Desarrollo de Sistemas Distribuidos

- Práctica 3. RMI -

Índice

Ignacio Yuste López. 3ºA1.

54141533Q

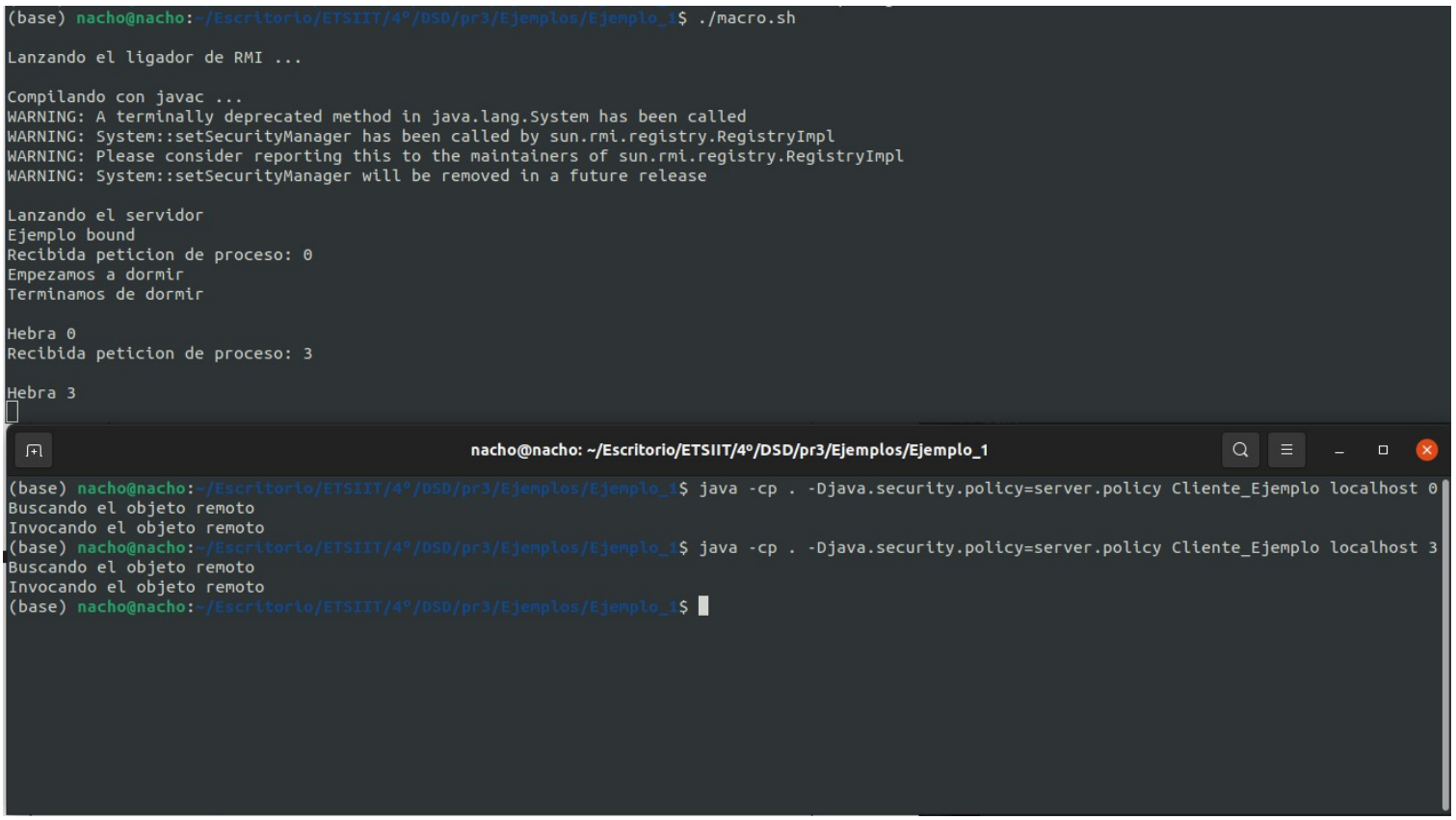
1. Ejecución de ejemplos.

1.1. Ejemplo 1:

El ejemplo 1 es un pequeño ejemplo de programa cliente-servidor. En este ejemplo el servidor (Ejemplo.java) exporta los métodos contenidos en la interfaz Ejemplo_I.java del objeto remoto instanciado como Ejemplo_I de la clase definida en Ejemplo.java.

