebra Cliente 8
Sale Hebra Cliente 8
```

Ahora cuando entra una hebra el synchronized evita que entre otra hebra en escribir_mensaje simultáneamente haciendo que la siguiente hebra tenga que esperar hasta que termina la que está actualmente en el método. Por esto las hebras terminadas en 0 ahora no son las que terminan las últimas.

Preguntas Ejemplo 2:

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

Como hemos visto en la primera ejecución del ejemplo 2 las hebras cuyo nombre acaba en 0 al tener un sleep suelen acabar las últimas. Las demás hebras entran y salen sin tener que hacer nada por lo tanto si se entrelazan los mensajes. Será casi impredecible para nosotros saber cual entrará y saldrá primero exceptuando las hebras terminadas en 0.

Ejemplo 3:

```
AS/Practica3/Ejemplo3$ ./script.sh

Lanzando el ligador de RMI …

Compilando con javac ...

Lanzando el servidor
Servidor RemoteException | MalformedURLExceptiondor preparado

Lanzando el primer cliente

Poniendo contador a 0
Incrementando...
Media de las RMI realizadas = 0.075 msegs
RMI realizadas = 1000

Lanzando el segundo cliente

Poniendo contador a 0
Incrementando...
Media de las RMI realizadas = 0.073 msegs
RMI realizadas = 1000
```

Aquí tenemos la ejecución del ejemplo 3

```
public class Cliente{
    public static void main(String[] args){

        // Crea e instala el gestor de seguridad
        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new SecurityManager());
        }
        try {
            // Crea el stub para el cliente especificando el nombre del servidor
            Registry mireg = LocateRegistry.getRegistry("127.0.0.1", 1099);

            icontador micontador = (icontador)mireg.lookup("mmicontador");

            // Pone el contador al valor inicial 0
            System.out.println("Poniendo contador a 0");
            micontador.sumar(0);

            // Obtiene hora de comienzo
            long horacomienzo = System.currentTimeMillis();

            // Incrementa 1000 veces
            System.out.println("Incrementando...");
            for (int i = 0; i < 1000; i++ ) {
                micontador.incrementar();
            }

            // Obtiene hora final, realiza e imprime calculos
            long horafin = System.currentTimeMillis();
            System.out.println("Media de las RMI realizadas = " + ((horafin - horacomienzo)/1000f) + "
msegs");
            System.out.println("RMI realizadas = " + micontador.sumar());
        } catch(NotBoundException | RemoteException e) {
            System.err.println("Exception del sistema: " + e);
        }
    System.exit(0);
    }
}
```

Busca el objeto remoto y usa la función sumar para poner el contador a 0 y luego usa incrementar 1000 veces, obtiene el tiempo transcurrido y calcula la media y el total de las operaciones que realiza. En ambas ejecuciones son el

mismo número de operaciones y la media de tiempo también es muy parecida.

Parte 2

Cliente.java:

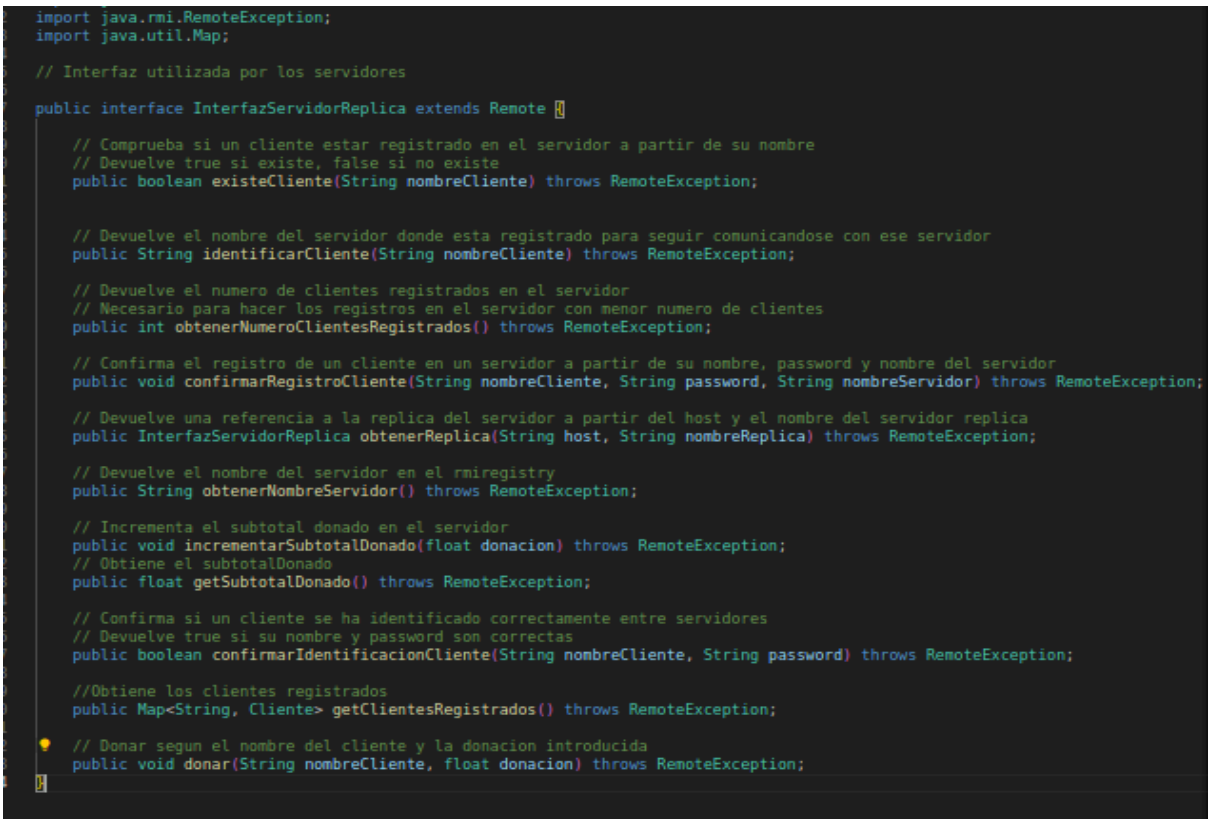
```java
import java.util.ArrayList;
import java.io.Serializable;

// Clase utilizada para representar a los clientes
public class Cliente implements Serializable{
    private String nombre;
    private String password;
    private ArrayList<Float> donaciones;
    private float donacionTotal;
    private int numeroDonaciones;
    private float donacionMaxima;

