

Práctica 3 - Ejemplos Java RMI

Desarrollo de Sistemas Distribuidos

Jose Luis Gallego Peña - DSD2 Todos los ejemplos han sido escritos y compilados con IntelliJ IDEA.

Para poner los ejemplos en funcionamiento, se compilan tanto el cliente como el servidor y se lanzan cada uno de forma separada.

1. Ejemplo 1

Tenemos el fichero **server.policy** en el que indicamos lo siguiente:

```
grant {  
    permission java.security.AllPermission;  
};
```

Lo que nos permite ejecutar el cliente y el servidor con todos los permisos. Si no, nos daría un error de excepción. Para usar este fichero usamos la siguiente opción al compilar:

```
-Djava.security.manager  
-Djava.security.policy=$Sourcepath$/server.policy
```

Interfaz Remota

Se importa la clase Remote y UnicastRemoteObject de Java, que son el mecanismo que tiene Java RMI para poder definir objetos remotos.

Example_I es la interfaz remota, que simplemente declara los métodos. Extiende a la clase Remote.

Example define esos métodos para el objeto remoto. Implementa la interfaz anterior.

La interfaz tiene un sólo método: **writeMessage**, que simplemente muestra un mensaje por pantalla (y si el thread es el 0 además duerme 5 segundos).

Cliente

Tanto en el cliente como en el servidor se implementa un gestor de seguridad que evita intrusiones de código maligno:

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

El cliente pide por pantalla el host (que será localhost en nuestro caso) y el PID del thread que se va a lanzar.

Para poder obtener los objetos remotos del servidor es necesario obtener el ligador rmiregistry, que es creado por el servidor, proporcionando el host anteriormente conseguido y el puerto en el que escuchará, que es el 1099.

```
Registry registry = LocateRegistry.getRegistry(host, 1099);
```

Los clientes RMI interactúan con los objetos remotos por medio de interfaces, es por esto que al objeto que llama a los métodos se le hace un casting con la interfaz:

```
Example_I localInstance = (Example_I) registry.lookup("example");
```

Lo que ha hecho ha sido buscar el objeto remoto con nombre “example” que ya tenía que haber definido el servidor.

Las interfaces sólo declaran y no definen los métodos, el cliente no conoce el tipo de los objetos remotos, realmente este objeto está ligado a una clase stub que se crea automáticamente.

Con este objeto se llama al método declarado en la interfaz, en el cual el valor se pasa por copia y no por referencia. Este valor en concreto es el identificador del thread, que es un número que elige el cliente:

```
localInstance.writeMessage(processId);
```

RMI localiza el objeto remoto para poder ejecutarlo. Y con esto ya estaría el cliente.

Servidor

El servidor es el encargado de lanzar el servicio de ligadura RMI (rmiregistry) para poder tener correctamente los tipos de los objetos remotos. Para ello, desde Java, hacemos lo siguiente:

```
Registry registry = LocateRegistry.createRegistry(1099);
```

Lo que hace es crear un rmiregistry que escuche nombres simples en el puerto 1099.

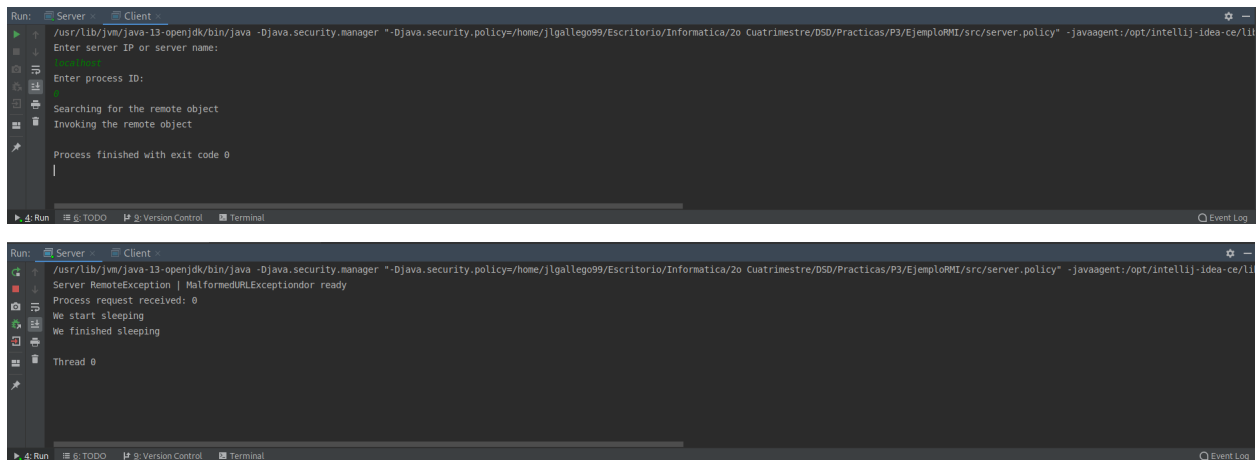
Para declarar uno de esos nombres simples, y por tanto lanzar el objeto remoto, se asocia un nombre a una clase que implementa una interfaz remota:

```
Example example = new Example();  
Naming.rebind("example", example); // Nombre, Stub
```

