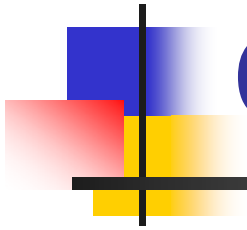
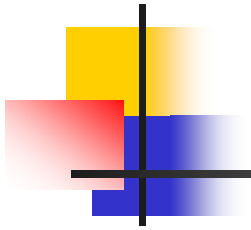


CORBA: Anexos

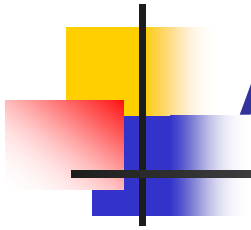
Ingeniería del Software II
Curso 2009/2010

Sergio Ilarri Artigas
silarri@unizar.es



Índice

- Anexo I: Ejemplo *HelloWorld*
- Anexo II: Ejemplo con Objeto *callback*
- Anexo III: Algunos Conceptos Avanzados
- Anexo IV: *CORBA* en Java 1.3
- Anexo V: Ejemplos de Mapping de IDL en Java
- Anexo VI: Resumen de Terminología

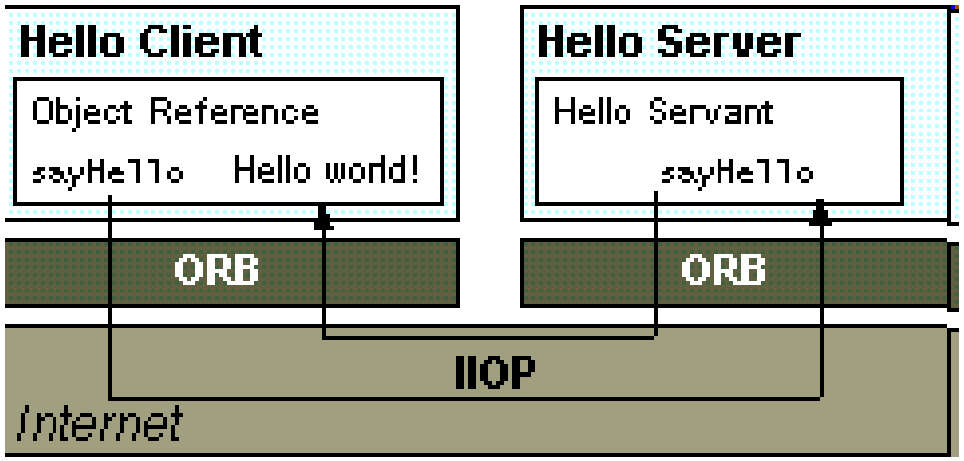


Anexo I

Ejemplo HelloWorld

(en Java 1.4)

Ej.: HelloWorld (I)



Pasos

1.- Definir el *interfaz* remoto
2.- Compilar el *interfaz* remoto
3.- Implementar el servidor
4.- Implementar el cliente
5.- Arrancar la aplicación

http://java.sun.com/j2se/1.5.0/docs/guide/idl/GShome.html

Ej.: HelloWorld (II)

- Paso 1: definir el *interfaz* remoto *IDL*
  - Si el servicio ya estuviera implementado, tendrían que pasarnos el *interfaz IDL* para programar el cliente

Hello.idl

```
module HelloApp
{
   interface Hello
   {
      string sayHello();
      oneway void shutdown();
   };
};
```

En directorio “HelloApp”

Module: espacio de nombres (*package*)

Ej.: HelloWorld (III)

- Paso 2: compilar el *interfaz* remoto
  - El *IDL* permite independencia del lenguaje
  - Podríamos usar un compilador *IDL* de Java para implementar el cliente en Java y un compilador *IDL* de Java para implementar el servidor en C++

idlj -fall Hello.idl

-fall para que cree no sólo los *stubs* del cliente sino también los *skeletons* del servidor

Ej.: HelloWorld (IV)

- *Hello.java*:
  - Versión Java del interfaz IDL
  - Proporciona funcionalidad de objeto CORBA estándar
  - Extiende *org.omg.CORBA.Object, HelloOperations*, y *org.omg.CORBA.portable.IDLEntity*
- *HelloHelper.java*:
  - Funcionalidad requerida para convertir objetos CORBA a sus tipos (*narrow*)
  - Responsable de leer y escribir los tipos de/a flujos de datos
  - También de insertar y extraer valores en un *Any*
- *HelloHolder.java*:
  - Para implementar parámetros out/inout
  - Encapsula un objeto que implementa *Hello*
  - Implementa el interfaz *org.omg.CORBA.portable.Streamable*
  - Delega en *HelloHelper* para leer y escribir
  - Proporciona operaciones para argumentos: *org.omg.CORBA.portable.OutputStream* y *org.omg.CORBA.portable.InputStream*

Downcasting de interfaces remotos

Ficheros obtenidos (I)

Recuerda que en Java, el paso de parámetros es por valor...

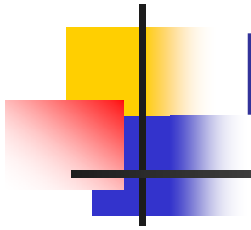
Luego se compilarán con *javac* y se pondrán en el CLASSPATH

Ej.: HelloWorld (V)

Ficheros obtenidos (II)

- *_HelloStub.java* (*stub* del cliente)
  - Objeto local representando al remoto
  - Proporciona la funcionalidad *CORBA* básica al cliente
  - El cliente no lo usa directamente
  - Extiende *org.omg.CORBA.portable.ObjectImpl*
  - Implementa el interfaz *Hello.java*
- *HelloPOA.java* (*skeleton* del servidor)
  - Punto de entrada al objeto distribuido
  - Proporciona la funcionalidad CORBA básica al servidor
  - Extiende *org.omg.PortableServer.Servant*
  - Implementa los interfaces *org.omg.CORBA.portable.InvokeHandler* y *HelloOperations*
- *HelloOperations.java*
  - *Mappings* de las operaciones definidas en el interfaz *IDL*
  - Compartido por *stubs* y *skeletons*

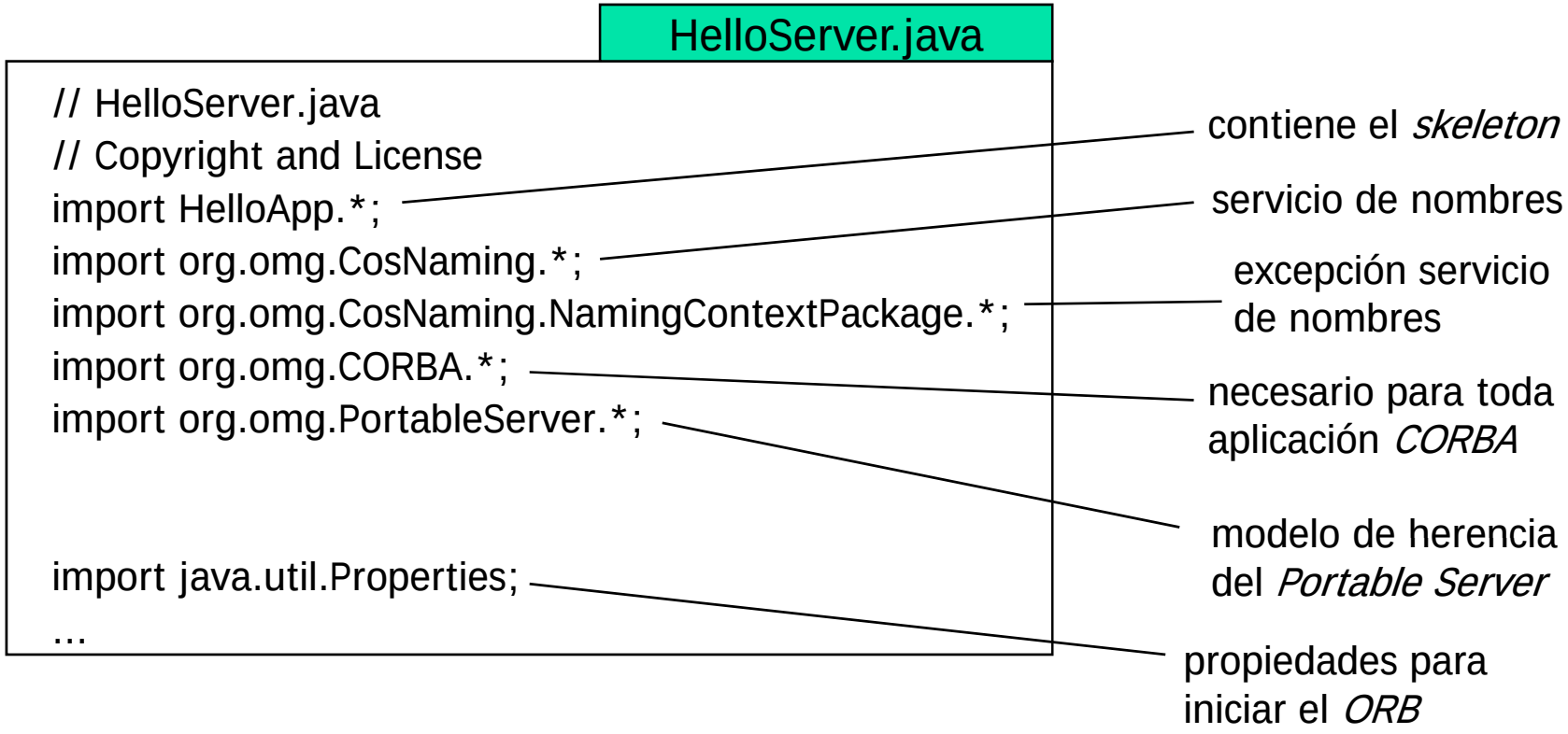
Luego se compilarán con *javac* y se pondrán en el CLASSPATH



Ej.: HelloWorld (VI): Serv.

