

A dark blue vertical bar on the left side of the page. A blue arrow points to the right from the bar, containing the date.

6-5-2022

# Práctica 3: RMI

Lenguajes y Middlewares para  
Programación Distribuida

Several thin, curved lines in dark blue and light grey originate from the bottom left and sweep upwards and to the right.

Alberto Llamas González  
3º GII, IS

## **Parte 1: Implementación de los ejemplos**

En esta primera parte, hemos probado los ejemplos del guion.

### ***Ejemplo 1***

En este ejemplo, el cliente solicita al servidor la utilización del método **escribir\_mensaje**. Para ello, el cliente mira primero si se ha establecido algún tipo de seguridad en el cliente, si no hay seguridad se crea. Después mira el **rmiregistry** buscando el objeto remoto **Ejemplo\_I** para crear una instancia local. Para terminar, se llama a dicho método desde la instancia local del objeto.

En el programa servidor, **Ejemplo**, cuando recibe una petición del cliente, el método **escribir\_mensaje** mira el número del proceso que pasamos como argumento al cliente y si es 0 hace una espera de 5 segundos. Siempre imprime el número del proceso.

Veamos un ejemplo de compilación y ejecución. Para compilar, basta con ejecutar desde la carpeta del ejemplo:

```
//Iniciamos rmiregistry, el ligador de RMI
rmiregistry &
//Compilamos con javac
javac *.java
//Lanzamos el servidor en otra terminal o podemos usar &
java -cp . -Djava.rmi.server.codebase=file:./ -
Djava.rmi.server.hostname=localhost -
Djava.security.policy=server.policy Ejemplo
//Lanzamos el cliente
java -cp . -Djava.security.policy=server.policy Cliente_Ejemplo
localhost 0
```

**Resultado:**

```

albertollamas@linux:~/Escritorio/OSD/Practica3/Ejemplo1$ rmiregistry &
[1] 21957
albertollamas@linux:~/Escritorio/OSD/Practica3/Ejemplo1$ javac *.java
albertollamas@linux:~/Escritorio/OSD/Practica3/Ejemplo1$ java -cp . -Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy Ejemplo &
[2] 22082
albertollamas@linux:~/Escritorio/OSD/Practica3/Ejemplo1$ Ejemplo bound

albertollamas@linux:~/Escritorio/OSD/Practica3/Ejemplo1$ java -cp . -Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy Cliente_Ejemplo localhost 0
Buscando el objeto remoto
Invocando el objeto remoto
Recibida petición de proceso: 0
Empezamos a dormir
Terminamos de dormir

Hebra 0
albertollamas@linux:~/Escritorio/OSD/Practica3/Ejemplo1$ java -cp . -Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy Cliente_Ejemplo localhost 3
Buscando el objeto remoto
Invocando el objeto remoto
Recibida petición de proceso: 3

Hebra 3

```

**Ejemplo 2**

En este ejemplo en lugar de lanzar varios clientes, se crean varias tareas que realizan esa tarea de imprimir un mensaje. El programa pasa por argumento el host y el número de hilos a ejecutar y el cliente lanza los hilos al servidor. Si el número de hilo termina en 0 hay un delay.

Si ponemos **synchronized** en el método **escribir\_mensaje** vemos que la ejecución de los hilos es secuencial ya que se lanza el hilo y hasta que no acabe su ejecución no se sigue con el siguiente. Veamos las diferencias. Para compilar y ejecutar:

```

//Iniciamos rmiregistry, el ligador de RMI
rmiregistry &
//Compilamos con javac
javac *.java
//Lanzamos el servidor en otra terminal o podemos usar &
java -cp . -Djava.rmi.server.codebase=file:./ -
Djava.rmi.server.hostname=localhost -
Djava.security.policy=server.policy Ejemplo
//Lanzamos el cliente
java -cp . -Djava.security.policy=server.policy
Cliente_Ejemplo_Multi_Threaded localhost 10

```

### Resultado:

## **Con** synchronized en el método

[illegible]