Lo cual hace que el servidor tenga esa interfaz disponible para servir a los clientes y le ha puesto el nombre “example”, es decir, se registra en el ligador y con esto el servidor ya estaría disponible.

Ejemplo de ejecución

Se lanza primero el servidor y luego se lanza el cliente, especificando el host (localhost o 127.0.0.1) y el id del thread de ese cliente. El servidor mostrará que ha recibido la petición y puesto que el id es 0, esperará 5 segundos. Pasados esos 5 segundos, simplemente mostrará un mensaje por pantalla y el cliente se cerrará.



2. Ejemplo 2: Multihebra

Este ejemplo implementa lo definido en el ejemplo 1 pero esta vez creamos varias hebras en lugar de lanzar varios clientes y lo que se pasa en el método no es el id del proceso si no un string que indica el nombre del thread.

Tiene las siguientes particularidades en el cliente:

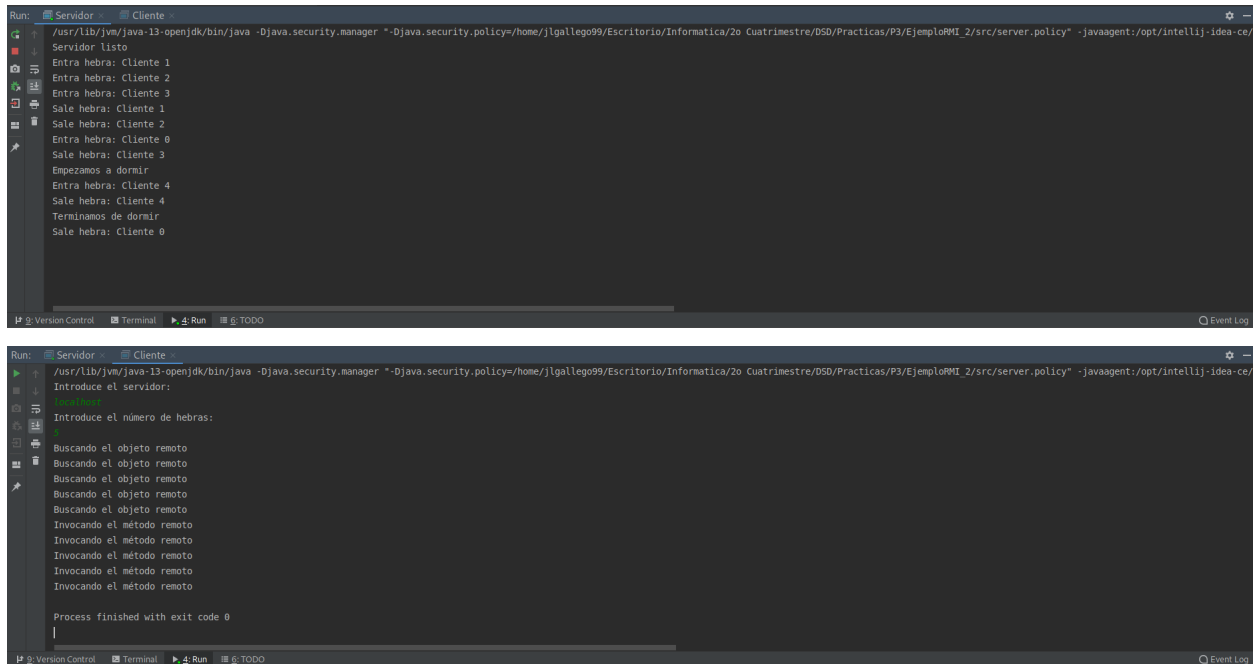
Ahora la clase Cliente implementa la interfaz Runnable para poder ejecutar las hebras. Por tanto define un método **run()**, que ejecutarán todas las hebras, y que hace lo que hacía antes un solo cliente: conseguir el rmiregistry, buscar el objeto remoto y ejecutar el método.

