Aspectos avanzados de Java: RMI, JDBC y Extensiones Multimedia

Dr. Antonio LaTorre

e-mail: atorre@fi.upm.es

Índice

- Java RMI
- JDBC
- Extensiones Multimedia

Java RMI

Java RMI

- RMI = **R**emote **M**ethod **I**nvocation
- Es la implementación de Java de RPC (Remote Procedure Call)

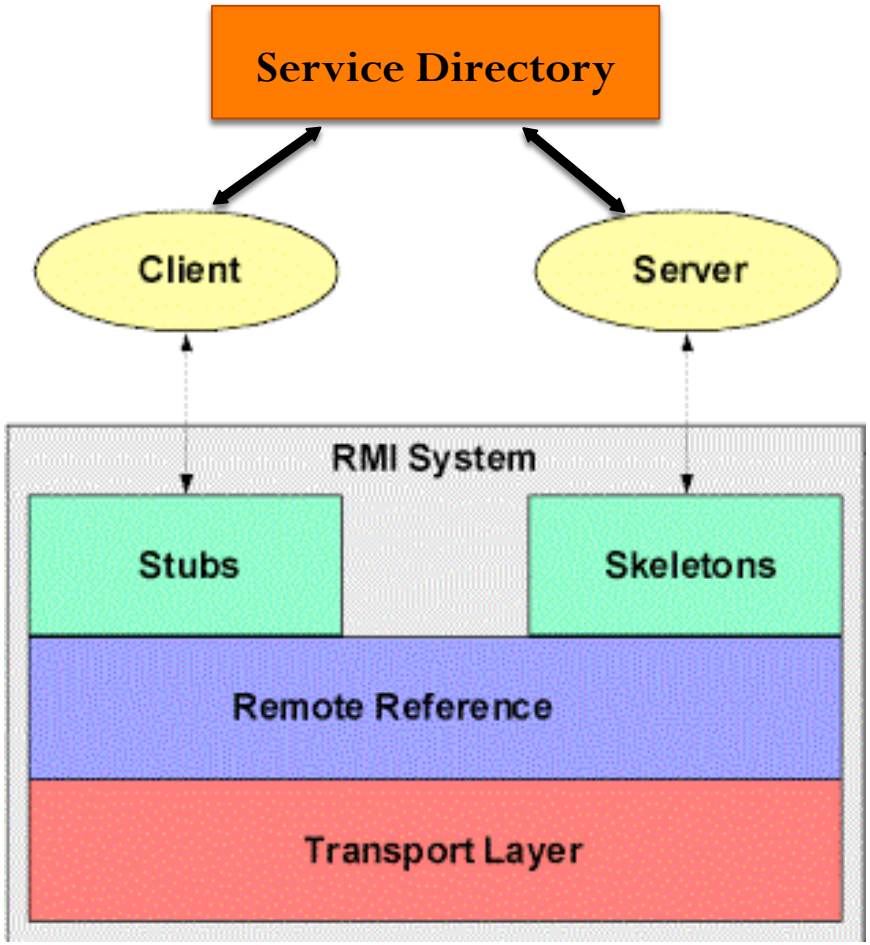
- Permite invocar a métodos de objetos remotos a través del protocolo TCP/IP
  - Objetos en la misma máquina
  - Objetos en máquinas distintas

- Arquitectura Cliente/Servidor
  - Ambos conocen y comparten una interfaz
  - Comunicación a través de la red transparente

Características de Java RMI

- Uso de un Middleware: capa de abstracción para la comunicación entre procesos remotos. Oculta:
  - Localización de objetos
  - Protocolos de comunicación
  - Transferencia de datos
  - Hardware subyacente
  - Sistema Operativo
- Recogida de basura distribuida
- Orientado a acciones
  - Hace hincapié en la invocación de métodos

Arquitectura RMI



Arquitectura de un programa RMI

- Interfaz
- Implementación
  - Stub
  - Skeleton
- Servidor
- Cliente
- Servidor de Nombres RMI

Interfaz

- Conjunto de métodos que serán implementados por el objeto remoto y que pueden ser accedidos por el cliente
  - Especificación de nombre de los métodos y argumentos
  - **No** se proporciona una implementación

- La interfaz debe extender **java.rmi.Remote** y ser pública

- Todos los métodos declarados deben poder lanzar la excepción **java.rmi.RemoteException**

Implementación

- Es una clase Java normal
  - Extiende la clase **java.rmi.server.UnicastRemoteObject**
  - Implementa el Interfaz definido anteriormente
    - Debe proporcionar una implementación para todos los métodos
- En el constructor de esta clase hay que llamar al constructor de la clase **UnicastRemoteObject**
  - Hay que usar *super()*

Stub y Skeleton

- Son clases intermedias que abstraen de la comunicación por red entre cliente y servidor

Cliente ⟷ Stub ⟷ [Red] ⟷ Skeleton ⟷ Servidor

- Implementan la misma interfaz que *"Interfaz"*

- Antes de Java 1.2
  - rmic Implementación
- Ahora no es necesario…
  - Skeleton se crea automáticamente a partir de Java 1.2
  - Stub se crea automáticamente a partir de Java 5

Servidor

- Crea el objeto que será accedido remotamente
  - Instancia el objeto que implementa la interfaz

- Registra el objeto remoto (asignándole un nombre) en un Servidor de Nombres RMI
  - Sin reemplazo: **Naming.bind()**
  - Con reemplazo: **Naming.rebind()**

    rmi://<nombre máquina>:<puerto>/<nombre referencia>

- El servidor puede implementarse en una clase aparte o dentro de la clase Implementación
  - La instanciación y el registro habría que hacerlos en el ***main***

