Práctica 3: RMI

Antonio Ángel Granados Luque

- Parte 1: Ejemplos

Ejemplo 1: En este ejemplo se imprime por pantalla un mensaje de forma remota en función del identificador de proceso que lo invoca y que pasa como parámetro, si el identificador es 0, suspende el proceso por 5 segundos.

Lanzamos el servidor tal y como viene en el guión. Primero ejecutamos RMI registry con la orden `rmiregistry &` (& para dejarlo en segundo plano). Por defecto el registry se ejecuta en el puerto 1099. Compilamos todos los fuentes con la opción `javac *.java` y a continuación ya si lanzamos el servidor con:

```
java -cp. -Djava.rmi.server.codebase=file./
-Djava.rmi.server.hostname=localhost
-Djava.security.policy=server.policy Ejemplo
```

Todo esto dentro del directorio donde están los .class.
Ya tenemos el servidor funcionando, ahora ejecutamos varios clientes y comprobamos los resultados.

Los clientes los ejecutamos con la orden: `java -cp . -Djava.security.policy=server.policy Cliente_Ejemplo localhost 0`

El número pasado como argumento dependerá de nosotros, por lo que hacemos pruebas con el 3, el 5, y el 0, que es el caso especial.

```
antonio@antonio-VirtualBox:~/Escritorio/DSD/p3/Ejemplo_1$ java -cp . -Djava.security.policy=server.po
licy Cliente_Ejemplo localhost 3
Buscando el objeto remoto
Invocando al objeto remoto
antonio@antonio-VirtualBox:~/Escritorio/DSD/p3/Ejemplo_1$ java -cp . -Djava.security.policy=server.po
licy Cliente_Ejemplo localhost 5
Buscando el objeto remoto
Invocando al objeto remoto
antonio@antonio-VirtualBox:~/Escritorio/DSD/p3/Ejemplo_1$ java -cp . -Djava.security.policy=server.po
licy Cliente_Ejemplo localhost 0
Buscando el objeto remoto
Invocando al objeto remoto
antonio@antonio-VirtualBox:~/Escritorio/DSD/p3/Ejemplo_1$
```

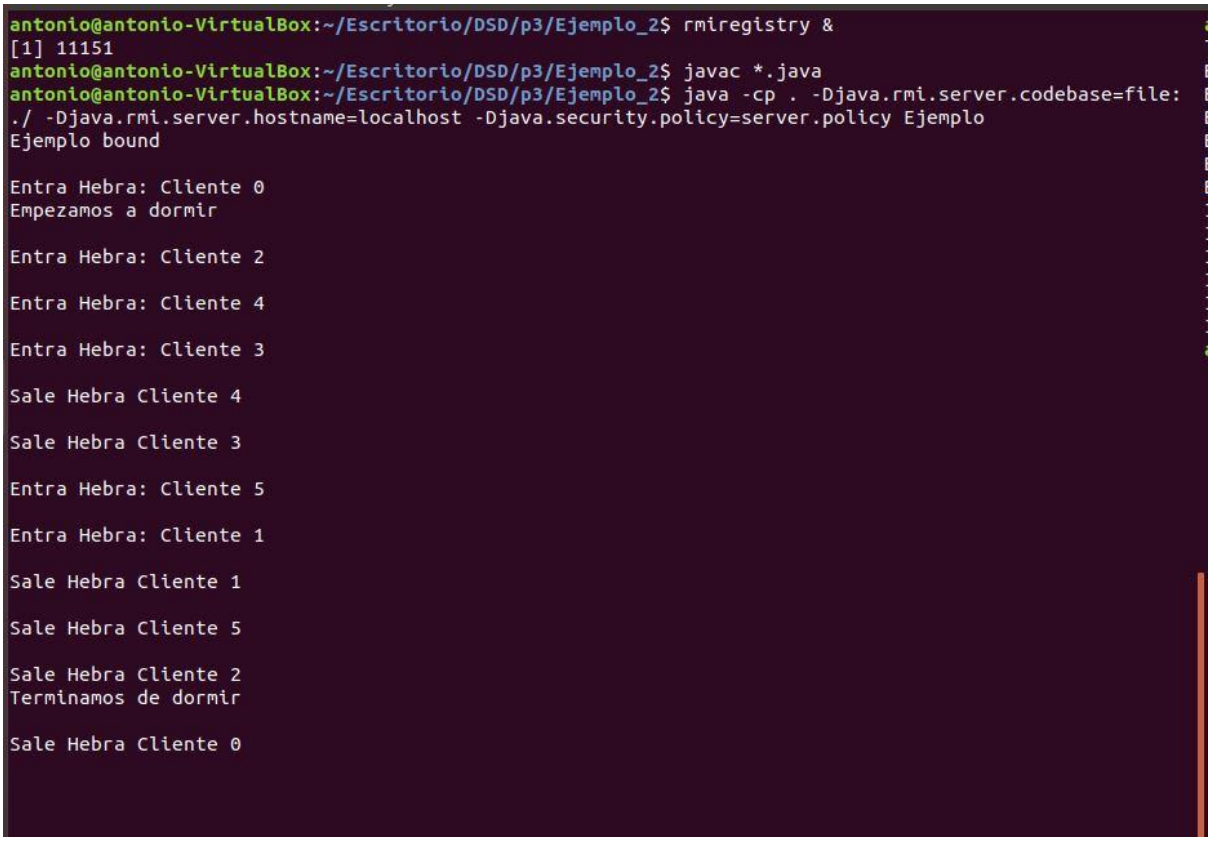
En el servidor nos aparece lo siguiente:

