E.6 The Naming Service

- Binding of object references to symbolic names (like the RMI registry)
- Hierarchical name space

1 IDL Interface

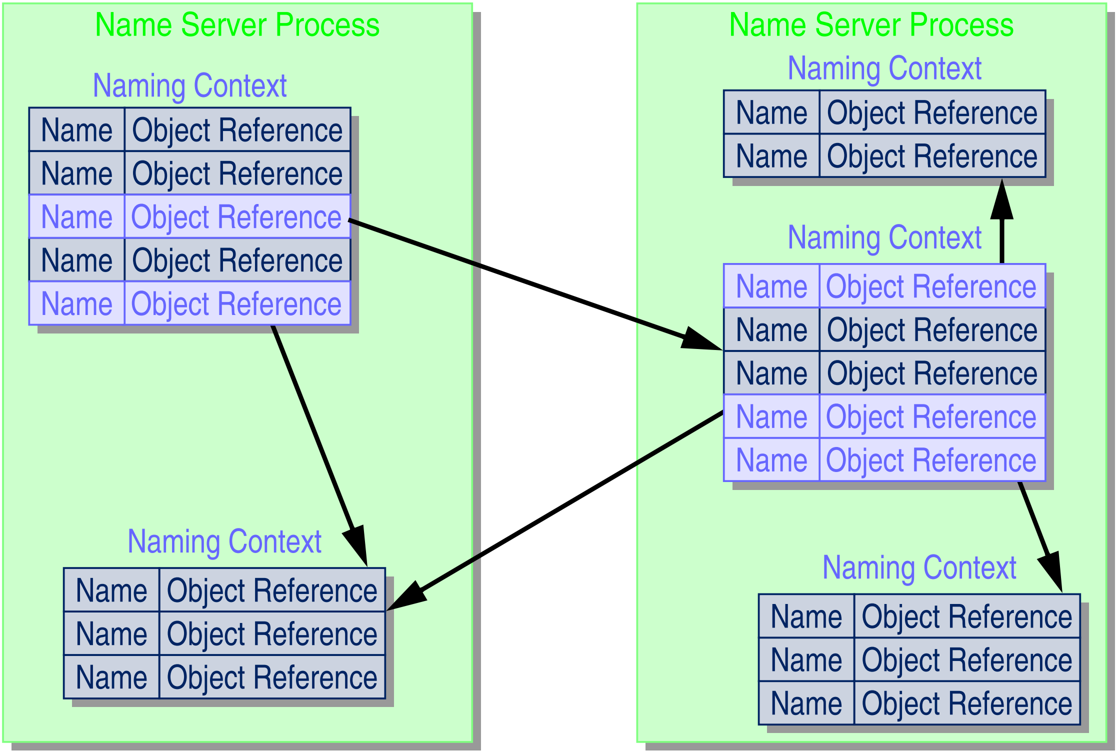
```
module CosNaming {

    typedef string Istring;
    struct NameComponent {
        Istring id;
        Istring kind;
    };
    typedef sequence <NameComponent> Name;

    interface NamingContext {

        void bind(in Name n, in Object obj) raises(NotFound,
            CannotProceed, InvalidName, AlreadyBound);
        void rebind(in Name n, in Object obj) raises(
            NotFound, CannotProceed, InvalidName);
```


1 IDL Interface

```
        void bind_context(in Name n, in NamingContext nc)
            raises(NotFound, CannotProceed, InvalidName,
                   AlreadyBound);
        void rebind_context(in Name n, in NamingContext nc)
            raises(NotFound, CannotProceed, InvalidName);
        void unbind(in Name n)
            raises(NotFound, CannotProceed, InvalidName);

        Object resolve(in Name n)
            raises(NotFound, CannotProceed, InvalidName);

        NamingContext new_context();
        NamingContext bind_new_context(in Name n)
            raises(NotFound, AlreadyBound,
                   CannotProceed, InvalidName);
        void destroy() raises(NotEmpty);

        void list( in unsigned long    how_many,
                   out BindingList     bl,
                   out BindingIterator bi);
        ...
    }; // end of interface NamingContext
    ...
}; // end of module CosNaming
```

