École Nationale Supérieure
d'Informatique et d'Analyse des Systèmes
- Rabat

Intégration des Services et des Objets

Compte Rendu du TP1 : Application CORBA Hello World

Élève :
WIAME YOUSFI

Enseignante:
M. MAHMOUD NASSAR

Table des matières

1 Introduction

1.1 Contexte du TP

Ce travail pratique s'inscrit dans le cadre du module **Intégration des Services et des Objets** et vise à introduire les concepts fondamentaux de l'architecture **CORBA (Common Object Request Broker Architecture)**. **CORBA** représente une technologie mature pour la communication entre objets distribués dans un environnement hétérogène.

1.2 Objectifs pédagogiques

Les objectifs principaux de ce TP sont :

— **Comprendre l'architecture CORBA** : Maîtriser les composants essentiels (ORB, POA, IOR, IDL)

— **Développer une application distribuée** : Implémenter un système client-serveur fonctionnel

— **Manipuler l'IDL** : Définir des interfaces indépendantes du langage de programmation

— **Gérer la communication inter-processus** : Établir des communications entre objets distants

— **Déployer sur des machines distantes** : Configurer et exécuter l'application en réseau

1.3 Environnement technique

L'environnement de développement utilisé comprend :

— **Langage** : Java

— **JDK** : Version 1.8 (incluant l'ORB et le compilateur IDL)

— **Compilateur IDL** : `idlj` (fourni avec le JDK)

— **Système d'exploitation** : Compatible multi-plateforme

1.4 Présentation de l'application

L'application *Hello World* constitue un exemple canonique en programmation distribuée. Dans notre contexte CORBA, elle démontre :

1. La définition d'une interface de service via IDL
2. L'implémentation d'un serveur hébergeant l'objet distant
3. Le développement d'un client consommant le service
4. La communication transparente entre composants distribués

2 Fondements théoriques de CORBA

2.1 Qu'est-ce que CORBA ?

CORBA (Common Object Request Broker Architecture) est une norme définie par l'**OMG (Object Management Group)** permettant à des objets logiciels de collaborer indépendamment de leur langage de programmation ou de leur localisation réseau.

2.1.1 Principe fondamental

CORBA repose sur le paradigme de l'**invocation de méthodes distantes** (Remote Method Invocation). Un client peut invoquer des méthodes sur un objet distant comme s'il s'agissait d'un objet local, le middleware **CORBA** se chargeant de toute la complexité de communication.

2.1.2 Architecture générale

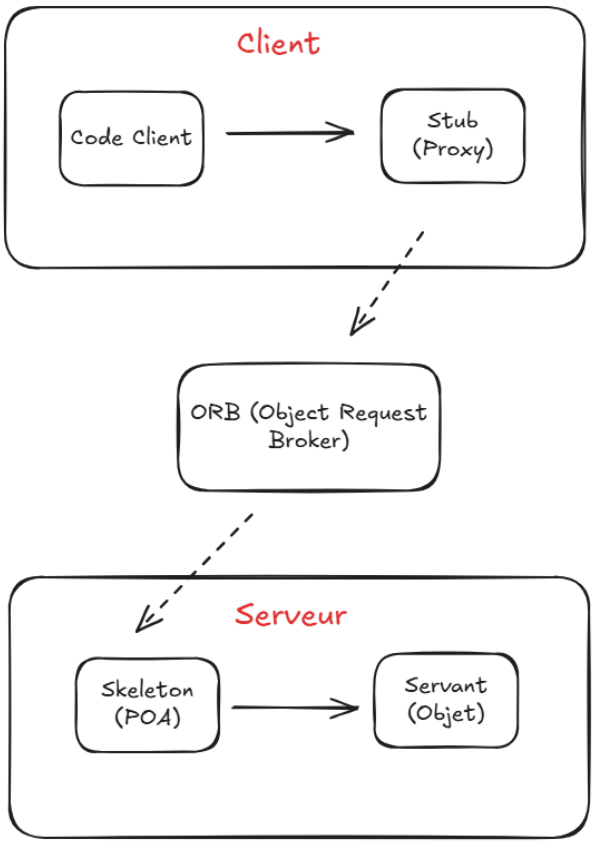

FIGURE 1 – Architecture **CORBA** simplifiée

2.2 Composants essentiels de CORBA

2.2.1 ORB (Object Request Broker)

L'**ORB** constitue le cœur du système **CORBA**. Il s'agit d'un bus logiciel responsable de :

