Sistemas de Información

Tecnologías de objetos distribuidos:

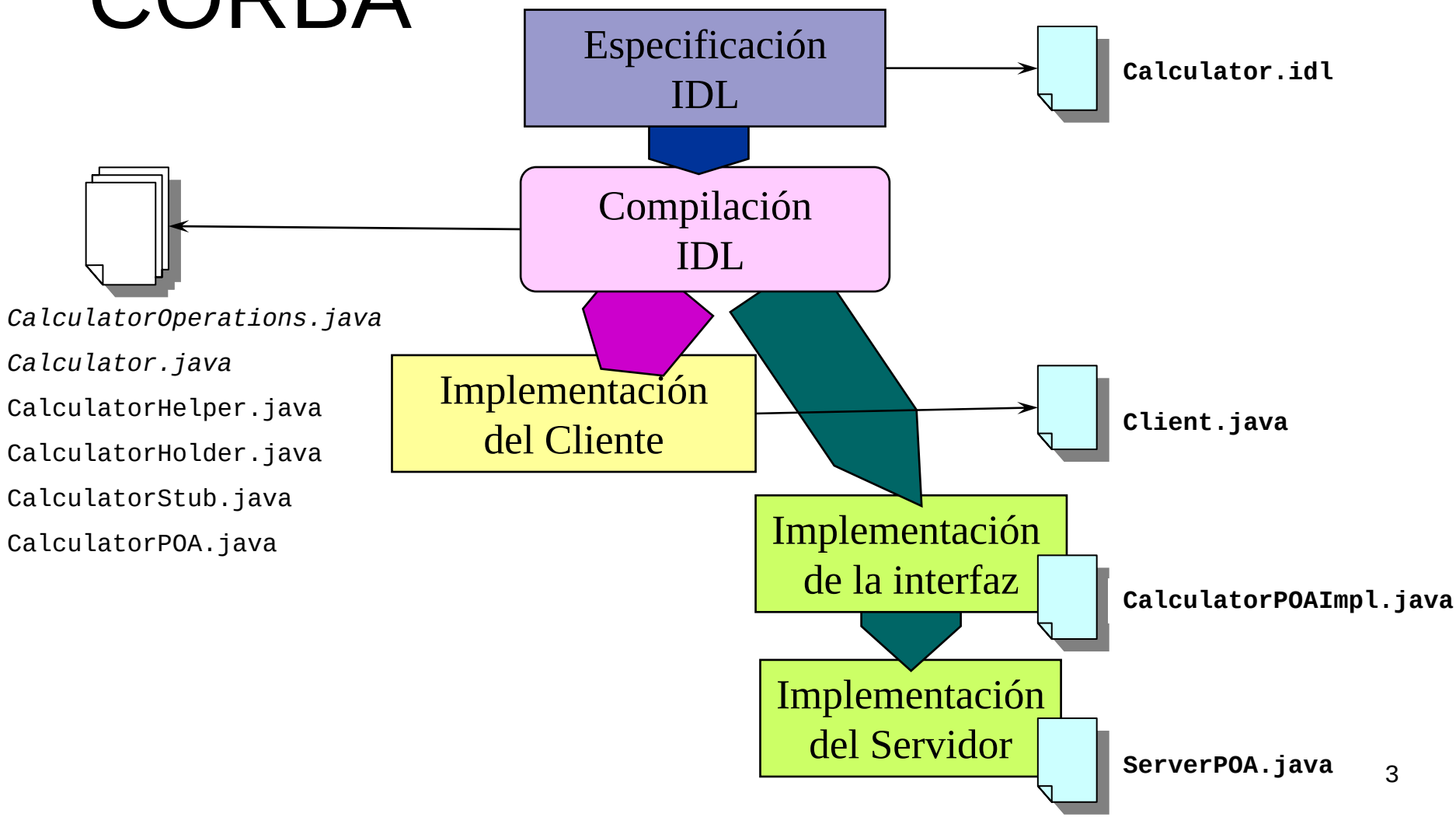
CORBA: Invocación estática

Agradecimientos: Jesus Villamor Lugo, Simon Pickin de IT/UCIIIM, Juan Pavón UCM

Ejemplo CORBA

- Servidor Corba que actua como una calculadora
- Los clientes se conectan al servidor y le piden que ejecute una operación
  - Suma
  - División
- En la petición van incluidos los parámetros necesarios para realizar la operación
- El servidor una vez realizada la operación devuelve el resultado

Construcción de una aplicación CORBA



CORBA Ejemplo 1

Declarando Interfaz en IDL

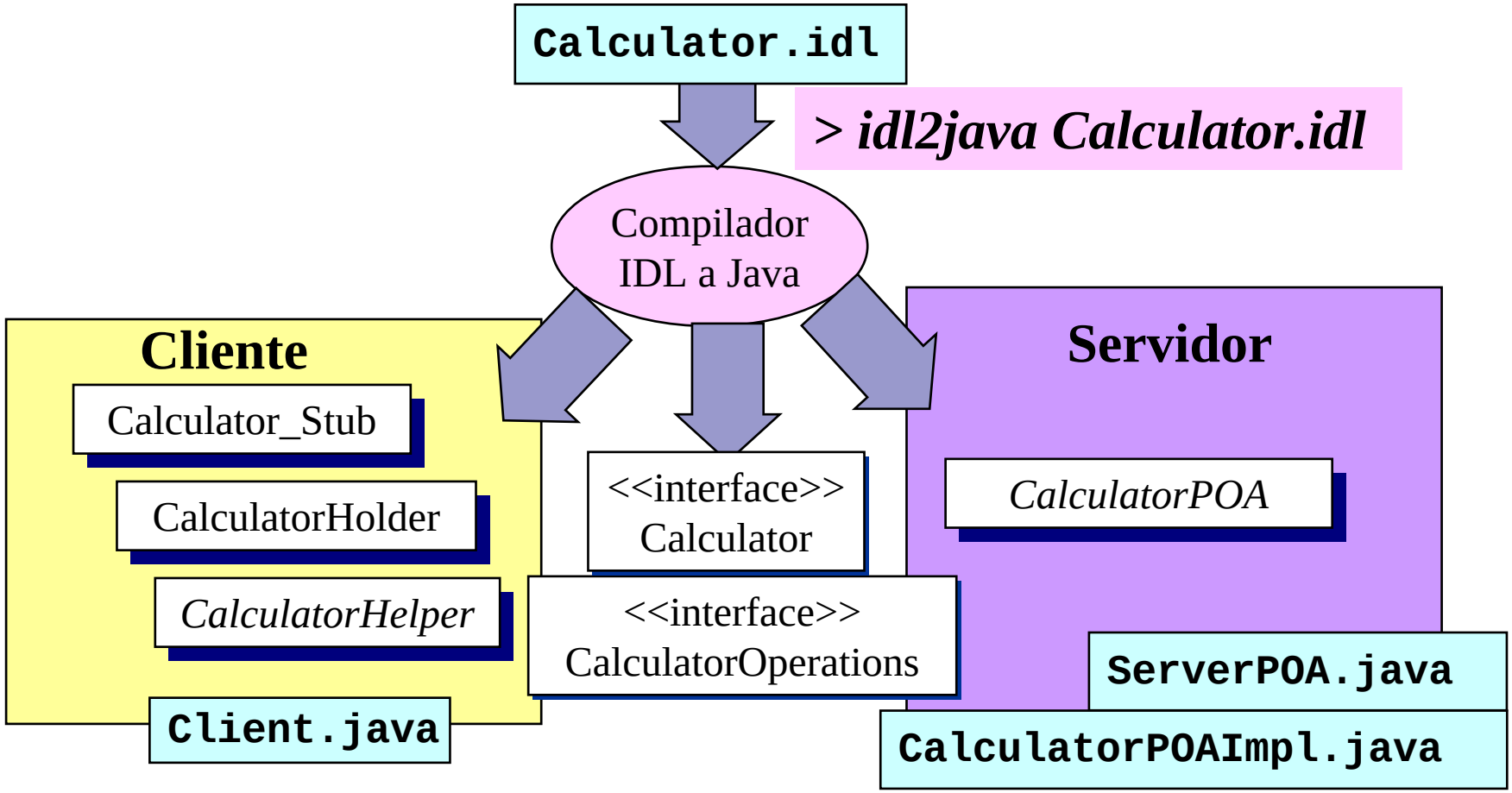
```
// Descripcion de una excepcion
exception DivisionPorCero{
  float op1;
  float op2;
};
```

```
interface Calculator{
  // operacion de Suma
  float add ( in float nb1, in float nb2 );

  // operacion de Division
  float div ( in float nb1, in float nb2 ) raises ( DivisionPorCero );
};
```

