

Práctica 3

Invocación remota (RMI)

Luis Miguel Guirado Bautista

Curso 2024-2025

Universidad de Granada

Ejemplos	2
Ejemplo 1. Cliente-servidor simple	2
Ejemplo 2. Cliente-servidor multihebra	3
Ejemplo 3. Contador	4
Servidor replicado	5
Características adicionales	5
Servidor	5
Métodos auxiliares (propias de Servidor)	5
Interfaz servidor-servidor: IServidor	6
Cliente	6
Interfaz cliente-servidor: ICliente	6
Funcionamiento	7
Scripts para iniciar el cliente y el servidor	7
Preparar el sistema	7
Capturas de pantalla	7

Ejemplos

Es MUY IMPORTANTE que se utilice una versión antigua de Java (p.e Java 8) debido a que el gestor de seguridad es una característica que se ha marcado como desuso y ha dejado de utilizarse en versiones modernas, por lo que al intentar ejecutar estos programas con versiones más modernas de Java dará lugar a que salte una `UnsupportedOperationException`. Se puede utilizar la herramienta `update-java-alternatives` para cambiar la versión de Java si se tiene más de una instalada en el sistema.

Ejemplo 1. Cliente-servidor simple

```
# servidor.sh
java -cp . -Djava.rmi.server.codebase=file:./ \
      -Djava.rmi.server.hostname=localhost \
      -Djava.security.policy=server.policy Ejemplo
```

```
# cliente.sh
java -cp . -Djava.security.policy=server.policy \
      Cliente_Ejemplo localhost $1
```

```
luismg@kubuntu:~/Escritorio/OSD/2024-2025/P3/examples/ejemplo1$ source servidor.sh
Recibida petición de proceso 0
Durmiendo
Fin escribir en 0
Recibida petición de proceso 5
Fin escribir en 5
█
```

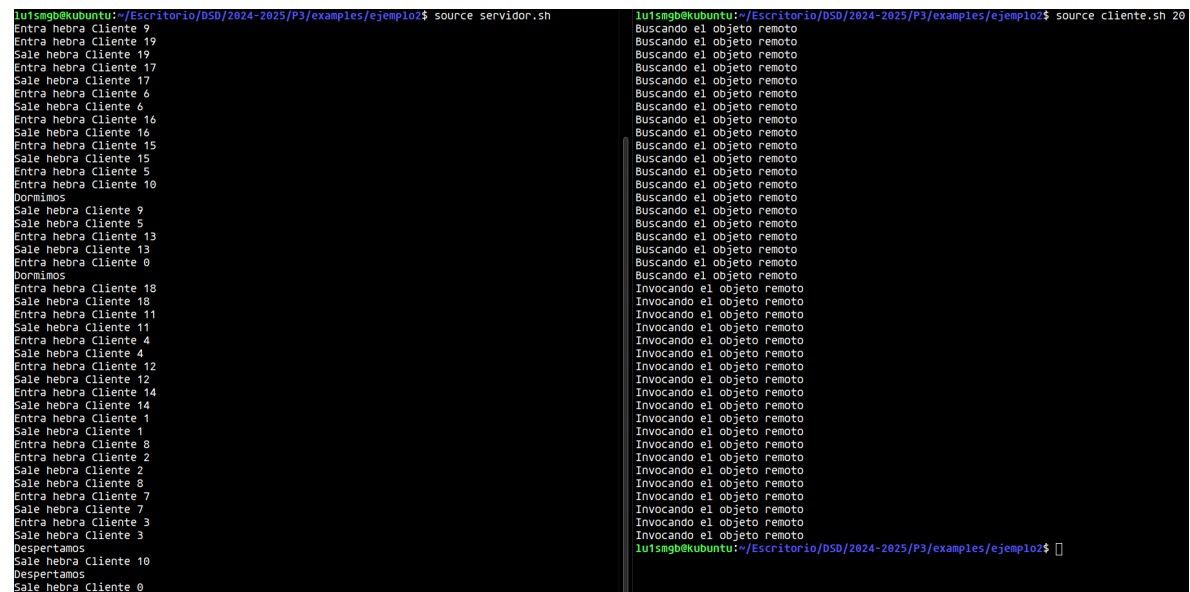
```
luismg@kubuntu:~/Escritorio/OSD/2024-2025/P3/examples/ejemplo1$ source cliente.sh 0
Buscando el objeto remoto
Invocando objeto remoto
luismg@kubuntu:~/Escritorio/OSD/2024-2025/P3/examples/ejemplo1$ source cliente.sh 5
Buscando el objeto remoto
Invocando objeto remoto
luismg@kubuntu:~/Escritorio/OSD/2024-2025/P3/examples/ejemplo1$ █
```