- Paso 3: implementación del lado servidor
  - Implementar la clase *servant*
    - Extiende *HelloPOA* (*skeleton*)
    - Un método por cada operación del *IDL*
    - Son métodos Java ordinarios (el código necesario para operar con el *ORB* lo proporciona el *skeleton*)
  - Implementar la clase *servidor*:
    - Pone los servicios de los objetos a disposición de los interesados

Ej.: HelloWorld (VII): Serv.



Ej.: HelloWorld (VIII): Serv.

HelloServer.java: clase *servant*

```
class HelloImpl extends HelloPOA {
  private ORB orb;

  public void setORB(ORB orb_val) {
     orb = orb_val;
  }

  // implement sayHello() method
  public String sayHello() {
     return "\nHello world !!\n";
  }

  // implement shutdown() method
  public void shutdown() {
     orb.shutdown(false);
  }
}
```

definido por nosotros para poder guardar una referencia al *ORB* (que se utiliza en *shutdown*)

Método de *org.omg.CORBA.ORB*. Con argumento *false*, solicita que se termine el *ORB* inmediatamente, sin esperar que termine ningún procesamiento

Ej.: HelloWorld (IX): Serv.

HelloServer.java: clase servidora

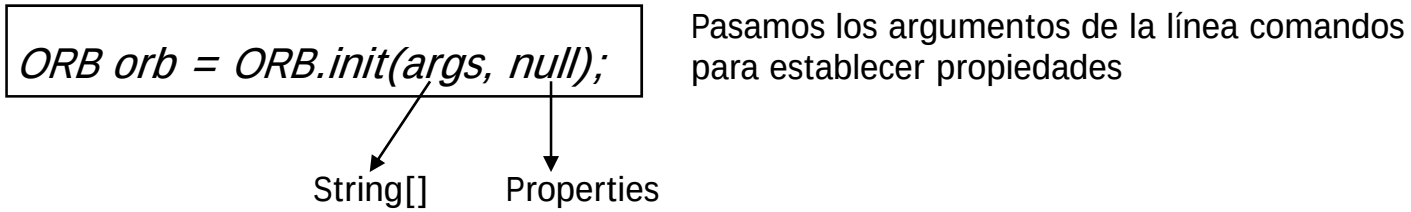
```
public class HelloServer {

   public static void main(String args[]) {
      try {
         /* Main code here */
      }
      catch (Exception e) {
         System.err.println("ERROR: " + e);
         e.printStackTrace(System.out);
      }

      System.out.println("HelloServer Exiting ...");
   }
}
```

Desarrollado en las siguientes transparencias

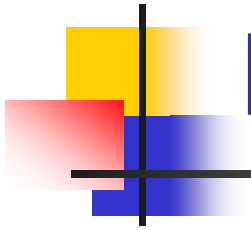
captura, en particular, excepciones *CORBA* que puedan producirse

Ej.: HelloWorld (X): Serv.

1) Inicializar el ORB (método factoría):



Por ejemplo, una propiedad estandarizada es omg.ORB.CORBA.ORBClass, que permite establecer la implementación de ORB deseada



Reflexión

¿Algún patrón de diseño relacionado con esto?

El patrón ***Factory***

¿Ventajas?

Permite “enchufar” cualquier *ORB* de *CORBA*

Ej.: HelloWorld (XI): Serv.

2) Obtener una referencia al *Root POA*:

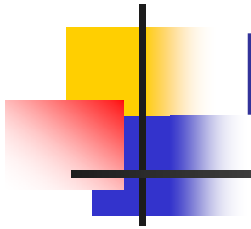
```
POA rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
```

Permite obtener referencias Iniciales a determinados objetos: RootPOA, NameService, etc.

3) Activar el *POAManager*:

```
rootpoa.the_POAManager().activate();
```

Hace que los POAs asociados comiencen a procesar peticiones



Ej.: HelloWorld (XII): Serv.

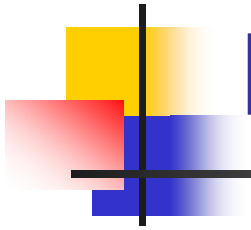
4) Instanciar el objeto *servant*:

```
HelloImpl helloImpl = new HelloImpl();
```

5) Pasar una referencia al *ORB*:

```
helloImpl.setORB(orb);
```

Esto lo requiere este ejemplo completo para implementar el *shutdown*; también podría ocuparse el servidor y el objeto remoto simplemente activar un *flag* para avisarle de la necesidad



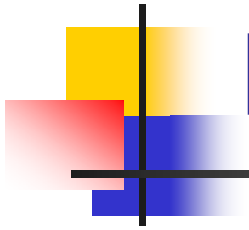
Ej.: HelloWorld (XIII): Serv.

6) Obtener una referencia al *servant*:

```
org.omg.CORBA.Object ref = rootpoa.servant_to_reference(helloImpl);
Hello href = HelloHelper.narrow(ref)
```

7) Obtener el contexto de nombres inicial:

```
org.omg.CORBA.Object objRef =
   orb.resolve_initial_references("NameService");

NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
```

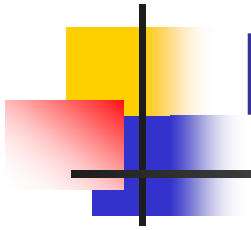


Ej.: HelloWorld (XIV): Serv.

8) Registrar el *servant* en el servicio de nombres:

```
String name = "Hello";
NameComponent path[] = ncRef.to_name(name);
ncRef.rebind(path, href);
```

Cuando el cliente llame a *resolve("Hello")* en el contexto de nombres inicial, el servicio de nombres devolverá una referencia de objeto al *servant Hello*



Ej.: HelloWorld (XV): Serv.

9) Esperar invocaciones en un bucle:

```
orb.run();
```

-Permite al *ORB* realizar trabajo utilizando el thread principal del servidor

-Cuando llegue una invocación del cliente y se complete, el servidor se pondrá a esperar de nuevo

-El método *run* termina la ejecución cuando se haga un *shutdown* sobre el *ORB*

Ej.: HelloWorld (XVI): Cliente

- Paso 4: implementar el cliente

HelloClient.java

```
// HelloClient.java
// Copyright and License
import HelloApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
...
```

contiene el *stub*

servicio de nombres

excepción servicio de nombres

necesario para toda aplicación *CORBA*

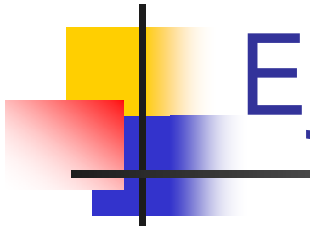
Ej.: HelloWorld (XVII): Cliente

HelloClient.java: clase cliente

```
public class HelloClient {

   public static void main(String args[]) {
      try {
         /* Main code here */
      }
      catch (Exception e) {
         System.err.println("ERROR: " + e);
         e.printStackTrace(System.out);
      }
   }
}
```

Desarrollado en las siguientes transparencias

captura, en particular, excepciones *CORBA* que puedan producirse



Ej.: HelloWorld (XVIII): Cliente

1) Inicializar el *ORB*:

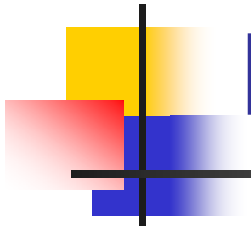
```
ORB orb = ORB.init(args, null);
```

Pasamos los argumentos de la línea comandos para establecer propiedades

2) Obtener el contexto de nombres inicial:

```
org.omg.CORBA.Object objRef =
   orb.resolve_initial_references("NameService");

NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
```

Hasta ahora... nada nuevo por parte del cliente



Ej.: HelloWorld (XIX): Cliente

3) Obtener una referencia al *servant*:

```
String name = "Hello";
helloImpl = HelloHelper.narrow(ncRef.resolve-str(name));
System.out.println("Obtained a handle on server object: " + helloImpl);
```

4) Invocar la operación *sayHello* y terminar el *ORB*:

```
System.out.println(helloImpl.sayHello());
helloImpl.shutdown();
```

Ej.: HelloWorld (XX): Ejec.

Primero, arrancar el *ORB*:

```
orbd -ORBInitialPort 1050 -ORBInitialHost localhost
```

Demonio del servicio de nombres

< 1024 requiere privilegios de administrador

Cliente y servidor en la misma máquina (opción obligatoria)

Después, arrancar el servidor *Hello*:

```
java HelloServer -ORBInitialPort 1050 -ORBInitialHost localhost
```

Máquina donde reside el servidor de nombres

Ej.: HelloWorld (XXI): Ejec.

Finalmente, ejecutar la aplicación cliente:

java HelloClient -ORBInitialPort 1050 -ORBInitialHost localhost

Máquina donde reside el servidor de nombres

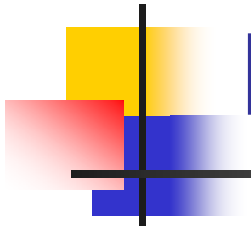
Salida

Obtained a handle on server object:

IOR:000000000000001749444c3a48656c6c6f4170702f48656c6c6f3a312e30000
0000000010000000000000008a000102000000000f3135352e3231302e3135322e
353700000d2d000000000031afabcb0000000002076335af80000000100000000000
000001000000008526f6f74504f41000000000800000001000000001400000000000
000200000001000000200000000000010001000000020501000100010020000010
109000000010001010000000026000000020002

Hello world !!

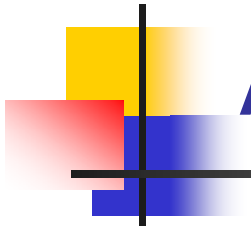
El servidor de nombres *orbd* queda vivo hasta que se mata (p.ej., con CTRL-C)



Reflexión

¿Y si no hay servicio de nombres disponible?