(*) Ejemplo tomado de http://www.programacion.com/tutorial/acscorba

Compilación de la interfaz



Compilando la interfaz **`Calculator y CalculatorOperations`**

- **`CalculatorOperations`** Interfaz Java que define todos los métodos correspondientes a la interfaz idl. Es la que implementa el servant
  - Cada operación de la interfaz corresponde a un método java
  - Cada atributo del interfaz corresponde a dos métodos java (leer, escribir)

```
public interface CalculatorOperations {
        //...
        float add(float nb1, float nb2);
        float div(float nb1, float nb2);
}
```

- **`Calculator`** Interfaz java que extiende a la anterior. Es la que implementa el stub

```
public interface Calculator extends CalculatorOperations,
                                    org.omg.CORBA.Object,
                        org.omg.CORBA.portable.IDLEntity {...}
```

Compilando la interfaz **`Calculator_Stub.java`**

- Clase java que implementa el stub de la interfaz Calculator en el lado del cliente
  - Hace marshalling (serialización) para los parámetros de cada método de la interfaz antes de pasárselos al ORB. (También unmarshalling en la recepción)
  - Define también su propio constructor y otros métodos de apoyo (Comprobar en prácticas editando el fichero)

```
public class Calculator_Stub
    extends org.omg.CORBA.portable.ObjectImpl
    implements Calculator{

    // ...

}
```

Compilando la interfaz **`CalculatorHelper`**

- Clase Java que proporciona métodos estáticos útiles para los usuarios de objetos Calculator (cliente y servidor).
- Por ejemplo método narrows para convertir de objetos Corba genéricos a objetos Calculator

```
final public class CalculatorHelper {

 public static void  insert(org.omg.CORBA.Any any, Calculator val)
{...}
 public static Contador extract(org.omg.CORBA.Any any) {...}
 public static Contador narrow (org.omg.CORBA.Object val)  {...}

}
```

Compilando la interfaz **`CalculatorHolder`**

- Clase Java que se utiliza si es necesario pasar objetos Calculator como parámetros `out` o `inout` en operaciones de otra interfaz

```
final public class CalculatorHolder
                        implements org.omg.CORBA.portable.Streamable {

    public Calculator value;
    public CalculatorHolder(){    }
    public CalculatorHolder(Calculator initial){
        value = initial;
    }

        //...
}
```

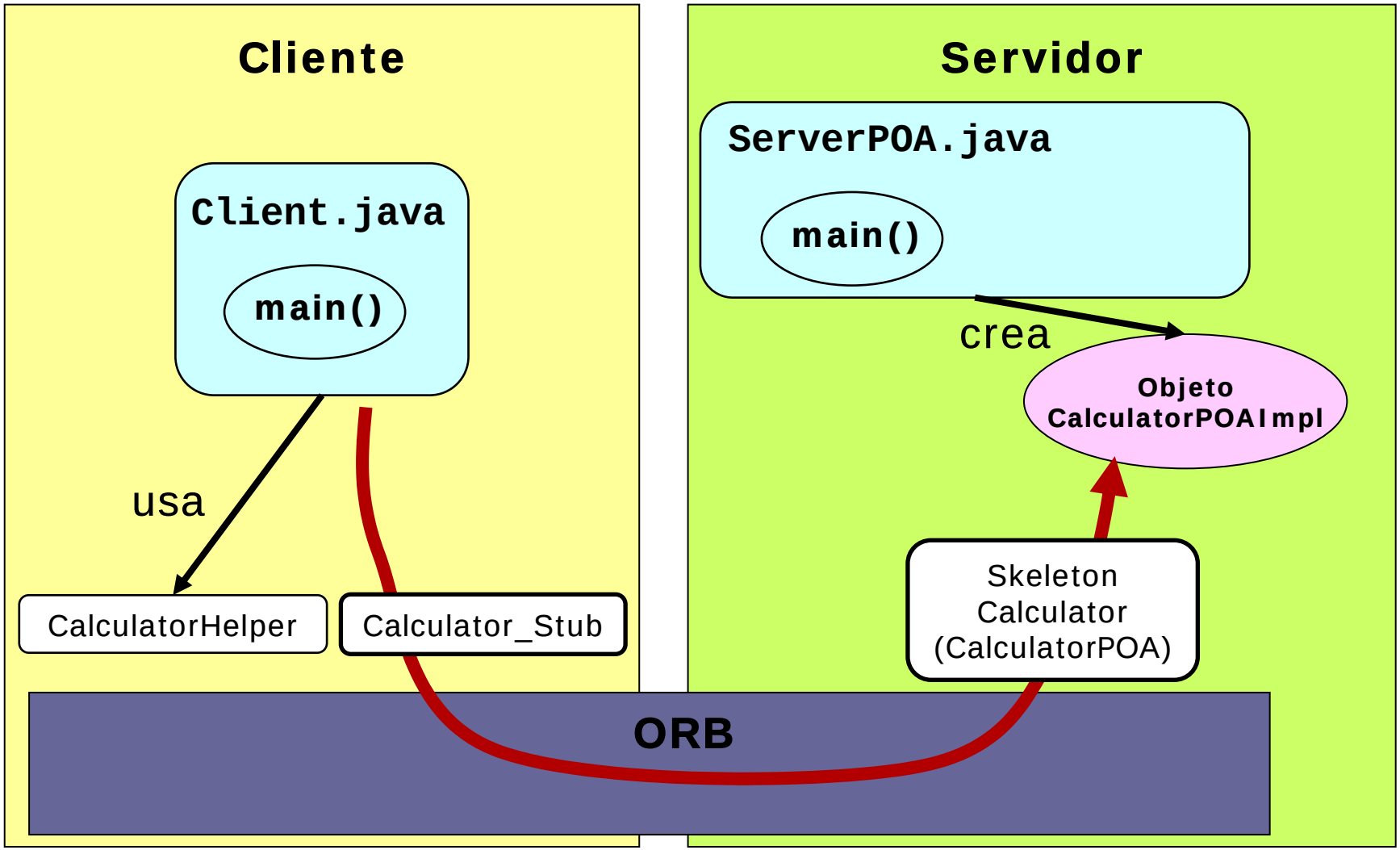
Compilando la interfaz **`CalculatorPOA`**

