

<http://www.corba.org>

# CORBA Programming



# What is CORBA?

- **Common Object Request Broker Architecture**
  - An *industry standard* developed by OMG to help in distributed programming
  - A *specification* for creating and using distributed objects
  - A *tool* for enabling multi-language, multi-platform communication
- A CORBA based-system is a collection of objects that isolates the requestors of services (*clients*) from the providers of services (*servers*) by an encapsulating interface

# Who should use CORBA?

- “The architecture and specifications described in the Common Object Request Broker Architecture and Specifications book are aimed at software designers and developers who want to produce applications that **comply with OMG standards** for the Object Request Broker (ORB)” [OMG].
- “The benefit of compliance is, in general, to be able to produce **interoperable applications** that are based on distributed, interoperating objects” [OMG].

# CORBA Features (1)

- **Heterogeneity**

- Hardware devices, Operating System, Network protocols, Programming languages

- **Object Orientation**

- Encapsulation, Polymorphism, Inheritance, Instantiation

- **Dynamic binding and typing**

# CORBA Features (2)

## ■ **Transparencies**

- *Location transparency* : the physical address of a server is masked
- *Access transparency* : the access method is masked
- *Data transparency* : the different data representations are masked

...

# CORBA Evolution

[http://www.corba.org/history\\_of\\_corba.htm](http://www.corba.org/history_of_corba.htm)

- CORBA 1.0 (October 1991)
  - CORBA Object model
  - Interface Definition Language (IDL)
  - Core Dll and Interface Repository
  - single language mapping for the C
- CORBA 1.1 (February 1992)
  - The first widely published version
  - Interfaces for the Basic Object Adapter
  - Clarified some ambiguities

# CORBA Evolution

- CORBA 2.3 (June 1999)
  - COM/CORBA Part A and B
  - Portability IDL/Java
  - Objects by value
  - Java to IDL Language Mapping
  - IDL to Java Language Mapping
  - C++ Language Mapping

# CORBA Evolution

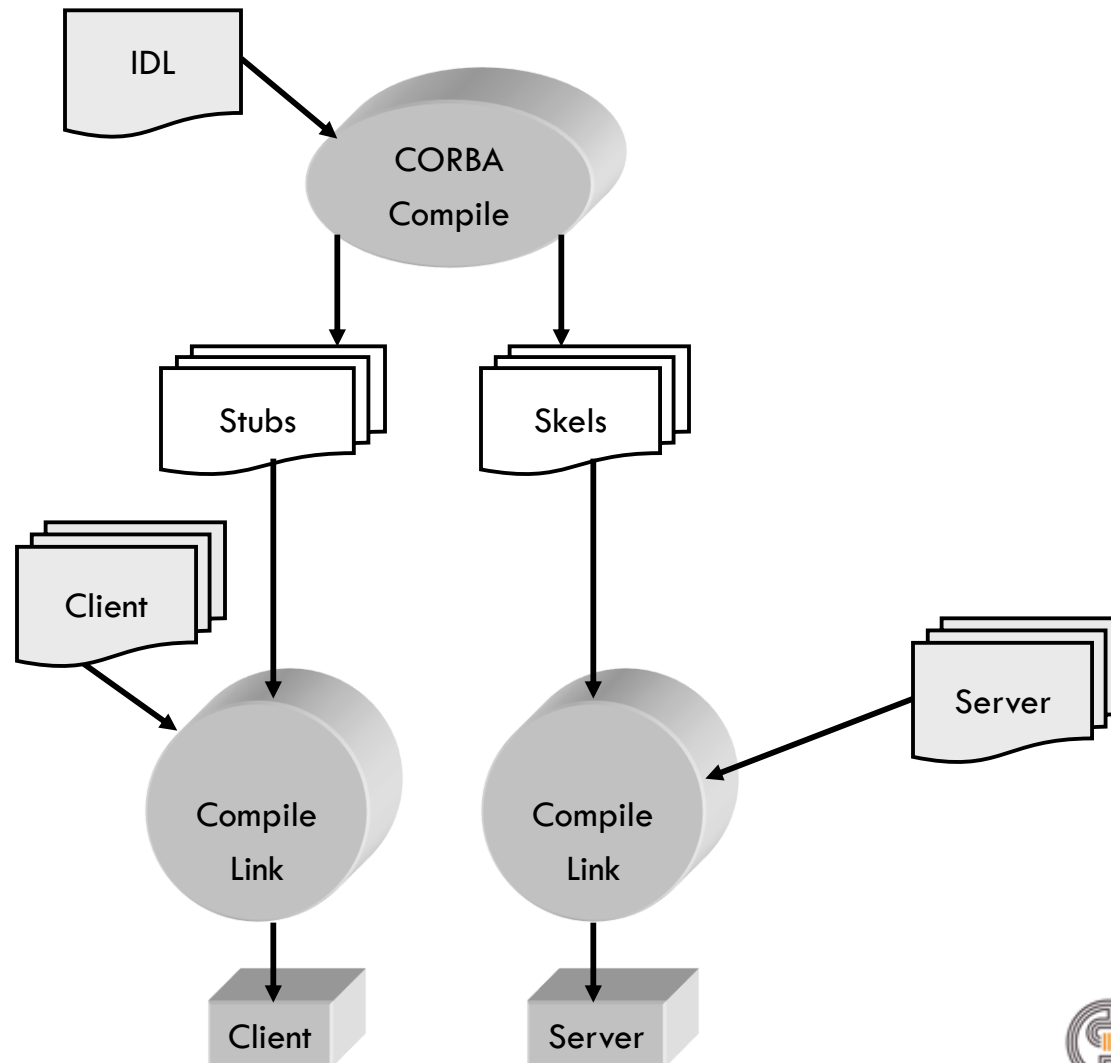
- CORBA 3.0 (June 2002)
  - Java and Internet Integration
    - Objects Passable by Value (first in ver 2.3)
    - Java-to-IDL Mapping (first in ver 2.3)
    - Interoperable Name Service (first in ver 2.4)
    - Firewall Specification
  - Quality of Service Control
- **CORBA 3.3 (November 2012)**