— **Localisation des objets distants** : Trouver l'objet cible à partir de sa référence
— **Marshalling/Unmarshalling** : Sérialiser les paramètres et résultats
— **Transport des requêtes** : Acheminer les appels via le réseau
— **Gestion du protocole** : Utiliser **IIOP (Internet Inter-ORB Protocol)**

Analogie : L'**ORB** fonctionne comme un service postal qui achemine des messages entre expéditeurs et destinataires, quels que soient leurs emplacements.

2.2.2 IDL (Interface Definition Language)

L'**IDL** est un langage déclaratif neutre permettant de définir les interfaces des objets **CORBA** indépendamment de tout langage d'implémentation.

Caractéristiques principales :

— Syntaxe similaire à C++
— Définit uniquement les signatures (pas d'implémentation)
— Supporte les types de données complexes
— Permet l'héritage d'interfaces

Exemple de définition IDL :

```
module hello {
    interface Hello {
        string sayHello();
    };
};
```

2.2.3 POA (Portable Object Adapter)

Le **POA** est un adaptateur qui fait le lien entre les objets serveurs (servants) et l'**ORB**. Il gère :

— **Activation des objets** : Créer et activer les servants
— **Génération des références** : Produire les IOR
— **Dispatching des requêtes** : Acheminer les appels vers les bons objets
— **Gestion du cycle de vie** : Contrôler la durée de vie des objets

2.2.4 IOR (Interoperable Object Reference)

L'**IOR** est une référence unique et sérialisable identifiant un objet CORBA. Elle contient :

— L'identifiant du type de l'objet
— L'adresse réseau du serveur (IP et port)
— La clé d'objet permettant de l'identifier sur le serveur
— Des informations de protocole (IIOP)

Format typique :

```
IOR:000000000000001C49444C3A68656C6C6F2F48656C6C6F3A312E300000...
```

2.3 Flux de communication CORBA

Le processus complet d'une invocation CORBA suit ces étapes :

1. **Initialisation du serveur**
   - Création de l'ORB
   - Configuration du POA
   - Instanciation du servant
   - Génération et publication de l'IOR
2. **Connexion du client**
   - Initialisation de l'ORB client
   - Récupération de l'IOR
   - Conversion en référence d'objet (narrowing)
3. **Invocation de méthode**
   - Appel de la méthode sur le stub
   - Marshalling des paramètres
   - Transmission via l'ORB
   - Réception par le skeleton (POA)
   - Exécution sur le servant
   - Retour du résultat

2.4 Avantages et limitations de CORBA

2.4.1 Avantages

- **Interopérabilité** : Communication entre langages hétérogènes (Java, C++, Python)
- **Transparence de localisation** : Le client ignore où se trouve l'objet
- **Standardisation** : Norme OMG largement adoptée
- **Typage fort** : Vérification des types à la compilation

2.4.2 Limitations

- **Complexité** : Courbe d'apprentissage élevée
- **Performance** : Overhead du marshalling
- **Configuration** : Gestion des IOR et du réseau
- **Obsolescence relative** : Concurrence de technologies plus modernes (REST, gRPC)