Entra Hebra Cliente0	Empezamos a dormir
Terminamos de dormir	Salí Hebra Cliente0
Entra Hebra Cliente1	Salí Hebra Cliente1
Entra Hebra Cliente4	Salí Hebra Cliente4
Entra Hebra Cliente8	Salí Hebra Cliente8
Entra Hebra Cliente6	Salí Hebra Cliente6
Entra Hebra Cliente5	Salí Hebra Cliente5
Entra Hebra Cliente3	Salí Hebra Cliente3
Entra Hebra Cliente7	Salí Hebra Cliente7
Entra Hebra Cliente2	Salí Hebra Cliente2
Entra Hebra Cliente9	Salí Hebra Cliente9

### ***Sin* synchronized**

[illegible]

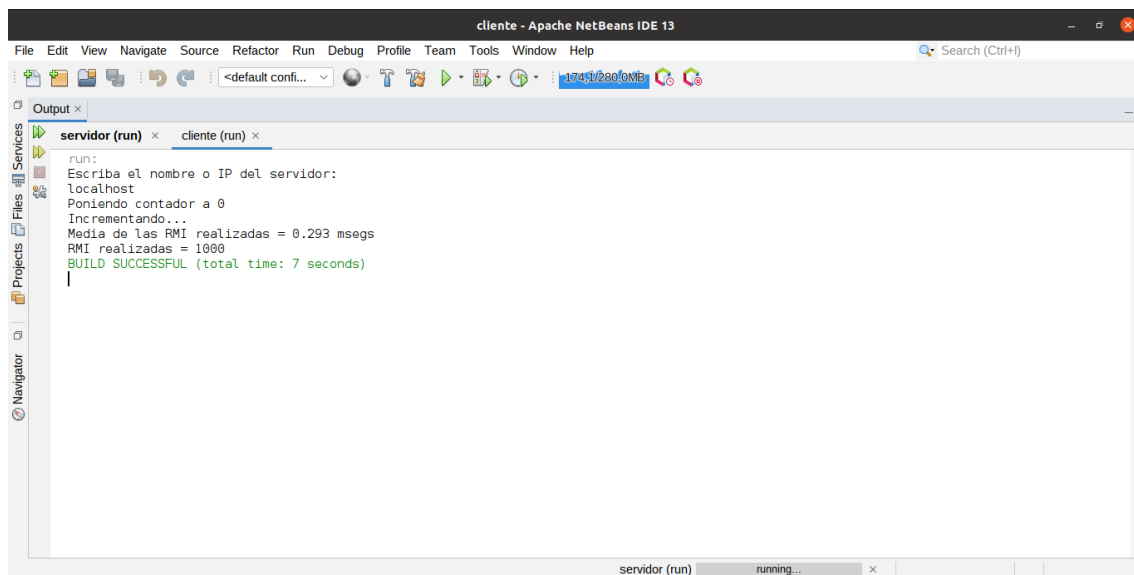
Entra Hebra Cliente3
Entra Hebra Cliente8
Sale Hebra Cliente8
Sale Hebra Cliente3
Entra Hebra Cliente0
Empiezas a dormir
Entra Hebra Cliente6
Sale Hebra Cliente6
Entra Hebra Cliente4
Sale Hebra Cliente4
Entra Hebra Cliente9
Sale Hebra Cliente9
Entra Hebra Cliente2
Entra Hebra Cliente1
Sale Hebra Cliente1
Sale Hebra Cliente2
Entra Hebra Cliente7
Sale Hebra Cliente7
Entra Hebra Cliente5
Sale Hebra Cliente5
Terminamos de dormir
Sale Hebra Cliente2

### Ejemplo 3

En este ejemplo, en el servidor hay una clase **contador** con su respectiva interfaz **icontador**. El servidor en este caso lo único que realiza es crear una instancia de contador y enlaza(bind) el mensaje. En el cliente ponemos el contador a 0 y lo incrementamos 1000 veces cronometrando el tiempo e imprimiendo al final por pantalla el tiempo empleado en realizar las operaciones y el número de ellas.

