

**Licence en génie logiciel et systèmes d'information**  
**Niveau: 3<sup>ème</sup> année**  
**Semestre 1**

**Chapitre 5:**  
**Le middleware CORBA**

**Dhikra KCHAOU**  
**dhikrafsegs@gmail.com**

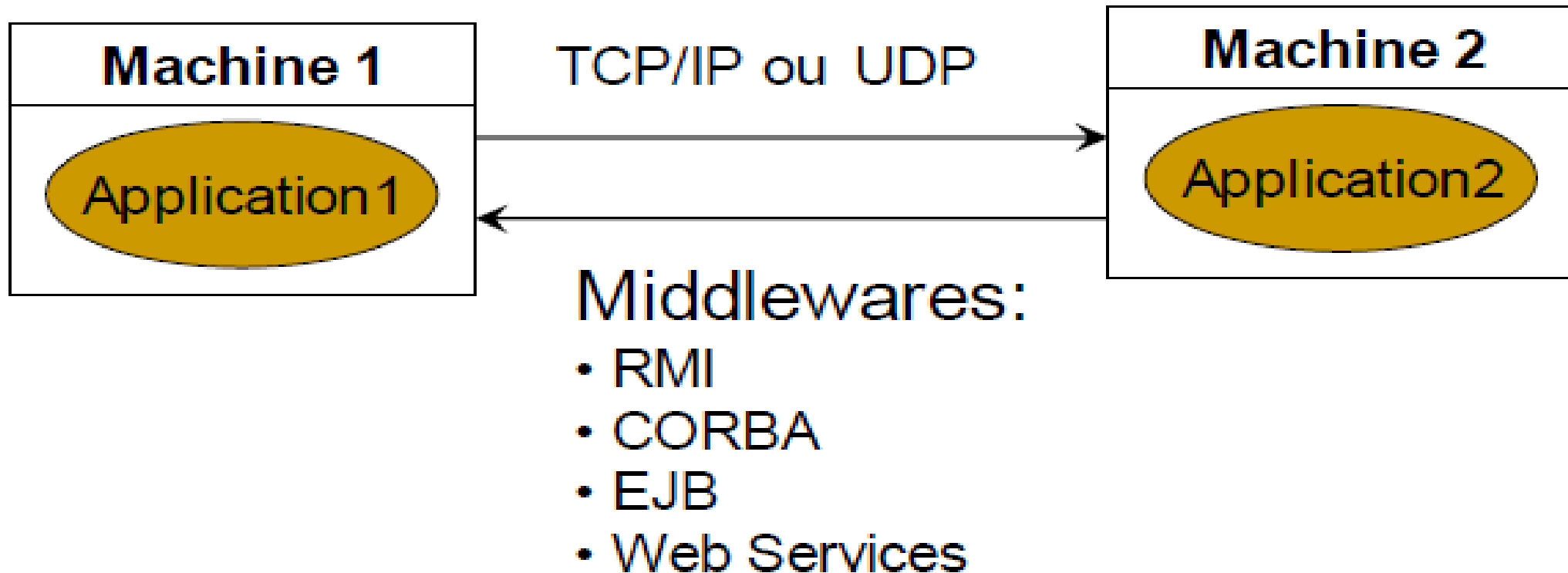
# Plan du chapitre

---

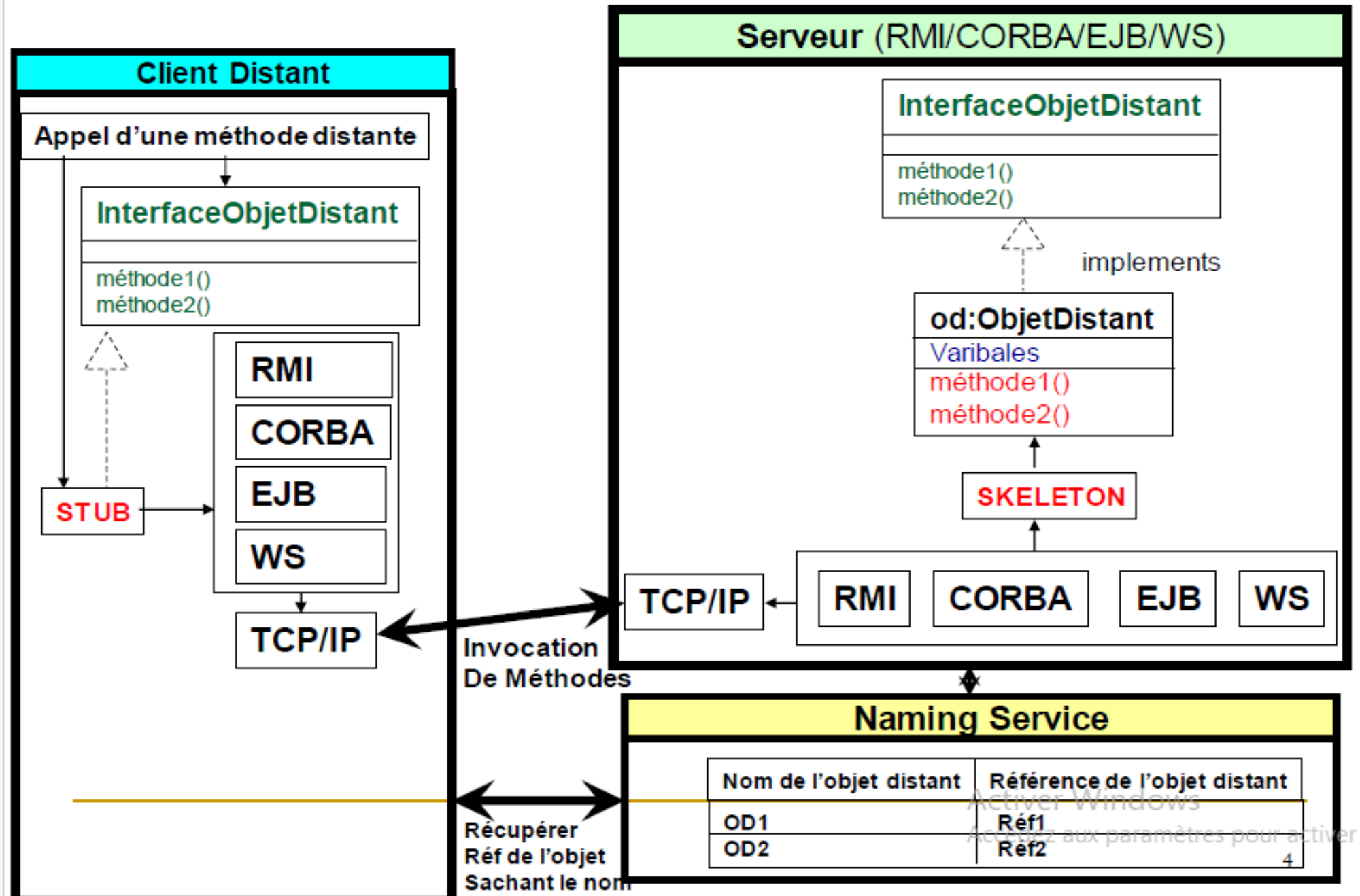
1. Rappel
2. Qu'est ce que C.O.R.B.A. ?
3. Les amorces CORBA
4. Protocoles de communication CORBA
5. Le Langage IDL
6. Démarche à suivre pour développer un projet CORBA
7. Exemple d'application