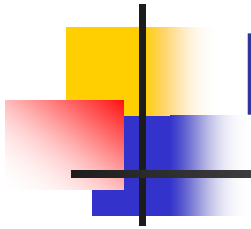
orb.object_to_string(remoteObj)
orb.string_to_object(stringifiedReference)



Anexo II

Ejemplo con Objeto Callback

(en Java 1.4)



Ej.: Objeto *Callback* (I)

- Si el cliente tiene que reaccionar ante cambios en el servidor, dos opciones:
  1) *Polling/pull model*: preguntarle periódicamente
  2) *Callback/Push model*: pasarle un objeto *callback* y que él nos avise

- En el siguiente ejemplo:
  - Podemos tener cualquier número de clientes
  - Los mensajes que escribamos se envían a todos ellos
  - La distinción entre cliente y servidor no está clara: el "cliente" también atiende peticiones remotas del "servidor"

http://java.sun.com/j2se/1.5.0/docs/guide/idl/jidlExample3.html

Ej.: Objeto *Callback* (II)

callback.idl

```
interface Listener {
   void message(in string msg);
};

interface MessageServer {
   void register(in Listener lt);
};
```

En directorio “CallbackSample”

MessageServerImpl.java

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.Vector;
import java.util.Iterator;

public class MessageServerImpl extends MessageServerPOA {
   ...
}
```

Ej.: Objeto *Callback* (III)

```
    private Vector clients = new Vector();
    private ReadThread rt = null;

    public MessageServerImpl() {rt = new ReadThread(this);}

    public void register(Listener lt) {clients.add(lt);}

    public void startReadThread() {rt.start();}

    public void message(String msg) {
       Iterator it = clients.iterator();
       while (it.hasNext()) {
          Listener lt = (Listener) it.next();
          lt.message(msg);
       }
    }
}
```

MessageServerImpl.java

Ej.: Objeto *Callback* (IV)

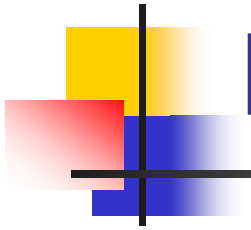
```
class ReadThread extends Thread {
   MessageServerImpl msImpl = null;

   public ReadThread(MessageServerImpl msImpl) {this.msImpl = msImpl;}

   public void run() {
      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

      try {
         for (;;) {
            System.out.print("message > ");
            String msg = br.readLine();
            msImpl.message(msg);
         }
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

MessageServerImpl.java

Ej.: Objeto *Callback* (V)

Server.java

```
import java.util.Properties;
import org.omg.CORBA.ORB;
import org.omg.PortableServer.POA;
import org.omg.PortableServer.POAHelper;
import org.omg.CosNaming.NameComponent;
import org.omg.CosNaming.NamingContext;
import org.omg.CosNaming.NamingContextHelper;

public class Server {

   public static void main(String[] args) {
      try {
         ...
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```



Ej.: Objeto *Callback* (VI)

```
Properties props = System.getProperties();
props.put("org.omg.CORBA.ORBInitialPort", "1050");
props.put("org.omg.CORBA.ORBInitialHost", "<MyHost>");
ORB orb = ORB.init(args, props);
System.out.println("Initialized ORB");

POA rootPOA = POAHelper.narrow(
orb.resolve_initial_references("RootPOA"));
MessageServerImpl msImpl = new MessageServerImpl();
rootPOA.activate_object(msImpl);
MessageServer msRef = MessageServerHelper.narrow(
   rootPOA.servant_to_reference(msImpl));
...
```

Server.java, main (1 de 2)

Ej.: Objeto *Callback* (VII)

```
...

org.omg.CORBA.Object objRef =
   orb.resolve_initial_references("NameService");
NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
System.out.println("Resolved NameService");
String name = "MessageServer";
NameComponent path[] = ncRef.to_name(name);
ncRef.rebind(path, msRef);

rootPOA.the_POAManager().activate();

System.out.println("Server ready and running ....");

msImpl.startReadThread();

orb.run();
```

Server.java, main (2 de 2)

Ej.: Objeto *Callback* (VIII)

Client.java

```
import java.util.Properties;
import org.omg.CORBA.ORB;
import org.omg.PortableServer.POA;
import org.omg.PortableServer.POAHelper;
import org.omg.CosNaming.NameComponent;
import org.omg.CosNaming.NamingContext;
import org.omg.CosNaming.NamingContextHelper;

public class Server {
   public static void main(String[] args) {
      try {
         ...
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

Ej.: Objeto *Callback* (IX)

```
Properties props = System.getProperties();
props.put("org.omg.CORBA.ORBInitialPort", "1050");
props.put("org.omg.CORBA.ORBInitialHost", "<MyHost>");
ORB orb = ORB.init(args, props);
System.out.println("Initialized ORB");

POA rootPOA = POAHelper.narrow(
   orb.resolve_initial_references("RootPOA"));
ListenerImpl listener  = new ListenerImpl();
rootPOA.activate_object(listener);
Listener ref = ListenerHelper.narrow(
   rootPOA.servant_to_reference(listener));
org.omg.CORBA.Object objRef =
   orb.resolve_initial_references("NameService");
NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
System.out.println("Resolved NameService");

...
```

Client.java, main (1 de 2)

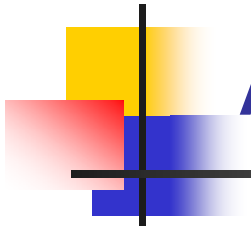
Ej.: Objeto *Callback* (X)

```
...
String name = "MessageServer";
MessageServer msgServer = MessageServerHelper.narrow(ncRef.resolve_str(name));
msgServer.register(ref);
System.out.println("Listener registered with MessageServer");

rootPOA.the_POAManager().activate();

System.out.println("Wait for incoming messages");
orb.run();
```

Client.java, main (2 de 2)

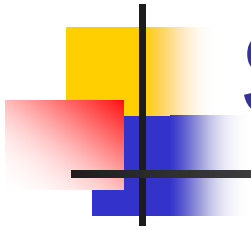
ListenerImpl.java

```
public class ListenerImpl extends ListenerPOA {
   public void message(String msg) {
      System.out.println("Message from server : " + msg);
   }
}
```



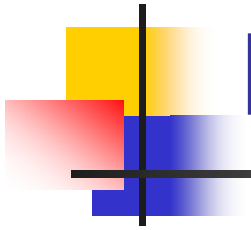
Anexo III

Algunos Conceptos Avanzados



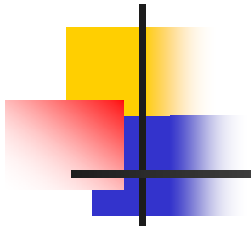
Servicio de Eventos

- Suministradores: disparan eventos
- Consumidores: procesan eventos
- Canal de eventos
- Modelos de eventos:
  - *Tipos:*
    - *push*
    - *pull*
  - Comunicaciones asíncronas
  - Múltiples productores y consumidores



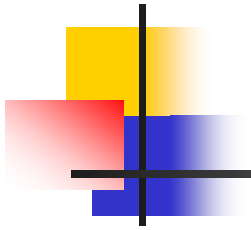
Facilidades Comunes *CORBA*

- *Corba Facilities (horizontal facilities)*
- Interfaces útiles para muchas aplicaciones
- Pero no tan fundamentales como los distintos servicios *CORBA*
- Ejemplos: internacionalización, tiempo, etc.



Interfaces del Dominio

- Interfaces útiles para dominios de aplicación concretos
- Ejemplos:
  - Salud
  - Telecomunicaciones
  - Finanzas
  - Negocios

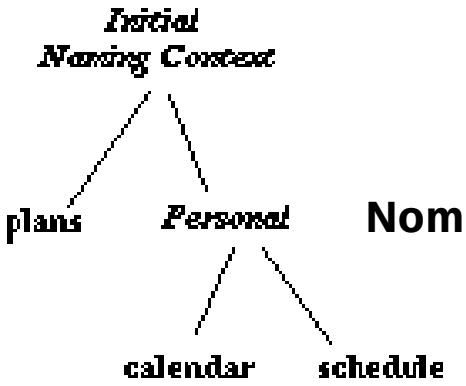


Interfaces de Aplicación

- Los desarrollados para una aplicación concreta
- ¿Están estandarizados?
  - Evidentemente, no. ¡OMG no puede adivinar lo que vamos a necesitar en nuestras aplicaciones!
  - Pero si se usan mucho en distintas aplicaciones, puede que se estandaricen en el futuro

Sobre el Servicio de Nombres (I)