Para la ejecución de este ejercicio se ha probado la implementación en Netbeans por lo que la ejecución será en dicho programa. En este caso, era necesario crear las librerías para **icontador** y **contador** y añadirlas tanto al cliente como al servidor con el classpath respectivo (Para facilitar la ejecución desde cualquier ordenador, se ha utilizado classpath relativo, por lo que no es necesario modificar nada y debería funcionar en otro ordenador sin problemas). Sin embargo, la ejecución del ejercicio propuesto se realizará por comandos como en los ejemplos 1 y 2.

### Resultado:



```
run:
Escriba el nombre o IP del servidor:
localhost
Poniendo contador a 0
Incrementando...
Media de las RMI realizadas = 0.293 msecs
RMI realizadas = 1000
BUILD SUCCESSFUL (total time: 7 seconds)
```

## **Parte 2: Servidor replicado**

### ***Antes de empezar***

Esta práctica se ha realizado utilizando **openjdk 11.0.14.1** en el sistema operativo **Ubuntu**.

Tenemos dos formas de **ejecutar** el ejercicio:

- Script **ejecutar.sh**. El problema de utilizar este archivo, es que para matar luego los procesos es necesario hacer **lsof -i**, ver que PIDs son de los archivos ejecutados, normalmente contienen **java** y **rmiregistry** y matarlos con **kill**.
- Mediante comandos por la terminal:

```
//Iniciamos rmiregistry, el ligador de RMI
rmiregistry &

//Compilamos con javac
javac *.java

//Lanzamos las 3 réplicas del servidor en otra terminal o podemos
usar &
java -cp . -Djava.rmi.server.hostname=localhost -
Djava.security.policy=server.policy Servidor &

//Lanzamos el cliente
java -cp . -Djava.rmi.server.hostname=localhost -
Djava.security.policy=server.policy Cliente serverX
```