# Rappel

---



# Rappel



# Qu'est ce que l'O.M.G. ?

- ▶ O.M.G. ( Object Management Group ) est un consortium qui regroupe plus 800 entreprises du monde entier.
- ▶ Consortium ouvert aux horizons autres que les concepteurs de logiciels ( industriels, chercheurs, université, etc... ).
- ▶ Ce consortium définit des spécifications pour fournir un modèle de coopération entre objets répartis.



BPM+  
Business Process Management  
Plus



CORBA®  
Common Object Request Broker  
Architecture™



DDS™  
Data-Distribution Service for  
Real-Time Systems™



IEF™  
The Information Exchange  
Framework™



MOF™  
MetaObject Facility  
Specification™



SYSML®  
Systems Modeling Language™



UAF®  
Unified Architecture  
Framework®



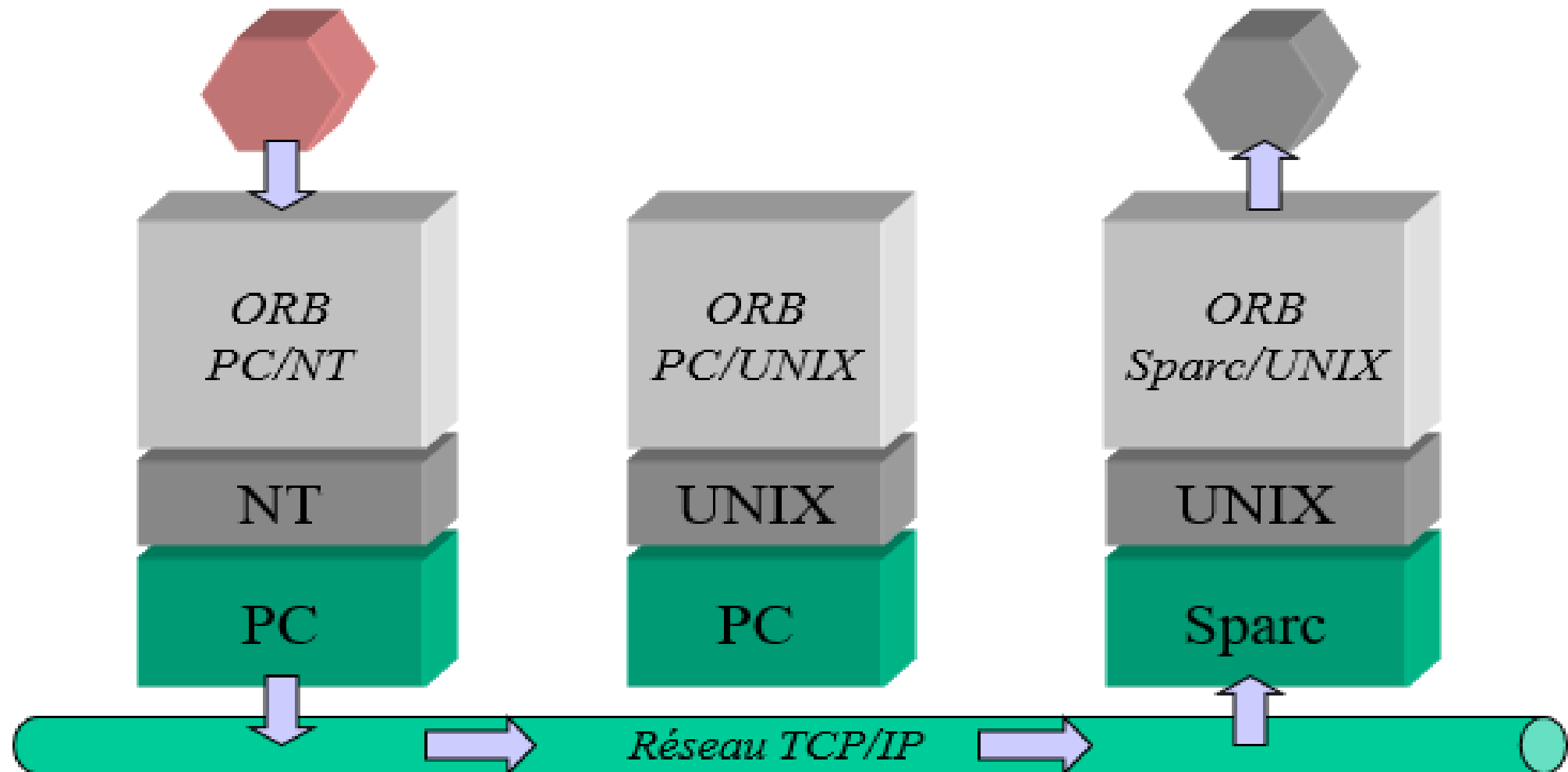
UML®  
Unified Modeling Language™  
Activer Windows  
Accédez aux paramètres pour activer