2 Hierarchy of Naming Contexts

- Naming Contexts in multiple name server processes



3 Getting the Root Naming Context

- ORB has the reference

```
org.omg.CORBA.Object o =
        orb.resolve_initial_references("NameService");
org.omg.CosNaming.NamingContext root_context =
        org.omg.CosNaming.NamingContextHelper.narrow( o );
```

- ORB gets Root Naming Context via command line parameter
  - ◆ Since Interoperable Naming Service (INS) Specification (November 2000)
  - ◆ ORB evaluates all parameters of the form **-ORB**<suffix> <value>
  - ◆ For initial references:

    -ORBInitRef *ObjectID=Reference*

  - ◆ Human readable object references:

    corbaloc: *Protocol* **:** *Host* **:** *Port* **/** *ObjectId*

  - ◆ Example:

```
-ORBInitRef NameService=corbaloc::faui40u.informatik.uni-
erlangen.de:4711/NameService
```

4 Hello World Client With Naming Service

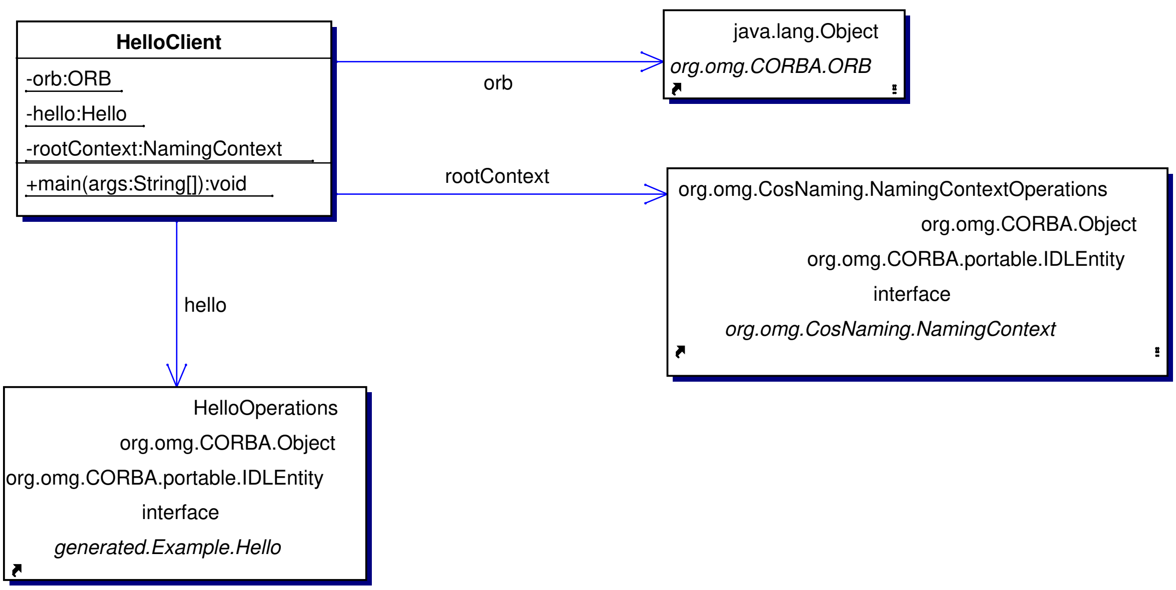
- Import additional package

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

- Changes to client's main method

```
// Initialise ORB
ORB orb = ORB.init( args, null );
// Get Name Service reference
org.omg.CORBA.Object objRootContext =
        orb.resolve_initial_references("NameService");
// Narrow it to the NamingContext interface
NamingContext rootContext =
        NamingContextHelper.narrow( objRootContext );
// Create a name as an array of NameComponent objects
NameComponent name[] = new NameComponent[2];
name[0] = new NameComponent("pub", "");
name[1] = new NameComponent("Hello", "");
// Look the object up
org.omg.CORBA.Object o = rootContext.resolve(name);
// Narrow to the Hello interface
Hello h = HelloHelper.narrow( o );
```


