Departamento de Ingeniería Telemática
Escuela Superior de Ingenieros
Universidad de Sevilla

PRÁCTICA 4.
RMI: Referencias a objetos remotos

1. OBJETIVO

El objetivo es mostrar el paso de referencias a objetos remotos como parámetros. También se pueden devolver como valor de retorno de un método. Esto último se verá en la siguiente práctica.

2. DESCRIPCIÓN

En los ejemplos que se han visto en la práctica anterior, los clientes obtenían las referencias de servicios remotos a través del registro rmi (rmiregistry). Sin embargo, teniendo en cuenta que estas referencias son objetos Java convencionales, éstas se pueden recibir también como parámetros de un método, como se ilustra en la sección 2.1 (mediante el ejemplo de servicio de chat), o como valor de retorno del mismo, tal como se mostrará en la siguiente práctica.

De esta forma, se podría decir que el registro sirve como punto de contacto inicial para obtener la primera referencia remota, pero que, a continuación, los procesos implicados pueden pasarse referencias remotas adicionales entre sí.

2.1 Referencias remotas como parámetros (para *callbacks*): servicio de chat

Es importante resaltar que cuando se especifica como parámetro de un método RMI una referencia remota, a diferencia de lo que ocurre con el resto de los objetos, que se transfieren por valor, ésta se pasa por referencia (se podría decir que se pasa por valor la propia referencia).

Para ilustrar el uso de referencias remotas como parámetros se plantea en esta sección un servicio de chat. Este servicio permitirá que cuando un usuario, identificado por un apodo, se conecte al mismo, reciba todo lo que escriben el resto de los usuarios conectados y, a su vez, éstos reciban todo lo que escribe el mismo.

Esta aplicación se va a organizar con procesos clientes que atienden a los usuarios y un servidor que gestiona la información sobre los clientes/usuarios conectados.

De manera similar a los ejemplos previos, el servidor ofrecerá un servicio remoto para darse de alta y de baja, así como para enviarle la información que escribe cada usuario.

Sin embargo, en este caso, se requiere, además, que los clientes ofrezcan una interfaz remota para ser notificados de lo que escriben los otros clientes.

A continuación, se muestra la interfaz remota proporcionada por el servidor:

```
import java.rmi.*;

interface ServicioChat extends Remote {
    void alta(Cliente c) throws RemoteException;
    void baja(Cliente c) throws RemoteException;
    void envio(Cliente c, String apodo, String m) throws
RemoteException;
}
```

ServicioChat.java

El tipo `Cliente` que aparece como parámetro de los métodos corresponde a una interfaz remota implementada por el cliente y que permite notificar a un usuario de los mensajes recibidos por otros usuarios (a esta llamada a contracorriente, del servidor al cliente, se le suele denominar *callback*). Se trata, por tanto, de una referencia remota recibida como parámetro, sin necesidad de involucrar al `rmiregistry` en el proceso. A continuación, se muestra esa interfaz remota proporcionada por el cliente:

```
import java.rmi.*;

interface Cliente extends Remote {
    void notificacion(String apodo, String m) throws
RemoteException;
}
```

Cliente.java

Pasamos a la implementación del servicio de chat (fichero `ServicioChatImpl.java`) que usa un contenedor de tipo lista para guardar los clientes conectados:

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

class ServicioChatImpl extends UnicastRemoteObject implements
ServicioChat {
    List<Cliente> l;
    ServicioChatImpl() throws RemoteException {
        l = new LinkedList<Cliente>();
    }
    public void alta(Cliente c) throws RemoteException {
        l.add(c);
    }
    public void baja(Cliente c) throws RemoteException {
        l.remove(l.indexOf(c));
    }
    public void envio(Cliente esc, String apodo, String m)
```