Como se ve realiza una correcta impresión de los identificadores que han llamado a los métodos, y con el 0 el proceso queda dormido 5 segundos.

Ejemplo 2: En lugar de lanzar varios clientes, creamos varias hebras que realizan la misma tarea de imprimir un mensaje remoto accediendo al stub de un objeto remoto. En esta ocasión en vez de pasar al objeto remoto un número entero, pasamos una cadena String.

Lanzamos el servidor como antes y ahora ejecutamos los clientes de la misma manera, pero teniendo en cuenta que ahora el número que pasamos como parámetro será el número de hebras que se van a crear.

Probamos con 6 hebras:



Como se ve se crean 6 hebras con identificadores del 0 al 5, por lo que la primera hebra se queda suspendida mientras que entran y salen otras hebras. Se puede apreciar que las hebras no se sincronizan y los mensajes se entrelazan.

Agregando el modificador *synchronized* en el método de la implementación remota conseguimos que esto no ocurra, teniendo las hebras sincronizadas, ninguna entra hasta que la anterior no termina.

Hacemos el mismo ejemplo con 6 hebras:

```
antonio@antonio-VirtualBox:~/Escritorio/DSD/p3/Ejemplo_2$ java -cp . -Djava.security.policy=server.po
licy Cliente_Ejemplo_Multi_Threaded localhost 6
Buscando el objeto remoto ...
Buscando el objeto remoto ...
Buscando el objeto remoto ...
Buscando el objeto remoto ...
Buscando el objeto remoto ...
Buscando el objeto remoto ...
Invocando al objeto remoto ...
Invocando al objeto remoto ...
Invocando al objeto remoto ...
Invocando al objeto remoto ...
Invocando al objeto remoto ...
Invocando al objeto remoto ...
antonio@antonio-VirtualBox:~/Escritorio/DSD/p3/Ejemplo_2$ java -cp . -Djava.security.policy=server.po
licy Cliente_Ejemplo_Multi_Threaded localhost 6
Buscando el objeto remoto ...
Buscando el objeto remoto ...
Buscando el objeto remoto ...
Buscando el objeto remoto ...
Buscando el objeto remoto ...
Buscando el objeto remoto ...
Invocando al objeto remoto ...
Invocando al objeto remoto ...
Invocando al objeto remoto ...
Invocando al objeto remoto ...
Invocando al objeto remoto ...
Invocando al objeto remoto ...
antonio@antonio-VirtualBox:~/Escritorio/DSD/p3/Ejemplo_2$
```