Cliente

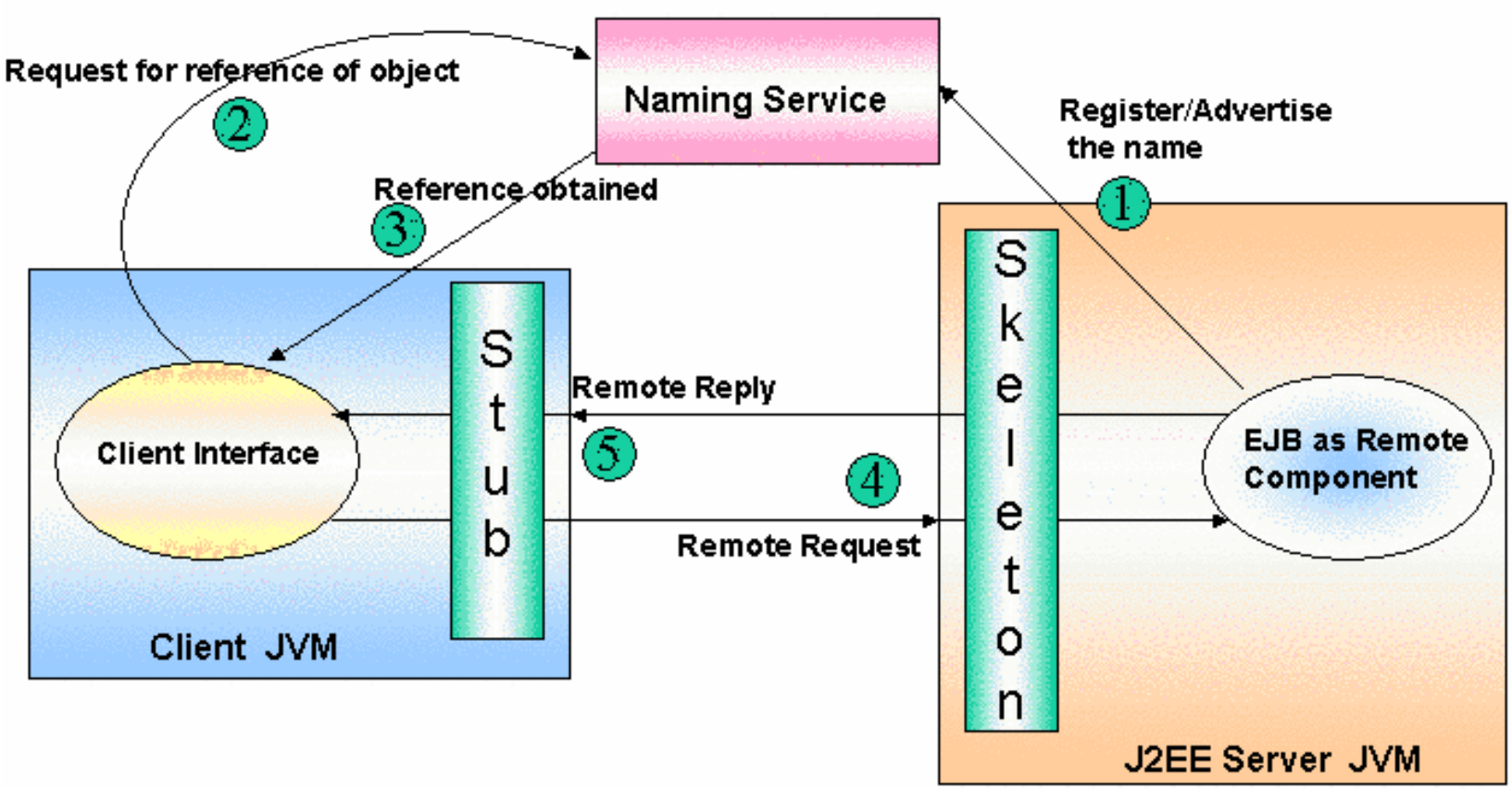
- Obtiene una referencia al objeto remoto con el que desea conectar
  - Realiza una petición a un Servidor de Nombres RMI
    - Método **Naming.lookup()** pasando como argumento el nombre con el que se registró el objeto remoto (en el mismo formato cualificado)

- Una vez recibido el objeto remoto, se pueden invocar sus métodos como si fuera un objeto local
  - Los objetos pasados como argumento recibidos como valor de retorno son **serializados** (proceso conocido como *marshalling*)

Servidor de Nombres RMI

- Repositorio centralizado de objetos remotos
  - Los servidores, los registran
  - Los clientes, los recuperan
- Puede iniciarse de dos maneras
  - Desde consola: **rmiregistry [puerto]**
  - Desde el código del servidor **LocateRegistry.createRegistry()**
- Al crearlo hay que decirle en qué puerto debe escuchar (por defecto, el 1099)
- El cliente puede acceder a él de dos formas
  - A través de **Naming**
  - Mediante una instancia del registro: **LocateRegistry.getRegistry()**

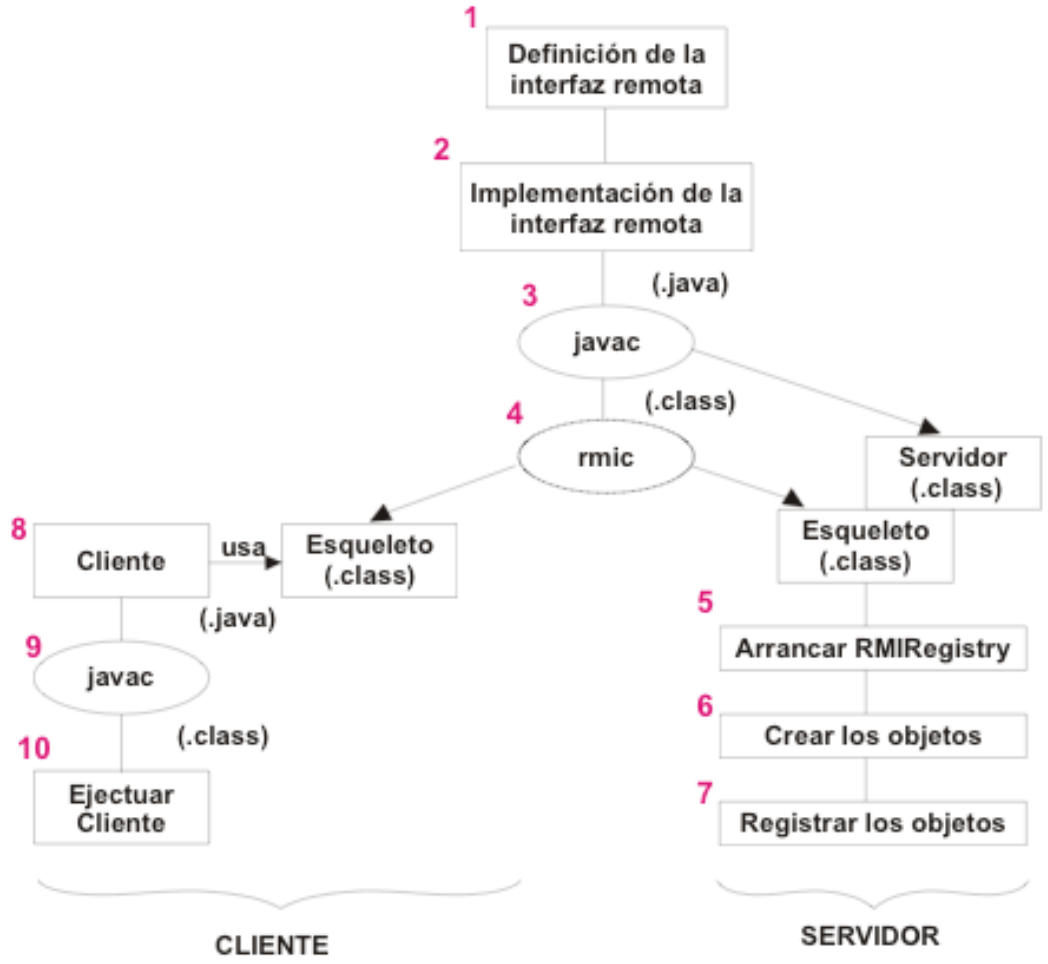
Arquitectura de un programa RMI



Programando con RMI

- Paquetes necesarios
  - java.rmi
  - jama.rmi.server
- Los métodos de la interfaz y su implementación deben lanzar *RemoteException*
- Siempre que se invoque a un método remoto o un método del servicio de nombres hay que capturar sus excepciones
  - Errores en la comunicación entre los procesos (fallos de acceso o de conexión)
  - Fallo en la invocación del objeto remoto (objeto no disponible)
  - Fallo en el registro de un objeto
  - Etc.