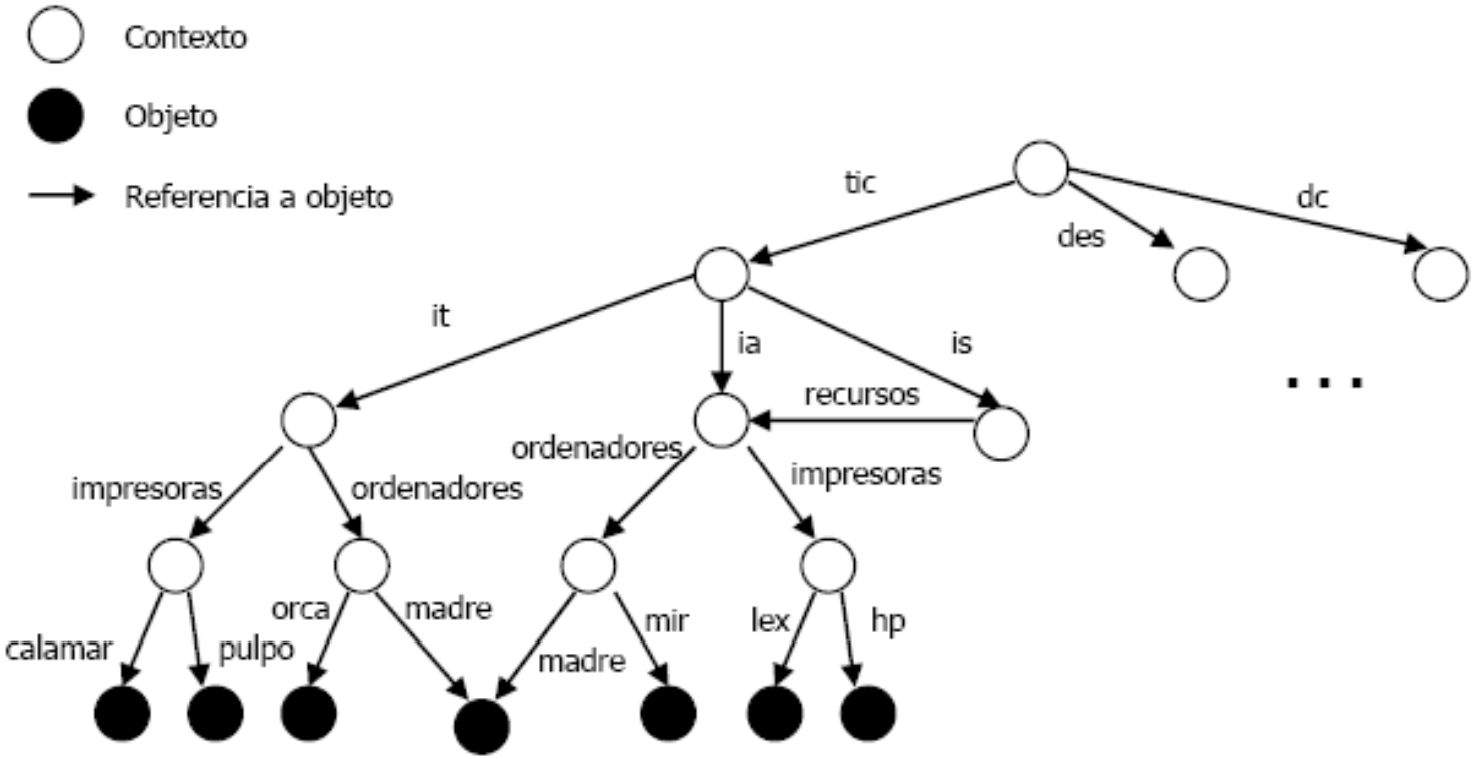
- Un contexto es como cualquier otro objeto. Por tanto, su nombre también puede ligarse a otro contexto



Grafo de nombres
(los nodos son contextos)

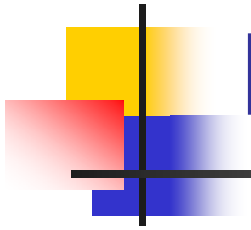
Nombres compuestos (jerárquicos) permiten recorrer el grafo

Sobre el Servicio de Nombres (II)



bind("/tic/it/impresoras/calamar", calamar);

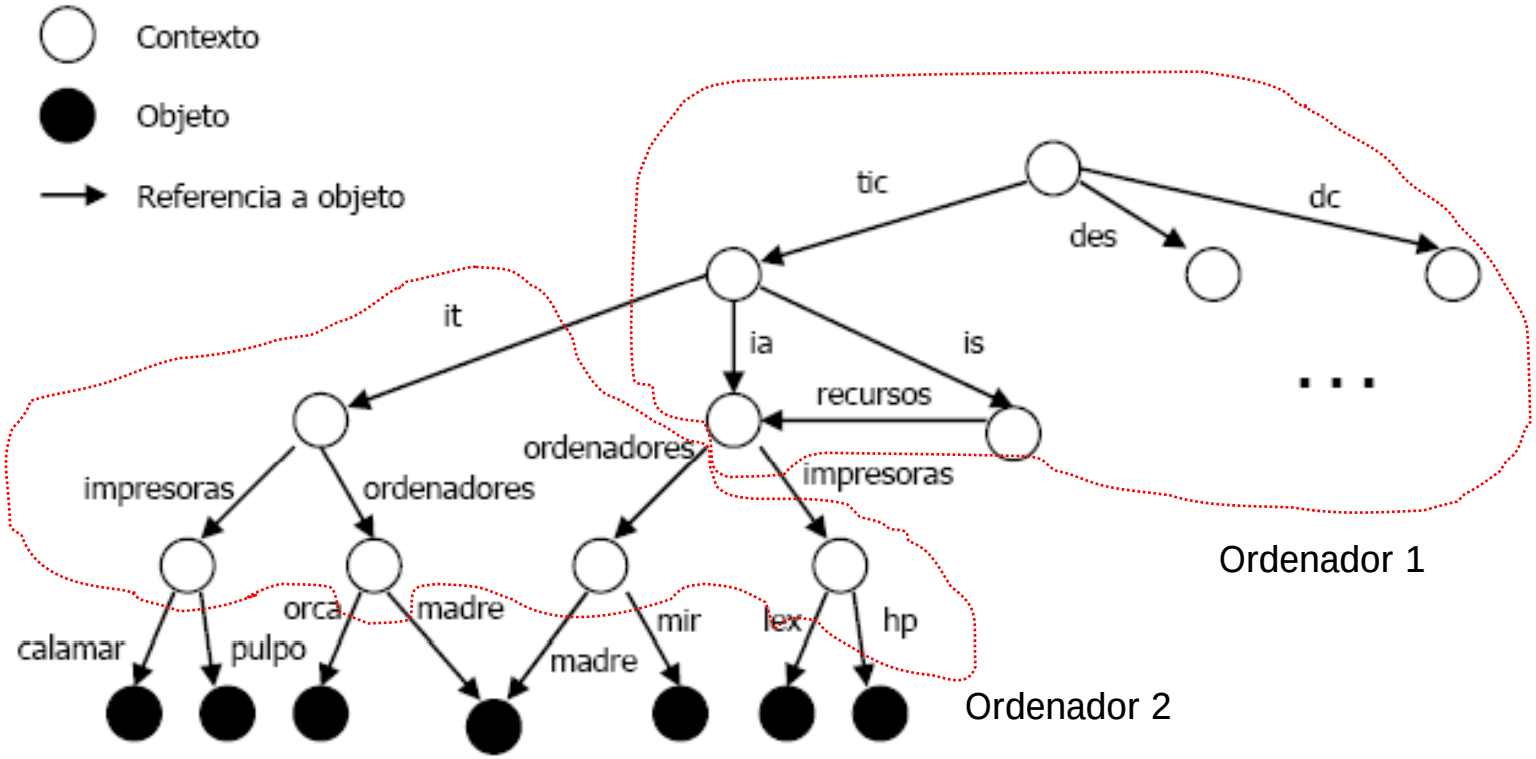
Calamar calamar = resolve("/tic/it/impresoras/calamar");



Reflexión

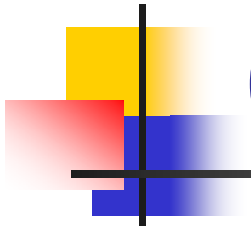
¿Podría distribuirse fácilmente el servicio nombres?

Reflexión



Sí. Afortunadamente, un contexto es un objeto CORBA, de modo que el espacio de nombres puede distribuir fácilmente...

Sobre el Servicio de Nombres (III)



- Interfaz *NamingContext* (factoría de contextos):
  - Creación de contextos: *new_context*, *bind_new_context*
  - Destrucción de contextos: *destroy*, *unbind*
  - Búsqueda: *resolve*
  - Registro: *bind*, *rebind*
  - Deregistro: *unbind*

- Toda aplicación se configura con un contexto inicial

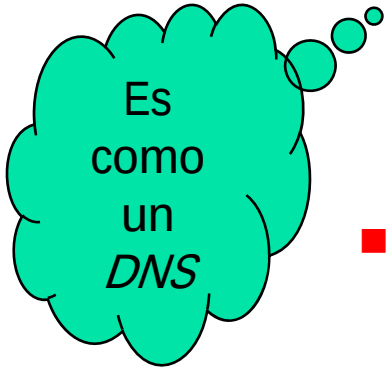
```
ORB orb = ORB.init(args, null);
org.omg.CORBA.Object initialNamingContextObject =
        orb.resolve_initial_references("NameService");
```

Código Java

Configurable con *–ORBInitRef NameService=<ior>*

Sobre el Servicio de Nombres (IV)

- Para que una aplicación pueda usar el servicio de nombres:
  - Su *ORB* debe conocer el puerto de una máquina que ejecuta un servicio de nombres
  - O bien tener un contexto de nombres inicial en forma de cadena de caracteres (*stringified initial naming context*)



Veamos un par de ejemplos de ficheros generados por el compilador de *IDL*: *Hello.java* y *HelloOperations.java*

Ej.: HelloWorld

Hello.java

```
//Hello.java
package HelloApp;


/**
* HelloApp/Hello.java
* Generated by the IDL-to-Java
* compiler (portable), version "3.0"
* from Hello.idl
*/

public interface Hello extends
      HelloOperations,
      org.omg.CORBA.Object,
org.omg.CORBA.portable.IDLEntity
{
} // interface Hello
```

Sentencia IDL	Sentencia Java
module HelloApp	package HelloApp;
interface Hello	public interface Hello

Todo objeto CORBA, debe extender ***org.omg.CORBA.Object*** para disponer de la funcionalidad de CORBA

Signature interface

Ej.: HelloWorld

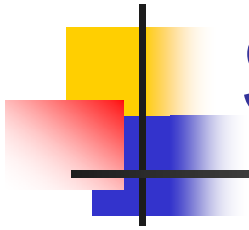
HelloOperations.java

```
//HelloOperations.java
package HelloApp;

/**
* HelloApp/HelloOperations.java
* Generated by the IDL-to-Java
* compiler (portable), version "3.0"
* from Hello.idl
*/

public interface HelloOperations
{
   String sayHello ();
   void Shutdown ();
} // interface HelloOperations
```

Sentencia IDL	Sentencia Java
string sayHello();	String sayHello();
oneway void shutdown();	void Shutdown ();

Interfaz usada del lado del servidor utilizado para optimizar llamadas cuando cliente y servidor están en la misma máquina

Operations interface



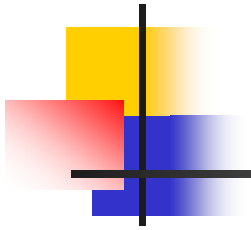
Servicios de Nombres

- Dos opciones de servicios de nombres en Java 1.4:
  - ***orbd:*** incluye un *Transient Naming Service* y un *Persistent Naming Service*, además de un *Server Manager*.
  - ***tnameserv***: un *Transient Naming Service*.