# CORBA vs. Java RMI

- **CORBA was designed for language independence** whereas RMI was designed for a single language where objects run in a homogeneous environment
- CORBA interfaces are defined in **IDL**, while RMI interfaces are defined in Java
- CORBA objects are not garbage collected because they are language independent and they have to be consistent with languages that do not support garbage collection, on the other hand RMI objects are garbage collected automatically

# Development Steps



# CORBA Objects

They are different from typical programming objects in three ways:

- CORBA objects can run on any platform
- CORBA objects can be located anywhere on the network
- CORBA objects can be written in any language that has IDL mapping

# Object Request Broker (1)

## Responsibilities

- Object location transparency
  - Find the object implementation for the request
- Object activation
  - Prepare the object implementation to receive the request
- Communication
  - Communicate the data making up the request

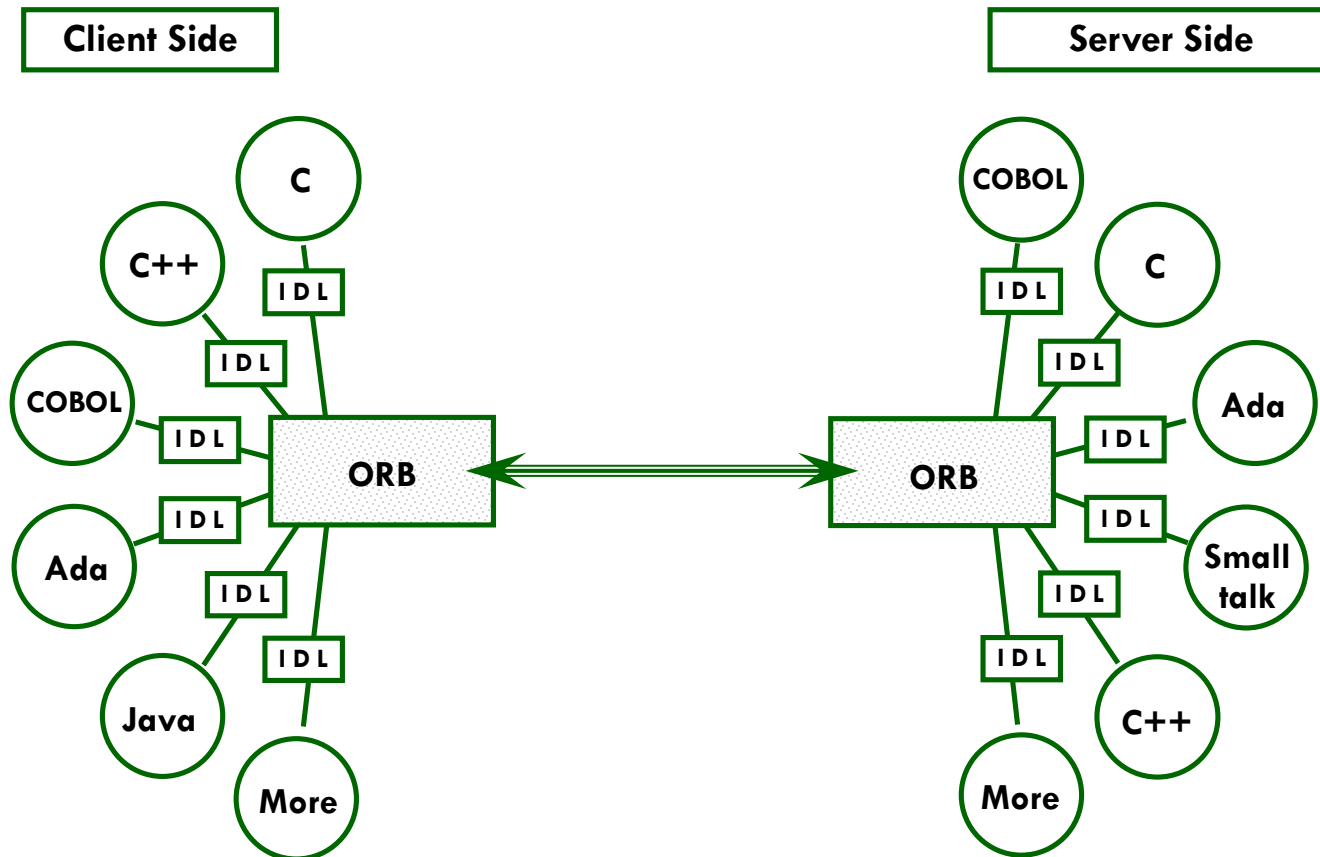
# Object Request Broker (2)