    Cliente(String nombre, String password)
    {
        this.nombre = nombre;
        this.password = password;
        this.donaciones = new ArrayList<>();
        this.donacionTotal = 0.0f;
        this.numeroDonaciones = 0;
        this.donacionMaxima = 0.0f;
    }

    public String obtenerNombre()
    {
        return nombre;
    }

    public String obtenerPassword()
    {
        return password;
    }

    public ArrayList<Float> obtenerHistorialDonaciones()
    {
        return donaciones;
    }

    public float obtenerUltimaDonacion()
    {
        return donaciones.get(donaciones.size() - 1);
    }

    public float obtenerDonacionTotal()
    {
        return donacionTotal;
    }

    public int obtenerNumeroDonaciones()
    {
        return numeroDonaciones;
    }

    public float obtenerDonacionMaxima()
    {
        return donacionMaxima;
    }

    public void cambiarNombre(String nombre)
    {
        this.nombre = nombre;
    }

    public void cambiarPassword(String password)
    {
        this.password = password;
    }

    public void donar(float donacion)
    {
        this.donaciones.add(donacion);
        this.donacionTotal += donacion;

        if(donacion > this.donacionMaxima)
            this.donacionMaxima = donacion;

            this.incrementarNumeroDonaciones();
    }

    public void incrementarNumeroDonaciones()
    {
        this.numeroDonaciones++;
    }
}
```

Es una clase para crear una instancia cliente que se guardará en el servidor y réplica, para así tener todos los datos de los clientes mejor estructurados. De aquí no hay mucho que comentar, tienen los get/obtener de todos los atributos e incluso algunos set como nombre y contraseña ya que tenía planeado poder cambiar tu nombre e incluso tu contraseña pero me quedé sin tiempo. La función más importante es la de donar donde se aumenta el número de donaciones, el total donado y si es el nuevo récord donado se cambia también.

InterfazServidorCliente.java

```java
import java.rmi.RemoteException;

import java.util.ArrayList;

// Interfaz utilizada por los servidores de cara al cliente

public interface InterfazServidorCliente extends Remote {

    // Comprueba si un cliente esta registrado
    public boolean registrado(String nombreCliente) throws RemoteException;

    // Realiza registro de un cliente a partir de su nombre y password
    // Devuelve true si se registra el cliente (no existe), false si no se registra (ya existe)
    public boolean realizarRegistro(String nombreCliente, String password) throws RemoteException;

    // Realiza una donacion de un cliente a partir de su nombre y la cantidad donada
    public void realizarDonacion(String nombreCliente, float donacion) throws RemoteException;

    // Comprueba que el nombre y la contraseña coincide devolviendo true para iniciar sesion
    // devuelve false si no concuerdan para para que no pueda iniciar sesion
    public Boolean inicioSesion(String nombreCliente, String password) throws RemoteException;

    // Obtiene la cantidad total donada en un servidor
    public float obtenerSubtotalDonado(String nombreCliente) throws RemoteException;

    // Obtiene la cantidad total donada a un servidor por un cliente
    public float obtenerDonacionCliente(String nombreCliente) throws RemoteException;

    // Obtiene el numero de donaciones de un cliente
    public int obtenerNumeroDonacionesCliente(String nombreCliente) throws RemoteException;

    // Obtiene la donacion maxima de un servidor hecha por un cliente
    public float obtenerDonacionMaximaCliente(String nombreCliente) throws RemoteException;

    // Obtiene el historial de donaciones de un cliente a un servidor
    public ArrayList<Float> obtenerHistorialDonaciones(String nombreCliente) throws RemoteException;

    //obtiene el total donado en ambos servidores
    public float obtenerTotalDonadoServidores() throws RemoteException;

    //obtiene la lista de donantes sin orden
    public ArrayList<Cliente> obtenerListeDonantes() throws RemoteException;

    //obtiene la lista de donantes ordenado de mayor a menorsegun el total de donaciones
    public ArrayList<Cliente> obtenerRanking() throws RemoteException;
}
```

La interfaz que usará el cliente, tiene algunas funciones como la de iniciar sesión, obtener las donaciones y realizar donaciones.



```java
import java.rmi.RemoteException;
import java.util.Map;

// Interfaz utilizada por los servidores

public interface InterfazServidorReplica extends Remote {

    // Comprueba si un cliente estar registrado en el servidor a partir de su nombre
    // Devuelve true si existe, false si no existe
    public boolean existeCliente(String nombreCliente) throws RemoteException;


    // Devuelve el nombre del servidor donde esta registrado para seguir comunicandose con ese servidor
    public String identificarCliente(String nombreCliente) throws RemoteException;

    // Devuelve el numero de clientes registrados en el servidor
    // Necesario para hacer los registros en el servidor con menor numero de clientes
    public int obtenerNumeroClientesRegistrados() throws RemoteException;

    // Confirma el registro de un cliente en un servidor a partir de su nombre, password y nombre del servidor
    public void confirmarRegistroCliente(String nombreCliente, String password, String nombreServidor) throws RemoteException;

    // Devuelve una referencia a la replica del servidor a partir del host y el nombre del servidor replica
    public InterfazServidorReplica obtenerReplica(String host, String nombreReplica) throws RemoteException;

    // Devuelve el nombre del servidor en el rmiregistry
    public String obtenerNombreServidor() throws RemoteException;

    // Incrementa el subtotal donado en el servidor
    public void incrementarSubtotalDonado(float donacion) throws RemoteException;
    // Obtiene el subtotalDonado
    public float getSubtotalDonado() throws RemoteException;

    // Confirma si un cliente se ha identificado correctamente entre servidores
    // Devuelve true si su nombre y password son correctas
    public boolean confirmarIdentificacionCliente(String nombreCliente, String password) throws RemoteException;

    //Obtiene los clientes registrados
    public Map<String, Cliente> getClientesRegistrados() throws RemoteException;

    // Donar segun el nombre del cliente y la donacion introducida
    public void donar(String nombreCliente, float donacion) throws RemoteException;
}
```

La interfaz que usarán los servidores, estas funciones solo la usan los servidores, Hay funciones para poder comunicarse entre réplicas y para tener las donaciones y saber si existe el usuario, etc. Explicaré su implementación a continuación.

DonacionesServidor.java:

Aquí tenemos la implementación de todas las funciones anteriormente comentadas.