```
        throws RemoteException {
          for (Cliente c: l)
              if (!c.equals(esc))
                  c.notificacion(apodo, m);
      }
  }
```

ServicioChatImpl.java

Obsérvese como en `envio` se invoca el método `notificacion` de todos los clientes (es decir, de todas las interfaces remotas de cliente) en la lista, exceptuando la del propio escritor.

Se muestra a continuación el código del servidor (fichero ServidorChat) que es similar a todos los servidores de los ejemplos previos.

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

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

ServidorChat.java

Es interesante el código del cliente (fichero `ClienteChat`), puesto que tiene que hacer el doble rol de cliente y de servidor: debe buscar en el `rmiregistry` el servicio de chat remoto, pero también tiene que instanciar un objeto que implemente la interfaz remota de cliente.

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

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

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

        try {

            ServicioChat srv = (ServicioChat)
Naming.lookup("//" + args[0] + ":" + args[1] + "/Chat");
            ClienteImpl c = new ClienteImpl();
            srv.alta(c);
            Scanner ent = new Scanner(System.in);
            String apodo = args[2];
            System.out.print(apodo + "> ");
            while (ent.hasNextLine()) {
                srv.envio(c, apodo, ent.nextLine());
                System.out.print(apodo + "> ");
            }
            srv.baja(c);
            System.exit(0);
        }
        catch (RemoteException e) {
            System.err.println("Error de comunicacion: " +
e.toString());
        }
        catch (Exception e) {
            System.err.println("Excepcion en ClienteChat:");
            e.printStackTrace();
        }
    }
```

ClienteChat.java

Nótese que al tener también un perfil de servidor, es necesario terminar su ejecución explícitamente con `System.exit`.

Por último, se muestra la implementación (fichero `ClienteImpl.java`):

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

class ClienteImpl extends UnicastRemoteObject implements
Cliente {
    ClienteImpl() throws RemoteException {
    }
    public void notificacion(String apodo, String m)
                                    throws RemoteException {
        System.out.println("\n" + apodo + "> " + m);
    }
}
```

ClienteImpl.java

Hay que resaltar que el método `notificacion` se ejecutará de forma asíncrona con respecto al flujo de ejecución del cliente.

Con respecto a la compilación y ejecución de estos programas, en este caso es necesario también disponer en la máquina que ejecuta el servidor del fichero `class` correspondiente a la interfaz remota de cliente (`Cliente.class`).

⇒ 1. Descargue y descomprima el fichero .zip correspondiente a la práctica. Sitúe todo el código dentro de un directorio cuyo nombre sea tu login.

⇒ 2. Compile y ejecute el ejemplo de chat. Para ello hay que tener en cuenta que la interfaz Cliente es necesaria para la compilación del servidor y que la interfaz ServicioChat es necesaria para la compilación del cliente. Por eso, los pasos a seguir para la compilación son los siguientes:

- Compilar en el cliente Cliente.java

- Copiar Cliente.class en el directorio del servidor.

- Compilar el servidor.

- Copiar ServicioChat.class del servidor en el cliente.

- Compilar el cliente.

Para la ejecución en el centro de cálculo hay que tener en cuenta las opciones que ya se comentaron en la práctica anterior.

En la ejecución, la opción server.hostname hay que indicarla tanto en el servidor como en el cliente, ya que los dos hacen de servidor.

⇒ 3. En el directorio alarmaTemp, realice un servicio de alarma que avise a los observadores cuando la temperatura supere un límite máximo (por ejemplo 40). Para ello utilice la siguiente interfaz:

```
interface ServicioAlarma extends Remote {

   void addObservador(Observador o) throws RemoteException;

   void delObservador(Observador o) throws RemoteException;

   void setTemperatura(int temp) throws RemoteException;

   int getTemperatura() throws RemoteException;

}
```

Implemente el servicio en ServicioAlarmaImpl.java, y el servidor en ServidorAlarma.java.

El servicio de alarma cuando se actualiza la temperatura con setTemperatura, debe avisar a todos los observadores que se han añadido como tales, llamando al método de la interfaz de Observador.

La interfaz del observador debe ser la siguiente:

```
interface Observador extends Remote {

   void temperaturaMaxAlcanzada(int tempAct) throws RemoteException;

}
```

Cree un cliente llamado ObservadorAlarma que obtenga la referencia al Servicio de alarma, cree un objeto ObservadorImpl y lo añada a los observadores del servicio de alarma.

Cree otro cliente llamado ClienteAlarma que actualice la temperatura desde 35 a 50 dejando transcurrir 1 s en cada actualización.

Para la compilación de ClienteAlarma será necesario tener tanto el ServicioAlarma.class como Observador.class

Para ver el funcionamiento ejecute ClienteAlarma y varios observadores.

⇒ 4. 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 p4RMIcodigo con todo su trabajo).