# Qu'est ce que C.O.R.B.A. ?

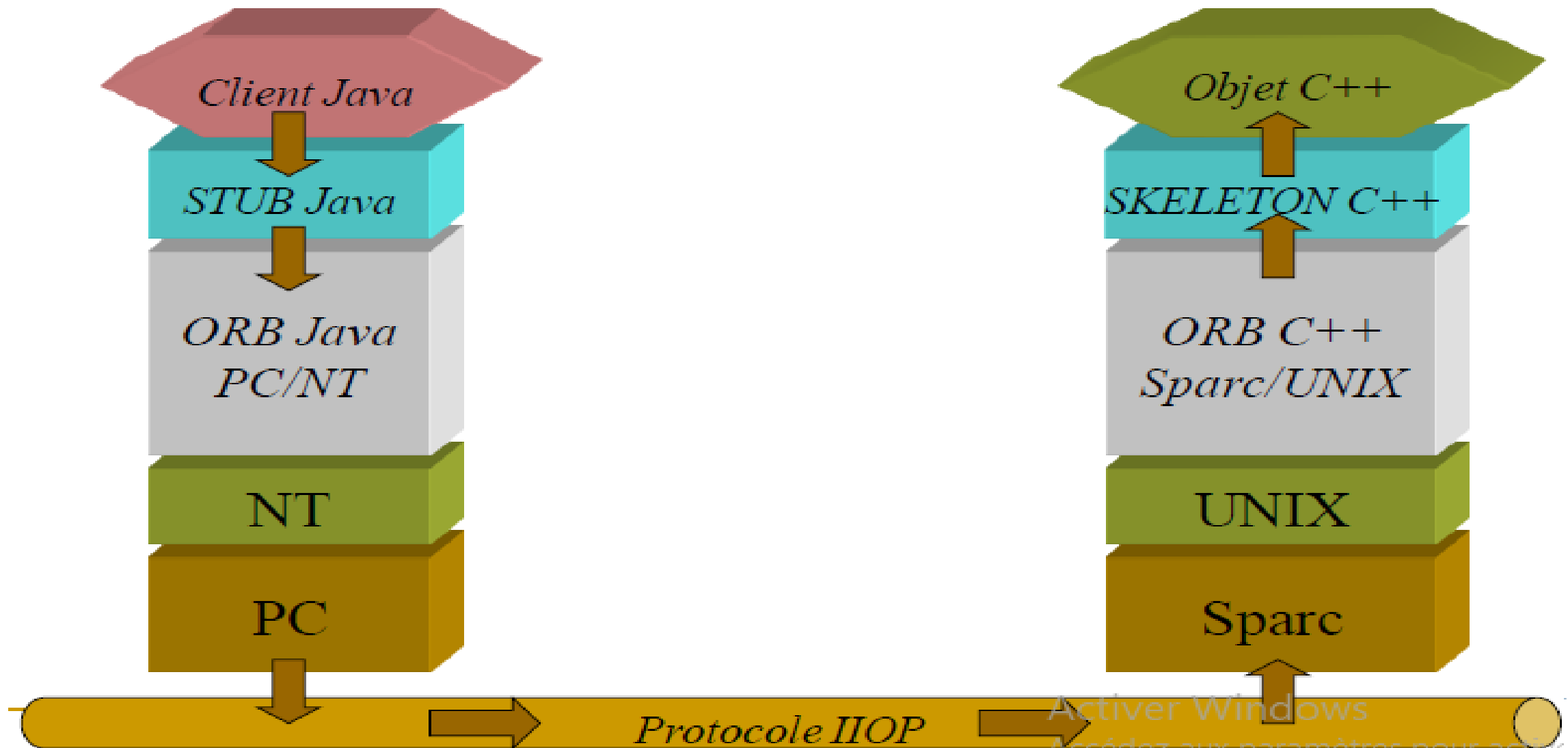
---

- ▶ CORBA ( **C**ommon **O**bject **R**equest **B**roker **A**rchitecture )
- ▶ C'est un **Middleware** qui permet de créer des applications réparties multi langages et multiplateforme.
- ▶ CORBA est standardisé par le groupe OMG (Object Management Group).
  - ▶ première spécification : 1991
  - ▶ seconde version majeure : 1995
  - ▶ troisième version majeure : 2002
  - Dernière version: la version 3.4 en février 2021 <https://www.corba.org/>

# Environnement du middleware CORBA



# Les amorces CORBA





# Protocoles de communication CORBA

---

- ▶ Les participants à un échange CORBA communiquent à l'aide d'un protocole spécifique à CORBA : **IIOP ( Internet Inter-ORB Protocol )**.
- ▶ Le protocole IIOP est indépendant du langage de programmation, du système d'exploitation et de la machine utilisée.
- ▶ Protocoles d'interopérabilité entre ORBs conformes à CORBA 2:
  - ▶ **GIOP : General Inter-ORB Protocol**
    - ▶ Messages : request, reply, cancelrequest, ...
    - ▶ nécessite un protocole de transport fiable,
  - ▶ **IIOP (Internet IOP) : instantiation de GIOP sur TCP**

# Serveurs et Objets

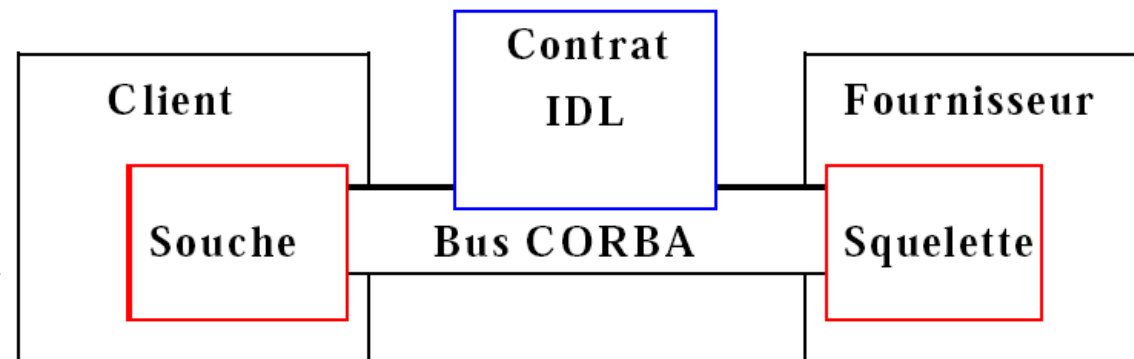
---

- ▶ Un serveur CORBA peut héberger plusieurs objets CORBA.
- ▶ Chaque objet est accessible indépendamment des autres objets du serveur.
- ▶ Chaque objet exprime son offre de services.
- ▶ Pour chaque objet, CORBA propose la définition d'une interface écrite avec un langage de description de services appelée **Interface Definition Language IDL**.