4 Hello World Client With Naming Service



4 Hello World Client With Naming Service

- Go to the example directory

```
> cd /proj/i4oods/pub/hello_java_client_nameservice
```

- Compile

```
> /local/ORBacus-4.0.3/bin/jidl --package generated
Hello.idl
> /local/java-1.3/bin/javac -classpath /local/ORBacus-4.0.3/
lib/OB.jar:/local/ORBacus-4.0.3/lib/OBNaming.jar:. client/
HelloClient.java
```

- Run

```
> /local/java-1.3/bin/java -classpath /local/ORBacus-4.0.3/
lib/OB.jar:/local/ORBacus-4.0.3/lib/OBNaming.jar:.
-Dorg.omg.CORBA.ORBClass=com.ooc.CORBA.ORB
-Dorg.omg.CORBA.ORBSingletonClass=com.ooc.CORBA.ORBSingleto
n client.HelloClient -ORBInitRef
NameService=corbaloc::faui40u.informatik.uni-
erlangen.de:4711/NameService
```


5 Summary

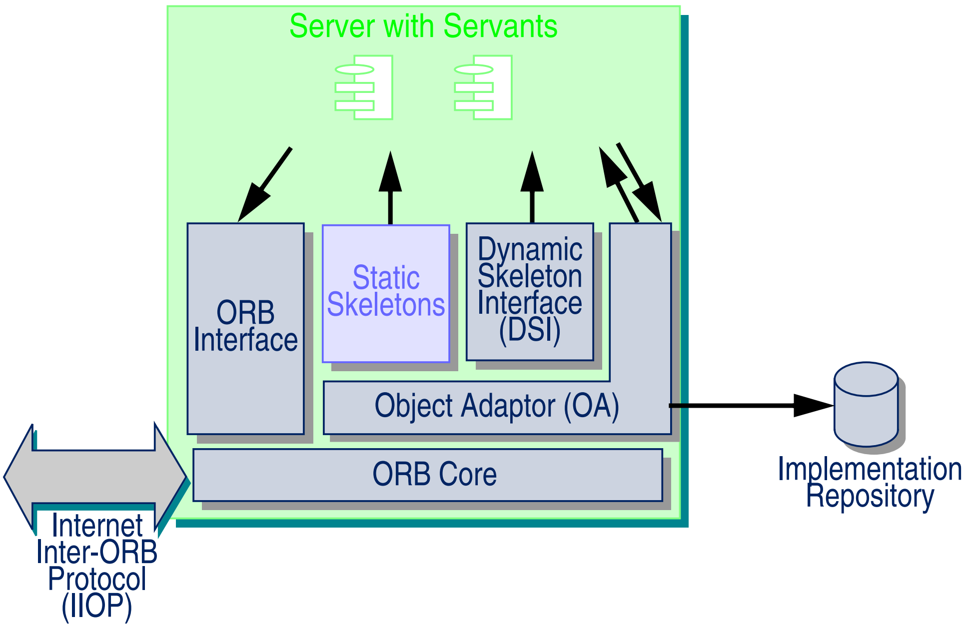
- Bind object references to names
- Naming Contexts are normal CORBA objects (IDL interface)
- Naming Contexts reside in Name Server processes
- Configuration of the Root Naming Context for an application via command line parameter