```java
public class DonacionesServidor extends UnicastRemoteObject implements InterfazServidorCliente, InterfazServidorReplica {
    private String servidor;
    private String replica;
    private float subtotalDonado;

    private Map<String, Cliente> clientesRegistrados;

    // Constructor
    public DonacionesServidor(String servidor, String replica) throws RemoteException {
        this.servidor = servidor;
        this.replica = replica;
        this.subtotalDonado = 0.0f;
        clientesRegistrados = new HashMap<>();
    }

    // Comprueba si un cliente ya esta registrado o en la replica
    @Override
    public boolean registrado(String nombreCliente) throws RemoteException {
        boolean Registrado = false;

        InterfazServidorReplica servidorReplica = this.obtenerReplica(host:"localhost", this.replica);

        if(this.existeCliente(nombreCliente) || servidorReplica.existeCliente(nombreCliente)){
            Registrado = true;
        }

        return Registrado;

    }

    // Realiza el registro de un cliente a partir de su nombre y password
    // El registro se realiza en el servidor donde haya menos clientes registrados
    // Devuelve true si el cliente se registra en uno de los servidores, false si no
    // se registra
    @Override
    public boolean realizarRegistro(String nombreCliente, String password) throws RemoteException {
        // Se comprueba si existe en este servidor
        boolean registrado = this.existeCliente(nombreCliente);
        boolean resultado = false;

        if (!registrado) {
            // Se comprueba si existe en la replica
            InterfazServidorReplica servidorReplica = this.obtenerReplica(host:"localhost", this.replica);
            boolean registradoEnReplica = servidorReplica.existeCliente(nombreCliente);

            if (!registradoEnReplica) {
                int clientesEnServidor = this.obtenerNumeroClientesRegistrados();
                int clientesEnReplica = servidorReplica.obtenerNumeroClientesRegistrados();

                if (clientesEnServidor < clientesEnReplica)
                    this.confirmarRegistroCliente(nombreCliente, password, this.servidor);
                else
                    servidorReplica.confirmarRegistroCliente(nombreCliente, password, this.replica);

                resultado = true;
            }
        }

        return resultado;
    }
```

Tiene como atributos servidor(nombre del servidor), réplica(nombre de replica), subtotaldonado en el servidor y clientesRegistrados en ese servidor.

El constructor se pasa el nombre del servidor y de la réplica para cuando necesite buscarlo.
La función registrado comprueba si existe el cliente en este servidor tanto como en la réplica, si existe devuelve true.

La función realizarRegistro primero comprueba si está registrado en algunas de las replicas, si no esta en ninguna, comprueba cuantos clientes hay en cada réplica y usa confirmaResgistroCliente en la réplica con menor clientes.

Esta función sólo aumenta el clientesRegistrado con su nombre,contraseña sin comprobar nada ya que todo ha sido comprobado.
retorna true o false dependiendo si se ha podido registrar o no.

```java
// Realiza una donacion por parte de un cliente a partir de su nombre
@Override
public void realizarDonacion(String nombreCliente, float donacion) throws RemoteException {
    String nombreServidor = identificarCliente(nombreCliente);
    if(nombreServidor.equals(this.replica)){
        InterfazServidorReplica servidorReplica = this.obtenerReplica(host:"localhost", this.replica);
        servidorReplica.donar(nombreCliente, donacion);
    }else{
        this.donar(nombreCliente,donacion);
    }

}
// Inicia sesion
@Override
public Boolean inicioSesion(String nombreCliente, String password) throws RemoteException {
    String nombreServidor = identificarCliente(nombreCliente);
    boolean identificado = false;

    if (nombreServidor.equals(this.replica)) {
        InterfazServidorReplica servidorReplica = this.obtenerReplica(host:"localhost", this.replica);
        boolean registradoEnReplica = servidorReplica.existeCliente(nombreCliente);

        if(registradoEnReplica){
            if(servidorReplica.confirmarIdentificacionCliente(nombreCliente,password)){
                identificado = true;
            }
        }

    } else if(nombreServidor.equals(this.servidor)) {
        boolean registrado = this.existeCliente(nombreCliente);

        if(registrado){
            if(this.confirmarIdentificacionCliente(nombreCliente,password)){
                identificado = true;
            }
        }
    }

    return identificado;
}


// Devuelve el nombre del servidor en el que se ha registrado el cliente
@Override
public String identificarCliente(String nombreCliente) throws RemoteException {
    String nombreServidor = "";
    boolean registrado = this.existeCliente(nombreCliente);

    if (!registrado) {
        InterfazServidorReplica servidorReplica = this.obtenerReplica(host:"localhost", this.replica);
        boolean registradoEnReplica = servidorReplica.existeCliente(nombreCliente);

        if(registradoEnReplica){

            nombreServidor = this.replica;
        }

    } else if(registrado) {
            nombreServidor = this.servidor;
    }

    return nombreServidor;
}

// Comprueba si existe el cliente en clientesRegistrados
// Devuelve true si existe, false si no existe
@Override
public boolean existeCliente(String nombreCliente) throws RemoteException {
    return clientesRegistrados.containsKey(nombreCliente);
}

// Devuelve el numero de clientes registrados
@Override
public int obtenerNumeroClientesRegistrados() throws RemoteException {
    return clientesRegistrados.size();
}
```

La función realizarDonacion, primero usa la función identificarCliente que te devuelve el nombre donde está registrado el cliente, con ese nombre dependiendo de cuál es busca la réplica o usa this con la función donar() donde no hay comprobaciones, se obtiene el cliente y se usa la función donar de la clase cliente para aumentar la donaciones.

La función inicioSesion, buscar donde está registrado y usa la función confirmar para ver si su nombre y contraseña coincide.

La función identificar y existe ya la hemos comentado, identificar te devuelve el nombre del servidor donde el cliente está registrado y existe simplemente buscar en el mapa de clientesRegistrados si hay algún objeto con el nombre Cliente.

```java
// Comprueba que servidor esta el cliente registrado para buscar el subtotal del servidor
@Override
public float obtenerSubtotalDonado(String nombreCliente) throws RemoteException {
    String nombreServidor = identificarCliente(nombreCliente);
    float donado = 0.0f;

    if (nombreServidor.equals(this.replica)) {
        InterfazServidorReplica servidorReplica = this.obtenerReplica(host:"localhost", this.replica);
        donado = servidorReplica.getSubtotalDonado();

    } else if(nombreServidor.equals(this.servidor)) {
        donado = this.getSubtotalDonado();

    }

    return donado;
}

// Devuelve el subtotaldonado
@Override
public float getSubtotalDonado()throws RemoteException {
    return this.subtotalDonado;
}


// Introduce a un cliente en clientesRegistrados
@Override
public void confirmarRegistroCliente(String nombreCliente, String password, String nombreServidor) throws RemoteException {
    clientesRegistrados.put(nombreCliente, new Cliente(nombreCliente, password));
    System.out.println("Se acaba de registrar el cliente " + nombreCliente);
}

// Devuelve una referencia a la replica del servidor, es una referencia a null si no se encuentra la replica
@Override
public InterfazServidorReplica obtenerReplica(String host, String nombreReplica) throws RemoteException {
    InterfazServidorReplica servidorReplica = null;

    try {
        Registry registroRMI = LocateRegistry.getRegistry(host, 1099);
        servidorReplica = (InterfazServidorReplica) registroRMI.lookup(nombreReplica);
    } catch (NotBoundException | RemoteException e) {
        System.out.println("No se encuentra el servidor replica con el nombre " + nombreReplica);
    }

    return servidorReplica;
}

// Devuelve el nombre del servidor
@Override
public String obtenerNombreServidor() throws RemoteException {
    return this.servidor;
}
```

La función obtenerSubtotalDonado, comprueba donde está registrado el cliente y usa la función getsubtotaldonado() el cual simplemente devuelve el subtotaldonado de la réplica.

La función comfirmarRegistroCliente simplemente mete en el map el cliente ya que cuando se usa esta función ya se han hecho las comprobaciones correspondientes.

La función obtenerReplica te devuelve la conexión con la réplica, creé esta función ya que me quitaba mucha repetición de código. Si no la encuentra te lo muestra el error y además te devuelve null.

La función obtenerNombreServidor te devuelve el nombre del servidor.