3 Architecture de l'application

3.1 Structure du projet

Notre application CORBA est organisée selon la structure suivante :

```
TP1/
 hello.idl                    # Interface IDL
 ior.txt                      # Référence IOR générée
 hello/                       # Package Java
    Client.java               # Client CORBA
    Server.java               # Serveur CORBA
    HelloImpl.java            # Implémentation
    Hello.java                # Interface (générée)
    HelloHelper.java          # Helper (généré)
    HelloHolder.java          # Holder (généré)
    HelloOperations.java      # Opérations (générées)
    HelloPOA.java             # POA (généré)
    _HelloStub.java           # Stub (généré)
 README.md                    # Documentation
 .gitignore                   # Fichiers exclus
```

3.2 Composants de l'application

3.2.1 Interface IDL (hello.idl)

Le fichier IDL définit le contrat de service de notre application :

```
1  module hello {
2      interface Hello {
3          string sayHello();
4      };
5  };
```

Analyse ligne par ligne :

— **Ligne 1** : Déclaration d'un module (équivalent à un package Java)

— **Ligne 2** : Définition de l'interface `Hello`

— **Ligne 3** : Méthode `sayHello` retournant une chaîne

Cette interface est **neutre** : elle peut être implémentée dans n'importe quel langage supportant CORBA.

3.2.2 Serveur (Server.java)

Le serveur héberge l'objet CORBA et attend les connexions :

```
1  package hello;
2
3  public class Server {
4      public static void main(String[] args) {
5          java.util.Properties props = System.getProperties();
6          int status = 0;
7          org.omg.CORBA.ORB orb = null;
8
9          try {
10             // Initialisation de l'ORB
11             orb = org.omg.CORBA.ORB.init(args, props);
12             status = run(orb);
```

```
13          } catch (Exception ex) {
14              ex.printStackTrace();
15              status = 1;
16          }
17
18          if (orb != null) {
19              try {
20                  orb.destroy();
21              } catch (Exception ex) {
22                  ex.printStackTrace();
23                  status = 1;
24              }
25          }
26          System.exit(status);
27      }
28
29      static int run(org.omg.CORBA.ORB orb)
30              throws org.omg.CORBA.UserException {
31          try {
32              // Récupération du POA racine
33              org.omg.PortableServer.POA rootPOA =
34                  org.omg.PortableServer.POAHelper.narrow(
35                      orb.resolve_initial_references("RootPOA"));
36
37              // Activation du POA manager
38              org.omg.PortableServer.POAManager manager =
39                  rootPOA.the_POAManager();
40              manager.activate();
41
42              // Création du servant
43              HelloImpl helloImpl = new HelloImpl();
44              org.omg.CORBA.Object ref =
45                  rootPOA.servant_to_reference(helloImpl);
46
47              // Génération de l'IOR
48              String ior = orb.object_to_string(ref);
49
50              // Sauvegarde de l'IOR dans un fichier
51              java.nio.file.Files.write(
52                  java.nio.file.Paths.get("ior.txt"),
53                  ior.getBytes()
54              );
55
56              System.out.println("Server ready. IOR saved to ior.txt");
57
58              // Mise en attente des requêtes
59              orb.run();
60
61              return 0;
62          } catch (Exception ex) {
63              ex.printStackTrace();
64              return 1;
65          }
66      }
67  }
```

3.2.3 Client (Client.java)

Le client se connecte au serveur et invoque la méthode distante :

```java
package hello;

import java.nio.file.Files;
import java.nio.file.Paths;

public class Client {
    public static void main(String[] args) {
        java.util.Properties props = System.getProperties();
        int status = 0;
        org.omg.CORBA.ORB orb = null;

        try {
            orb = org.omg.CORBA.ORB.init(args, props);
            status = run(orb);
        } catch (Exception ex) {
            ex.printStackTrace();
            status = 1;
        }

        if (orb != null) {
            try {
                orb.destroy();
            } catch (Exception ex) {
                ex.printStackTrace();
                status = 1;
            }
        }

        System.exit(status);
    }

    static int run(org.omg.CORBA.ORB orb) {
        try {
            // Lecture de l'IOR depuis le fichier
            String ior = new String(
                Files.readAllBytes(Paths.get("ior.txt")));

            // Conversion IOR en objet CORBA
            org.omg.CORBA.Object obj = orb.string_to_object(ior);

            // Narrowing vers l'interface Hello
            Hello helloRef = HelloHelper.narrow(obj);

            // Invocation de la méthode distante
            String response = helloRef.sayHello();
            System.out.println("Response from Server: " + response);

            return 0;
        } catch (Exception ex) {
            ex.printStackTrace();
            return 1;
        }
    }
}
```