1. El servidor es iniciado
2. El cliente busca en el registro el objeto con el identificador “MiObjeto”
3. Una vez localizado el objeto, el cliente invoca al método `escribir` de ese objeto con el identificador numérico que le hayamos pasado a la hora de iniciar el cliente
4. El servidor recibe la invocación y ejecuta el método `escribir`. Si el identificador es 0, el servidor dormirá durante 2 segundos

Ejemplo 2. Cliente-servidor multihebra

```
# servidor.sh
java -cp . -Djava.rmi.server.codebase=file:./ \
      -Djava.rmi.server.hostname=localhost \
      -Djava.security.policy=server.policy Ejemplo
```

```
# cliente.sh
java -cp . -Djava.security.policy=server.policy \
      Cliente_Ejemplo localhost $1
```



The screenshot shows two terminal windows side-by-side. The left window is titled 'luisngb@kubuntu:~/Escritorio/DSD/2024-2025/P3/examples/ejemplo2\$ source servidor.sh' and displays the output of the server script. It shows 20 clients entering and leaving, with a sleep of 5 seconds after every 10 clients. The right window is titled 'luisngb@kubuntu:~/Escritorio/DSD/2024-2025/P3/examples/ejemplo2\$ source cliente.sh 20' and displays the output of the client script. It shows 20 threads searching for a remote object and then invoking the 'escribir' method on the remote object. The output of the client script shows that all 20 threads successfully invoked the method.

1. El servidor es iniciado
2. Iniciamos 20 clientes a partir del programa cliente, cada cliente corresponde a una hebra
3. Cada hebra/cliente busca el objeto remoto en el registro con identificador "MiObjeto" e invoca al metodo escribir con el mensaje "Cliente i " donde i es el número de hebra/cliente, que puede ir desde 0 hasta $n - 1$, siendo n el número de hebras/clientes
4. El servidor recibe cada una de las invocaciones de los clientes y ejecuta el metodo escribir por cada una. Si el mensaje termina en 0 (el número de hebra es 0 o multiplo de 10), entonces empezara a dormir durante 5 segundos

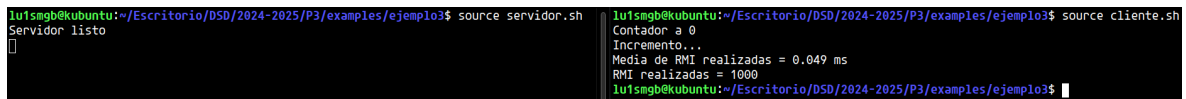
Observaciones:

- Se han probado distintos valores de n (20, 50 y 100) y en todos estos casos, todas las hebras que preveen que deberian llamar a las versiones que duermen de escribir, en total, tardan lo mismo en despertar. Es decir, el servidor tarda el tiempo bloqueante en despertar a las hebras bloqueantes a la vez.
- Cuando una hebra duerme, siguen entrando hebras en el servidor. Esto cambia al emplear synchronized en la cabecera de la implementación del metodo, forzando a que las hebras tengan que esperar cuando una está ocupada.

