

MEMORIA - PRÁCTICA 3

Primera parte - Implementación de ejemplos

Ejemplo 1

-En este primer ejemplo nos encontramos el programa más simple, tenemos una interfaz (Ejemplo_1.java) que tiene una operación de escribir un mensaje y recibe el id de proceso. Luego el programa servidor implementa este proceso mandando a dormir en caso de id = 0 y solo escribiendo mensajes por consola en caso contrario.

-Para ejecutar este ejemplo tenemos un script, pero este script se resume en que se compilan las clases con **javac *.java** luego tenemos que lanzar el **rmiregistry** (se puede lanzar en segundo plano para que sea más cómodo) y ya ejecutar primero el servidor y luego el cliente:

1. Ejecutamos el servidor: **java -cp . -Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost -Djava-security.policy=server.policy Ejemplo &**
2. Ejecutamos el cliente: **java -cp . -Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost -Djava-security.policy=server.policy Cliente_Ejemplo &**

-Podemos usar tanto el script como los comandos, pero el script hace estos mismos comandos.

-Capturas del funcionamiento:

```
adam@adam-laptop:~/Clases/3ºing/SEGUNDO SEMESTRE/Desarrollo de sistemas distribuidos (DSD)/PracticasDSD/RMI_ejemplo1$ ./ejemplo1_script.sh
Lanzando el ligador de RMI ...

Compilando con javac ...
WARNING: A terminally deprecated method in java.lang.System has been called
WARNING: System::setSecurityManager has been called by sun.rmi.registry.RegistryImpl
WARNING: Please consider reporting this to the maintainers of sun.rmi.registry.RegistryImpl
WARNING: System::setSecurityManager will be removed in a future release

Lanzando el servidor
Ejemplo bound

Lanzando el primer cliente
adam@adam-laptop:~/Clases/3ºing/SEGUNDO SEMESTRE/Desarrollo de sistemas distribuidos (DSD)/PracticasDSD/RMI_ejemplo1$ ./ejemplo1_script.sh
Lanzando el ligador de RMI ...

Compilando con javac ...
WARNING: A terminally deprecated method in java.lang.System has been called
WARNING: System::setSecurityManager has been called by sun.rmi.registry.RegistryImpl
WARNING: Please consider reporting this to the maintainers of sun.rmi.registry.RegistryImpl
WARNING: System::setSecurityManager will be removed in a future release

Lanzando el servidor
Ejemplo bound

Lanzando el primer cliente
```

Ejemplo 2

-Este ejemplo es algo más complejo (pero sigue siendo simple) ya que usa varias hebras, es decir, es multihebrado, por lo que (en un entorno real) tendría que aplicar sincronización de hebras para que sea seguro.

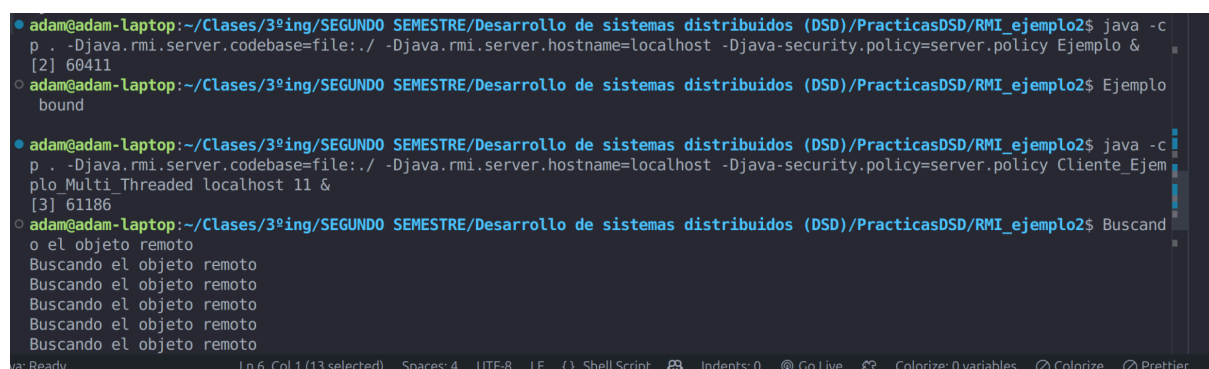
-La interfaz es exactamente igual salvo por el hecho de que ahora recibe un mensaje el método y no un proceso, el servidor cambia un poco porque ahora va a comprobar que el mensaje acabe por 0 y entonces dormirá (luego se vé que los mensajes que se pasan son los identificadores de las hebras), en caso de no hacerlo termina y ya está.

