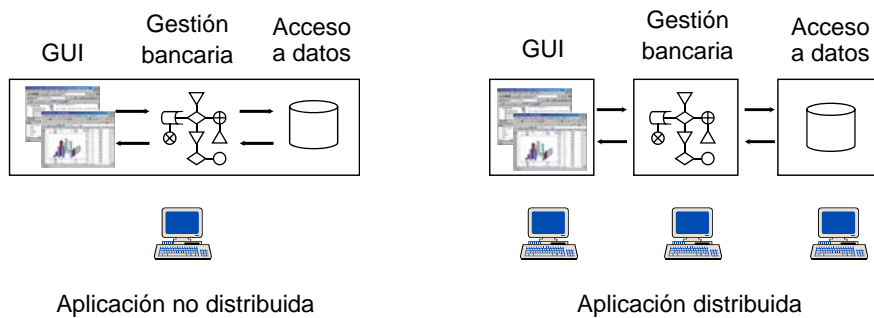


Sistemas cliente-servidor

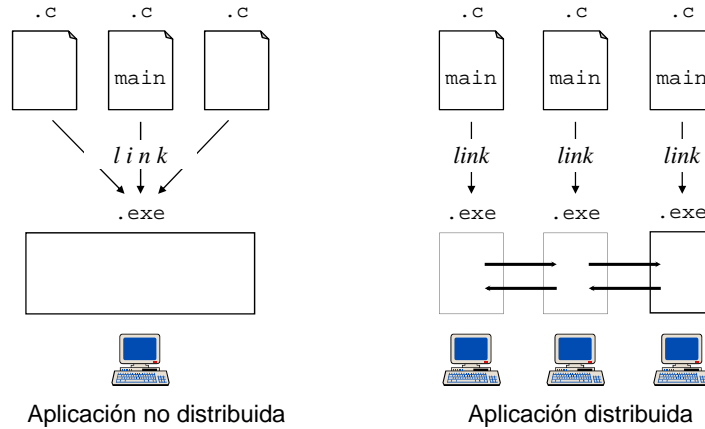
Aplicaciones distribuidas (I)

- Componentes intercambiables
 - Mantenimiento independiente
 - Necesidad inherente de distribución
- } separación interfaz / implementación



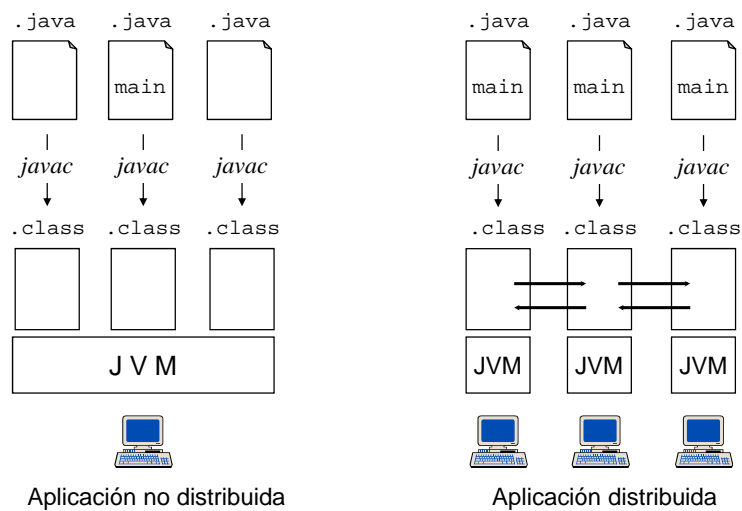
Aplicaciones distribuidas (II)

- El código de la aplicación está repartido en varios programas que se ejecutan independientemente



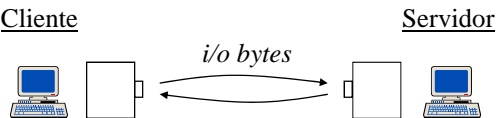
3

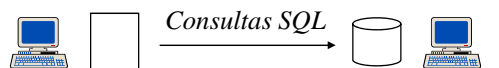
Aplicaciones distribuidas (III)

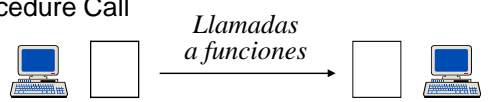


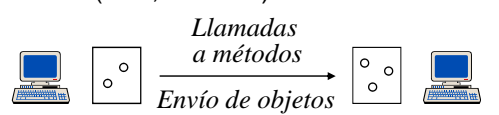
4

Comunicación entre procesos

- Sockets


The diagram shows a Client (represented by a computer icon and a box) and a Server (represented by a computer icon and a box) connected by a double-headed arrow labeled "i/o bytes".
- Conexión a bases de datos (JDBC, ODBC)


The diagram shows a Client (represented by a computer icon and a box) connected to a database server (represented by a cylinder icon and a computer icon) with the label "Consultas SQL".
- Remote Procedure Call


The diagram shows a Client (represented by a computer icon and a box) calling a function on a Server (represented by a box and a computer icon) with the label "Llamadas a funciones".
- Objetos distribuidos (RMI, CORBA)


The diagram shows a Client (represented by a computer icon and a box containing two small circles) sending objects to a Server (represented by a box containing two small circles and a computer icon) with the label "Llamadas a métodos" and "Envío de objetos". A hand icon points to this item.

5

Objetos distribuidos

POO en sistemas distribuidos

Implementación

Estructura de datos
Código de métodos
Localización del objeto
Localización clase (RMI)
Lenguaje (CORBA)

Interfaz

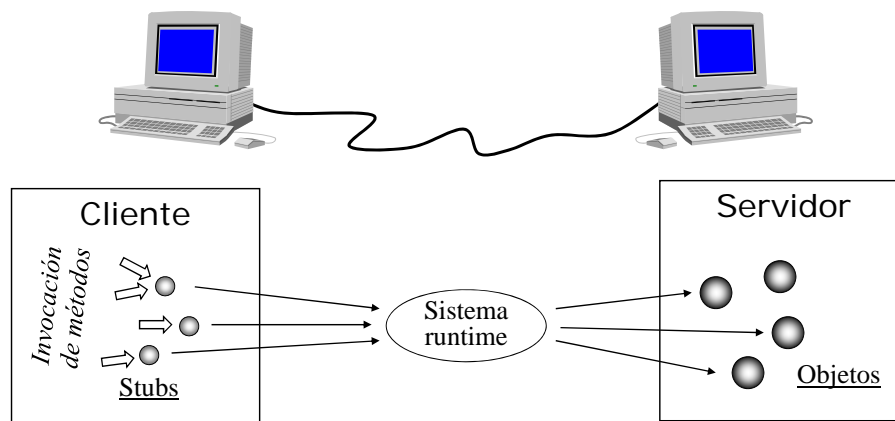
Signatura de los métodos

- Modularidad
- Interfaz para el cliente
- Implementación en el servidor
- Transparencia

Estándares: RMI, CORBA, DCOM

7

Objetos distribuidos



8

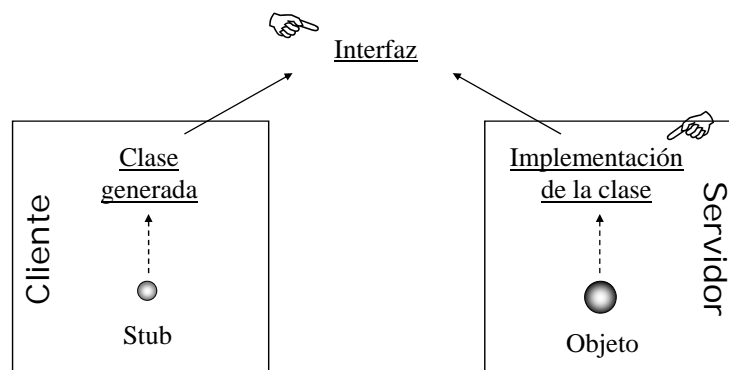
Elementos principales

- Interfaces remotas
 - Interfaz acordada entre servidor y cliente
 - Métodos que el cliente puede invocar
- Implementación de los objetos
 - Implementación de los métodos de las interfaces
 - Objetos creados en el servidor
- Stubs
 - Actúan como referencias a objetos remotos desde el cliente
 - Retransmiten llamadas desde el cliente hacia el servidor
- Sistema runtime
 - Mediador entre stubs y objetos
- Acceso a los objetos (obtención de stubs)
 - Servicio de nombres
 - Métodos que envían objetos

9

Interfaz de los objetos

- Interfaz común acordada entre cliente y servidor
- Tipos de objetos en el servidor que van a ser accesibles desde el cliente
- Los métodos incluidos en la interfaz son los que podrá invocar el cliente



10

Interfaz: ejemplo (RMI)

```
public interface CuentaRemota extends Remote {  
    public long mostrarSaldo ()  
        throws RemoteException;  
    public void ingreso (long importe)  
        throws RemoteException;  
    public void reintegro (long importe)  
        throws RemoteException;  
};
```

11

Objetos y stubs

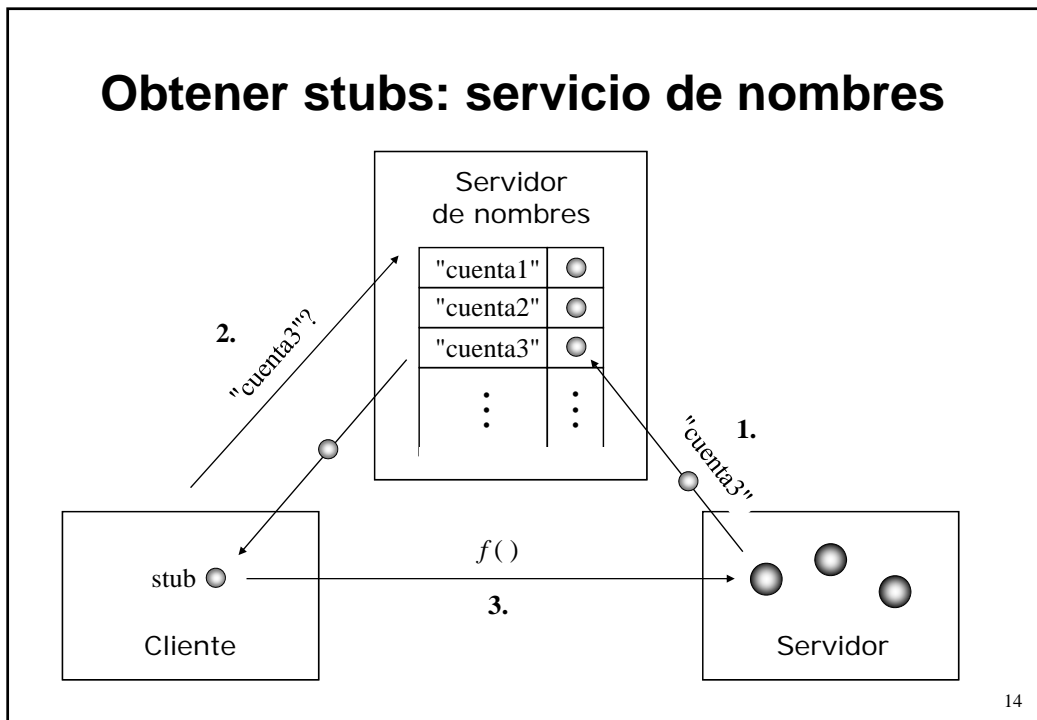
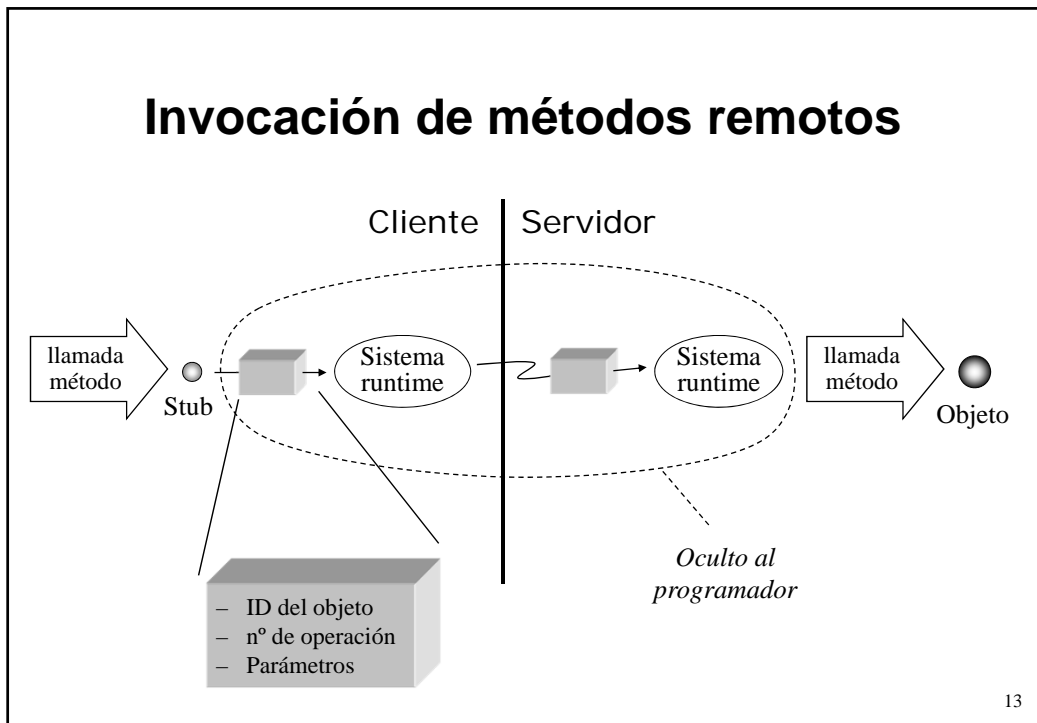
Implementación de los objetos

- Forma parte del programa servidor
- Debe concordar con la interfaz, puede extenderla
- La implementación de los métodos se ejecuta en el servidor

Stubs

- Son objetos que residen en el cliente
- Representan objetos remotos (host + puerto + ID del objeto)
- Su clase es generada automáticamente a partir de la interfaz
- Implementan la interfaz ⇒ se pueden invocar los métodos sobre ellos

12



Plataformas de desarrollo para objetos distribuidos

- Lenguaje de definición de interfaces
 - Puede ser distinto al de la implementación
- Compilador de interfaces
 - Genera clases para los stubs
- Sistema runtime
 - Intermediario entre los stubs y la implementación de los objetos
- Servicio de nombres
 - Localización de objetos por nombre desde el cliente
- RMI, CORBA, DCOM

15

Estándares para objetos distribuidos

Common Object Request Broker Architecture (CORBA)

- Creado por el Object Management Group (OMG) → consenso
- Integración de programas en distintos lenguajes → heterogeneidad
- Interoperabilidad (IIOP)
- Servicios adicionales integrados
- CORBA 1.0: 1992, CORBA 2.0: 1997, CORBA 2.6: dic 2001
- Java 1.2+ incorpora CORBA 2.0 / 2.3 con el nombre de Java IDL



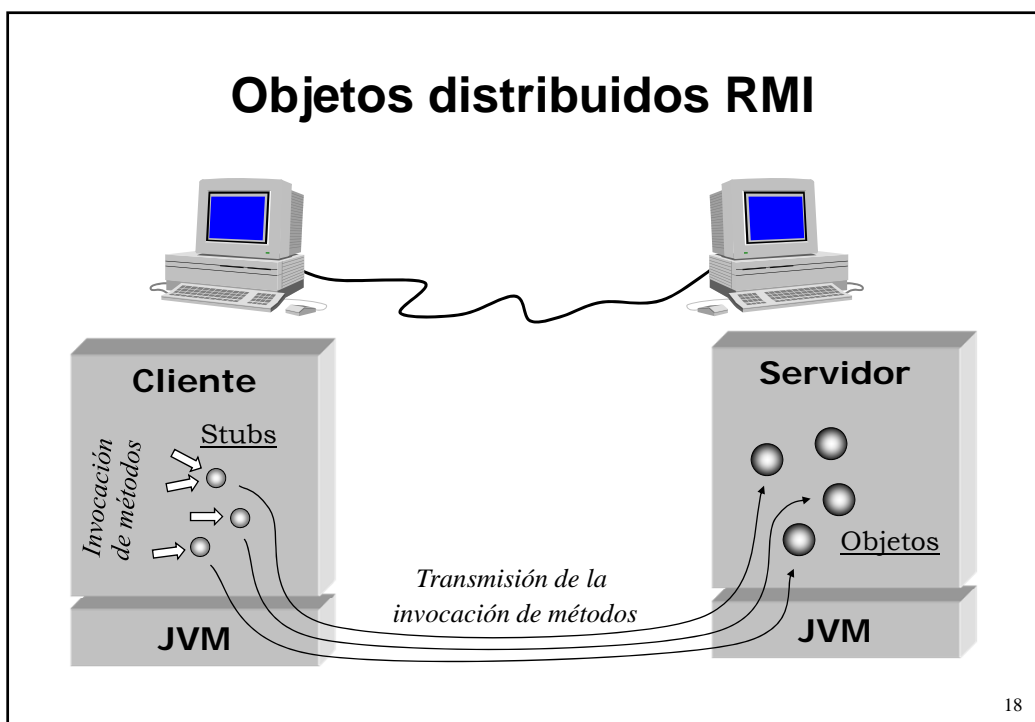
Remote Method Invocation (RMI)

- Integrado en el lenguaje Java
- Facilidad de programación, portabilidad de las implementaciones
- Paso de objetos por valor
- Carga dinámica de clases

