Capítulo 5

- **Introduciendo RMI**
- **Creación de Objetos Remotos**
- **Publicación de Objetos Remotos**
- **Acceso a Objetos Remotos**
- **Funcionamiento interno de RMI**
- **Compilación y Ejecución con RMI**
- **Pasaje de Parámetros en RMI**
- **Manejo de Concurrencia con RMI**

Introduciendo RMI

Remote Method Invocation

- Java es un lenguaje que provee diversos mecanismos para desarrollar aplicaciones distribuidas (esto es, que se ejecutan entre varios equipos). En este capítulo vamos a estudiar uno de esos mecanismos, conocido como **RMI (Remote Method Invocation)**.
- Dicho mecanismo consiste en permitirle a un objeto residente en una máquina invocar métodos de un objeto residente en otra máquina. A este último objeto comúnmente se lo denomina **Objeto Remoto**.

Introduciendo RMI

Protocolo TCP

- Existen diversos **protocolos** de red mediante los cuales dos programas pueden comunicarse. Los mecanismos existentes en Java para desarrollar aplicaciones distribuidas permiten utilizar dichos protocolos.

  Ejemplos de Protocolos:

  TCP, HTTP, UDP, FTP, POP, IMAP, etc.

- Concretamente, **RMI** utiliza el protocolo de red ***TCP*** el cual funciona (en forma general) de la siguiente manera:
  - Un programa residente en un equipo (programa **host,** también llamado **servidor**) espera que otros programas se conecten con él. Para ello, habilita un **número de puerto** por el cual aceptará conexiones.
  - Otros programas (desde otros equipos) establecen la conexión con el programa anterior. Para ello deben conocer el **nombre** o la **dirección IP** del equipo que lo alberga junto con el **número de puerto** habilitado.

Introduciendo RMI

Protocolo TCP (continuación)

- Para que este protocolo funcione, el programa **host** debe estar a la espera de recibir conexiones. Si un cliente intenta conectarse cuando el host **NO** está en ejecución, la conexión no se producirá. Una vez que los 2 programas están conectados, la información puede fluir entre ellos en ambos sentidos durante todo el tiempo que permanezca establecida la conexión.
- Muchos lenguajes permiten escribir aplicaciones bajo este protocolo. Incluso en Java hay varios mecanismos que lo usan (RMI es sólo uno de ellos).

¿ Cómo se aplica el protocolo TCP a RMI ?

- El **Objeto Remoto** es quien juega el papel de programa **host**.
- Desde otras máquinas, otros objetos acceden a él e invocan sus métodos (a efectos de pasarle o pedirle información). Para poder hacerlo deben conocer el **nombre** o la **dirección IP** del equipo que alberga al Objeto Remoto junto con el **número de puerto** bajo el cual está publicado.

Creación de Objetos Remotos

- Para implementar un Objeto Remoto debemos realizar los siguientes pasos:

  1) Escribir una **interface** que defina los métodos que tendrá el Objeto Remoto. Para ello debemos extender otra interface predefinida llamada **Remote** (definida en el paquete **`java.rmi`**)

  2) Escribir una **clase** que implemente los métodos de la interface anterior. Dicha clase debe extender otra clase predefinida llamada **UnicastRemoteObject** (definida en el paquete **`java.rmi.server`**)

- **Ejemplo:** Queremos definir un Objeto Remoto que represente a una Cuenta Bancaria. La misma debe brindar las siguientes operaciones:
  - Una operación que permita **depositar** dinero en la Cuenta
  - Una operación que permita **retirar** dinero de la Cuenta
  - Una operación que permita consultar el **saldo** de la Cuenta

Creación de Objetos Remotos

Ejemplo: Escribimos la Interfaz del Objeto Remoto para la Cuenta Bancaria

```
// ICuentaBancaria.java
package cuenta;

import java.rmi.Remote;
import java.rmi.RemoteException;

import excepciones.MontoInvalidoException;

public interface ICuentaBancaria extends Remote
{
    public void depositar (double monto) throws
           RemoteException, MontoInvalidoException;

    public void retirar (double monto) throws
           RemoteException, MontoInvalidoException;

    public double getSaldo () throws
           RemoteException;
}
```