3.2.4 Implémentation (HelloImpl.java)

La classe servant implémente la logique métier :

```java
package hello;

public class HelloImpl extends HelloPOA {
    @Override
    public String sayHello() {
        return "Hello, World!";
    }
}
```

Analyse :

— **Héritage de HelloPOA** : Classe de base générée par le compilateur IDL

— **Implémentation de sayHello()** : Logique applicative simple

— **Pas de code CORBA** : L'implémentation ne contient que la logique métier

3.3 Classes générées par le compilateur IDL

Le compilateur `idlj` génère automatiquement 6 classes à partir de `hello.idl` :

Classe	Description
`Hello.java`	Interface Java correspondant à l'interface IDL
`HelloHelper.java`	Fournit des méthodes utilitaires pour le narrowing, le marshalling et la manipulation de types
`HelloHolder.java`	Utilisé pour les paramètres `out` et `inout` dans les appels de méthodes
`HelloOperations.java`	Interface contenant les signatures des méthodes métier
`HelloPOA.java`	Classe de base pour l'implémentation côté serveur (skeleton)
`_HelloStub.java`	Stub côté client permettant d'invoquer les méthodes distantes

4 Implémentation et développement

4.1 Environnement de développement

4.1.1 Configuration système

— **Système d'exploitation** : Windows/Linux/macOS

— **JDK** : Version 1.8 (JDK 8)

— **Variables d'environnement** : `JAVA_HOME` et `PATH` configurés

4.1.2 Vérification de l'installation

```
# Vérifier la version de Java
java -version

# Vérifier la présence du compilateur IDL
idlj -version
```

4.2 Processus de compilation

4.2.1 Étape 1 : Compilation de l'interface IDL

La première étape consiste à générer les classes Java à partir du fichier IDL :

```
idlj -fall hello.idl
```

Options utilisées :

— `-fall` : Génère toutes les classes (client et serveur)

— Alternative `-fclient` : Génère uniquement les classes client

— Alternative `-fserver` : Génère uniquement les classes serveur

Résultat attendu : Création du répertoire `hello/` contenant les 6 classes générées.

4.2.2 Étape 2 : Compilation des classes Java

Une fois les classes générées, nous compilons l'ensemble du projet :

```
javac hello/*.java
```

Cette commande compile :

— Les 6 classes générées par `idlj`

— `Server.java`

— `Client.java`

— `HelloImpl.java`

Vérification : Les fichiers `.class` doivent apparaître dans le répertoire `hello/`.