```java
// Incrementa el subtotal donado al servidor en una cantidad
@Override
public void incrementarSubtotalDonado(float donacion) throws RemoteException {
    this.subtotalDonado += donacion;
}

// Comprueba si el nombre y password de un cliente son correctos entre servidores
@Override
public boolean confirmarIdentificacionCliente(String nombreCliente, String password) throws RemoteException {
    String passwordCliente = clientesRegistrados.get(nombreCliente).obtenerPassword();

    return password.equals(passwordCliente);
}

// Obtiene la cantidad total donada por un cliente
@Override
public float obtenerDonacionCliente(String nombreCliente) throws RemoteException {
    String nombreServidor = identificarCliente(nombreCliente);

    if(nombreServidor.equals(this.replica)){
        InterfazServidorReplica servidorReplica = this.obtenerReplica(host:"localhost", this.replica);
        Cliente cliente = servidorReplica.getClientesRegistrados().get(nombreCliente);
        return cliente.obtenerDonacionTotal();
    }else if(nombreServidor.equals(this.servidor)){
        Cliente cliente = this.clientesRegistrados.get(nombreCliente);
        return cliente.obtenerDonacionTotal();
    }
    return 0;
}

// Obtiene el numero de donaciones que ha hecho un cliente
@Override
public int obtenerNumeroDonacionesCliente(String nombreCliente) throws RemoteException {
    String nombreServidor = identificarCliente(nombreCliente);

    if(nombreServidor.equals(this.replica)){
        InterfazServidorReplica servidorReplica = this.obtenerReplica(host:"localhost", this.replica);
        Cliente cliente = servidorReplica.getClientesRegistrados().get(nombreCliente);
        return cliente.obtenerNumeroDonaciones();
    }else if(nombreServidor.equals(this.servidor)){
        Cliente cliente = this.clientesRegistrados.get(nombreCliente);
        return cliente.obtenerNumeroDonaciones();
    }
    return 0;
}



// Obtiene la donacion maxima hecha por un cliente al servidor
@Override
public float obtenerDonacionMaximaCliente(String nombreCliente) throws RemoteException {
    String nombreServidor = identificarCliente(nombreCliente);

    if(nombreServidor.equals(this.replica)){
        InterfazServidorReplica servidorReplica = this.obtenerReplica(host:"localhost", this.replica);
        Cliente cliente = servidorReplica.getClientesRegistrados().get(nombreCliente);
        return cliente.obtenerDonacionMaxima();
    }else if(nombreServidor.equals(this.servidor)){
        Cliente cliente = this.clientesRegistrados.get(nombreCliente);
        return cliente.obtenerDonacionMaxima();
    }
    return 0;
}
```

La función confirmarIdentificacion comprueba que el nombre y la contraseña coinciden ya usada anteriormente. Esta función se usa cuando ya se sabe a que servidor pertenece por lo tanto no hay comprobaciones.

La función ObtenerDonacion comprueba en qué servidor está, busca el cliente y obtiene el total donado.

La función obtenerNumeroDonaciones igual, pero con el número de donaciones del cliente.

La función obtenerDonacionMaxima hace los mismo pero te devuelve la donación más grande del cliente.

```java
// Obtiene el historial de donaciones de un cliente que ha hecho al servidor
@Override
public ArrayList<Float> obtenerHistorialDonaciones(String nombreCliente) throws RemoteException {
    String nombreServidor = identificarCliente(nombreCliente);

    if(nombreServidor.equals(this.replica)){
        InterfazServidorReplica servidorReplica = this.obtenerReplica(host:"localhost", this.replica);
        Cliente cliente = servidorReplica.getClientesRegistrados().get(nombreCliente);
        return cliente.obtenerHistorialDonaciones();
    }else if(nombreServidor.equals(this.servidor)){
        Cliente cliente = this.clientesRegistrados.get(nombreCliente);
        return cliente.obtenerHistorialDonaciones();
    }
    ArrayList<Float> lista = new ArrayList<Float>();
    return lista;
}

public Map<String, Cliente> getClientesRegistrados() {
    return clientesRegistrados;
}
// Donar en el servidor correspondiente cuando ya se ha comprobado en que servidor esta el cliente
public void donar(String nombreCliente, float donacion) throws RemoteException{
    Cliente cliente = clientesRegistrados.get(nombreCliente);
    this.incrementarSubtotalDonado(donacion);
    cliente.donar(donacion);
    System.out.println("El cliente con el nombre " + nombreCliente + " ha donado " + donacion);
}

public float obtenerTotalDonadoServidores() throws RemoteException{
    InterfazServidorReplica servidorReplica = this.obtenerReplica(host:"localhost", this.replica);
    return servidorReplica.getSubtotalDonado() + this.getSubtotalDonado();
}
```

La función obtenerHistorialDonaciones comprueba el servidor como siempre y obtiene el un arraylist de float que es el historial de donaciones.

La función getClientesRegistrado obtiene el mapa de clientesRegistrados para poder buscar en ellas lo que se necesario.

La función donar , que se usa después de todas las comprobaciones obtiene el cliente y se llama la función donar de cliente.

La función obtenerTotalDonadoServidores suma el subtotal de ambos servidores y returna la suma.