Ejemplo 3. Contador

```
# servidor.sh
java -cp . -Djava.rmi.server.codebase=file:./ \
      -Djava.rmi.server.hostname=localhost \
      -Djava.security.policy=server.policy Servidor
```

```
# cliente.sh
java -cp . -Djava.security.policy=server.policy Cliente
```



```
luismgb@kubuntu:~/Escritorio/DSO/2024-2025/P3/examples/ejemplo3$ source servidor.sh
Servidor listo

luismgb@kubuntu:~/Escritorio/DSO/2024-2025/P3/examples/ejemplo3$ source cliente.sh
Contador a 0
Incremento...
Media de RMI realizadas = 0.049 ms
RMI realizadas = 1000
luismgb@kubuntu:~/Escritorio/DSO/2024-2025/P3/examples/ejemplo3$
```

1. Lanzamos el servidor
2. Lanzamos el cliente, que busca el objeto remoto. Pone el contador a cero y lo incrementa 1000 veces, todo ello con invocaciones remotas a través de un stub, es el servidor el que realmente modifica el contador
3. El cliente calcula y muestra estadísticas en base a la ejecución realizada como la media de llamadas realizadas por segundo y las llamadas realizadas totales en función del valor final del contador (esto último por invocación remota)

Observaciones:

- El programa debe ejecutarse desactivando el `rmiregistry`. El propio servidor actúa como registro de objetos remotos. En caso contrario dará error, ya que estamos intentando crear un registro de objetos en un puerto que ya está en uso.

Servidor replicado

El ejercicio consiste en implementar un servidor replicado para registrar donaciones de distintas entidades (clientes).

Características adicionales

- Se ha implementado el ejercicio de manera que el programa servidor inicie N réplicas de manera multihebrada (1 réplica por hebra)
- La lista de donantes también muestra el subtotal donado por cada una de las entidades
- Se ha añadido, además del registro de entidades, un registro de transacciones que el usuario puede consultar (solo se mostraran sus propias transacciones). Cada transacción guarda la entidad donante, la fecha y hora del momento en el que se realiza y la cantidad

Servidor

El servidor inicializa N servidores de manera multihebrada. Durante la inicialización, el programa maestro establece los vecinos o servidores hermanos en cada réplica, para que puedan comunicarse entre sí. Los vecinos sólo pueden comunicarse mediante su interfaz servidor-servidor definida en IServidor.

Cuando un servidor necesite delegar una operacion (por ejemplo, entidad A solicita donar en servidor 1 pero está registrado en servidor 2, de modo que la donacion efectiva tiene lugar en el servidor 2), lo que hará será intentar localizar el servidor donde se encuentre registrada esta entidad mediante búsqueda.

Métodos auxiliares (propias de Servidor)

- `private void aniadirVecino(IServidor vecino)`
 - Añade un servidor vecino al conjunto de vecinos de un servidor
- `private Map<IServidor, Integer> obtenerLongitudesRegistros()`
 - Genera un mapa como clave el servidor y como valor la longitud del registro, o lo que es igual, el número de entidades registradas
- `private IServidor estimarMejorServidor()`
 - Estima el mejor servidor donde realizar el registro de una entidad, es decir, devuelve el servidor con menos entidades registradas
- `private Map<String, Integer> obtenerRegistroGlobal()`
 - Genera un registro global a partir de los registros de cada uno de los servidores
- `private boolean haDonado(String idCliente)`
 - Comprueba si una entidad ha realizado una donacion en el sistema
- `private IServidor localizarEntidad(String idCliente)`
 - Intenta localizar una entidad en el conjunto de servidores y devuelve el servidor en el que se encuentra o null en caso contrario
- `private Integer obtenerDonadoPorEntidad(String idCliente)`
 - Obtiene la cantidad total que ha donado una entidad

Interfaz servidor-servidor: IServidor

- `public Map<String, Integer> obtenerRegistro()`
 - Obtiene el registro de un servidor
- `public Set<Transaccion> obtenerTransacciones()`
 - Obtiene el conjunto local de transacciones de un servidor
- `private String obtenerId()`
 - Obtiene el identificador del servidor
- `public void grabarEntidad(String idCliente)`
 - Registra la entidad con nombre `idCliente` en el servidor
- `public void grabarDeposito(String idCliente)`
 - Registra una donacion en el servidor junto con su correspondiente `transaccion`

Cliente

Cada cliente puede conectarse al servidor que el decida, proporcionando el número de servidor desde el 1 hasta N, siendo N el número de servidores totales.