# Le Langage IDL

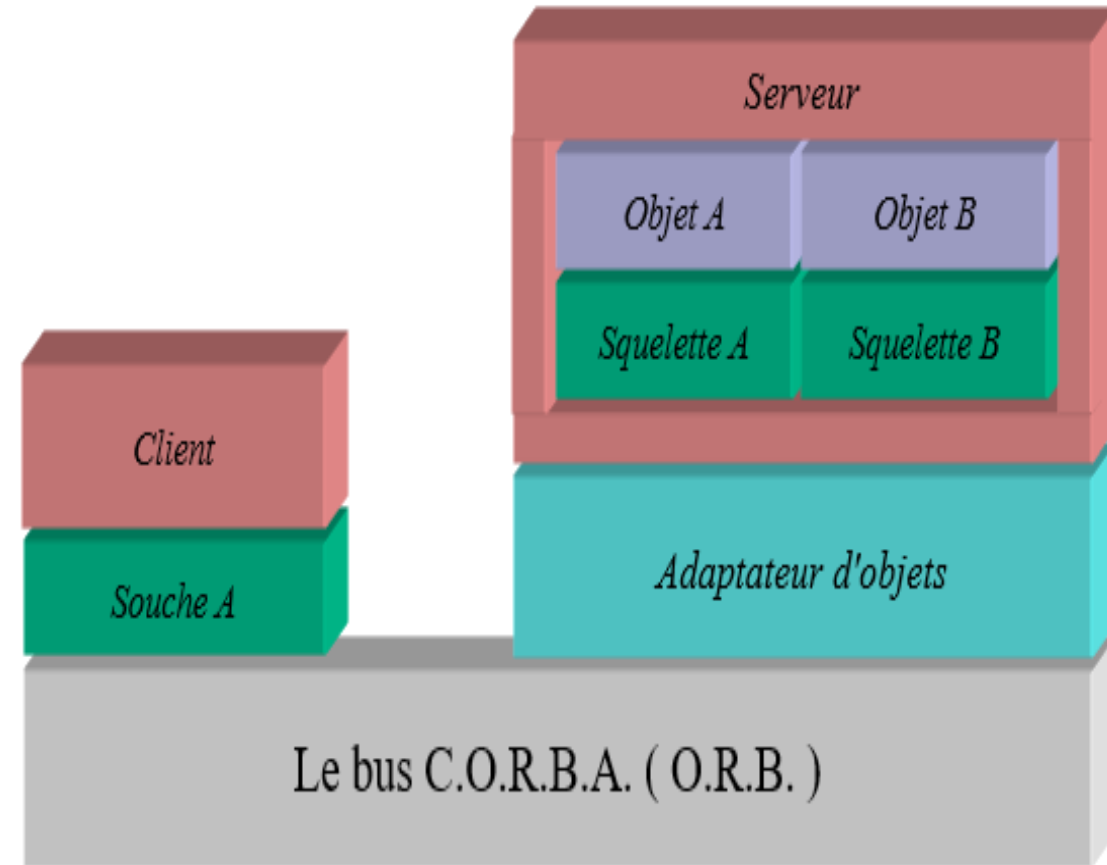
---

- ▶ Le langage OMG-IDL permet d'exprimer, sous la forme de **contrats IDL**, la coopération entre les fournisseurs et les utilisateurs de services, en séparant l'interface de l'implémentation des objets
- ▶ Un contrat IDL spécifie les types manipulés par un ensemble d'applications réparties,
- ▶ Le contrat IDL isole ainsi les clients et fournisseurs de l'infrastructure logicielle et matérielle les mettant en relation à travers **le bus CORBA**.



# L'adaptateur d'objets (POA)

- ▶ Fonctions du POA:
  - ▶ Interface entre les objets CORBA et l'ORB
  - ▶ Enregistrement et recherche des implantations d'objets
  - ▶ Génération de références pour les objets
  - ▶ Gestion de l'instanciation des objets serveurs
  - ▶ Activation des processus dans le serveur
  - ▶ Aiguillage des invocations de méthodes vers les objets serveurs

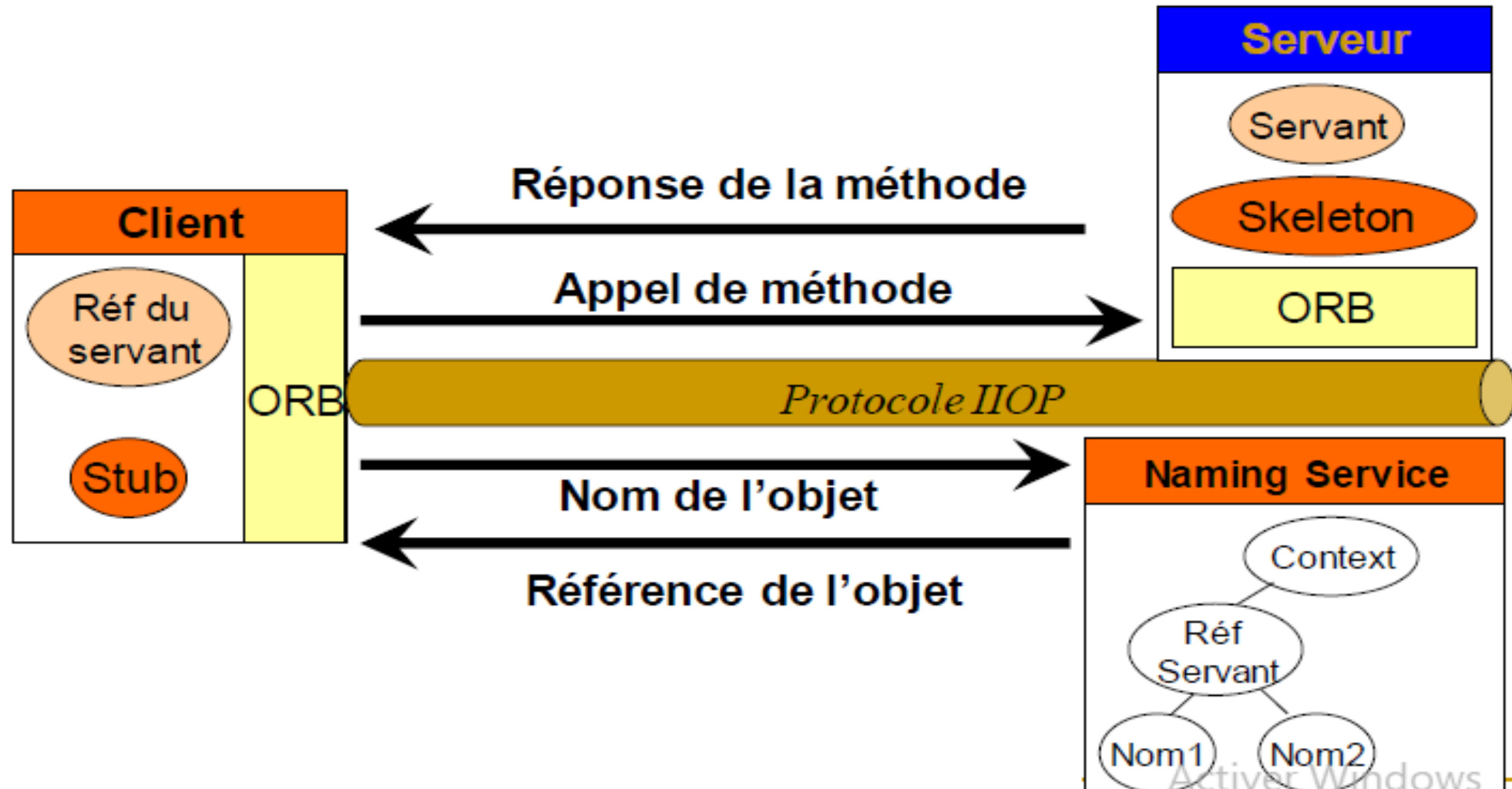


# L'adaptateur d'objets (POA)

---

- ▶ Différents types d'adaptateurs sont envisageables :
  - ▶ **BOA** ou Basic Object Adapter : les structures d'accueil sont matérialisées par des processus systèmes.
  - ▶ **POA** ou Portable Object Adapter : l'implantation des objets est réalisée par des objets fournis par un langage de programmation.
  - ▶ **OODA** ou Object-Oriented Database Adapter : la structure d'accueil est une base de données orientée objet.
  - ▶ **LOA** ou Library Object Adapter : le code d'implantation des objets est stocké dans des bibliothèques chargées dans l'espace des applications clientes.
  - ▶ **COA** ou Card Object Adapter : l'implantation des objets est stockée dans une carte à microprocesseur. Cet adaptateur a été expérimenté conjointement par le LIFL et la société Gemplus.