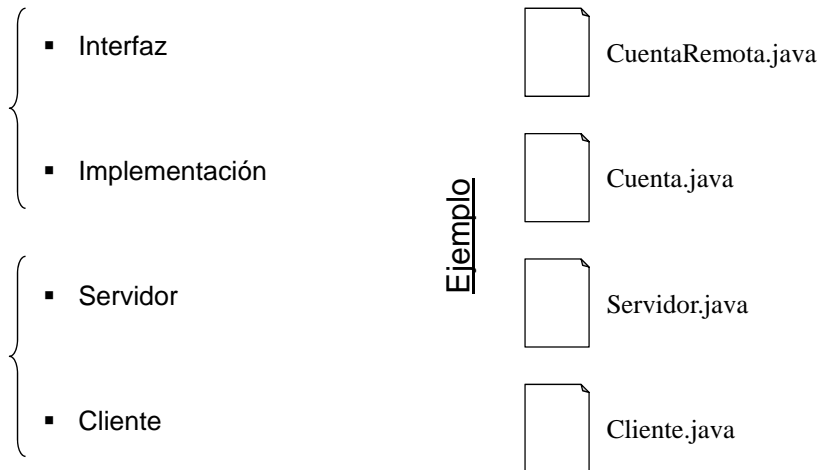
16

Objetos distribuidos

Remote Method Invocation (RMI)



Construcción de una aplicación en RMI



19

Interfaz e implementación

Interfaces

- Interfaces Java que derivan de la interfaz `java.rmi.Remote`
- Todos los métodos deben declarar `java.rmi.RemoteException`
- Argumentos que pueden tomar los métodos:
 - Tipos primitivos
 - Stubs y objetos remotos
 - Objetos locales serializables (implementan la clase `java.io.Serializable`)

Implementación

- Subclase de `java.rmi.server.UnicastRemoteObject` que implemente la interfaz remota
- Implementar todos los métodos de la interfaz

20

Definir la interfaz

CuentaRemota.java

```
public interface CuentaRemota extends Remote {  
    public long mostrarSaldo ()  
        throws RemoteException;  
    public void ingreso (long importe)  
        throws RemoteException;  
    public void reintegro (long importe)  
        throws RemoteException;  
    public void transferencia (CuentaRemota cuenta,  
                               long importe)  
        throws RemoteException;  
};
```

21

Implementar la interfaz

Cuenta.java

```
class Cuenta extends UnicastRemoteObject  
    implements CuentaRemota {  
    private long saldo;  
    private long numero;  
    private static long numCuentas;  
    Cuenta () throws RemoteException {  
        numero = ++numCuentas;  
    }  
    public long mostrarSaldo ()  
        throws RemoteException {  
        return saldo;  
    }  
};
```

22

```
public void ingreso (long importe)
    throws RemoteException {
    saldo += importe;
}
public void reintegro (long importe)
    throws RemoteException {
    saldo -= importe;
}
public void transferencia (CuentaRemota cuenta,
                           long importe)
    throws RemoteException {
    reintegro (importe);
    cuenta.ingreso (importe);
}
}
```

23

Programa servidor y programa cliente

- Programa servidor
 - Crea instancias de las clases remotas y las registra en el servidor de nombres
- Programa cliente
 - Declara objetos de la interfaz remota y obtiene stubs del servidor de nombres
 - Invoca métodos sobre los objetos
- Servicio de nombres
 - El programa servidor de nombres: rmiregistry
 - La clase `java.rmi.Naming`

| | | |
|--|---|--------------------------------|
| <code>Naming.rebind (String, Remote)</code> | } | <i>Asignación de nombres</i> |
| <code>Naming.unbind (Remote)</code> | | |
| <code>Naming.lookup (String) → Remote</code> | } | <i>Localización de objetos</i> |

24

Programa servidor

Servidor.java

```
public class Servidor {  
    public static void main (String args[])  
        throws Exception {  
        Cuenta c1 = new Cuenta ();  
        Naming.rebind ("cuenta1", c1);  
        Cuenta c2 = new Cuenta ();  
        Naming.rebind ("cuenta2", c2);  
    }  
}
```

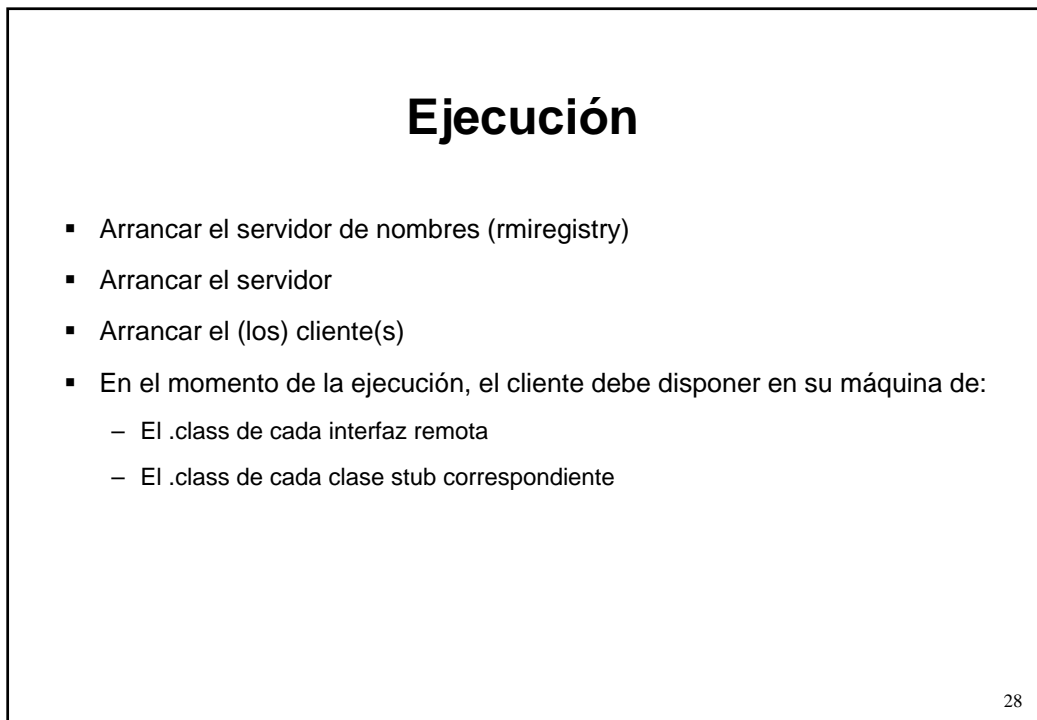
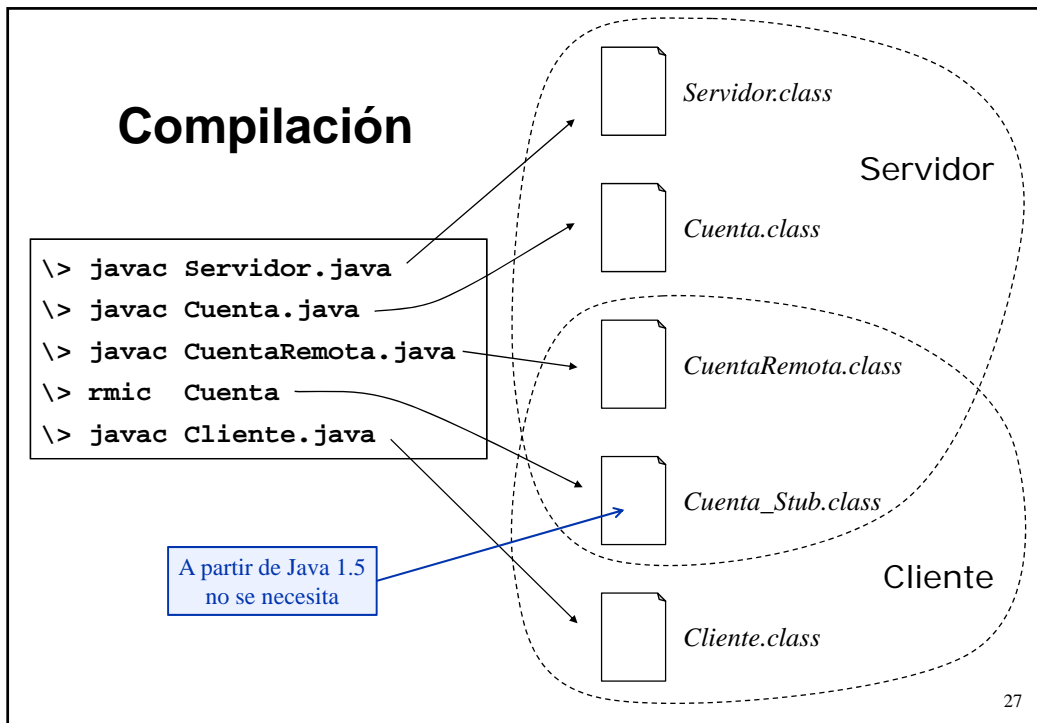
25

Programa cliente

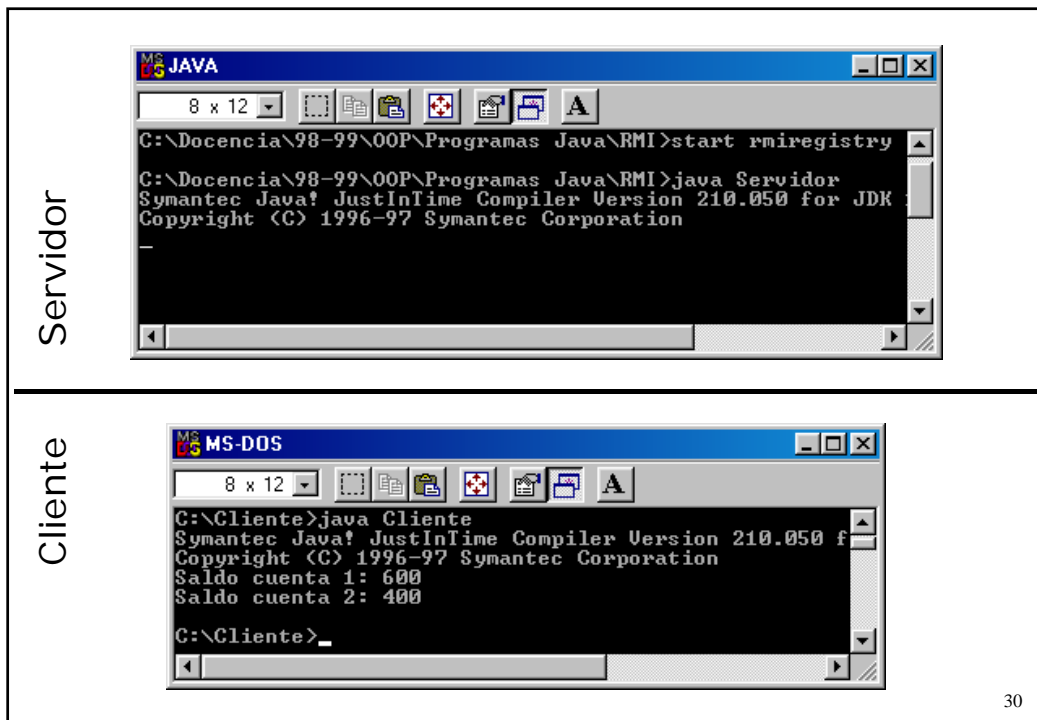
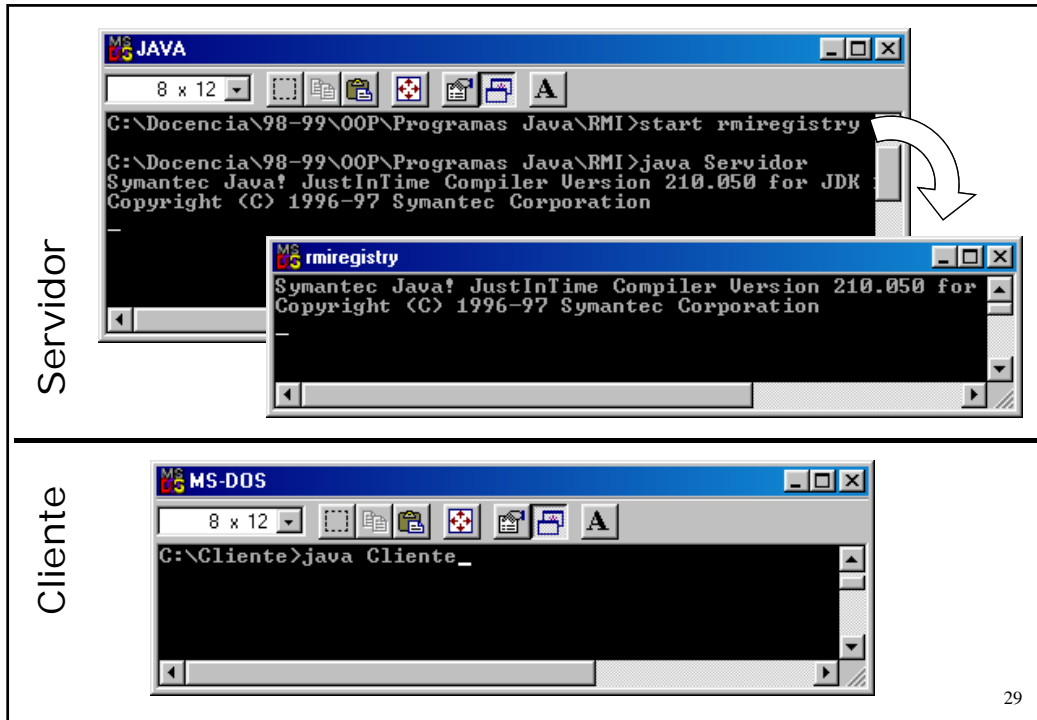
Cliente.java

```
public class Cliente {  
    public static void main (String args[])  
        throws Exception {  
        CuentaRemota c1 = (CuentaRemota) Naming.lookup (  
            "/ejemplo.eps.uam.es/cuenta1");  
        CuentaRemota c2 = (CuentaRemota) Naming.lookup (  
            "/ejemplo.eps.uam.es/cuenta2");  
        c1.ingreso (1000);  
        c1.transferencia (c2, 400);  
        System.out.println ("Saldo cuenta 1: "  
            + c1.mostrarSaldo ()  
            + "\nSaldo cuenta 2: "  
            + c2.mostrarSaldo ());  
    }  
}
```

26



8. Sistemas cliente-servidor



Invocación de métodos (I)

- Los stubs se manejan como si fuesen los propios objetos remotos
- Mediante los stubs el programador maneja los objetos remotos igual que los objetos locales, misma sintaxis
- El proceso cliente invoca métodos sobre el stub, y el stub los retransmite (por la red en su caso) al objeto real en el proceso en que reside
- El método se ejecuta en el servidor
- El valor devuelto lo recibe el cliente por el camino inverso
- RMI gestiona los detalles de la comunicación subyacente

31

Invocación de métodos (II)

- El cliente invoca un método sobre un stub
- El stub empaqueta (serializa) los argumentos, envía la llamada a su JVM local, y queda a la espera de la respuesta
- La JVM local inicia una conexión con la JVM remota que contiene el objeto, y transmite la llamada con sus argumentos
- El servidor puede o no ejecutar cada método en un thread independiente
En todo caso, el servidor debe ser thread-safe (p.e. utilizar métodos `synchronized`, p.e. operaciones en cuentas bancarias)
- Cuando la llamada retorna, el stub desempaqueta el valor devuelto (si lo hay)
- El stub devuelve el valor al programa cliente (esto puede incluir excepciones emitidas por el método en el servidor)

32

Stubs

- Una vez compiladas las clases remotas, las clases de stubs se generan automáticamente mediante: `rmic xxx.class → xxx_stub.class`
 - A partir de Java 1.5, la JVM genera dinámicamente estas clases sin necesidad de `rmic`
- La implementación de los métodos remotos en la clase del stub es distinta de la implementación de los mismos métodos en la clase remota
 - Método en stub: intermediación entre el cliente y los objetos del servidor, empaqueta los argumentos (serialización) y transmite la invocación al servidor
 - Método en servidor: contiene la semántica del método
- El stub contiene información sobre el objeto remoto al que representa
 - Localización del objeto: host y puerto
 - Clase del objeto
 - Un ID que identifica a cada objeto remoto en el servidor
- El stub oculta
 - Serialización
 - Detalles de comunicación con el servidor

33

Gestión interna de la comunicación entre procesos

RMI se ocupa de:

- Abrir, controlar, cerrar conexiones entre cliente y servidor
- Empaquetamiento y desempaquetamiento de datos
- Preparar al servidor para que permanezca a la espera de llamadas a métodos desde los clientes mediante sockets
- Atender a las llamadas transfiriéndolas a los objetos correspondientes