Donde X = 1 ó 2 ó 3

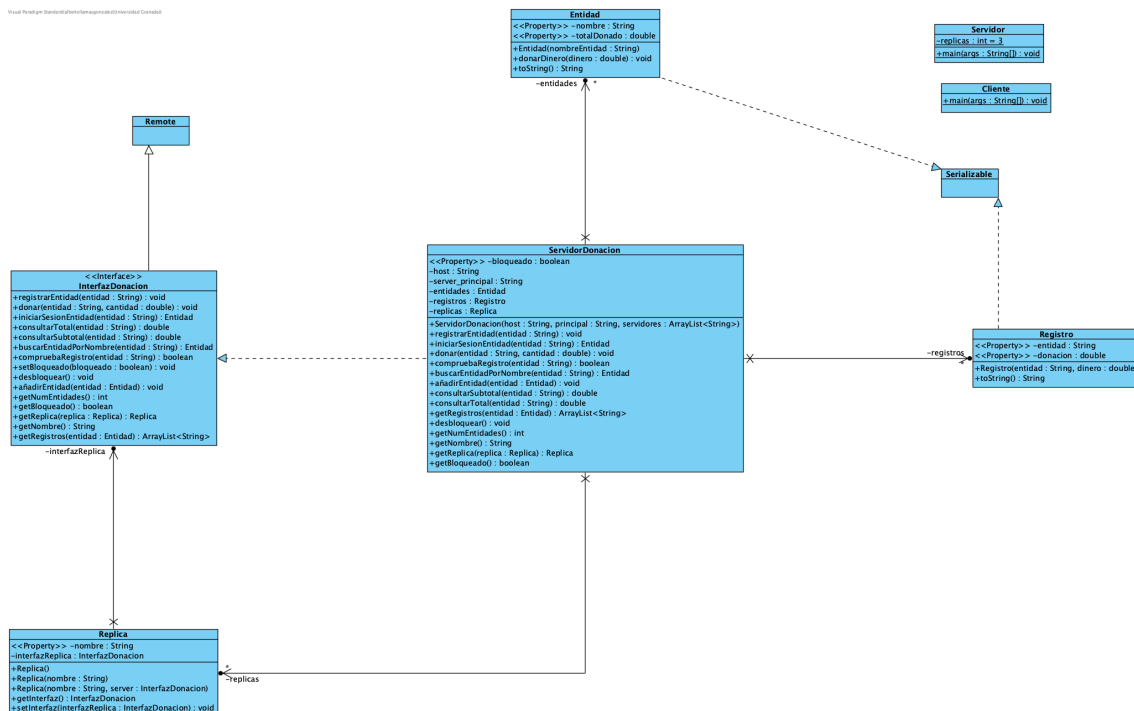
## Implementación del ejercicio

El objetivo del ejercicio era desarrollar en RMI un sistema cliente-servidor donde el servidor se encargaba de recibir donaciones de entidades (clientes) para una causa humanitaria. Debía tener 5 operaciones principales:

- Registrarse
- Iniciar sesión
- Donar a la causa
- Consultar el subtotal de la réplica
- Consultar el total (entre todas las réplicas)

Veamos el diagrama de clases del ejercicio resultante.

### Diagrama de clases:



Como podemos ver tenemos 6 clases principales (luego hablaremos de la funcionalidad extra, clase **Registro**). Hablaremos de ellas por orden de implementación.

- **InterfazDonacion:** se trata de la interfaz remota donde, como vemos, derivamos de la interfaz *java.rmi.Remote* y cada método lanza una excepción *java.rmi.RemoteException*.
- **ServidorDonacion:** implementación de la interfaz remota. Para implementarlo, se tuvo que añadir a **InterfazDonacion** métodos y clases auxiliares necesarias. Esta clase, al ser más extensa, tiene los métodos comentados en detalle para aclarar el uso de cada uno.
- **Replica:** clase con dos atributos que almacena la réplica, *nombre* (de la réplica), *interfazReplica* (interfaz para poder modificar el servidor de la réplica).
- **Entidad:** entidad la cual dona en el servidor.
- **Cliente:** se localiza el objeto remoto y se interactúa con el usuario desde esta clase.
- **Servidor:** creación de los servidores con implementación del gestor de seguridad.

Hablemos un poco de la implementación y, posteriormente, veamos un ejemplo de ejecución. Para implementar el servidor (clase Servidor) se ha seguido lo aprendido con los ejercicios (en especial el ejercicio 3), al igual que al implementar la comunicación del cliente con el servidor.

Para que los servidores cooperen entre sí, se ha intentado implementar de la mejor manera que se ha podido la técnica de actualización de datos. En algunos métodos se ha utilizado un poco de recursividad (que realmente no es tan recursivo ya que va llamando a las otras réplicas del servidor) para actualizar los datos a la hora de donar y cuando nos registramos, al buscar la entidad. Para garantizar el correcto funcionamiento y que no haya problemas de que más de un proceso entre en una sección crítica, se ha intentado garantizar la exclusión mutua



mediante una variable booleana (bloqueado) que bloqueaba los servidores donde NO se estaban realizando las operaciones.

El cliente se ejecutará hasta que el usuario desee salir. Al principio, nos ofrece 3 opciones; **registrar una entidad, iniciar sesión, salir**. Una vez iniciada la sesión, tenemos 5 opciones; **donar, cuánto se ha donado en ESE servidor, cuánto se ha donado en total, historial de donaciones (función extra), volver al menú principal**.

Como se ha comentado anteriormente, la explicación de dichos métodos se encuentra en el propio archivo **ServidorDonación**.

### Funciones extra:

Como función extra, se ha implementado algo sencillo, un registro de la gente que ha donado en el servidor. Para ello se ha creado otra clase **Registro** la cual tiene el nombre de la entidad y el dinero donado. En el servidor tenemos un ArrayList de Registro, con todos los registros realizados.