En el main se pasa por teclado el nombre del servidor y n hebras, para crear n clientes que reciban como parámetro el nombre de ese servidor al que tienen que conectarse.

Se crea un array de clientes y otro de hebras, y se van inicializando cada uno de los clientes y la hebra asociada a ese cliente, poniéndole a cada uno su nombre asociado a un número.

Las hebras cuyo nombre acaban en 0, al igual que antes, dormirán durante 5 segundos.

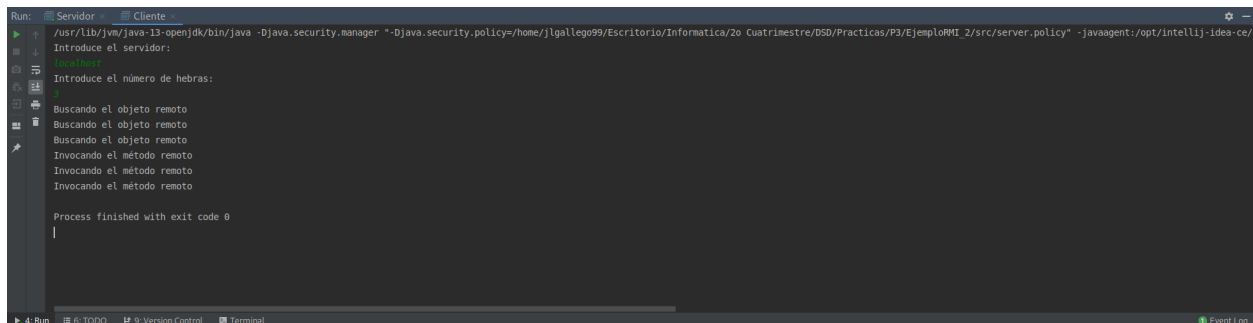
Como antes, se lanza primero el servidor y luego el cliente, especificando el servidor y el número de hebras. Se puede ver por pantalla como concurrentemente se van ejecutando las hebras, y que no tienen por que ir necesariamente en el orden en el que se han creado:



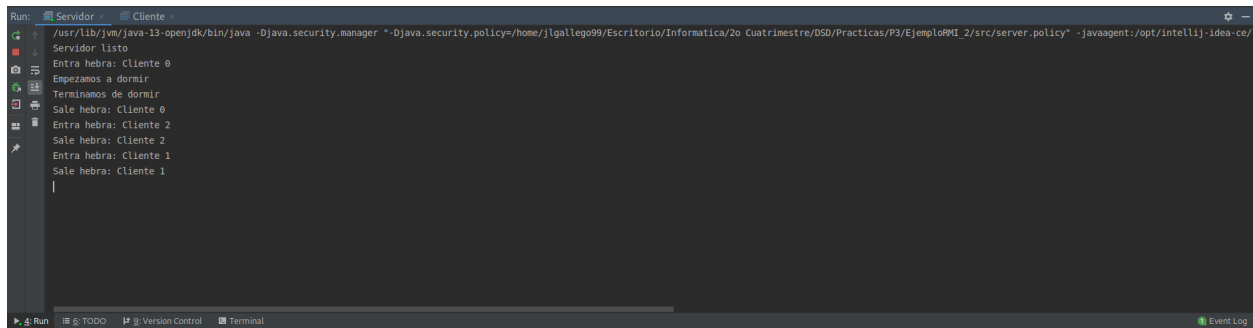
```
Run: Servidor Cliente
/usr/lib/jvm/java-13-openjdk/bin/java -Djava.security.manager *-Djava.security.policy=/home/jlgallego99/Escritorio/Informatica/2o Cuatrimestre/DSD/Practicas/P3/EjemploRMI_2/src/server.policy" -javaagent:/opt/intellij-idea-ce/
Servidor listo
Entra hebra: Cliente 1
Entra hebra: Cliente 2
Entra hebra: Cliente 3
Sale hebra: Cliente 1
Sale hebra: Cliente 2
Entra hebra: Cliente 0
Sale hebra: Cliente 3
Empezamos a dormir
Entra hebra: Cliente 4
Sale hebra: Cliente 4
Terminamos de dormir
Sale hebra: Cliente 0

Run: Servidor Cliente
/usr/lib/jvm/java-13-openjdk/bin/java -Djava.security.manager *-Djava.security.policy=/home/jlgallego99/Escritorio/Informatica/2o Cuatrimestre/DSD/Practicas/P3/EjemploRMI_2/src/server.policy" -javaagent:/opt/intellij-idea-ce/
Introduce el servidor:
Introduce el número de hebras:
Buscando el objeto remoto
Buscando el objeto remoto
Buscando el objeto remoto
Buscando el objeto remoto
Buscando el objeto remoto
Invocando el método remoto
Invocando el método remoto
Invocando el método remoto
Invocando el método remoto
Invocando el método remoto
Process finished with exit code 0
```