E.7 Implementing CORBA Objects

1 CORBA Objects revisited

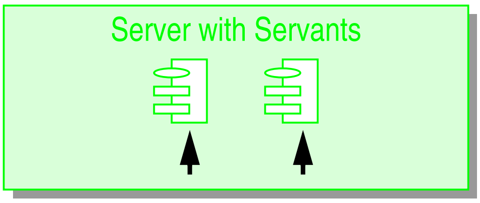
- CORBA objects are accessed via object references
  - ◆ CORBA object conceptually exists when first reference was created
- Functionality of CORBA objects realised by servants
  - ◆ Servants written in a real programming language
  - ◆ Servant may not yet exist when CORBA object is invoked
  - ◆ At most one servant per CORBA object at each moment in time
  - ◆ But many different servants at different times

2 Server Architecture



3 The Server and Servants

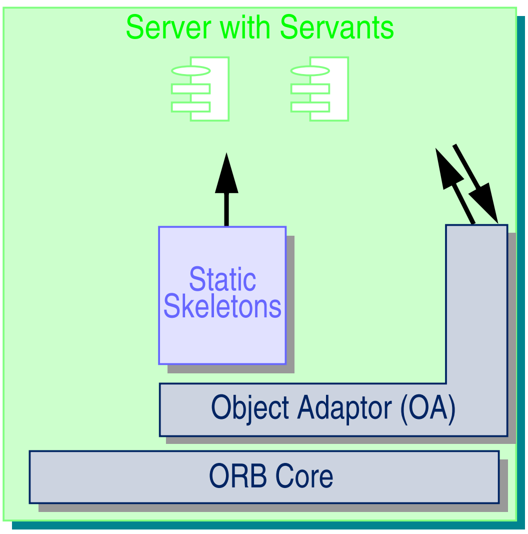
- Server
  - Process that hosts implementations (Servants) of CORBA objects



- Servant
  - Implementation of exactly one CORBA object
  - In OO languages: a special object that implements the IDL interface
  - In non-OO languages: a set of functions that implement the IDL interface and a data structure to identify the instance
  - Many Servants per Server

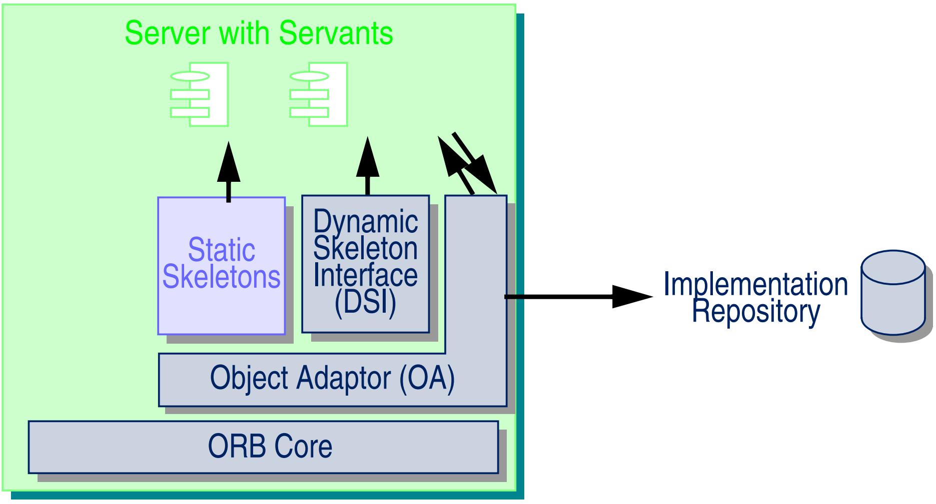
4 Static Skeletons

- Can also be created automatically from the IDL interface
- Demarshalling of invocation parameters, Call dispatching to Servant
- Marshalling of return values or exceptions from the invocation



