Chapitre 7: Architecture CORBA et RIM

D’après le cours de Patrick Pleczon

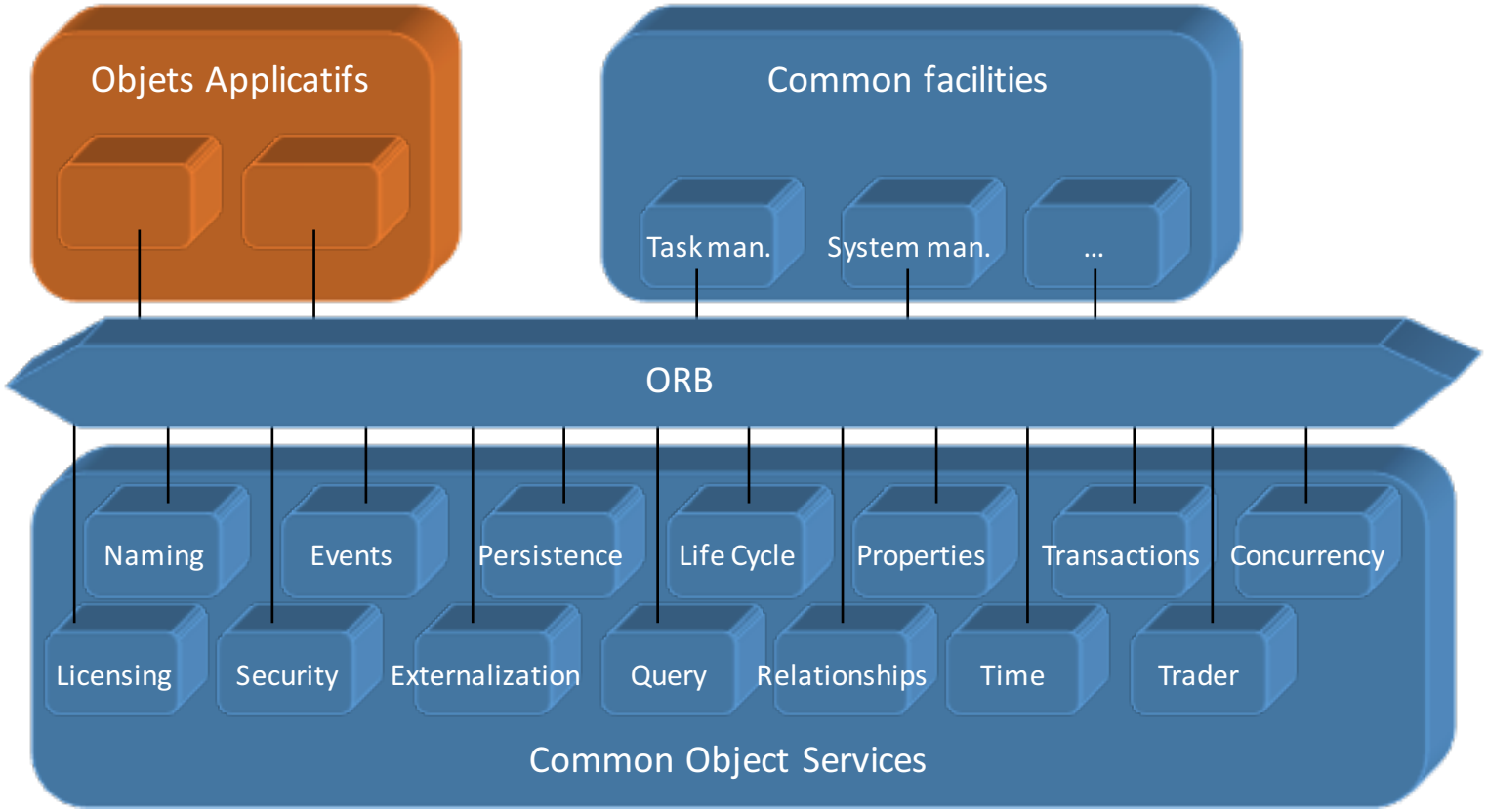
CORBA

- Common Object Request Broker Architecture
  - « Bus » logiciel Normalisé par l'OMG
  - Basé sur un protocole standard (GIOP : General Inter-ORB Protocol)
    - IIOP (Internet IOP) est la version TCP/IP du protocole
  - Un langage de définition d ’interface : IDL
  - Intégration de technologies hétérogènes
    - Java, C++, ADA, SmallTalk, Cobol, …
  - Des services
    - Object services : persistance, nommage, cycle de vie,...
    - CORBA facilities :impression, messagerie,...

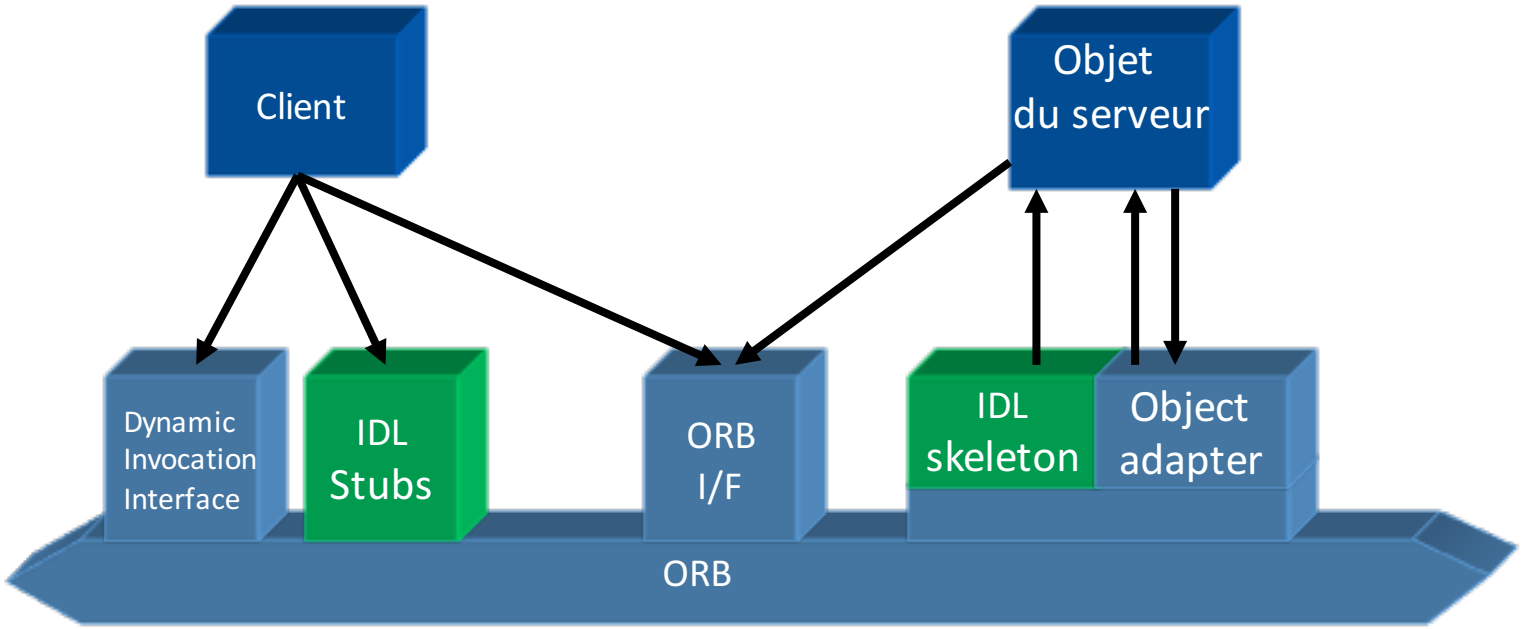
CORBA

- Faciliter la programmation des systèmes distribués:
  - Orienté Objet.
  - Transparence de la localisation des objets.
  - Transparence des accès (local/distant) aux objets.
  - Masquage de l ’hétérogénéité :
    - des OS
    - des langages
- Offrir un standard non propriétaire