Creación de Objetos Remotos

Ejemplo: Escribimos la Clase que implementa la Interfaz del Objeto Remoto

```
// CuentaBancaria.java
package cuenta;

import java.rmi.server.UnicastRemoteObject;
import java.rmi.RemoteException;

import excepciones.MontoInvalidoException;

public class CuentaBancaria extends UnicastRemoteObject
                            implements ICuentaBancaria
{   private double saldo;

    public CuentaBancaria() throws RemoteException
    {  saldo = 500;  }

    public void depositar (double monto) throws ...
    public void retirar (double monto) throws ...
    public double getSaldo () throws ...
}
```

Creación de Objetos Remotos

Ejemplo: Implementamos el método **depositar** de la Cuenta Bancaria

```
public void depositar (double monto) throws
                    RemoteException, MontoInvalidoException
{
     if (monto < 0)
     {
        String msg = "el monto no puede ser negativo";
        throw new MontoInvalidoException (msg);
     }

     saldo = saldo + monto;
}
```

Creación de Objetos Remotos

Ejemplo: Implementamos el método **retirar** de la Cuenta Bancaria

```
public void retirar (double monto) throws
                    RemoteException, MontoInvalidoException
{
     if (monto < 0)
     {
        String msg = "el monto no puede ser negativo";
        throw new MontoInvalidoException (msg);
     }

     if (monto > saldo)
     {
        String msg = "el monto no puede superar al saldo";
        throw new MontoInvalidoException (msg);
     }

     saldo = saldo - monto;
}
```

Creación de Objetos Remotos

Ejemplo: Implementamos el método **getSaldo** de la Cuenta Bancaria

```
public void getSaldo () throws RemoteException
{  return saldo;  }
```

Observaciones:

- Todos los métodos de la cuenta bancaria **deben** indicar en sus encabezados que pueden lanzar una **RemoteException**.
- Si en tiempo de ejecución llega a ocurrir algún error de comunicación, el método que esté siendo invocado lanzará dicha excepción.
- El programa que (desde otro equipo) esté invocando al método, atrapará la Remote Exception como forma de enterarse de lo ocurrido.
- Nótese que el método constructor de la clase CuentaBancaria **NO** se definió en la Interfaz del Objeto Remoto (ICuentaBancaria) ¿Por qué?

Publicación de Objetos Remotos

- Una vez programada la Clase que implementa los métodos de la Interfaz Remota, estamos en condiciones de crear el Objeto Remoto y **publicarlo**.
- Publicar un Objeto Remoto significa dejarlo accesible para que pueda ser accedido desde programas corriendo en otros equipos. Esto se hace utilizando un proceso de Java llamado **rmiregistry**.
- El rmiregistry de Java es un proceso de bajo nivel cuya función es mantener en memoria uno (o más) Objetos Remotos bajo un número de puerto, registrándolos bajo un nombre que los identifica.
- En resumen, los pasos para publicar un Objeto Remoto son los siguientes:

  1) Poner a correr el **rmiregistry** de Java en la misma máquina donde vamos a publicar el Objeto Remoto (luego veremos cómo)

  2) Crear el Objeto Remoto (instanciar la Clase que implementa los métodos de su Interfaz Remota).

  3) Pedirle al **rmiregistry** que publique nuestro Objeto Remoto.

Publicación de Objetos Remotos

Ejemplo: Escribimos un programa de prueba que crea y publica un Objeto Remoto de la clase CuentaBancaria.

```
// Servidor.java
package servidor;

import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;

import cuenta.CuentaBancaria;

public class Servidor
{
     public static void main (String [] args)
     {
        // instancio mi Objeto Remoto y lo publico
        // en el rmiregistry
     }
}
```

Publicación de Objetos Remotos

Ejemplo: Implementamos el método **main** de la Clase anterior:

```
public static void main (String [] args)
{
     try
     {  // pongo a correr el rmiregistry
        LocateRegistry.createRegistry(1099);
        // instancio mi Objeto Remoto y lo publico
        CuentaBancaria cuenta = new CuentaBancaria();
        System.out.println ("Antes de publicarlo");
        Naming.rebind("//pc00197:1099/cuenta", cuenta);
        System.out.println ("Luego de publicarlo");
     }
     catch (RemoteException e)
     {  e.printStackTrace(); }
     catch (MalformedURLException e)
     {  e.printStackTrace(); }
}
```

Acceso a Objetos Remotos

- Una vez publicado el Objeto Remoto estamos en condiciones de accederlo desde otros equipos.
- Para ello, no tenemos más que escribir un programa que (desde otra máquina) acceda al Objeto Remoto mediante el nombre o la dirección IP del equipo que lo alberga, el número de puerto bajo el cual está publicado y el nombre que lo identifica.