-El cliente utiliza la interfaz de Runnable para hacer ejecución con más de una hebra, y básicamente recibe por argumento la dirección (en nuestro caso localhost) y luego el número de hebras que tiene que lanzar, que las crea en el main y luego utiliza el método run para que cada una tenga una instancia local del servidor y ejecute la función mandando su identificador.

-El proceso es como antes se ha descrito:

1. Compilar con: **javac *.java**
2. Usamos: **rmiregistry &**
3. Ejecutamos el servidor: **java -cp . -Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost -Djava-security.policy=server.policy Ejemplo &**
4. Ejecutamos el cliente: **java -cp . -Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost -Djava-security.policy=server.policy Cliente_Ejemplo_Multi_Threaded localhost nºhebras &**

-Las siguientes capturas son de la ejecución tal cual está el código:



```
adam@adam-laptop:~/Clases/3ºing/SEGUNDO SEMESTRE/Desarrollo de sistemas distribuidos (DSD)/PracticasDSD/RMI_ejemplo2$ java -c
p . -Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost -Djava-security.policy=server.policy Ejemplo &
[2] 60411
adam@adam-laptop:~/Clases/3ºing/SEGUNDO SEMESTRE/Desarrollo de sistemas distribuidos (DSD)/PracticasDSD/RMI_ejemplo2$ Ejemplo
bound
adam@adam-laptop:~/Clases/3ºing/SEGUNDO SEMESTRE/Desarrollo de sistemas distribuidos (DSD)/PracticasDSD/RMI_ejemplo2$ java -c
p . -Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost -Djava-security.policy=server.policy Cliente_Ejem
plo_Multi_Threaded localhost 11 &
[3] 61186
adam@adam-laptop:~/Clases/3ºing/SEGUNDO SEMESTRE/Desarrollo de sistemas distribuidos (DSD)/PracticasDSD/RMI_ejemplo2$ Buscand
o el objeto remoto
Buscando el objeto remoto
Buscando el objeto remoto
Buscando el objeto remoto
Buscando el objeto remoto
Buscando el objeto remoto
```

Ln 6, Col 1

Java: Ready Ln 6, Col 1

Ln 6, Col 1 (13 selected) Spaces: 4 UTF-8 LF {} Shell Script Indents: 0 Go Live Colorize: 0 variables Colorize Prettier

[illegible]

Ejemplo 3

-En este ejemplo se hace una separación entre el objeto con las funciones y el servidor. Tenemos la interfaz del contador y luego una clase que implementa las funciones, y se creará por una parte el objeto remoto de dicha clase y luego el servidor por otra. El servidor crea una instancia de este contador y exporta sus funciones usando Naming para que el cliente pueda identificar y poder acceder a las funciones de dicho objeto.

-Se implementan varias funciones muy simples que permiten asignar un valor inicial, luego hacer incrementos de una unidad y también obtener el valor (un contador vaya).

-Esto es precisamente lo que hará el cliente, **obteniendo primero el objeto remoto** y luego **hace 1000 llamadas a la función de incrementar** y mientras lo hace calcula el tiempo que tarda e **imprime todo por pantalla**.

-El proceso es como antes se ha descrito:

1. Compilar con: **javac *.java**
2. **No** usamos rmiregistry porque en el propio código el servidor crea un registro con el puerto 1099 y el cliente hace `getRegistry(localhost, 1099)`, es decir, que es como si rmiregistry ya lo lanzasen los programas y no necesitamos hacerlo nosotros.
3. Ejecutamos el servidor: **java -cp . -Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost -Djava-security.policy=server.policy servidor &**
4. Ejecutamos el cliente: **java -cp . -Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost -Djava-security.policy=server.policy cliente**