- Both the client and the object implementation are isolated from the ORB by an IDL interface.
- All requests are managed by the ORB, which means that invocation of a CORBA object is passed to an ORB
- CORBA objects implemented in different ORBs from different vendors should be able to communicate with each other because all CORBA compliant ORBs are able to interoperate via IIOP (Internet Inter-ORB Protocol)

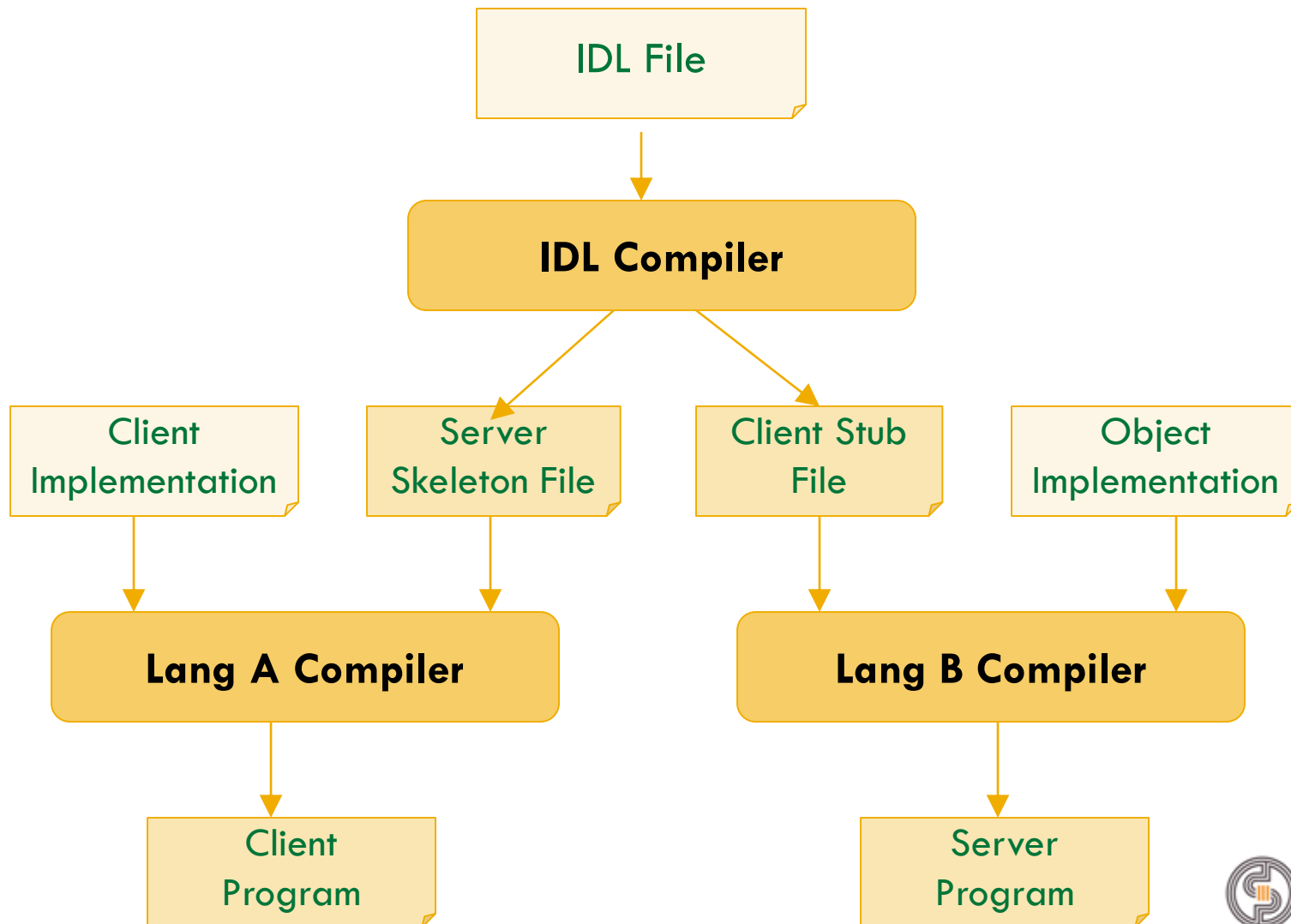
# Interface Definition Language (1)