Object Request Broker Deamon

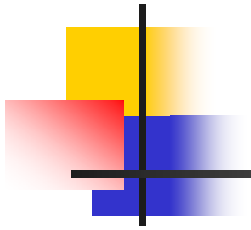
La cadena “NameService” está definida para todos los ORBs de CORBA:
El servicio de nombres será persistente si se usa *orbd* y transitorio si se usa *tnameserv*
La cadena propietaria “TNameService” para usar el servicio de nombres transitorio de *orbd*

Persistente implica que se restaura automáticamente el grafo de nombres al re-arrancar el *orbd*



Invocación Dinámica (I)

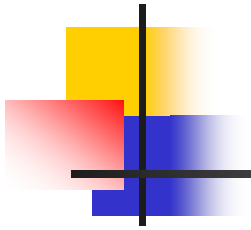
- Flexible, permite al cliente invocar objetos que descubre en ejecución (sin tener *stubs*)
- No comprueba tipos en compilación
- Es posible obtener información acerca de las operaciones disponibles y los tipos de los parámetros en tiempo de ejecución, consultando el *Interface Repository*



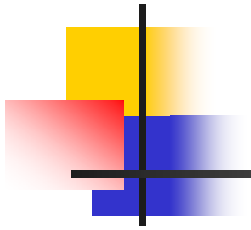
Invocación Dinámica (II)

- Utiliza un objeto *Request*, que almacena lo necesario para la invocación:
  - Referencia al objeto
  - Nombre del método
  - Parámetros
  - Espacio para el resultado
- Clave: capacidad del *Request* para mantener datos autodescriptivos:
  - Tipo *Any* en OMG IDL (*typecode*+ *value*)

Invocación Dinámica (III)



- *org.omg.CORBA.Object*
  - *Object _get_interface_def()*
  - *Request _create_request(...), Request _request(String operation)*
- *org.omg.CORBA.Request*
  - *Any add_inout_arg(), Any add_in_arg(), Any add_out_arg(),...*
  - *void invoke()*
  - *void send_oneway()*
  - *void send_deferred(), boolean poll_response(), void get_response()*
- *org.omg.CORBA.NVList* ——— *org.omg.CORBA.NamedValue*
  - *NamedValue add_item(String item_name, int flags)*
  - *NamedValue add_value(String item_name, Any val, int flags)*
  - *int count()*



Invocación Dinámica (IV)

- *org.omg.CORBA.ORB*
  - NVList create_list(int count)
  - NVList create_operation_list(Object oper)
  - void send_multiple_requests_oneway(Request[] req)
  - void send_multiple_requests_deferred(Request[] req),
    boolean poll_next_response(),
    Request get_next_response()

Invocación Dinámica (V)

Lado del cliente

- *Dynamic Invocation Interface* (*DII*)
  - Permite realizar invocaciones remotas sin tener los *stubs*

org.omg.CORBA.Request

Lado del servidor

- *Dynamic Skeleton Interface* (*DSI*)
  - Permite implementar operaciones remotas sin conocimiento estático de los *skeletons*

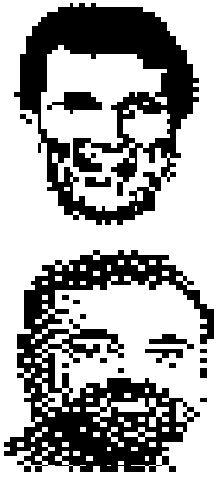
método genérico *invoke*

org.omg.CORBA.ServerRequest

org.omg.PortableServer.DynamicImplementation

IORs (I)

IOR:000000000000000d49444c3a54696d653a312e3000000000000000001000000
00000000f000010100000000066d6572676500060b000000d7030231310c000016
7e0000175d360aed118143582d466163653a20457348795e426e5851664e527333
3d4d7268787b72643b4b4c4e59295a526a4c3a39564628296e4345633637533d6a
2c77245879727c7b6371752b7434567d61383b3422535e514a2b48322e772f354f
245e573e69512b6b24717a412f7822265c2172772d577d303927537d5e715c5757
70784a2734385832694f3e7433483753276f4825305a2858382e4a30667577487b
3647343e3e7e5b554b21643d67613c6d367a4e784d414f7a7658606d214a45677e



272f737756642420000000000000..11....~...]6....CX-Face: EsHy^BnXQfNRs3=Mrhx{rd;KLNY)ZRjL:9VF()n
CEc67S=j,w$Xyr|{cqu+t4V}a8;4"S^QJ+H2.w/5O$^W>iQ+k$qzA/x"&\!rw-
W}09'S}^q\WWpxJ'48X2iO>t3H7S'oH%0Z(X8.J0fuwH{6G4>>~[UK!d=ga<m6zNxMAOzvX`m!JEg~'/swVd$.

El formato de los *IORs* no es muy legible...

http://www.triodia.com/staff/michi/advanced_corba/easter_result.html

IORs (II)

Más legibles que los IOR

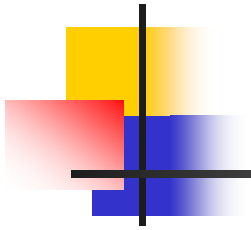
- Formatos tipo *URL* definidos en *CORBA*:
  - *corbaloc*
    - Ej.: corbaloc::hendrix:5000/NameService
    - Ej.: corbaloc::155.210.155.63:5000/Test
    - Ej.: corbaloc::rdp10.cps.unizar.es:5000/Test
  - *corbaname*
    - Similar a corbaloc, pero concatenando # y un nombre
    - Ej.: corbaname::hendrix:5000/NameService#es/test

Referencia a un contexto

Nombre que puede Resolverse en ese contexto

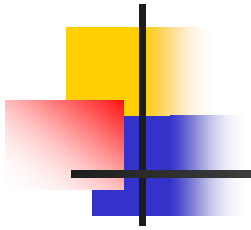
extensión del *CosNaming service*

Interoperable Naming Service (INS)



Interoperabilidad: *IIOP*

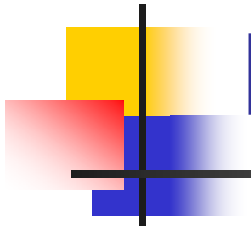
- *IIOP* permite interoperar también con:
  - *RMI sobre IIOP*
  - *EJBs*
  - Diversos servidores de aplicaciones



Interoperabilidad: *ESIOPs*

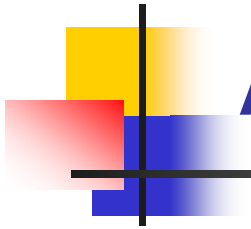
- *Environment-Specific Inter-ORB Protocols*
- No se basan en *GIOP*, dado que hay una estructura subyacente que quieren aprovechar
- Ej.: *DCE CIOP*

  (*DCE Common Inter-ORB Protocol*)



Más Interoperabilidad...

- También está estandarizada la interoperabilidad *COM/DCOM*

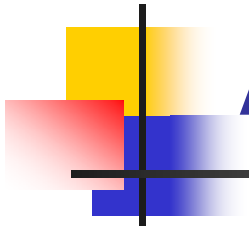


Adaptadores de Objetos (I)

- *Object Adapter (OA)*

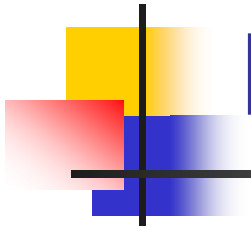
- Mantiene un registro de objetos activos y sus implementaciones

- Enruta una petición usando una referencia a un objeto remoto con el código que la implementa



Adaptadores de Objetos (II)

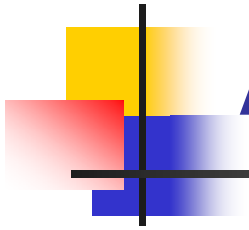
- Define cómo se activan los objetos
  - Creando un nuevo proceso
  - Creando un nuevo thread
  - Reutilizando un proceso/thread



Reflexión

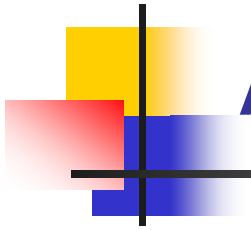
¿Algún patrón de diseño que os venga a la mente con todo esto?

El patrón ***Adapter***



Adaptador Básico de Objetos

- *Basic Object Adapter* (*BOA*)
- Estandarizado por *CORBA* hasta *CORBA* 2.1 (inclusive)
  - Especificación ambigua y poco potente
  - Los fabricantes añadían sus extensiones
    - Problemas de portabilidad



Adaptador Portable (I)

- ***Portable*** *Object Adapter* (*POA*)
- Definido desde *CORBA* 2.2 (inclusive)
- Potente, pero complejo
- El *BOA* ya no es parte de *CORBA*

Adaptador Portable (II)

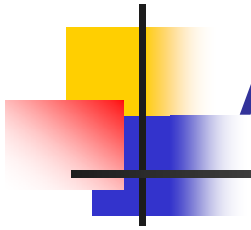
- Crear un nuevo *POA* permite al desarrollador:
  - Proporcionar un *adapter activator* y *servant manager* distinto
  - Partir el espacio de nombres de los objetos
  - Controlar independiente el procesamiento de peticiones para distintos conjuntos de objetos

crea POAs hijos si es necesario

crea servants bajo demanda

- Objeto *POAManager*:
  - Controla el estado de procesamiento de los *POAs* asociados
    - Por ejemplo, si las peticiones al *POA* se descartan o encolan. También puede desactivar el *POA*.

