TD 3: Middleware CORBA .

Exemple : Service Banque avec CORBA

l'IDL (Interface Definition Language)

L'**IDL** (Interface Definition Language) est un langage de spécification utilisé dans les systèmes distribués, notamment dans l'architecture **CORBA** (Common Object Request Broker Architecture). Son rôle principal est de définir les interfaces des objets logiciels de manière indépendante des langages de programmation, permettant ainsi une interopérabilité entre des systèmes hétérogènes.

Principales caractéristiques de l'IDL :

- **Indépendance du langage** :
  L'IDL est conçu pour être indépendant des langages de programmation. Les interfaces définies en IDL sont traduites en code source (stubs et skeletons) dans des langages spécifiques comme Java, C++, Python, etc.
- **Définition des interfaces** :
  Il permet de spécifier les méthodes (noms, paramètres, types de retour) qu'un objet distant expose, ainsi que les exceptions qu'il peut générer. Cela agit comme un contrat entre le client et le serveur.
- **Interopérabilité** :
  L'IDL facilite la communication entre des composants logiciels développés dans différents langages ou fonctionnant sur différents systèmes d'exploitation.
- **Structures de données** :
  IDL prend en charge les structures de données complexes, telles que :
- Types primitifs (int, float, char, etc.)
- Types définis par l'utilisateur (struct, union)
- Collections (tableaux, séquences)
- Exceptions
- Interfaces et modules pour structurer le code.

L'IDL suit une grammaire formelle similaire à celle des langages de programmation comme C et C++, mais avec des concepts adaptés à la définition d'interfaces pour les systèmes distribués. Voici un aperçu des principaux éléments de sa syntaxe et de ses structures :

---

1. Modules

- Les modules permettent de regrouper des définitions sous un espace de noms.
- Syntaxe :

```
module ModuleName {
   // Déclarations
};
```

2. Interfaces

- Les interfaces définissent les contrats des objets distants, spécifiant les méthodes accessibles aux clients.
- Syntaxe :

```
interface InterfaceName {
   returnType methodName(in paramType paramName, out
paramType paramName);
};
```

- **Paramètres** :
  - `in` : Paramètre d'entrée.
  - `out` : Paramètre de sortie.
  - `inout` : Paramètre d'entrée et de sortie.

3. Types de données

- **Primitifs** : `short`, `long`, `float`, `double`, `char`, `boolean`, etc.
- **Composés** :
  - **Structures** :

```
struct StructName {
   type1 member1;
   type2 member2;
};
```

Séquences :

```
sequence<type, maxSize> seqName;
```

Tableaux :

```
type arrayName[size];
```

4. Exceptions

- Les exceptions permettent de signaler des erreurs spécifiques.
- Syntaxe :

```
exception ExceptionName {
   memberType memberName;
};
```

5. Constantes

- Syntaxe :

```
const type constantName = value;
```

6. Attributs

- Syntaxe :

```
attribute attributeType attributeName;
readonly attribute attributeType attributeName;
```

Héritage

- Une interface peut hériter d'une ou plusieurs interfaces.
- Syntaxe :

```
interface DerivedInterface : BaseInterface1,
BaseInterface2 {
   // Méthodes additionnelles
};
```

Fonctionnement :

- Le fichier .idl est compilé par un compilateur IDL spécifique à l'ORB (Object Request Broker), générant des stubs pour les clients et des skeletons pour les serveurs.
- Ces stubs et skeletons servent de "ponts" entre le client, le serveur et le middleware CORBA.

Applications :

L'IDL est utilisé dans :