CORBA


Objets Applicatifs
Common facilities
Task man.
System man.
...
ORB
Naming
Events
Persistence
Life Cycle
Properties
Transactions
Concurrency
Licensing
Security
Externalization
Query
Relationships
Time
Trader
Common Object Services

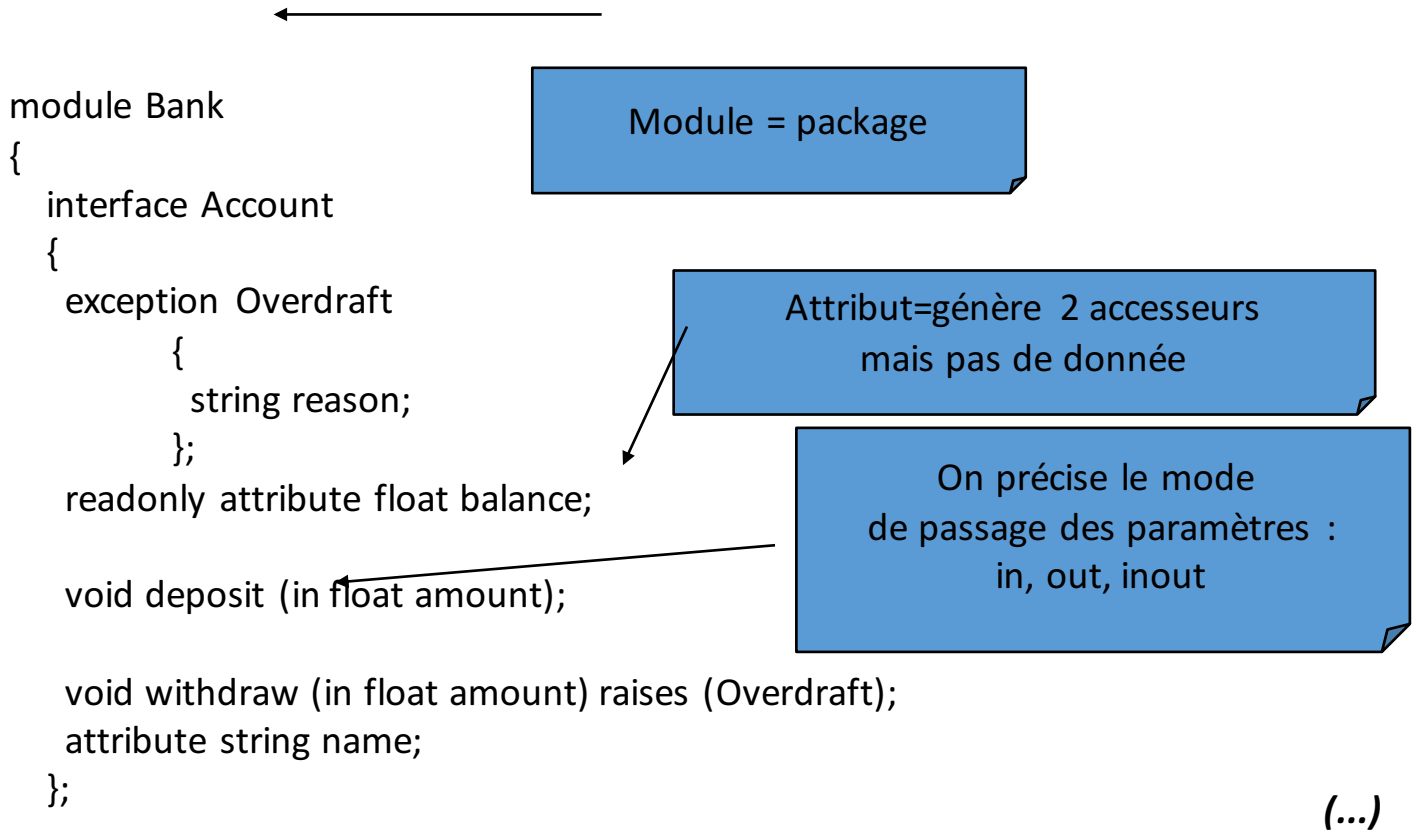
L ’ORB


Client
Objet
du serveur
Dynamic
Invocation
Interface
IDL
Stubs
ORB
I/F
IDL
skeleton
Object
adapter
ORB

Le langage IDL

- Un langage de description d'interfaces indépendant des langages d'implémentation.
- Des compilateurs génèrent le code client et serveur dans différents langages à partir de l'IDL

IDL : exemple


module Bank
{
interface Account
{
exception Overdraft
{
string reason;
};
readonly attribute float balance;
void deposit (in float amount);
void withdraw (in float amount) raises (Overdraft);
attribute string name;
};
Module = package
Attribut=génère 2 accesseurs
mais pas de donnée
On précise le mode
de passage des paramètres :
in, out, inout
(...)

IDL : exemple *(suite)*

```
interface AccountManager
   {
     Account open (in string name,
                   in float initial_balance);

     void close (in Account account);
   };
};
```

On peut renvoyer
une interface

IDL : types de base

- short
- long
- unsigned short
- unsigned long
- float
- double
- char
- boolean
- octet
- any
- string

IDL : héritage

- Héritage simple ou multiple entre interfaces.
  - Exemple

```
interface CompteRemunere : Compte
  {
    attribute short taux;
    float calculeInterets()
  };
```