5 Object Adaptor

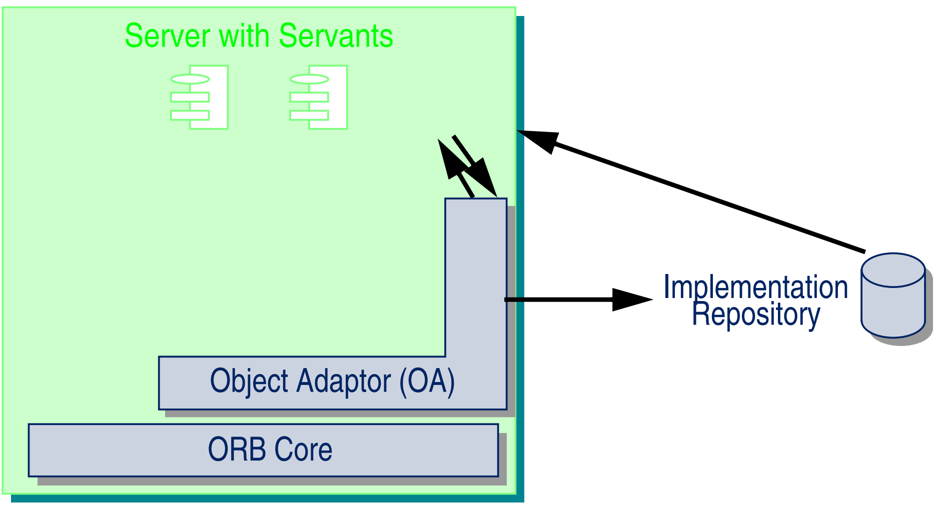
- Call dispatching from ORB Core to the Skeletons
- Creation and management of object references
- Dynamic activation of Servants
- up to CORBA 2.1: Basic Object Adaptor (BOA), now Portable OA (POA)



© Uwe Rastofer • Universität Erlangen-Nürnberg • IMMD IV, 2001

6 Implementation Repository

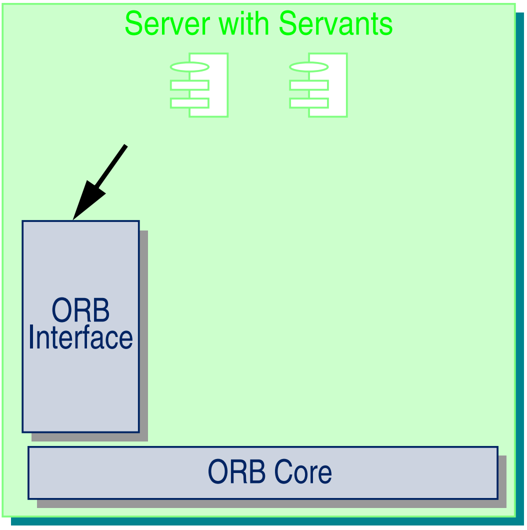
- Database for implementations of CORBA objects
  - ◆ Information about which object is implemented by which Servant
- Often combined with Location Forwarding Service
  - ◆ Dynamically starts Server processes
  - ◆ Forwards calls to Servants in dynamically started Servers



© Uwe Rastofer • Universität Erlangen-Nürnberg • IMMD IV, 2001

7 ORB Interface

- Export of initial object references (ORB, OAs, Naming Service, ...)
- Manipulation of object references (conversion into strings and back)



8 Server Summary

- Servants implement CORBA objects
- Server process hosts Servants
- Skeletons dispatch calls to Servants
- Object Adaptor manages Servant live cycle
- Implementation Repository contains information about active Servers and their Servants

E.8 Simple Java Server (POA)

- Using the Portable Object Adaptor
  - ◆ No need to use the POA interface for now
  - ◆ Use the so-called RootPOA
- Creating and Activating Servants

1 Servant

- Object that implements the functionality of a CORBA object
- Semantics for now:
  - ◆ Each creation of a Servant at the same time creates an associated CORBA object
  - ◆ After destruction of a Servant the associated CORBA object ceases to exists
- IDL:

```
module PortableServer {                              // PIDL
    native Servant;
};
```

- Java Mapping into an abstract class:

```
package org.omg.PortableServer;

public abstract class Servant {
    ...
};
```

2 Skeleton

- Base class for your implementation
- IDL:

```
module module {
    interface name { ... };
};
```

