



Tarea: juego números en RMI

Alberto Mateos Checa

Implementación del juego de averiguar un número utilizando la tecnología RMI de Java.

1. Introducción.-

RMI es una tecnología propia de Java que permite ejecutar métodos de forma remota. Para ello, es necesario el uso de un registro (RMIRegistry) en el que se registran los objetos del servidor sobre los que, posteriormente, se realizarán las llamadas remotas.

Para la elaboración de la tarea es imprescindible realizar los siguientes pasos:

1. Definir una interfaz remota.
2. Implementar el servidor.
3. Implementar el cliente.

A continuación se detallan los pasos anteriores.

2. Definición de la interfaz remota.-

Para hacer uso de RMI hay que disponer de una interfaz remota. Ésta no es mas que una interfaz (tipo de clase) que debe de extender de la clase **java.rmi.Remote**. En este caso, la interfaz remota se ha implementado con el nombre **Juego.java**.

En la interfaz se definen los métodos que se van a ejecutar sobre los objetos remotos pero no se implementa el código propio de dichos métodos, tal y como se hace con las clases interfaz de Java habitualmente. De esta forma, en este caso, los métodos definidos han sido dos:

- **iniciarJuego()**: el cliente ejecutará este método para comenzar una nueva partida. Devuelve un String que será el mensaje de respuesta del servidor.
- **comprobarNumero(int numeroCliente)**: el cliente ejecutará este método para pedirle al servidor que compruebe si el número que se le pasa como parámetro es el número buscado. Devuelve un String como mensaje de respuesta del servidor.

3. Implementación del servidor.-

La clase servidor será la encargada de implementar los métodos remotos que han sido definidos en la interfaz. Es por ello que la clase **Servidor.java** **implementa Juego.java**. De esta forma, tanto los métodos **iniciarJuego()** como **comprobarNumero()** son implementados en esta clase, definiendo el código que se ejecutará en cada caso. Debido a la sencillez de la aplicación no se va a explicar el código empleado para cada caso.

Además de implementar los métodos remotos, la clase **Servidor.java** incluye en el método main la creación del objeto que se va a compartir así como su registro. Para ello se utiliza el siguiente código:

```
Servidor obj = new Servidor();
Juego stub = (Juego)UnicastRemoteObject.exportObject(obj, 0);
Registry registry = LocateRegistry.getRegistry();
registry.bind("Juego", stub);
```

Como puede verse, tras crearse un objeto de la clase Servidor, éste se define como remoto del tipo Juego (la interfaz remota). Posteriormente, tras obtenerse el registro de RMI, se une el objeto al registro para que sea accesible desde los clientes.

4. Implementación del cliente.-

La clase **Cliente.java** es también muy sencilla, realizándose las operaciones mediante un bucle principal del tipo do-while que se encuentra dentro del método main de la clase. De esta forma, se solicita que se introduzca un número a través de la entrada estándar (consola) hasta que se acierta el número que propone el servidor o se supera el número máximo de intentos, momento en el cual se cierra la aplicación.

Para comprobar si el número introducido es mayor, menor o el buscado se hace uso del método remoto comprobarNumero(). De la misma forma, para comenzar la partida se usa el método iniciarJuego().

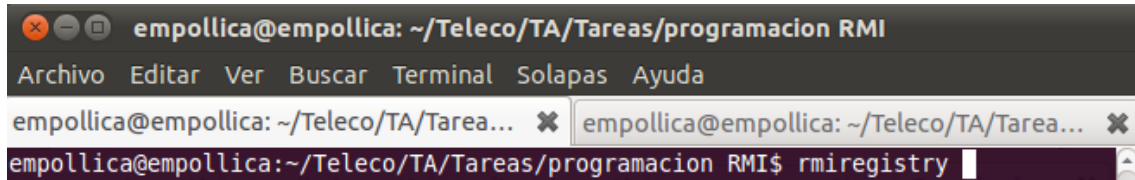
Por tanto, ya que se hace uso de métodos remotos, es necesario disponer un objeto que se encuentre en el registro sobre el que ejecutar los métodos. Para ello, se implementa el siguiente código:

```
Registry registry = LocateRegistry.getRegistry("127.0.0.1");
Juego stub = (Juego) registry.lookup("Juego");
String respuesta = stub.iniciarJuego();
```

De esta forma, lo que se hace es obtener el registro RMI a través de la dirección IP en la que se encuentra éste (localhost en este caso) y, posteriormente, buscar el objeto sobre el que se van a ejecutar los métodos. Este objeto se tiene de forma local y se puede utilizar para ejecutar cualquier método de los que fueron definidos en la interfaz remota. En el código anterior vemos que se indica la orden de iniciar un nuevo juego, obteniéndose la respuesta del servidor en un String que es lo que devuelve el método iniciarJuego().