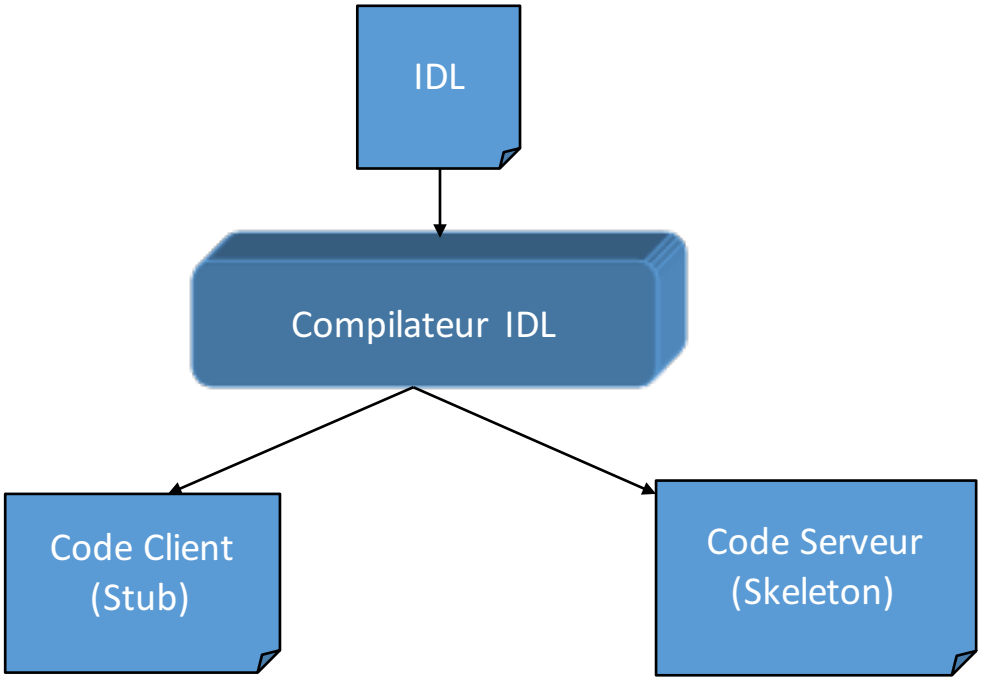
IDL : autres types

- constant
- struct
- enum
- union
- typedef
- sequence
- array

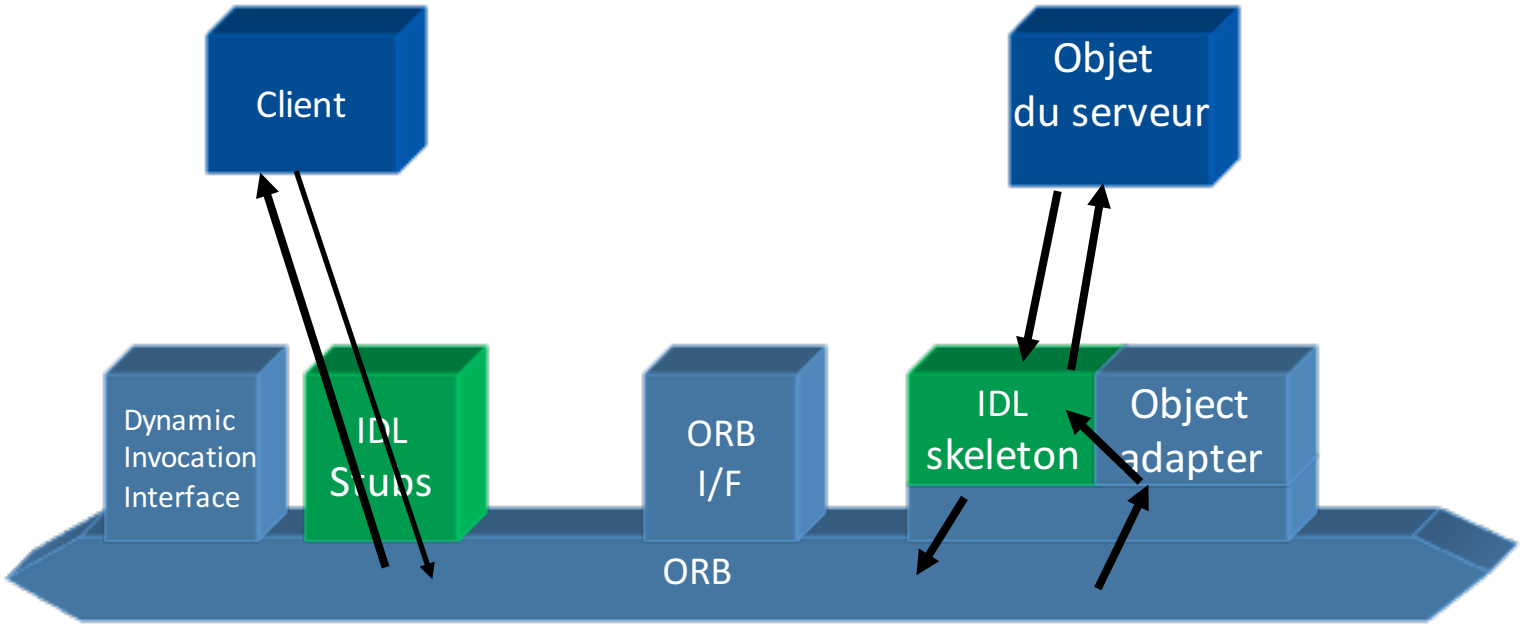
```
struct UserAccess
{
  string name;
  string password;
};
```

```
typedef sequence<Compte>  ListeCompte;
```