- Clase abstracta que sirve de base para la implementación del servant (código del servidor que implementa las operaciones definidas en la interfaz idl)
- Esta clase es la que contiene el skeleton del servidor

```
public abstract class CalculatorPOA
    extends org.omg.PortableServer.Servant
    implements org.omg.CORBA.portable.InvokeHandler,
                                    CalculatorOperations{
        // ...

    public org.omg.CORBA.portable.OutputStream _invoke(String opName,
                                org.omg.CORBA.portable.InputStream in,
                        org.omg.CORBA.portable.ResponseHandler handler){
        // ...
    }
}
```

Arquitectura de la aplicación



Programación del cliente:
Modelos de implementación

- Modelos de implementación
  - Clase java con método main
  - Applet de java con método init
- Tareas a realizar
  1. Conectar el cliente al orb (Inicializar ORB)
  2. Obtener referencia a un objeto CORBA que implemente interfaz Calculator
     - Usando método string_to_object()
     - Usando servicio de nombres (solución estándar)
  3. Usar el objeto CORBA como si fuese local

Programación del Cliente:
Conectarse al ORB

- Para conectarse al ORB es necesario invocar al método estático org.omg.CORBA.orb.init() que devuelve un objeto orb
- Sobre este objeto ose pueden invocar el resto de los métodos que definen la funcionalidad del orb
- El método init() sin parámetros devuelve un orb único
- Llamadas sucesivas a este orb devoverían ref al mismo objeto
- También se puede invocar al método con parámetros para funcionalidad adicional:
  - En aplicaciones: init(args, null)
  - En applets: init(Applet app, java.util.Properties props) ej: init(this, null)

Programación del Cliente:
Obtención de refs. a objetos corba

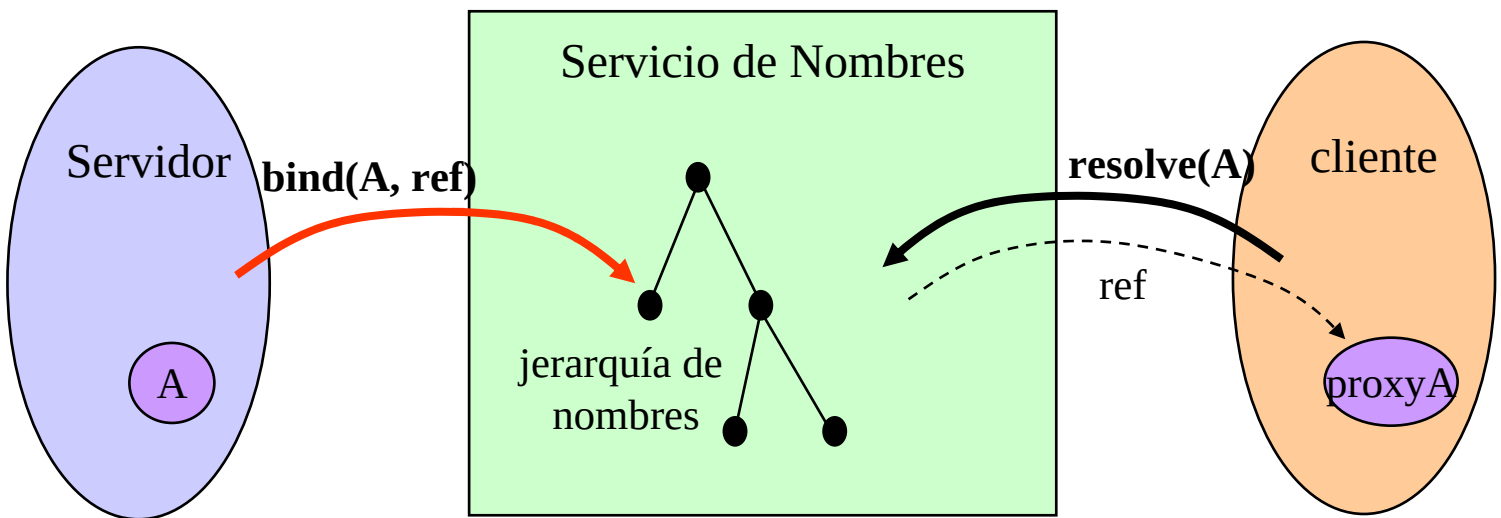
- El ORB es capaz de convertir de un string a una ref a un objeto corba y viceversa.
- La ref a un objeto corba encapsula
  - Dirección de red del proceso servidor
  - Un identificador único (puesto por el servidor) que identifica la implementación concreta a la que va dirigida la petición
- Al obtenerse una ref a un objeto corba en realidad obtenemos una ref a un objeto java que implementa en el cliente el representante (stub) del proceso servidor
- Cuando el cliente obtiene ref a objeto corba
  - El ORB instancia un proxi (stub) en el lenguaje apropiado en el espacio del cliente. El cliente no puede instanciar estas referencias lo hace siempre el ORB
  - Una vez creado el stub, el cliente realiza operaciones sobre él.
  - El stub hace marshalling de las peticiones y se las pasa al ORB
  - El ORB localiza al servidor y establece las conexiones de transporte necesarias de forma transparente para el cliente.
  - Con la respuesta del servidor se realiza el proceso inverso