Este programa consiste en un modelo cliente-servidor donde al llamar al método del objeto remoto desde el cliente el servidor muestra por pantalla el proceso que lo ha invovado.

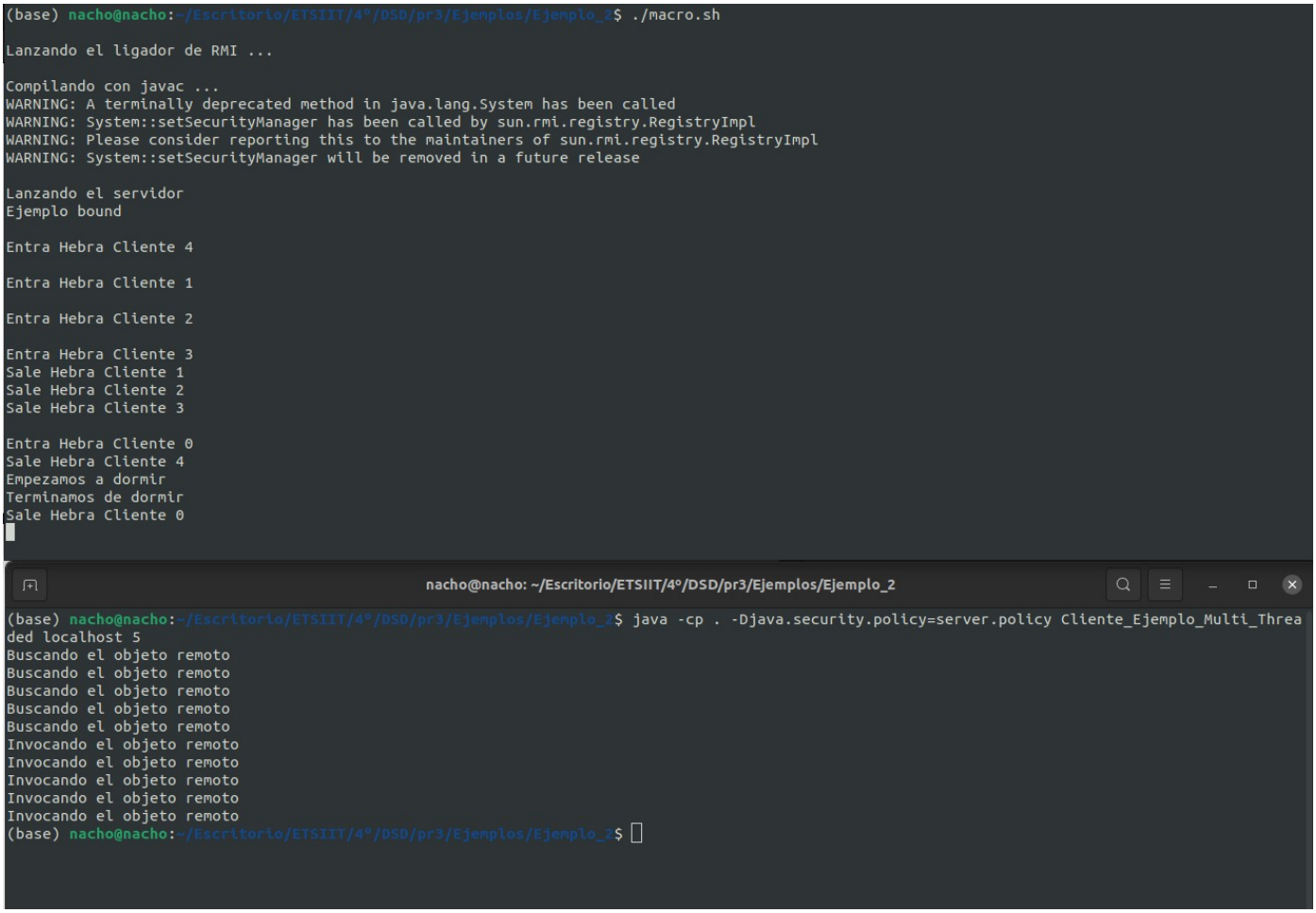
Para ejecutarlo hemos de, en primer lugar, ejecutar el archivo *macro.sh*. Este se encargará de compilar el código y de ejecutar el servidor. Las líneas para ejecutar los clientes está también escritas ahí pero al lanzar el servidor la ejecución del script no llega a ejecutarlas. Por tanto, en segundo lugar, debemos lanzar manualmente los clientes desde otra terminal. A continuación muestro el ejemplo de ejecución.