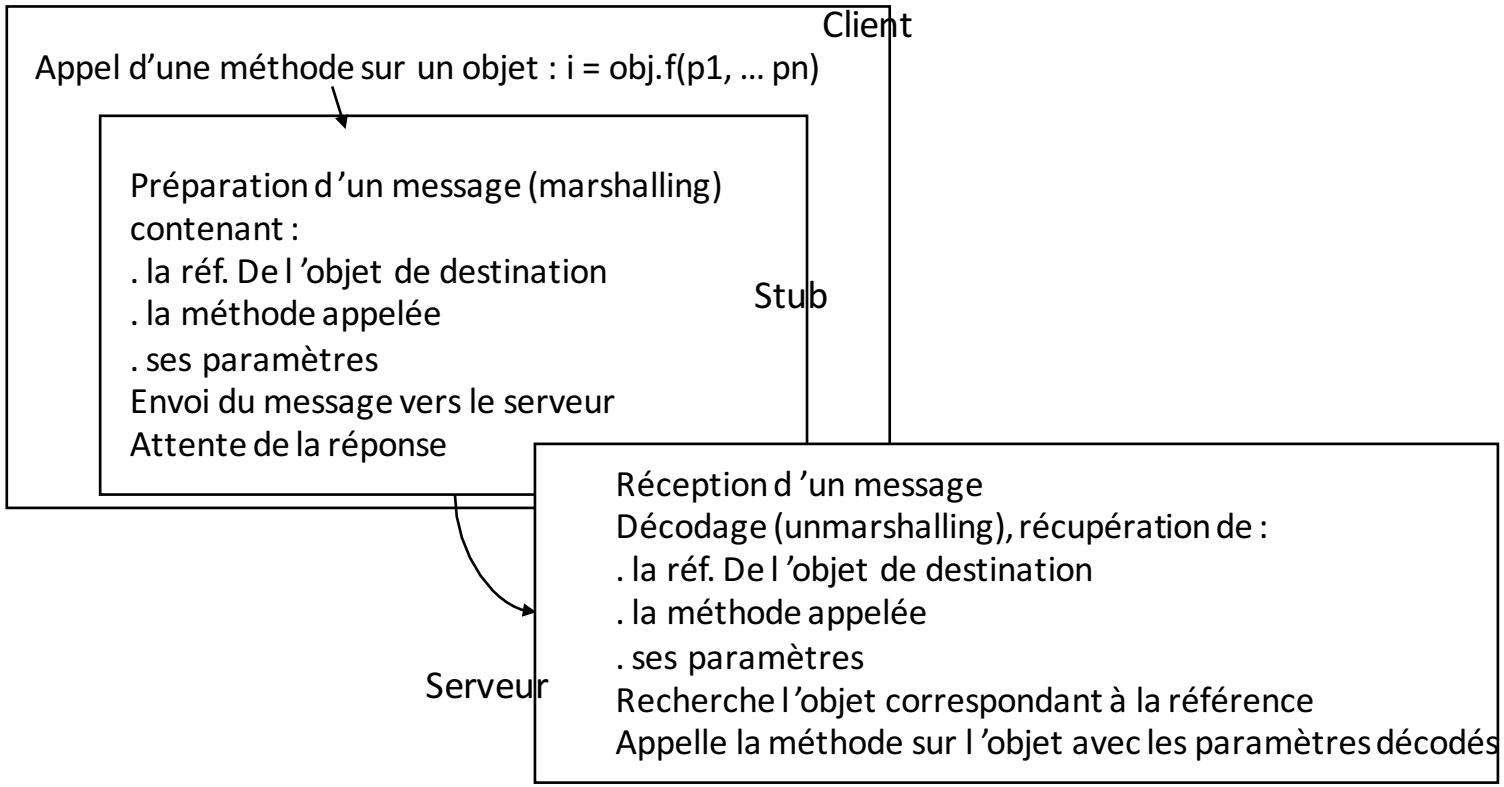
IDL : génération de code



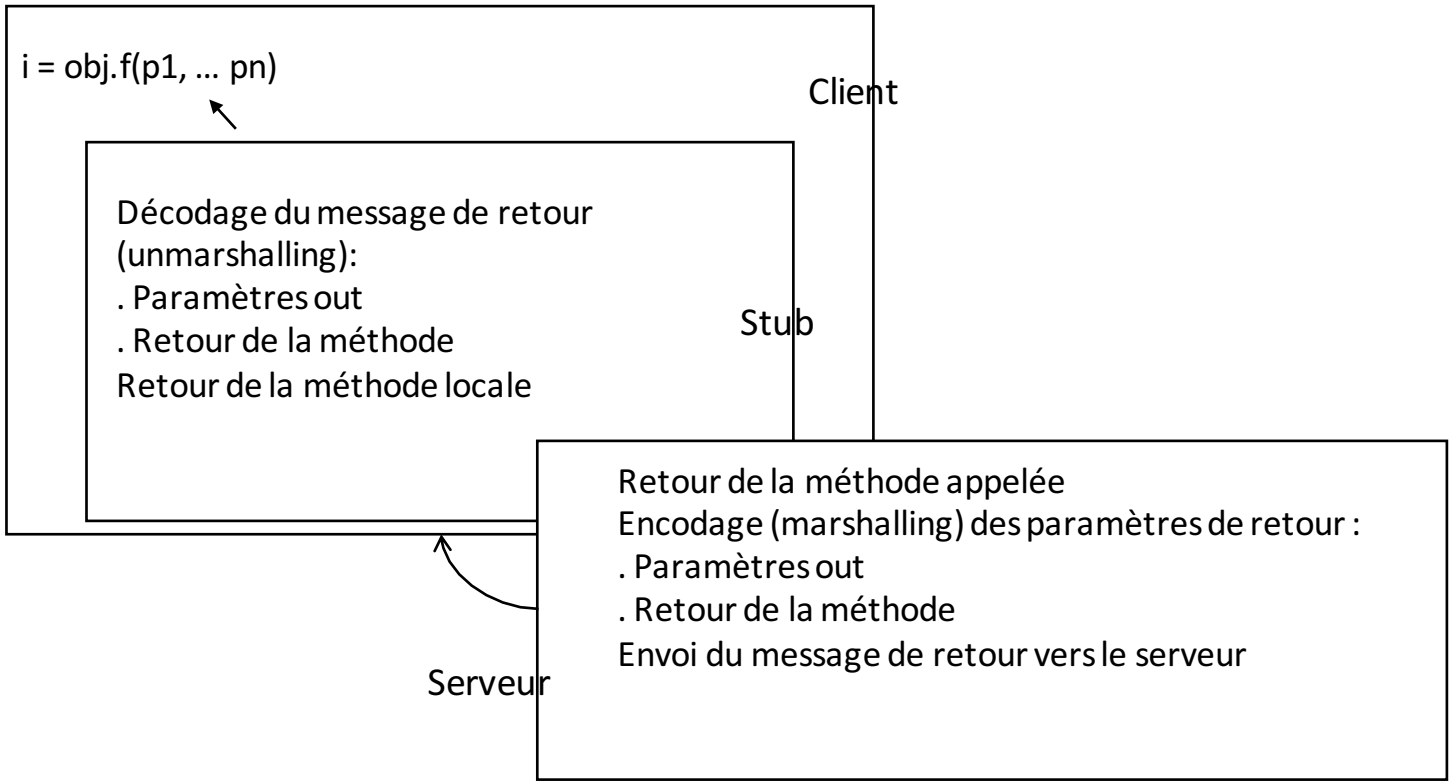
ORB : envoi d'une requête



ORB : envoi d'une requête



ORB : réception de la réponse

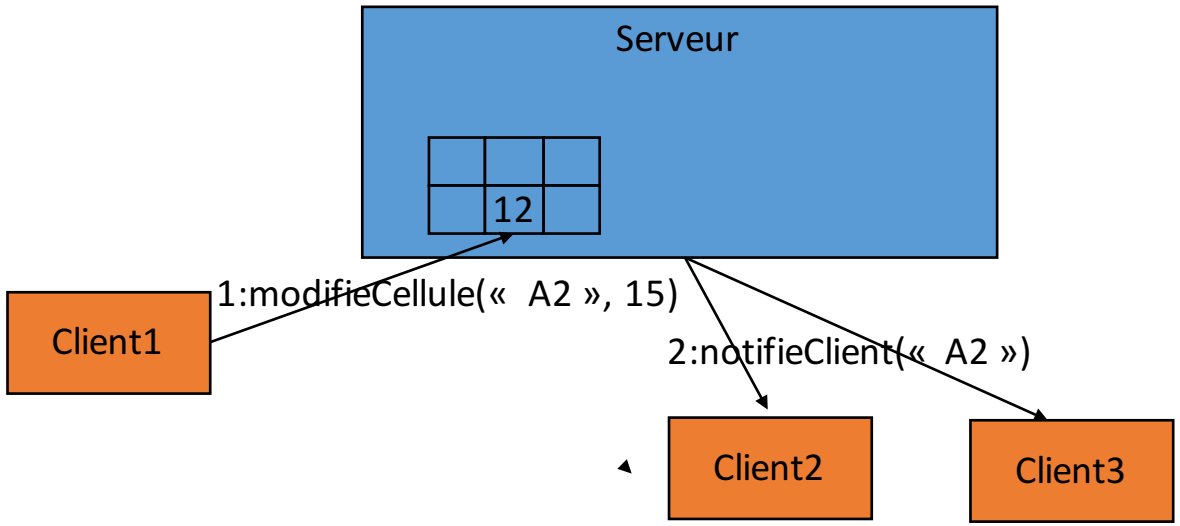


L ’interface d ’invocation dynamique

- Possibilité d ’appeler des méthodes d ’un objet serveur sans stub dans le client :
  - La requête est préparée « à la main » :
    - Définition du nom de la méthode et des paramètres.
  - Appel de la requête et récupération « à la main » des paramètres de retour.