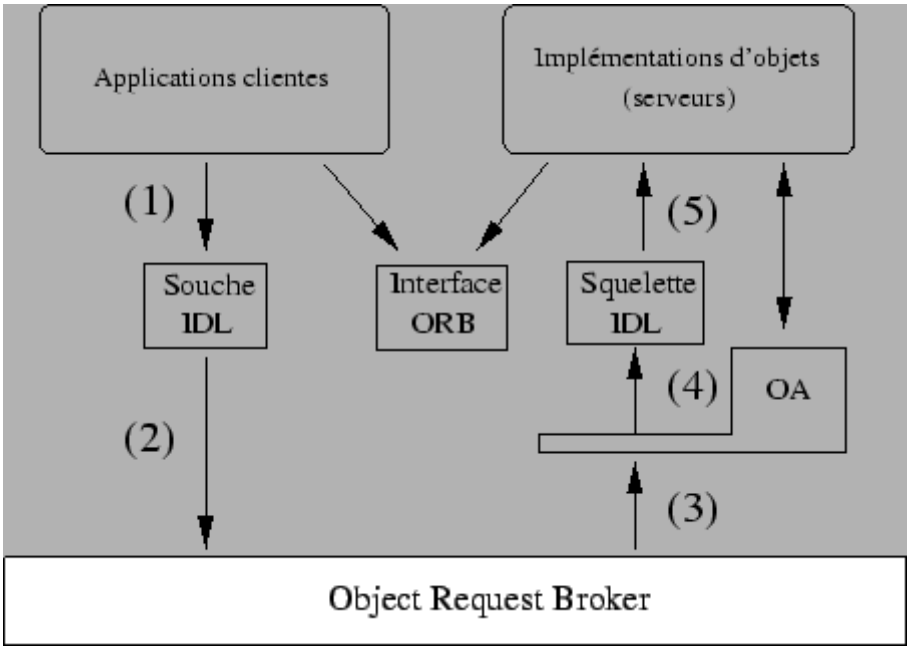
- CORBA pour les systèmes distribués.
- D'autres architectures similaires, comme COM/DCOM de Microsoft ou certaines parties des systèmes basés sur SOAP/WSDL.

Avantages :

- **Standardisation** : Garantit la compatibilité entre divers systèmes.
- **Flexibilité** : Permet l'écriture de composants réutilisables.
- **Interopérabilité** : Facilite les communications entre langages et plateformes.

Supposant qu'on veut créer un serveur CORBA qui offre à un client CORBA les services suivant:

Interface Banque qui offre deux services :
1 : convertir un montant Euro en Dinar
2 : Renvoyer le solde d'un compte donné.
En suivant le schéma de base CORBA sur la figure suivante :



Environnement de développement :

Pour développer ce service on va utiliser le JDK 8 qui offre un compilateur « idlj » et un serveur « tnameserv » nécessaire pour les applications CORBA en plus il faut rajouter le plugin OpenORB à Eclipse :

Interface Banque :

Pour commencer, il faut écrire un fichier d'interface en langage IDL (Interface Definition Language) qui décrit les prototypes des méthodes de l'objet distant (on décrit le nom de la méthode et les types des paramètres et du résultat)

1. Le module "corbaBanque" :
    - Le module est une manière de regrouper des définitions, des structures et des interfaces connexes dans un seul espace de nom. Dans ce cas, "corbaBanque" est le nom du module (représente la notion de Package dans jabva).
2. La structure "Compte" :
    - La structure "Compte" est définie avec deux membres :
        - "code" de type "long" : Il s'agit d'un numéro de compte.
        - "solde" de type "double" : Il s'agit du solde du compt
3. L'interface "IBanqueRemote" :
    - L'interface "IBanqueRemote" est définie avec deux méthodes :

- "conversion" : Cette méthode prend un argument d'entrée de type "double" nommé "mt" et renvoie un résultat de type "double". Cette méthode est utilisée pour effectuer une conversion
- "getCompte" : Cette méthode prend un argument d'entrée de type "long" nommé "code" et renvoie un résultat de type "Compte". Elle est utilisée pour obtenir les détails d'un compte en fonction de son "code".

Interface Banque : Code IDL :

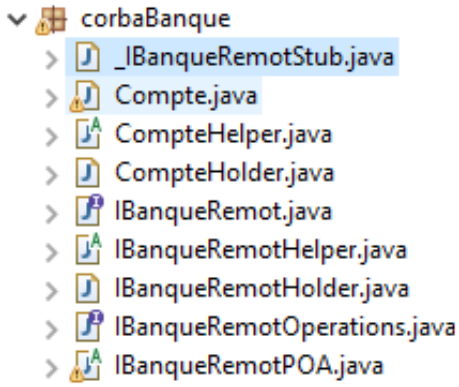
```
module corbaBanque {
            struct Compte {
                        long code;
                        double solde;
                        };
            interface IBanqueRemot {
                        double conversion(in double mt);
                        Compte getCompte(in long code);
                        };
```

Compilation du fichier IDL :

Dans une ligne de commande cmd : lancer le compilateur idlj avec les options suivantes :

```
>idlj –fall –v Banque.idl
```

Cette commande va générer les classes suivantes :



Le Package `corbaBanque` qui représente le Module IDL
La classe `Compte.java` qui représente le type `construct Compte` définie dans l'IDL
La classe `IBanqueRemote.java/ IBanqueRemoteOperations.java` qui représente l'interface `IbanqueRemote` définie dans le fichier IDL
La classe `_IBnaqueRemoteStub.java` qui représente le Stub Client
La classe `IBanqueRemotPOA.java` qui représente le skeleton Coté Serveur.