1.2. Ejemplo 2:

Este ejemplo es similar al anterior, pero 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. Este ejemplo nos permite ver la gestión de la concurrencia en RMI. En esta ocasión, en vez de pasar al objeto remoto un número entero, pasamos una cadena *String*.

Al igual que antes, debemos lanzar el servidor y los clientes por separado. A continuación se muestra un ejemplo de ejecución con 5 hebras:



Vemos como el orden de llegada es completamente arbitrario (aparentemente). La primera hebra que lanza el cliente es la hebra 0, y en esta ocasión ha sido la última en llegar.

1.3. Ejemplo 3:

El programa contador que se muestra a continuación es un pequeño ejemplo de programa 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

En este caso, el cliente usará el objeto remoto para incrementar un contador 1000 veces a partir de un valor inicial que él fija. Todos esto procesos los invoca el cliente pero los procesa el servidor, que es el encargado de instanciar la clase *contador*. Una vez terminado de incrementar se imprime el valor final obtenido.

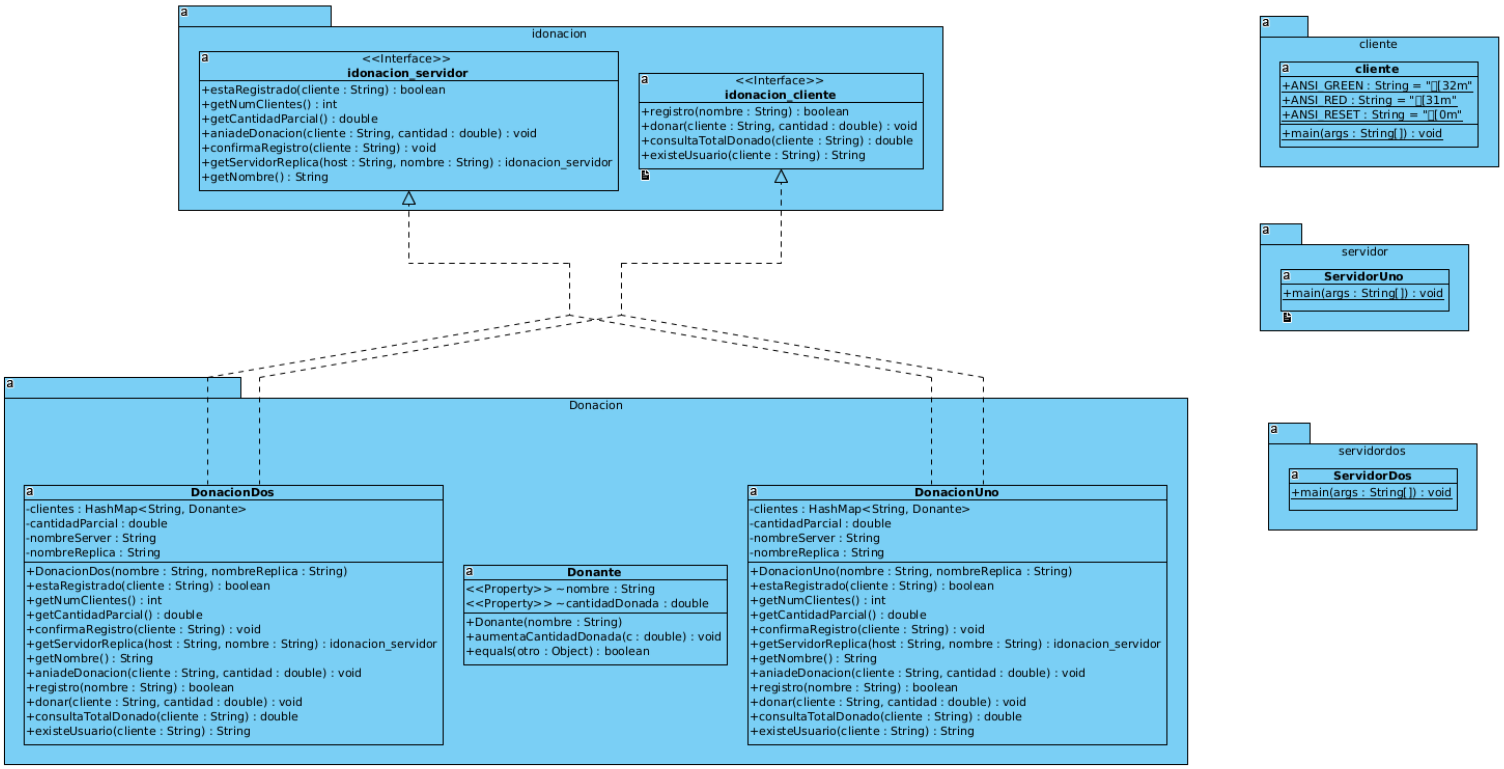
Al igual que en los casos anteriores, debemos ejecutar por separado servidor y cliente. A continuación se muestra un ejemplo de ejecución.

