

Séance 01 : TP - Découverte CORBA

- Quelques ORBs disponibles gratuits :
 - ORB disponible dans le Java Development Kit
 - JacORB
 - <https://www.jacorb.org/download.html>

- OpenORB

counter poa: skeleton server

stub : cote client

- <https://sourceforge.net/projects/openorb/files/>

- Le développement d'une application basée sur CORBA commence toujours par la définition de l'interface IDL.
- Exemple d'un compteur simple :
 - Attribut IDL de type « long » (« readonly » car n'est pas destiné à être manipulé directement.)
 - Deux opérations « inc() » et « dec() » invoquées pour incrémenter et décrémenter la valeur courante du compteur.

```
1 // Counter.idl
2 interface Counter
3 {
4     readonly attribute long value;
5     void inc();
6     void dec();
7 };
```

- Le compilateur IDL utilisé dépend aussi bien du produit que de la plateforme utilisée (hardware, système opératoire) ainsi que le langage de programmation spécifique.
- Le compilateur IDL de JDK :

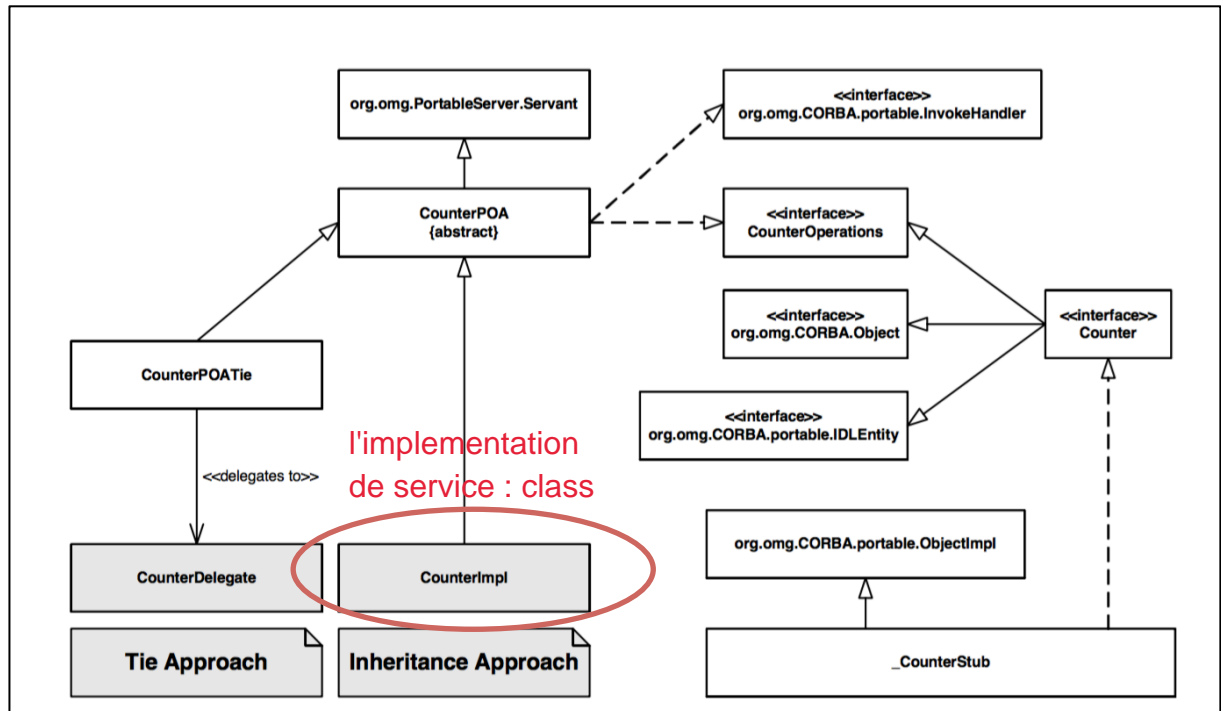
```
idlj -fall Counter.idl
```

- Le paramètre « -fall » permet de générer les squelettes (proxy côté client), ainsi que les squelettes (proxy côté serveur)
 - (-fclient parameter : Génération côté client uniquement)
 - (-fserver parameter : Génération côté serveur uniquement)
 - (-fallTIE parameter : Approche par délégation)
 - (-fserverTIE : Délégation côté serveur)
- Autres configurations :
Altération de la variable d'environnement pour l'invocation de « idlj »
- Fichiers générés (paramètre -fall)
 - CounterPOA.java
 - _CounterStub.java

- CounterHolder.java
- CounterHelper.java
- Counter.java
- CounterOperations.java

Lorsque l'exécution a lieu sur un seul poste, l'exécution du client et du serveur doit se faire sur leurs propres fenêtres de commande avec leurs propres configurations de variables d'environnement.

- Implémentation de « Counter » utilisant l'approche par héritage (JDK) :



L'implémentation fournie par les programmeurs doit hériter de la classe POA.