Desarrollo de aplicaciones RMI



Ejemplo: Hola Mundo (Interfaz)

```
package hello;

import java.rmi.*;

public interface HelloInterface extends Remote {

    public void sendMessage(String msg) throws
  RemoteException;
    public String getMessage() throws RemoteException;

}
```

Ejemplo: Hola Mundo (Implementación)

```
package hello;

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

public class HelloImpl extends UnicastRemoteObject implements
   HelloInterface {
    public HelloImpl() throws RemoteException {
         super();
    }
    public void sendMessage(String msg) throws RemoteException {
         System.out.println(msg);
    }
    public String getMessage() throws RemoteException {
         return "Hello from the server";
    }
}
```

Ejemplo: Hola Mundo (Servidor)

```
package hello;

import java.net.*;
import java.rmi.*;
import java.rmi.server.*;

public class HelloServer {

// ...
```

Ejemplo: Hola Mundo (Servidor)

```
    //...
    public static void main(String args[]) {
        try {
                HelloImpl obj = new HelloImpl();
                Naming.bind("HelloServ", obj);
        }
        catch (RemoteException e) {
                System.out.println("RemoteException   " + e);
        }
        catch (AlreadyBoundException e) {
                System.out.println("AlreadyBoundException");
        }
        catch (MalformedURLException e) {
                System.out.println("MalformedURLException");
        }
    }
}
```

Ejemplo: Hola Mundo (Cliente)

```
package hello;

import java.net.*;
import java.rmi.*;

public class HelloClient {

    static public void main(String args[]) {

        try {
                HelloInterface obj = (HelloInterface)
                Naming.lookup("HelloServ");
                obj.sendMessage("Hello from the client");
                System.out.println(obj.getMessage());
        }

        //...
```

Ejemplo: Hola Mundo (Cliente)

```
            //...

            catch (RemoteException e) {
                    System.out.println("RemoteException  " + e);
            }
            catch (NotBoundException e) {
                    System.out.println("AlreadyBoundException");
            }
            catch (MalformedURLException e) {
                    System.out.println("MalformedURLException");
            }

    }

}
```

Ejemplo: Hola Mundo – Puesta en marcha

- **Compilación**

  javac hello/*.java
- **Generación del Stub (Opcional en Java 5+)**

  rmic hello.HelloImpl
- **Ejecución del Servicio de Nombres RMI**

  rmiregistry <puerto>

  Puerto por defecto: 1099
- **Ejecución del servidor**

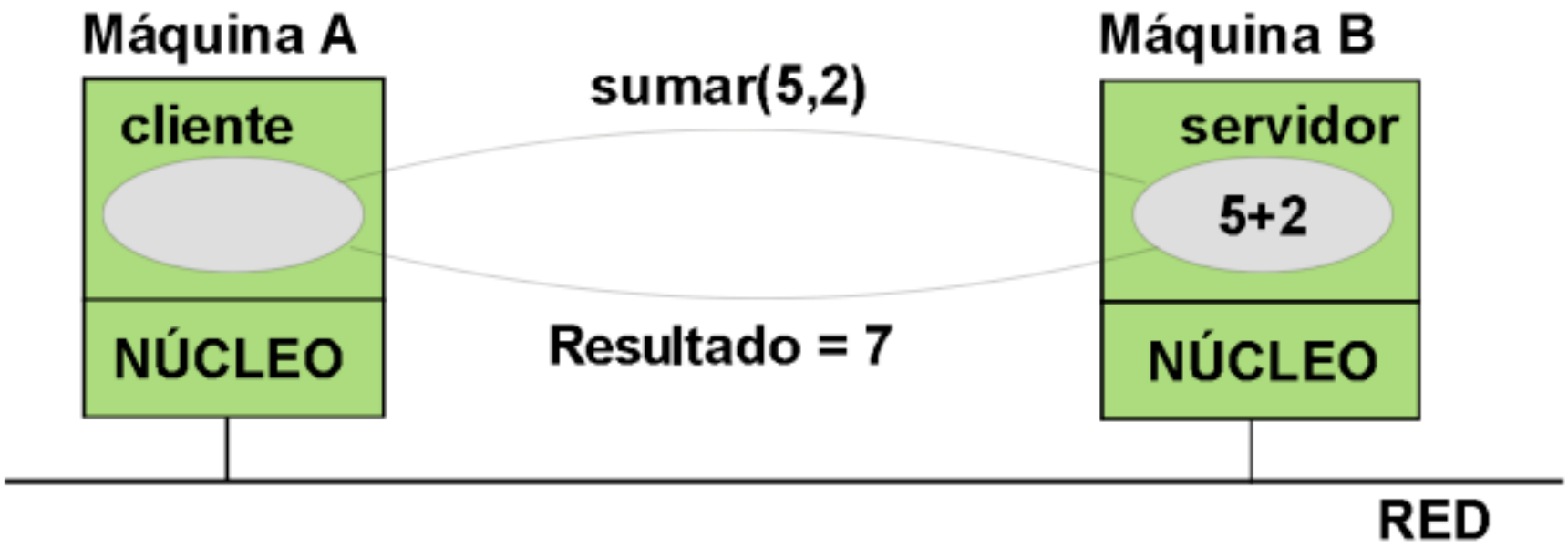
  java hello.HelloServer
- **Ejecución del cliente**

  java hello.HelloClient

Ejercicio

- Crear una aplicación que implemente una calculadora
- La ventana tendrá dos cajas de texto para los operandos y otra para mostrar el resultado
- La operación a realizar se seleccionará de la manera que nos parezca más oportuna (ComboBox, Buttons, RadioButtons, etc.)
- Sólo se podrá seleccionar una operación si los operandos son válidos
- La operación debe realizarse en un servidor local
- La interfaz tendrá un método por cada operación válida
- El cliente invocará a dichos métodos a través de una instancia
- El servidor realizará la operación correspondiente y devolverá el resultado

Ejercicio


Máquina A
cliente
NÚCLEO
sumar(5,2)
Resultado = 7
Máquina B
servidor
5+2
NÚCLEO
RED

Paso de parámetros: Serialización

- La serialización consiste en traducir un objeto en una lista de bytes
  - Almacenamiento (persistencia)
  - Transmisión (invocación de procedimientos remotos)
- Procesado en ambos extremos: serialización y deserialización
- ¿Cómo se pasan los distintos tipos de datos?
  - Tipos primitivos: por valor
  - Objetos: por valor (en lugar de por referencia, como se haría normalmente?
  - Objeto remoto exportado: se transmite el stub del objeto

Paso de parámetros: Serialización

- Un objeto puede ser simple…
- … o estar compuesto de otros objetos o tipos primitivos
- … o que extienda a una clase base
  - Es necesario que todos los atributos de un objeto y las clases de las que hereda sean serializables