# L'annuaire C.O.R.B.A.



# Démarche à suivre pour développer un projet CORBA

---

- ▶ Conception :
  - ▶ UML (Cas d'utilisation et diagramme de classes)
  - ▶ Maquettes de l'application
- ▶ Fichier IDL (indépendant du langage)
- ▶ Compilation IDL
  - ▶ Génération des STUBS et SKELETONS
  - ▶ Génération des interfaces des objets distants
  - ▶ Génération des classes du modèle.
- ▶ Implémentation de l'objet distant (Servant)
- ▶ Création du serveur
- ▶ Création du client
- ▶ Déploiement et lancement
  - ▶ Lancer le Naming Service
  - ▶ Lancer le serveur
  - ▶ Lancer le Client

# IDL : Interface Definition Language

---

- ▶ **Premières règles sur l'IDL :**

- ▶ Une interface est une énumération d'opérations et de définitions de types de données.

```
interface Exemple
{
    // contenu de l'interface
};
```

- ▶ Une interface supporte l'héritage multiple.

```
interface AutreExemple : Exemple1, Exemple2
{
    // contenu de l'interface
};
```



# Décrire une opération dans l'IDL

- Les opérations décrites dans une interface respectent le format suivant :

*type\_de\_retour* nom\_de\_l'operation ( *liste\_des\_paramètres* ) ;

C.O.R.B.A. offre plusieurs types de données :

- les types de bases
- les types complexes

La liste des paramètres peut être vide.

# Les types de bases de CORBA

---

- ▶ boolean
- ▶ octet
- ▶ short ( ushort )
- ▶ long ( ulong )
- ▶ long long ( ulonglong )
- ▶ float
- ▶ double
- ▶ long double
- ▶ char
- ▶ wchar
- ▶ string
- ▶ wstring

# Les paramètres d'une opération dans l'IDL

---

- ▶ La description d'un paramètre comporte trois parties :
  - ▶ le modificateur
  - ▶ le type de l'argument ( type de base ou type complexe )
  - ▶ le nom de l'argument
- ▶ Le modificateur spécifie le sens d'échange du paramètre :
  - ▶ **in** : du client vers l'objet CORBA
  - ▶ **out** : en retour, de l'objet CORBA le client
  - ▶ **inout** : équivalent à un passage par adresse.

## Un exemple de description IDL: Interface

---

- ▶ L'exemple suivant décrit un objet qui offre une interface appelée « **Premier** ».
- ▶ Cette interface comporte :
  - ▶ une méthode dénommée « affiche » qui entraîne l'affichage d'un message sur le serveur ( message passé en tant que paramètre par le client ).
  - ▶ Une méthode produit qui retourne le produit de deux nombres a et b.

```
interface Premier
{
    void affiche ( in string message );
    double produit (in double a, in double b);
};
```

# Un exemple de description IDL: Déclarer une structure

---

```
module corbaBanque {  
    struct Compte {  
        long code;  
        float solde;  
    };  
    typedef sequence<Compte> tabComptes;  
    interface IBanqueRemote {  
        double conversion(in double mt);  
        Compte getCompte(in long code);  
        tabComptes getComptes();  
    };  
};
```

# Compilation d'une description IDL

---

- ▶ La description doit être compilée afin de générer les amorces ( souche et squelette ) requises pour l'établissement de la communication inter-processus.
- ▶ Exemple de compilation IDL : `idlj -fall -v Premier.idl`
- ▶ A l'issu de la compilation, plusieurs fichier sont créés :
  - ▶ PremierOperations.java : il s'agit des opérations de l'interface
  - ▶ Premier.java : il s'agit de l'interface
  - ▶ PremierPOA.java : il s'agit du squelette,
  - ▶ \_PremierStub.java : il s'agit de la souche,

## Concept de « mapping »

---

- ▶ Une description IDL est **traduite** vers un langage de programmation.
- ▶ Les règles de traduction sont appelées « **mapping** » et font partie de la spécification CORBA.
- ▶ Grâce au mapping, deux fournisseurs d'ORBs offriront le même modèle de programmation.

→ **portabilité des applications**

# Correspondance des types de bases

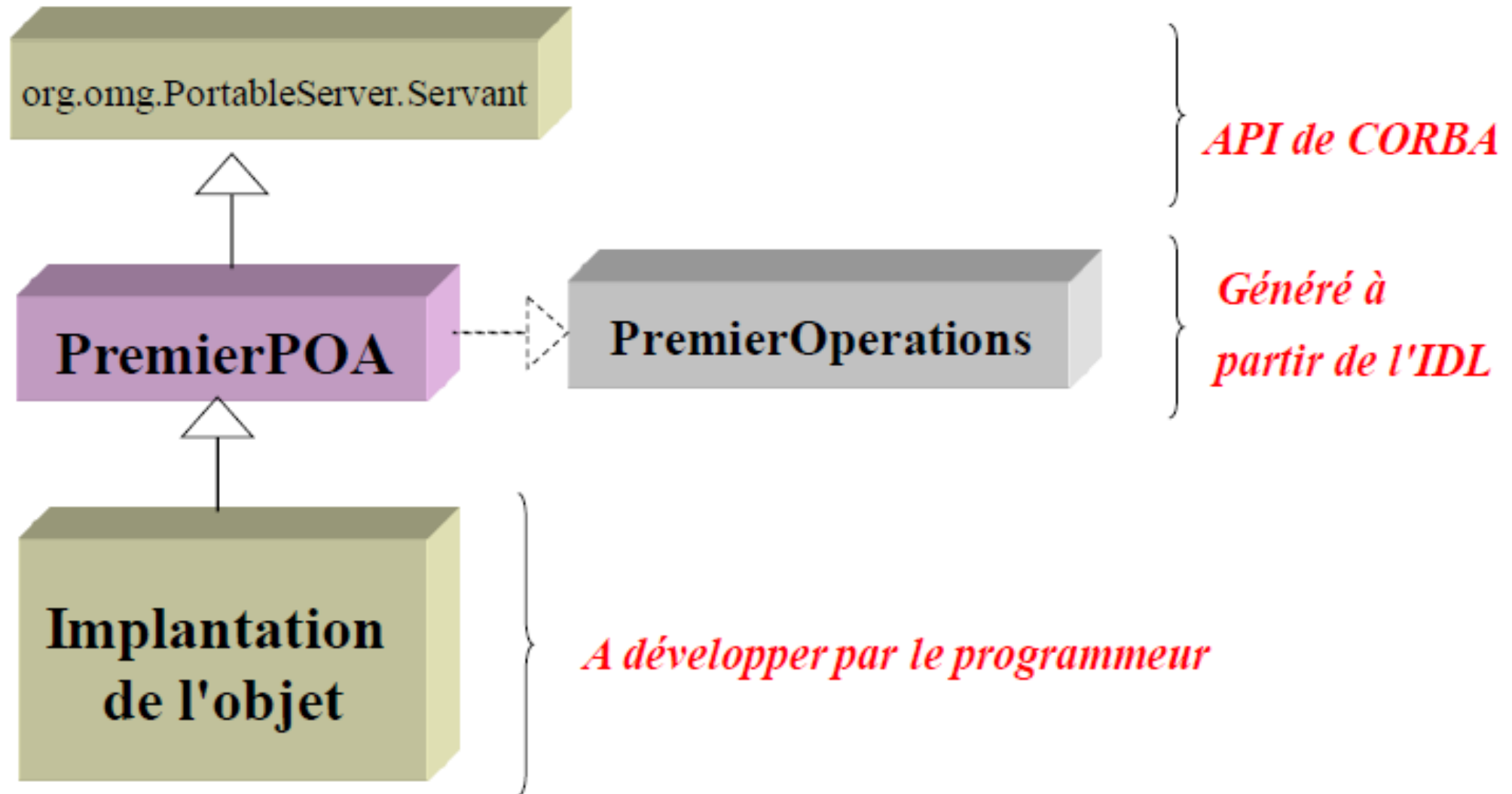
IDL	boolean	octet	short ushort	long ulong	long long ulong long
Java	boolean	byte	short	int	long