-Captura de la ejecución:

```
adam@adam-Laptop:~/Clases/3ºing/SEGUNDO SEMESTRE/Desarrollo de sistemas distribuidos (DSD)/PracticasDSD/RMI_ejemplo3$ javac *.java
adam@adam-Laptop:~/Clases/3ºing/SEGUNDO SEMESTRE/Desarrollo de sistemas distribuidos (DSD)/PracticasDSD/RMI_ejemplo3$ java -cp . -Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy servidor &
[4] 84641
adam@adam-Laptop:~/Clases/3ºing/SEGUNDO SEMESTRE/Desarrollo de sistemas distribuidos (DSD)/PracticasDSD/RMI_ejemplo3$ java -cp . -Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy cliente &
[5] 84836
adam@adam-Laptop:~/Clases/3ºing/SEGUNDO SEMESTRE/Desarrollo de sistemas distribuidos (DSD)/PracticasDSD/RMI_ejemplo3$ Poniendo contador a 0
Incrementando...
Media de las RMI realizadas = 0.147 msecs
RMI realizadas = 1000

[5]+ Hecho java -cp . -Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy cliente
adam@adam-Laptop:~/Clases/3ºing/SEGUNDO SEMESTRE/Desarrollo de sistemas distribuidos (DSD)/PracticasDSD/RMI_ejemplo3$ java -cp . -Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy cliente &
[5] 85137
adam@adam-Laptop:~/Clases/3ºing/SEGUNDO SEMESTRE/Desarrollo de sistemas distribuidos (DSD)/PracticasDSD/RMI_ejemplo3$ Poniendo contador a 0
Incrementando...
Media de las RMI realizadas = 0.114 msecs
RMI realizadas = 1000

[5]+ Hecho java -cp . -Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy cliente
adam@adam-Laptop:~/Clases/3ºing/SEGUNDO SEMESTRE/Desarrollo de sistemas distribuidos (DSD)/PracticasDSD/RMI_ejemplo3$
```

Segunda parte

Pensamientos iniciales

-La implementación requiere de 2 servidores replicados, es decir, podemos implementar el mismo servidor con funciones apropiadas y asignación de puertos para que se puedan lanzar 2 servidores que sean el mismo código en distinto puerto y conectarse entre sí sabiendo el puerto del otro. Hay que iniciar bien los servidores para que creen el registro en su puerto concreto.

-Haría falta implementar una clase de Cliente para que los servidores puedan tener una lista de los mismos y contener toda la información necesaria como donaciones y todo eso.

-Hay que implementar tanto programa cliente como programa servidor. El cliente tendrá que hacer típica entrada por consola para elegir que hacer, se podría hacer que la primera entrada sea el nombre del cliente, luego hacer una consulta de si está registrado (se puede tener una lista y que se haga búsqueda en base al parámetro "nombre" del objeto Cliente) y luego devolver si está registrado (y el servidor en el que está, se podría devolver el puerto para que el programa lo use tal cual ya todo con ese servidor. En caso de no estar registrado se puede devolver directamente el número de clientes de ambos servidores (podemos hacer que se conecte al primero y luego que el primer servidor contacte con el otro). Una vez devuelva el servidor al que se conectaría el cliente se le dan las opciones de registrarse o salir, salir termina ejecución y registrarse lo registra en el servidor que era.

Descripción de la implementación

-Al final la implementación la he llevado a cabo utilizando: Una interfaz común (DonationInterface), un programa (Server1), otro programa (Server2), una clase que representa los clientes (Client) y el programa que implementa las operaciones del servidor (DonationProgram), en total 5 clases.

-Creo que se podría hacer con solo un programa servidor pero ya no tengo tiempo de comprobarlo, fue mi primer acercamiento pero me dió algunos errores.