```java
    // obtiene la lista de clientes que han donado en ambos servidores
    public ArrayList<Cliente> obtenerListeDonantes() throws RemoteException{
        InterfazServidorReplica servidorReplica = this.obtenerReplica(host:"localhost", this.replica);
        Map<String,Cliente> clientesReplica = servidorReplica.getClientesRegistrados();

        Map<String,Cliente> clientesServidor = this.getClientesRegistrados();

        ArrayList<Cliente> listaClientes = new ArrayList<>();

          // Agregar los clientes de la réplica al ArrayList
        for (Cliente cliente : clientesReplica.values()) {
            if(cliente.obtenerNumeroDonaciones() > 0){
                listaClientes.add(cliente);
            }
        }

        // Agregar los clientes del servidor actual al ArrayList
        for (Cliente cliente : clientesServidor.values()) {
            if(cliente.obtenerNumeroDonaciones() > 0){
                listaClientes.add(cliente);
            }
        }

        return listaClientes;
    }

    //obtiene la lista de clientes que ordenado de mayor a menor segun el total donado
    public ArrayList<Cliente> obtenerRanking() throws RemoteException{
        InterfazServidorReplica servidorReplica = this.obtenerReplica(host:"localhost", this.replica);
        Map<String,Cliente> clientesReplica = servidorReplica.getClientesRegistrados();

        Map<String,Cliente> clientesServidor = this.getClientesRegistrados();

        ArrayList<Cliente> listaClientes = new ArrayList<>();

        // Agregar los clientes de la réplica al ArrayList
        for (Cliente cliente : clientesReplica.values()) {
            if(cliente.obtenerNumeroDonaciones() > 0){
                listaClientes.add(cliente);
            }
        }

        // Agregar los clientes del servidor actual al ArrayList
        for (Cliente cliente : clientesServidor.values()) {
            if(cliente.obtenerNumeroDonaciones() > 0){
                listaClientes.add(cliente);
            }
        }

        listaClientes.sort((c1, c2) -> Float.compare(c2.obtenerDonacionTotal(), c1.obtenerDonacionTotal()));

            return listaClientes;
    }


}
```

Finalmente la función obtenerListaDonantes(), obtiene todos los donantes de ambos servidores y obtiene los usuarios que mínimo tienen 1 o más número de donaciones.

La función obtenerRanking es igual que la anterior nada mas que te los ordena haciendo un ranking de mayor a menor de los donantes.

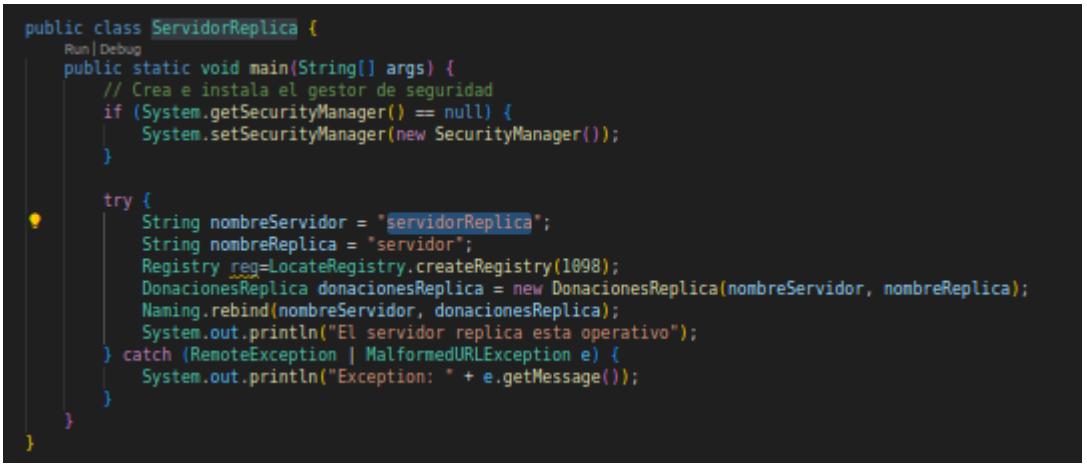
DonacionesReplica.java es igual, siempre comprueba en donde esta y luego busca la información.

Servidor.java:

```
public class Servidor {
    Run | Debug
    public static void main(String[] args) {
        // Crea e instala el gestor de seguridad
        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new SecurityManager());
        }

        try {
            String nombreServidor = "servidor";
            String nombreReplica = "servidorReplica";
            Registry reg=LocateRegistry.createRegistry(1099);
            DonacionesServidor donacionesServidor = new DonacionesServidor(nombreServidor, nombreReplica);
            Naming.rebind(nombreServidor, donacionesServidor);
            System.out.println("El servidor principal esta operativo");
        } catch (RemoteException | MalformedURLException e) {
            System.out.println("Exception: " + e.getMessage());
        }
    }
}
```

Inicializamos el servidor y mandamos tanto el nombre como el nombre de su réplica.

ServidorReplica.java:



```
public class ServidorReplica {
    Run | Debug
    public static void main(String[] args) {
        // Crea e instala el gestor de seguridad
        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new SecurityManager());
        }

        try {
            String nombreServidor = "servidorReplica";
            String nombreReplica = "servidor";
            Registry reg=LocateRegistry.createRegistry(1098);
            DonacionesReplica donacionesReplica = new DonacionesReplica(nombreServidor, nombreReplica);
            Naming.rebind(nombreServidor, donacionesReplica);
            System.out.println("El servidor replica esta operativo");
        } catch (RemoteException | MalformedURLException e) {
            System.out.println("Exception: " + e.getMessage());
        }
    }
}
```

Lo mismo que el Servidor.java nada mas cambiando el puerto y los nombre que mandamos.

ProgramaCliente.java:
Como en las anteriores prácticas he creado un menú para que el usuario vaya eligiendo su decisión.

```java
public class ProgramaCliente {
    public static void imprimirMenuPrincipal() {
        System.out.println("1--> Registrarse");
        System.out.println("2--> Iniciar sesión");
        System.out.println("3--> Salir");
    }

    public static void imprimirMenuOperaciones() {
        System.out.println("1--> Realizar una donación");
        System.out.println("2--> Consultar el total donado al servidor(Replica)");
        System.out.println("3--> Consultar el total donado por el cliente");
        System.out.println("4--> Consultar la donación más grande del cliente");
        System.out.println("5--> Consultar el numero de donaciones del cliente");
        System.out.println("6--> Consultar el historial de donaciones del cliente");
        System.out.println("7--> Consultar total donado en ambos servidores");
        System.out.println("8--> Consultar la lista de donantes");
        System.out.println("9--> Consultar Ranking de donantes");
        System.out.println("10--> Salir");
    }

    public static void imprimirHistorialDonaciones(String nombreCliente, ArrayList<Float> historial) {

        if(historial.isEmpty()) {
            System.out.println("El cliente " + nombreCliente + " no ha realizado ninguna donacion");
        }
        else {
            System.out.println("Historial de donaciones del cliente " + nombreCliente + ":");

            for(int i = 0; i < historial.size(); i++) {
                System.out.println("Donacion " + (i+1) + ": " + historial.get(i));
            }
        }
    }

    public static void imprimirRanking(ArrayList<Cliente> Ranking){
        if(Ranking.isEmpty()) {
            System.out.println("No hay donantes");
        }
        else {
            System.out.println("Ranking de donantes: ");

            for(int i = 0; i < Ranking.size(); i++) {
                System.out.println("Donante " + (i+1) + ": " + Ranking.get(i).obtenerNombre() + " ha donado: " + Ranking.get(i).obtenerDonacionTotal());
            }
        }
    }

    public static void imprimirListadoDonantes(ArrayList<Cliente> Donantes){
        if(Donantes.isEmpty()) {
            System.out.println("No hay donantes");
        }
        else {
            System.out.println("Historial de donantes: ");

            for(int i = 0; i < Donantes.size(); i++) {
                System.out.println("Donante " + (i+1) + ": " + Donantes.get(i).obtenerNombre() + " ha donado: " + Donantes.get(i).obtenerDonacionTotal());
            }
        }
    }
```

Estas son funciones auxiliares para mostrar las opciones o incluso mostrar la listas de donantes, ranking, que nos mandan.