Les callbacks (1)

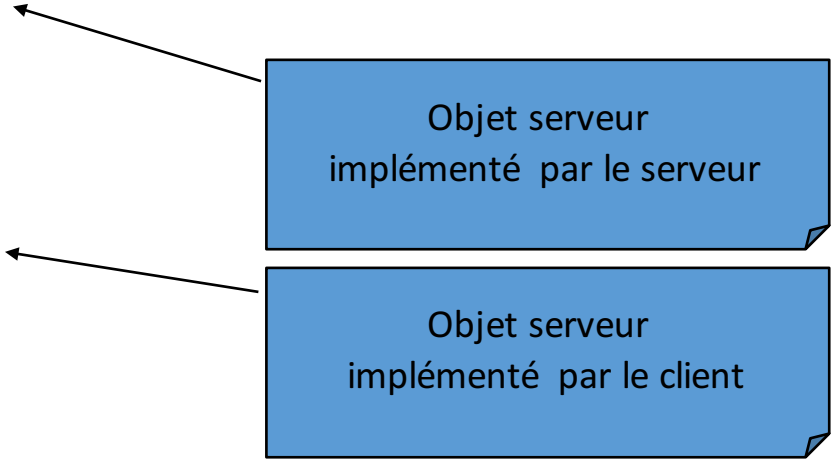
- Permettent à un serveur de notifier un ou des client(s).
- Exemple: tableur distribué:



Les callbacks (2): exemple d'IDL

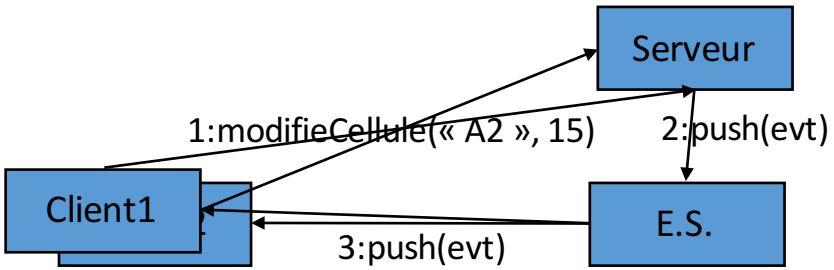
- Interface Tableur
  {
  void mofifieCellule(in string cell, in any valeur);
  void enregistreClient(in ClientTableur clt);
  …
  };

- interface ClientTableur
  {
  void notifieClient(in string cell);
  };



Callback : les services CORBA

- L'utilisation de services CORBA peut simplifier la gestion des callbacks:
  - Service d'événements
    - abonnement à des « topics »
    - push ou pull
  - Service de notification
    - Ajoute des possibilités de filtrage sur le contenu des événements.



CORBA/Java

- Présentation générale
- Process de développement
- Description du service de nommage
- Le compilateur IDL
- Mapping IDL/Java - code généré
- Le POA (Portable Object Adapter)
- Implémentation du serveur par héritage et par délégation
- Implémentation du client
- Execution
- Compléments sur l ’IDL

CORBA/Java - Java IDL

- CORBA/Java : Mapping CORBA IDL vers Java
- JAVA IDL
  - est inclus dans le JDK (depuis 1.2)
  - comprend un compilateur IDL : **idlj**
  - comprend un service de nommage (non persistant) : **tnameserv**
  - un service de nommage plus évolué : **orbd**

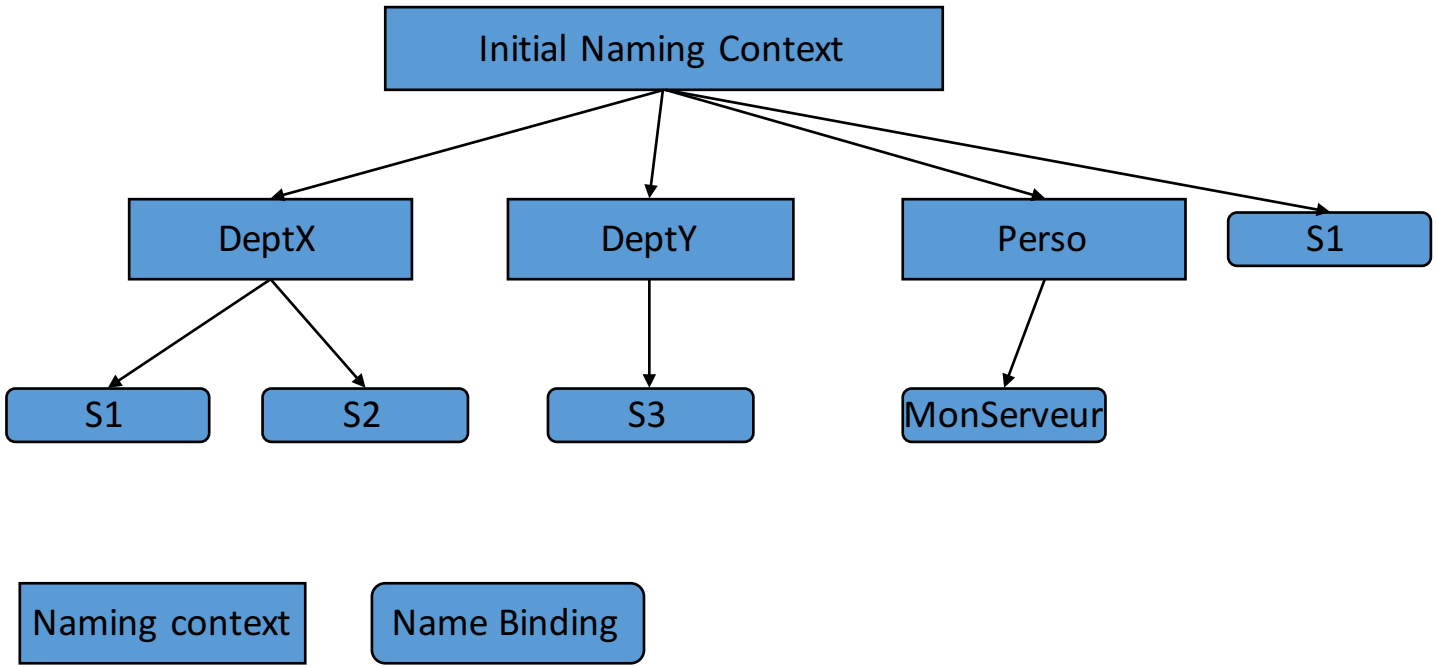
Process de développement

- Définir l ’IDL du serveur.
- Générer le code Java (stub et skeleton) avec le compilateur IDL.
- Implémenter le serveur.
- Implémenter le client.