- Java mapping into an abstract class *name***POA**:

```
package module;
public abstract class namePOA
        extends org.omg.PortableServer.Servant
        implements org.omg.CORBA.portable.InvokeHandler,
            nameOperations
{
    public org.omg.CORBA.portable.OutputStream _invoke(
        String op, org.omg.CORBA.portable.InputStream i,
        org.omg.CORBA.portable.ResponseHandler handler)
    { ... }
    public name _this( org.omg.CORBA.ORB orb ) { ... }
    ...
}
```


2 Skeleton

- **`OutputStream _invoke( String op, InputStream i, ... )`**
  - Unmarshals parameters from InputStream
  - Dispatches call using Operation name **`op`**
  - Marshals return values to OutputStream
- *name* **`_this( org.omg.CORBA.ORB orb )`**
  - Activates a new CORBA object
  - Associates the servant with the new CORBA object
  - Returns an object reference (i.e. local stub)
  - Mechanism is also called *Implicit Activation*
- Skeleton automatically generated by the IDL compiler

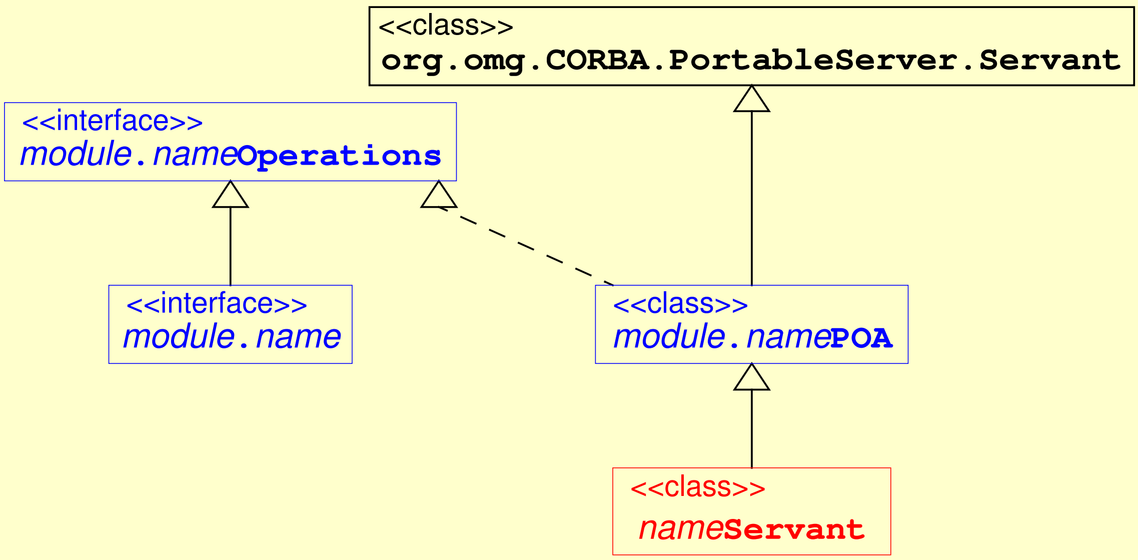
3 Your Servant

- Your implementation of the object's functionality
- Inherits from Skeleton class
- Naming convention: *name***Servant**

```
public class nameServant extends module.namePOA {
      Implementation of methods for attributes and operations
}
```


4 Servant Implementation Hierarchy

- Class hierarchy for IDL interface *module::name*



4 Servant Implementation Hierarchy

- Example: Generated interface and classes

```
public interface AccountOperations {
    public void withdraw( double amount )
        throws AccountPackage.Overdraft;
}
public interface Account extends org.omg.CORBA.Object,
    AccountOperations, ...
{}

public abstract class AccountPOA
    extends org.omg.PortableServer.Servant
    implements AccountOperations, ...
{ ... }
```

- Example: Your Servant

```
public class AccountServant extends AccountPOA {
    public void withdraw( double amount )
        throws AccountPackage.Overdraft {
        // Implementation of withdraw
    }
}
```