- Separates the Interface from the Implementation
- Multiple-inheritance, strongly typed, public interface specification language
- Independent of any particular language and compiler
- Mappings will be provided for many languages and compilers
- Enables Interoperability
- **Not a programming language**

# Interface Definition Language (2)



# Interface Definition Language (3)





# Interface Definition Language (4)

## IDL Compiler

- It will accept as input an IDL file written using any text editor (fileName.idl)
- It generates the stub and the skeleton code in the target programming language (e.g., Java stub and C++ skeleton)
- The stub is given to the client as a tool to describe the server functionality and the skeleton file is implemented at the server

# Interface Definition Language (5)

IDL	Java
module	package
interface	Interface
operation	method
exception	exception

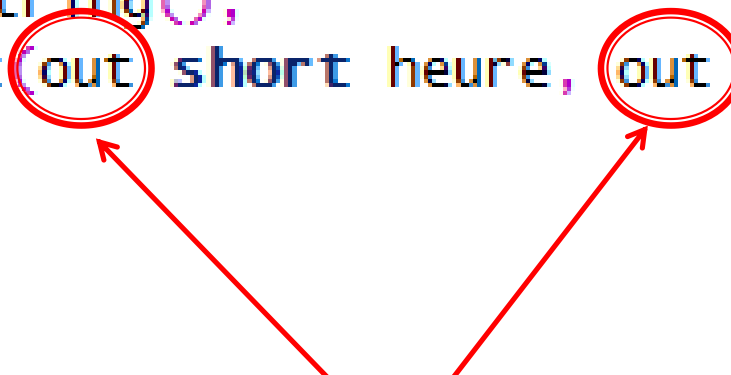
Type IDL	Type Java
boolean	boolean
char/wchar	char
octet	byte
short / unsigned short	short
long / unsigned long	int
long long / unsigned long long	long
float	float
double	double
string/wstring	String

# Steps to Implement “Clock”

1. Define an interface using IDL
2. Use the IDL compiler
3. Implement the Hello object
4. Implement the server
5. Implement the client
6. Compile
7. Run

# 1. Define Interface in IDL

```
interface Horloge {  
    // Corps de l'interface : attributs et méthodes  
    string get_time_string();  
    void get_time_int(out short heure, out short min);  
};
```



**in:** parameter is passed from the client to the server

**out:** parameter is passed from the server to the client

**inout:** parameter is passed in both directions

# Implement the Hello Object

## 2. Use the IDL compiler

```
idlj -fserver -oldImplBase Horloge.idl
```

## 3. Implement the HorlogeServant

Compliant with old versions (use of BOA)

```
import java.io.*;
import java.util.Date;

public class HorlogeServant extends _HorlogeImplBase {

    public String get_time_string() {
        return((new Date()).toLocaleString());
    }

    public void get_time_int(org.omg.CORBA.ShortHolder heure, org.omg.CORBA.ShortHolder min) {
        Date date = new Date();

        heure.value = (short) date.getHours();
        min.value = (short) date.getMinutes();
    }
}
```