Notre classe d'implémentation sera nommée « Counter-Impl » (Convention CORBA : <Interface>Impl ou <Interface>_Impl)

- « CounterPOA » est une classe abstraite qui implémente les opérations de l'interface de l'interface « CounterOperations » mais ne comporte aucune déclaration des méthodes héritées. La classe « CounterImpl » doit implémenter toutes les méthodes déclarées au niveau de l'interface « Operations »

```

1  CounterOperations.java x
2  /**
3   * CounterOperations.java .
4   * Generated by the IDL-to-Java compiler (portable), version "3.2"
5   * from Counter.idl
6   * jeudi 17 mai 2018 05 h 21 WET
7   */
8
9
10 // Counter.idl
11 public interface CounterOperations
12 {
13     int value ();
14     void inc ();
15     void dec ();
16 } // interface CounterOperations
17

```

```

1  CounterImpl.java x
2  // CounterImpl.java
3  public class CounterImpl extends CounterPOA {
4      private int count;
5      public CounterImpl() {
6          count = 0;
7      }
8      public void inc() {
9          count++;
10     }
11     public void dec() {
12         count--;
13     }
14     public int value() {
15         return count;
16     }
17 }

```

- Implémentation du serveur (Approche par héritage)

```

1 // Server.java
2 import java.io.*;
3 import java.util.Properties;
4 import org.omg.CORBA.*;
5 import org.omg.PortableServer.*;
6 import static java.lang.System.*;
7 public class Server {
8     public static void main(String[] args) {
9         try {
10             Properties props = getProperties();
11             ORB orb = ORB.init(args, props);
12             org.omg.CORBA.Object obj = null;
13             POA rootPOA = null;
14             try {
15                 obj = orb.resolve_initial_references("RootPOA");
16                 rootPOA = POAHelper.narrow(obj);
17             } catch (org.omg.CORBA.ORBPackage.InvalidName e) {}
18             CounterImpl c_impl = new CounterImpl();
19             Counter c = c_impl._this(orb);
20             try {
21                 FileOutputStream file =
22                     new FileOutputStream("Counter.ref");
23                 PrintWriter writer = new PrintWriter(file);
24                 String ref = orb.object_to_string(c);
25                 writer.println(ref);
26                 writer.flush();
27                 file.close();
28                 out.println("Server started.");
29                 + " Stop: Ctrl-C";
30             } catch (IOException ex) {
31                 out.println("File error: "
32                     + ex.getMessage());
33             }
34             exit(2);
35             rootPOA.the_POAManager().activate();
36             orb.run();
37             } catch (Exception ex) {
38                 out.println("Exception: " + ex.getMessage());
39             }
40             exit(1);
41         }
42     }
43 }

```

recupérer
l'adaptateur obj

narrow pour
faire la conversion

- Initialisation de l' ORB « `ORB.init()` »
- Déterminer la référence de rootPOA
- Changement de type (`org.omg.CORBA.Object` → `org.omg.PortableServer.POA`) à travers la méthode « `narrow` » de la classe `POAHelper`
- Création d'une instance « `CounterImpl` » (Servant), associer avec l'ORB qui contient rootPOA, puis l'activer (« `_this()` »)
- Transformer IOR construit à travers l'instance de « `CounterImpl` » à une chaîne de caractère qui sera stockée au niveau du fichier « `Counter.ref` »
- Activation du Manger

`Orb.run()` : Serveur prêt à recevoir les requêtes

- Implémentation du client (Approche par héritage)

```

1 // Client.java
2 import java.io.*;
3 import java.util.*;
4 import org.omg.CORBA.*;
5 import static java.lang.System.*;
6 public class Client {
7     public static void main(String[] args) {
8         try {
9             Properties props = getProperties();
10             ORB orb = ORB.init(args, props);
11             String ref = null;
12             org.omg.CORBA.Object obj = null;
13             try {
14                 Scanner reader =
15                     new Scanner(new File("Counter.ref"));
16                 ref = reader.nextLine();
17             } catch (IOException ex) {
18                 out.println("File error: " + ex.getMessage());
19             }
20             exit(2);
21             obj = orb.string_to_object(ref);
22             if (obj == null) {
23                 out.println("Invalid IOR");
24             }
25             Counter c = null;
26             try {
27                 c = CounterHelper.narrow(obj);
28             } catch (BAD_PARAM ex) {
29                 out.println("Narrowing failed");
30             }
31             exit(3);
32             int inp = -1;
33             do {
34                 out.print("Counter value: " + c.value()
35                     + "\nAction (+/-/e)? ");
36                 out.flush();
37                 do {
38                     try {
39                         inp = in.read();
40                     } catch (IOException ioe) {}
41                 } while (inp != '+' && inp != '-' && inp != 'e');
42                 if (inp == '+')
43                     c.inc();
44                 else if (inp == '-')
45                     c.dec();
46                 } while (inp != 'e');
47             } catch (Exception ex) {
48                 out.println("Exception: " + ex.getMessage());
49             }
50             exit(1);
51         }
52     }
53 }

```

Initialisation de l'ORB

Lecture de l'IOR du serveur à partir du
fichier « `Counter.ref` »

Conversion en type
« `orb.string_to_object()` »

Reconversion vers le type « `Counter` » à
travers la méthode « `narrow` » de
« `CounterHelper` »)

Invocation des méthodes à travers des entrées
'+' ou '-'

- Execution :

L'objet IOR du serveur est enregistré au niveau du fichier « `Counter.ref` » , Celui-ci contient toutes les informations qu'un client peut nécessiter à distance afin de localiser l'objet serveur et invoquer les opérations : (Adresse IP, Numéro du port du poste serveur, Référence de l'objet etc..)

Le fichier «`Counter.ref`» doit être récupéré au niveau du poste client.

On verra que le service de nommage fourni par CORBA offre une meilleure alternative.