```java
Run | Debug
public static void main(String[] args) {
    String host = "localhost";
    String nombreServidor = "servidorReplica";
    String cliente = "";
    String password = "1234";
    float cantidad = 0.0f;
    boolean sesionIniciada = false;

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

    try {
        Registry mireg = LocateRegistry.getRegistry(host, 1099);
        InterfazServidorCliente servidor = (InterfazServidorCliente) mireg.lookup(nombreServidor);

        System.out.println("Bienvenido al servidor de donaciones");
        System.out.println("Seleccione una opcion: ");

        Scanner sc = new Scanner(System.in);
        int opcion = 0;

        do {
            imprimirMenuPrincipal();
            System.out.print("Opción: ");
            opcion = sc.nextInt();
            System.out.println();

            if(opcion < 1 || opcion > 3)
                System.out.println("La opción seleccionada no es válida");

        } while (opcion < 1 || opcion > 3);

        switch (opcion) {
            case 1:
                System.out.print("Introduzca el nombre: ");
                cliente = sc.nextLine();
                cliente = sc.nextLine();

                if(servidor.registrado(cliente)) {
                    System.out.println("El nombre introducido ya está registrado, pruebe con otro");
                    System.out.println ("Intentelo de nuevo, saliendo...");
                } else {
                    System.out.print("Introduzca la contraseña: ");
                    password = sc.nextLine();
                    System.out.println();

                    if (servidor.realizarRegistro(cliente, password)) {
                        System.out.println(cliente + " se ha registrado");
                        sesionIniciada = true;
                    }
                    else
                        System.out.println("No se ha podido registrar el cliente");
                }
                break;

            case 2:
                System.out.print("Introduzca el nombre: ");
                cliente = sc.nextLine();
                cliente = sc.nextLine();

                System.out.print("Introduzca la contraseña: ");
                password = sc.nextLine();

                if (servidor.inicioSesion(cliente, password)) {
                    System.out.println("El cliente se ha identificado");
                    sesionIniciada = true;
                }
                else
                    System.out.println("No se ha podido identificar el cliente");

                break;

            case 3:
                    System.out.println("Saliendo...");
                break;
        }
```

buscamos el servidor y empezamos con el menuPrincipal con las opciones de registrar, iniciarSesion y salir.
un switch, si se registra bien, inicia sesión directamente con el bool y en iniciarSesion usa la función iniciosesion donde si todo está bien inicia sesion con el bool también. Si usa la opción 3 sale, como las anteriores prácticas hacemos las comprobaciones de que introducimos bien las opciones.

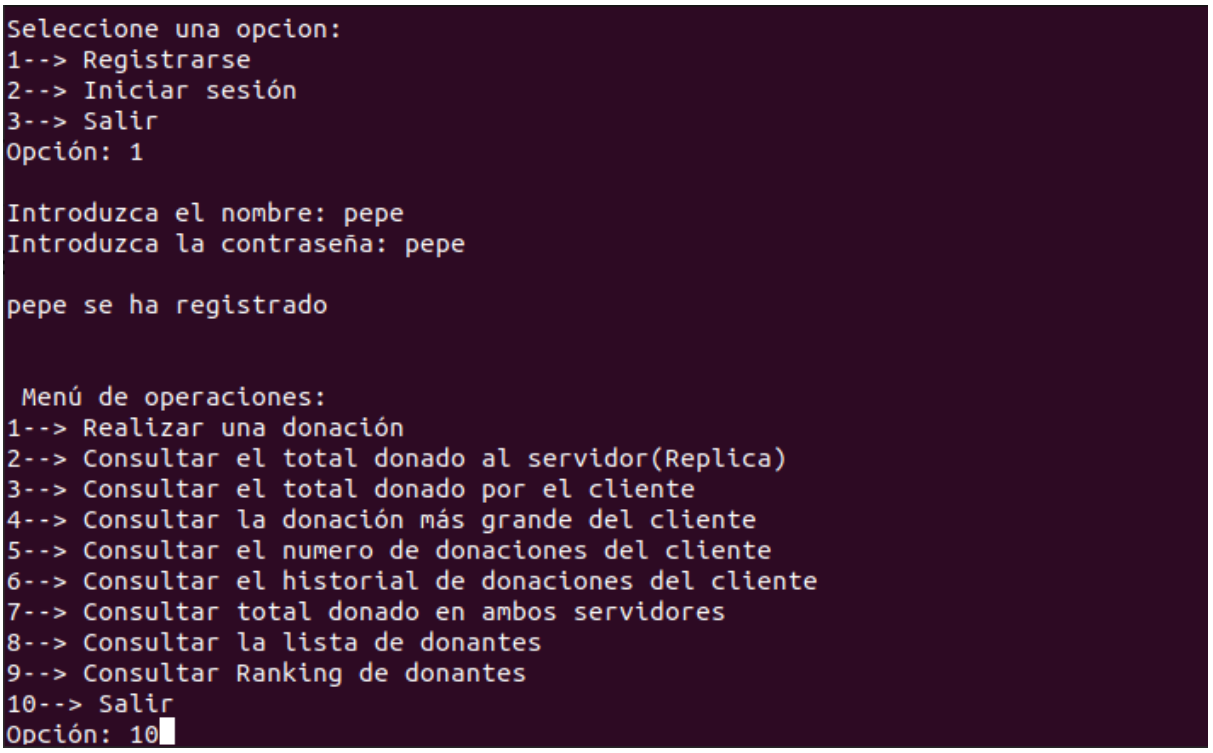
```
    if (sesionIniciada) {
        while(true){
        System.out.println("\n Menú de operaciones: ");
        do {
            imprimirMenuOperaciones();
            System.out.print("Opción: ");
            opcion = sc.nextInt();
            System.out.println();

            if(opcion < 1 || opcion > 10)
                System.out.println("La opción seleccionada no es válida");