34

Clases remotas

UnicastRemoteObject

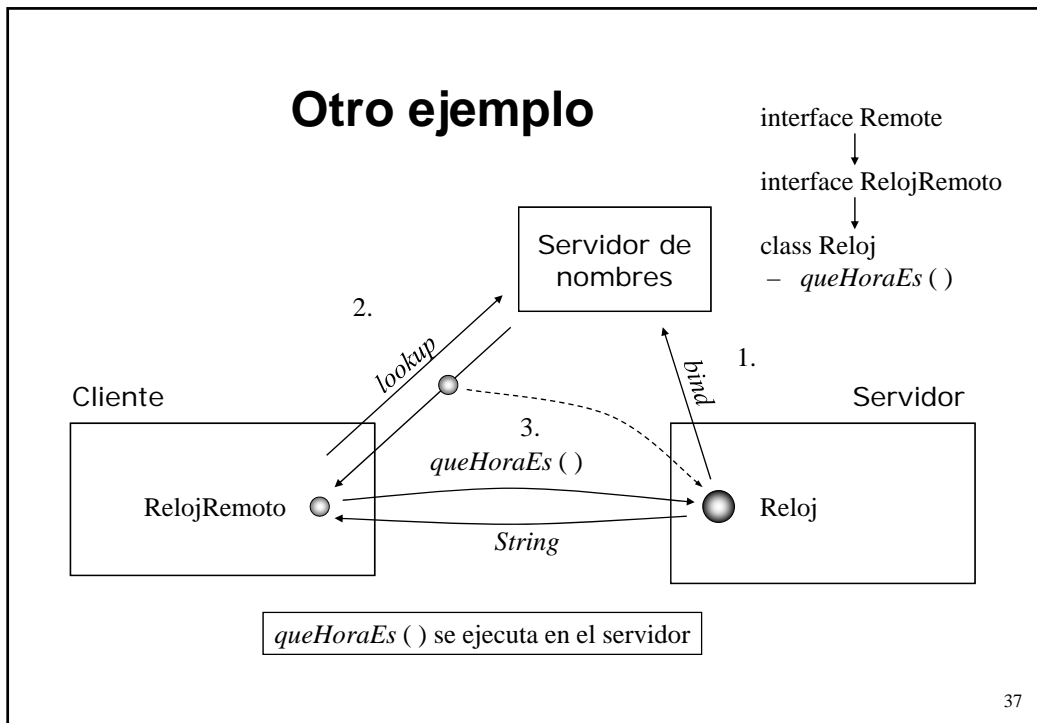
- `java.rmi.server.UnicastRemoteObject`
 - Invoca a `UnicastRemoteObject.exportObject(this)` en su constructor
 - Redefine algunos métodos de `Object`: `equals`, `hashCode`, `toString`
- No es obligatorio extender `UnicastRemoteObject`
 - Basta con invocar `exportObject(obj[,puerto])` sobre las instancias (p.e. en el constructor)
- `exportObject(Remote)`
 - Registra la existencia del objeto en el sistema runtime RMI
 - El objeto queda a la espera de llamadas de clientes utilizando un puerto anónimo elegido por RMI o el sistema operativo
- `getClientHost()` permite obtener la dirección IP del cliente

35

Ciclo de vida del servidor

- Main crea objetos y los registra en el servidor de nombres
- Main termina pero el servidor sigue activo: la JVM sigue ejecutándose
- El sistema RMI mantiene activo un servidor mientras exista alguna referencia a alguno de los objetos creados en él
- Esto incluye las referencias a los objetos en el servidor de nombres
- Cuando un objeto ya no tiene referencias desde ningún cliente ni dentro del propio servidor, queda disponible para GC

36



RelojRemoto.java

```
import java.rmi.*;

public interface RelojRemoto extends Remote {
    public String queHoraEs () throws RemoteException;
}
```

38

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

class Reloj extends UnicastRemoteObject
    implements RelojRemoto {
    private String zona;
    private DateFormat formatoHora;
    Reloj (String zona) throws RemoteException {
        this.zona = zona;
        formatoHora = DateFormat.getTimeInstance ();
        formatoHora.setTimeZone (TimeZone.getTimeZone (zona));
    }
    public String queHoraEs () throws RemoteException {
        String hora = formatoHora.format (new Date ());
        System.out.println ("Cliente solicita la hora en "
            + zona + ": " + hora);
        return hora;
    }
}
```

Servidor.java

39

```
public class Servidor {
    public static void main (String args[]) throws Exception {
        Reloj peninsula = new Reloj ("Europe/Madrid");
        Naming.rebind ("Hora_peninsular", peninsula);
        Reloj canarias = new Reloj ("Europe/Lisbon");
        Naming.rebind ("Hora_Canarias", canarias);
    }
}
```

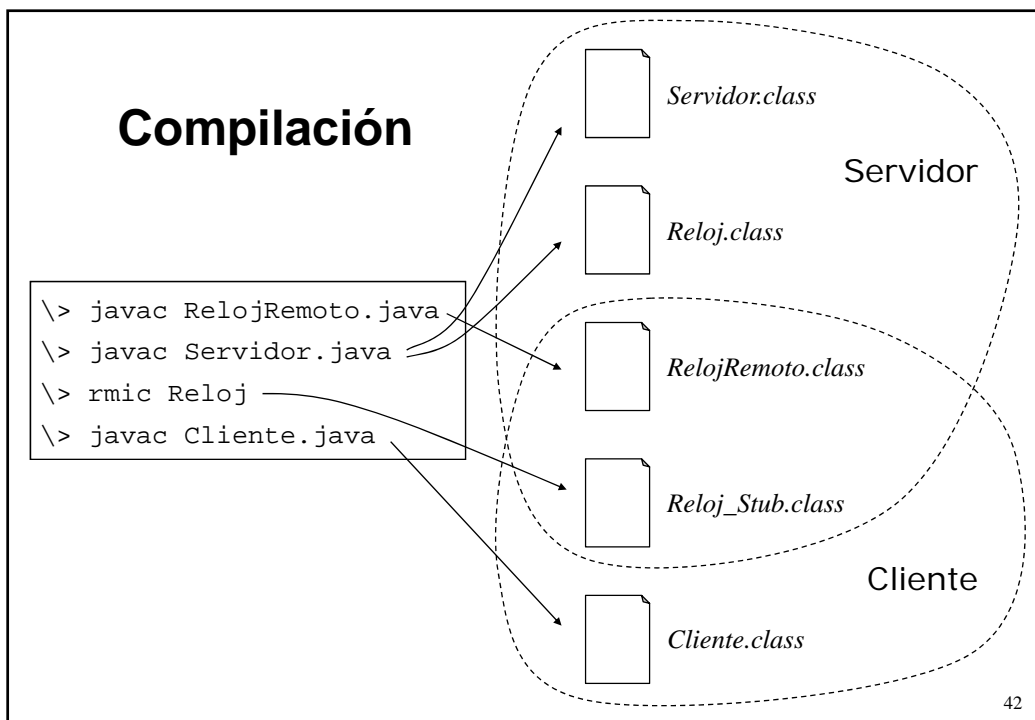
40

Cliente.java

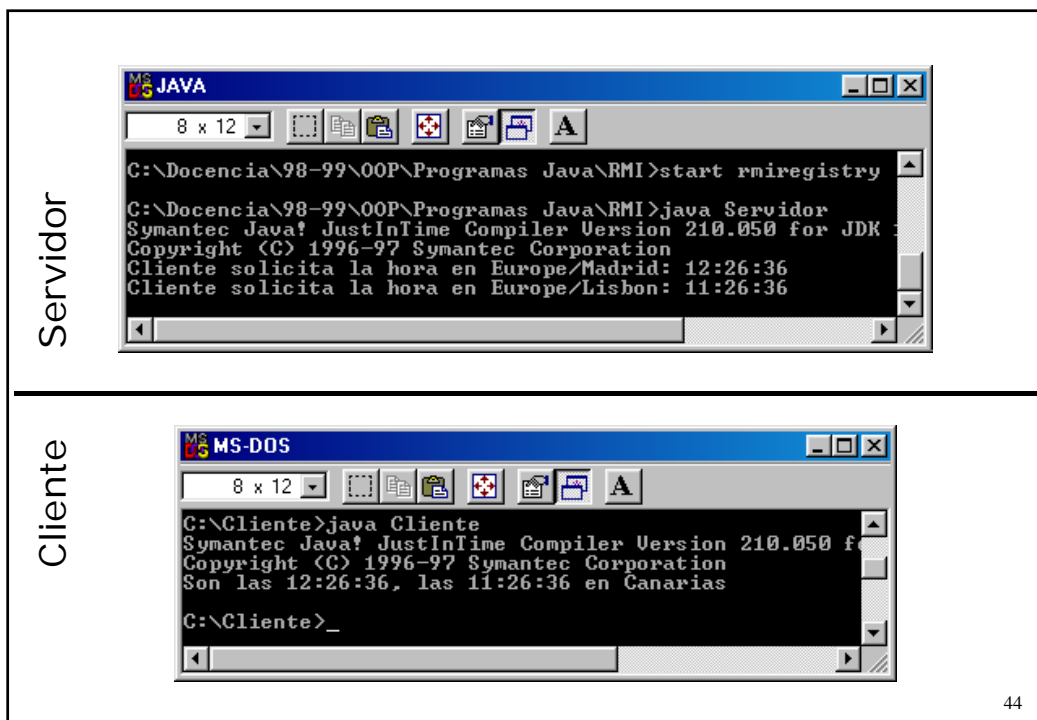
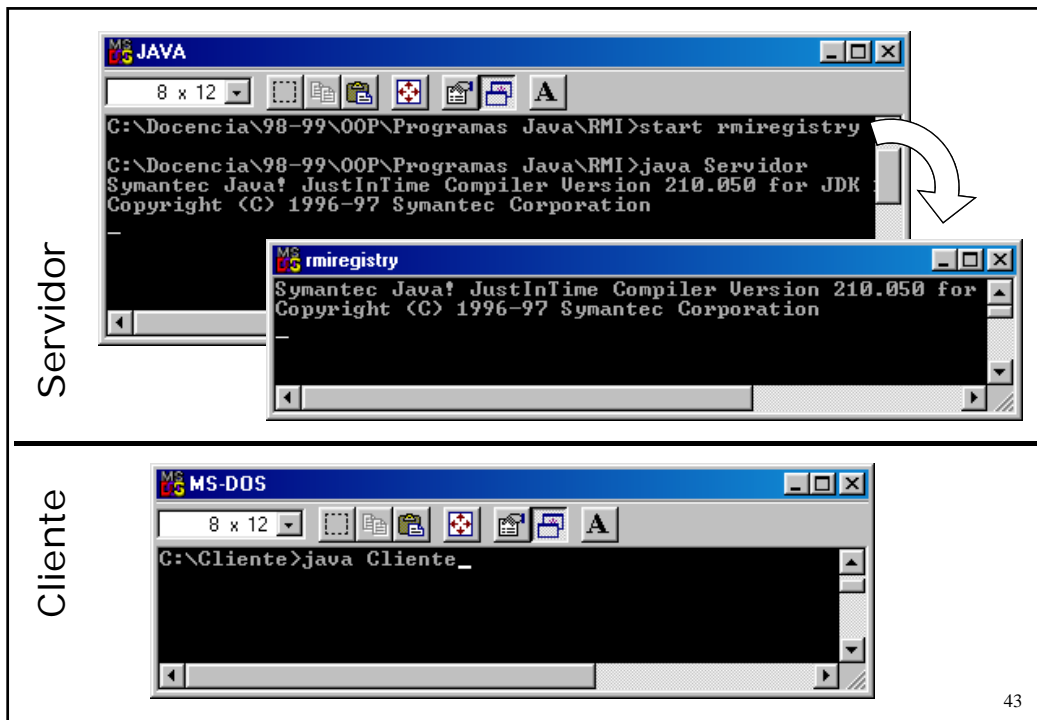
```
import java.rmi.*;

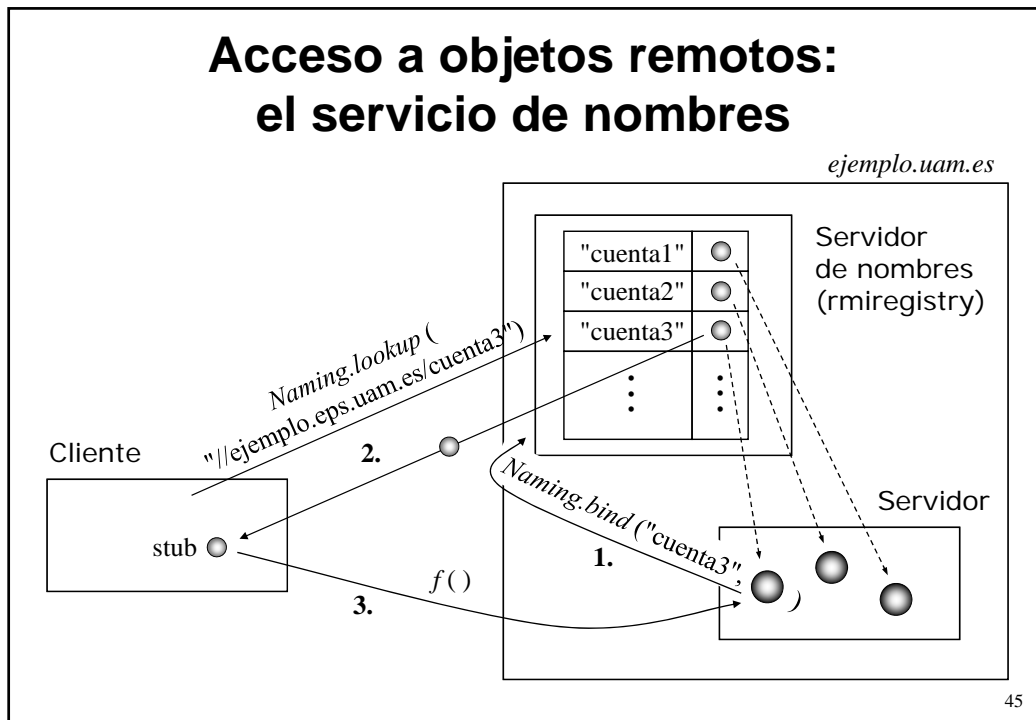
public class Cliente {
    public static void main (String args[]) throws Exception {
        RelojRemoto reloj1 = (RelojRemoto) Naming.lookup (
            "//ejemplo.eps.uam.es/Hora_peninsular");
        RelojRemoto reloj2 = (RelojRemoto) Naming.lookup (
            "//ejemplo.eps.uam.es/Hora_Canarias");
        System.out.println ("Son las "
            + reloj1.queHoraEs ()
            + ", las "
            + reloj2.queHoraEs ()
            + " en Canarias");
    }
}
```

41



8. Sistemas cliente-servidor





El programa servidor de nombres: rmiregistry

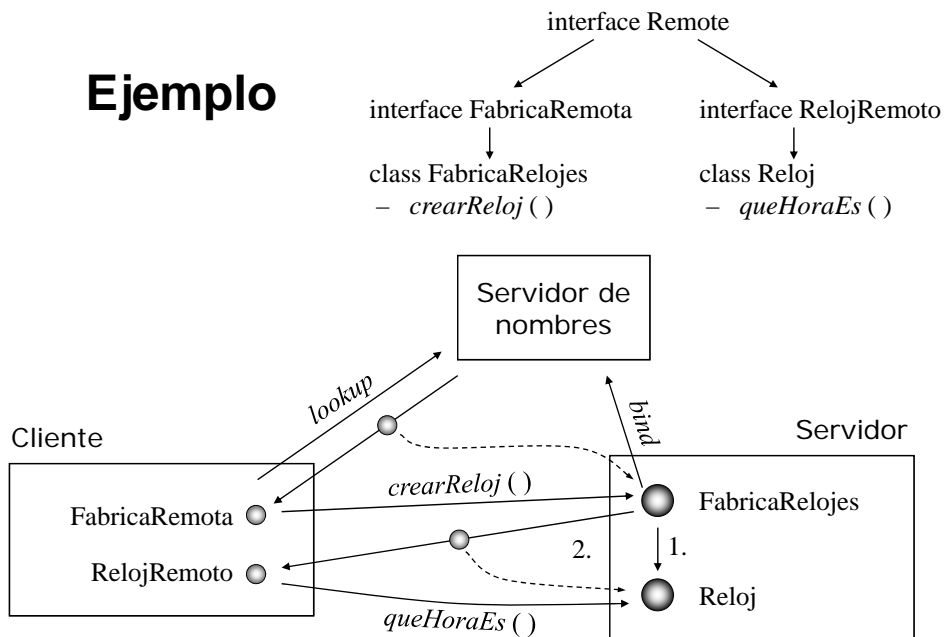
- Por defecto, rmiregistry utiliza el puerto 1099
- Se pueden arrancar otros rmiregistry utilizando puertos distintos:
`rmiregistry puerto`
- Para hacer rebind y lookup en un rmiregistry con puerto distinto de 1099, utilizar `"//host:puerto/nombreObjeto"` como argumento
- Un objeto remoto sólo puede registrar su nombre en un rmiregistry local, para evitar que un cliente pueda alterar el registro de nombres

Otra forma de obtener stubs: objetos fábrica

- Forma frecuente de obtener stubs en los clientes
- La obtención de stubs va unida a la creación de los objetos a petición del cliente
- Fábricas: objetos con métodos para crear objetos de las distintas clases de objetos remotos en un servidor
- El servidor crea un único objeto, una fábrica, y le asocia un nombre
- El cliente accede por nombre al objeto fábrica
- El cliente solicita al objeto fábrica la creación de otros objetos
- La fábrica crea objetos y devuelve stubs al cliente

47

Ejemplo



48

FabricaRemota.java

```
import java.rmi.*;

public interface FabricaRemota extends Remote {
    public RelojRemoto crearReloj (String zona)
        throws RemoteException;
}
```

49

Servidor.java

```
class Reloj implements RelojRemoto {
    ...
}
    ← rmic Reloj FabricaReloj
    ←
class FabricaReloj implements FabricaRemota {
    FabricaReloj () throws RemoteException {
        UnicastRemoteObject.exportObject (this);
    }
    public RelojRemoto crearReloj (String zona)
        throws RemoteException {
        return new Reloj (zona);
    }
}

public class Servidor {
    public static void main (String args[]) throws Exception {
        FabricaReloj fabrica = new FabricaReloj ();
        Naming.rebind ("Fabrica_relojes", fabrica);
    }
}
```

50

Cliente.java