Válvula que regula el flujo de peticiones



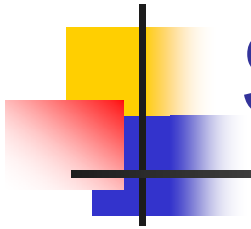
Adaptador Portable (III)

- Al menos, siempre habrá un ***RootPOA***
  - Utiliza una política *multithreading*
  - *CORBA* no estandariza el modelo *multithread*:
    - Un *thread* por petición
    - Un *thread* por conexión
    - *Pool* de *threads*
    - ...
  - Normalmente, las implementaciones de *CORBA* permiten especificar un modelo concreto

Adapter Activator y *Servant Manager*

- Crear un nuevo *POA* permite al desarrollador:
  - Proporcionar un específico:
    - ***adapter activator***: permite crear *POAs* hijos bajo demanda (con las políticas requeridas), al recibir una petición que nombra a dicho *POA* hijo, no necesario si todos los *POAs* se crean al ejecutar la aplicación
    - ***servant manager***: permite al *POA* activar (y desactivar) *servants* bajo demanda, cuando se recibe una petición para un objeto inactivo; no necesario si el servidor activa todos los objetos al arrancar
  - Partir el espacio de nombres de los objetos
  - Controlar independiente el procesamiento de peticiones para distintos conjuntos de objetos

Una petición referencia a un POA y a un identificador de objeto

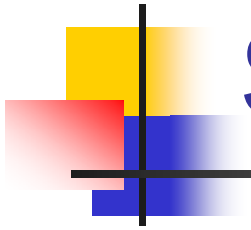


Semántica de Llamadas (I)

El modelo de objetos define 2 tipos de semánticas para las operaciones:

1) At-most-once:

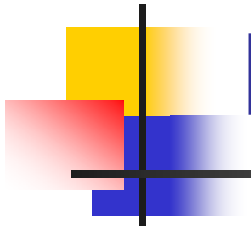
- Si la llamada termina con éxito, se realizó exactamente una vez
- Si devuelve excepción, se realizó como mucho una vez



Semántica de Llamadas (II)

2) Best-effort:

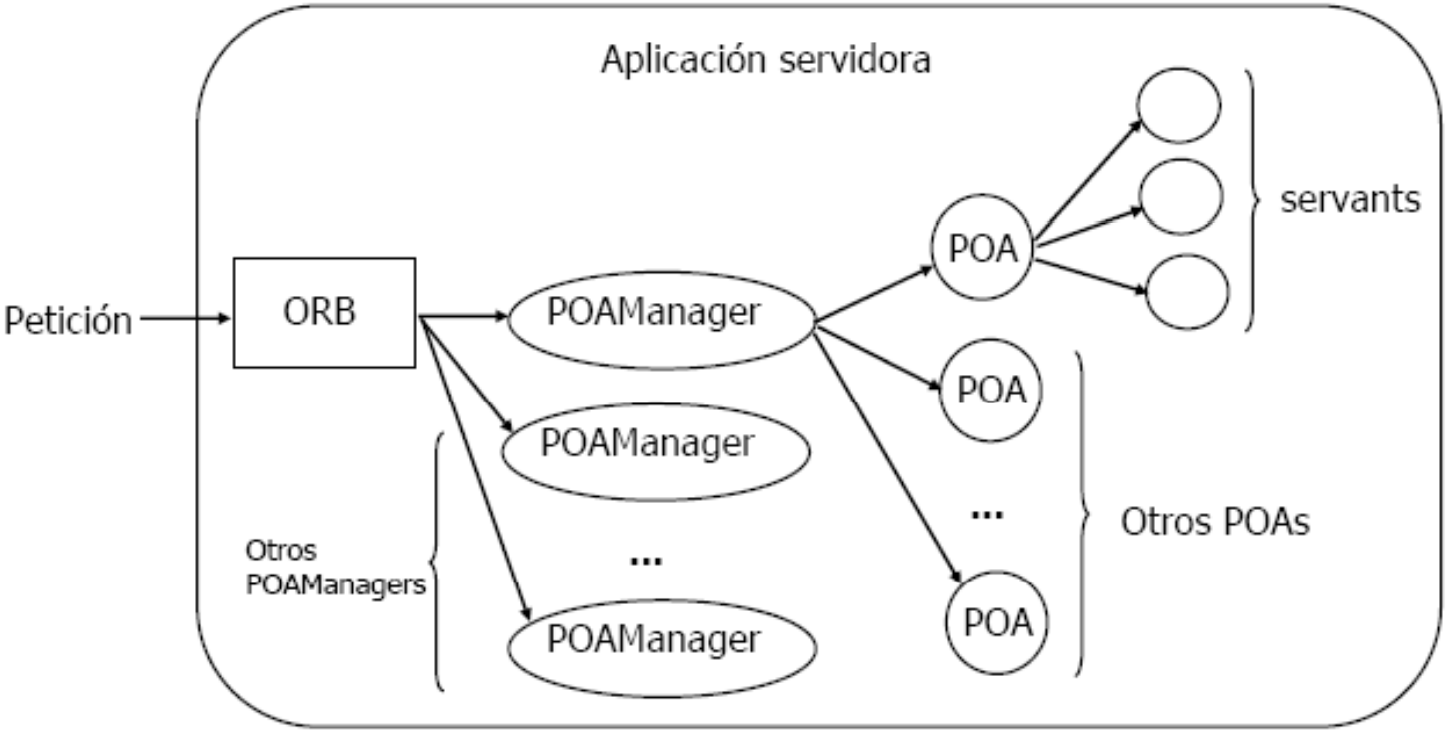
- La operación no devuelve ningún resultado
- El invocador no queda esperando a que se termine la operación
- No hay garantía de que la operación se complete

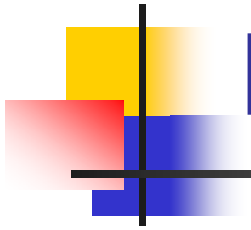


Modelos de servidor

- Single-thread
- Multithread por cliente
- Multithread por petición
- Pool de threads

Adaptador Portable





Pseudo-Objetos

- Son objetos que los crea directamente el ORB pero que pueden invocarse como si fueran un objeto normal
- Son objetos proporcionados por el ORB (no necesariamente remotos)
- Se especifican en IDL pero son específicos del lenguaje
- Ejemplos: BOA, POA, Object (de CORBA), Request (de CORBA), el propio ORB

Modelos de Implementación de un *Servant*

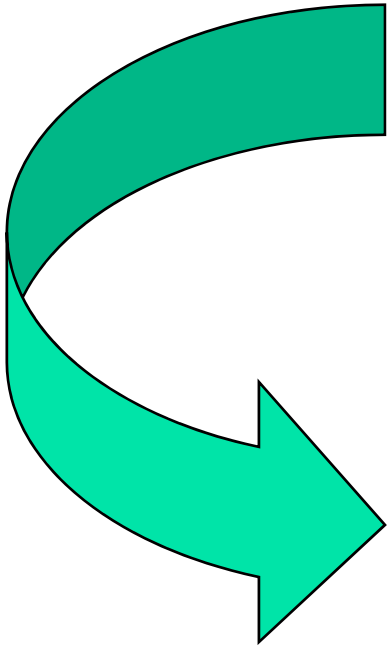
- Modelos soportados:
  1) **El modelo de herencia**:
     a) El estándar de *OMG*, *POA*:
        - Dado un interfaz *My* definido en *My.idl*, el compilador de *IDL idlj* genera *MyPOA.java (skeleton que extiende org.omg.PortableServer.Servant*)
        - Hay que dar una implementación de *My* que herede de *MyPOA*
     - B) ImplBase:
        - Dado un interfaz *My* definido en *My.idl*, el compilador de IDL idlj genera *_MyImplBase.java*
        - Hay que dar una implementación de *My* que herede de *MyImplBase*

  2) **El modelo de delegación**

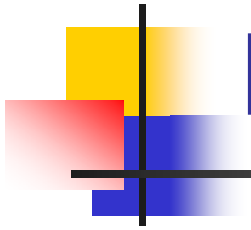
El modelo de Delegación

El modelo de delegación (*Tie model*, *Tie Delegation model*):

- Una clase *Tie,* generada por el *IDL* hereda del *skeleton*, pero delega todas las llamadas a otra clase
- Esa otra clase debe implementar las operaciones definidas en el *IDL*
- Útil si la clase de implementación tiene que heredar de alguna otra (no puede utilizarse el modelo de herencia)

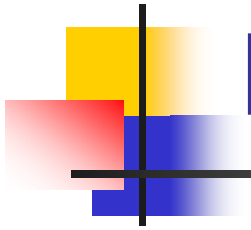


El skeleton del que hereda la clase *Tie* puede ser *POA* o *ImplBase*: *POA/Tie*, *ImplBase/Tie*



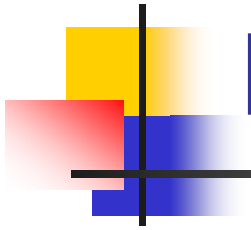
Resumen Clases Java (I)

- *org.omg.CORBA.Object*
- *org.omg.CORBA.Any*
- *org.omg.CORBA.ORB*
- *org.omg.CORBA.Request*
- *org.omg.CORBA.NameValuePair*