Los clientes se comunican con los servidores exclusivamente por `ICliente`, la interfaz cliente-servidor que implementan los servidores para tal fin.

El cliente tendrá que proporcionar el nombre de la entidad cada vez que entre al sistema y realizar el registro la primera vez que acceda a este para poder realizar donaciones.

Una vez que realice una donación, podrá acceder al resto de funcionalidades, como consultar el total donado por todas las entidades, la lista de entidades junto a lo que han donado y su historial personal de transacciones.

Interfaz cliente-servidor: ICliente

- `public boolean registrar(String idCliente)`
 - Realiza el registro de una entidad en el sistema
- `public boolean depositar(String idCliente, int cantidad)`
 - Realiza el depósito especificado en `cantidad` como `idCliente`
- `public boolean registrar(String idCliente)`
 - Obtiene el total donado por todas las entidades de todo el sistema
- `public Map<String, Integer> obtenerDonantes()`
 - Obtiene la lista de donantes de todo el sistema, junto con las cantidades totales que ha donado cada entidad
- `public boolean obtenerHistorial(String idCliente)`
 - Obtiene el historial de transacciones de la entidad `idCliente`

Funcionamiento

Scripts para iniciar el cliente y el servidor

```
# servidor.sh (para 4 replicas)
java -cp . -Djava.rmi.server.codebase=file:./ \
      -Djava.rmi.server.hostname=localhost \
      -Djava.security.policy=server.policy \
      Servidor localhost 4

# cliente.sh
java -cp . -Djava.security.policy=server.policy Cliente localhost
```

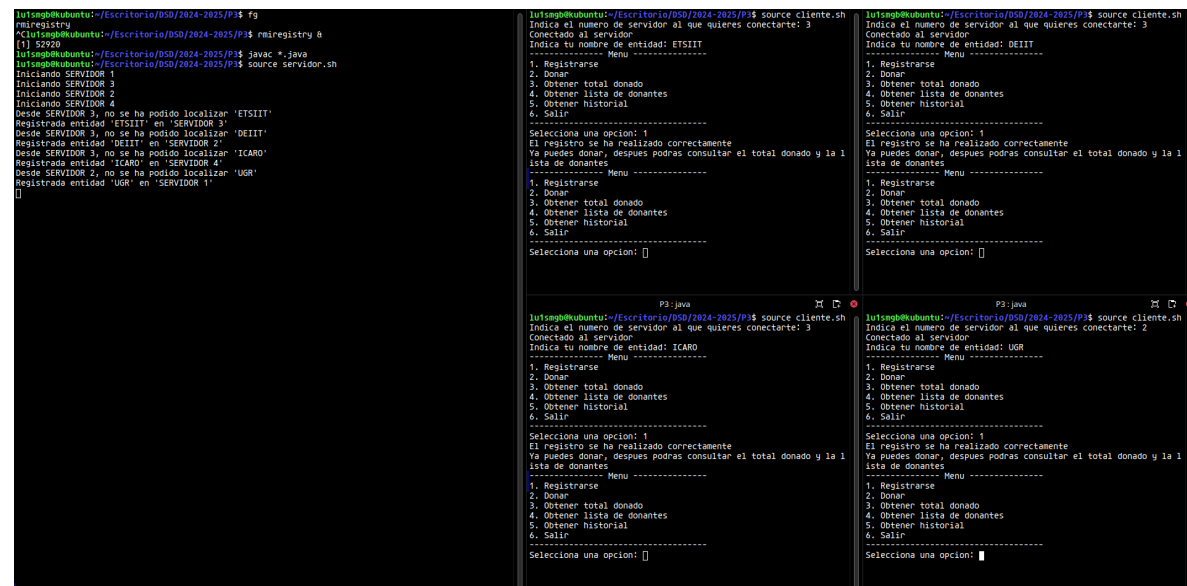
Preparar el sistema

```
# En el directorio donde se encuentran los ficheros java y class
$ rmiregistry &
$ javac *.java

# Dividimos la vista de terminal tantas veces como clientes necesitemos
$ source servidor.sh

# Ahora en cada vista de cliente
$ source cliente.sh
```