Otro extra añadido a última hora, ha sido el añadir una contraseña a la hora de iniciar sesión / registrarse. Si la contraseña es incorrecta, se pide la contraseña hasta que el usuario introduzca la correcta.

### Ejemplo de ejecución:

#### *Compilación y ejecución del cliente y servidor*

```
albertollamas@cvi041137:~/Escritorio/DSD/Practica3/Ejercicio$ javac *.java
albertollamas@cvi041137:~/Escritorio/DSD/Practica3/Ejercicio$ rmiregistry &
[1] 116610
albertollamas@cvi041137:~/Escritorio/DSD/Practica3/Ejercicio$ java -cp . -Djava.rmi.server.hostname=localhost -Djava.security.
policy=server.policy Servidor &
[2] 116676
albertollamas@cvi041137:~/Escritorio/DSD/Practica3/Ejercicio$ -> Server1 listo
-> Server2 listo
-> Server3 listo

albertollamas@cvi041137:~/Escritorio/DSD/Practica3/Ejercicio$ java -cp . -Djava.rmi.server.hostname=localhost -Djava.security.
policy=server.policy Cliente server3
Estas en el servidor: server3

Bienvenida/o! Elige una opción
1 -> Registrar una entidad
2 -> Iniciar sesión (ya debes estar registrado)
-1 -> Salir
```

## Registro

```
Bienvenida/o! Elige una opción
1 -> Registrar una entidad
2 -> Iniciar sesión (ya debes estar registrado)
-1 -> Salir
1
Dime el nombre de la entidad
entidad1
Creada la entidad: entidad1
Se ha registrado la entidad.
```

## Inicio de sesión

```
Bienvenida/o! Elige una opción
1 -> Registrar una entidad
2 -> Iniciar sesión (ya debes estar registrado)
-1 -> Salir
2
Dime el nombre de la entidad
entidad1
Iniciando sesión con entidad1
Ha iniciado sesion correctamente

Iniciada sesion como entidad1. Elige una opción
1 -> Donar
2 -> Ver cuánto se ha donado en ESTE servidor: server3
3 -> Ver cuánto se ha donado en total (3 servidores)
4 -> Ver el historial de donaciones en ESTE servidor: server3
-1 -> Volver a menu principal
```

## Donación

```
Iniciada sesion como entidad1. Elige una opción
1 -> Donar
2 -> Ver cuánto se ha donado en ESTE servidor: server3
3 -> Ver cuánto se ha donado en total (3 servidores)
4 -> Ver el historial de donaciones en ESTE servidor: server3
-1 -> Volver a menu principal
1
Introduzca la cantidad a donar:
10
entidad1 ha donado 10.0€ a la causa
Iniciada sesion como entidad1. Elige una opción
1 -> Donar
2 -> Ver cuánto se ha donado en ESTE servidor: server3
3 -> Ver cuánto se ha donado en total (3 servidores)
4 -> Ver el historial de donaciones en ESTE servidor: server3
-1 -> Volver a menu principal
```

## Ver donado en servidor actual

```
Iniciada sesion como entidad1. Elige una opción
1 -> Donar
2 -> Ver cuánto se ha donado en ESTE servidor: server3
3 -> Ver cuánto se ha donado en total (3 servidores)
4 -> Ver el historial de donaciones en ESTE servidor: server3
-1 -> Volver a menu principal
2
Nombre : entidad1 Total donado : 10.0€.
Entidad entidad1, Dinero donado: 10.0
La entidad entidad1 ha donado 10.0€ a la causa en este servidor.
```

## Historial de donaciones

```
Iniciada sesion como entidad1. Elige una opción
1 -> Donar
2 -> Ver cuánto se ha donado en ESTE servidor: server3
3 -> Ver cuánto se ha donado en total (3 servidores)
4 -> Ver el historial de donaciones en ESTE servidor: server3
-1 -> Volver a menu principal
4
Host      Nombre servidor Cantidad
localhost server3 10.0
```