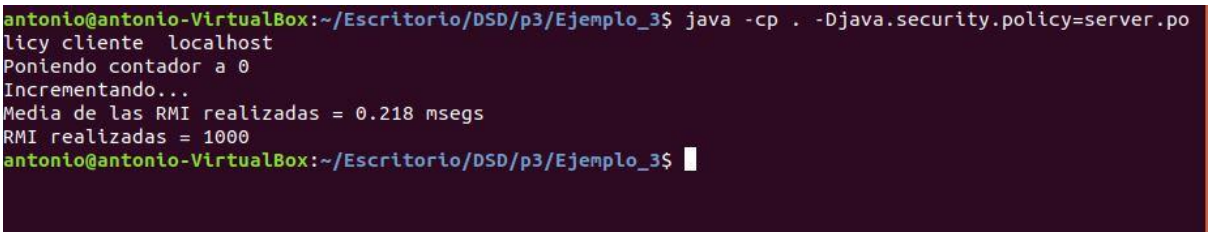
```
antonio@antonio-VirtualBox:~/Escritorio/DSD/p3/Ejemplo_2$ rmiregistry &
[1] 11284
antonio@antonio-VirtualBox:~/Escritorio/DSD/p3/Ejemplo_2$ javac *.java
antonio@antonio-VirtualBox:~/Escritorio/DSD/p3/Ejemplo_2$ java -cp . -Djava.rmi.server.codebase=file:
./ -Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy Ejemplo
Ejemplo bound

Entra Hebra: Cliente 3

Sale Hebra Cliente 3

Entra Hebra: Cliente 0
Empezamos a dormir
Terminamos de dormir

Sale Hebra Cliente 0

Entra Hebra: Cliente 1

Sale Hebra Cliente 1

Entra Hebra: Cliente 4

Sale Hebra Cliente 4

Entra Hebra: Cliente 2

Sale Hebra Cliente 2

Entra Hebra: Cliente 5

Sale Hebra Cliente 5
```

Se puede apreciar que efectivamente las hebras están sincronizadas.

Ejemplo 3: Este programa es un pequeño ejemplo cliente - servidor. En este ejemplo se crea por una parte el objeto remoto y por otra el servidor. El Objeto remoto consta de varias funciones accesibles remotamente. El servidor (servidor.java) exporta los métodos contenidos en la interfaz icontador.java del objeto remoto instanciado como micontador de la clase definida en contador.java.

El programa cliente es un programa normal Java que realiza las siguientes acciones: Pone un valor inicial en el contador del servidor e invoca el método incrementar del contador 1.000 veces

Lanzamos el servidor como las anteriores veces y ejecutamos el cliente, esta vez sin argumentos.



Como se ve primero se pone el contador a 0, y luego se realiza 1000 veces el incremento del contador.

- Parte 2: Ejercicio Propuesto

  Para este ejercicio he definido 2 interfaces, una que implementan los servidores de cara al cliente y otra para que un servidor actúe como réplica, podría decirse para que actúe como cliente de otro servidor.

  La primera interfaz tiene las operaciones de registrar un usuario, realizar el depósito, consultar el total donado, identificar a un usuario y obtener el total donado de un usuario.

  Para la segunda interfaz se proponen operaciones para devolver datos que necesite el otro servidor, como comprobar si existe un cliente registrado, devolver el número de clientes registrados en ese servidor (necesario para realizar el registro en el servidor con menos clientes), devolver el subtotal donado, devolver una referencia de la réplica e incrementar el subtotal debido a una donación.

  Para almacenar los datos de un cliente he propuesto un diseño basado en una clase, la cual contiene los datos nombre, contraseña y total donado por ese cliente. Para hacer los cálculos más sencillos, en la implementación de las interfaces (ficheros Donaciones.java y Donaciones2.java) he trabajado con un map que contiene un par<String, Cliente> donde el String es el nombre del cliente y Cliente es el cliente con todos sus datos. Gracias a algunas funciones que proporciona algunas de las funcionalidades han sido más fáciles de implementar.

  El cliente consta con un menú inicial con el que puede registrarse o logearse. Las opciones incorrectas tienen su correspondiente mensaje de error. Después de logearse tiene otro menú con operaciones como donar, consultar el total donado, consultar lo que él ha donado y cerrar sesión.

  A continuación vamos a ver un ejemplo de cómo se realizaría todo, y cómo actúan los servidores.