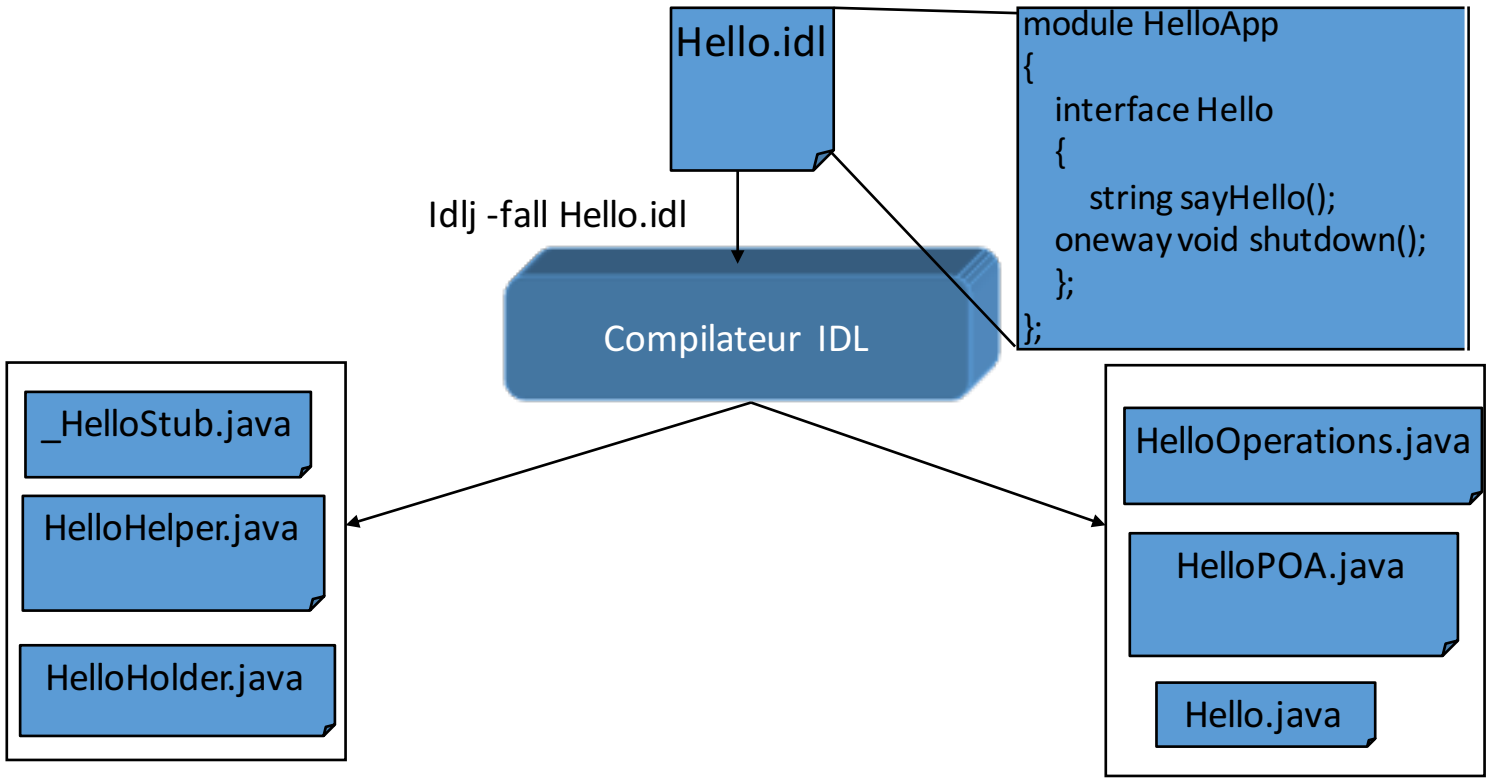
Le service de nommage

- Permet à un client de retrouver un objet serveur à partir de son nom.
- Les objets sont référencés par des « composants de noms » comportants :
  - Un nom d ’enregistrement (ID)
  - Un type d ’objet (kind)
- Le service de nommage offre une structure arborescente de type « directory ».

Le service de nommage *(suite)*



Java IDL - Le compilateur



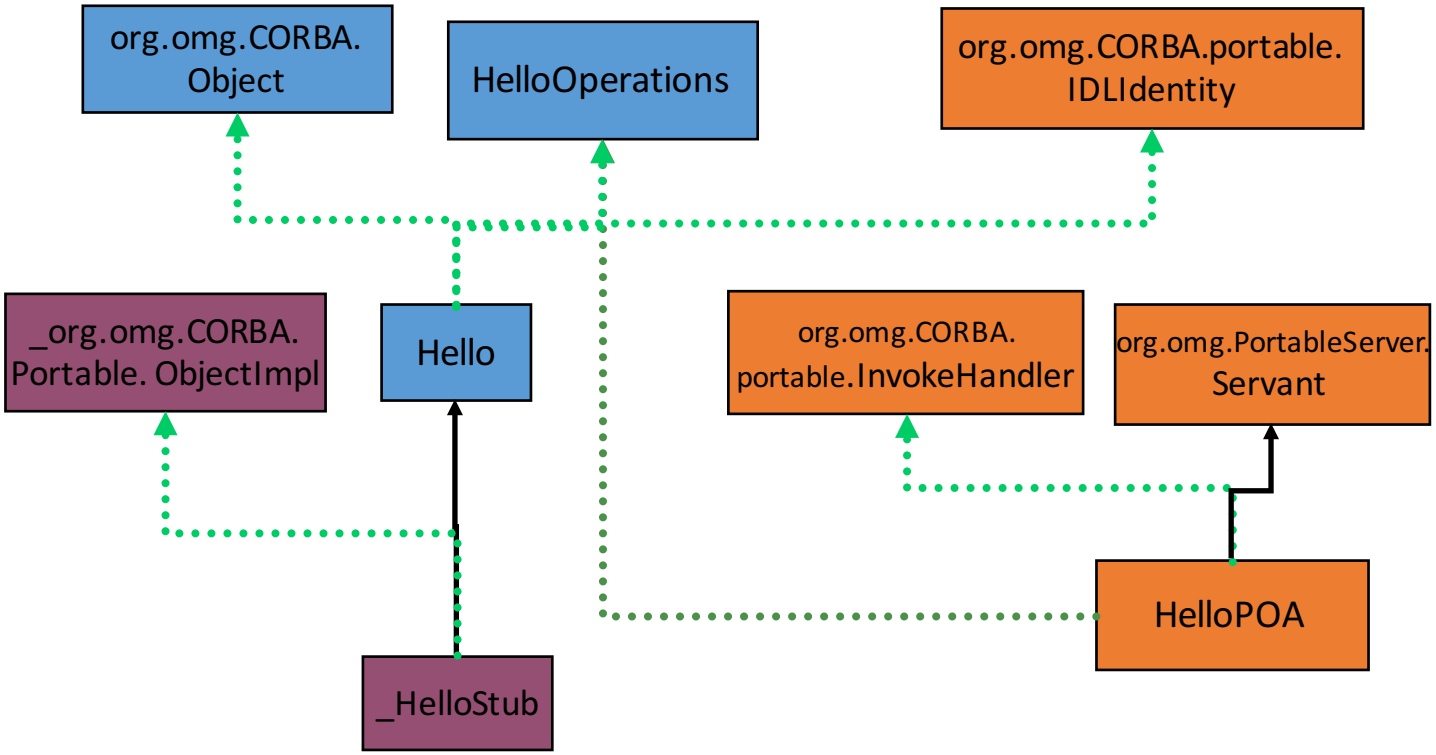
Mapping IDL/Java

IDL	Java
module	package
interface	interface, classe helper, classe holder
constant	public static final
boolean	boolean
char, wchar	char
octet	byte
string, wstring	java.lang.String
short, unsigned short	short
long, unsigned long	int
long long, unsigned long long	long
float	float
double	double
enum, struct, union	class
sequence, array	array
exception	class
readonly attribute	methode d'accès en lecture
readwrite attribute	methods d'accès en lecture et méthode d'accès en écriture
operation	methode