## Inicio de sesión en otro servidor y donación

```
Estas en el servidor: server1
Bienvenida/o! Elige una opción
1 -> Registrar una entidad
2 -> Iniciar sesión (ya debes estar registrado)
-1 -> Salir
2
Dime el nombre de la entidad
entidad1
Iniciando sesión con entidad1
Ha iniciado sesion correctamente

Iniciada sesion como entidad1. Elige una opción
1 -> Donar
2 -> Ver cuánto se ha donado en ESTE servidor: server1
3 -> Ver cuánto se ha donado en total (3 servidores)
4 -> Ver el historial de donaciones en ESTE servidor: server1
-1 -> Volver a menu principal
1
Introduzca la cantidad a donar:
5
entidad1 ha donado 5.0€ a la causa
```

## Donado en total entre los 3 servidores

```
Iniciada sesion como entidad1. Elige una opción
1 -> Donar
2 -> Ver cuánto se ha donado en ESTE servidor: server1
3 -> Ver cuánto se ha donado en total (3 servidores)
4 -> Ver el historial de donaciones en ESTE servidor: server1
-1 -> Volver a menu principal
3
La entidad entidad1 ha donado 15.0€ a la causa.

Iniciada sesion como entidad1. Elige una opción
1 -> Donar
2 -> Ver cuánto se ha donado en ESTE servidor: server1
3 -> Ver cuánto se ha donado en total (3 servidores)
4 -> Ver el historial de donaciones en ESTE servidor: server1
-1 -> Volver a menu principal
```

## Inicio de sesión con contraseña (funcionalidad extra)

```
albertollamas@cvl041137:~/Escritorio/OSD/Practica3/Ejercicio$ java -cp . -Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy Cliente server1
Estas en el servidor: server1
Bienvenida/o! Elige una opción
1 -> Registrar una entidad
2 -> Iniciar sesión (ya debes estar registrado)
-1 -> Salir
1
Introduce el nombre de la entidad
entidad1
Introduce una contraseña
hola
Creada la entidad: entidad1
Se ha registrado la entidad.
```

```
Bienvenida/o! Elige una opción
1 -> Registrar una entidad
2 -> Iniciar sesión (ya debes estar registrado)
-1 -> Salir
2
Introduce el nombre de la entidad
entidad1
Introduce la contraseña
e
Iniciando sesión con entidad1
Contraseña incorrecta.

Introduce la contraseña
hola
Iniciando sesión con entidad1
Ha iniciado sesion correctamente
```

### **Problemas encontrados durante la realización de la práctica**

Aparte de los típicos errores de compilación que podemos obtener al realizar un programa, tuve dos problemas principales: puerto 1099 ocupado, y necesidad de que ciertas clases fuesen "serializables".

```
Dime el nombre de la entidad
entidad1
Iniciando sesión con entidad1
java.rmi.UnmarshalException: error unmarshalling return; nested exception is:
    java.io.WriteAbortedException: writing aborted; java.io.NotSerializableException: Entidad
```

*(Error Serializable)*

El error del puerto, lo solucioné cambiando el puerto del servidor, ya que, aunque no hubiese nada en el puerto 1099, me seguía diciendo que estaba ocupado. Para arreglarlo use el puerto 1100. El segundo error se arreglaba fácilmente ya que simplemente había que heredar de la clase de java **Serializable**. No sabía por qué y decidí investigar. Esto se debe a que, si heredamos de esta clase, podremos enviar copias exactas de los objetos (como el paso por valor en C/C++) y los cambios realizados en otro ordenador, no se verían reflejados en el objeto original. Esto se utiliza cuando el objeto es relativamente pequeño y se van a invocar varias veces los métodos del objeto sin que nos interese modificar el estado interno de éste.

### **Bibliografía**

- <https://www.simplilearn.com/tutorials/java-tutorial/serialization-in-java#:~:text=Serialization%20in%20Java%20is%20the,then%20de%2Dserialize%20it%20there.>
- <https://stackoverflow.com/questions/4548816/when-should-we-implement-serializable-interface>