Referencias a objetos

Uso del Servicio de Nombres

- El Servicio de Nombres guarda pares

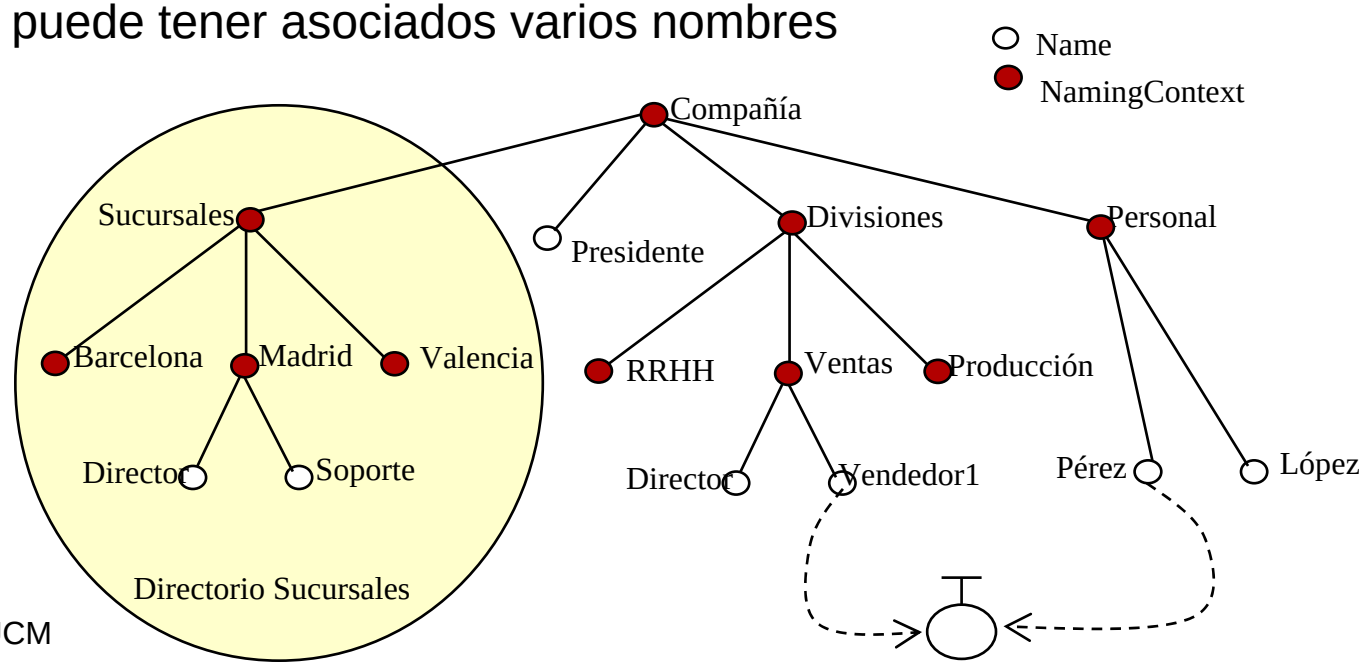
  <**nombre, referencia a objeto**>
  - Los nombres están organizados en una jerarquía
- El Servicio de Nombres es usado por cliente y servidor:
  - El servidor asocia (*bind*) en el Servicio de Nombres una referencia a objeto con un nombre
  - El cliente puede pedirle al Servicio de Nombres que a partir de un nombre le dé (*resolve*) una referencia a un objeto CORBA



(*) Gráfico: Juan Pavón. UCM

Referencias a objetos
Uso del Servicio de Nombres

- Los nombres en el Servicio de Nombres se organizan jerárquicamente (como sist de ficheros)
  - Cada nodo en la jerarquía de nombres puede ser:
    - **NamingContext**: Define un espacio de nombres
    - **Name**: Tiene asociada una referencia a un objeto
    - Cada NameComponent es un par identificador, clase **<id, kind>**. Kind es opcional
  - Un objeto puede tener asociados varios nombres



(*) Gráfico: Juan Pavón. UCM

CORBA Ejemplo 1

Desarrollo del cliente

```
public class Client{
  public static void main( String args[] )  {
    org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);
    org.omg.CORBA.Object obj = null;
    org.omg.CosNaming.NamingContext naming = null;
    try{
    }
    org.omg.CosNaming.NameComponent [] name = new
    org.omg.CosNaming.NameComponent[1];
    name[0] = new org.omg.CosNaming.NameComponent();
    name[0].id = "Calculator";
    name[0].kind = "Example";
    (...)
 } // fin del main
} // fin de la clase
```

1) Inicializar el ORB

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

CORBA Ejemplo 1

Desarrollo del cliente

```
public class Client{
  public static void main( String args[] )  {
      org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);
      org.omg.CORBA.Object obj = null;
      org.omg.CosNaming.NamingContext naming = null;
      try{
            obj = orb.resolve_initial_references("NamingService");
            naming = org.omg.CosNaming.NamingContextHelper.narrow(obj);
      }catch ( org.omg.CORBA.ORBPackage.InvalidName name ){
            System.out.println("No se ha podido obtener el NamingService");
            System.exit(0);
      }
      org.omg.CosNaming.NameComponent [] name = new
      org.omg.CosNaming.NameComponent[1];
```

2) Localizar el naming Service y obtener ref a él:

```
obj = orb.resolve_initial_references("NamingService");
naming = org.omg.CosNaming.NamingContextHelper.narrow(obj)
```

```
} // fin de la clase
```

CORBA Ejemplo 1

Desarrollo del cliente

3) Construir nombre del objeto calculator:

```
org.omg.CosNaming.NameComponent [ ] name =
                    new org.omg.CosNaming.NameComponent[1];
name[0] = new org.omg.CosNaming.NameComponent();
name[0].id = "Calculator";
name[0].kind = "Example";
```

```
pu
    naming = org.omg.CosNaming.NamingContextHelper.narrow(obj);
  }catch ( org.omg.CORBA.ORBPackage.InvalidName name ){
        System.out.println("No se ha podido obtener el NamingService");
        System.exit(0);
  }
  org.omg.CosNaming.NameComponent [] name = new org.omg.CosNaming.NameComponent[1];
  name[0] = new org.omg.CosNaming.NameComponent();
  name[0].id = "Calculator";
  name[0].kind = "Example";
  (...)
 } // fin del main
} // fin de la clase
```

CORBA Ejemplo 1

Desarrollo del cliente

```
public class Client{
  public static void main( String args[] )  {
  (...)
      try{
            obj = naming.resolve(name);
      }catch ( org.omg.CosNaming.NamingContextPackage.NotFound ex ){
            System.out.println("Objeto no encontrado en el NamingService");
            System.exit(0);
      }catch ( org.omg.CosNaming.NamingContextPackage.CannotProceed ex ){
            System.out.println("No se ha podido continuar");
            System.exit(0);
      }catch ( org.omg.CosNaming.NamingContextPackage.InvalidName ex ){
            System.out.println("Nombre invalido");
            System.exit(0);
      }
 (...)
```

