Práctica 3

Nombre: Ana López Mohedano
Curso: 3º / Grupo: 2

En esta práctica se utilizará RMI (Remote Method Invocation). Con este mecanismo lograremos invocar métodos de forma remota. El servidor se lanzará y esperará peticiones, que el cliente pedirá, el propio cliente podrá invocar los métodos el objeto al que quiere acceder.

La primera parte de la práctica consiste en la realización de 3 ejemplos para aprender a usar RMI, y en la segunda se hará una aplicación cliente-servidor con 2 réplicas de un servidor, para gestionar donaciones de clientes.

Ejemplos

Ejemplo I

Aquí el servidor va a recibir una petición por parte del cliente y va a imprimir el argumento que va en la petición. El servidor “Ejemplo” implementa la interfaz “Ejemplo_I” , la cual va a ser invocada por el cliente y posteriormente exportará un objeto de tipo “Ejemplo”. El único método que tendrá la interfaz es el método “escribir_mensaje”, que comprobará el valor que se recibe del argumento y mostrarlo en el mensaje. Si recibe 0, esperará 5 segundos antes de mostrarlo, en caso contrario, se realizará sin esperas.

El script para ejecutar el programa es el siguiente:

```
#!/bin/sh -e
# ejecutar = Macro para compilacion y ejecucion del programa ejemplo
# en una sola maquina Unix de nombre localhost.
echo
echo "Lanzando el ligador de RMI ... "
rmiregistry &
echo
echo "Compilando con javac ..."
javac *.java
sleep 2
echo
echo "Lanzando el servidor"
java -cp . -Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy Ejemplo &
sleep 2

echo
echo "Lanzando el primer cliente"
echo
java -cp . -Djava.security.policy=server.policy Cliente_Ejemplo localhost 0
sleep 2
echo
echo "Lanzando el segundo cliente"
echo
java -cp . -Djava.security.policy=server.policy Cliente_Ejemplo localhost 3

```

Primero lanzamos rmiregistry, luego compilamos los archivos, y por último lanzamos el servidor en segundo plano y después los clientes.

Para ejecutar el servidor habrá que esepcificar la ubicación donde se descargan las definiciones de las clases del servidor, el nombre de la máquina donde se ejecutará (localhost) y el archivo de políticas de seguridad. Por otra parte, en el cliente, el archivo de políticas de seguridad también, donde está el servidor (localhost), y el argumento que se pasará al método.

Ejemplo II

En este ejemplo haremos algo parecido al primer ejemplo, pero con hebras en vez de varios clientes. Aquí vemos la concurrencia de RMI. Pasaremos una cadena String, en vez de un número entero.

El script:

```
#!/bin/sh -e
# ejecutar = Macro para compilacion y ejecucion del programa ejemplo
# en una sola maquina Unix de nombre localhost.
echo
echo "Lanzando el ligador de RMI ... "
rmiregistry &
echo
echo "Compilando con javac ..."
javac *.java
sleep 2
echo
echo "Lanzando el servidor"
java -cp . -Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy Ejemplo &
sleep 2

echo
echo "Lanzando el primer cliente"
echo
java -cp . -Djava.security.policy=server.policy Cliente_Ejemplo_Multi_Threaded localhost 3
sleep 2
#echo
#echo "Lanzando el segundo cliente"
#echo
#java -cp . -Djava.security.policy=server.policy Cliente_Ejemplo localhost 3

```

Primero lanzamos el ligador de RMI (rmiregistry), luego compilamos los archivos. Lanzamos el servidor en segundo plano (Ejemplo) y el cliente con los argumentos localhost (donde está alojado el servidor) y el número de hebras (en este caso, 3).

En la ejecución, la hebra 0 empieza a dormir y las demás se ejecutan de manera asíncrona.

Al añadir synchronized en escribir_mensaje no me funciona el ejemplo, me da error el propio synchronized porque no me lo reconoce, pero deberían ejecutarse las hebras de forma síncrona, es decir, esperarían a que las hebras terminen en vez de ejecutarse todas a la vez sin esperar a ninguna (por ejemplo, no se ejecutaría ninguna hebra hasta que termine la 0).