Capturas de pantalla



```
luis@ubuntu:~/Escritorio/DSD/2024-2025/P3$ fg
rmiregistry
^C
luis@ubuntu:~/Escritorio/DSD/2024-2025/P3$ rmiregistry &
[1] 5920
luis@ubuntu:~/Escritorio/DSD/2024-2025/P3$ javac *.java
luis@ubuntu:~/Escritorio/DSD/2024-2025/P3$ source servidor.sh
Iniciando SERVIDOR 1
Iniciando SERVIDOR 2
Iniciando SERVIDOR 3
Iniciando SERVIDOR 4
Desde SERVIDOR 3, no se ha podido localizar 'ETSII'
Registrada entidad 'ETSII' en 'SERVIDOR 3'
Desde SERVIDOR 3, no se ha podido localizar 'DEIIT'
Registrada entidad 'DEIIT' en 'SERVIDOR 2'
Desde SERVIDOR 3, no se ha podido localizar 'ICARO'
Registrada entidad 'ICARO' en 'SERVIDOR 4'
Desde SERVIDOR 2, no se ha podido localizar 'UGR'
Registrada entidad 'UGR' en 'SERVIDOR 1'
[]

luis@ubuntu:~/Escritorio/DSD/2024-2025/P3$ source cliente.sh
Indica el numero de servidor al que quieres conectarte: 3
Conectado al servidor
Indica tu nombre de entidad: ETSII
----- Menu -----
1. Registrarse
2. Donar
3. Obtener total donado
4. Obtener lista de donantes
5. Obtener historial
6. Salir
-----
Selecciona una opcion: 1
El registro se ha realizado correctamente
Ya puedes donar, despues podras consultar el total donado y la 1
ista de donantes
----- Menu -----
1. Registrarse
2. Donar
3. Obtener total donado
4. Obtener lista de donantes
5. Obtener historial
6. Salir
-----
Selecciona una opcion: []

luis@ubuntu:~/Escritorio/DSD/2024-2025/P3$ source cliente.sh
Indica el numero de servidor al que quieres conectarte: 3
Conectado al servidor
Indica tu nombre de entidad: ICARO
----- Menu -----
1. Registrarse
2. Donar
3. Obtener total donado
4. Obtener lista de donantes
5. Obtener historial
6. Salir
-----
Selecciona una opcion: 1
El registro se ha realizado correctamente
Ya puedes donar, despues podras consultar el total donado y la 1
ista de donantes
----- Menu -----
1. Registrarse
2. Donar
3. Obtener total donado
4. Obtener lista de donantes
5. Obtener historial
6. Salir
-----
Selecciona una opcion: []

luis@ubuntu:~/Escritorio/DSD/2024-2025/P3$ source cliente.sh
Indica el numero de servidor al que quieres conectarte: 2
Conectado al servidor
Indica tu nombre de entidad: UGR
----- Menu -----
1. Registrarse
2. Donar
3. Obtener total donado
4. Obtener lista de donantes
5. Obtener historial
6. Salir
-----
Selecciona una opcion: 1
El registro se ha realizado correctamente
Ya puedes donar, despues podras consultar el total donado y la 1
ista de donantes
----- Menu -----
1. Registrarse
2. Donar
3. Obtener total donado
4. Obtener lista de donantes
5. Obtener historial
6. Salir
-----
Selecciona una opcion: []
```

Preparando un servidor de 4 réplicas en el que se registran 4 entidades diferentes, 3 de ellas desde un mismo servidor, pero quedando registradas realmente cada una en un servidor diferente

Las entidades realizan donaciones, y una de ellas consulta el total donado y la lista de donantes

Las entidades consultan su historial de transacciones, ordenadas de la más antigua a la más reciente