Código e implementación

-Lo primero de todo tenemos la interfaz remota que implementa las operaciones básicas (no he implementado más de eso):

```
public interface DonationInterface extends Remote{

    // Funciones de registro de clientes, la primera es para
    // controlar las opciones del cliente, la segunda para registrarlo
    // en caso de que se lanzase register y ya estuviera registrado
    // (no debería ya que se controla desde el cliente) no se haría nada.

    // En caso de que esté registrado devuelve el nombre del servidor (que tiene el puerto)
    // Cuando lo registra le devuelve el puerto también

    public int registered(String clientName, ArrayList<String> visitedServers) throws RemoteException;
    public int register(String clientName) throws RemoteException;

    // La restricción es que esté registrado
    public void donate(String clientName, float amount) throws RemoteException;

    // Como la restricción es que haya donado una vez se comprueba ese valor
    public float totalAmountDonated(String clientName, boolean onlyGetValue) throws RemoteException;

    // Devolverá tal cual el listado de clientes que hayan donado,
    // esto se puede Shacer teniendo 2 arrayList(tedioso)
    // o se puede hacer filtrando el arrayList que se tenga de clientes (mas práctico)
    public ArrayList<String> benefactors(String clientName, boolean onlyGetValue) throws RemoteException;

    public int getNumClients() throws RemoteException;
}
```

-Son las operaciones listadas en la práctica junto a getNumClients que se usa para hacer la comprobación del número de clientes a la hora de hacer el registro. Importante remarcar el uso de los argumentos visitedServers y onlyGetValue, muy necesarios ya que ambos servidores comparten la implementación del objeto remoto y se harían llamadas el uno al otro infinitamente en algunos casos, por lo que estos argumentos sirven de comprobación para evitar bucles.

```
public class DonationProgram extends UnicastRemoteObject implements DonationInterface{

    private String serverName = "";
    private String replicaName = "";
    private int port = 0;
    private int replicaPort = 0;
    private Map<String, Client> registeredClients = new HashMap<>();
    private float totalLocalDonations = 0;

    /*
     * El constructor lo que hará es comprobar que servidor se tiene
     * que asignar y que servidor será su vecino, para hacerlo utiliza el
     * booleano que se manda desde el servidor al construirlo
     */
    DonationProgram(String serverName1, String serverName2) throws RemoteException {

        this.serverName = serverName1;
        this.replicaName = serverName2;

        this.port = Integer.parseInt(this.serverName.split("_")[1]);
        this.replicaPort = Integer.parseInt(this.replicaName.split("_")[1]);
    }
}
```

-En la implementación tenemos los atributos que se pueden ver, quiere decir que ambas instancias de DonationProgram tendrán en cuenta cuál es su nombre y puerto además de su compañero, esto se podría extender a más servidores usando listas de nombres u otra implementación.

-Luego están los clientes registrados, que están en un Map porque es lo más efectivo a la hora de acceder a los datos y guardarlos con un key, que en este caso es el nombre de cada cliente.

-Las funciones comprueban (usando los argumentos importantes que antes he mencionado) si tienen que devolver un valor o pedirle un valor o una función a su réplica y además las funciones: benefactors y totalAmountDonated comprueban que el cliente ya ha realizado una donación.

-Tanto donate como las dos previamente mencionadas no tienen en cuenta si podría estar registrado en la otra copia porque son funciones que se lanzan una vez ya se ha realizado el registro, es decir, se tiene que asegurar el programa main de haber hecho el registro y asignarle al cliente el puerto apropiado de su copia (lo devuelven register o registered), porque en caso de no estar ahí registrados simplemente devolverá un valor no útil.

```
public class Server1 {
    Run | Debug
    public static void main(String[] args){
        try {
            // Declaramos lo necesario
            int port = 1099;
            int portReplica = 1100;
            String nameServer = "servidor_"+port;
            String nameServerReplica = "servidor_"+portReplica;

            Registry reg = LocateRegistry.createRegistry(port); The value of the local variable re

            DonationProgram donator = new DonationProgram(nameServer, nameServerReplica);

            Naming.rebind("rmi://localhost:1099/"+nameServer, donator);

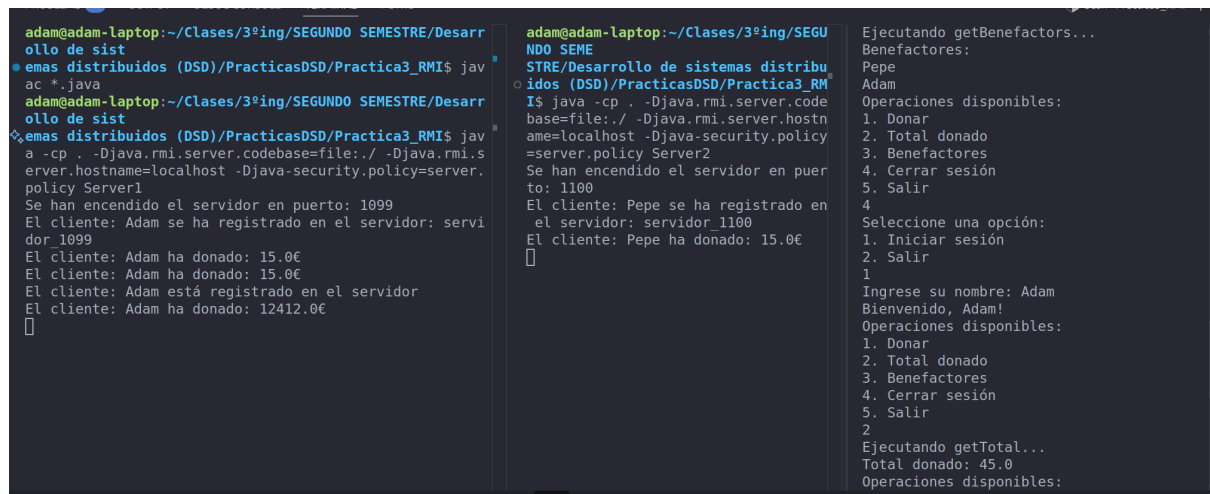
            System.out.println("Se han encendido el servidor en puerto: "
                               + port);

        } catch (RemoteException | MalformedURLException e){
            System.out.println("Exception: " + e.getMessage());
        }
    }
}
```