Pour une interface IDL, voici les fichiers générés et leur rôle respectif :

1. **`MyService.java`** : Interface principale exposée au client.
2. **`MyServiceOperations.java`** : Interface listant les méthodes à implémenter côté serveur.

3. **`MyServiceHelper.java`** : Classe utilitaire pour les objets CORBA.
4. **`MyServiceHolder.java`** : Encapsulation pour les paramètres `out` et `inout`.
5. **`_MyServiceStub.java`** : Classe cliente pour les appels distants.
6. **`MyServicePOA.java`** : Classe abstraite générée pour l'implémentation côté serveur.

2. Serveur :

Dans le côté serveur on a besoin d'implémenter le service distant ainsi que le serveur lui-même qui va créer l'objet distant et publier sa référence

I**Classe BanqueServiceImpl : (Code Java)**

```
package Servant;

import corbaBanque.Compte;
import corbaBanque.IBanqueRemotPOA;

public class BanqueServiceImpl extends IBanqueRemotPOA{

	@Override
	public double conversion(double mt) {
		// TODO Auto-generated method stub
		return mt*215;
	}

	@Override
	public Compte getCompte(int code) {
		// TODO Auto-generated method stub
		return new Compte(code, Math.random()*3000);
	}
}
```

Classe ServeurCORBA (Code Java)

Ce code Java illustre un serveur CORBA simple qui publie un objet distant dans un annuaire JNDI (Java Naming and Directory Interface). Voici une explication étape par étape du code :

1. **Import des Classes** : Le code commence par importer les classes Java nécessaires pour le serveur CORBA, JNDI, et l'implémentation de l'objet distant (`BanqueServiceImpl`).
2. **Initialisation de l'ORB** : L'ORB (Object Request Broker) est initialisé en utilisant la méthode `ORB.init(args, null)`. L'ORB est le composant central de CORBA qui gère la communication entre les objets distants.
3. **Récupération du POA (Portable Object Adapter)** : Le code récupère une référence vers le POA en utilisant `ORB.resolve_initial_references("RootPOA")`. Le POA est responsable de la création et de l'activation d'objets distants.
4. **Activation du POA** : Après avoir récupéré le POA, il est activé en appelant `rootPOA.the_POAManager().activate()`. Cela permet au POA de gérer les objets distants.
5. **Création de l'Objet Distant** : Une instance de l'objet distant `BanqueServiceImpl` est créée. Cet objet représente l'implémentation de l'interface CORBA distante.

6. **Initialisation de l'annuaire JNDI** : Une instance de `InitialContext` est créée pour accéder à l'annuaire JNDI.
7. **Publication de la Référence** : L'objet distant (`od`) est publié dans l'annuaire JNDI avec la ligne `ctx.rebind("BANQUE", rootPOA.servant_to_reference(od))`. Cela permet aux clients de rechercher l'objet distant sous le nom "BANQUE" dans l'annuaire JNDI.
8. **Démarrage de l'ORB** : Enfin, l'ORB est démarré en appelant `orb.run()`. Cela permet à l'ORB de commencer à écouter les appels des clients.
9. **Gestion des Exceptions** : Le code gère les exceptions qui peuvent survenir pendant l'exécution. Les exceptions possibles incluent des erreurs liées au POA, à JNDI, à l'objet distant, etc.

```
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

import org.omg.CORBA.ORBPackage.InvalidName;
import org.omg.CORBA_2_3.ORB;
import org.omg.PortableServer.POA;
import org.omg.PortableServer.POAHelper;
import org.omg.PortableServer.POAManagerPackage.AdapterInactive;
import org.omg.PortableServer.POAPackage.ServantNotActive;
import org.omg.PortableServer.POAPackage.WrongPolicy;

import Servant.BanqueServiceImpl;

public class ServeurCORBA {

	public static void main(String[] args) {
		try {
		org.omg.CORBA.ORB orb = ORB.init(args, null);
		System.out.println("CRéer  l'ORB ....");
		/// intialisation de l'ORB
		POA rootPOA=
POAHelper.narrow(orb.resolve_initial_references("RootPOA"));// récupérer le POA
		System.out.println("Récupérer le POA ...");
		//caster l'objet ou la référence en objet POA en utlilisant narrow qui
permet de caster les objet CORBA
		rootPOA.the_POAManager().activate(); // Activer le POA
		System.out.println("Activer le POA ....");
		BanqueServiceImpl od = new BanqueServiceImpl();// Créer l'objet distant
		System.out.println("creer Objet CORBA ....");
		Context ctx = new InitialContext();// crear l'annuair jndi
		System.out.println("Initialiser l'annuair JNDI ....");
		Object ref = rootPOA.servant_to_reference(od); //générer la référence
		ctx.rebind("BANQUE", ref);// publier la référence
		System.out.println("Publier l'Objet CORBA ....");
		orb.run();// démarrer l'ORB

		} catch (InvalidName | AdapterInactive | NamingException |
ServantNotActive | WrongPolicy e) {
		e.printStackTrace();
		}
	}
}
```