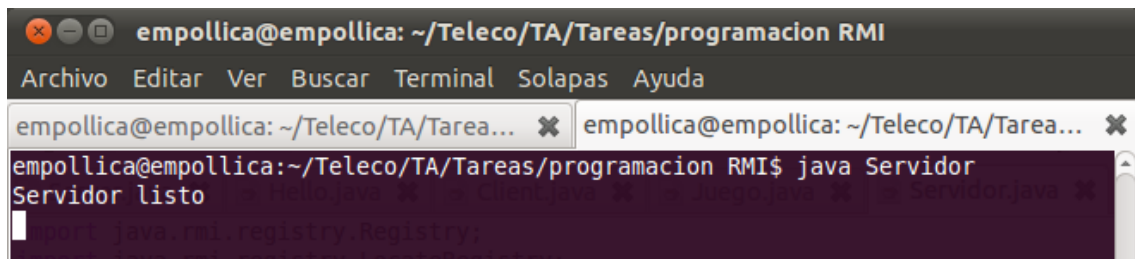
5. Resultados.-

Para la ejecución del juego lo primero que hay que hacer es iniciar el registro de RMI en el servidor:



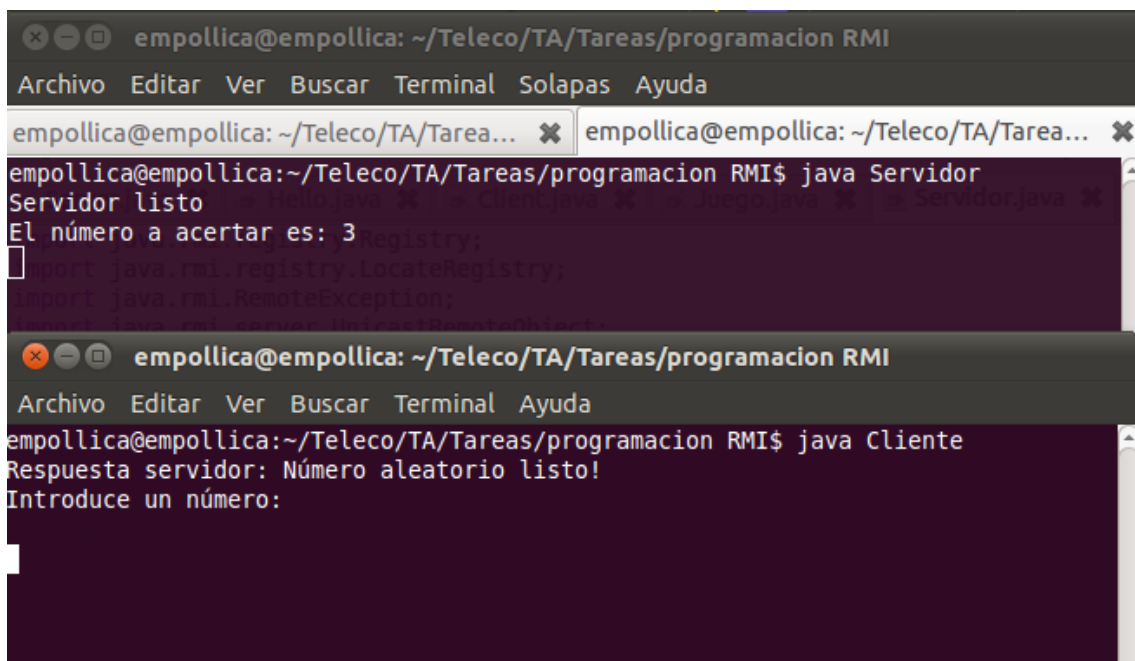
```
empollica@empollica: ~/Teleco/TA/Tareas/programacion RMI
Archivo Editar Ver Buscar Terminal Solapas Ayuda
empollica@empollica: ~/Teleco/TA/Tarea... x empollica@empollica: ~/Teleco/TA/Tarea... x
empollica@empollica:~/Teleco/TA/Tareas/programacion RMI$ rmiregistry
```