IDL	float	double	long double	char	wchar
Java	float	double		char	char

IDL	string	wstring
Java	string	string



# Implantation de l'objet distant (Servant)



## Exemple d'une application

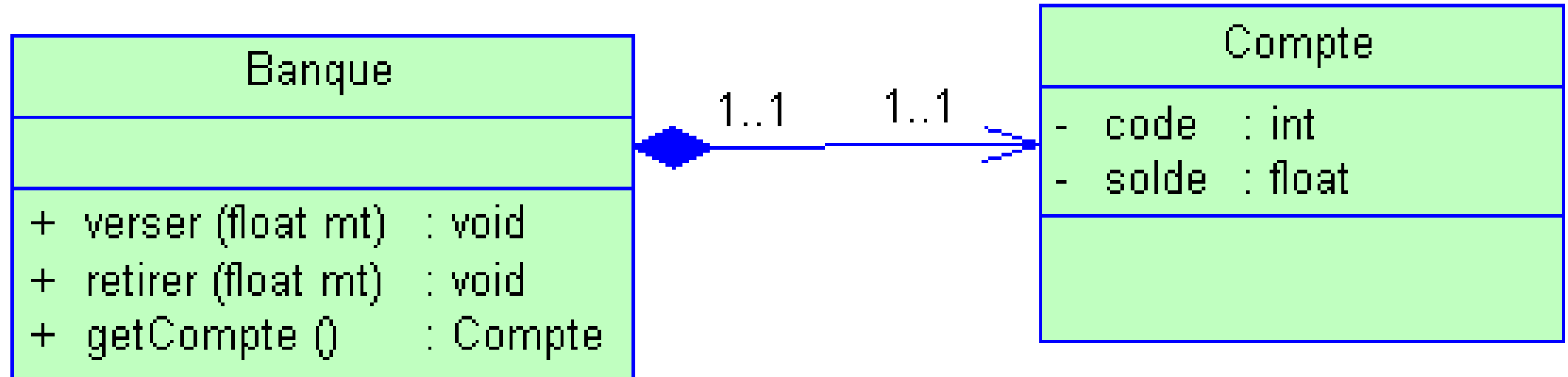
---

- ▶ Supposons que l'on souhaite créer une application client serveur qui permet de gérer des comptes bancaires, en utilisant le middleware CORBA.
- ▶ Dans cette partie, le serveur crée un objet distant qui représente un compte qui est défini par son code et son solde et permet à un client distant faire des versements, des retraits et des consultations de ce compte.

# 1 – Conception :

---

## ► Diagramme de classes:



# Fichier IDL

---

```
module corbaBank{  
    struct Compte{  
        long code;  
        float solde;  
    };  
    typedef Compte cpte;  
    interface Banque{  
        void verser(in float mt);  
        void retirer(in float mt);  
        cpte getCompte();  
    };  
};
```

## Fichier IDL:

---

- ▶ Ce fichier déclare les éléments suivants :
  - ▶ Un module nommé **corbaBanque** (représente un package dans java).
  - ▶ Une structure **Compte** qui déclare les variables d'un compte à savoir le code de type long (équivalent de **int** en java) et le solde de type float.
  - ▶ L'interface de l'objet distant nommé « **Banque** » qui déclare des méthodes abstraites :
    - ▶ **verser** : qui possède un paramètre « **mt** » de type float en entrée
    - ▶ **retirer** : qui possède un paramètre « **mt** » de type float en entrée
    - ▶ **getCompte()** : qui retourne l'objet **Compte**.