```
import java.rmi.Naming;  
  
public class Cliente {  
    public static void main (String args[]) throws Exception {  
        FabricaRemota fabrica = (FabricaRemota) Naming.lookup (  
            "//ejemplo.eps.uam.es/Fabrica de Relojes");  
        RelojRemoto reloj1 = fabrica.crearReloj ("Europe/Madrid");  
        RelojRemoto reloj2 = fabrica.crearReloj ("Europe/Lisbon");  
        System.out.println ("Son las "  
            + reloj1.queHoraEs ()  
            + ", las "  
            + reloj2.queHoraEs ()  
            + " en Canarias");  
    }  
}
```

51

Paso de objetos por valor

- Objetos locales pueden ser pasados como argumentos a métodos remotos o devueltos como valor de retorno de los métodos remotos
- Se envía una copia del objeto
 - El objeto se serializa en el programa de salida en forma de stream de bytes
 - El programa que recibe el objeto restaura el objeto a partir del stream
 - RMI se ocupa de la serialización de forma transparente para el programador
- El paso de objetos remotos es distinto: por referencia en lugar de por valor
 - Los objetos remotos se convierten en stubs al pasar del servidor al cliente
- ¿Cualquier objeto se puede pasar por valor?
 - Sólo objetos de clases que implementan `java.io.Serializable`

52

Serialización de objetos

- Empaquetamiento de objetos en forma de stream de bytes
- `java.io.Serializable` no declara ningún método
 - Se trata sólo de marcar clases como serializables
 - No se necesita añadir nueva funcionalidad
- La funcionalidad de serialización es genérica y está contenida en las clases `Object{Input|Output}Stream` y los métodos correspondientes `default{Read|Write}Object`
- La serialización por defecto guarda:
 - Información sobre la clase del objeto y el host donde se encuentra
 - Valores de las variables que no sean `static` ni `transient` (serializadas recursivamente si son objetos a su vez)
- Modificar la serialización por defecto: `readObject(ObjectInputStream)`, `writeObject(ObjectOutputStream)`
- ¿Las clases de la librería estándar de java son serializables?
 - La mayoría sí, algunas no (p.e. `java.io.FileDescriptor`)

53

Manipulación de objetos por valor en el programa de llegada

- Para que tenga sentido enviar objetos por valor, el programa al que llegan debe poder manipularlos, i.e. invocar sus métodos
- Esto significa que el programa receptor incluye (de antemano) código en que se invocan métodos de los objetos pasados por valor
- El receptor puede tener el código que implementa los métodos o bien cargar la clase dinámicamente (la obtiene del host de salida)
- Si el receptor no tiene las clases es necesario definir una interfaz para estos objetos, compartida por ambos programas, emisor y receptor
- Así un cliente puede transferir al servidor la ejecución de programas arbitrarios definidos en el cliente siempre que se ajusten a una interfaz

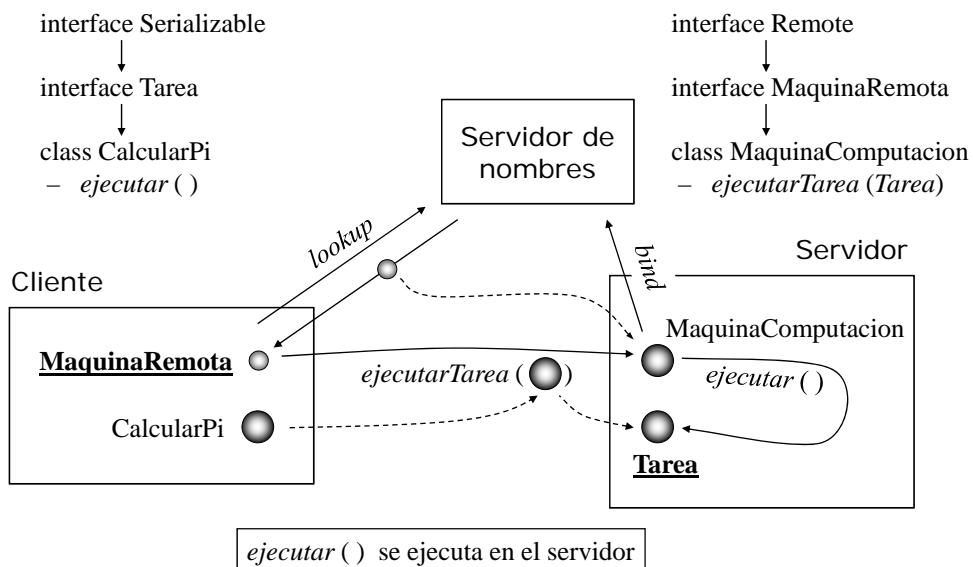
54

Acceso remoto a las clases

- Un programa puede obtener automáticamente por la web el código de las clases de objetos serializables recibidos desde otro programa remoto
- Para ello es necesario que el host de origen tenga instalado un servidor de HTTP
- Es necesario arrancar el programa de origen con el parámetro
`-Djava.rmi.server.codebase=file: /<directorio>`
Indicando el directorio donde encontrar los .class
- El programa de llegada carga automáticamente las clases al recibir los objetos
- También es posible utilizar este mecanismo para que un cliente obtenga en tiempo de ejecución el código de las clases de stub de objetos remotos

55

Objetos serializados: ejemplo



56

MaquinaRemota.java

```
import java.rmi.*;

public interface MaquinaRemota extends Remote {
    Object ejecutarTarea (Tarea t) throws RemoteException;
}
```

Tarea.java

```
public interface Tarea extends java.io.Serializable {
    Object ejecutar ();
}
```

57

MaquinaComputacion.java

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

public class MaquinaComputacion extends UnicastRemoteObject
    implements MaquinaRemota {
    public MaquinaComputacion () throws RemoteException {}
    public Object ejecutarTarea (Tarea t)
        throws RemoteException {
        return t.ejecutar ();
    }
}
```

58

Servidor.java

```
import java.rmi.*;

public class Servidor {
    public static void main(String[] args) {
        try {
            MaquinaComputacion maq = new MaquinaComputacion ();
            Naming.rebind ("Maquina_computacion", maq);
            System.out.println ("Maquina lista");
        } catch (Exception e) { e.printStackTrace(); }
    }
}
```

59

CalcularPi.java

```
import java.math.*;

public class CalcularPi implements Tarea {
    private int decimales;
    public CalcularPi (int n) { decimales = n; }
    public Object ejecutar () {
        double pi = 0, i = 0, termino;
        do {
            termino = 4 * Math.pow(-1,i) * 1/(2*i+1);
            pi += termino;
            i++;
        }
        while (Math.abs (termino) >
            Math.pow (10, -decimales-1));
        return new Double (pi);
    }
}
```

60

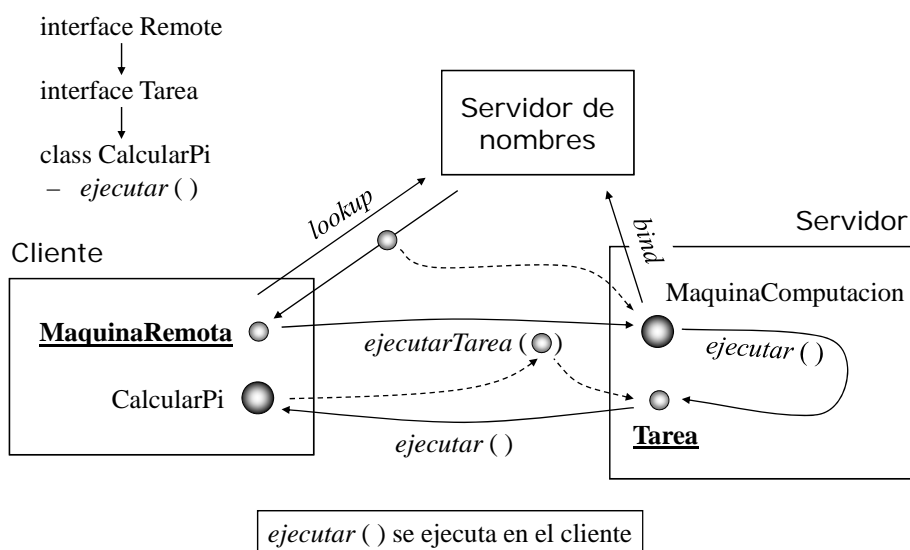
Cliente.java

```
import java.rmi.*;

public class Cliente {
    public static void main (String args[]) {
        try {
            MaquinaRemota maq = (MaquinaRemota) Naming.lookup (
                "//ejemplo.eps.uam.es/Maquina_computacion");
            CalcularPi tarea = new CalcularPi (4);
            Double pi = (Double) (maq.ejecutarTarea (tarea));
            System.out.println (pi);
        } catch (Exception e) { e.printStackTrace(); }
    }
}
```

61

Mismo ejemplo con objetos remotos



62

Tarea.java

```
import java.rmi.*;

public interface Tarea extends Remote {
    Object ejecutar () throws RemoteException;
}
```

63

CalcularPi.java

```
public class CalcularPi extends UnicastRemoteObject
    implements Tarea {
    private int decimales;
    public CalcularPi (int n) throws RemoteException {
        decimales = n;
    }
    public Object ejecutar () throws RemoteException {
        double pi = 0, i = 0, termino;
        do {
            termino = 4 * Math.pow(-1,i) * 1/(2*i+1);
            pi += termino; i++;
        }
        while (Math.abs (termino) > Math.pow (10,-decimales-1));
        return new Double (pi);
    }
}
```

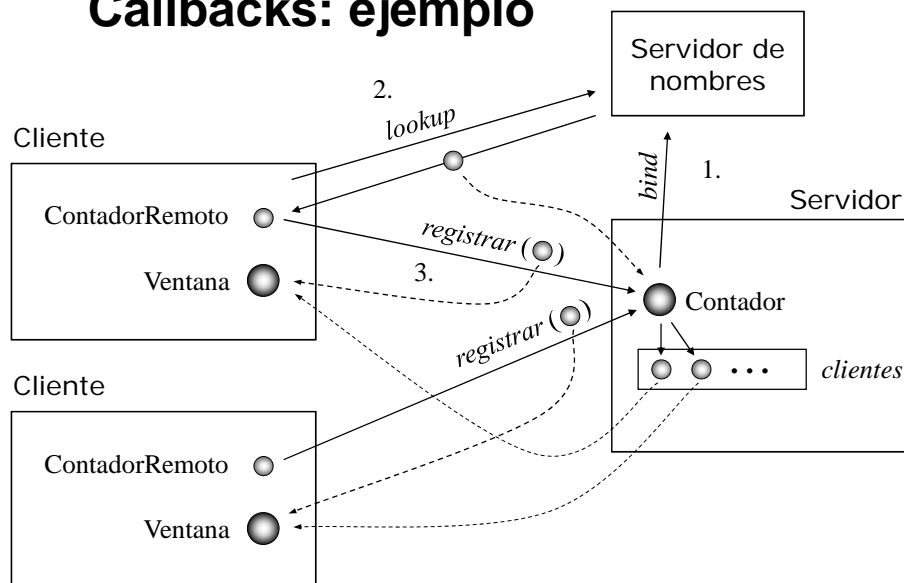
64

Callbacks

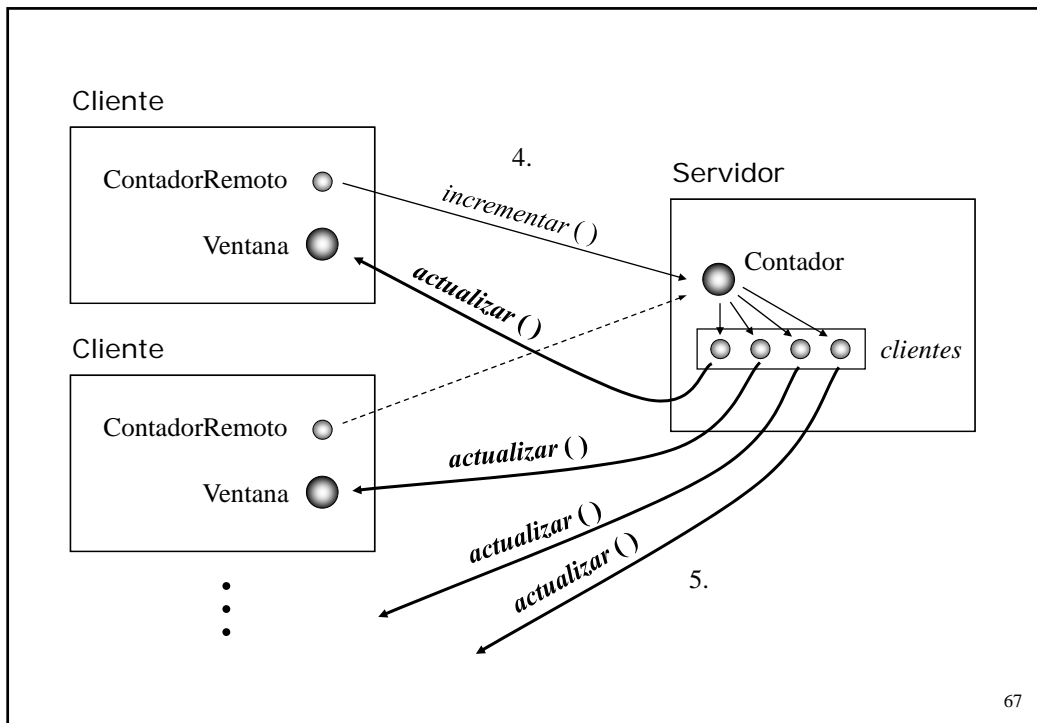
- El servidor actúa como cliente respecto de ciertas clases de sus clientes
- El servidor obtiene un stub a algún objeto remoto del cliente
- Normalmente el cliente proporciona este objeto remoto pasándolo como argumento de algún método de un objeto remoto del servidor
- El servidor invoca métodos sobre el objeto remoto del cliente
- La llamada desde el servidor está anidada dentro de la llamada del cliente
- Utilidad: mecanismos de notificación del servidor a sus clientes

65

Callbacks: ejemplo



66



67

```
ContadorRemoto.java  
import java.rmi.*;  
  
public interface ContadorRemoto extends Remote {  
    public void incrementar ()  
        throws RemoteException;  
    public void registrarCliente (VentanaRemota v)  
        throws RemoteException;  
    public void eliminarCliente (VentanaRemota v)  
        throws RemoteException;  
}  
  
VentanaRemota.java  
import java.rmi.*;  
  
public interface VentanaRemota extends Remote {  
    public void actualizar (int n)  
        throws RemoteException;  
}
```

68

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

public class Ventana extends Frame
    implements ActionListener, VentanaRemota {
    private Label label;
    private ContadorRemoto contador;
    public Ventana () throws Exception {
        add (label = new Label ("", Label.CENTER));
        Button boton = new Button ("Incrementar");
        add ("South", boton);
        boton.addActionListener (this);
        this.addWindowListener (new CerrarVentana ());
        UnicastRemoteObject.exportObject (this);
        contador = (ContadorRemoto) Naming.lookup (
            "//ejemplo.eps.uam.es/Servicio_contador");
        contador.registrarCliente (this);
    }
    ...
}
```

Ventana.java
69

```
...
public void actualizar (int n) throws RemoteException {
    label.setText ("Valor: " + n);
}

public void actionPerformed (ActionEvent e) {
    try { contador.incrementar (); }
    catch (RemoteException ex) { ex.printStackTrace (); }
}

class CerrarVentana extends WindowAdapter {
    public void windowClosing (WindowEvent e) {
        Ventana v = (Ventana) e.getSource ();
        try { contador.eliminarCliente (v); System.exit(0); }
        catch (RemoteException ex) { ex.printStackTrace(); }
    }
}
}
```

70

Cliente.java

```
import java.rmi.*;
public class Cliente {
    public static void main (String args[]) {
        try {
            Ventana v = new Ventana ();
            v.setVisible (true);
        } catch (Exception e) { e.printStackTrace (); }
    }
}
```

71

Contador.java

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

public class Contador extends UnicastRemoteObject
    implements ContadorRemoto {
    private int contador = 0;
    private ArrayList clientes = new ArrayList ();
    public Contador () throws RemoteException { }

    public synchronized void incrementar ()
        throws RemoteException {
        System.out.println ("Valor actual: " + ++contador);
        Iterator ventanas = clientes.iterator ();
        while (ventanas.hasNext ())
            ((VentanaRemota) ventanas.next ())
                .actualizar (contador);
    }
    ...
}
```

72

```
...
public void registrarCliente (VentanaRemota ventana)
    throws RemoteException {
    clientes.add (ventana);
    ventana.actualizar (contador);
}

public void eliminarCliente (VentanaRemota ventana)
    throws RemoteException {
    clientes.remove (ventana);
}
```