5 Missing Pieces

- ORB initialization (see Client section)
- POA activation
  - ◆ Get reference to RootPOA via **resolve_initial_references**

```
org.omg.CORBA.Object o = orb.resolve_initial_references(
                                                  "RootPOA" );
```

  - ◆ Narrow to interface **org.omg.PortableServer.POA**

```
org.omg.PortableServer.POA poa =
                org.omg.PortableServer.POAHelper.narrow( o );
```

  - ◆ Activate POA via **org.omg.PortableServer.POAManager**

```
poa.the_POAManager().activate();
```

- ORB main loop
  - ◆ Start processing requests via **run()** method in **org.omg.CORBA.ORB**

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

6 Hello World Server With Naming Service

- IDL-Interface

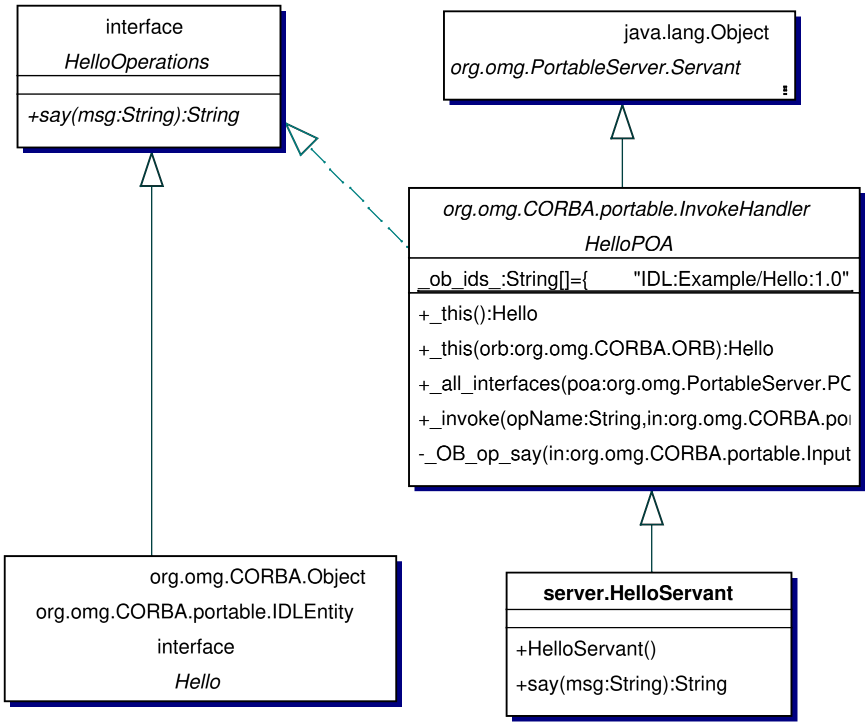
```
// Hello.idl

module Example {
    interface Hello {
        string say( in string msg );
    };
};
```


6 Hello World Server With Naming Service

- Servant implementation via inheritance

```
// server/HelloServant.java

import generated.Example.*;

public class HelloServant extends HelloPOA {

    // Constructor
    public HelloServant() {
        super();
    }

    // Operation Example::Hello::say from IDL
    public String say( String msg ) {
        System.out.println( "say() called" );
        return "Hello" + msg;
    }
}
```

6 Hello World Server With Naming Service



E.8 Simple Java Server (POA)