A continuación, se ejecuta el servidor, el cual se mantiene a la espera de peticiones del cliente:



```
empollica@empollica: ~/Teleco/TA/Tareas/programacion RMI
Archivo Editar Ver Buscar Terminal Solapas Ayuda
empollica@empollica: ~/Teleco/TA/Tarea... x empollica@empollica: ~/Teleco/TA/Tarea... x
empollica@empollica:~/Teleco/TA/Tareas/programacion RMI$ java Servidor
Servidor listo
```

Una vez que el servidor está a la espera, se procede a la ejecución del cliente, con el consecuente inicio del juego:

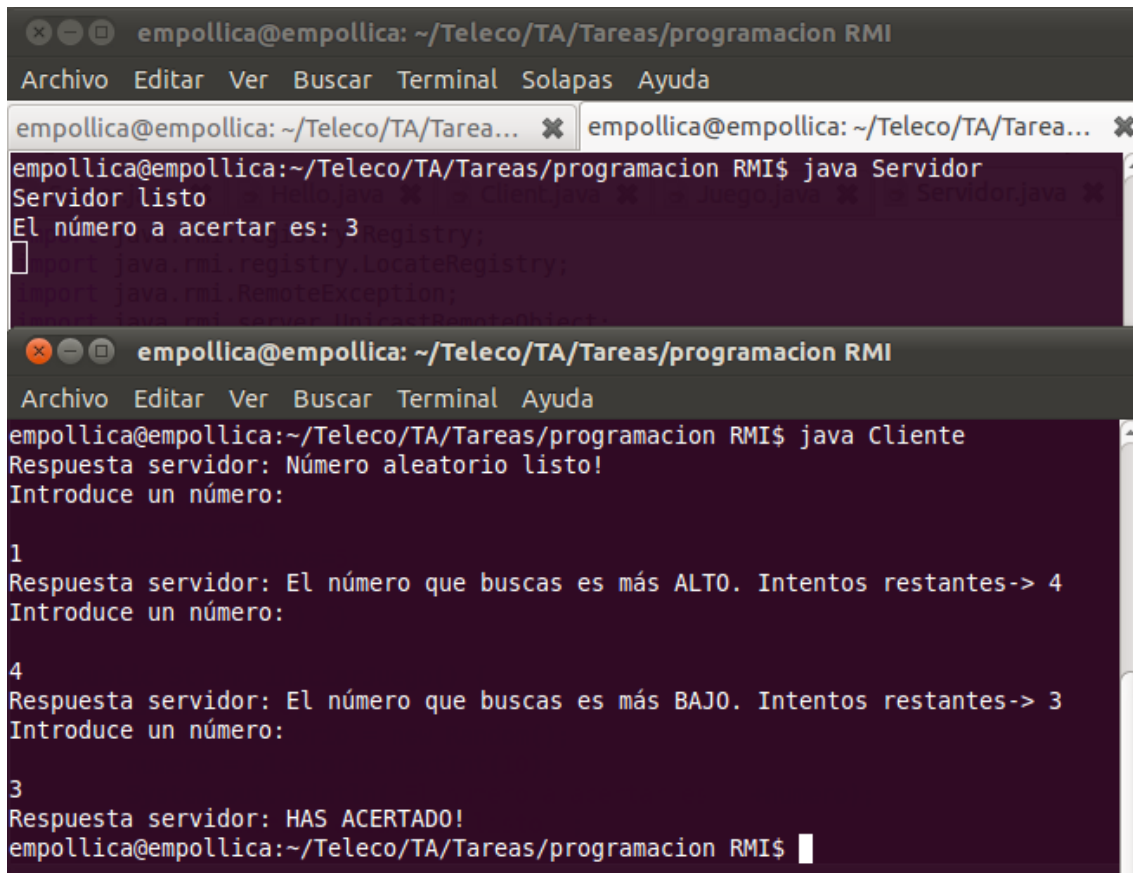


```
empollica@empollica: ~/Teleco/TA/Tareas/programacion RMI
Archivo Editar Ver Buscar Terminal Solapas Ayuda
empollica@empollica: ~/Teleco/TA/Tarea... x empollica@empollica: ~/Teleco/TA/Tarea... x
empollica@empollica:~/Teleco/TA/Tareas/programacion RMI$ java Servidor
Servidor listo
El número a acertar es: 3

empollica@empollica: ~/Teleco/TA/Tareas/programacion RMI
Archivo Editar Ver Buscar Terminal Ayuda
empollica@empollica:~/Teleco/TA/Tareas/programacion RMI$ java Cliente
Respuesta servidor: Número aleatorio listo!
Introduce un número:
```