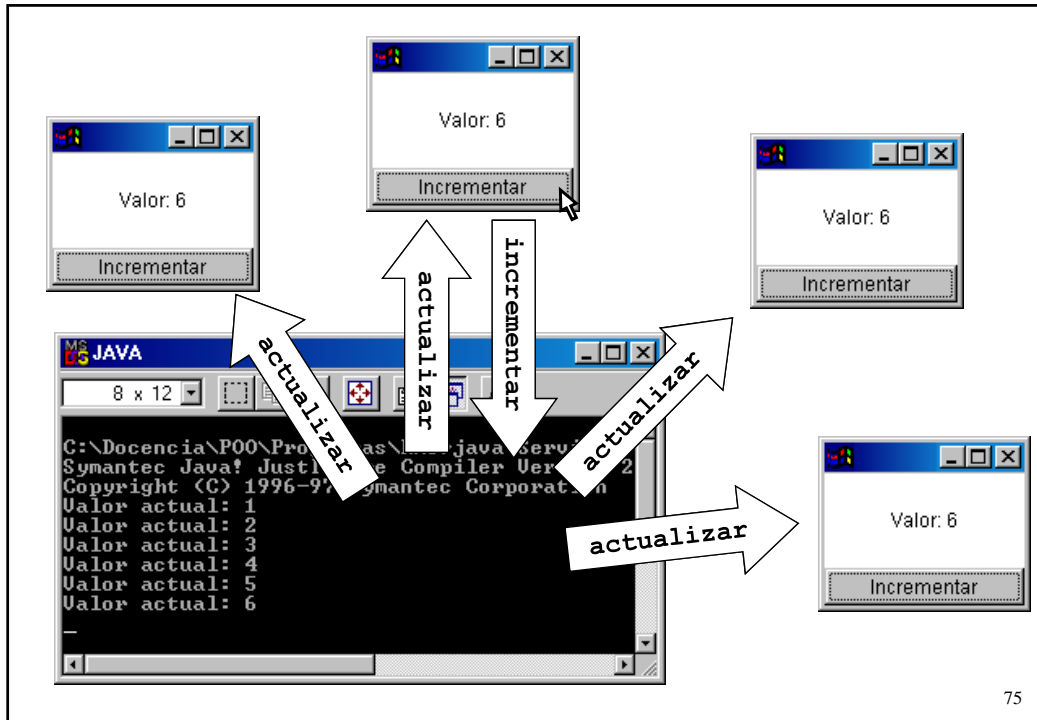
73

Servidor.java

```
import java.rmi.*;

public class Servidor {
    public static void main (String[] args) {
        try {
            Contador cont = new Contador ();
            Naming.rebind ("Servicio_contador", cont);
        } catch (Exception e) { e.printStackTrace (); }
    }
}
```

74



Objetos distribuidos

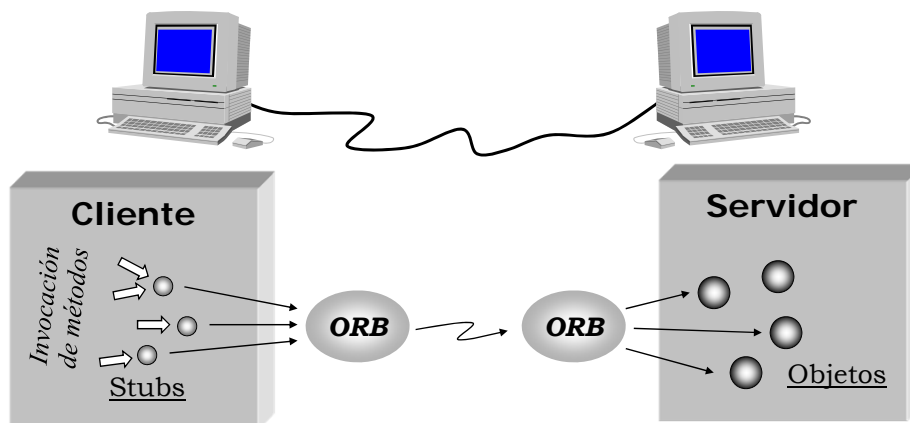
Common Object Request Broker Architecture (CORBA)

Common Object Request Broker Architecture

- Similar a RMI, estándar para objetos distribuidos
- Permite integrar programas en distintos lenguajes
 - Java, C++, C, Smalltalk, Ada, COBOL, etc.
- Incluye servicios adicionales integrados
 - Nombres, persistencia, eventos, etc.
- Creado por el consorcio OMG (Object Management Group)
 - OMG = más de 800 empresas: Sun, IBM, Apple, Digital, Netscape, Oracle, Borland, NCR, Siemens, ILOG, Telefónica I+D...
 - OMG fundada en 1988
 - CORBA 1.0: 1992, CORBA 2.0: 1997, CORBA 2.6: dic 2001
- JDK 1.2 / 1.4 incorpora CORBA 2.0 / 2.3 con el nombre de Java IDL
- Otros estándares existentes: competencia CORBA / RMI / DCOM

77

Objetos distribuidos CORBA



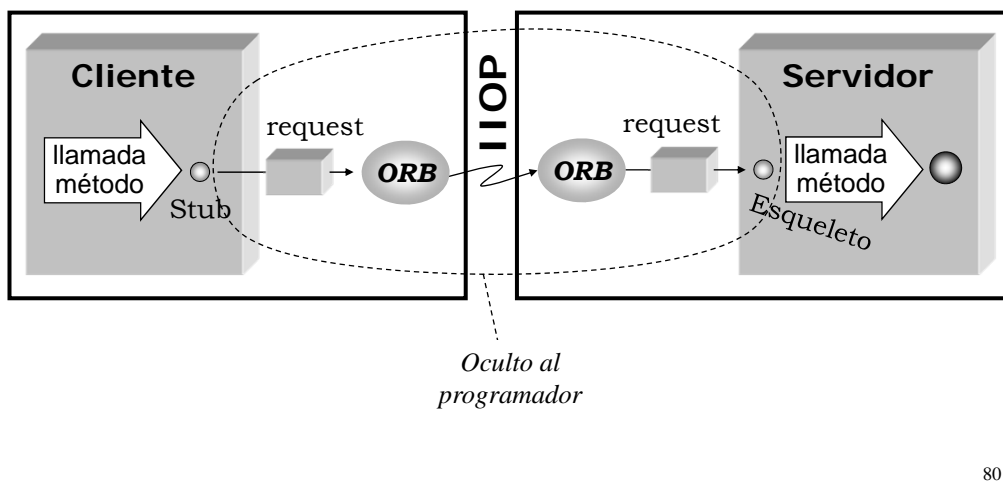
78

La arquitectura CORBA

- ORB: intermediario entre clientes y servidores
 - Transmite las peticiones cliente-servidor y las respuestas servidor-cliente
 - Se necesita un ORB en cada máquina (en Java IDL, uno en cada proceso)
- Stub: intermediario entre clientes y ORB
 - Recoge del cliente llamadas a métodos y las transmite al ORB
 - Una clase de stub por cada clase remota
- Esqueleto: intermediario entre ORB y objetos del servidor
 - Recibe llamadas del ORB y ejecuta los métodos correspondientes en el servidor sobre el objeto que corresponda

79

Arquitectura



80

Definición de interfaces: el lenguaje IDL

- Interface Definition Language
- Descripción de la interfaz de los objetos: operaciones y tipos de argumentos
- Equivalente a la definición de interfaces `Remote` en RMI
- Compatible con distintos lenguajes de programación
- Conceptos y sintaxis semejantes a C++ / Java
- IDL permite herencia entre interfaces
- IDL no admite sobreescritura de métodos o atributos heredados
- IDL no permite pasar objetos por valor (sólo structs)

81

Ejemplo

```
interface Cliente {
    attribute string nombre;
    attribute long dni;
};

interface Cuenta {
    attribute long saldo;
    attribute long numero;
    attribute Cliente titular;
    void ingreso (in long importe);
    void reintegro (in long importe);
};

interface CuentaCorriente : Cuenta {
    ...
};
```

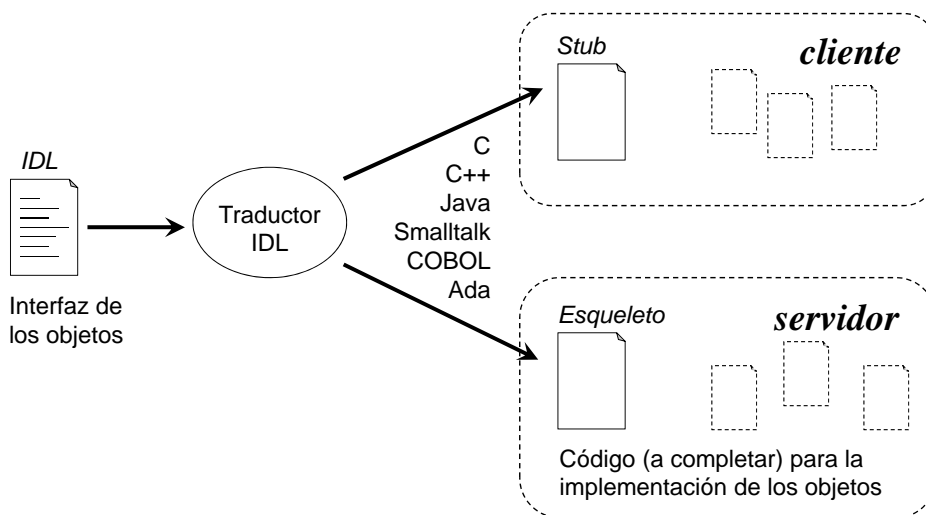
82

Tipos IDL

- Tipos básicos: `long`, `short`, `float`, `char`, `boolean`, `enum`, `string`, `any`
- Tipos compuestos: `struct`, `union`
- Tipos derivados: `sequence <tipo>`
- Tipos de objeto: `interface`

83

Compilación del código IDL



84

| Mapping IDL → Java | <u>IDL Construct</u> | <u>Java Construct</u> |
|---------------------------|-----------------------|---|
| | module | package |
| | interface | interface, helper and holder class |
| | constant | public static final |
| | boolean | boolean |
| | char, wchar | char |
| | octet | byte |
| | string, wstring | java.lang.String |
| | short, unsigned short | short |
| | long, unsigned long | int |
| | long long | long |
| | unsigned long long } | |
| | float | float |
| | double | double |
| | enum, struct, union | final class |
| | sequence, array | array |
| | exception | final class |
| | readonly attribute | method for accessing attribute |
| | readwrite attribute | methods for accessing and setting attribute |
| | operation | method |

85

Implementaciones de CORBA

Una implementación de CORBA incluye:

- Componentes:
 - Un traductor de IDL a distintos lenguajes de programación
 - Librerías para la generación y transmisión de mensajes entre procesos cuando los clientes invocan métodos de un objeto
 - Un ORB: puede ser un programa, una DLL o una clase
- A nivel de diseño:
 - Detalles no concretados por el estándar
 - Extensiones propias de la versión

86

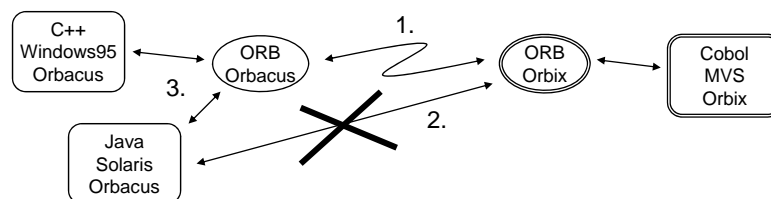
Productos CORBA

- Versiones comerciales:
 - Iona: Orbix
 - Visigenic: Visibroker
 - Digital: ObjectBroker
 - HP: ORB Plus
 - Chorus Systems: CHORUS/COOL ORB
 - IBM: ComponentBroker
- Versiones gratuitas:
 - Object Oriented Concepts: ORBacus
 - Olivetti Research Lab: OmniORB
 - Sun: Java IDL
- Ver también:
 - <http://www.cs.wustl.edu/~schmidt/corba-products.html>
 - <http://adams.patriot.net/~tvaesky/freecorba.html>

87

Compatibilidades e incompatibilidades

1. Distintas versiones de ORB se pueden comunicar entre sí (IIOP)
2. El código desarrollado para una versión de ORB no se puede comunicar directamente con otras versiones de ORB
3. Aplicaciones desarrolladas para una versión de CORBA en distintos lenguajes de programación, OS y hardware, pueden integrarse entre sí por medio de un mismo ORB

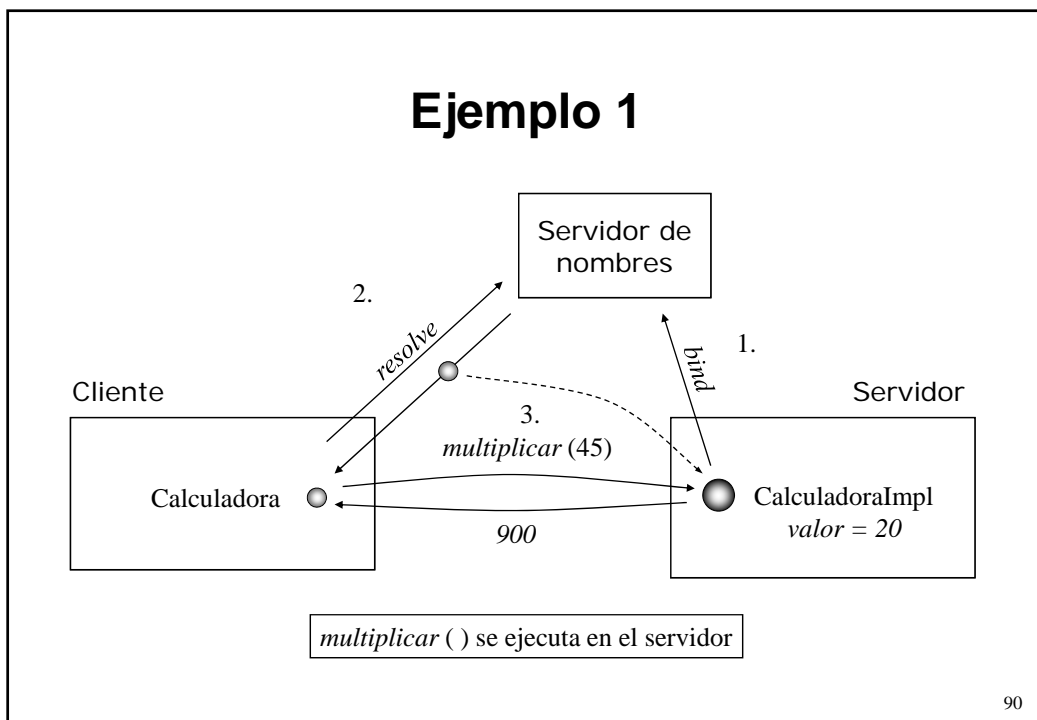


88

CORBA vs. RMI

| | |
|--------------|--|
| CORBA | <ul style="list-style-type: none">▪ Integración de aplicaciones legadas▪ La heterogeneidad puede ser deseable: lenguajes distintos para distintas necesidades▪ Servicios integrados |
| RMI | <ul style="list-style-type: none">▪ Facilidad de programación, portabilidad de las implementaciones▪ Paso de objetos por valor▪ Java incorpora en el propio lenguaje:<ul style="list-style-type: none">– Funcionalidad web– Interfaces de usuario– ...▪ RMI + JNI permite integrar distintos lenguajes, sin embargo...<ul style="list-style-type: none">– La integración es a nivel de llamadas a funciones, no de objetos– Programación mucho más tediosa que CORBA▪ JDK 1.3 adopta IIOP |

89



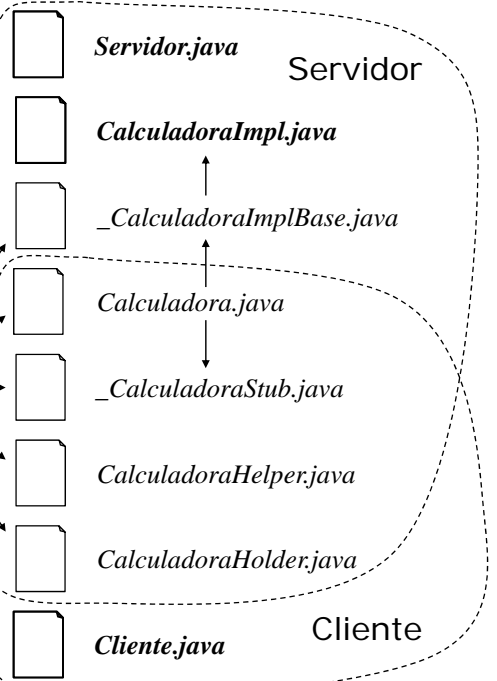
Calculadora.idl

```
interface Calculadora {  
    attribute double valor;  
    double sumar (in double x);  
    double multiplicar (in double x);  
    double raiz ();  
    void reset ();  
};
```

91

Compilación del IDL

idltojava Calculadora.idl



92

Calculadora.java

```
//-----  
// Código generado por idltojava  
//-----  
  
public interface Calculadora  
    extends org.omg.CORBA.Object,  
           org.omg.CORBA.portable.IDLEntity {  
    double valor ();  
    void valor (double arg);  
    double sumar (double n);  
    double multiplicar (double n);  
    double raiz ();  
    void reset ();  
}
```

93

_CalculadoraImplBase.java

```
//-----  
// Código generado por idltojava  
//-----  
  
public abstract class _CalculadoraImplBase  
    extends org.omg.CORBA.DynamicImplementation  
    implements Calculadora {  
    ...  
}
```

94

CalculadoraImpl.java

```
import org.omg.CORBA.*;

class CalculadoraImpl extends _CalculadoraImplBase {
    private double valor;
    public double valor () { return valor; }
    public void valor (double x) { valor = x; }
    ...
}
```

95

```
...
public double sumar (double x) {
    valor += x;
    System.out.println ("+ " + x + " = " + valor);
    return valor;
}
public double multiplicar (double x) {
    valor *= x;
    System.out.println ("* " + x + " = " + valor);
    return valor;
}
public double raiz () {
    valor = Math.sqrt (valor);
    System.out.println ("raiz = " + valor);
    return valor;
}
public void reset () { valor = 0; }
}
```