2. Sistema de donaciones.

Se pide crear un sistema de donaciones por el cual un cliente podrá registrarse en un servidor y donar dinero. Una vez registrado y habiendo hecho al menos una donación, podrá consultar el dinero total donado al sistema.

El sistema está compuesto por dos servidores idénticos, los cuales se distribuirán la carga equitativamente, repartiendo los usuarios entre ambos.

El diagrama de clases de mi implementación es el siguiente:



El sistema consta de 5 paquetes (en mi caso 5 proyectos de NetBeans).

En primer lugar está el paquete *idonacion*. En este paquete encontramos las dos interfaces que permiten la comunicación a través de la red. Consiste en dos interfaces, una para la comunicación cliente-servidor (*idonacion_cliente*) y otra para la comunicación entre servidores (*idonacion_servidor*). En la primera encontramos los métodos para registrar un usuario, donar dinero y obtener el total donado. Por el otro lado, encontramos todos los métodos necesarios para que el sistema funcione con los dos servidores, de tal manera que hay una comunicación constante entre ellos. El objetivo de esta comunicación en la mayoría de los casos es para obtener una instancia de la réplica para comprobar si el cliente existe en el otro sistema o para obtener el total donado, de tal manera que se produce una comunicación cliente-servidor entre ambos servidores de manera bilateral.

A continuación encontramos el paquete *Donacion*. En él encontramos las clases cuyas instancias mantendrán los servidores para acceder a su funcionalidad. Ambos, DonacionUno y DonacionDos son exactamente iguales, sólo cambia el nombre del si mismos y de la réplica, los cuales son contrarios. Estos objetos equilibran la carga de usuarios de manera equitativa automáticamente, ya que cuando se registra un usuario nuevo se hace en el servidor con menos usuarios y, cuando este inicia sesión, se devuelve el nombre del servidor al cual está conectado para poder hacer todas las operaciones de una manera directa. De esta manera obtenemos el equilibrio entre servidores de una manera transparente para el cliente.

También está la clase *Donante*, la cual se usa para almacenar el nombre de un cliente y cuanto dinero ha donado.

A continuación, encontramos los paquetes de los servidores. He decidido separar los servidores en dos proyectos de NetBeans ya que facilitaba mucho el script de ejecución, el cual más adelante comentaré su uso. Los servidores son idénticos, salvo que el *ServidorUno* es el encargado de crear el registro y asignar un puerto a ambos, además de que cada uno instancian un objeto distinto de los antes comentados.

Por último, el cliente se encarga de usar la interfaz *idonacion_cliente* para hacer uso de todas las funcionalidades antes descritas. Implementa una sencilla interfaz de texto con la cual puedes crear nuevos usuarios, iniciar sesión y hacer donaciones.

2.1. Ejecución del sistema.

En la carpeta raíz se incluye un archivo llamado *macro.sh*, el cual ejecuta los servidores y el cliente automáticamente.

Es necesario que los proyectos se hayan construido ya que el script usa los *.jar* de los mismos. Por este motivo he separado los servidores en dos proyectos, ya que de esta manera era más sencilla la creación del script al poder usar los *.jar.* Los únicos datos de entrada que se precisan son el nombre de los servidores, los cuales están incluidos ya dentro del propio script para facilitar aún más su uso. Por último, el script incorpora una línea la cual termina los procesos que se estén ejecutando en el puerto 1099, de esta manera termina con los servidores automáticamente. La línea puede ser comentada si es necesario.

Si se desea se puede ejecutar desde NetBeans. Para ello se tendrán que ejecutar los archivos dándo click derecho → run file. Lo único que hay que tener en cuenta es que es necesario ejecutarlos en este orden: *ServidorUno* → *ServidorDos* → *Cliente*. Esto se debe a que es el *ServidorUno* el que crea el registro, como ya he comentado antes.