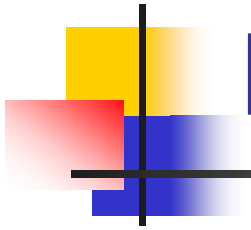
Resumen Clases Java (II)

- *org.omg.CosNaming.NamingContext*
- *org.omg.CosNaming.NamingContextExtHelper*
- *org.omg.CosNaming.NamingContextExt*
- *org.omg.CosNaming.NamingContext*
- *org.omg.CosNaming.NameComponent*



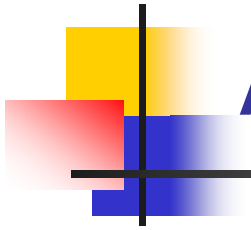
Resumen Clases Java (III)

- *org.omg.CORBA.portable.Servant*
- *org.omg.CORBA.portable.OutputStream*
- *org.omg.CORBA.portable.InputStream*
- *org.omg.CORBA.portable.IDLEntity*
- *org.omg.CORBA.portable.Streamable*
- *org.omg.CORBA.portable.ObjectImpl*
- *org.omg.CORBA.portable.InvokeHandler*



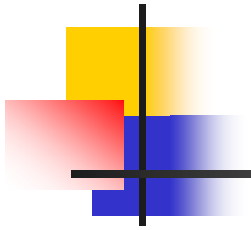
Resumen Clases Java (IV)

- *org.omg.PortableServer.Servant*
- *org.omg.PortableServer.POAHelper*
- *org.omg.PortableServer.POA*
- *org.omg.PortableServer.POAManager*



Anexo IV

CORBA en Java 1.3

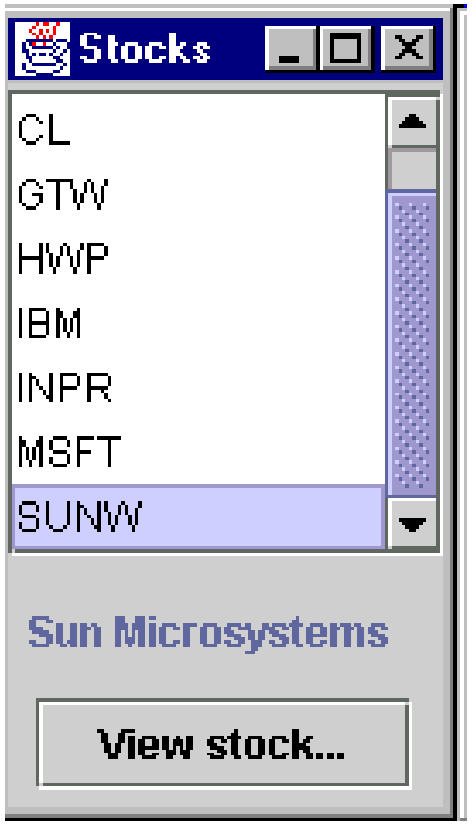


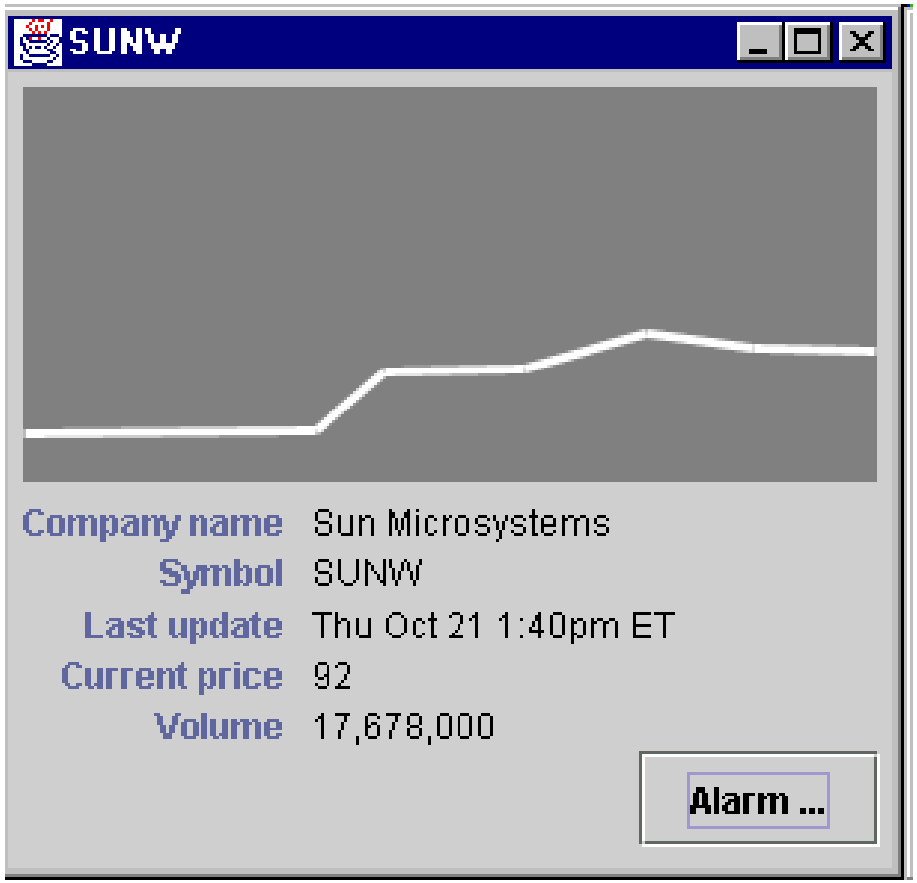
CORBA en Java 1.3

- En este anexo, presentamos un ejemplo con la implementación *CORBA* de Java 1.3
- **El propósito es meramente informativo de que hay diferencias**; hoy no tiene sentido seguir usando esa implementación. Por tanto, lo único importante es lo visto anteriormente
- El compilador de *IDL* que venía con esta versión creaba un fichero *_XImplBase* para un objeto *CORBA* remoto X, que ha sido reemplazado por la aproximación del *POA* en la nueva versión

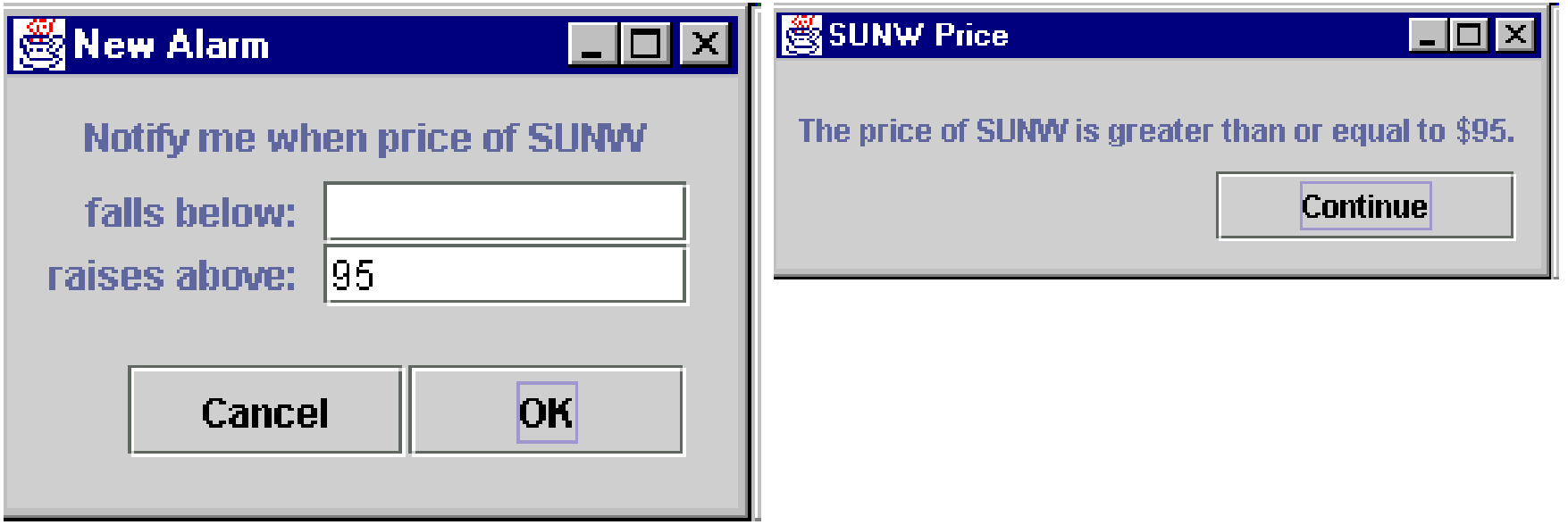
Ejemplo en: http://java.sun.com/developer/onlineTraining/corba/corba.html

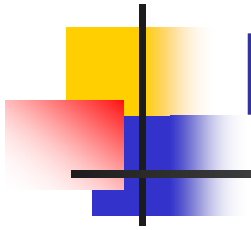
Ej.: Aplicación de Bolsa (I)





Ej.: Aplicación de Bolsa (II)





Ej.: Aplicación de Bolsa (III)

Objeto	Significado
Stock	Objeto distribuido que representa la cotización de cierta empresa
StockPresentation	Objeto distribuido del interfaz gráfico de usuario que muestra la cotización de cierta empresa
Alarm	Objeto distribuido que representa la "alarma" establecida por el usuario
AlarmPresentation	Objeto distribuido del interfaz gráfico de usuario que muestra una "alarma" que se dispara