96

Servidor.java

```

import org.omg.CORBA.*;
import org.omg.CosNaming.*;

public class Servidor {
    public static void main (String args[]) {
        try {
            ORB orb = ORB.init (args, null);
            CalculadoraImpl calc = new CalculadoraImpl ();
            orb.connect (calc);
            System.out.println ("Creada calculadora\nValor: " +
                               calc.valor ());

            NamingContext naming = NamingContextHelper.narrow (
                orb.resolve_initial_references ("NameService"));
            NameComponent nombre[] =
                { new NameComponent ("Mi calculadora", "") };
            naming.rebind (nombre, calc);

            java.lang.Object sync = new java.lang.Object ();
            synchronized (sync) { sync.wait(); }
        } catch (Exception e) { e.printStackTrace (); }
    }
}

```

97

Cliente.java

```

import org.omg.CORBA.*;
import org.omg.CosNaming.*;

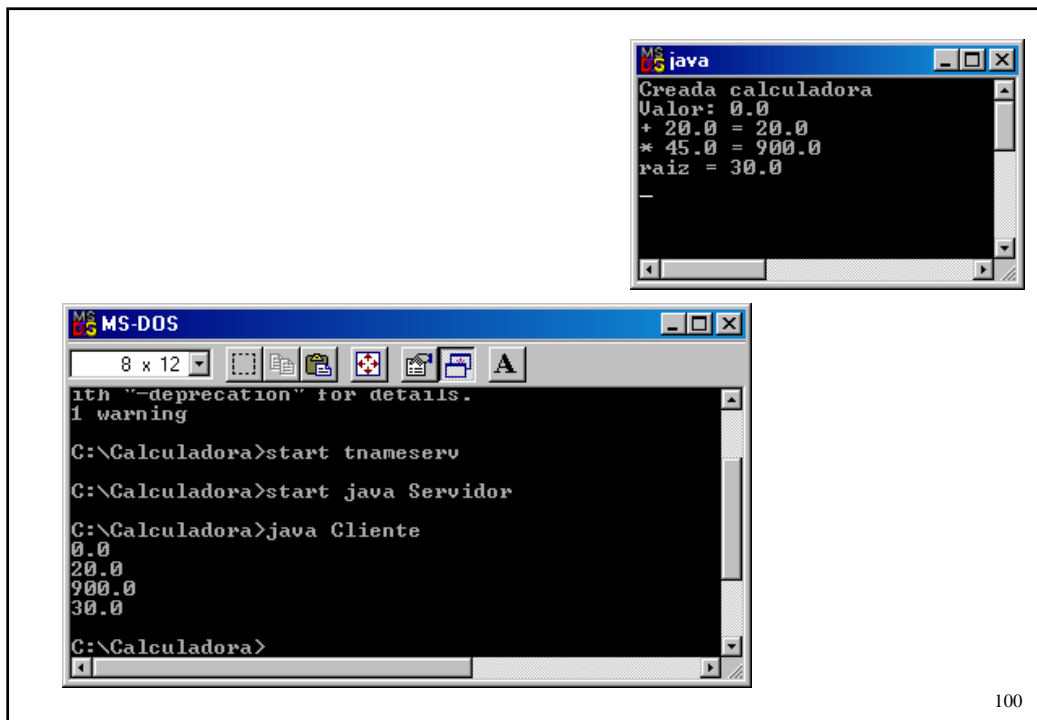
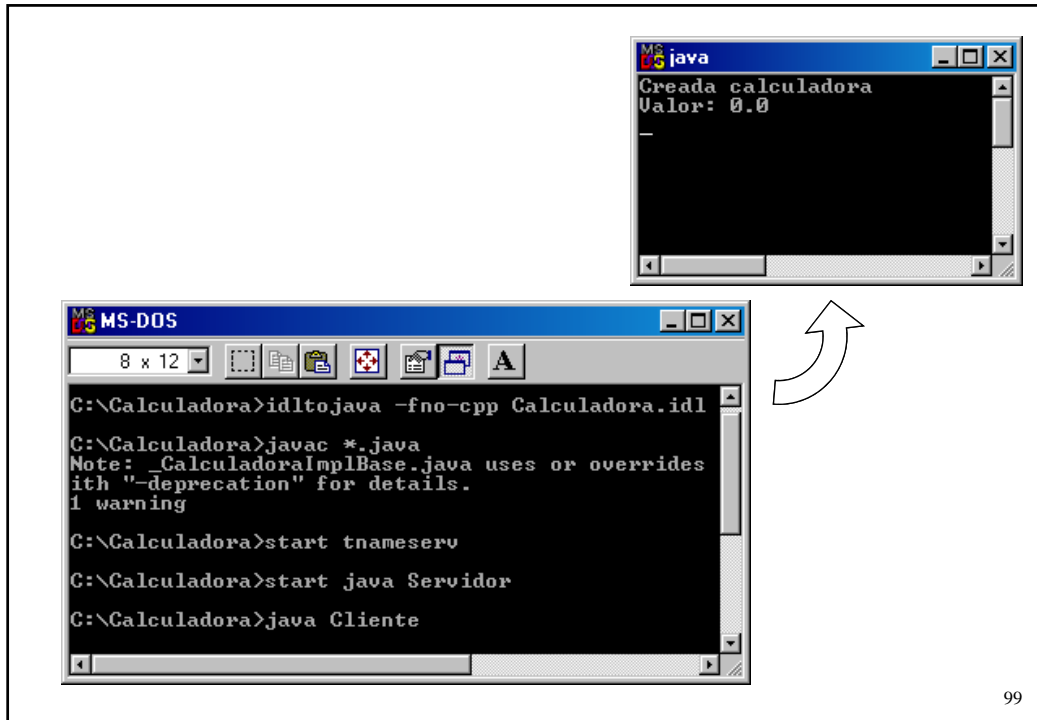
public class Cliente {
    public static void main (String args []) {
        try {
            ORB orb = ORB.init (args, null);

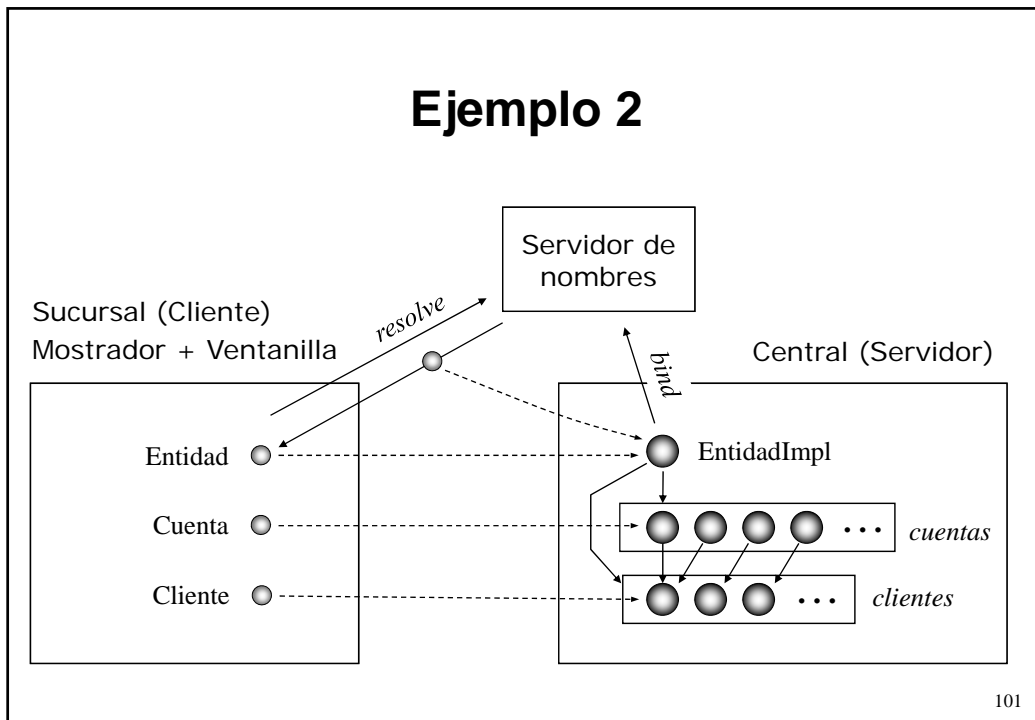
            NamingContext naming = NamingContextHelper.narrow (
                orb.resolve_initial_references ("NameService"));
            NameComponent nombre[] =
                { new NameComponent ("Mi calculadora", "") };
            Calculadora calc = CalculadoraHelper.narrow (
                naming.resolve (nombre));

            System.out.println (calc.valor ());
            calc.sumar (20);          System.out.println (calc.valor ());
            calc.multiplicar (45);  System.out.println (calc.valor ());
            calc.raiz ();           System.out.println (calc.valor ());
        } catch (Exception e) { e.printStackTrace (); }
    }
}

```

98





```
interface Cliente {
    attribute string nombre;
    attribute long dni;
};

interface Cuenta {
    exception SaldoInsuficiente { long saldo; };
    attribute long saldo;
    attribute long numero;
    attribute Cliente titular;
    void ingreso (in long importe);
    void reintegro (in long importe) raises (SaldoInsuficiente);
};

interface Entidad {
    attribute string nombre;
    Cuenta abrirCuenta (in Cliente titular);
    Cuenta buscarCuenta (in long numero);
    void cancelarCuenta (in long numero);
    Cliente nuevoCliente (in string nombre, in long dni);
    Cliente buscarCliente (in long dni);
    void bajaCliente (in long dni);
};
```

Banco.idl

102

SaldoInsuficiente.java

```
//-----  
// Código generado por idltojava  
//-----  
  
package CuentaPackage;  
  
public final class SaldoInsuficiente  
    extends org.omg.CORBA.UserException  
    implements org.omg.CORBA.portable.IDLEntity {  
    public int saldo;  
    public SaldoInsuficiente () { super (); }  
    public SaldoInsuficiente (int __saldo) {  
        super ();  
        saldo = __saldo;  
    }  
}
```

103

EntidadImpl.java

```
import org.omg.CORBA.*;  
import java.util.*;  
  
public class EntidadImpl extends _EntidadImplBase {  
    ORB orb;  
    private String nombre;  
    private ArrayList cuentas = new ArrayList ();  
    private ArrayList clientes = new ArrayList ();  
    EntidadImpl (String str, ORB orb) {  
        nombre = str;  
        this.orb = orb;  
        orb.connect (this);  
    }  
  
    public String nombre () { return nombre; }  
    public void nombre (String str) { nombre = str; }  
    ...  
}
```

104

```

...
public Cuenta abrirCuenta (Cliente titular) {
    CuentaImpl cuenta = new CuentaImpl (titular, orb);
    cuentas.add (cuenta);
    return cuenta;
}

public Cuenta buscarCuenta (int numero) {
    Iterator iter = cuentas.iterator ();
    while (iter.hasNext ()) {
        Cuenta cuenta = (Cuenta) iter.next ();
        if (cuenta.numero () == numero) return cuenta;
    }
    return null; // Exception
}

public void cancelarCuenta (int numero) {
    cuentas.remove (buscarCuenta (numero));
}
...

```

105

```

...
public Cliente nuevoCliente (String nombre, int dni) {
    ClienteImpl cliente = new ClienteImpl (nombre, dni, orb);
    clientes.add (cliente);
    return cliente;
}

public Cliente buscarCliente (int dni) {
    Iterator iter = clientes.iterator ();
    while (iter.hasNext ()) {
        Cliente cliente = (Cliente) iter.next ();
        if (cliente.dni () == dni) return cliente;
    }
    return null; // Exception
}

public void bajaCliente (int dni) {
    clientes.remove (buscarCliente (dni));
}
};

```

106

```
import org.omg.CORBA.*;

public class CuentaImpl
    extends _CuentaImplBase {
    private static int ncuentas = 0;
    private int numero, saldo;
    private Cliente titular;
    CuentaImpl (Cliente clt, ORB orb) {
        numero = ++ncuentas;
        saldo = 0; titular = clt;
        orb.connect (this);
    }

    public int saldo () { return saldo; }
    public void saldo (int n) { saldo = n; }
    public int numero () { return numero; }
    public void numero (int n) { numero = n; }
    public Cliente titular () { return titular; }
    public void titular (Cliente clt) { titular = clt; }
    ...
}
```

CuentaImpl.java

107

```
...
public synchronized void ingreso (int importe) {
    saldo += importe;
}

public synchronized void reintegro (int importe)
    throws CuentaPackage.SaldoInsuficiente {
    if (saldo < importe)
        throw new CuentaPackage.SaldoInsuficiente (saldo);
    saldo -= importe; }
}
```

108

ClientImpl.java

```
import org.omg.CORBA.*;

public class ClienteImpl extends _ClienteImplBase {
    private String nombre;
    private int dni;
    ClienteImpl (String str, int n, ORB orb) {
        nombre = str;
        dni = n;
        orb.connect (this);
    }

    public String nombre () { return nombre; }
    public void nombre (String str) { nombre = str; }
    public int dni () { return dni; }
    public void dni (int n) { dni = n; }
}
```

109

Central.java (servidor)

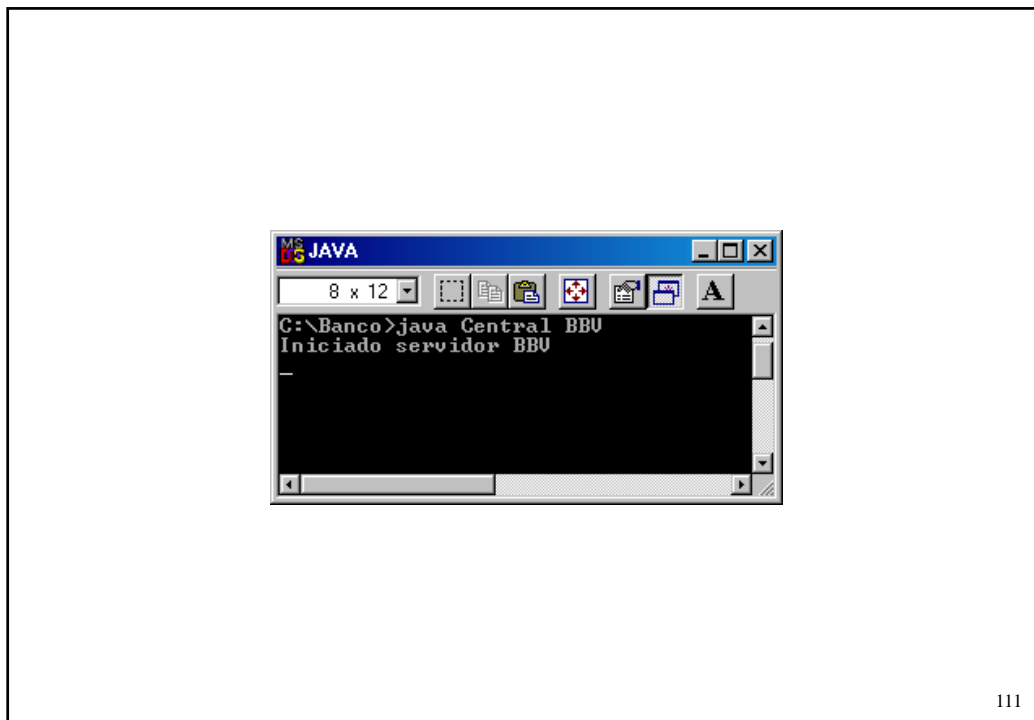
```
import org.omg.CORBA.*;
import org.omg.CosNaming.*;

public class Central {
    public static void main (String args[]) {
        try {
            ORB orb = ORB.init (args, null);
            EntidadImpl entidad = new EntidadImpl (args[0], orb);

            NamingContext naming = NamingContextHelper.narrow (
                orb.resolve_initial_references ("NameService"));
            NameComponent nombre[] =
                { new NameComponent (args[0], "") };
            naming.rebind (nombre, entidad);

            System.out.println ("Iniciado servidor " + args[0]);
            java.lang.Object sync = new java.lang.Object ();
            synchronized (sync) { sync.wait(); }
        } catch (Exception e) { e.printStackTrace (); }
    }
}
```

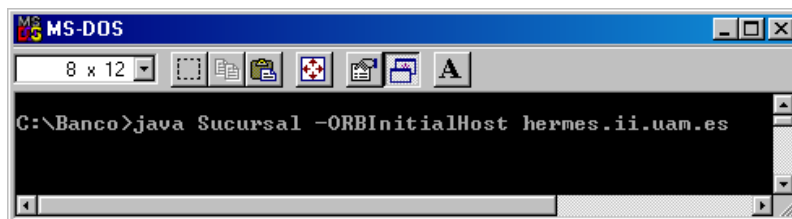
110



111

Sucursal.java (main cliente)

```
import org.omg.CORBA.*;  
  
public class Sucursal {  
    public static void main (String args []) {  
        try {  
            ORB orb = ORB.init (args, null);  
            new Mostrador (orb) .setVisible (true);  
            new Ventanilla (orb) .setVisible (true);  
        } catch (Exception e) { e.printStackTrace (); }  
    }  
}
```