Ejemplo III

Aquí tenemos un programa contador, como ejemplo de programa cliente-servidor. Tenemos el objeto remoto (contador, que implementa icontador) y el servidor (servidor) separados. Nuestro objeto remoto contador tiene varias funciones (sumar, incrementar, etc.) a los que podremos acceder remotamente. El servidor exporta los métodos de la interfaz

icontador.java del objeto remoto instanciado como mi contador de la clase definida en contador.java. El programa cliente realiza las siguientes acciones:

- Pone un valor inicial en el contador del servidor
- Invoca al método incrementar 1000 veces
- Imprime el valor final del contador junto con el tiempo de respuesta medio.

El script es este:

```
#!/bin/sh -e
# ejecutar = Macro para compilacion y ejecucion del programa ejemplo
# en una sola maquina Unix de nombre localhost.
echo
#echo "Lanzando el ligador de RMI ... "
#rmiregistry 1099 &
echo
echo "Compilando con javac ..."
javac *.java
sleep 2

echo
echo "Lanzando el servidor"
java -cp . -Djava.rmi.server.codebase=file:./ -Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy servidor &
sleep 2

echo
echo "Lanzando el cliente"
echo
java -cp . -Djava.security.policy=server.policy cliente
sleep 2
```

aquí comentamos la línea de rmiregistry ya que de eso se encarga el servidor y el cliente,

Ejercicio

En el ejercicio debemos implementar dos servidores replicados. Debe cumplir los siguientes requisitos:

- Cada réplica desplegada en una máquina diferente, y estará encargado de recibir donaciones de entidades (clientes) para una causa humanitaria.
- El servidor proporcionará dos operaciones, registro de una entidad interesada (cliente) en la causa, y depósito de una donación a la causa. No es posible realizar un depósito (o más) sin haberse registrado el cliente previamente.
- Cuando una entidad desea registrarse y contacta con cualquiera de las dos réplicas del servidor, entonces el registro del cliente debe ocurrir realmente y de forma transparente en la réplica con menos entidades registradas. Es decir, el cliente sólo se ha dirigido a una réplica aunque esta no haya sido donde realmente ha quedado registrado, pero a partir de ese momento, el cliente realizará los depósitos en la réplica del servidor donde ha sido registrado.
- Cada réplica del servidor mantendrá el subtotal de las donaciones realizadas en dicha réplica.
- Un cliente no podrá registrarse maś de una vez, ni siquiera en réplicas distintas.
- Los servidores también ofrecerán una operación de consulta del total donado en un momento dado. Dicha operación sólo podrá llevarse a cabo si el cliente previamente se ha registrado y ha realizado al menos un depósito. Cuando un cliente consulte la cantidad total donada hasta el momento, sólo hará la petición a la réplica donde se

encuentra registrado, y ésta será la encargada, realizando la operación oportuna con la otra réplica, de devolver el total donado hasta el momento.

Tendremos varias clases, Servidor1.java y Servidor2.java, Cliente.java, Donacion_I.java que será la interfaz que proporcione los métodos, y Donacion.java, que implementará esos métodos.

Servidores

Ambos servidores son similares, solo que se crearán en registros diferentes para poder simular que están en máquinas diferentes. Esto se hace con la línea *LocateRegistry.createRegistry(1098);*.
En ambos servidores añadiremos el objeto remoto Donacion, llamándolos Donacion1 y Donacion2 respectivamente. Esto se hace creando un objeto Donacion y luego usando la línea *Naming.rebind("Donacion1");* en el caso del Servidor1, o *Naming.rebind("Donacion2");*, en el caso del Servidor2.

Servidor1:

```
String nombre1 = "servidor2";
Registry reg = LocateRegistry.createRegistry(1097);
Donacion miDonacion = new Donacion(nombre1);
Naming.rebind("servidor1", miDonacion);
```

Servidor2:

```
String nombre2 = "servidor1";
Registry reg = LocateRegistry.createRegistry(1098);
Donacion miDonacion = new Donacion(nombre2);
Naming.rebind("servidor2", miDonacion);
```