4) Localizar ref del objeto calculator en el naming Service:
obj = naming.resolve(name);

CORBA Ejemplo 1

Desarrollo del cliente

```
public class Client{
  public static void main( String args[] )  {
(...)
    Calculator calc = CalculatorHelper.narrow(obj);
    try{
            System.out.println("5 + 3 = " + calc.add(5,3) );
            System.out.println("5 / 0 = " + calc.div(5,0) );
    }
    catch ( DivisionPorCero ex ){
            System.out.println("Interceptada intento de divisón por cero");
            System.out.println(ex.getMessage());
    }
 } // fin del main
} // fin de la clase
```

5) Narrow de obj genérico Corba al objeto que buscamos:

```
Calculator calc = CalculatorHelper.narrow(obj);
```

CORBA Ejemplo 1

Desarrollo del cliente

```
public class Client{
  public static void main( String args[] )  {
(...)
    Calculator calc = CalculatorHelper.narrow(obj);
    try{
            System.out.println("5 + 3 = " + calc.add(5,3) );
            System.out.println("5 / 0 = " + calc.div(5,0) );
    }
    catch ( DivisionPorCero ex ){
            System.out.println("Interceptada intento de divisón por cero");
            System.out.println("La division era "+ex.op1+" / "+ex.op2);
    }
    catch ( org.omg.CORBA.SystemException ex ){
            System.out.println("Interceptada una excepcion CORBA System");
            System.out.println(ex.getMessage());
    }
```

6) Uso de la ref objeto remoto como si fuera local:

```
System.out.println("5+3 = " + calc.add(5,3));
System.out.println("5/0 = " + calc.div(5,0));
```

Programación del servidor
Tareas a realizar

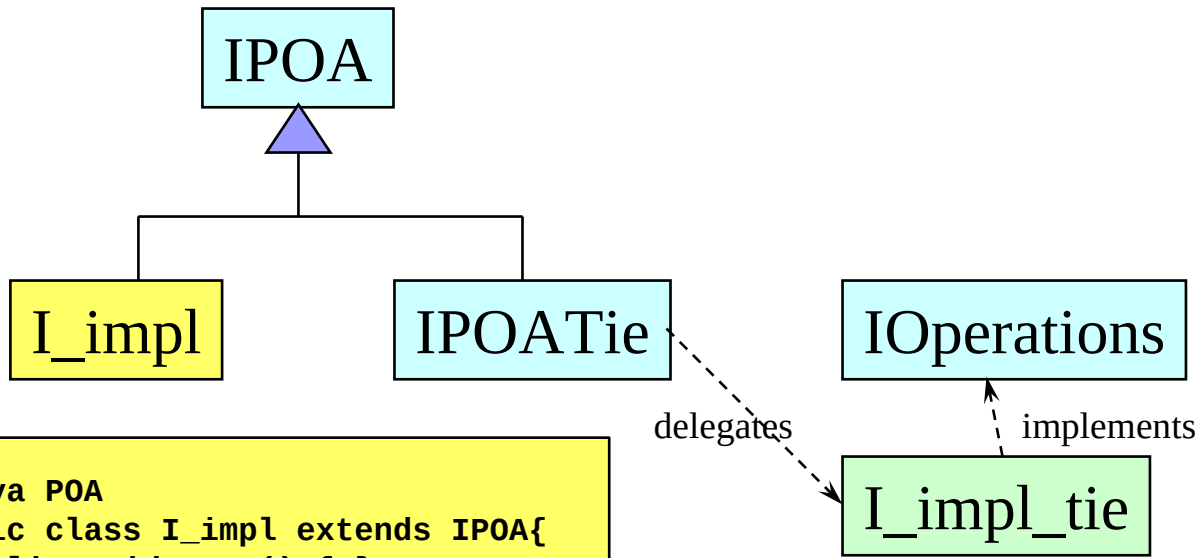
1. Programación de servants.
   - Implementación de las interfaces idl usando skeletons generados por el compilador idl (2 modelos)
     - Por herencia (CalculatorPOA)
     - Por delegación (CalculatorPOATie)
2. Creación del programa principal (Server)
   - Contiene el método main que se encarga de:
     - Inicializar el ORB y POA
     - Crear objetos servant que implementan las interfaces
     - Pasar el control al ORB

Programación del servidor:
Modelos de implementación servants

- **Herencia**
  - Se hereda de la clase POA
    - El adaptador de objetos ya nos da una implementación de base
    - El adaptador de objetos hace las funciones de skeleton
  - Pros
    - Más limpio
  - Cons
    - No permite herencia múltiple (Java: no permite herencia)
- **Delegación**
  - Se implementa la interfaz Operations
    - Una clase generada TIE hace las funciones de skeleton y
    - TIE delega las peticiones a nuestra implementación
  - Pros
    - Evita jerarquía de herencia complejas
  - Cons
    - Inicialmente no era un mecanismo estándar (ahora sí)
    - Es menos eficiciente

Herencia vs Delegación

Implementación del Servant




```
//IDL
interface A {
  void op_a();
}
inteface B {
  void op_b();
}
interface I: A, B {
  void op_i();
}
```

```
//Java POA
public class I_impl extends IPOA{
  public void op_a() { }
  public void op_b() { }
  public void op_i() { }
}
```

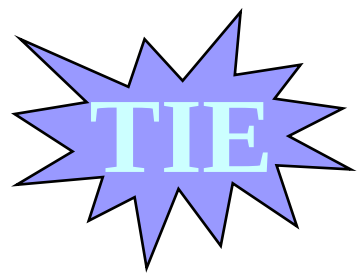
```
//Java TIE
public class A_impl implements AOperations{
  public void op_a() { }
}
public class B_impl implements BOperations{
  public void op_b() { }
}
public class I_impl extends B_impl implements IOperations{
  public void op_a() { }
  public void op_i() { }
}
```

Herencia vs Delegación

Creación y activación de Servants

```
// Java POA
// Creación
I_impl impl= new I_impl();
I_impl otraImpl= new I_impl();
// Activación
Orb.omg.CORBA.ORB orb=...
I_ref= impl._this(orb);
I_otraRef= otraImpl._this(orb);
```


```
// Java TIE
// Creación
I_impl_tie impl= new I_impl_tie();
IPOATie tie= new IPOATie(impl);
// Activación
Orb.omg.CORBA.ORB orb=...
I_ref= tie._this(orb);
```