Ejemplo: Escribimos otro programa de prueba que accede al Objeto Remoto de la clase CuentaBancaria (desde otro equipo).

```
// Cliente.java
package cliente;
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.NotBoundException;
import cuenta.ICuentaBancaria;
import excepciones.MontoInvalidoException;
```

Acceso a Objetos Remotos

Ejemplo (continuación):

```
...
public class Cliente
{   public static void main (String [] args)
    {  try
       {  ICuentaBancaria cuenta = (ICuentaBancaria)
                  Naming.lookup("//pc00197:1099/cuenta");
          cuenta.depositar(1500);
          cuenta.retirar(500);
          System.out.println(cuenta.getSaldo());
       }
       catch (MalformedURLException e) ...
       catch (RemoteException e) ...
       catch (NotBoundException e) ...
       catch (MontoInvalidoException e) ...
    }
} // fin de la clase
```

Publicación y Acceso a Objetos Remotos

- **Observaciones:**
  - El método **createRegistry** de la Clase **LocateRegistry** se encarga de poner a correr el rmiregistry en el puerto pasado por parámetro.
  - El método **rebind** de la Clase **Naming** se encarga de pedirle al rmiregistry que publique un Objeto Remoto, mientras que el método **lookup** de la misma clase sirve para accederlo desde otro equipo.
  - En ambos casos, la sintaxis para indicar la ubicación del Objeto Remoto es la siguiente: `"//nombre_host:nro_puerto/nombre_objeto "`
  - Los programas que accedan (remotamente) al Objeto Remoto **no** pueden hacer referencia (en su código fuente) al nombre de la **Clase** que define al Objeto Remoto. Sólo pueden hacer referencia a la **Interface** Remota (en caso contrario, ocurre una excepción).
  - En los ejemplos presentados dejamos **hard – code** la ubicación del Objeto Remoto. Lo hicimos sólo por simplicidad de los ejemplos. Lo correcto sería obtener dichos datos del usuario o bien cargarlos desde un **archivo de configuración** (Ver Capítulo 3 del curso).

Funcionamiento Interno de RMI

- En esta sección describiremos un par de consideraciones relativas al funcionamiento interno de RMI que es necesario tener en cuenta a la hora de utilizar dicho mecanismo en nuestras aplicaciones.
- Cuando accedemos (remotamente) a un Objeto Remoto publicado en otra máquina, en el código fuente no notamos (salvo por el **lookup**) que el Objeto se encuentra realmente en otra máquina.
- De hecho, las invocaciones a los métodos del Objeto Remoto se hacen con la misma sintaxis que se harían si el Objeto fuera local.

Funcionamiento Interno de RMI

- Lo anterior ocurre porque el manejo específico de la **red** a la hora de comunicarnos (remotamente) con el Objeto Remoto está encapsulado en dos Objetos auxiliares que Java utiliza en forma conjunta con nuestro Objeto Remoto.
- Dichos Objetos auxiliares reciben el nombre de **Stub** y **Skeleton**. El Stub es el “representante” del Objeto Remoto en la máquina cliente, mientras que el Skeleton es el “representante” del Objeto Remoto en la máquina servidor (la que lo tiene publicado).
- Cuando invocamos algún método del Objeto Remoto, lo que ocurre internamente es lo siguiente:

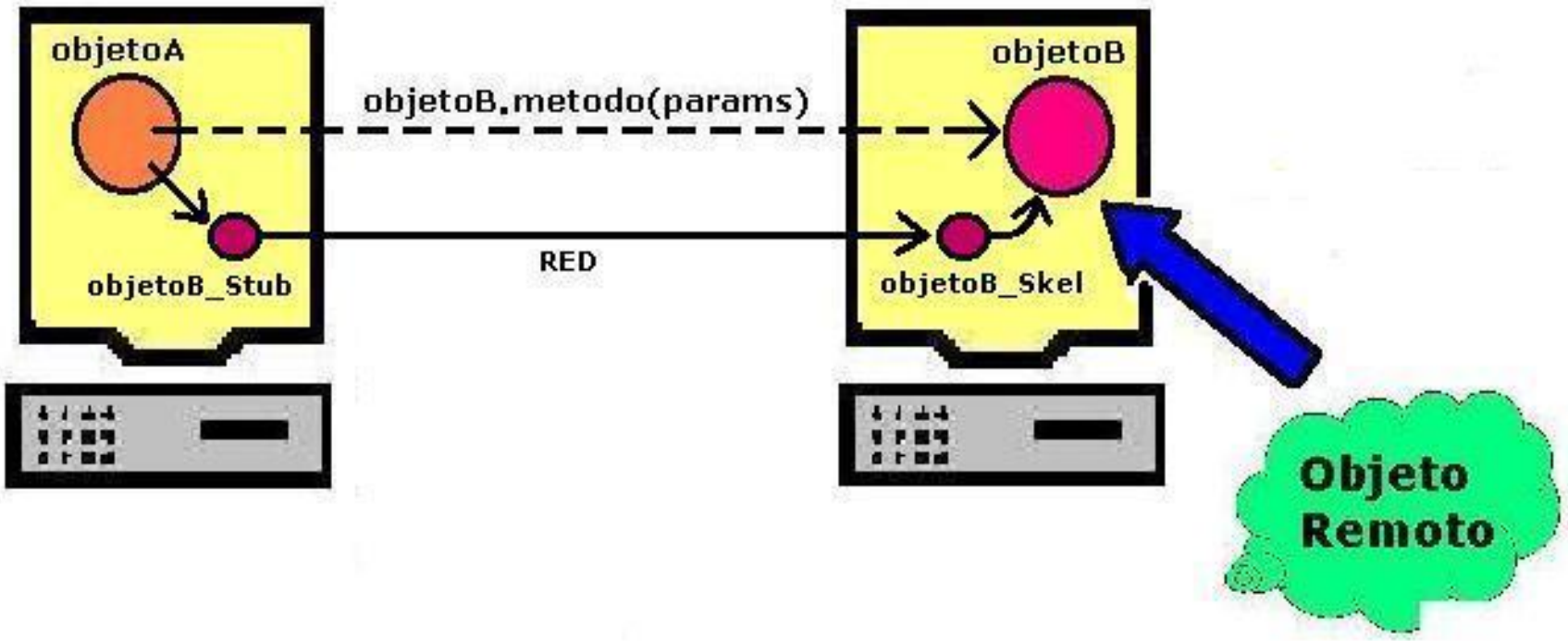
  1) La invocación es realizada sobre el Stub en la máquina cliente

  2) El Stub envía los parámetros al Skeleton a través de la red

  3) El Skeleton recibe los parámetros y se los pasa al Objeto Remoto, quien ejecuta realmente el método invocado (en el equipo Servidor).

  4) Los resultados son devueltos al cliente en orden inverso al anterior.

Funcionamiento Interno de RMI



- Las Clases que definen al **Stub** y al **Skeleton** de un Objeto Remoto son generadas por Java en forma ***dinámica***. Esto significa que el código fuente de las mismas es generado por Java en tiempo de ejecución.

Compilación y Ejecución con RMI

Desde un Entorno de Desarrollo (Eclipse)

1) Abrimos nuestro proyecto de trabajo normalmente en el equipo donde vamos a correr nuestro programa servidor.

2) Ejecutamos el programa servidor encargado de publicar el Objeto Remoto del mismo modo que usualmente ejecutamos cualquier programa en eclipse.

3) En el equipo donde vamos a ejecutar el programa cliente encargado de acceder (remotamente) al Objeto Remoto abrimos una **copia** de nuestro proyecto. Lo **único** que debemos hacer allí es ejecutar el programa cliente del mismo modo que usualmente ejecutamos cualquier programa en eclipse.

Observación: Podemos utilizar el **mismo** equipo (nombre `localhost`, o bien dirección IP `127.0.0.1`) como servidor y cliente mientras probamos la aplicación. En tal caso, podemos usar el **mismo** proyecto en Eclipse para ejecutar tanto el programa servidor como el programa cliente.