- ¿Cómo se consigue esto?
  - Los objetos deben implementar la interfaz **Serializable**
    - No define ningún método
    - Es una "marca" que indica que el objeto puede ser convertido en una lista de bytes
    - Si se desea, se pueden redefinir los métodos de serialización
  - Los tipos primitivos y otros objetos de Java ya son serializables

Redefiniendo los métodos de serialización

- Hay que implementar los siguientes métodos

```
private void writeObject (ObjectOutputStream stream) throws IOException
   {
   stream.defaultWriteObject();
   ...
}

private void readObject (ObjectInputStream stream) throws IOException {
   stream.defaultReadObject();
   ...
}
```

- Es importante respetar la signatura y, recomendado, llamar a los serializadores por defecto
- Si los redefinimos en una jerarquía de clases, en cada clase sólo hay que serializar lo correspondiente a dicha clase

Ejercicio

- Escribir un programa que trabaje con Personas y Estudiantes
- La clase **Persona** almacena nombre, apellidos y edad
- La clase **Estudiante** extiende la clase Persona y almacena un vector de *Asignaturas*
- Crear un objeto remoto que devuelva objetos de tipo Persona y Estudiante por medio de sendos métodos
- Imprimir en el cliente las personas y estudiantes devueltos

- Recordatorio: Los objetos a enviar deben ser serializables…

Servidor de Nombres RMI

- RMI puede utilizar distintos servicios de nombres:
  - JNDI (Java Naming and Directory Interface)
  - **RMI Registry**: servicio sencillo incluido con RMI

- Interfaz remota Registry:
  - Métodos lookup(), bind(), rebind(), unbind() y list()
  - Asocia objetos a cadenas de caracteres

- Clase LocateRegistry
  - Permite crear y recuperar objetos que implementan Registry
- Clase Naming
  - Invoca métodos de un objeto remoto que implementa Registry

Interfaz Registry

- Éstos son sus métodos más importantes:

  Registrar un objeto sin reemplazo (excepción si ya existe):

  void bind(String name, Remote obj)

  Registrar un objeto con reemplazo:

  void rebind(String name, Remote obj)

  Buscar un objeto en el servidor:

  Remote lookup(String name)

  Des-registrar un objeto (excepción si no existe):

  void unbind(String name)

  Recuperar la lista de objetos registrados:

  String[] list (String name)

Clase LocateRegistry

- Antes vimos que el registro de nombres podía iniciarse desde la línea de mandatos: **rmiregistry**
- Sin embargo, se puede crear desde un programa Java directamente
- Clase **LocateRegistry**
  - Método ***createRegistry(int port)***
    - Crea un servidor de nombres en el puerto especificado
  - Métodos ***getRegistry(String host, int port)***
    - Devuelve una referencia al registro localizado en ese host y puerto
    - Si algún valor no se especifica, se usan valores por defecto (*localhost* y *1099*)
    - Devuelve un objeto de tipo **Registry** (misma interfaz que Naming)

Clase Naming

- Permite acceder a los métodos de un objeto remoto que implementa la interfaz Registry directamente
- No hace falta recuperar una instancia del objeto

Ejercicio

- Modificar el programa Hola Mundo de ejemplo para que use la clase LocateRegistry
- El servidor tendrá ahora que crear su propio servidor de nombres (createRegistry)
- El cliente recuperará el registro (getRegistry) que ejecuta en la IP y puerto donde lo creó el servidor
- El cliente utilizará la interfaz del objeto devuelto para realizar la búsqueda del objeto remoto

Ejercicio

- Modificar el ejercicio anterior para probar los métodos de la interfaz Registry que no habíamos probado hasta ahora
  - bind y rebind: ¿qué diferencia hay? ¿qué excepciones lanzan uno y otro?
  - list: desde el cliente, imprimir la lista de objetos registrados en el servidor de nombres
  - unbind: des-registrar el objeto remoto desde el servidor. ¿Qué pasa si el cliente intenta acceder a dicho objeto más tarde?

Método exportObject()

- Método de la clase **UnicastRemoteObject**
- Permite exportar un objeto

RMI Security Manager

- Gestor de seguridad para aplicaciones RMI
- Controla qué se puede hacer y quién puede hacerlo
  - Conectar a determinados puertos
  - Conectar desde determinadas IPs
  - Escribir ficheros

  http://java.sun.com/j2se/1.5.0/docs/guide/security/permissions.html
- Se puede incluso establecer permisos por clase
- Sin embargo…
  - Difícil de configurar
  - Configuración por defecto bastante restrictiva
  - Muchas veces se deja abierto por lo laborioso de configurar
  - Interesante si queremos habilitar la carga de clases remotas

RMI Security Manager

- Se define en java.rmi.RMISecurityManager
- En el código del programa, hay que incluir

```
if (System.getSecurityManager() == null)
      System.setSecurityManager(new RMISecurityManager());
```

- Ahora habría que decirle dónde se encuentra el fichero de configuración

  java -Djava.security.policy=D:/directorio/java.policy program

El fichero java.policy

- Fichero de sintaxis muy completa (y compleja)
- Permite configurar permisos con un hilo muy fino

```
grant {
   permission java.net.SocketPermission "*:1024-65535",
   "connect,accept";
   permission java.io.FilePermission "c:\\home\\ann\\public_html\
   \classes\\-", "read";
   permission java.io.FilePermission "c:\\home\\jones\\public_html\
   \classes\\-", "read";
};
```

- Si queremos dar todos los permisos

```
grant {
   permission java.security.AllPermission;
};
```

Ejercicio

- Escribir un programa cliente/servidor
- Hacer que el cliente use un RMISecurityManager
- Asociarle un fichero de política vacío. Observar lo que pasa.
- Garantizar permisos para conectar a los puertos 1024-65535
- Ahora, añadir al cliente la funcionalidad de escribir un fichero en el directorio actual. Observar lo que pasa.
- Garantizar permisos para escribir en dicho directorio en el fichero de política de seguridad

Carga dinámica de clases

- En una arquitectura cliente/servidor normalmente los programas se distribuyen por separado
  - Problema: comparten clases
  - Solución: almacenar esas clases en un repositorio y cargarlas dinámicamente

- Otras veces interesa poder definir una interfaz y hacer que los clientes puedan enviar objetos que implementen esa interfaz al servidor y viceversa

Carga dinámica de clases

- Para poder hacerlo, necesitamos usar un RMISecurityManager (como ya hemos visto)

- Además, hay que especificar la propiedad "*codebase*", que es de donde se descargarán las clases

java -Djava.rmi.server.codebase=file:/D:/directorio/ program

java –Djava.rmi.server.codebase=http://host/directorio/ program

Carga dinámica de clases

- Por último, hay que establecer una política de seguridad sobre qué pueden hacer las clases descargadas desde esa ubicación

```
grant CodeBase "file:/D:/direcorio/" {
  permission java.security.AllPermission;
};
```

- No hay que olvidar pasarle la ruta con el fichero de seguridad

```
java -Djava.security.policy=D:/directorio/
  java.policy -Djava.rmi.server.codebase=file:/D:/
  directorio/ program
```

Carga dinámica de clases

- A partir de ahora, podemos hacer que un método remoto reciba como argumento un objeto que implemente otra interfaz y que sea descargable desde la ubicación establecida

```
public interface InterfaceRemota extends Remote {
    public int suma (InterfaceSumandos sumandos) throws
  RemoteException;
}
```

- La interfaz *InterfaceSumandos* podría ser

```
public interface InterfaceSumandos extends Serializable {
    public int getSumando1();
    public int getSumando2();
}
```

Ejercicio

- Modificar el programa Hola Mundo que hicimos anteriormente para que el servidor y el cliente ejecuten en máquinas distintas
  - Modificar en el cliente la URL donde buscar el objeto remoto
  - ¿Qué sucede?
- Crear un fichero de reglas adecuado que permita al cliente acceder a la IP del servidor
  - ¿Funciona ahora?