112

```
import org.omg.CORBA.*; import org.omg.CosNaming.*;
import java.awt.*; import java.awt.event.*;

public class Mostrador extends Frame implements ActionListener {
    TextField nombreEntidad, nombreTitular, dniTitular;
    Label numeroCuenta, saldoCuenta;
    Entidad entidad;
    Cuenta cuenta;
    NamingContext naming;
    Mostrador (ORB orb) throws Exception {
        setTitle ("Mostrador");
        add (new Label ("Nombre entidad"));
        add (nombreEntidad = new TextField ("", 30));
        add (new Label ("Nombre del titular"));
        add (nombreTitular = new TextField (""));
        add (new Label ("DNI"));
        add (dniTitular = new TextField (""));
        add (new Label ("Nº cuenta:"));
        add (numeroCuenta = new Label (""));
        add (new Label ("Saldo:"));
        add (saldoCuenta = new Label (""));
        ...
    }
}
```

Mostrador.java
(cliente)

113

```
...
Panel buttonPanel = new Panel ();
add (buttonPanel);
Button b = new Button ("Crear");
buttonPanel.add (b);
b.addActionListener (this);
b = new Button ("Cerrar");
buttonPanel.add (b);
b.addActionListener (this);
... // definir layout

naming = NamingContextHelper.narrow (
    orb.resolve_initial_references ("NameService"));
}

void contactarEntidad () throws Exception {
    NameComponent nombre[] =
        { new NameComponent (nombreEntidad.getText (), "") };
    entidad = EntidadHelper.narrow (naming.resolve (nombre));
}
...
}
```

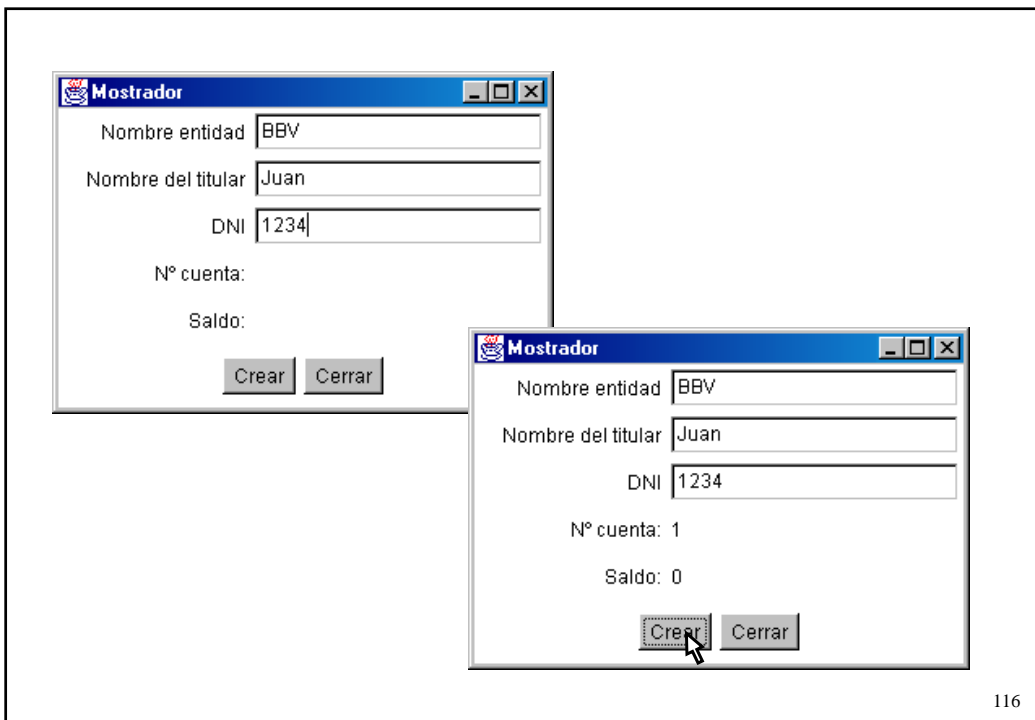
114

```
...
public void actionPerformed (ActionEvent e) {
    if (e.getActionCommand () == "Cerrar") dispose ();
    else try {
        contactarEntidad ();

        String nombre = nombreTitular.getText ();
        int dni = Integer.valueOf (dniTitular.getText ())
            .intValue ();
        Cliente titular = entidad.buscarCliente (dni);
        if (titular == null)
            titular = entidad.nuevoCliente (nombre, dni);

        cuenta = entidad.abrirCuenta (titular);
        nombreTitular.setText (cuenta.titular().nombre ());
        numeroCuenta.setText (String.valueOf(cuenta.numero()));
        saldoCuenta.setText (String.valueOf(cuenta.saldo()));
    } catch (Exception ex) { ex.printStackTrace (); }
}
```

115



116

Ventanilla.java (cliente)

```
import org.omg.CORBA.*; import org.omg.CosNaming.*;
import java.awt.*; import java.awt.event.*;

public class Ventanilla extends Frame
    implements ActionListener {
    TextField nombreEntidad, numeroCuenta, importeOperacion;
    Label nombreTitular, dniTitular, saldoCuenta;
    Entidad entidad;
    Cuenta cuenta;
    NamingContext naming;
    ...
}
```

117

```
...
Ventanilla (ORB orb) throws Exception {
    setTitle ("Ventanilla");
    add (new Label ("Nombre entidad"));
    add (nombreEntidad = new TextField ("", 30));
    add (new Label ("Nº cuenta"));
    add (numeroCuenta = new TextField (""));
    add (new Label ("Importe"));
    add (importeOperacion = new TextField ("0"));
    add (new Label ("Nombre del titular:"));
    add (nombreTitular = new Label (""));
    add (new Label ("DNI:"));
    add (dniTitular = new Label (""));
    add (new Label ("Saldo:"));
    add (saldoCuenta = new Label (""));
    ...
}
```

118

```

...
Panel buttonPanel = new Panel ();
addPart (buttonPanel, layout, c);
Button b = new Button ("Saldo");
buttonPanel.add (b);
b.addActionListener (this);
b = new Button ("Ingreso");
buttonPanel.add (b);
b.addActionListener (this);
b = new Button ("Reintegro");
buttonPanel.add (b);
b.addActionListener (this);
b = new Button ("Cerrar");
buttonPanel.add (b);
b.addActionListener (this);
... // definir layout

naming = NamingContextHelper.narrow (
    orb.resolve_initial_references ("NameService"));
} // Fin de constructor
...

```

119

```

...
void contactarEntidad () throws Exception {
    NameComponent nombre[] =
        { new NameComponent (nombreEntidad.getText (), "") };
    entidad = EntidadHelper.narrow (naming.resolve (nombre));
}

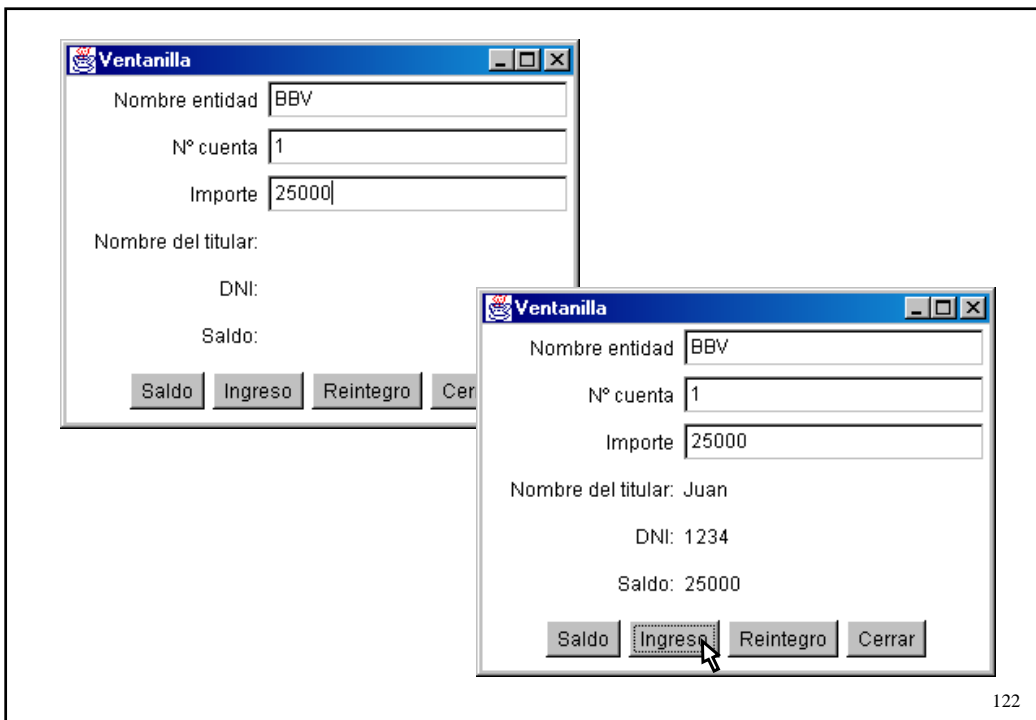
public void actionPerformed (ActionEvent e) {
    String command = e.getActionCommand ();
    if (command == "Cerrar") dispose ();
    else try {
        contactarEntidad ();
        int numero = Integer.valueOf (numeroCuenta.getText ())
            .intValue ();
        cuenta = entidad.buscarCuenta (numero);
        nombreTitular.setText (cuenta.titular () .nombre ());
        dniTitular.setText (String.valueOf (
            cuenta.titular () .dni ());
        ...
    }
}

```

120

```
...
int importe =
    Integer.valueOf (importeOperacion.getText ())
                .intValue ();
if (command == "Ingreso")
    cuenta.ingreso (importe);
else if (command == "Reintegro")
    cuenta.reintegro (importe);
saldoCuenta.setText (String.valueOf (cuenta.saldo ()));
} catch (Exception ex) { ex.printStackTrace (); }
}
```

121



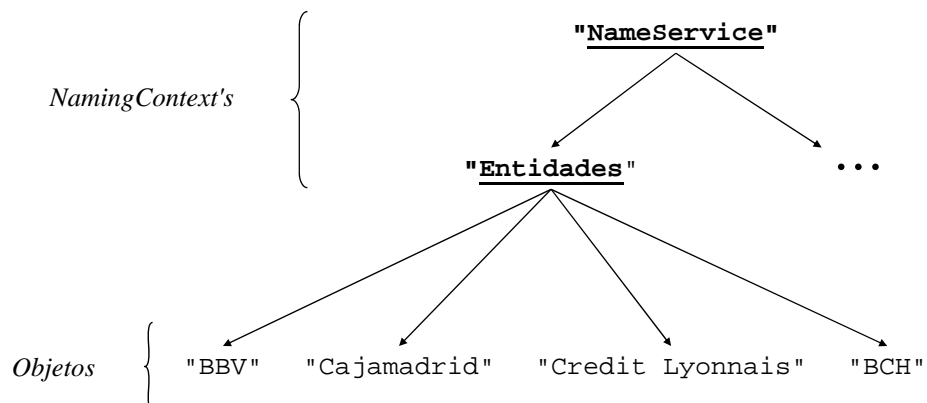
122

Servicio de nombres

- Más sofisticado que el de RMI
- Nombre = array de nombres \equiv rama (path) en estructura de tipo árbol de directorios
 - Los elementos intermedios del array son nombres que corresponden a directorios: `NamingContext`'s
 - El último elemento del array es el nombre que corresponde a un objeto
- Previamente es necesario crear los `NamingContext`
- Browsing de los nombres existentes en el servidor de nombres
 - A menudo se desconocen los nombres de antemano
 - Es posible obtener una lista de todos los nombres existentes bajo un determinado `NamingContext` (como hacer 'dir' de un directorio)

123

Ejemplo



124

```
public class Central {
    public static void main (String args[]) {
        try {
            ORB orb = ORB.init (args, null);
            EntidadImpl entidad = new EntidadImpl (args[0], orb);

            NamingContext naming = NamingContextHelper.narrow (
                orb.resolve_initial_references ("NameService"));
            NameComponent nombre[] = {
                new NameComponent ("Entidades", "") };
            try { naming.resolve (nombre); }
            catch (NotFound ex) { naming.bind_new_context (nombre); }
            NameComponent nombreEnt[] = {
                new NameComponent ("Entidades", ""),
                new NameComponent (args[0], "")
            };
            naming.rebind (nombreEnt, entidad);

            System.out.println ("Iniciado servidor " + args[0]);
            java.lang.Object sync = new java.lang.Object ();
            synchronized (sync) { sync.wait(); }
        } catch (Exception e) { e.printStackTrace (); }
    }
}
```

Central

125

Mostrador, Ventanilla (I)

```
void contactarEntidad () throws Exception {
    NameComponent nombreEnt[] = {
        new NameComponent ("Entidades", ""),
        new NameComponent (nombreEntidad.getText (), "")
    };
    entidad = EntidadHelper.narrow (naming.resolve (nombreEnt));
}
```

126

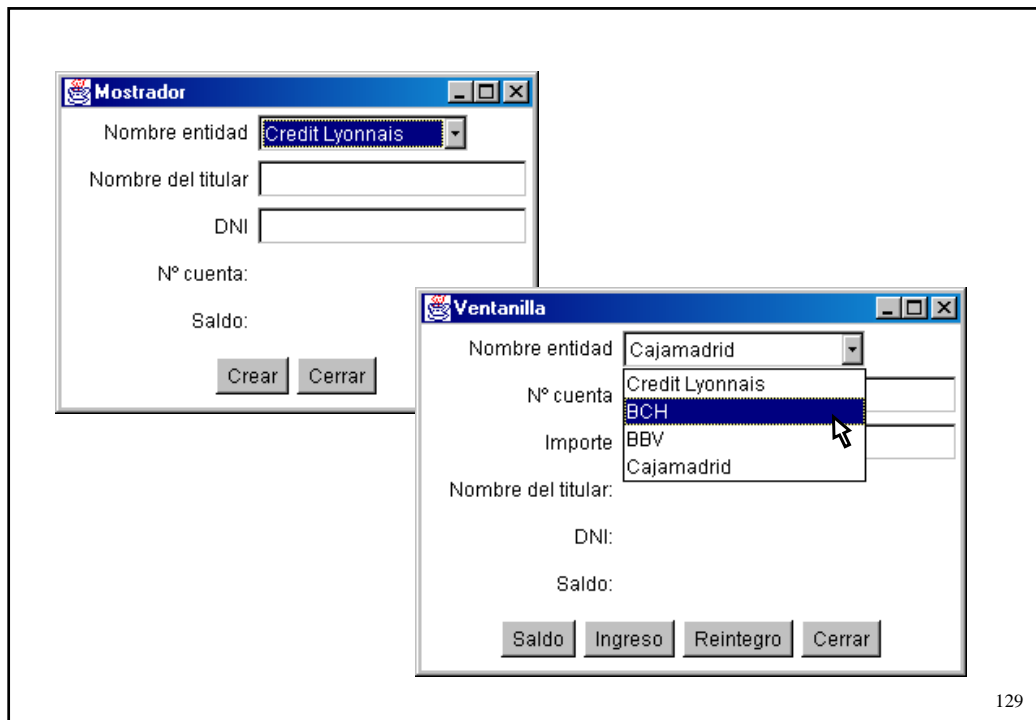
```
public class { Mostrador | Ventanilla }
    extends Frame implements ActionListener {
Choice nombreEntidad;
    ...
    Ventanilla (ORB orb) throws Exception {
        ...
        add (nombreEntidad = new Choice ());
        ...
        naming = NamingContextHelper.narrow (
            orb.resolve_initial_references ("NameService"));
        NameComponent nombresEnt[] = {
            new NameComponent ("Entidades", "") };
        NamingContext entidades = NamingContextHelper.narrow (
            naming.resolve (nombresEnt));
        BindingIteratorHolder iter = new BindingIteratorHolder ();
        BindingListHolder nombres = new BindingListHolder ();
        entidades.list (1000, nombres, iter);
        for (int i = 0; i < nombres.value.length; i++) {
            String nombre = nombres.value[i].binding_name[0].id;
            nombreEntidad.addItem (nombre);
        }
    }
    ...
}
```

**Mostrador,
Ventanilla
(II)**

127

```
...
void contactarEntidad () throws Exception {
    NameComponent nombreEnt[] = {
        new NameComponent ("Entidades", ""),
        new NameComponent (nombreEntidad.getSelectedItem (), "")
    };
    entidad = EntidadHelper.narrow (
        naming.resolve (nombreEnt));
}
...
}
```

128



Otras formas de programación distribuida en Java

Lectura y escritura en una URL

- Establecer conexión a URL
 - `url.openConnection()` → `URLConnection`
- Obtener streams de lectura / escritura
 - `getInputStream()` → `InputStream`
 - `getOutputStream()` → `OutputStream`
- Estos métodos emiten `IOException`

133

Lectura de una URL: ejemplo

```
import java.net.*;
import java.io.*;
public class LeerURL {
    public static void main(String[] args) throws Exception {
        URL uam = new URL ("http://www.uam.es");
        URLConnection conexion = uam.openConnection ();
        BufferedReader entrada = new BufferedReader (
            new InputStreamReader (conexion.getInputStream ()));
        String linea;
        while ((linea = entrada.readLine()) != null)
            System.out.println(linea);
        entrada.close();
    }
}
```

137

```
<html>
<head>
...
<title>Universidad Autónoma de Madrid</title>
...
</head>

<body>

...

</body>
</html>
```

135

Escritura en una URL: ejemplo

```
public static void main (String[] args) throws Exception {
    URL sun = new URL ("http://java.sun.com/cgi-bin/backwards");
    URLConnection conexion = sun.openConnection();
    conexion.setDoOutput (true); // permitir escribir

    PrintWriter salida =
        new PrintWriter (conexion.getOutputStream ());
    salida.println ("string=" + args[0]);
    salida.close ();

    BufferedReader entrada = new BufferedReader (
        new InputStreamReader (conexion.getInputStream ()));
    String linea;
    while ((linea = entrada.readLine ()) != null)
        System.out.println (linea);
    entrada.close();
}
```