Client CORBA (ClientCORBA.java) :

Ce code Java d'un client CORBA qui utilise un annuaire JNDI pour rechercher et accéder à un objet distant (implémenté par `IBanqueRemot`) via une référence distante. Voici une explication du code étape par étape :

1. **Import des Classes** : Le code commence par importer les classes Java nécessaires pour le client CORBA, JNDI, et les interfaces distantes (`IBanqueRemot` et `Compte`).
2. **Initialisation du Contexte JNDI** : Une instance de `InitialContext` est créée pour initialiser le contexte JNDI. Le contexte JNDI est utilisé pour effectuer des opérations de recherche d'objets distants.
3. **Recherche de la Référence de l'Objet Distant** : Le code utilise le contexte JNDI pour rechercher la référence de l'objet distant sous le nom "BANQUE" avec la ligne `ctx.lookup("BANQUE")`. Cela permet de récupérer la référence de l'objet distant enregistrée dans l'annuaire JNDI.
4. **Casting de la Référence** : La référence récupérée est ensuite castée en tant qu'objet de type `IBanqueRemot` en utilisant `IBanqueRemotHelper.narrow((org.omg.CORBA.Object) ref)`. Cela permet au client d'interagir avec l'objet distant en utilisant les méthodes de l'interface `IBanqueRemot`.
5. **Appel de Méthodes sur l'Objet Distant** : Le code effectue des appels de méthodes sur l'objet distant. Par exemple, il appelle `stub.conversion(23)` pour effectuer une conversion

```
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

import corbaBanque.Compte;
import corbaBanque.IBanqueRemot;
import corbaBanque.IBanqueRemotHelper;

public class ClientCORBA {

	public static void main(String[] args) {
		try {
		Context ctx = new InitialContext();// chercher le context JNDI
		System.out.println("Initialiser le Context JNDI ....");
		Object ref = ctx.lookup("BANQUE");// recuperer la reference du service
		System.out.println("Chercher la référence de l'objet distant BANQUE ");
		IBanqueRemot stub = IBanqueRemotHelper.narrow((org.omg.CORBA.Object)
ref);// caster la reference
		System.out.println("Caster la référence en un Objet IBanqueRemot ....");
		System.out.println("23 EURO = " + stub.conversion(23) + " en DA ");
		Compte cp = stub.getCompte(3);
		System.out.println("crer un compte  ....");
		System.out.println("code =" + cp.code);
		System.out.println(" Solde =" + cp.solde);

		} catch (NamingException e) {
		e.printStackTrace();
		}
	}
}
```

Déploiement et exécution :

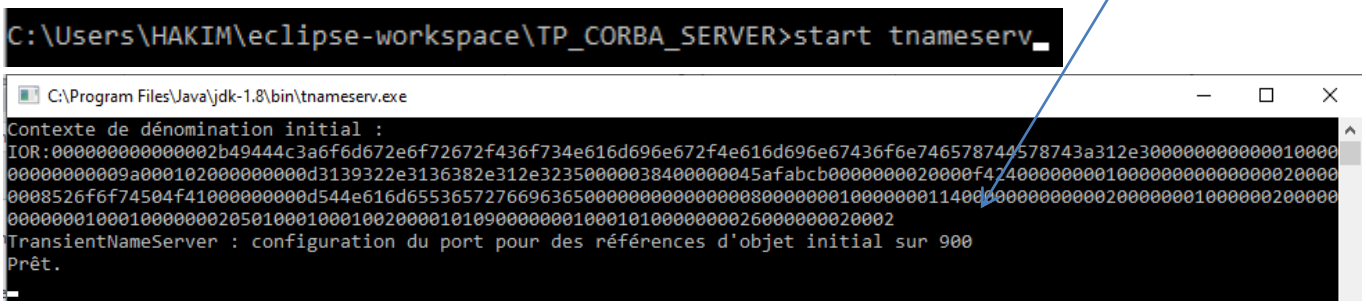
1. Avant d'exécuter le serveur, il faut configurer JNDI pour qu'il puisse travailler dans un contexte CORBA . le fichier **`jndi.properties`** contient les paramètres nécessaires à son fonctionnement, il doit être placé dans le dossier 'chemindupojet'/src comme ressource

```
java.naming.factory.initial=com.sun.jndi.cosnaming.CNCtxFactory
java.naming.provider.url=iiop://localhost:900
```