Code généré par Java IDL

- Un module IDL correspond à un package Java
  - Toutes les classes générées correspondant à un module sont dans le même package.
- _HelloStub.java
  - Classe stub de Hello
- HelloHelper.java
  - Classe Helper de Hello : fournit une méthode « narrow » de conversion des références CORBA
- HelloHolder.java
  - Classe holder de Hello pour passage de paramètres out et inout
- HelloOperations.java
  - Interface définissant les opérations.
- HelloPOA.java
  - Classe skeleton de Hello
- Hello.java
  - Interface Hello

Graphe des classes générées



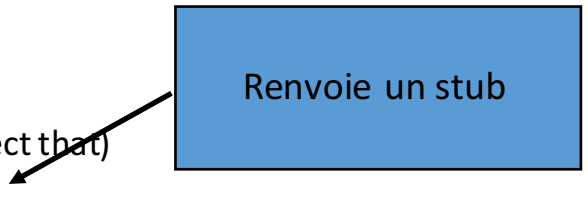
Classes Helper

- Les classes Helper permettent de manipuler les types IDL.
  - Insertion/extraction dans un « Any ».
  - Conversion du type CORBA.Object vers les types dérivés (opérateur **narrow**)
  - Une classe helper est générée pour chaque interface IDL.

(…)

Classes Helper *(suite)*

```
package HelloApp;
public class HelloHelper {
    // It is useless to have instances of this class
    private HelloHelper() { }

   public static void write(org.omg.CORBA.portable.OutputStream out, HelloApp.Hello that) {... }
   public static HelloApp.Hello read(org.omg.CORBA.portable.InputStream in) {...}
  public static HelloApp.Hello extract(org.omg.CORBA.Any a) {...}
  public static void insert(org.omg.CORBA.Any a, HelloApp.Hello that) {...}
  private static org.omg.CORBA.TypeCode _tc;
  synchronized public static org.omg.CORBA.TypeCode type() {...}
  public static String id() {
     return "IDL:HelloApp/Hello:1.0";
  }
  public static HelloApp.Hello narrow(org.omg.CORBA.Object that)
                throws org.omg.CORBA.BAD_PARAM {...}
}
```



Classes Holder

- Java ne supportant que des paramètres « in », les classes Holder permettent le passage de paramètres out ou inout.
- Il existe des classes Holder pour tous les types de base (ex: ShortHolder).
- Une classe Holder est générée pour chaque interface IDL..

(…)

Classes Holder *(suite)*

```
package HelloApp;
public final class HelloHolder
    implements org.omg.CORBA.portable.Streamable{
   //       instance variable
   public HelloApp.Hello value;
   //       constructors
   public HelloHolder() {
            this(null);
   }
   public HelloHolder(HelloApp.Hello __arg) {
            value = __arg;
   }
   public void _write(org.omg.CORBA.portable.OutputStream out) {...}
   public void _read(org.omg.CORBA.portable.InputStream in) {...}
   public org.omg.CORBA.TypeCode _type() {...}
}
```

Le POA (Portable Object Adapter)

- Object Adapter:
  - Mécanisme qui permet de relier une référence objet utilisée dans une requête avec le code d'exécution.
- POA: défini dans la spécification CORBA.
  - Assure la portabilité des Implémentations entre ORB.
  - Permet la persistance d'identités.
  - Permet la gestion du cycle de vie des objets:
    - Création, activation, passivation, suppression.

Implémentation par héritage des classes du serveur

- La classe d'implémentation dérive de la classe *Type*POA :
  - Ex : class HelloServant extends HelloPOA
- Elle implémente les méthodes de l 'IDL.

Implémentation du serveur

```
// HelloServer.java
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import org.omg.PortableServer.POA;
import java.util.Properties;
class HelloImpl extends HelloPOA {
        private ORB orb;
        public void setORB(ORB orb_val) { orb = orb_val; }
        public String sayHello() { return "\nHello world !!\n"; }
        public void shutdown() { orb.shutdown(false); }
}
```

(...)

Implémentation du serveur *(suite)*

```
public class HelloServer
{
  public static void main(String args[])
  {
  try{
            // initialisation de l'ORB et création de l'objet servant
            ORB orb = ORB.init(args, null);

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

            rootpoa.the_POAManager().activate();

            HelloImpl helloImpl = new HelloImpl();

            helloImpl.setORB(orb);
```

Implémentation du serveur *(suite)*

```
            // enregistrement de l'objet dans le naming service
            org.omg.CORBA.Object ref = rootpoa.servant_to_reference(helloImpl);
            Hello href = HelloHelper.narrow(ref);

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

            NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

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

            System.out.println("HelloServer ready and waiting ...");
            orb.run();
        } catch (Exception e) { System.err.println("ERROR: " + e); e.printStackTrace(System.out); }
        System.out.println("HelloServer Exiting ...");
    }
}
```

Implémentation du client

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

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

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

(...)

Implémentation du client *(suite)*

```
// resolve the Object Reference in Naming
        NameComponent nc = new NameComponent("Hello", "");
        NameComponent path[] = {nc};
        Hello helloRef = HelloHelper.narrow(ncRef.resolve(path))

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

      } catch (Exception e) {
        System.out.println("ERROR : " + e) ;
        e.printStackTrace(System.out);
      }
   }
}
```

Execution

- Lancer le service de nommage:
  - tnameserv ou ordb
- Lancer le serveur
- Lancer le client

Compléments sur l ’IDL : les structures

- Exemple IDL :

```
struct PersonStruct
{
  string nom;
  string prenom;
  short age;
}
```

- Code Java généré

```
final public class PersonStruct {
        // instance variables
        public String nom;
        public String prenom;
        public short age;
        // constructors
        public PersonStruct () {}
        public PersonStruct (String nom, String prenom, short age)
                {...}
        }
final public class PersonStructHolder ...
```

Compléments sur l'IDL : les séquences

- Exemple IDL :
  typedef sequence<PersonStruct> PersonSeq;
- Code Java généré
  La séquence est implémentée par un tableau.
  Une classe holder est générée par type de séquence.