  Primero lanzamos el primer servidor, fichero Servidor1, ya que es éste el que crea el registro RMI

```
antonio@antonio-VirtualBox:~/Escritorio/DSD/P3/Ejercicio$ java -cp . -Djava.rmi.
server.codebase=file:./ -Djava.rmi.server.hostname=localhost -Djava.security.pol
icy=server.policy Servidor1
Servidor1 preparado.
Nuevo cliente: antonio
Intentando identificar al cliente antonio.
Resultado = true
El cliente antonio está consultando su cantidad donada
Donación de 5.0 del cliente antonio.
El cliente antonio está consultando su cantidad donada
```

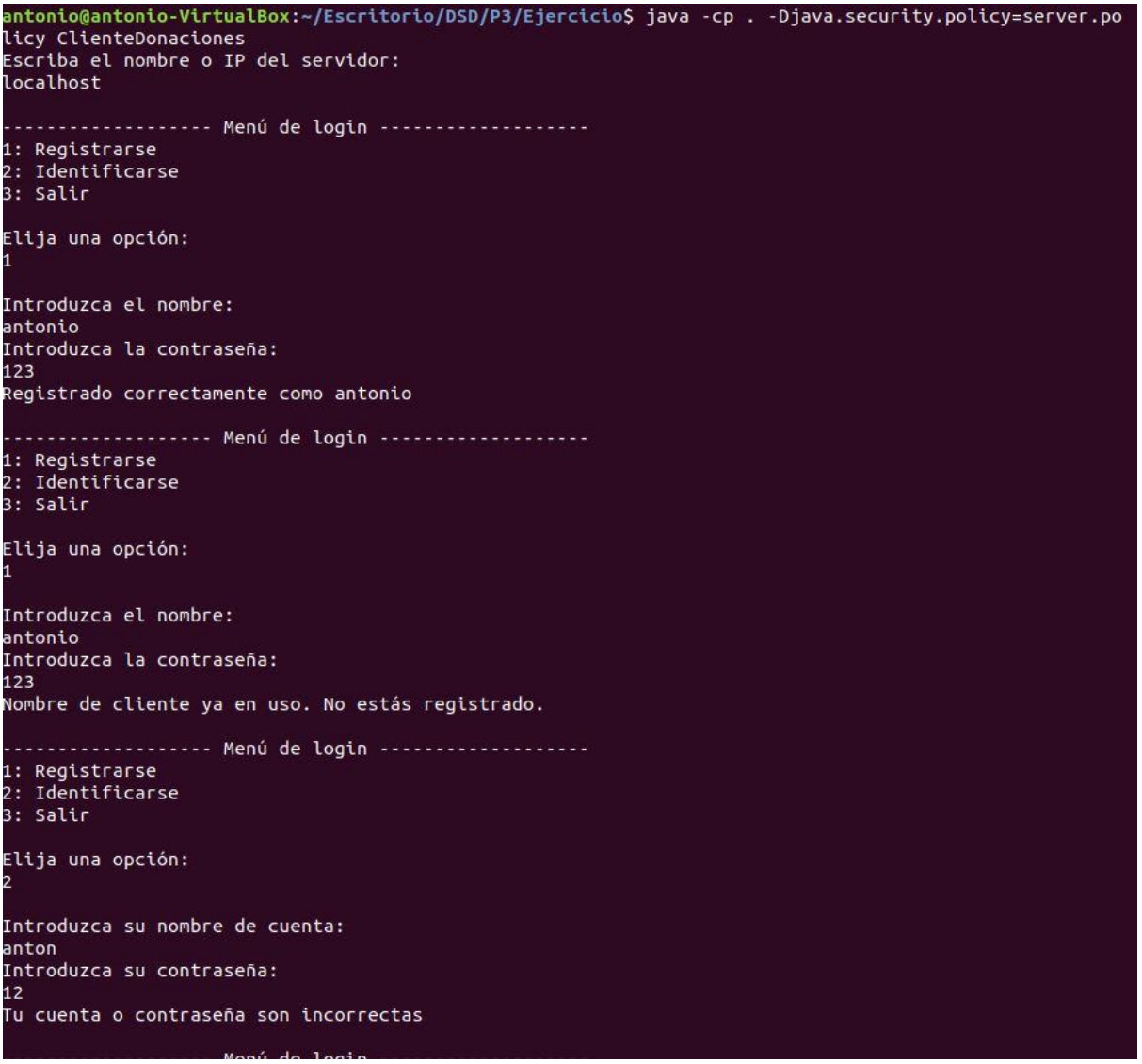
```
antonio@antonio-VirtualBox:~/Escritorio/DSD/P3/Ejercicio$ java -cp . -Djava.rmi.
server.codebase=file:./ -Djava.rmi.server.hostname=localhost -Djava.security.pol
icy=server.policy Servidor2
Servidor Servidor2 preparado.
Nuevo cliente: angel
Intentando identificar al cliente angel.
Resultado = true
El cliente angel está consultando su cantidad donada
Donación de 123.0 del cliente angel.
El cliente angel está consultando su cantidad donada
```

