Departamento de Ingeniería Telemática
Escuela Superior de Ingenieros
Universidad de Sevilla

PRÁCTICA 5.

RMI: Referencias a objetos remotos II

1. OBJETIVO

El objetivo es mostrar el paso de referencias a objetos remotos como valor devuelto por un método.

2. DESCRIPCIÓN

En los ejemplos que se han visto en las prácticas anteriores, los clientes obtenían las referencias de servicios remotos a través del registro rmi (rmiregistry) o bien como referencias a objetos remotos que se envían como parámetros de un método. En esta práctica se ilustra la obtención de referencias a objetos remotos mediante el valor de retorno de un método.

2.1 Referencias remotas como valor retornado (fábricas de referencias remotas): servicio simple de banco

Además de poder ser recibidas como parámetros de un método, puede obtenerse una referencia remota como el valor de retorno de un método (al fin y al cabo, eso es lo que hace el método `lookup` del `rmiregistry`).

Dentro de este ámbito, es muy frecuente el uso de un esquema de tipo *fábrica de referencias remotas*. Este esquema se suele usar cuando se requiere ir creando dinámicamente objetos remotos (por ejemplo, un objeto que actúe como cerrojo). Con este modelo, el servidor crea, y registra en el `rmiregistry`, un servicio remoto que ofrece una operación para crear un nuevo objeto remoto (en el ejemplo, un servicio de *fabricación* de cerrojos con un método para crear uno nuevo). Esa operación instancia un nuevo objeto remoto y retorna una referencia remota al mismo.

Para ilustrar este escenario típico, vamos a crear un servicio bancario trivial, que permite crear dinámicamente cuentas bancarias.

A continuación, se muestra la interfaz remota correspondiente a la *fábrica* de cuentas (fichero `Banco.java`), que será la que se registre en el `rmiregistry`:

```
import java.rmi.*;

interface Banco extends Remote {
    Cuenta crearCuenta(String nombre) throws RemoteException;
}
```

Banco.java

La clase que implementa esta interfaz (fichero `BancoImpl`) crea un nuevo objeto que implementa la interfaz remota `Cuenta`:

```
import java.rmi.*;
import java.rmi.server.*;

class BancoImpl extends UnicastRemoteObject implements Banco {
    BancoImpl() throws RemoteException {
    }
    public Cuenta crearCuenta(String nombre)
                                throws RemoteException {
        return new CuentaImpl(nombre);
    }
}
```

BancoImpl.java

La interfaz remota correspondiente a una cuenta bancaria (fichero `Cuenta`) especifica unos métodos para operar, hipotéticamente, con esa cuenta:

```
import java.rmi.*;

interface Cuenta extends Remote {
    String obtenerNombre() throws RemoteException;
    float obtenerSaldo() throws RemoteException;
    float operacion(float valor) throws RemoteException;
}
```

Cuenta.java

Y, a continuación, se incluye la clase (fichero `CuentaImpl`) que implementa esos métodos:

```
import java.rmi.*;
import java.rmi.server.*;

class CuentaImpl extends UnicastRemoteObject implements Cuenta
{
    private String nombre;
    private float saldo = 0;
```

```
    CuentaImpl(String n) throws RemoteException {
        nombre = n;
    }
    public String obtenerNombre() throws RemoteException {
        return nombre;
    }
    public float obtenerSaldo() throws RemoteException {
        return saldo;
    }
    public float operacion(float valor) throws RemoteException
{
        saldo += valor;
        return saldo;
    }
}
```

ClienteImpl.java

Tanto el cliente (fichero `ClienteBanco.java`) como el servidor (fichero `ServidorBanco.java`) son similares a los de los ejemplos previos, y se muestran a continuación:

```
class ClienteBanco {
    static public void main (String args[]) {
        if (args.length!=3) {
            System.err.println("Uso: ClienteBanco hostregistro
numPuertoRegistro nombreTitular");
            return;
        }

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

        try {
            Banco srv = (Banco) Naming.lookup("//" + args[0] +
":" + args[1] + "/Banco");
            Cuenta c = srv.crearCuenta(args[2]);
            c.operacion(30);
            System.out.println(c.obtenerNombre()  +  ":  "  +
c.obtenerSaldo());
            c.operacion(-5);
            System.out.println(c.obtenerNombre()  +  ":  "  +
c.obtenerSaldo());
        }
        catch (RemoteException e) {
            System.err.println("Error  de  comunicacion:  "  +
e.toString());
        }
        catch (Exception e) {
```

```
            System.err.println("Excepcion en ClienteBanco:");
            e.printStackTrace();
        }
    }
}
```

ClienteBanco.java

```
import java.rmi.*;
import java.rmi.server.*;

class ServidorBanco  {
    static public void main (String args[]) {
       if (args.length!=1) {
            System.err.println("Uso:             ServidorBanco
numPuertoRegistro");
            return;
        }
        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new
RMISecurityManager());
        }
        try {
            BancoImpl srv = new BancoImpl();
            Naming.rebind("rmi://localhost:"   +   args[0]   +
"/Banco", srv);
        }
        catch (RemoteException e) {
            System.err.println("Error  de  comunicacion:  "  +
e.toString());
            System.exit(1);
        }
        catch (Exception e) {
            System.err.println("Excepcion en ServidorBanco:");
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

ServidorBanco.java

Por último, es conveniente hacer algún comentario sobre el ciclo de vida de los objetos remotos creados dinámicamente. Al igual que ocurre con cualquier objeto en Java, el objeto seguirá *vivo* mientras haya alguna referencia al mismo. En el caso de Java RMI, esto se extiende a toda la aplicación distribuida: el objeto que implementa una interfaz remota seguirá vivo mientras haya una referencia local o remota al mismo. Java RMI, por tanto, implementa un

recolector de basura distribuido para poder hacer un seguimiento de la evolución de los objetos remotos.

⇒ 1. Descargue el fichero de código correspondiente a esta práctica. Compile y ejecute este ejemplo.

⇒ 2. Partiendo del ServicioLog visto en la práctica 3 correspondiente a RMI, crear una fábrica de logs llamada FabricaLog que contenga el método de creación de un servicio de log. Al método de creación se le pasa como parámetro un nombre identificativo del log, este nombre servirá como el nombre del fichero log que se va a utilizar. La impresión por pantalla que haga el servicio log debe incluir el nombre identificativo. Crea varios clientes que creen servicios de log.

⇒ 3. En la tarea habilitada para la práctica, entregue un fichero comprimido (.zip) con nombre su login que sea el directorio de nombre su login comprimido (que contiene el directorio p5RMIcodigo con todo su trabajo).