Como puede verse, al lanzar el cliente se inicia el juego, generándose un número de forma aleatoria en el servidor. El cliente se mantiene a la espera de que el jugador introduzca un número a través de la consola.

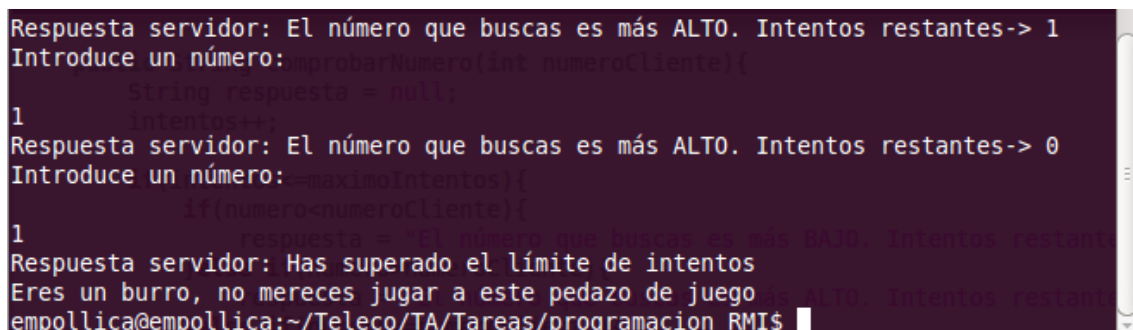
Los números que introduzca el cliente pueden ser mayores, menores o iguales al que se ha generado en el servidor (3 en este caso). De esta forma, según el número introducido, el servidor devuelve al cliente un mensaje indicando el resultado:



```
empollica@empollica: ~/Teleco/TA/Tareas/programacion RMI
Archivo Editar Ver Buscar Terminal Solapas Ayuda
empollica@empollica: ~/Teleco/TA/Tarea... x empollica@empollica: ~/Teleco/TA/Tarea... x
empollica@empollica:~/Teleco/TA/Tareas/programacion RMI$ java Servidor
Servidor listo
El número a acertar es: 3
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

empollica@empollica: ~/Teleco/TA/Tareas/programacion RMI
Archivo Editar Ver Buscar Terminal Ayuda
empollica@empollica:~/Teleco/TA/Tareas/programacion RMI$ java Cliente
Respuesta servidor: Número aleatorio listo!
Introduce un número:
1
Respuesta servidor: El número que buscas es más ALTO. Intentos restantes-> 4
Introduce un número:
4
Respuesta servidor: El número que buscas es más BAJO. Intentos restantes-> 3
Introduce un número:
3
Respuesta servidor: HAS ACERTADO!
empollica@empollica:~/Teleco/TA/Tareas/programacion RMI$
```

Como puede verse en la captura anterior, en caso de que se produzca un acierto el programa cliente finaliza, indicándose antes que se ha acertado el número. Además, cada vez que se introduce un número, el servidor indica también el número de intentos que restan. En caso de superar el número de intentos posibles el programa cliente finaliza con el siguiente resultado:



```
Respuesta servidor: El número que buscas es más ALTO. Intentos restantes-> 1
Introduce un número:
1
Respuesta servidor: El número que buscas es más ALTO. Intentos restantes-> 0
Introduce un número:
1
Respuesta servidor: Has superado el límite de intentos
Eres un burro, no mereces jugar a este pedazo de juego
Introduce un número:
empollica@empollica:~/Teleco/TA/Tareas/programacion RMI$
```