Ejercicio

- Crear un programa donde:
  - Cliente y servidor estarán en directorios distintos
  - Los ficheros *ObjetoRemotoInterfaz* y *OperacionInterfaz* deben estar en ambos directorios (y ser iguales)
  - El servidor publica un objeto remoto cuya interfaz tiene un método del tipo: *int realizarOperacion(OperacionInterfaz op)*
  - La interfaz *OperacionInterfaz* tendrá un método *calcular()* que realizará una determinada operación con dos valores que recibirá en el constructor
  - La implementación concreta de la operación será una clase que sólo estará en el directorio del cliente

Ejercicio (continuación...)

- El servidor debe estar configurado para poder descargar clases de una URL (o del directorio del cliente, lo que se prefiera)
- Hay que restringir lo que las clases cargadas desde ese directorio puede hacer
  - Probar a dar y quitar permisos de escritura en un directorio y probar qué pasa si el método *calcular* intenta escribir en él
- La estructura sería:
  - Servidor: Servidor.java, ObjetoRemotoInterfaz.java, OperacionInterfaz.java, ObjetoRemotoImpl.java
  - Cliente: Cliente.java, ObjetoRemotoInterfaz.java, OperacionInterfaz.java, Operacion1.java, Operacion2.java, etc.

Callbacks de clientes

- ¿Qué pasa si queremos que el servidor pueda comunicarse con el cliente (p.ej., para notificarle algún evento)
  - Problema: llamada a método remoto es unidireccional
- Posibles soluciones:
  - ***Polling***: El usuario invoca otro método del servidor hasta que le devuelva un valor determinado. Muy costoso.
  - **Callback:** Cada cliente interesado en un evento registra en el servidor un objeto a través del cual el servidor puede invocar un método para notificarle al cliente cuando ocurra dicho evento.
    - Ahora tenemos canales bidireccionales

Callbacks

- El cliente define una interfaz como ésta

```
public interface CallbackInterface extends Remote {
   public String notifyme(String msg) throws RemoteException;
}
```

- Hay que proporcionar una implementación para esta interfaz: ***CallbackImpl***
- En la clase cliente hay que añadir código para instanciar un callback y registrarlo en el servidor (a través de un método que habremos de implementar)

```
Server srv = Naming.lookup("Server");
CallbackImpl callback = new CallbackImpl();
server.registerCallback(callback);
```

Callbacks

- En el objeto remoto del servidor, habría que añadir un método para registrar los callbacks de los clientes

```
public void registerCallback(CallbackInterface callback)
   throws RemoteException;
```

- Se puede proporcionar el método análogo para eliminar un callback: *removeCallback(CallbackInterface callback)*
- Ambos métodos guardarán y borrarán las referencias a los callbacks de los clientes en una estructura común (por ejemplo, un Vector)

Ejercicio: JMessenger

JDBC

Conceptos de Bases de Datos

- Base de datos
  - Conjunto de datos interrelacionados
  - Almacenados sistemáticamente
- Gestor de base de datos
  - **Programa** informático
  - Almacena los datos de forma eficiente
  - Recupera y accede a los datos rápidamente
- SQL – Structured Query Language
  - **Lenguaje estándar** para consultar una base de datos
  - Es un estándar ANSI/ISO

Conceptos de Bases de Datos

- Tabla
  - Conjunto de datos relacionados
  - Indexados mediante una “clave”
    - La clave debe ser única
  - La información puede ser de distinto tipo

- Relación entre elementos de distintas tablas
  - Proporcionan coherencia y elimina la redundancia de la BBDD
  - Se consigue mediante el uso de “claves foráneas o ajenas”

Ejemplo de Tabla (Empresas)

Código	Nombre	Tipo	Teléfono	Saldo
8788	Suministros Ruiz	Electricidad	91 3987872	1288.00
8789	Transportes TSS	Transporte	91 5993725	433.00
8790	Segurinsa S.A.	Seguridad	923 847002	-120.50
8791	Desarrollos Tyna	Diseño	91 4513172	3901.00
8792	Elektronics	Electricidad	91 3342244	-233.50
8793	Obras Marcos	Albañilería	91 7111132	2921.00
8794	Muebles Prados	Mobiliario	91 3120072	90.55
8795	Internal	Mobiliario	93 4287910	340.00
8796	Trabajos Cruz	Fontanería	672 112991	790.55

Consultando información: SELECT

SELECT columnas FROM tabla;

- columnas: Lista de columnas a presentar (separadas por comas). Puede ser un asterisco (*) para indicar todas las columnas
- tabla: Nombre de una tabla

- Ejemplo:

  SELECT Nombre,Telefono FROM Empresas;

Consultando información: SELECT

SELECT Nombre, Telefono FROM Empresas;

Código	Nombre	Tipo	Teléfono	Saldo
8788	Suministros Ruiz	Electricidad	91 3987872	1288.00
8789	Transportes TSS	Transporte	91 5993725	433.00
8790	Segurinsa S.A.	Seguridad	923 847002	-120.50
8791	Desarrollos Tyna	Diseño	91 4513172	3901.00
8792	Elektronics	Electricidad	91 3342244	-233.50
8793	Obras Marcos	Albañilería	91 7111132	2921.00
8794	Muebles Prados	Mobiliario	91 3120072	90.55
8795	Internal	Mobiliario	93 4287910	340.00
8796	Trabajos Cruz	Fontanería	672 112991	790.55

Consultando información: SELECT

SELECT Nombre, Telefono FROM Empresas;

Código	Nombre	Tipo	Teléfono	Saldo
8788	**Suministros Ruiz**	Electricidad	**91 3987872**	1288.00
8789	**Transportes TSS**	Transporte	**91 5993725**	433.00
8790	**Segurinsa S.A.**	Seguridad	**923 847002**	-120.50
8791	**Desarrollos Tyna**	Diseño	**91 4513172**	3901.00
8792	**Elektronics**	Electricidad	**91 3342244**	-233.50
8793	**Obras Marcos**	Albañilería	**91 7111132**	2921.00
8794	**Muebles Prados**	Mobiliario	**91 3120072**	90.55
8795	**Internal**	Mobiliario	**93 4287910**	340.00
8796	**Trabajos Cruz**	Fontanería	**672 112991**	790.55