6 Hello World Server With Naming Service

- Server startup code

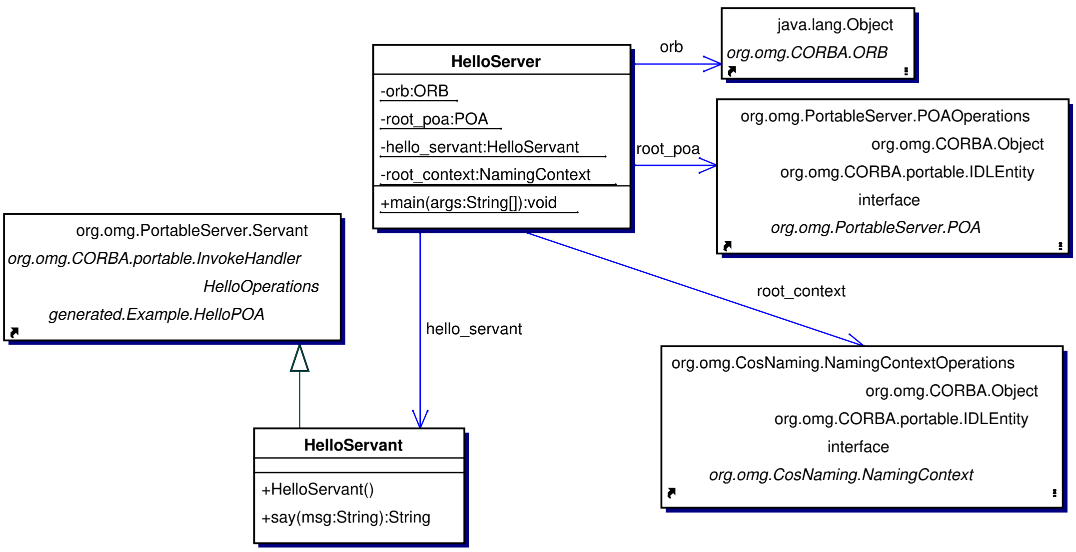
```
// server/HelloServer.java
import generated.Example.*;
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import org.omg.PortableServer.*;

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

            // Get the RootPOA
            POA poa = POAHelper.narrow(
                orb.resolve_initial_references( "RootPOA" ));
            // Activate the RootPOA
            poa.the_POAManager().activate();

            // Create the hello servant object
            HelloServant hServ = new HelloServant();
            // Activate the object
            Hello h = hServ._this( orb );
```

6 Hello World Server With Naming Service

- Server startup code (cont.)

```
            // Get Name Service reference
            org.omg.CORBA.Object obj_root_context =
               orb.resolve_initial_references("NameService");
            // Narrow it to the NamingContext interface
            root_context = NamingContextHelper.narrow(
                obj_root_context );
            // Create a name as an array of NameComponents
            NameComponent name[] = new NameComponent[2];
            name[0] = new NameComponent("pub", "");
            name[1] = new NameComponent("Hello", "");
            // Register the Hello object
            root_context.rebind(name, hello);

            // Wait for requests
            System.out.println( "Hello server is ready" );
            orb.run();
        }
        catch(Throwable t) {
            t.printStackTrace();
        }
    }
}
```


6 Hello World Server With Naming Service



6 Hello World Server With Naming Service

- Go to the example directory

```
> cd /proj/i4oods/pub/hello_java_server_nameservice
```

- Compile

```
> /local/ORBacus-4.0.3/bin/jidl --package generated
Hello.idl
> /local/java-1.3/bin/javac -classpath /local/ORBacus-4.0.3/
lib/OB.jar:/local/ORBacus-4.0.3/lib/OBNaming.jar:. server/
HelloServant.java server/HelloServer.java
```

- Run

```
> /local/java-1.3/bin/java -classpath /local/ORBacus-4.0.3/
lib/OB.jar:/local/ORBacus-4.0.3/lib/OBNaming.jar:.
-Dorg.omg.CORBA.ORBClass=com.ooc.CORBA.ORB
-Dorg.omg.CORBA.ORBSingletonClass=com.ooc.CORBA.ORBSingleto
n server.HelloServer -ORBInitRef
NameService=corbaloc::faui40u.informatik.uni-
erlangen.de:4711/NameService
```


7 Summary

- Initialize ORB
- Activate POA
- Instantiate Servant(s)
- Activate Servant(s)
- Register Servant(s) with Name Service
- Start ORB main loop