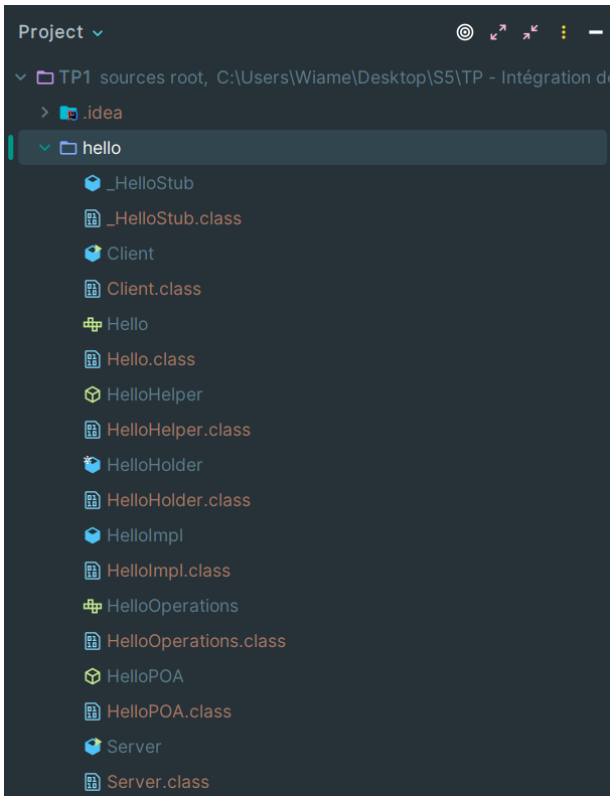

FIGURE 2 – Compilation des classes Java

4.3 Exécution de l'application

4.3.1 Lancement du serveur

Le serveur doit être démarré en premier :

```
java hello.Server
```

Sortie attendue :

```
Server ready. IOR saved to ior.txt
```

```
PS C:\Users\Wiame\Desktop\S5\TP - Intégration des Services et des Objets\TP1> java hello.Server
hello.Server ready. IOR saved to ior.txt
```

FIGURE 3 – Démarrage du serveur CORBA

Le serveur :

1. Initialise l'ORB
2. Crée l'objet servant
3. Génère l'IOR et le sauvegarde dans `ior.txt`
4. Entre en boucle d'attente (`orb.run()`)

4.3.2 Contenu du fichier IOR

Le fichier `ior.txt` contient une chaîne encodée représentant la référence de l'objet :

FIGURE 4 – Contenu du fichier IOR généré

Structure de l'IOR :

— Préfixe `IOR:`
— Informations de type (IDL ID)
— Adresse IP et port du serveur
— Clé d'objet (Object Key)
— Métadonnées de protocole (IIOP)

4.3.3 Exécution du client

Dans un second terminal, nous lançons le client :

```
java hello.Client
```

Sortie attendue :

```
Response from Server: Hello, World!
```

```
PS C:\Users\Wiame\Desktop\S5\TP - Intégration des Services et des Objets\TP1> java hello.Client
Response from hello.Server: Hello, World!
```

FIGURE 5 – Exécution du client CORBA

Le client :

1. Lit le fichier `ior.txt`
2. Convertit l'IOR en référence d'objet
3. Effectue le narrowing vers `Hello`
4. Invoque `sayHello()`
5. Affiche la réponse

5 Analyse détaillée du code

5.1 Anatomie du serveur

5.1.1 Séquence d'initialisation

Le serveur suit un processus d'initialisation rigoureux :

```
// 1. Création de l'ORB
org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, props);
```

Cette ligne crée l'instance de l'Object Request Broker qui gérera toutes les communications.

```
// 2. Récupération du POA racine
org.omg.PortableServer.POA rootPOA =
    org.omg.PortableServer.POAHelper.narrow(
        orb.resolve_initial_references("RootPOA"));
```

Le POA (Portable Object Adapter) est obtenu via l'ORB. Le `narrow` convertit l'objet CORBA générique en type spécifique POA.

```
// 3. Activation du POA Manager
org.omg.PortableServer.POAManager manager =
    rootPOA.the_POAManager();
manager.activate();
```

Le POA Manager doit être activé pour que le POA accepte les requêtes.

```
// 4. Création et enregistrement du servant
HelloImpl helloImpl = new HelloImpl();
org.omg.CORBA.Object ref =
    rootPOA.servant_to_reference(helloImpl);
```

Le servant (objet d'implémentation) est créé et enregistré auprès du POA, qui génère une référence CORBA.

```
// 5. Génération et sauvegarde de l'IOR
String ior = orb.object_to_string(ref);
java.nio.file.Files.write(
    java.nio.file.Paths.get("ior.txt"),
    ior.getBytes()
);
```

L'IOR est généré puis sauvegardé pour que le client puisse s'y connecter.

```
// 6. Mise en attente des requêtes
orb.run();
```

L'ORB entre dans une boucle infinie, attendant et traitant les requêtes entrantes.