Programación del Servidor

Programa principal

- Conecta el servidor al orb (init)
- Obtiene referencia al POA raíz
  - POA se encarga de redirigir las peticiones realizadas a un objeto corba al servant correspondiente
- Crea y activa los servants
  - Se crean como cualquier objeto java
  - Se ctivan con el método _this() esto permite
    - Darlo a conocer al POA
    - Devuelve una referencia del objeto CORBA asociado
- Publica referencias para hacerlas accesibles a los clientes
- Activa el gestor del POA raíz.
  - El gestor se encarga de decidir si las peticiones se ponen en cola, se descartan o se pasan al POA (varias políticas para hacerlo)
- Inicia bucle de recepción de eventos

CORBA Ejemplo 1

Desarrollo del servidor (servant)

```
public class CalculatorPOAImpl extends CalculatorPOA{
  public float add(float nb1, float nb2){
    System.out.println("Suma = "+nb1+" + "+nb2);
    return nb1 + nb2;
  }
  public float div(float nb1, float nb2) throws
    DivisionByZero {
    System.out.println("Division = "+nb1+" / "+nb2);
    if ( nb2 == 0 ){
            throw new DivisionPorCero(nb1,nb2);
    }
    return nb1 / nb2;
  }
}
```

CORBA Ejemplo 1

Desarrollo del servidor (main)

```
public class ServerPOA{
  public static void main( String args[] ) {
     org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);
     org.omg.CORBA.Object objPoa = null;
     org.omg.PortableServer.POA rootPOA = null;
     try{
     objPoa = orb.resolve_initial_references("RootPOA");
     }catch ( org.omg.CORBA.ORBPackage.InvalidName ex ){}
     rootPOA = org.omg.PortableServer.POAHelper.narrow(objPoa);
     CalculatorPOAImpl calc = new CalculatorPOAImpl();
  (...)
 } // fin del main
} // fin de la clase
```

1) Inicializar el ORB

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

CORBA Ejemplo 1

Desarrollo del servidor (main)

```
public class ServerPOA{
  public static void main( String args[] ) {
      org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);
      org.omg.CORBA.Object objPoa = null;
      org.omg.PortableServer.POA rootPOA = null;
      try{
      objPoa = orb.resolve_initial_references("RootPOA");
      }catch ( org.omg.CORBA.ORBPackage.InvalidName ex ){}
      rootPOA = org.omg.PortableServer.POAHelper.narrow(objPoa);
      CalculatorPOAImpl calc = new CalculatorPOAImpl();
  (...)
 } // fin del main
} // fin de la clase
```

2) Resolver referencia al adaptador de objetos raíz RootPOA

```
objPoa = orb.resolve_initial_references("RootPOA");
```

CORBA Ejemplo 1

Desarrollo del servidor (main)

```
public class ServerPOA{
  public static void main( String args[] ) {
      org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);
      org.omg.CORBA.Object objPoa = null;
      org.omg.PortableServer.POA rootPOA = null;
      try{
      objPoa = orb.resolve_initial_references("RootPOA");
      }catch ( org.omg.CORBA.ORBPackage.InvalidName ex ){}
      rootPOA = org.omg.PortableServer.POAHelper.narrow(objPoa);
      CalculatorPOAImpl calc = new CalculatorPOAImpl();
  (...)
 } // fin del main
} // fin de la clase
```

3) Narrow de ref genérica a ref de adaptador (POA)

```
org.omg.PortableServer.POA rootPOA = null;
rootPOA =
org.omg.PortableServer.POAHelper.narrow(objPoa);
```

CORBA Ejemplo 1

Desarrollo del servidor (main)

```
public class ServerPOA{
  public static void main( String args[] ) {
      org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);
      org.omg.CORBA.Object objPoa = null;
      org.omg.PortableServer.POA rootPOA = null;
      try{
      objPoa = orb.resolve_initial_references("RootPOA");
      }catch ( org.omg.CORBA.ORBPackage.InvalidName ex ){}
      rootPOA = org.omg.PortableServer.POAHelper.narrow(objPoa);
      CalculatorPOAImpl calc = new CalculatorPOAImpl();
  (...)
 } // fin del main
} // fin de la clase
```

4) Crear el objeto Calculator

```
CalculatorPOAImpl calc = new CalculatorPOAImpl();
```

CORBA Ejemplo 1

Desarrollo del servidor (main)

```
public class ServerPOA{
  public static void main( String args[] ) {
 (...)
    try {
      byte[] servantId = rootPOA.activate_object(calc);
      org.omg.CORBA.Object ref = rootPOA.id_to_reference(servantId);
      org.omg.CORBA.Object obj = null;
      org.omg.CosNaming.NamingContext naming = null;
      try {
        obj = orb.resolve_initial_references("NamingService");
        System.out.println("Localizado el NamingService");
        naming = org.omg.CosNaming.NamingContextHelper.narrow(obj);
        System.out.println("Narrow del NamingService");
      } catch ( org.omg.CORBA.ORBPackage.InvalidName name ){
 } // fin del main
} // fin de la clase
```

5) Activar el servidor dentro del ORB

```
byte[] servantId = rootPOA.activate_object(calc);
```

CORBA Ejemplo 1

Desarrollo del servidor (main)

```
public class ServerPOA{
  public static void main( String args[] ) {
 (...)
    try {
      byte[] servantId = rootPOA.activate_object(calc);
      org.omg.CORBA.Object ref = rootPOA.id_to_reference(servantId);
      org.omg.CORBA.Object obj = null;
      org.omg.CosNaming.NamingContext naming = null;
      try {
        obj = orb.resolve_initial_references("NamingService");
        System.out.println("Localizado el NamingService");
        naming = org.omg.CosNaming.NamingContextHelper.narrow(obj);
        System.out.println("Narrow del NamingService");
      } catch ( org.omg.CORBA.ORBPackage.InvalidName name ){
```

6)Obtener la referencia al servidor

```
org.omg.CORBA.Object ref =
 rootPOA.id_to_reference(servantId);
```

CORBA Ejemplo 1

Desarrollo del servidor (main)

7) Acceder al Naming Service

```
obj = orb.resolve_initial_references("NamingService");
naming = org.omg.CosNaming.NamingContextHelper.narrow(obj);
```

```
        byte[] servantId = rootPOA.activate_object(calc);
        org.omg.CORBA.Object ref = rootPOA.id_to_reference(servantId);
        org.omg.CORBA.Object obj = null;
        org.omg.CosNaming.NamingContext naming = null;
        try {
          obj = orb.resolve_initial_references("NamingService");
          System.out.println("Locallizado el NamingService");
          naming = org.omg.CosNaming.NamingContextHelper.narrow(obj);
          System.out.println("Narrow del NamingService");
        } catch ( org.omg.CORBA.ORBPackage.InvalidName name ){
          System.out.println("No se ha podido obtener el NamingService");
          System.exit(0);
        }
 (...)
 } // fin del main
} // fin de la clase
```