        } while (opcion < 1 || opcion > 10);

        switch (opcion) {
            case 1:
                System.out.print("Introduzca la cantidad a donar: ");
                cantidad = sc.nextFloat();
                servidor.realizarDonacion(cliente, cantidad);
                System.out.println(cliente + " ha donado " + cantidad + " al servidor");
                break;

            case 2:
                System.out.println("Se ha donado en total al servidor " + servidor.obtenerSubtotalDonado(cliente));
                break;

            case 3:
                System.out.println(cliente + " ha donado en total al servidor " + servidor.obtenerDonacionCliente(cliente));
                break;

            case 4:
                System.out.println("La donación más grande de " + cliente + " es " + servidor.obtenerDonacionMaximaCliente(cliente));
                break;

            case 5:
                int vecesDonadas = servidor.obtenerNumeroDonacionesCliente(cliente);

                if(vecesDonadas == 0)
                    System.out.println(cliente + " todavia no ha hecho ninguna donación");
                else if (vecesDonadas == 1)
                    System.out.println(cliente + " ha donado 1 vez");
                else
                    System.out.println(cliente + " ha donado " + vecesDonadas + " veces");

                break;

            case 6:
                ArrayList<Float> historial = servidor.obtenerHistorialDonaciones(cliente);
                imprimirHistorialDonaciones(cliente, historial);
                break;
            case 7:
            System.out.println("Se ha donado en total al servidor " + servidor.obtenerTotalDonadoServidores());
                break;
            case 8:
                ArrayList<Cliente> listado = servidor.obtenerListeDonantes();
                imprimirListadoDonantes(listado);
                break;
            case 9:
                ArrayList<Cliente> ranking = servidor.obtenerRanking();
                imprimirRanking(ranking);
                break;
            case 10:
                break;

        }
        if(opcion == 10){
            break;
        }
        }

    }
} catch(NotBoundException | RemoteException e) {
    System.err.println("Exception del sistema: " + e);
}
```

Si sesionIniciada es true significa que ha accedido correctamente y le muestra el siguiente menú anteriormente mostrado.

Comprueba bien que las opciones son válidas y dependiendo de lo que elija usa una función del servidor diferente. Muestra las listas que le pasan gracias a las funciones auxiliares que he mostrado al principio.

Funcionamiento y capturas de pantalla

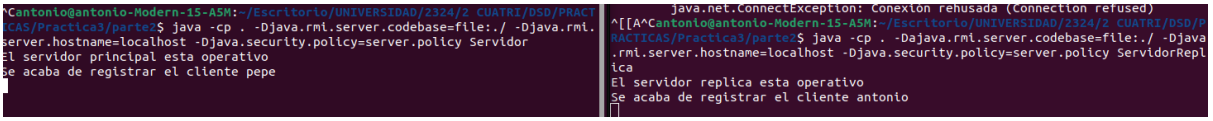
En el readme esta los comandos que se han de ejecutar, aquí mostraré directamente el funcionamiento y como usar el programa.



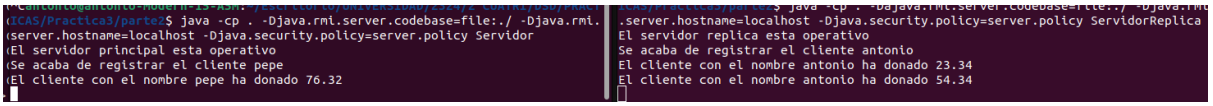
Registro a un usuario como pepe y ahora salgo para registrar a otro.

```
Seleccione una opcion:
1--> Registrarse
2--> Iniciar sesión
3--> Salir
Opción: 1

Introduzca el nombre: antonio
Introduzca la contraseña: antonio

antonio se ha registrado


 Menú de operaciones:
1--> Realizar una donación
2--> Consultar el total donado al servidor(Replica)
3--> Consultar el total donado por el cliente
4--> Consultar la donación más grande del cliente
5--> Consultar el numero de donaciones del cliente
6--> Consultar el historial de donaciones del cliente
7--> Consultar total donado en ambos servidores
8--> Consultar la lista de donantes
9--> Consultar Ranking de donantes
10--> Salir
Opción:
```

Como se puede ver en la captura de abajo el registro se hace en el servidor con menos clientes.



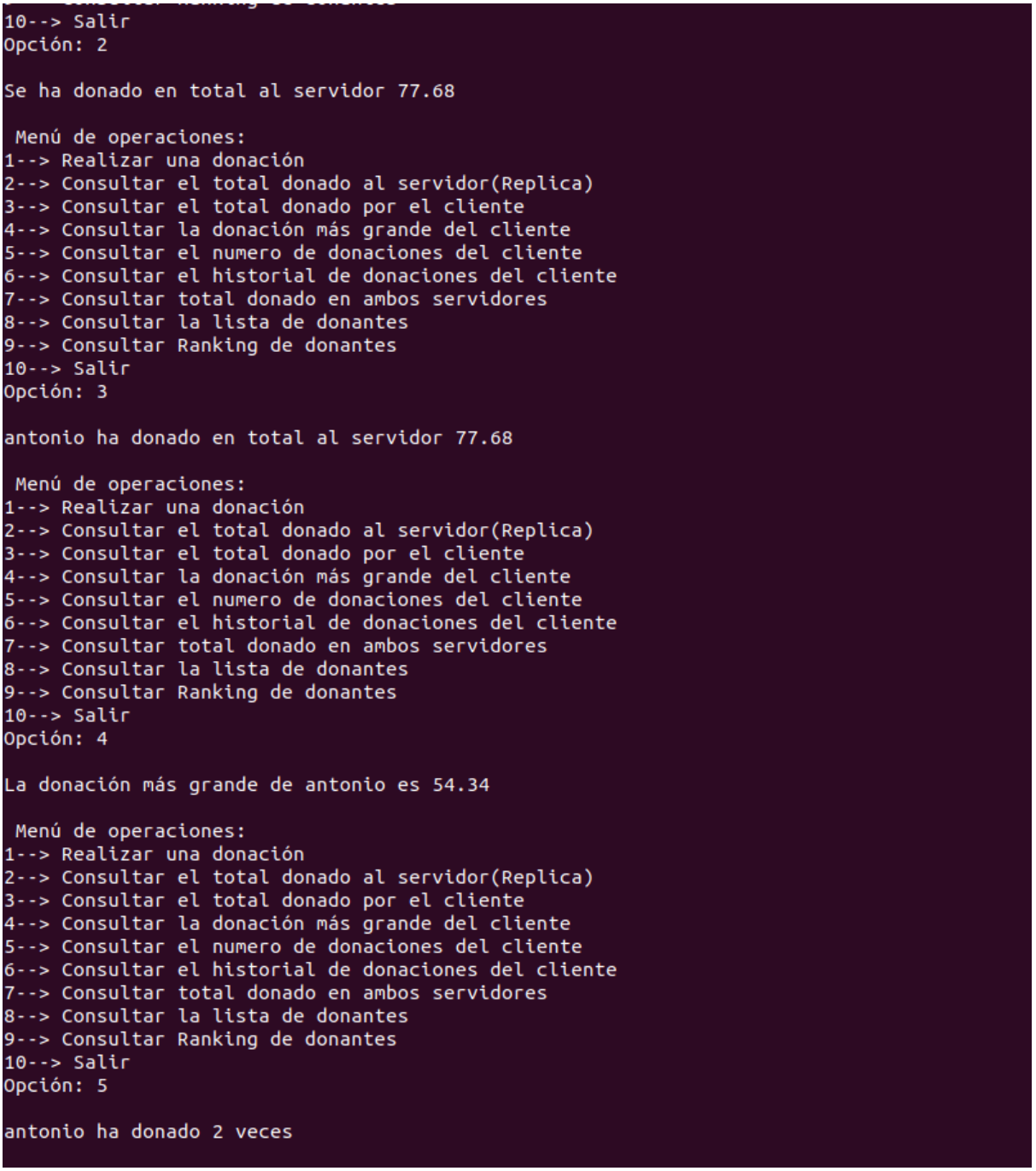
Ahora donare con ambos para ver el subtotal, el total, ranking ,etc.
primero he donado lo siguiente:



Nota: La donación hay que hacerlo por ejemplo → 34,54 (con comas siempre para que no de error).

Iniciar sesión erróneamente para comprobar como va y luego ejecutar todas las funciones.