136

Servlets

- Similares a los CGI
 - Sintaxis más sencilla
p.e. en C: `getenv (QUERY_STRING) → "param1=valor1¶m2=valor2&..."`
 - Necesitan un servidor (web) específico, p.e. Tomcat (ver www.apache.org)
- Se ejecutan en el servidor
- Se acceden (invocan) mediante una URL
La sintaxis de la URL depende del servidor de servlets
- Incluidos en Java J2EE: `packages javax.servlet.xxx`
- Típicamente se utilizan para la generación dinámica de páginas web

137

Crear un servlet

- Definir una subclase de `javax.servlet.http.HttpServlet`
- Implementar alguno de los métodos:
 - `doGet (ServletRequest request, ServletResponse response)`
 - `doPost (ServletRequest request, ServletResponse response)`
 - `service (ServletRequest request, ServletResponse response)`
- Dar de alta el servlet en el servidor de servlets
(Ver documentación del servidor de servlets)

138

Ejecutar un servlet

- El cliente se conecta al servlet como URL
 - Directamente tecleado en el navegador
 - En un enlace HTML ó desde un formulario HTML
 - En un objeto URL de java
- El servlet se carga (en el servidor)
 - Se carga la clase del servlet
 - Se crea una instancia
- El servlet se arranca (en el servidor)
 - Se ejecuta uno de sus métodos
- El servlet termina
 - Según la plataforma de servlets y su configuración, el objeto servlet puede permanecer activo por un tiempo (sesión), y la clase sigue cargada
- El cliente lee la salida del servlet, típicamente un string con código HTML para una página web

139

Comunicación entre cliente y servlet

- Streams de entrada y salida
 - La salida del servlet se utiliza para generar páginas web que son enviadas al cliente
 - La salida del cliente hacia el servlet es menos frecuente; para enviar datos al servlet normalmente se utilizan parámetros
- Parámetros HTTP
 - Como parte del URL:
`http://www.mihost.es/miservlet?param1=valor1¶m2=valor2&...`
 - Con un formulario HTML (tag `<form>` + tags `<input>` de diferentes tipos)

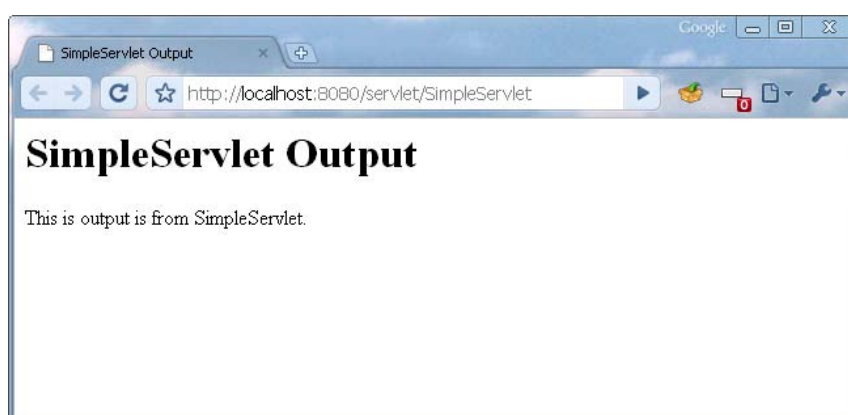
140

Ejemplo: salida de un servlet

```
public class SimpleServlet extends HttpServlet {
    public void doGet (HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException {
        PrintWriter out = response.getWriter ();
        out.println (           // HTML source here
            "<HTML><HEAD><TITLE> "
            + "SimpleServlet Output </TITLE></HEAD>"
            + "<BODY> <h1> SimpleServlet Output </h1>"
            + "This is output is from SimpleServlet."
            + "</BODY></HTML>"); // End HTML source
        out.close ();
    }
}
```

141

Salida del servlet

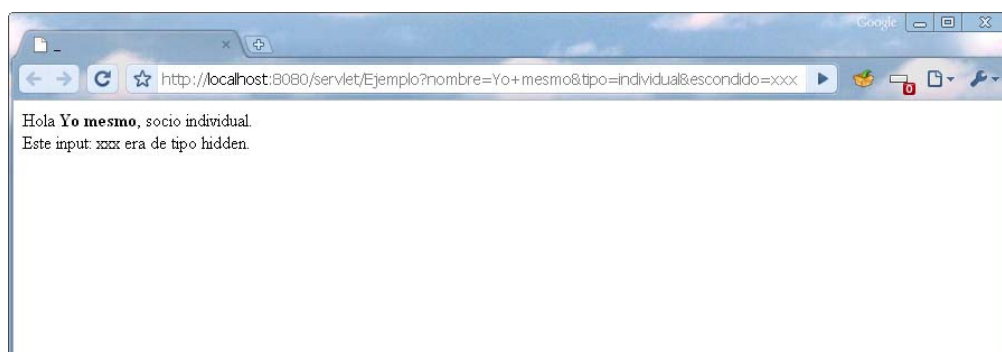


142


```
public class Ejemplo extends HttpServlet {
    public void doGet (HttpServletRequest request,
                       HttpServletResponse response)
        throws ServletException, IOException {
        String nombre = request.getParameter ("nombre");
        String tipo = request.getParameter ("tipo");
        String escondido = request.getParameter ("escondido");
        PrintWriter out = response.getWriter ();
        out.println (
            "<html><body>"
            + "Hola <b>" + nombre + "</b>, socio " + tipo + ".<br>"
            + "Este input: " + escondido + " era de tipo hidden."
            + "</body></html>"
        );
        out.close ();
    }
}
```

145

Salida del servlet



146

JavaServer Pages (JSP)

- Generación dinámica de páginas web
- Permiten mezclar código HTML y código Java
 - `<%= expresión %>`
 - `<% sentencia; %>`
 - Se pueden utilizar variables implícitas: request, response, y otras
- Necesita un servidor web que soporte JSP: p.e. Tomcat
- Se accede igual que a una página html desde un navegador
- El servidor de JSP compila el documento JSP la primera vez que es accedido
 - Genera un servlet en un .java (en el servidor)
 - Compila el servlet, lo carga, y lo ejecuta (en el servidor)
- En realidad la salida no tiene por qué ser HTML: XML, etc.

147

Ejemplo

ejemplo.jsp

```
<html><body>
Hola <b><%= request.getParameter ("nombre") %></b>,
socio <%= request.getParameter ("tipo") %>. <br>
Este input: <%= request.getParameter ("escondido") %>
era de tipo hidden.
</body></html>
```

148

Sentencias de control

ejemplo.jsp

```
<html><body>
Hola <b><%= request.getParameter ("nombre") %></b>,
socio <%= request.getParameter ("tipo") %>. <br>
Este input: <%= request.getParameter ("escondido") %>
era de tipo hidden.<br>
La cuota es de
<% String tipo = request.getParameter ("tipo"); %>
<% if (tipo.equals ("individual")) { %>
    30 euros
<% } else if (tipo.equals ("estudiante")) { %>
    15 euros
<% } else { %>
    300 euros
<% } %>
</body></html>
```

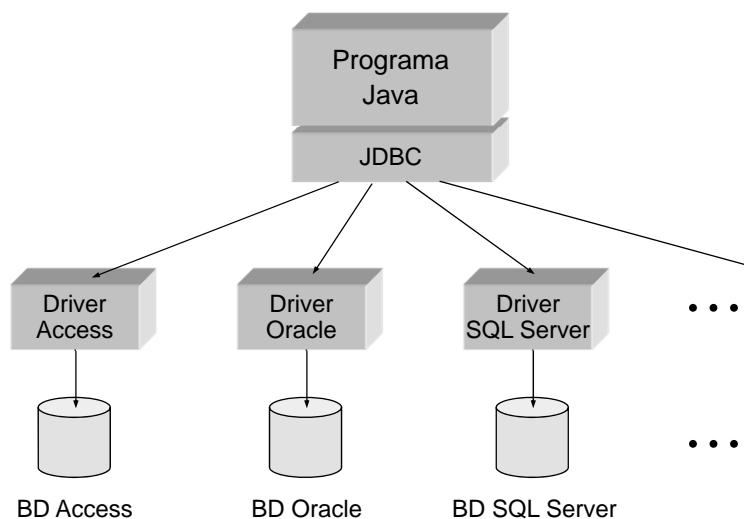
149

Salida del documento JSP



150

Java DataBase Connectivity (JDBC)



151

Ejemplo de programa JDBC

```
import java.sql.*;

class Ejemplo {
    public static void main (String args[]) throws Exception {
        // Iniciar conexión con base de datos
        Class.forName ("ids.sql.IDSDriver");
        String url = "jdbc:ids://localhost:12/conn?dsn='ejemplo'";
        Connection con = DriverManager.getConnection (url);
        Statement stmt = con.createStatement ();
        ...
    }
}
```

152

```

...
stmt.executeUpdate ( // Crear tabla Cuentas
    "CREATE TABLE Cuentas "
    + "(numero INTEGER, saldo INTEGER, dniTitular VARCHAR(20))");
stmt.executeUpdate (// Crear tabla Clientes
    "CREATE TABLE Clientes "
    + "(nombre VARCHAR(40), dni VARCHAR(20))");
// Insertar varios registros en las tablas creadas
stmt.executeUpdate ("INSERT INTO Cuentas (numero, saldo, dniTitular) "
    + "VALUES (123, 1000, '156898')");
stmt.executeUpdate ("INSERT INTO Cuentas (numero, saldo, dniTitular) "
    + "VALUES (456, 500, '230932')");
stmt.executeUpdate ("INSERT INTO Cuentas (numero, saldo, dniTitular) "
    + "VALUES (789, 3000, '156898')");
stmt.executeUpdate ("INSERT INTO Clientes (nombre, dni) "
    + "VALUES ('Juan', '156898')");
stmt.executeUpdate ("INSERT INTO Clientes (nombre, dni) "
    + "VALUES ('Luis', '230932')");
stmt.executeUpdate ("UPDATE Cuentas SET saldo = saldo * 1.2 "
    + "WHERE dniTitular = '156898' AND saldo < 2000");
stmt.executeUpdate ("DELETE FROM Cuentas WHERE saldo < 1000");
...

```

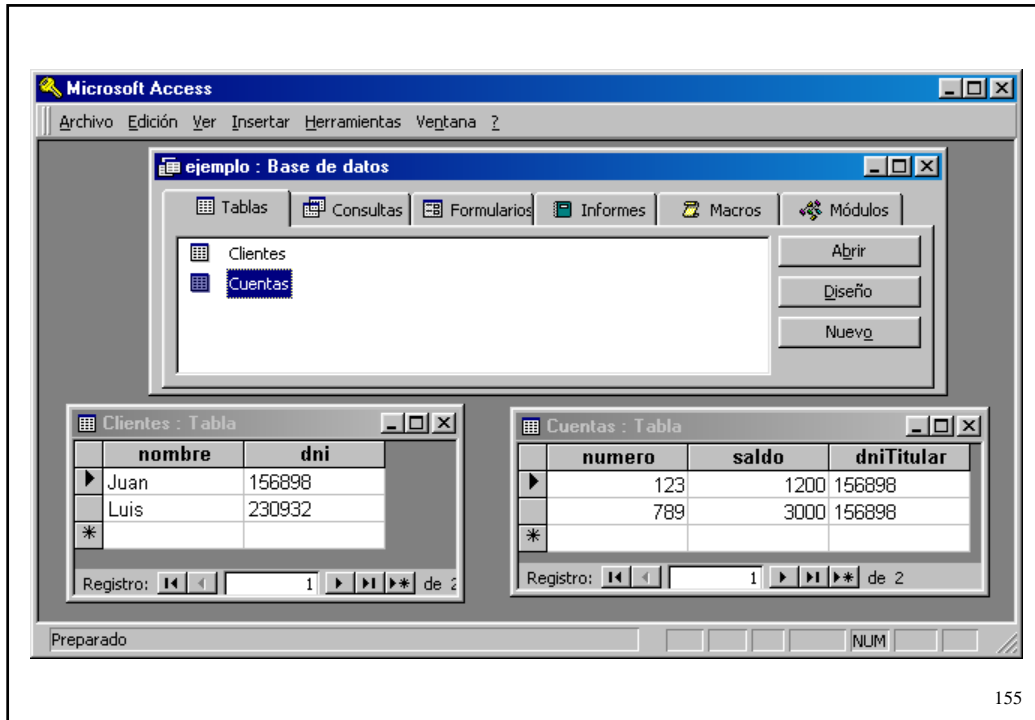
153

```

...
// Realizar una consulta sobre la base de datos
ResultSet rs = stmt.executeQuery (
    "SELECT Clientes.nombre, Clientes.dni, "
    + "Cuentas.numero, Cuentas.saldo "
    + "FROM Cuentas, Clientes "
    + "WHERE Cuentas.dniTitular = Clientes.dni "
    + "AND Cuentas.saldo > 2000");
while (rs.next ()) {
    String nombre = rs.getString ("nombre");
    String dni = rs.getString ("dni");
    int numero = rs.getInt ("numero");
    int saldo = rs.getInt ("saldo");
    System.out.println (nombre + " " + dni + " "
        + numero + " " + saldo); }
} // Fin de main
}

```

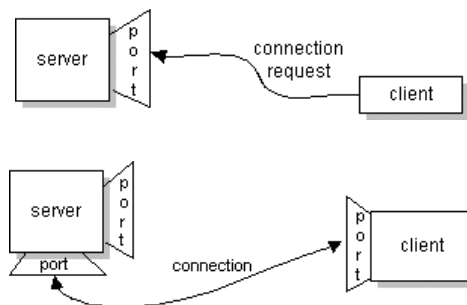
154



155

Sockets

- Conexión cliente / servidor basada en puertos (nivel más bajo que URL)
- Un socket en cada uno de los dos extremos de la comunicación
- Asociado a un número de puerto para identificar a la aplicación



156

java.net.Socket java.net.ServerSocket

- **Socket cliente:** `java.net.Socket`
 - Constructor: `Socket (String host, int port)`
Genera un número de puerto local, y toma nota de la dirección IP del host local
 - `getLocalPort(), getLocalAddress()`
 - Envío y recepción de datos (emiten `IOException`)
 - `getInputStream()`
 - `getOutputStream()`
- **Socket servidor:** `java.net.ServerSocket`
 - Constructor: `ServerSocket (int port)`
 - Esperar datos: `accept() → Socket` (emite `IOException`)
 - El programa servidor queda bloqueado a la espera de que un cliente se conecte
 - Cuando llega una conexión, `accept()` crea y devuelve un socket dirigido hacia el socket del cliente
 - Dirección del socket cliente: `getInetAddress(), getPort()`

157

Ejemplo: echo



Cliente

Servidor

```
> java EchoClient  
Dialogue client/server
```

☞ **hola**

```
(Server) You said: hola
```

☞ **adios**

```
(Server) You said: adios
```

158

Cliente

```
import java.io.*;
import java.net.*;

public class EchoClient {
    public static void main (String[] args)
        throws UnknownHostException, IOException {

        Socket echoSocket = new Socket ("ejemplo.uam.es", 4444);
        PrintWriter salida =
            new PrintWriter (echoSocket.getOutputStream (), true);

        BufferedReader entrada = new BufferedReader (
            new InputStreamReader (echoSocket.getInputStream ());
        ...
    }
}
```

159

```
...
BufferedReader stdIn = new BufferedReader (
    new InputStreamReader (System.in));
String fromServer;

while ((fromServer = entrada.readLine ()) != null) {
    System.out.println (fromServer);
    salida.println (stdIn.readLine ());
}

// primero se cierran los streams y luego los sockets
salida.close ();
entrada.close ();
stdIn.close ();
echoSocket.close ();
}
}
```

160

Servidor

```
import java.net.*;
import java.io.*;

public class EchoServer {
    public static void main (String[] args) throws IOException {
        ServerSocket serverSocket = new ServerSocket (4444);
        Socket clientSocket = serverSocket.accept ();
        ...
    }
}
```

161

```
...
BufferedReader entrada = new BufferedReader (
    new InputStreamReader (clientSocket.getInputStream ()));
PrintWriter salida =
    new PrintWriter (clientSocket.getOutputStream (), true);
String linea = "Dialogue client/server";
salida.println (linea);

while ((linea = entrada.readLine ()) != null)
    salida.println ("(Server) You said: " + linea);

salida.close ();
entrada.close ();
clientSocket.close ();
serverSocket.close ();
}
}
```

162

Servidor para varios clientes simultáneos

```
public class EchoServer {
    public static void main (String[] args) throws IOException {
        ServerSocket serverSocket = new ServerSocket (4444);

        while (true) {
            Socket clientSocket = serverSocket.accept ();
            new EchoServiceThread (clientSocket) .start ();
        }
    }
}
```

163

```
class EchoServiceThread extends Thread {
    Socket clientSocket;
    EchoServiceThread (Socket s) { clientSocket = s; }
    public void run () {
        try {
            BufferedReader entrada = new BufferedReader (
                new InputStreamReader (
                    clientSocket.getInputStream ());
            PrintWriter salida =
                new PrintWriter (
                    clientSocket.getOutputStream (), true);
            String linea = "Dialogue client/server";
            salida.println (linea);
            ...
        }
    }
}
```

164

```
...
while ((linea = entrada.readLine ()) != null)
    salida.println ("(Server) You said: " + linea);

salida.close ();
entrada.close ();
clientSocket.close ();
} // Fin de bloque try
catch (IOException e) {
    System.err.println ("Accept failed.");
}
}
}
```

165