Ahora, probando a cambiar el método de la interfaz a uno con el modificador **synchronized** hace que ese método se ejecute en exclusión mutua, es decir, sólo un thread a la vez puede ejecutarlo. Por tanto, ahora cuando empiece una hebra tendrá que terminar antes de que empiece otra, a diferencia de antes que una hebra empezaba y mientras podían terminar o empezar otras:



```
Run: Servidor Cliente
/usr/lib/jvm/java-13-openjdk/bin/java -Djava.security.manager *-Djava.security.policy=/home/jlgallego99/Escritorio/Informatica/2o Cuatrimestre/DSD/Practicas/P3/EjemploRMI_2/src/server.policy" -javaagent:/opt/intellij-idea-ce/
Introduce el servidor:
Introduce el número de hebras:
Buscando el objeto remoto
Buscando el objeto remoto
Buscando el objeto remoto
Buscando el objeto remoto
Buscando el objeto remoto
Invocando el método remoto
Invocando el método remoto
Invocando el método remoto
Invocando el método remoto
Invocando el método remoto
Process finished with exit code 0
```



3. Ejemplo 3

Este último ejemplo consiste en un programa contador.

Interfaz remota

En este caso la interfaz remota se ha declarado y definido en los archivos **icontador.java** y **contador.java**. Se han creado en un módulo nuevo, dentro del módulo principal, y se ha generado el .jar correspondiente para poder usarlo como biblioteca. Los pasos para hacer esto en IntelliJ se definen en el otro entregable que explica todo lo relacionado con esta práctica e IntelliJ.

Al igual que las otras interfaces, define sus métodos controlando excepciones remotas. Esta en concreto tiene tres métodos: uno que devuelve un valor, otro que asigna un valor y lo devuelve y otro que incrementa en uno el valor y lo devuelve;

Servidor

Al igual que en los ejemplos anteriores, el servidor exporta los métodos de la interfaz del objeto remoto instanciado, en este caso el objeto es del tipo contador y se le ha puesto el nombre “micontador”.

Cliente

Por su parte el cliente introduce un valor inicial al contador del servidor, invoca primero a un método para inicializar el contador y luego invoca al método que incrementa ese contador 1000 veces. Finalmente imprime el valor final del contador junto al tiempo de respuesta medio calculado a partir de cuántas invocaciones remotas se han hecho al método de incrementar:

```
// Obtiene hora de comienzo
long horacomienzo = System.currentTimeMillis();
// ...
// Obtiene hora fin
long horafin = System.currentTimeMillis();
// Cálculo tiempo transcurrido
float transcurrido = ((horafin - horacomienzo)/1000f);
```

Se conecta con el servidor de igual manera que en los dos ejemplos anteriores.

Ejemplo de ejecución

The image displays two screenshots of an IDE's Run console, showing the execution of a Java RMI application. The top screenshot shows the initial setup and the first increment of a counter. The bottom screenshot shows the server running and the client sending a request.

Top Screenshot:

```
Run: Servidor Cliente
/usr/lib/jvm/java-13-openjdk/bin/java -Djava.security.manager -Djava.security.policy=../src/server.policy -javaagent:/opt/intellij-idea-ce/lib/idea_rt.jar=39823:/opt/intellij-idea-ce/bin -Dfile.encoding=UTF-8 -classpath ~/home,
Introduce el servidor:
Introduce el primer valor del contador:
Buscando el objeto remoto
Invocando el método remoto
Iniciando contador
Incrementando...
Media de las RMI realizadas = 0.144 msecs
RMI realizadas = 1000
Process finished with exit code 0
```

Bottom Screenshot:

```
Run: Servidor Cliente
/usr/lib/jvm/java-13-openjdk/bin/java -Djava.security.manager -Djava.security.policy=../src/server.policy -javaagent:/opt/intellij-idea-ce/lib/idea_rt.jar=44057:/opt/intellij-idea-ce/bin -Dfile.encoding=UTF-8 -classpath ~/home,
Servidor funcionando
```