Consultando información: SELECT

SELECT columnas FROM tabla WHERE condición;

- condición: Una expresión (comparación) que afecta a alguna de las columnas de la tabla

- Ejemplo:

  SELECT * FROM Empresas WHERE Saldo < 0;

Consultando información: SELECT

```
SELECT * FROM Empresas WHERE Saldo < 0;
```

Código	Nombre	Tipo	Teléfono	Saldo
8788	Suministros Ruiz	Electricidad	91 3987872	1288.00
8789	Transportes TSS	Transporte	91 5993725	433.00
8790	Segurinsa S.A.	Seguridad	923 847002	-120.50
8791	Desarrollos Tyna	Diseño	91 4513172	3901.00
8792	Elektronics	Electricidad	91 3342244	-233.50
8793	Obras Marcos	Albañilería	91 7111132	2921.00
8794	Muebles Prados	Mobiliario	91 3120072	90.55
8795	Internal	Mobiliario	93 4287910	340.00
8796	Trabajos Cruz	Fontanería	672 112991	790.55

Consultando información: SELECT

```
SELECT * FROM Empresas WHERE Saldo < 0;
```

Código	Nombre	Tipo	Teléfono	Saldo
8788	Suministros Ruiz	Electricidad	91 3987872	1288.00
8789	Transportes TSS	Transporte	91 5993725	433.00
8790	Segurinsa S.A.	Seguridad	923 847002	-120.50
8791	Desarrollos Tyna	Diseño	91 4513172	3901.00
8792	Elektronics	Electricidad	91 3342244	-233.50
8793	Obras Marcos	Albañilería	91 7111132	2921.00
8794	Muebles Prados	Mobiliario	91 3120072	90.55
8795	Internal	Mobiliario	93 4287910	340.00
8796	Trabajos Cruz	Fontanería	672 112991	790.55

Consultando información: SELECT

- Comprueba si una cadena de caracteres está contenida en otra:
  - Se usa el símbolo % para sustituir a cualquier cosa (comodín)
    - Ma%: Comienza por "Ma"
    - %no: Termina en "no"
    - Ma%no: Empieza por "Ma" y termina por "no"
    - %ria%: contiene la cadena "ria" en cualquier sitio

- Ejemplo:

  SELECT * FROM Empresas WHERE Nombre LIKE '%os';

Consultando información: SELECT

SELECT * FROM Empresas WHERE Nombre LIKE '%os%';

Código	Nombre	Tipo	Teléfono	Saldo
8788	Suministros Ruiz	Electricidad	91 3987872	1288.00
8789	Transportes TSS	Transporte	91 5993725	433.00
8790	Segurinsa S.A.	Seguridad	923 847002	-120.50
8791	Desarrollos Tyna	Diseño	91 4513172	3901.00
8792	Elektronics	Electricidad	91 3342244	-233.50
8793	Obras Marcos	Albañilería	91 7111132	2921.00
8794	Muebles Prados	Mobiliario	91 3120072	90.55
8795	Internal	Mobiliario	93 4287910	340.00
8796	Trabajos Cruz	Fontanería	672 112991	790.55

Consultando información: SELECT

SELECT * FROM Empresas WHERE Nombre LIKE '%os%';

Código	Nombre	Tipo	Teléfono	Saldo
8788	Suministros Ruiz	Electricidad	91 3987872	1288.00
8789	Transportes TSS	Transporte	91 5993725	433.00
8790	Segurinsa S.A.	Seguridad	923 847002	-120.50
8791	Desarrollos Tyna	Diseño	91 4513172	3901.00
8792	Elektronics	Electricidad	91 3342244	-233.50
8793	Obras Marcos	Albañilería	91 7111132	2921.00
8794	Muebles Prados	Mobiliario	91 3120072	90.55
8795	Internal	Mobiliario	93 4287910	340.00
8796	Trabajos Cruz	Fontanería	672 112991	790.55

Añadiendo información: INSERT

INSERT INTO tabla [columnas] VALUES (valores);

- tabla: Ésta es la tabla donde se van a insertar los datos
- valores: Valores a insertar en cada una de las columnas de la tabla
- columnas: Nombres de las columnas de las que damos valores (opcional si se dan todos)

- Ejemplo:

  INSERT INTO Empresas VALUES
  (8797, 'Q&T Asociados', '', 917884520, 4590.40);

Añadiendo información: INSERT

INSERT INTO Empresas VALUES (8797, 'Q&T Asociados', '', 91 7884520, 4590.40);

Código	Nombre	Tipo	Teléfono	Saldo
8788	Suministros Ruiz	Electricidad	91 3987872	1288.00
8789	Transportes TSS	Transporte	91 5993725	433.00
8790	Segurinsa S.A.	Seguridad	923 847002	-120.50
8791	Desarrollos Tyna	Diseño	91 4513172	3901.00
8792	Elektronics	Electricidad	91 3342244	-233.50
8793	Obras Marcos	Albañilería	91 7111132	2921.00
8794	Muebles Prados	Mobiliario	91 3120072	90.55
8795	Internal	Mobiliario	93 4287910	340.00
8796	Trabajos Cruz	Fontanería	672 112991	790.55
8797	Q&T Asociados		91 7884520	4590.40

Añadiendo información: INSERT

INSERT INTO Empresas VALUES (8797, 'Q&T Asociados', '', 91 7884520, 4590.40);

Código	Nombre	Tipo	Teléfono	Saldo
8788	Suministros Ruiz	Electricidad	91 3987872	1288.00
8789	Transportes TSS	Transporte	91 5993725	433.00
8790	Segurinsa S.A.	Seguridad	923 847002	-120.50
8791	Desarrollos Tyna	Diseño	91 4513172	3901.00
8792	Elektronics	Electricidad	91 3342244	-233.50
8793	Obras Marcos	Albañilería	91 7111132	2921.00
8794	Muebles Prados	Mobiliario	91 3120072	90.55
8795	Internal	Mobiliario	93 4287910	340.00
8796	Trabajos Cruz	Fontanería	672 112991	790.55
8797	Q&T Asociados		91 7884520	4590.40

Modificando información: UPDATE

UPDATE tabla SET (valores)WHERE condición;

- tabla: Es la tabla donde se van a modificar los valores
- valores: Columnas que se van a modificar y valores que van a tomar (separados por comas)
- condición: Condición que deben cumplir las columnas para ser modificadas

- Ejemplo:

  UPDATE Empresas SET Tipo='Marketing'WHERE Codigo=8797;

Modificando información: UPDATE

UPDATE Empresas SET Tipo = 'Marketing' WHERE Codigo = 8797:

Código	Nombre	Tipo	Teléfono	Saldo
8788	Suministros Ruiz	Electricidad	91 3987872	1288.00
8789	Transportes TSS	Transporte	91 5993725	433.00
8790	Segurinsa S.A.	Seguridad	923 847002	-120.50
8791	Desarrollos Tyna	Diseño	91 4513172	3901.00
8792	Elektronics	Electricidad	91 3342244	-233.50
8793	Obras Marcos	Albañilería	91 7111132	2921.00
8794	Muebles Prados	Mobiliario	91 3120072	90.55
8795	Internal	Mobiliario	93 4287910	340.00
8796	Trabajos Cruz	Fontanería	672 112991	790.55
8797	Q&T Asociados	Marketing	91 7884520	4590.40

Modificando información: UPDATE

UPDATE Empresas SET Tipo = 'Marketing' WHERE Codigo = 8797:

Código	Nombre	Tipo	Teléfono	Saldo
8788	Suministros Ruiz	Electricidad	91 3987872	1288.00
8789	Transportes TSS	Transporte	91 5993725	433.00
8790	Segurinsa S.A.	Seguridad	923 847002	-120.50
8791	Desarrollos Tyna	Diseño	91 4513172	3901.00
8792	Elektronics	Electricidad	91 3342244	-233.50
8793	Obras Marcos	Albañilería	91 7111132	2921.00
8794	Muebles Prados	Mobiliario	91 3120072	90.55
8795	Internal	Mobiliario	93 4287910	340.00
8796	Trabajos Cruz	Fontanería	672 112991	790.55
8797	Q&T Asociados	Marketing	91 7884520	4590.40

Borrando información: DELETE

DELETE FROM tablaWHERE condición;

- tabla: Es la tabla de donde se van a eliminar los valores
- condición: Condición que deben cumplir las filas a eliminar

- Ejemplo:

  DELETE FROM Empresas WHERE Codigo < 8790;

Borrando información: DELETE

```
DELETE FROM Empresas WHERE Codigo < 8790;
```

Código	Nombre	Tipo	Teléfono	Saldo
8788	Suministros Ruiz	Electricidad	91 3987872	1288.00
8789	Transportes TSS	Transporte	91 5993725	433.00
8790	Segurinsa S.A.	Seguridad	923 847002	-120.50
8791	Desarrollos Tyna	Diseño	91 4513172	3901.00
8792	Elektronics	Electricidad	91 3342244	-233.50
8793	Obras Marcos	Albañilería	91 7111132	2921.00
8794	Muebles Prados	Mobiliario	91 3120072	90.55
8795	Internal	Mobiliario	93 4287910	340.00
8796	Trabajos Cruz	Fontanería	672 112991	790.55

Borrando información: DELETE

DELETE FROM Empresas WHERE Codigo < 8790;

Código	Nombre	Tipo	Teléfono	Saldo
8788	Suministros Ruiz	Electricidad	91 3987872	1288.00
8789	Transportes TSS	Transporte	91 5993725	433.00
8790	Segurinsa S.A.	Seguridad	923 847002	-120.50
8791	Desarrollos Tyna	Diseño	91 4513172	3901.00
8792	Elektronics	Electricidad	91 3342244	-233.50
8793	Obras Marcos	Albañilería	91 7111132	2921.00
8794	Muebles Prados	Mobiliario	91 3120072	90.55
8795	Internal	Mobiliario	93 4287910	340.00
8796	Trabajos Cruz	Fontanería	672 112991	790.55

Ejercicio: Consultas a una BBDD

- Realizar las siguientes consultas a la BBDD de ejemplo
  - Obtener el nombre y el teléfono de las empresas que se dedican a la electricidad
  - Obtener el nombre de las empresas madrileñas
  - Modificar el saldo de las empresas con identificador en el intervalo [8790, 8793] e incrementarlo en 1000 euros
  - Borrar de la BBDD las empresas con saldo negativo

Trabajando con JDBC

- Hay que tener instalado un Sistema Gestor de Base de Datos (SGBD) primero
  - MySQL en nuestro caso

- Hay que instalar el driver que traduce llamadas a funciones JDBC en el protocolo nativo del SGBD
  - MySQL Connector en nuestro caso
  - **IMPORTANTE**: Hay que asegurarse de que está en el CLASSPATH

- En el código del programa, hay que importar las clases que nos permiten trabajar con MySQL

  import java.sql.*;

Trabajando con JDBC

- Lo siguiente es cargar el driver y establecer una conexión

  Class.forName(**"com.mysql.jdbc.Driver"**);

  Connection conn = DriverManager.getConnection
  (**"jdbc:mysql://localhost/empresas"**, **"user"**, **"pass"**);

- A continuación, se declara un objeto de tipo ***Statement*** que nos permitirá ejecutar consultas en la BBDD

  Statement stat = conn.createStatement();

- Al ejecutar un Statement, se nos devuelve un objeto de tipo *ResultSet*

  ResultSet res = stat.executeQuery
  (**"SELECT * FROM Empresas"**);

Trabajando con JDBC

- Si la sentencia que vamos a ejecutar modifica la BBDD o su estructura, debemos usar *executeUpdate()*

  int rows = stat.executeUpdate("INSERT INTO Empresas
  VALUES (8797, 'Q&T Asociados', '', 917884520, 4590.40)");

- El objeto ResultSet contiene información de los datos devueltos, así como de los metadatos

```
ResultSetMetaData metaData = res.getMetaData();
    int nCols = metaData.getColumnCount();
    for (i = 1; i <= nCols; i++)
            System.out.print(" " + metaData.getColumnName(i));
```

- Tanto metadatos como resultados empiezan en índice 1

Trabajando con JDBC

- Para recorrer el conjunto de resultados devuelto hay que usar dos tipos de métodos
  - next(): Devuelve true/false en función de si hay más resultados
  - getXXX(int i) y getXXX(String s): Devuelven un objeto de tipo XXX
    - Se puede acceder por número de columna o por su nombre
    - Más eficiente el primero (pero hay que conocer el orden de antemano)

- Ejemplo:

```
while (res.next()) {
        for (i = 1; i <= nCols; i++)
                System.out.print(" " + res.getObject(i));
        System.out.println();
}
```

- **IMPORTANTE**: Siempre usar next() antes del primer acceso a resultados

Trabajando con JDBC

- Todos estos métodos pueden lanzar excepciones
  - SQLException principalmente

- Es recomendable cerrar las conexiones, los resultsets y los statements en grupo finally después de las excepciones para asegurarnos de que siempre se hace
  - Cada uno de estos objetos tiene un método *close()*
  - A su vez, estos métodos pueden lanzar excepciones
    - Hay que capturarlas también

Ejercicio

- Crear una aplicación que trabaje gestione una pequeña agenda de contactos
- Hay que crear una tabla en la BBDD que almacene la información de los contactos: DNI (clave), Nombre, Apellidos, Dirección, e-mail, etc.
- La la aplicación nos debe permitir: Introducir nuevos datos, visualizar los datos de un contacto (de una lista o mediante búsqueda), modificar dichos datos y borrar un contacto
  - Recomendación: Usar distintas ventanas para cada funcionalidad

Ejercicio 2

- Crear una aplicación que nos permita gestionar una biblioteca
- Habrá distintas tablas:
  - Libro: con información sobre libros: ISBN (clave), Título, Número de páginas, Resumen, Código de autor (clave ajena), etc.
  - Autor: con información sobre autores: Código de autor (clave), Nombre, Apellidos, etc.
  - Usuarios: con información sobre los usuarios de la biblioteca: DNI (clave), y resto de información personal
  - Préstamos: relaciona usuarios y libros prestados: Código del préstamo (clave), DNI del usuario (clave ajena), ISBN del libro (clave ajena), fecha de préstamo, etc.