5.1.2 Gestion des ressources

Le serveur implémente une gestion propre des ressources :

```
try {
    orb = org.omg.CORBA.ORB.init(args, props);
    status = run(orb);
} catch (Exception ex) {
    ex.printStackTrace();
    status = 1;
} finally {
    if (orb != null) {
        try {
            orb.destroy();  // Libération des ressources
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

Le bloc `finally` garantit que l'ORB est détruit même en cas d'exception.

5.2 Anatomie du client

5.2.1 Processus de connexion

Le client établit la connexion en plusieurs étapes :

```
// 1. Lecture de l'IOR
String ior = new String(
    Files.readAllBytes(Paths.get("ior.txt"))
);
```

L'IOR est lu depuis le fichier créé par le serveur.

```
// 2. Conversion en objet CORBA
org.omg.CORBA.Object obj = orb.string_to_object(ior);
```

L'ORB convertit la chaîne IOR en référence d'objet CORBA générique.

```
// 3. Narrowing vers le type spécifique
Hello helloRef = HelloHelper.narrow(obj);
```

Le `narrowing` est essentiel : il convertit l'objet générique en type `Hello` tout en vérifiant que l'objet distant implémente bien cette interface.

```
// 4. Invocation de la méthode distante
String response = helloRef.sayHello();
System.out.println("Response from Server: " + response);
```

L'appel de méthode est transparent : le client ne "voit" pas que l'objet est distant.

5.3 Rôle des classes générées

5.3.1 HelloHelper - Utilitaires CORBA

La classe `HelloHelper` fournit des méthodes statiques essentielles :

```
public static hello.Hello narrow (org.omg.CORBA.Object obj)
  {
    if (obj == null)
      return null;
    else if (obj instanceof hello.Hello)
      return (hello.Hello)obj;
    else if (!obj._is_a (id ()))
      throw new org.omg.CORBA.BAD_PARAM ();
    else
    {
      org.omg.CORBA.portable.Delegate delegate =
      ↪  ((org.omg.CORBA.portable.ObjectImpl)obj)._get_delegate ();
      hello._HelloStub stub = new hello._HelloStub ();
      stub._set_delegate(delegate);
      return stub;
    }
  }
```

5.3.2 HelloPOA - Classe de base serveur

`HelloPOA` est la classe abstraite que notre servant hérite :

```
public abstract class HelloPOA extends org.omg.PortableServer.Servant
 implements hello.HelloOperations, org.omg.CORBA.portable.InvokeHandler
{

  // Constructors

  private static java.util.Hashtable _methods = new java.util.Hashtable ();
  static
  {
    _methods.put ("sayHello", new java.lang.Integer (0));
  }

  public org.omg.CORBA.portable.OutputStream _invoke (String $method,
                                org.omg.CORBA.portable.InputStream in,
                                org.omg.CORBA.portable.ResponseHandler $rh)
  {
    org.omg.CORBA.portable.OutputStream out = null;
    java.lang.Integer __method = (java.lang.Integer)_methods.get ($method);
    if (__method == null)
      throw new org.omg.CORBA.BAD_OPERATION (0,
      ↪  org.omg.CORBA.CompletionStatus.COMPLETED_MAYBE);

    switch (__method.intValue ())
    {
       case 0:  // hello/Hello/sayHello
       {
         String $result = null;
         $result = this.sayHello ();
         out = $rh.createReply();
```