```
 -cp . -Djava.security.policy=server.policy ProgramaCliente
Bienvenido al servidor de donaciones
Seleccione una opcion:
1--> Registrarse
2--> Iniciar sesión
3--> Salir
Opción: 2

Introduzca el nombre: antonio
Introduzca la contraseña: 1
No se ha podido identificar el cliente

antonio@antonio-Modern-15-A5M:~/Escritorio/UNIVERSIDAD/2324/2 CUATRI/DSD/PRACTICAS/Practica3/parte2$ java
 -cp . -Djava.security.policy=server.policy ProgramaCliente
Bienvenido al servidor de donaciones
Seleccione una opcion:
1--> Registrarse
2--> Iniciar sesión
3--> Salir
Opción: 2

Introduzca el nombre: antonio
Introduzca la contraseña: antonio
El cliente se ha identificado
```



Como vemos aunque hayamos donado en con ambos usuarios cada réplica tiene su subtotal como muestra la opción 2.

```
9--> Consultar Ranking de donantes
10--> Salir
Opción: 6

Historial de donaciones del cliente antonio:
Donacion 1: 23.34
Donacion 2: 54.34

 Menú de operaciones:
1--> Realizar una donación
2--> Consultar el total donado al servidor(Replica)
3--> Consultar el total donado por el cliente
4--> Consultar la donación más grande del cliente
5--> Consultar el numero de donaciones del cliente
6--> Consultar el historial de donaciones del cliente
7--> Consultar total donado en ambos servidores
8--> Consultar la lista de donantes
9--> Consultar Ranking de donantes
10--> Salir
Opción: 7

Se ha donado en total al servidor 154.0

 Menú de operaciones:
1--> Realizar una donación
2--> Consultar el total donado al servidor(Replica)
3--> Consultar el total donado por el cliente
4--> Consultar la donación más grande del cliente
5--> Consultar el numero de donaciones del cliente
6--> Consultar el historial de donaciones del cliente
7--> Consultar total donado en ambos servidores
8--> Consultar la lista de donantes
9--> Consultar Ranking de donantes
10--> Salir
Opción: 8

Historial de donantes:
Donante 1: pepe ha donado: 76.32
Donante 2: antonio ha donado: 77.68

 Menú de operaciones:
1--> Realizar una donación
2--> Consultar el total donado al servidor(Replica)
3--> Consultar el total donado por el cliente
4--> Consultar la donación más grande del cliente
5--> Consultar el numero de donaciones del cliente
6--> Consultar el historial de donaciones del cliente
7--> Consultar total donado en ambos servidores
8--> Consultar la lista de donantes
9--> Consultar Ranking de donantes
10--> Salir
Opción: 9

Ranking de donantes:
Donante 1: antonio ha donado: 77.68
Donante 2: pepe ha donado: 76.32
```

Aquí vemos el historial, el total en ambos servidores, lista de donantes que no está en orden y el ranking ordenado de mayor a menor dependiendo de

quien ha donado más. A continuación con más donaciones y más usuarios.

```
6--> Consultar el historial de donaciones del cliente
7--> Consultar total donado en ambos servidores
8--> Consultar la lista de donantes
9--> Consultar Ranking de donantes
10--> Salir
Opción: 7

Se ha donado en total al servidor 5080.2896

 Menú de operaciones:
1--> Realizar una donación
2--> Consultar el total donado al servidor(Replica)
3--> Consultar el total donado por el cliente
4--> Consultar la donación más grande del cliente
5--> Consultar el numero de donaciones del cliente
6--> Consultar el historial de donaciones del cliente
7--> Consultar total donado en ambos servidores
8--> Consultar la lista de donantes
9--> Consultar Ranking de donantes
10--> Salir
Opción: 2

Se ha donado en total al servidor 436.65002

 Menú de operaciones:
1--> Realizar una donación
2--> Consultar el total donado al servidor(Replica)
3--> Consultar el total donado por el cliente
4--> Consultar la donación más grande del cliente
5--> Consultar el numero de donaciones del cliente
6--> Consultar el historial de donaciones del cliente
7--> Consultar total donado en ambos servidores
8--> Consultar la lista de donantes
9--> Consultar Ranking de donantes
10--> Salir
Opción: 9

Ranking de donantes:
Donante 1: pepe2 ha donado: 4567.32
Donante 2: martina ha donado: 358.97003
Donante 3: antonio ha donado: 77.68
Donante 4: pepe ha donado: 76.32

 Menú de operaciones:
1--> Realizar una donación
2--> Consultar el total donado al servidor(Replica)
3--> Consultar el total donado por el cliente
4--> Consultar la donación más grande del cliente
5--> Consultar el numero de donaciones del cliente
6--> Consultar el historial de donaciones del cliente
7--> Consultar total donado en ambos servidores
8--> Consultar la lista de donantes
9--> Consultar Ranking de donantes
10--> Salir
Opción:
```

A Continuación vemos como no te deja registrar nuevos usuarios repetido ya sea que esté registrado en la réplica o en el servidor.