Pasaje de Parámetros en RMI

- Al igual que todo método, los métodos de un Objeto Remoto pueden recibir **Objetos** en sus parámetros y retornarlos como resultado.

Ejemplo: Imaginemos que a la clase **CuentaBancaria** le agregamos su dueño como un nuevo atributo. Asumimos ya implementada la clase **Dueño**. La interfaz remota **ICuentaBancaria** queda ahora así:

```
package cuenta;
import ...

public interface ICuentaBancaria extends Remote
{
   ... // mantenemos los métodos que ya teníamos

   public Dueño getDueño() throws RemoteException
   public void setDueño (Dueño d) throws RemoteException
   // métodos nuevos que retornan y setean el dueño de la
   // cuenta, respectivamente
}
```

Pasaje de Parámetros en RMI

Ejemplo: La clase **CuentaBancaria** queda ahora así:

```
// CuentaBancaria.java
package cuenta;
import ...

public class CuentaBancaria extends UnicastRemoteObject
                            implements ICuentaBancaria
{   private double saldo;
    private Dueño dueño;

    // aquí constructor, depositar, retirar y getSaldo
    ...

    public Dueño getDueño () throws RemoteException
    {  return dueño; }

    public void setDueño (Dueño d) throws RemoteException
    {  dueño = d; }
}
```

Pasaje de Parámetros en RMI

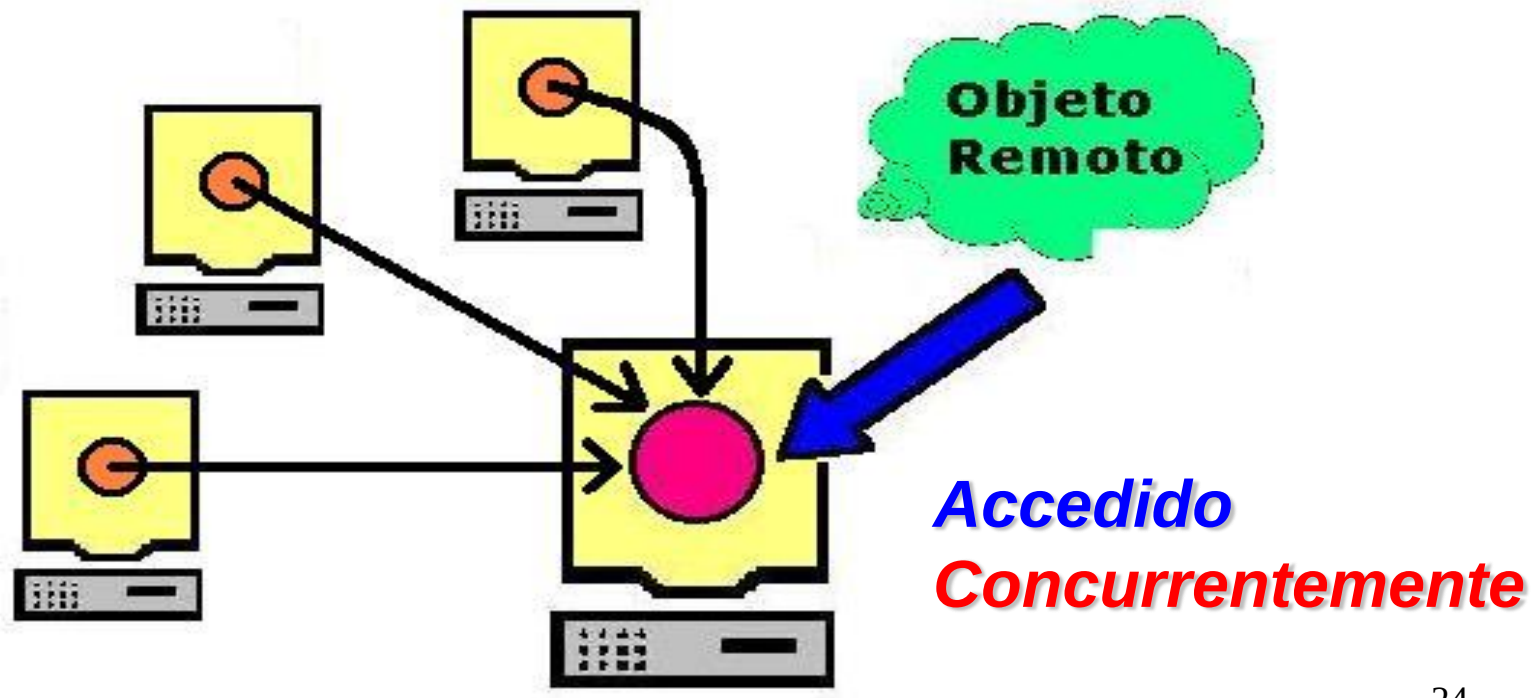
- Los métodos del Objeto Remoto son invocados desde **otras** máquinas. Esto significa que los objetos que se pasen como parámetros o devueltos como resultados son transferidos a través de un flujo de comunicación (la **red**). Para que esto funcione, los objetos transferidos por dicho flujo deberán ser **serializables** (como se vio en el capítulo 3).
- En el ejemplo de la cuenta bancaria, el dueño retornado por el método `getDueño` y el que se pasa por parámetro en el método `setDueño` son transferidos a través de la red. Para que esto sea posible, la clase `Dueño` debe ser **serializable**. De otro modo, ocurrirá una **excepción**.

```
public class Dueño implements Serializable
```