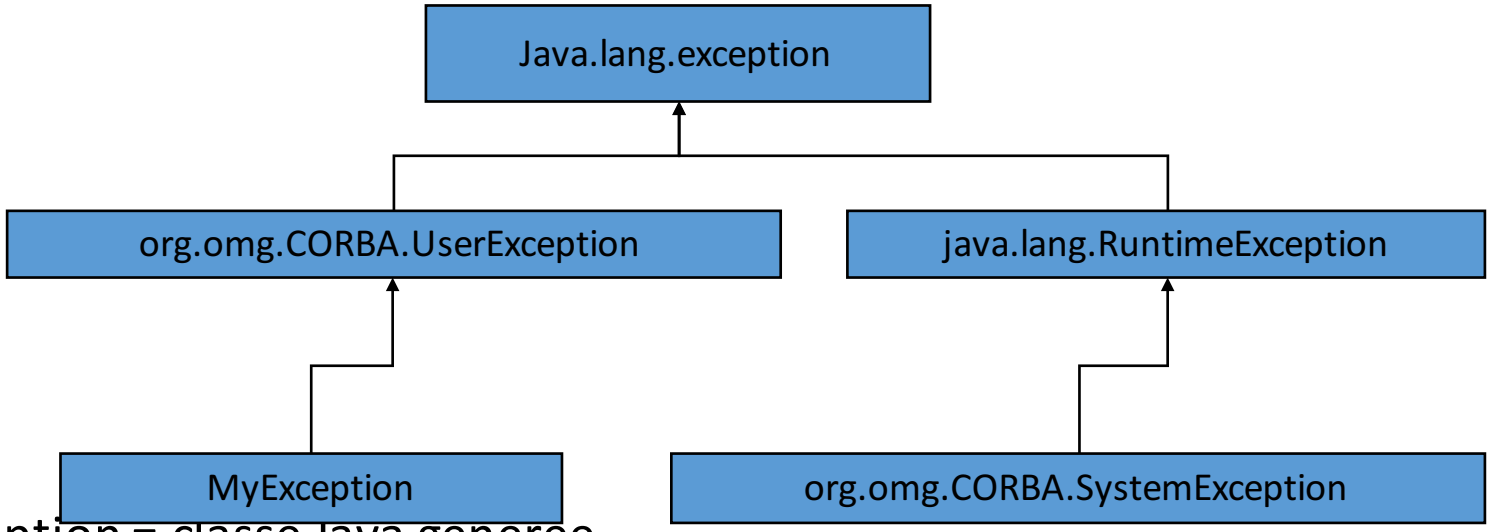
```
final public class PersonSeqHolder {
        public PersonStruct[] value;

        public PersonSeqHolder() {};
        public PersonSeqHolder(PersonStruct[] initial) {...};
        public void _read(org.omg.CORBA.portable.InputStream i)
                {...}
        public void _write(org.omg.CORBA.portable.OutputStream o)
                {...}
        public org.omg.CORBA.TypeCode _type() {...}
}
```

Compléments sur l'IDL : le type Any

- Le type Any permet de contenir une donnée de type IDL quelconque.
- Lorsqu'une donnée est insérée dans un Any, le type de la donnée est inséré dans le champ typeCode (identification du type IDL).
  - void insert_XXX(XXX x)
- L'extraction de la donnée renvoie une exception si le type extrait ne correspond pas au typeCode de l'any.
  - XXX extract_XXX() throws BAD_OPERATION
- Les anys sont créés par le singleton ORB
  - org.omg.CORBA.Any any = orb.create_any();

Compléments sur l ’IDL : les exceptions

- 2 types d ’exceptions : User et System.



- User Exception = classe Java generee

Java RMI

- Présentation générale
- Principes de communication
- Process de développement avec RMI
- Execution
- Chargement des stubs/Sécurité
- Exemple

RMI

- Remote Method Invocation
  - Solution 100% Java client et serveur
  - Protocole basé sur IIOP (RMI-IIOP)
    - Possibilité de choisir un protocole de transport (RMIClientSocketFactory)
  - Passage d'objets par valeur
    - Utilise la sérialisation
  - Peu de services
    - Nommage
    - Support de persistance et d'activation
  - Non standard
  - Modèle de programmation transparent
  - Contient un Distributed Garbage Collector (DGC)

RMI : communication

- Utilisation de stubs et de skeletons générés comme pour CORBA.

Process de développement avec RMI

- Définir l ' interface distante et la dériver de l 'interface **java.rmi.Remote.**
  - Chaque méthode doit déclarer l 'exception **java.rmi.RemoteException**.
- Implémenter l 'interface dans une classe dérivée de **java.rmi.UnicastRemoteObject**.
- Compiler la classe serveur.
- Générer les stub et skeleton avec **rmic**.
  - rmic [options] myserverClass
- Ecrire le code du client.
- Compiler le client.

Execution avec RMI

- Lancer le serveur d ’enregistrement RMI.
  - rmiregistry
- Démarrer le serveur.
  - Le serveur doit :
    - Créer un (des) instances d ’objets RMI.
    - s ’enregistrer auprès du RMI registry avec la classe **java.rmi.Naming**.
- Démarrer le client
  - Le client doit se connecter à l ’objet serveur en utilisant le RMI registry.

Chargement des stubs/Sécurité

- Les stubs des objets distants peuvent être chargés du serveur.
  - Lors de la connection du client sur le serveur, le serveur indique au client l'URL à utiliser pour charger le stub.
  - Nécessite un loader de classes particulier (RMIClassLoader) et un manager de sécurité particulier (RMISecurityManager)

Exemple RMI : interface remote

```
package examples.hello;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Hello extends Remote {
   String sayHello() throws RemoteException;
}
```

Exemple de serveur RMI

```
import java.rmi.Remote;
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.RMISecurityManager;
import java.rmi.server.UnicastRemoteObject;

public class HelloImpl extends UnicastRemoteObject implements Hello {

        public HelloImpl() throws RemoteException {
           super();
        }
        public String sayHello() {
           return "Hello World!";
        }
```

(…)

Exemple de serveur RMI *(suite)*

```
public static void main(String args[]) {
        // Create and install a security manager
        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new RMISecurityManager());
        }
        try {
            HelloImpl obj = new HelloImpl();

            // Bind this object instance to the name "HelloServer"
            Naming.rebind("//serverHost/HelloServer", obj);

            System.out.println("HelloServer bound in registry");
        } catch (Exception e) {
            System.out.println("HelloImpl err: " + e.getMessage());
            e.printStackTrace();
        } …
```

Exemple de client RMI

```
// Applet HelloApplet
…
Hello obj = null;
…
        public void init() {
            try {
                obj = (Hello)Naming.lookup("//" +
                    getCodeBase().getHost() + "/HelloServer");
                message = obj.sayHello();
            } catch (Exception e) {
                System.out.println("HelloApplet exception: " + e.getMessage());
                e.printStackTrace();
            }
        }
…
```

Lancement du serveur

- Au lancement du serveur, il faut préciser:
  - L ’emplacement des classes stub.
  - L ’emplacement du fichier de sécurité.
    - java -classpath "C:\myprojects\proj1\myclasses"
      -Djava.rmi.server.codebase="file:\myprojects\proj1\myclasses /"
      -Djava.security.policy="C:\myprojects\proj1\TradeServer\policy.txt"
      HelloServer

  - exemple de fichier de sécutité:

```
grant {
    // Allow everything
    permission java.security.AllPermission;
};
```

Prise en compte des firewalls

- Un client RMI essaie de se connecter au serveur par un socket.
  Si la connexion est refusée, il tentera automatiquement d'envoyer la requête par HTTP (post).
- Si le serveur RMI reçoit une requête HTTP post, il répond automatiquement avec le protocole HTTP.
- Ceci est complètement transparent (aucune configuration particulière nécessaire).

Comparaison CORBA/RMI

Protocole	IIOP	RMP ou IIOP
Indépendance/langage	oui	non
Passage d'objets par valeur	CORBA 3.0	oui
Passage de paramètres	in, out, inout	in
Performances	+	-
Facilité de mise en œuvre	-	+
Gestion de transactions	par CORBA OTS	non
Services	+	-