```
29          out.write_string ($result);
30          break;
31        }
32
33        default:
34          throw new org.omg.CORBA.BAD_OPERATION (0,
              ↪ org.omg.CORBA.CompletionStatus.COMPLETED_MAYBE);
35      }
36
37      return out;
38    } // _invoke
39
40    // Type-specific CORBA::Object operations
41    private static String[] __ids = {
42      "IDL:hello/Hello:1.0"};
43
44    public String[] _all_interfaces (org.omg.PortableServer.POA poa, byte[] objectId)
45    {
46      return (String[])__ids.clone ();
47    }
48
49    public Hello _this()
50    {
51      return HelloHelper.narrow(
52      super._this_object());
53    }
```

Responsabilités :

— **Dispatch** : Acheminer les appels vers les bonnes méthodes

— **Unmarshalling** : Désérialiser les paramètres

— **Marshalling** : Sérialiser les résultats

5.3.3 _HelloStub - Proxy client

Le stub côté client encapsule la logique de communication :

```
1  public class _HelloStub extends org.omg.CORBA.portable.ObjectImpl implements
   ↪ hello.Hello
2  {
3
4    public String sayHello ()
5    {
6              org.omg.CORBA.portable.InputStream $in = null;
7              try {
8                  org.omg.CORBA.portable.OutputStream $out = _request ("sayHello",
                   ↪ true);
9                  $in = _invoke ($out);
10                 String $result = $in.read_string ();
11                 return $result;
12             } catch (org.omg.CORBA.portable.ApplicationException $ex) {
13                 $in = $ex.getInputStream ();
14                 String _id = $ex.getId ();
15                 throw new org.omg.CORBA.MARSHAL (_id);
16             } catch (org.omg.CORBA.portable.RemarshalException $rm) {
17                 return sayHello (       );
```

```
18              } finally {
19                  _releaseReply ($in);
20              }
21    } // sayHello
22
23    // Type-specific CORBA::Object operations
24    private static String[] __ids = {
25      "IDL:hello/Hello:1.0"};
26
27    public String[] _ids ()
28    {
29      return (String[])__ids.clone ();
30    }
31
32    private void readObject (java.io.ObjectInputStream s) throws java.io.IOException
33    {
34       String str = s.readUTF ();
35       String[] args = null;
36       java.util.Properties props = null;
37       org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init (args, props);
38     try {
39       org.omg.CORBA.Object obj = orb.string_to_object (str);
40       org.omg.CORBA.portable.Delegate delegate = ((org.omg.CORBA.portable.ObjectImpl)
         ↪  obj)._get_delegate ();
41       _set_delegate (delegate);
42     } finally {
43       orb.destroy() ;
44     }
45    }
46
47    private void writeObject (java.io.ObjectOutputStream s) throws java.io.IOException
48    {
49       String[] args = null;
50       java.util.Properties props = null;
51       org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init (args, props);
52     try {
53       String str = orb.object_to_string (this);
54       s.writeUTF (str);
55     } finally {
56       orb.destroy() ;
57     }
58    }
59  } // class _HelloStub
60
```

Flux d'exécution :

1. Création d'un flux de sortie pour les paramètres
2. Invocation distante via `_invoke()`
3. Lecture du résultat depuis le flux d'entrée
4. Gestion des exceptions et du remarshalling
5. Libération des ressources

6 Conclusion

6.1 Synthèse des apprentissages

Ce travail pratique nous a permis de maîtriser les concepts fondamentaux de CORBA et de la programmation distribuée. Les compétences acquises incluent :

6.1.1 Compétences techniques

1. **Architecture CORBA**
   - Compréhension du rôle de l'ORB dans la communication distribuée
   - Maîtrise du POA et de ses politiques de gestion d'objets
   - Utilisation des IOR pour la localisation d'objets distants
2. **Langage IDL**
   - Définition d'interfaces indépendantes du langage
   - Compilation IDL vers Java avec idlj
   - Compréhension du mapping IDL-Java
3. **Développement distribué**
   - Implémentation d'un serveur d'objets CORBA
   - Développement d'un client CORBA
   - Gestion du cycle de vie des objets distants