-Ambos servidores son prácticamente idénticos (de ahí que primero quisiese hacer solo uno) pero cambian en el orden de nameServer y nameServerReplica y el puerto en el que se lanzan.

-Por último está el main, que hace entrada de datos y control con variables de si está registrado o no, además para mayor comodidad utiliza un while general controlado por un booleano que tiene un switch para comprobar las decisiones y dentro del switch se hace la comprobación de registro y luego si está registrado tiene otro while de sesión, que se mantiene activo para las funciones básicas mientras no se salga o cierre sesión.

Ejecución



```
adam@adam-laptop:~/Clases/3ºing/SEGUNDO SEMESTRE/Desarrollo de sistemas distribuidos (DSD)/PracticasDSD/Practica3_RMI$ javac *.java
adam@adam-laptop:~/Clases/3ºing/SEGUNDO SEMESTRE/Desarrollo de sistemas distribuidos (DSD)/PracticasDSD/Practica3_RMI$ java -cp . -Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy Server1
Se han encendido el servidor en puerto: 1099
El cliente: Adam se ha registrado en el servidor: servidor 1099
El cliente: Adam ha donado: 15.0€
El cliente: Adam ha donado: 15.0€
El cliente: Adam está registrado en el servidor
El cliente: Adam ha donado: 12412.0€
[]

adam@adam-laptop:~/Clases/3ºing/SEGUNDO SEMESTRE/Desarrollo de sistemas distribuidos (DSD)/PracticasDSD/Practica3_RMI$ java -cp . -Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy Server2
Se han encendido el servidor en puerto: 1100
El cliente: Pepe se ha registrado en el servidor: servidor 1100
El cliente: Pepe ha donado: 15.0€
[]

Ejecutando getBenefactors...
Benefactores:
Pepe
Adam
Operaciones disponibles:
1. Donar
2. Total donado
3. Benefactores
4. Cerrar sesión
5. Salir
4
Seleccione una opción:
1. Iniciar sesión
2. Salir
1
Ingrese su nombre: Adam
Bienvenido, Adam!
Operaciones disponibles:
1. Donar
2. Total donado
3. Benefactores
4. Cerrar sesión
5. Salir
2
Ejecutando getTotal...
Total donado: 45.0
Operaciones disponibles:
```

-Una ejecución simple del programa se puede ver así, a la derecha está el main que va dando las opciones y en el servidor hay algunos System.out para poder ver que se están haciendo las cosas bien.

-En conclusión, he realizado solo la implementación más simple, principalmente por falta de organización, motivación y estar ocupado con otras cosas, pero he aprendido bien la mecánica de cómo funciona RMI y creo que podría implementar las operaciones más complejas o algún algoritmo con más réplicas.