Más adelante se comentará las trazas que aparecen en los servidores, por ahora nos quedamos con que están iniciados.

Antes de lanzarlos por supuesto hay que tener todos los fuentes compilados, en mi caso tengo todos los fuentes en la misma carpeta, por lo que con la opción `javac *.java` es más que suficiente.

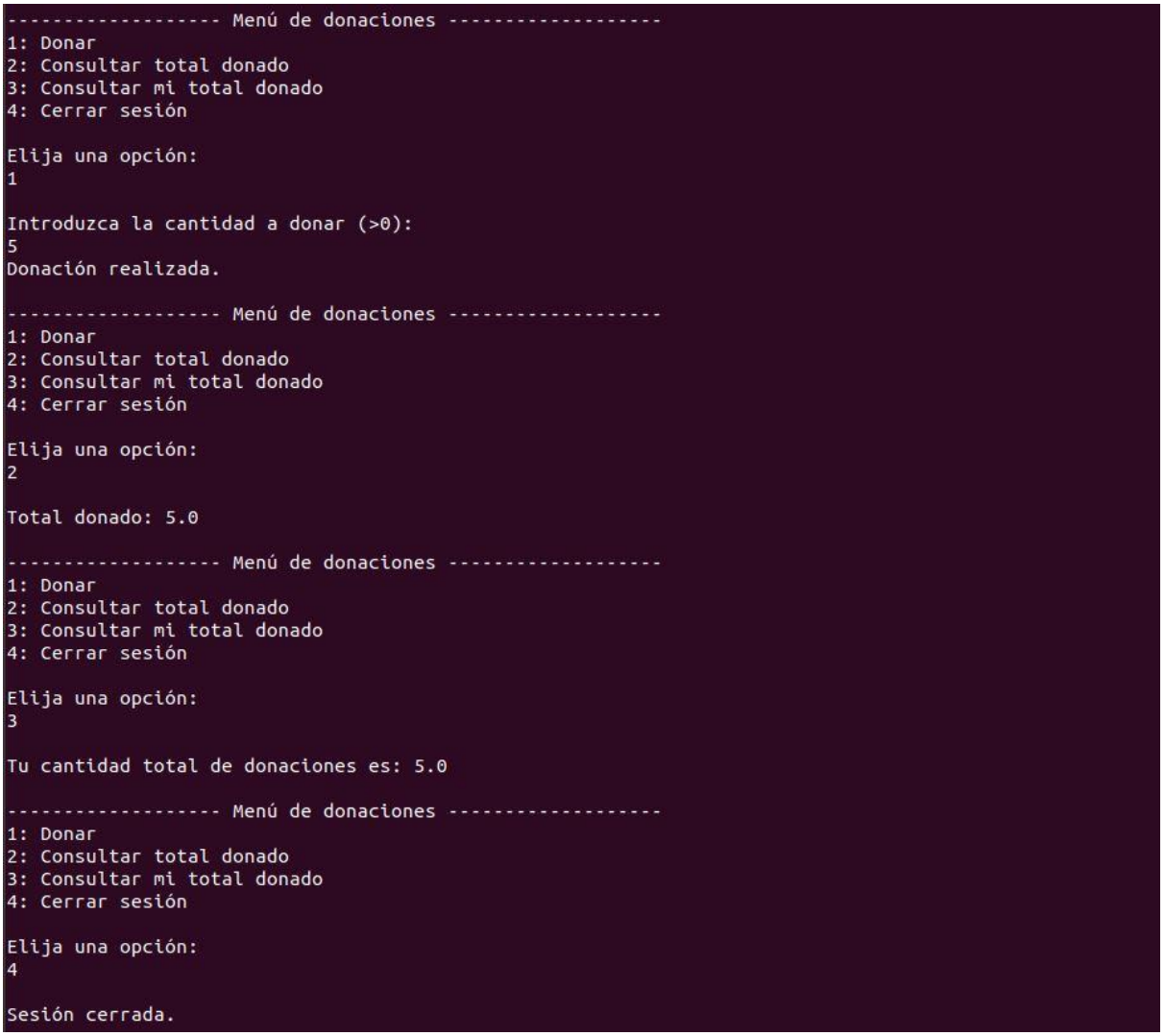
Ahora lanzamos un cliente, y realizamos una serie de pasos que se muestran a continuación:

Primero nos registramos con el usuario “antonio” con contraseña “123”. Luego probamos a logearnos un par de veces con credenciales incorrectas para comprobar efectivamente que no existe otro usuario.



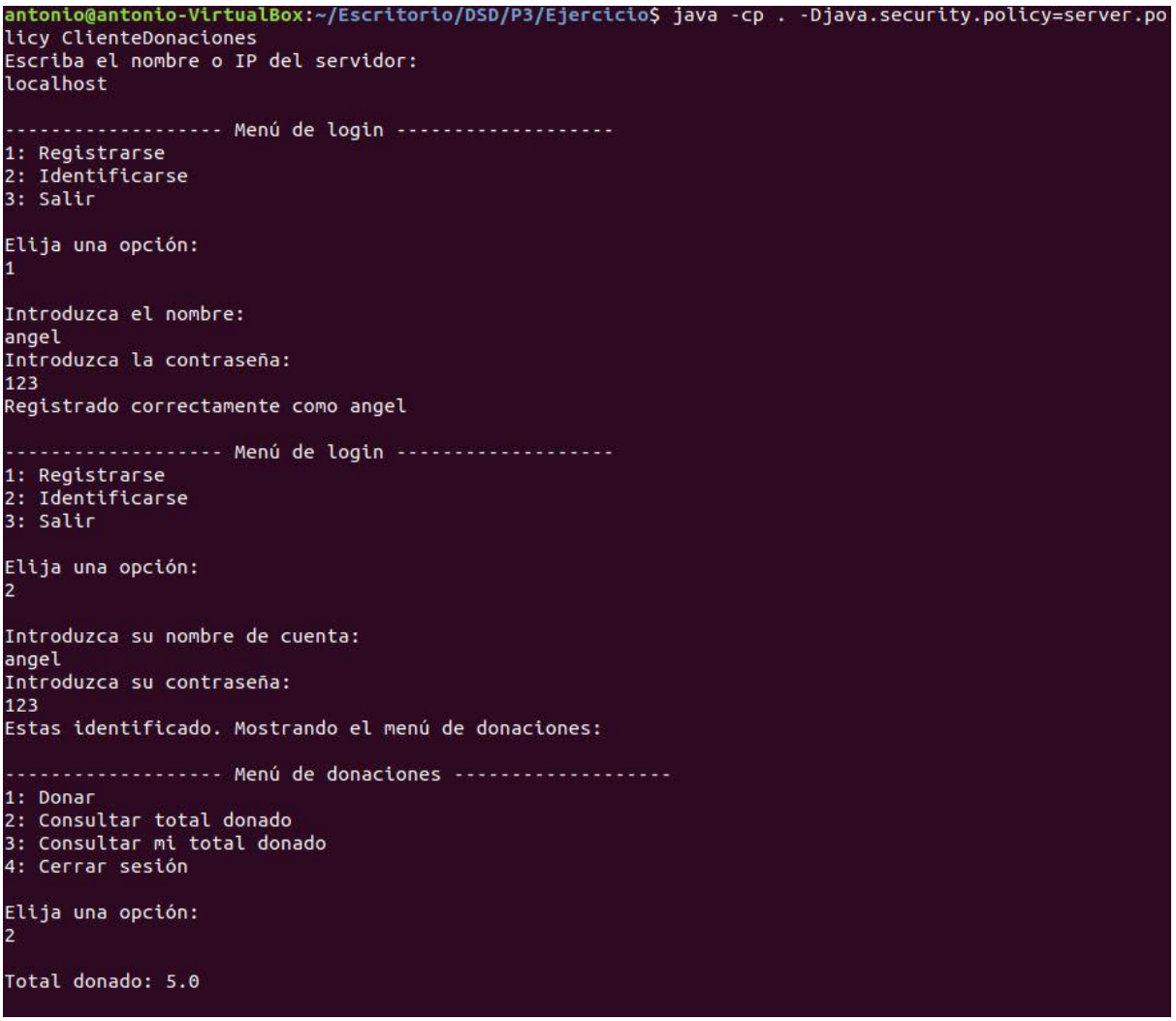
Nos logeamos con las credenciales correctas y nos muestra el menú de donaciones, probamos a consultar el total donado, pero como no se ha realizado ninguna donación no se puede consultar.