2. Sur la ligne de commande

Lancer le 'tnameserv' qui est un service de nommage et de registre fourni avec la plate-forme Java, et est utilisé pour enregistrer des noms de ressources Java et de services distants dans un annuaire.

```
Cmd> Start tnameserv
```



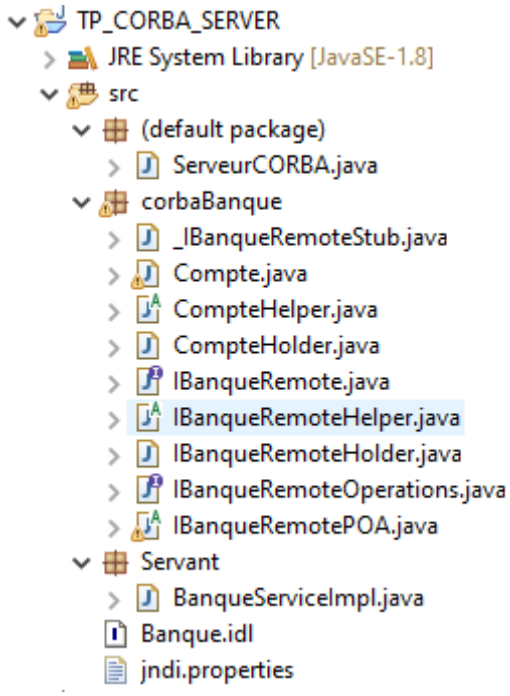
Lancer le Serveur

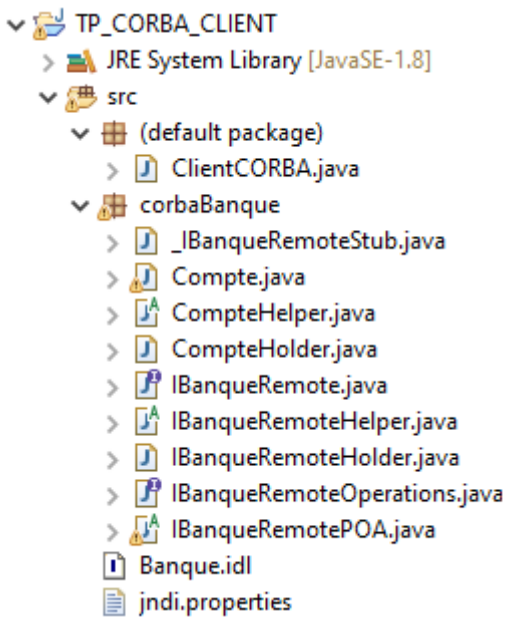
```
Cmd>    java ServeurCORBA
```

Lancer le client

```
Cmd>    java ClientCORBA
```

Structure du projet : Client et Serveur Corba :


TP_CORBA_SERVER
JRE System Library [JavaSE-1.8]
src
(default package)
ServeurCORBA.java
corbaBanque
_IBanqueRemoteStub.java
Compte.java
CompteHelper.java
CompteHolder.java
IBanqueRemote.java
IBanqueRemoteHelper.java
IBanqueRemoteHolder.java
IBanqueRemoteOperations.java
IBanqueRemotePOA.java
Servant
BanqueServiceImpl.java
Banque.idl
jndi.properties


TP_CORBA_CLIENT
JRE System Library [JavaSE-1.8]
src
(default package)
ClientCORBA.java
corbaBanque
_IBanqueRemoteStub.java
Compte.java
CompteHelper.java
CompteHolder.java
IBanqueRemote.java
IBanqueRemoteHelper.java
IBanqueRemoteHolder.java
IBanqueRemoteOperations.java
IBanqueRemotePOA.java
Banque.idl
jndi.properties