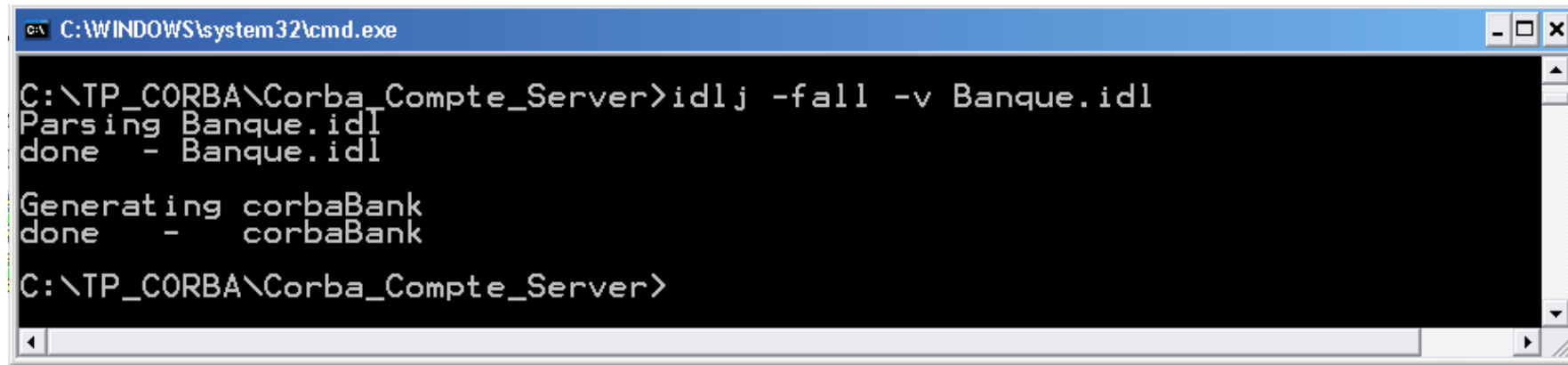
# Compilation IDL

---

- ▶ Cette étape consiste à utiliser un utilitaire fourni par L'ORB
  - ▶ Cette opération consiste à générer le code:
    - ▶ des STUBS (Souche)
    - ▶ des SKELETONS (Squelette)
    - ▶ Interface du servant
    - ▶ Classes représentant chaque structure du modèle
    - ▶ Et d'autres codes utilisées par le serveur et le client
- ▶ Dans java, ces fichiers sont générés à partir du fichier IDL en utilisant l'utilitaire **IDLJ** fourni avec le kit de développement java.
- ▶ Pour faire cette opération, il faut se placer dans le répertoire de votre projet sur ligne de commandes et exécuter la commande suivante.

**idlj -fall -v Banque.idl**

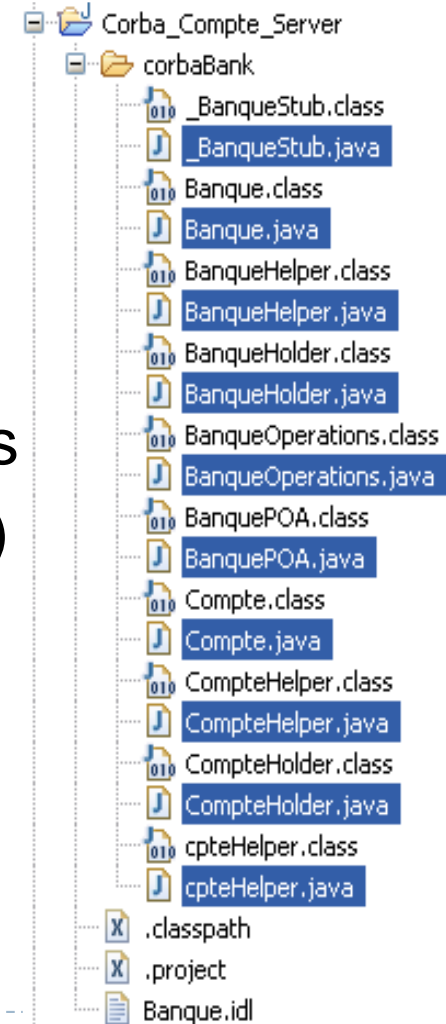
### 3- Compilation IDL



```
C:\WINDOWS\system32\cmd.exe
C:\TP_CORBA\Corba_Compte_Server>idlj -fall -v Banque.idl
Parsing Banque.idl
done - Banque.idl

Generating corbaBank
done - corbaBank

C:\TP_CORBA\Corba_Compte_Server>
```



- ▶ **BanqueOperations.java** : Interface qui déclare les méthodes abstraites
- ▶ **Banque.java** : Interface de l'objet distant (Hérite de BanqueOperations)
- ▶ **BanquePOA** : Squelette (Skeleton)
- ▶ **\_BanqueStub.java** : Souche (STUB)
- ▶ **BanqueHelper.java** : Utilisé par le client.
- ▶ **BanqueHolder.java** : Utilisé par le serveur.
- ▶ **Compte.java** : Classe qui représente le compte ;

## 4 – Implémentation du Servant

---

- ▶ L'implémentation d'un objet CORBA distant est une classe qui hérite du squelette et qui redéfinit toutes les méthodes abstraites de l'interface.
- ▶ Dans notre cas, cette classe sera nommée ImplBanque. Elle hérite du squelette BanquePOA
- ▶ Cette classe définit:
  - ▶ une variable qui représente un objet de la classe Compte ,
  - ▶ un constructeur qui créer un compte
  - ▶ les trois méthodes déclarés dans l'interface à savoir : verser, retirer et getCompte.
- ▶ Le code java de cette implémentation est le suivant :



## 4 – Implémentation du Servant

---

```
package implOD;
import corbaBank.BanquePOA;
import corbaBank.Compte;
public class BanquImpl extends BanquePOA {
    private Compte compte;
    public BanquImpl () {
        compte=new Compte(1,0);
    }
    public Compte getCompte() {
        return compte;
    }
    public void retirer(float mt) {
        compte.solde-=mt;
    }
    public void verser(float mt) {
        compte.solde+=mt;
    }
}
```

## 5 – Création du serveur java

```
import org.omg.CORBA.ORB; import org.omg.CosNaming.*;
import org.omg.PortableServer.POA;
import org.omg.PortableServer.POAHelper;
public class ServeurCorba {
    public static void main(String[] args) {
        try {
            // Initialiser l'ORB
            ORB orb=ORB.init(args,null);
            // Créer le POA Manager
            POA rootPOA= POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
            // Activer le POA Manager
            rootPOA.the_POAManager().activate();
            // Créer l'objet qui offre le service distant
            BanqueImpl od=new BanqueImpl ();
```

## 5 – Création du serveur java

```
// Créer un contexte de noms
NamingContext root=NamingContextHelper.narrow
    (orb.resolve_initial_references("NameService"));
//Créer un tableau de noms qui seront attribués à l'objet
NameComponent[] nsNom=new NameComponent[1];
// Définir un nom publique de l'objet distant
nsNom[0]=new NameComponent("MaBanque","");
//Enregistrer la référence de l'objet distant dans le Naming
//Service
root.rebind(nsNom,rootPOA.servant_to_reference(od));
// Démarrer le serveur
orb.run();
    } catch (Exception e) {
        e.printStackTrace();
    }
}}
```

## 6 – Projet Du Client

---

- ▶ Avant de créer la classe du client, il faut tout d'abord copier dans le projet du client les fichiers nécessaires au fonctionnement d'un client CORBA. Ces fichiers sont principalement :
  - ▶ le STUB
  - ▶ l'interface de l'objet distant
  - ▶ la classe Compte.
- ▶ Tous ces fichiers se trouvent dans le dossier corbaBank généré par l'utilitaire IDLJ. Copier donc le dossier corbaBank du projet du Serveur dans le projet du Client.
  - ▶ Une autre solution plus pratique est de copier uniquement l'IDL et de générer dans le projet du client.