Ejercicio 2 (Continuación)

- La aplicación debe permitir introducir libros, junto con su autor, y comprobar si el autor del libro ya existe en la tabla de Autores
- También debe permitir dar de alta usuarios, consultar información de libros y de usuarios, y prestar un libro desde la lista de libros recuperada

Sentencias preparadas: PreparedStatement

- Sentencias precompiladas
- Su ejecución es más eficiente que la de un Statement habitual
- Útiles si vamos a ejecutar la misma consulta múltiples veces
- Se les indica la consulta a ejecutar en el momento de la construcción del objeto

  PreparedStatement stat = con.prepareStatement("UPDATE Empresas SET Tipo= ?WHERE Codigo= ?");

- Los '?' nos permiten asignarles valores distintos en tiempo de ejecución

Sentencias preparadas: PreparedStatement

- Para asignar valores usamos los métodos siguientes

```
stat.setString(1, "Marketing");
stat.setInt(2, 8795);
stat.executeUpdate();
```

- Lo mismo en el caso de executeQuery()

- Los valores de los campos variables permanecen hasta que otra llamada los sobreescribe o se invoca a *clearParameters()*

Ejercicio

- Repetir el ejercicio de consultas a la BBDD usando PreparedStatements con y sin parámetros variables

Usando transacciones

- Permiten ejecutar bloques de sentencias conjuntamente
  - "O todas o ninguna"
  - Permiten volver al estado anterior al comienzo del bloque si algo va mal (alguna de las consultas/actualizaciones falla)
  - **No** se puede deshacer después de haber enviado definitivamente un bloque

- ¿Cómo se trabaja con transacciones?
  - Métodos commit() y rollback() de la clase Connection

Usando transacciones

- Después de ejecutar cada sentencia (executeQuery()/ executeUpdate()) se hace un commit() ***implícito***
  - Si queremos definir bloques personalizados, hay que cambiar este comportamiento

    conn.setAutoCommit(false);
  - Cuando acabemos, restauramos el comportamiento por defecto

    conn.setAutoCommit(true);

- Tras toda la secuencia de operaciones, hay que hacer el commit manualmente

  conn.commit();

Usando transacciones

- Ejemplo:

```
conn.setAutoCommit(false);
int rows = stat.executeUpdate("INSERT INTO Empresas
   VALUES (8797, 'Q&T Asociados', '', 917884520, 4590.40)");
int rows2 = stat.executeUpdate("UPDATE Empresas SET
   Saldo=0.0 WHERE Saldo < 0.0")
con.commit();
conn.setAutoCommit(true);
```

Ejercicio

- Rehacer el ejercicio de la biblioteca para que las inserciones/ modificaciones de datos de un libro y un autor se hagan como una única transacción

Extensiones Multimedia

Trabajando con imágenes

- Clase **abstracta** Image
  - Distintas implementaciones: BufferedImage, etc.

- Diferentes maneras de cargar una imagen

```
Image img = Toolkit.getDefaultToolkit().getImage(
        this.getClass().getResource("imagen.png"));
```

  - También funciona con URLs y Strings

```
BufferedImage img2 = ImageIO.read(new File("imagen.png"));
```

Trabajando con imágenes

- Clase ImageIcon
  - Implementa el interfaz Icon
  - También se puede utilizar para cargar imágenes

- Distintas formas de cargar imágenes

```
ImageIcon ico = ImageIcon("src/imagenes/imagen.png");
ImageIcon ico2 =
   ImageIcon(this.getClass().getResource("imagen.png"));
ImageIcon ico3 = ImageIcon(img);
```

Pintando imágenes

- Usamos los métodos **paint** de la superficie donde vayamos a pintar (un JFrame, un Applet)
- Estos métodos paint reciben un objeto de tipo **Graphics** como argumento sobre el que podemos pintar

```
graph.drawImage(img, 0, 0, this);
graph.drawImage(img2, 0, 120, 50, 50, this);
ico.paintIcon(ico, g, 180, 0);
```

Ejercicio

- Crear una ventana que muestre imágenes de distintos tipos simultáneamente
- Usar tanto objetos de tipo Image como ImageIcon
- Probar los distintos métodos para cargar imágenes
- Probar los distintos métodos de dibujado (con y sin escalado)

Reproduciendo sonidos

- Una opción es utilizar los métodos que nos proporcionan la clase Applet y la interfaz AudioClip
  - La clase Applet nos da dos métodos **play** para reproducir sonido mientras el applet esté ejecutando

    void play(URL fullurl);

    void play(URL url, String name);
  - La interfaz AudioClip nos da métodos para reproducir, parar y reproducir en bucle un sonido

    void play();

    void stop();

    void loop();

Reproduciendo sonidos

- Para recuperar una instancia de tipo AudioClip usaremos los métodos **getAudioClip** de la clase Applet

  AudioClip getAudioClip(URL fullurl);

  AudioClip getAudioClip(URL url, String name);

- Ejemplo (dentro de un Applet):

  AudioClip clip = getAudioClip(getDocumentBase(), “hi.au”);

  clip.play();

- Formatos soportados: .au, .wav, .aiff, .mid

Ejercicio

- Crear un reproductor de sonidos
- Debe permitir seleccionar el fichero desde un diálogo
- Debe tener tres botones (aparte del de selección del fichero)
  - Reproducir
  - Parar
  - Reproducir en bucle

Reproducción de contenido multimedia

- Una opción es el Java Multimedia Framework (JMF)
  - Soporte varios contenedores y codecs
  - Promovida por IBM y Sun
- Se puede descargar de

  java.sun.com/products/java-media/jmf/2.1.1/download.html
- Vamos a ver su funcionamiento con un pequeño ejemplo…

Reproducción de contenido multimedia

- Hay que importar los paquetes javax.media

  import javax.media.*;

- La clase **Manager** es el punto de partida para recuperar los controles multimedia
  - Podemos recuperar una instancia de un reproductor de medios (**Player**) adecuada para un tipo de fichero (URL)

  Player mp = Manager.createRealizedPlayer(mediaURL);

Reproducción de contenido multimedia

- Esa instancia contiene los componentes visual y de control del reproductor

- El componente visual lo podemos recuperar así

  Component video = mp.getVisualComponent();

- El componente de control lo recuperamos de una manera análoga

  Component controls = mp.getControlPanelComponent();

- Los componentes devueltos pueden ser directamente añadidos a un panel

Reproducción de contenido multimedia

- Finalmente, para iniciar la reproducción del medio usamos el método start

```
mp.start();
```

- La clase MediaPlayer proporciona todos los métodos necesarios para implementar nuestros propios controles…