Cliente, Donación y Entidad

En la clase donación tendremos todos los métodos y los datos que queremos gestionar.

```java
public class Donacion extends UnicastRemoteObject implements Donacion_I {
    private int subtotal_donado;
    int total_donado;
    ArrayList<Entidad> entidades;
    String nombreReplica;


    public Donacion(String nombreServer) throws RemoteException {
        entidades = new ArrayList<Entidad>();
        nombreReplica = nombreServer;
    }

    public Donacion_I getReplica() throws RemoteException, NotBoundException {
        Registry reg;
        if(nombreReplica == "servidor1")
            reg = LocateRegistry.getRegistry("127.0.0.1", 1097);
        else
            reg = LocateRegistry.getRegistry("127.0.0.1", 1098);


        return (Donacion_I) reg.lookup(nombreReplica);
    }

    public void registro_cliente(String nombre) throws RemoteException, NotBoundException{
        Donacion_I replica = this.getReplica();
        boolean esta_registrado = this.estaRegistrado(nombre);
        boolean esta_registrado_en_replica = replica.estaRegistrado(nombre);

        if(esta_registrado || esta_registrado_en_replica){
            System.out.println(nombre + " ya registrado en el sistema.");
        }
        else{ // no registrado
            int entidades_replica = replica.getTotalEntidades();

            if(entidades.size() <= entidades_replica){
                System.out.println("Registrado a "+nombre);
                Entidad entidad = new Entidad(nombre);
                entidades.add(entidad);
            }
            else{
                System.out.println("Se registra la entidad "+nombre+" en la replica");
                replica.registro_cliente(nombre);
            }
        }

```

Estos serán los métodos, definidos en Donacion_I.java:

```java
public interface Donacion_I extends Remote {
    public Donacion_I getReplica() throws RemoteException, NotBoundException;

    public void registro_cliente(String nombre) throws RemoteException, NotBoundException;

    public boolean estaRegistrado(String nombre) throws RemoteException;

    public int getTotalEntidades() throws RemoteException;

    public void donar(String nombre, int cantidad) throws RemoteException, NotBoundException;

    public void recibirActualizacion(int total) throws RemoteException;

    public Entidad getEntidad(String nombre) throws RemoteException;

    public int consultarSubtotalDonado(String nombre) throws RemoteException;
    public int consultarTotalDonado(String nombre) throws RemoteException, NotBoundException;
    public int consultarTotalDonadoUsuario(String nombre) throws RemoteException;

}
```

Y esta será la clase Entidad:

```java
public class Entidad {
    private String nombre;
    private int total_donado;

    Entidad(String nombre){
        this.nombre = nombre;
        total_donado = 0;
    }

    public int getTotalDonado(){
        return total_donado;
    }

    public void aniadirDonacion(float cantidad){
        total_donado += cantidad;
    }

    public String getNombre(){
        return nombre;
    }

    public void setDonacion(float cantidad){
        total_donado += cantidad;
    }
}
```

Aquí almacenamos tanto el nombre de la entidad como el total donado por dicha entidad. Tienemétodos para añadir donación, obtener el nombre o establecer una donación.

En el cliente, primero obtendremos los registros de ambos servidores, creados con puertos diferentes:

```java
// Crea el stub para el cliente especificando el nombre del servidor
Registry mireg = LocateRegistry.getRegistry("127.0.0.1", 1097);
Registry mireg2 = LocateRegistry.getRegistry("127.0.0.1", 1098);

System.out.println("Registros creados");
System.out.println("Buscando servidores de donacion...");

Donacion_I midonacion1 = (Donacion_I) mireg.lookup("servidor1");
Donacion_I midonacion2 = (Donacion_I) mireg2.lookup("servidor2");
System.out.println("servidores encontrados");
```

Después se obtienen de forma remota las clases Donacion, usando lookup para buscar por el nombre.

Realizaremos algunas actividades, primeramente registraremos a los clientes y luego haremos donaciones a los servidores, algunos clientes lo harán al primero y otros al segundo, pero luego se gestionará a cuál se dona en función de la cantidad que lleven.

```
// registramos
midonacion1.registro_cliente(nombre:"Paco");
midonacion2.registro_cliente(nombre:"Paco");
midonacion1.registro_cliente(nombre:"Juan");
midonacion2.registro_cliente(nombre:"Pepe");

// donar
midonacion1.donar(nombre:"Pepe", cantidad:200);
midonacion2.donar(nombre:"Paco", cantidad:100);
midonacion1.donar(nombre:"Juan", cantidad:70);
```