CORBA Ejemplo 1

Desarrollo del servidor (main)

```
public class ServerPOA{
  public static void main( String args[] ) {
      (...)
       if ( naming == null ) {
         System.out.println("No se ha encontrado el NamingService");
         System.exit(0);
       }
      org.omg.CosNaming.NameComponent [] name = new
      org.omg.CosNaming.NameComponent[1];
      name[0] = new org.omg.CosNaming.NameComponent();
      name[0].id = "Calculator";
      name[0].kind = "Example";
 (...)
```

7) Construir nombre del objeto calculator: *(=que 2 de cliente)*

```
org.omg.CosNaming.NameComponent [ ] name =
                  new org.omg.CosNaming.NameComponent[1];
name[0] = new org.omg.CosNaming.NameComponent();
name[0].id = "Calculator";
name[0].kind = "Example";
```

CORBA Ejemplo 1

Desarrollo del servidor (main)

```
public class ServerPOA{
  public static void main( String args[] ) {
      (...)
      try{
        naming.bind(name,ref);
      } catch ( org.omg.CosNaming.NamingContextPackage.NotFound ex ){
        System.out.println("Objeto no encontrado");
        System.exit(0);
      }catch ( org.omg.CosNaming.NamingContextPackage.AlreadyBound ex ){
        System.out.println("Ya hay un objeto con ese nombre");
        naming.unbind(name);
        System.exit(0);
      }catch ( org.omg.CosNaming.NamingContextPackage.InvalidName ex ){
        System.out.println("Nombre inválido");
        System.exit(0);
      }catch ( org.omg.CosNaming.NamingContextPackage.CannotProceed ex ){
        System.out.println("No se ha podido continuar");
        System.exit(0);
      }
```

8) Enlazar ref del objeto calculator con su nombre
`naming.bind(name,ref);`

CORBA Ejemplo 1

Desarrollo del servidor (main)

```
public class ServerPOA{
  public static void main( String args[] ) {
     (   )
     rootPOA.the_POAManager().activate();
      System.out.println("El servidor está preparado...");
      orb.run();
    } // fin del try
    catch ( java.lang.Exception ex )
    {
      System.out.println("Se ha capturado una excepción");
      ex.printStackTrace();
    }

  } // fin del main
} // fin de la clase
```

9) Activar gestor de invocaciones del POA
`rootPOA.the_POAManager().activate();`

CORBA Ejemplo 1

Desarrollo del servidor (main)

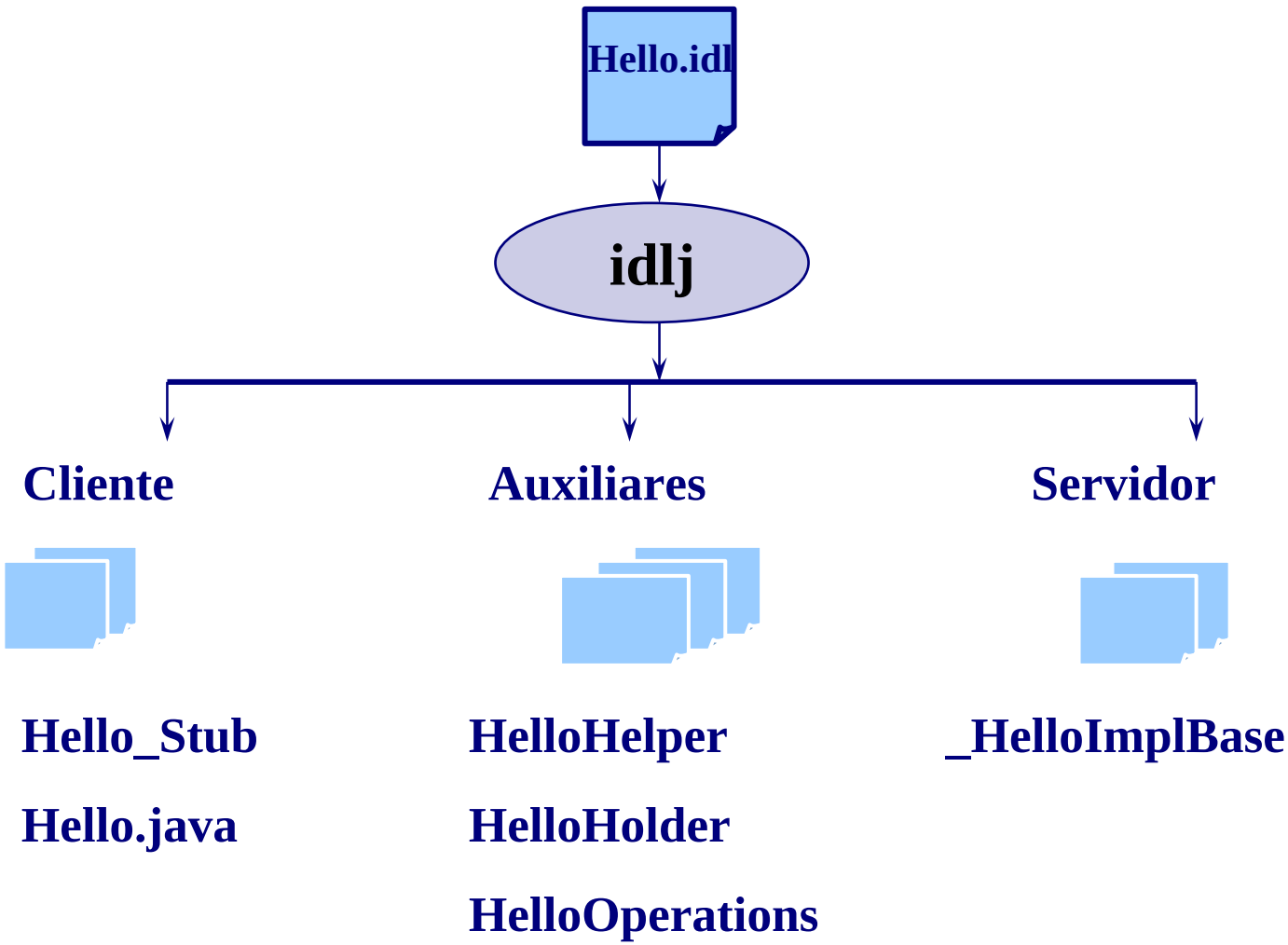
```
public class ServerPOA{
  public static void main( String args[] ) {
      (...)
      rootPOA.the_POAManager().activate();
      System.out.println("El servidor está preparado...");
      orb.run();
    } // fin del try
    catch ( java.lang.Exception ex )
    {
      System.out.println("Se ha capturado una excepción");
      ex.printStackTrace();
    }

  } // fin del main
} // fin de la clase
```