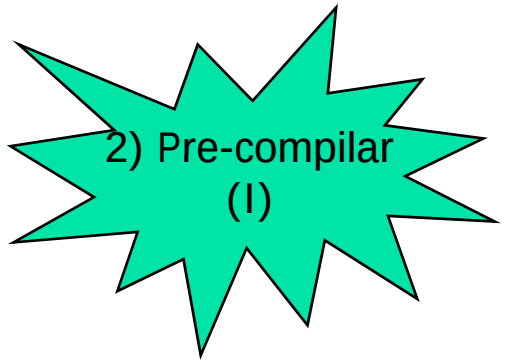
Ej.: Aplicación de Bolsa (IV)

```
module StockObjects {

  struct Quote {
    string symbol;
    long at_time;
    double price;
    long volume;
  };

  exception Unknown{};

  interface StockFactory {

    Stock create_stock(
      in string symbol,
      in string description
    );
  };

  interface Stock {

    // Returns the current stock quote.
    Quote get_quote() raises(Unknown);

    // Sets the current stock quote.
    void set_quote(in Quote stock_quote);

    // Provides the stock description,
    // e.g. company name.
    readonly attribute string description;
  };

}
```

Ej.: Aplicación de Bolsa (V)

- ***idlj***
  - compilador de *IDL* de Java 2 SDK
- ***idl2java***
  - compilador de *IDL* de VisiBroker

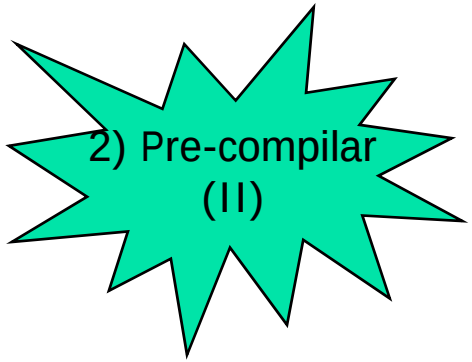
- En nuestro caso:
  - *idltojava Stock.idl*



Ej.: Aplicación de Bolsa (VI)

Ficheros obtenidos

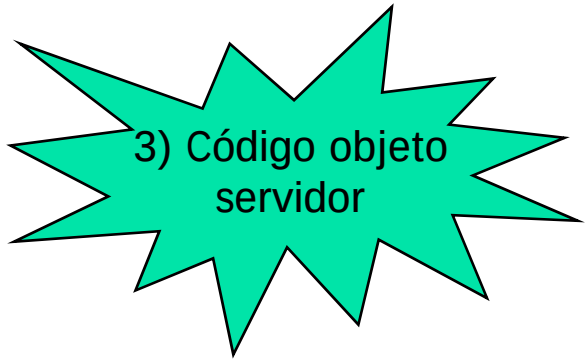
- *Stock.java*:
  - interfaz *IDL* representado como interfaz Java
- *StockHelper.java*:
  - Implementa operaciones de tipos para el interfaz
- *StockHolder.java*:
  - Para parámetros out/inout
- _*StockStub.java* (*stub* del cliente)
  - Objeto local representando al remoto
  - El cliente no lo usa directamente
  - Llamado _st_Stock.java en VisiBroker
- _*StockImplBase* (*skeleton* del servidor)
  - Punto de entrada al objeto distribuido



Luego se compilarán con *javac* y se pondrán en el CLASSPATH

Ej.: Aplicación de Bolsa (VII)

```
public class StockImpl extends StockObjects._StockImplBase {

  private Quote _quote=null;
  private String _description=null;

  public StockImpl(String name, String description) {
    super();
    _description = description;
  }

  public Quote get_quote() throws Unknown {
    if (_quote==null) throw new Unknown();
    return _quote;
  }

  public void set_quote(Quote quote) {
    _quote = quote;
  }

  public String description() {
    return _description;
  }
}
```



Ej.: Aplicación de Bolsa (VIII)

Aplicación Java que, cuando se ejecuta, pone los servicios de sus objetos a disposición de los clientes interesados

```
public class theServer {
  public static void main(String[] args) {
    try {
      ...
    }
    catch (Exception e) {
      System.err.println("Stock server error: " + e);
      e.printStackTrace(System.out);
    } /* End of catch */
  } /* End of main */
} /* End of class */
```

Ej.: Aplicación de Bolsa (IX)

- Paso 1: inicializar el ORB

  org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);

- Paso 2: instanciar al menos un objeto CORBA

  StockImpl theStock = new StockImpl("GII", "Global Industries Inc.");

- Paso 3: conectar los objetos CORBA al ORB

  orb.connect(theStock);

- Paso 4: esperar invocaciones de los clientes

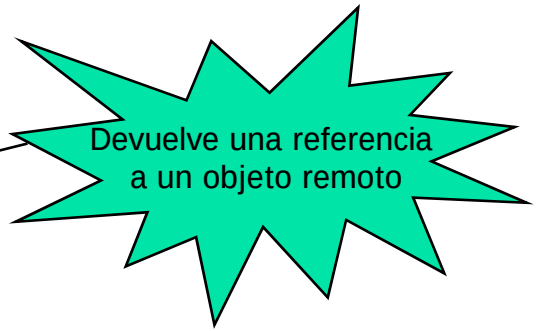
  java.lang.Object sync = new java.lang.Object();
  synchronized (sync) { sync.wait();}

Ej.: Aplicación de Bolsa (X)

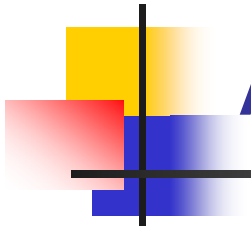
```
Stock theStock = ...
try {
    Quote current_quote =
            theStock.get_quote();
} catch (Throwable e) {
}
```

```
String stockString =
   orb.object_to_string(theStock);
org.omg.CORBA.Object obj =
    orb.string_to_object(stockString);
Stock theStock = StockHelper.narrow(obj);
```

```
StockFactory factory = ...
Stock theStock = ...
try {
   theStock = factory.create(
              "GII",
              "Global Industries Inc.");
} catch (Throwable e) {
}
```

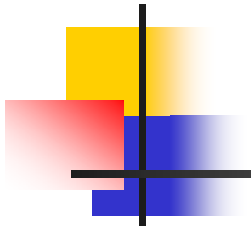


Trozos de código para un cliente



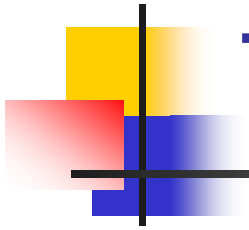
Anexo V

Ejemplos de Mapping de IDL en Java



IDL -> Java, C++

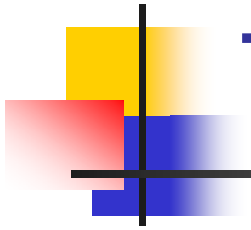
Ejemplos de *Bindings*		
IDL	**Java**	**C++**
Module	*package*	*namespace*
interface	*interface*	*abstract class*
operación	*method*	función miembro
attribute	par de métodos	par de funciones
exception	*exception*	*exception*



Tipos *IDL* -> Java (I)

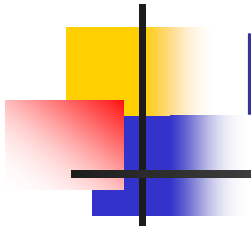
Bindings	
Tipo IDL	**Tipo Java**
boolean	*boolean*
char / wchar	*char*
octet	*byte*
short / unsigned short	*short*
long / unsigned long	*int*
long long / unsigned long long	*long*

Mapping de tipos de datos: *marshaling* para transmisión de parámetros



Tipos *IDL* -> Java (II)

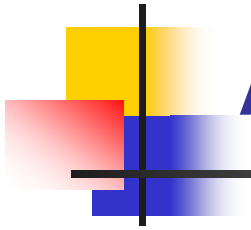
Bindings	
Tipo IDL	**Tipo Java**
float	*float*
double	*double*
string / wstring	*String*
any	*org.omg.CORBA.Any*
Object	*org.omg.CORBA.Object*



Reflexión

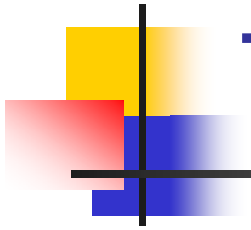
¿Cómo podríamos *mapear* a Java parámetros *out* e *inout* ?

Clases ***Holder***



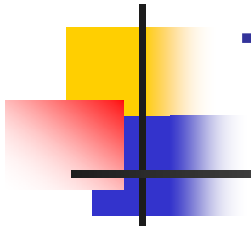
Anexo VI

Resumen de Terminología



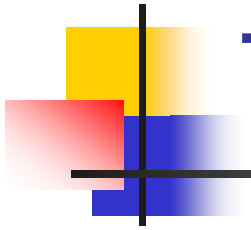
Terminología CORBA (I)

Acrónimo	Significado
CORBA	Common Object Request Broker Architecture
ORB	Object Request Broker
IDL	Interface Definition Language



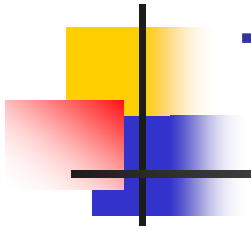
Terminología CORBA (II)

GIOP	General Inter-ORB Prococol
IIOP	Internet Inter-ORB Protocol
IOR	Interoperable Object Reference
ESIOP	Environment-Specific Inter-ORB Protocol
DCE CIOP	DCE Common Inter-ORB Protocol



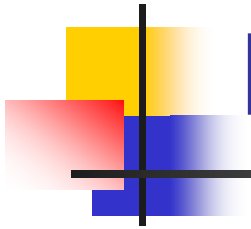
Terminología CORBA (III)

Acrónimo	Significado
IR	Implementation Repository
SII	Static Invocation Interface
SSI	Static Skeleton Interface
DII	Dynamic Invocation Interface
DSI	Dynamic Skeleton Interface



Terminología CORBA (IV)

Acrónimo	Significado
COS	Corba Services
OA	Object Adaptor
BOA	Basic Object Adaptor
POA	Portable Object Adaptor



Fin

Gracias por vuestra atención