## 6 – Projet Du Client

```
import org.omg.CORBA.ORB; import org.omg.CORBA.Object;
import org.omg.CosNaming.*; import corbaBank.*;
public class ClientCorba {
    public static void main(String[] args) {
        try{
            // Initialiser l'ORB
            ORB orb=ORB.init(args,null);
            // Créer le contexte de Naming Service
            NamingContext root=NamingContextHelper.narrow (orb.resolve_initial_references("NameService"));
            // Créer un tableau de noms
            NameComponent[] nsNom=new NameComponent[1];
            // Initialiser le nom de l'objet Distant
            nsNom[0]=new NameComponent("MaBanque","");
            // Récupérer la référence de l'objet distant à partir du Naming Service Object
            remoteRef=root.resolve(nsNom);
            // Créer le représentant local de l'objet distant
            Banque b=BanqueHelper.narrow(remoteRef);
            // Invocation des méthodes distantes.
            b.verser(4000); b.retirer(2000); b.verser(1000);Compte cpte=b.getCompte();
            System.out.println("Code="+cpte.code+"Solde="+cpte.solde);
        }catch (Exception e) { e.printStackTrace();}}
```

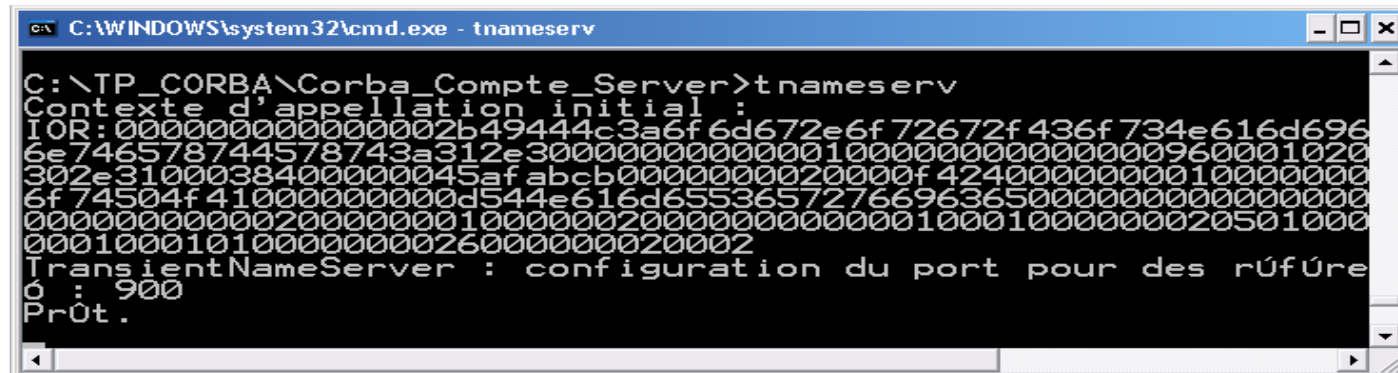


## 7- Lancement

---

### 1- Lancer le NamingService :


- ▶ Se placer dans le dossier du projet du serveur :
- ▶ Exécuter la commande : **tnameserv**



```
C:\WINDOWS\system32\cmd.exe - tnameserv
C:\TP_CORBA\Corba_Compte_Server>tnameserv
Contexte d'appellation initial :
IOR:00000000000000002b49444c3a6f6d672e6f72672f436f734e616d696
6e746578744578743a312e30000000000000100000000000000960001020
302e3100038400000045afabcb0000000020000f424000000010000000
6f74504f4100000000d544e616d655365727669636550000000000000
00000000002000000010000002000000000000010001000000020501000
000100010100000000260000000020002
TransientNameServer : configuration du port pour des r f re   : 900
Prot.
```

### 1- Lancer le Serveur :

- ▶ Se placer dans le dossier du projet du serveur :
- ▶ Exécuter la commande : **java ServeurCorba**



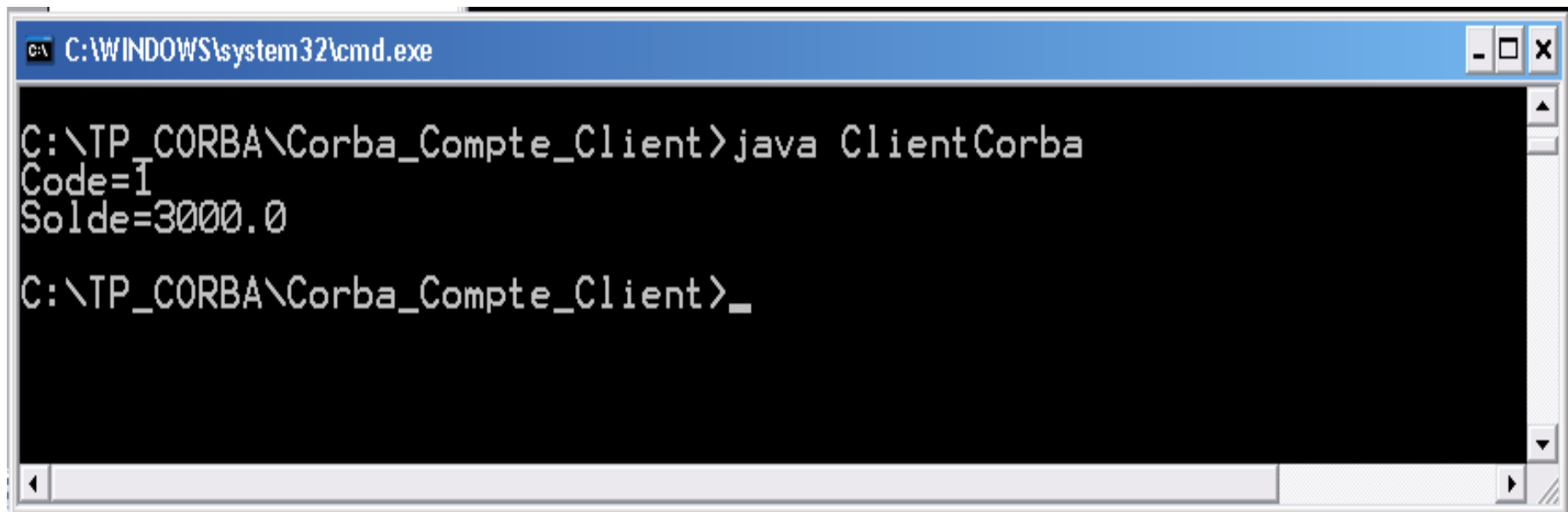
```
C:\WINDOWS\system32\cmd.exe - java ServeurCorba
C:\TP_CORBA\Corba_Compte_Server>java ServeurCorba
```

## 7- Lancement

---

### 3 - Lancer le Client :

- ▶ Se placer dans le dossier du projet du client :
- ▶ Exécuter la commande : **java ClientCorba**



```
C:\WINDOWS\system32\cmd.exe

C:\TP_CORBA\Corba_Compte_Client>java ClientCorba
Code=1
Solde=3000.0

C:\TP_CORBA\Corba_Compte_Client>_
```