10) Ceder control a ORB para que escuche peticiones
`orb.run();`

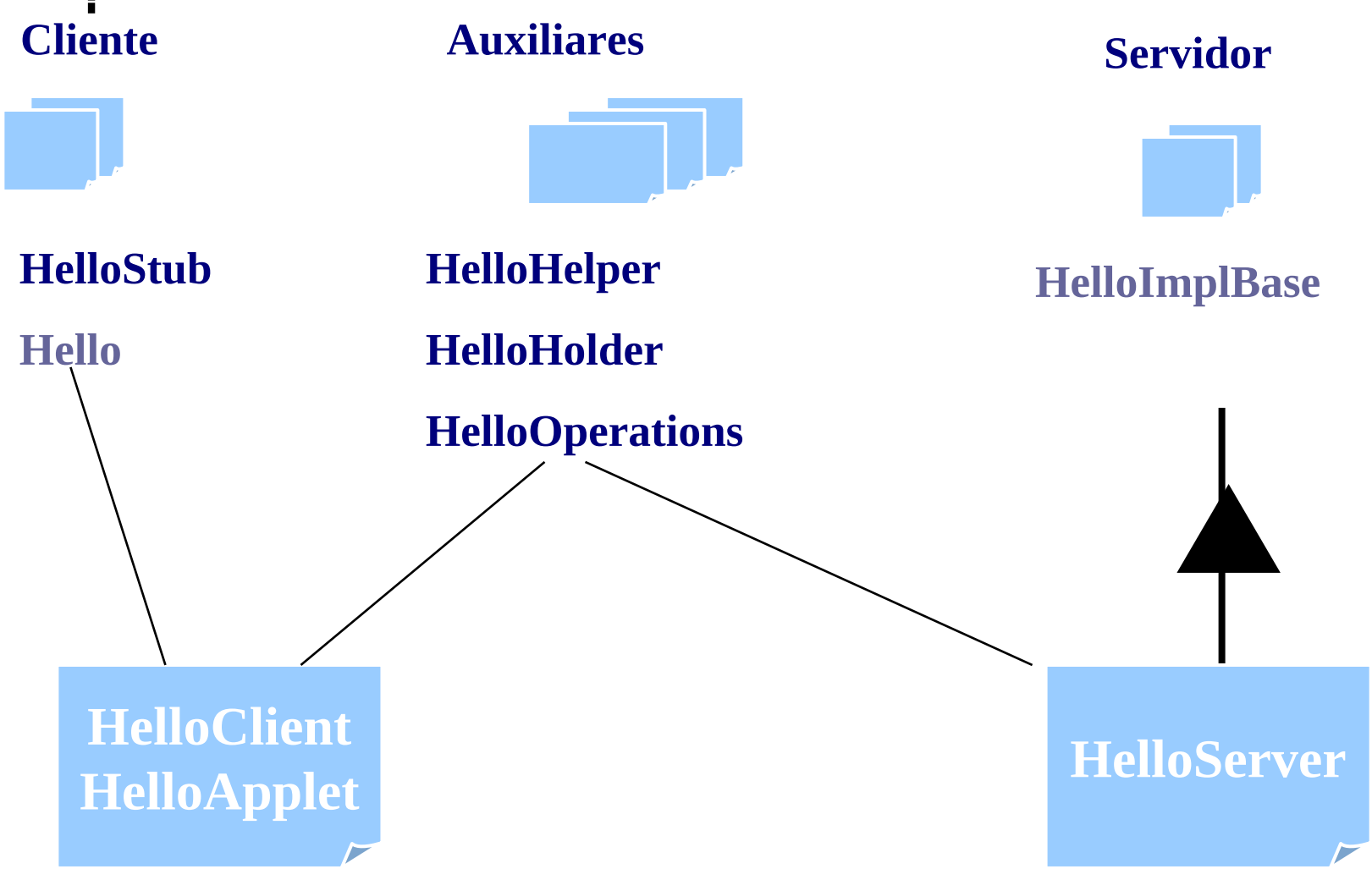
Ejemplo2: HelloWorld

Ficheros generados por (idl2java)



Un desarrollo sencillo

Implementación



Un desarrollo Sencillo (1/3)

```
// hello.idl

module HelloApp {

     interface Hello
     {
          string sayHello();
     };
};
```

```
// HelloServant.java

public class HelloServant extends _HelloImplBase
{
     public String sayHello()
     {
          return "\nHello world !!\n";
     }
}
```

Un desarrollo sencillo (2/3)

A notar: en este ejemplo utilizamos el servicio de nombres de base (y no el INS)

// HelloServer.java

```
public class HelloServer
{
    public static void main(String args[])
    {
        // create and initialize the ORB
        ORB orb = ORB.init(args, null);

        // create servant and connect it to the ORB
        HelloServant helloRef = new HelloServant();
        orb.connect(helloRef);

        // get the root naming context from Name Service
        org.omg.CORBA.Object objRef =
            orb.resolve_initial_references("NameService");
        NamingContext ncRef = NamingContextHelper.narrow(objRef);

        // bind the Hello object in Name Service
        NameComponent nc = new NameComponent("Hello", "");
        NameComponent path[] = {nc};
        ncRef.rebind(path, helloRef);

        // wait for invocations
    }
}
```

Un desarrollo Sencillo (3/3)

```
// HelloClient.java
public class HelloClient
{
    public static void main(String args[])
    {
        // create and initialize the ORB
        ORB orb = ORB.init(args, null);

        // get the root naming context from Name Service
        org.omg.CORBA.Object objRef =
            orb.resolve_initial_references("NameService");
        NamingContext ncRef = NamingContextHelper.narrow(objRef);

        // get the Hello server's object reference and narrow it
        NameComponent nc = new NameComponent("Hello", "");
        NameComponent[] path = {nc};
        Hello helloRef = HelloHelper.narrow(ncRef.resolve(path));

        // invoke the Hello server object and print results
        String hello = helloRef.sayHello();
        System.out.println(hello);
    }

}
```

Algunas referencias

– ***Dan Harkey, Robert Orfali***, ***Client/Server Programming with Java and CORBA, 2nd Edition*** *2nd. Edition (1998) John Wiley & Sons, Inc. ISBN: 0-471-24578-X Mirarse los capítulos 2, 3 y 21*

– ***Andreas Vogel and Keith Duddy (1998).*** *JAVA Programming with CORBA (Advanced Techniques for Building Distributed Applications). John Wiley & Sons, 1998. Inc. ISBN 0-471-24765-0*

–***Vinoski, Steve (1997).*** *CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments. IEEE Communications Magazine. Febrero 1997.*