Consultamos también antes nuestro total donado, y correctamente muestra 0.0. Seguidamente realizamos una donación de 5, y después consultamos tanto el total donado como nuestro total donado, y en las 2 aparece la cantidad donada anteriormente. Finalmente cerramos sesión.



Ahora probamos con otro cliente nuevo. Nos registramos y nos logeamos. Ahora si podemos comentar que al principio cuando antonio se registró, se registró en el servidor 1, y cuando angel se ha registrado, como el servidor 2 no tenía ningún cliente aún se registró ahí.

Ahora probamos que se realizó una correcta donación del usuario Antonio comprobando el total donado. Podemos ver que efectivamente el resultado es 5, la donación que hizo el usuario Antonio.



Comprobamos también que el total de donaciones de Ángel es 0. Ahora realizamos una donación con Ángel. Comprobamos que la nueva donación se ha sumado a la total que ya había en el servidor, y que la donación del usuario Ángel es igual a la que ha donado.

```
------------------- Menú de donaciones -------------------
1: Donar
2: Consultar total donado
3: Consultar mi total donado
4: Cerrar sesión

Elija una opción:
3

Tu cantidad total de donaciones es: 0.0

------------------- Menú de donaciones -------------------
1: Donar
2: Consultar total donado
3: Consultar mi total donado
4: Cerrar sesión

Elija una opción:
1

Introduzca la cantidad a donar (>0):
123
Donación realizada.

------------------- Menú de donaciones -------------------
1: Donar
2: Consultar total donado
3: Consultar mi total donado
4: Cerrar sesión

Elija una opción:
2

Total donado: 128.0
```

```
------------------- Menú de donaciones -------------------
1: Donar
2: Consultar total donado
3: Consultar mi total donado
4: Cerrar sesión

Elija una opción:
3

Tu cantidad total de donaciones es: 123.0

------------------- Menú de donaciones -------------------
1: Donar
2: Consultar total donado
3: Consultar mi total donado
4: Cerrar sesión

Elija una opción:
4

Sesión cerrada.
```

A tener en cuenta:

Para lanzar los servidores y clientes , se ha hecho desde el directorio Ejercicio, donde están todos los fuentes y el archivo server.policy. Se ha usado las siguientes órdenes:

```
java -cp . -Djava.rmi.server.codebase=file:./
-Djava.rmi.server.hostname=localhost
-Djava.security.policy=server.policy Servidor1

java -cp . -Djava.rmi.server.codebase=file:./
-Djava.rmi.server.hostname=localhost
-Djava.security.policy=server.policy Servidor2
```

Para lanzar los clientes, los ejecutamos de la siguiente forma:

```
java -cp . -Djava.security.policy=server.policy
ClienteDonaciones
```

Ficheros:

Declaración de interfaces: I_donacion.java, I_donacion_replica.java
Implementación de interfaces: Donaciones.java, Donaciones2.java
Cliente: Cliente.java (Información de la clase Cliente con sus métodos)
Ejecutable Cliente: ClienteDonaciones.java
Ejecutable Servidores: Servidor1.java, Servidor2.java