- Lo que sucederá en tiempo de ejecución es que el método `getDueño` retornará una **copia serializada** del dueño al programa cliente. Si éste lo modifica, esto **no** se verá reflejado en el dueño original almacenado en el servidor. El método `setDueño` recibirá también una **copia serializada** del dueño en el programa servidor. Si éste lo modifica, esto **no** se verá reflejado en el dueño original que fue enviado desde el programa cliente.

Manejo de Concurrencia con RMI

- Cuando publicamos un Objeto Remoto, lo hacemos con la finalidad de que el mismo pueda ser accedido desde programas corriendo en otros equipos.
- No obstante, podría suceder que varios programas quieran acceder a él en forma **concurrente**. En tal caso, el Objeto Remoto será **compartido** por todos esos programas que tratarán de invocar sus métodos en paralelo.



Manejo de Concurrencia con RMI

- Debemos entonces garantizar que el acceso en paralelo al Objeto Remoto se haga de forma controlada. Para ello tenemos que dotar al mismo de algún mecanismo de ***Control de concurrencia*** (al igual que hacemos en cualquier ambiente concurrente, involucre RMI o no).
- Como en cualquier ambiente concurrente, hay distintos niveles de control que se pueden manejar para el Objeto Remoto. Por ejemplo:
  - ***Sincronizar*** todos los métodos del Objeto. Este mecanismo es el más sencillo, pero también el más restrictivo. ¿Por qué?
  - Manejar la concurrencia ***internamente*** en cada método mediante algún mecanismo apropiado (quizás un ***Monitor de Lectura – Escritura***)
  - Manejar la concurrencia desde los programas cliente en forma previa a acceder al Objeto. En tal caso debemos tener cuidado de que **todos** los programas cliente apliquen esta estrategia.
- La elección del nivel de concurrencia a manejar sobre un objeto compartido (sea Remoto o no) dependerá de cada problema y diseño concreto.

Manejo de Concurrencia con RMI

Ejemplo: Sincronizamos los métodos de la Clase **CuentaBancaria**. Cuando un programa cliente invoque remotamente algún método del Objeto Remoto, todos los demás deberán **esperar** que el programa salga del método y libere al objeto.

```
public class CuentaBancaria extends ... implements ...
{    ...
     public CuentaBancaria() throws ...
     public synchronized void depositar(...) throws ...
     public synchronized void retirar(...) throws ...
     public synchronized double getSaldo() throws ...
     public synchronized Dueño getDueño() throws ...
     public synchronized void setDueño(...) throws ...
}
```

Observación: Cuando un programa cliente quiere acceder al Objeto Remoto, en el **rmiregistry** se lanza internamente un **Thread** (hilo) que se encarga de invocar a los métodos del **Skeleton** del Objeto Remoto. O sea, aunque **no** los vemos directamente, hay threads ejecutándose sobre el Objeto Remoto.