# Implement the Server (1)

## 4. Implement the Server (first version)

```
import java.io.*;

import org.omg.CORBA.*;

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

            // Creation instance
            HorlogeServant objet = new HorlogeServant();
            // Enregistrement aupres du BOA
            orb.connect(objet);

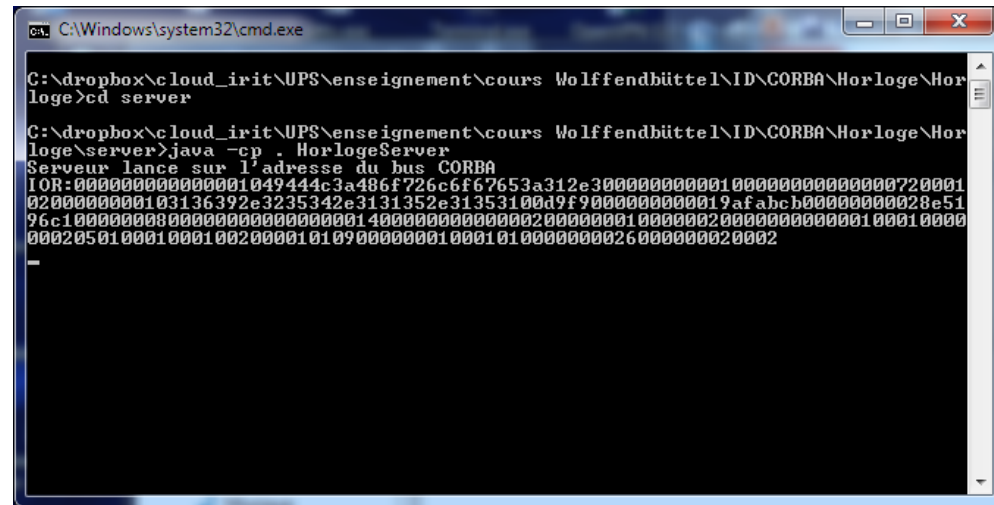
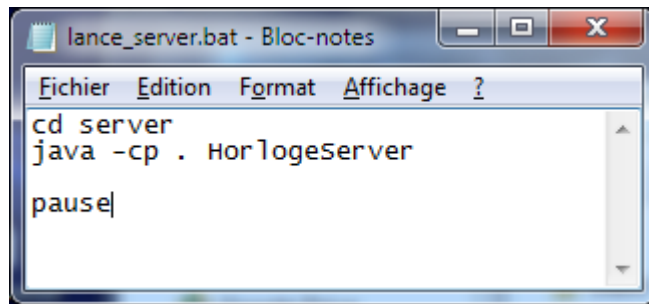
            // lancement du serveur
            System.out.println("Serveur lance sur l'adresse du bus CORBA \n"+ orb.object_to_string(objet));
            // GO
            java.lang.Object sync = new java.lang.Object();
            synchronized (sync) {
                sync.wait();
            }
        }
        catch(Exception e) {
            System.err.println("ERROR: " + e.getMessage());
            e.printStackTrace(System.out);
        }
    }
}
```

# Compile and Run

## 6. Compile

```
javac *.java  
pause
```

## 7. Run



# Implement the Client (1)

## 5. Implement the Client (first version)

```
import java.io.*;
import java.util.*;
import org.omg.CosNaming.*;
import org.omg.CORBA.*;

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

            /* Utilisation d'object to string */
            // ATTENTION : IL FAUT CHANGER L'ADRESSE DU BUS ET RECOMPILER !!!
            org.omg.CORBA.Object obj = orb.string_to_object("IOR:0000000000000000..");

            // La classe Helper fournit une fonction "narrow" pour obtenir un
            // objet sur lequel on peut invoquer les methodes
            Horloge horloge = HorlogeHelper.narrow(obj);

            // On invoque les méthodes sur l'objet distant comme s'il s'agissait
            // d'un objet local
            System.out.println(horloge.get_time_string());
            ShortHolder heure = new ShortHolder();
            ShortHolder min = new ShortHolder();
            horloge.get_time_int(heure, min);
            System.out.println(" "+heure.value+": "+min.value);
        }
        catch (Exception e)
        {
            System.out.println("Une exception a été levée");
        }
    }
}
```

Not really interesting



# Interoperable Object Reference(IOR)

- An ORB must create an IOR whenever an object reference is passed across ORB's
- Includes ORB's internal object reference and addressing information

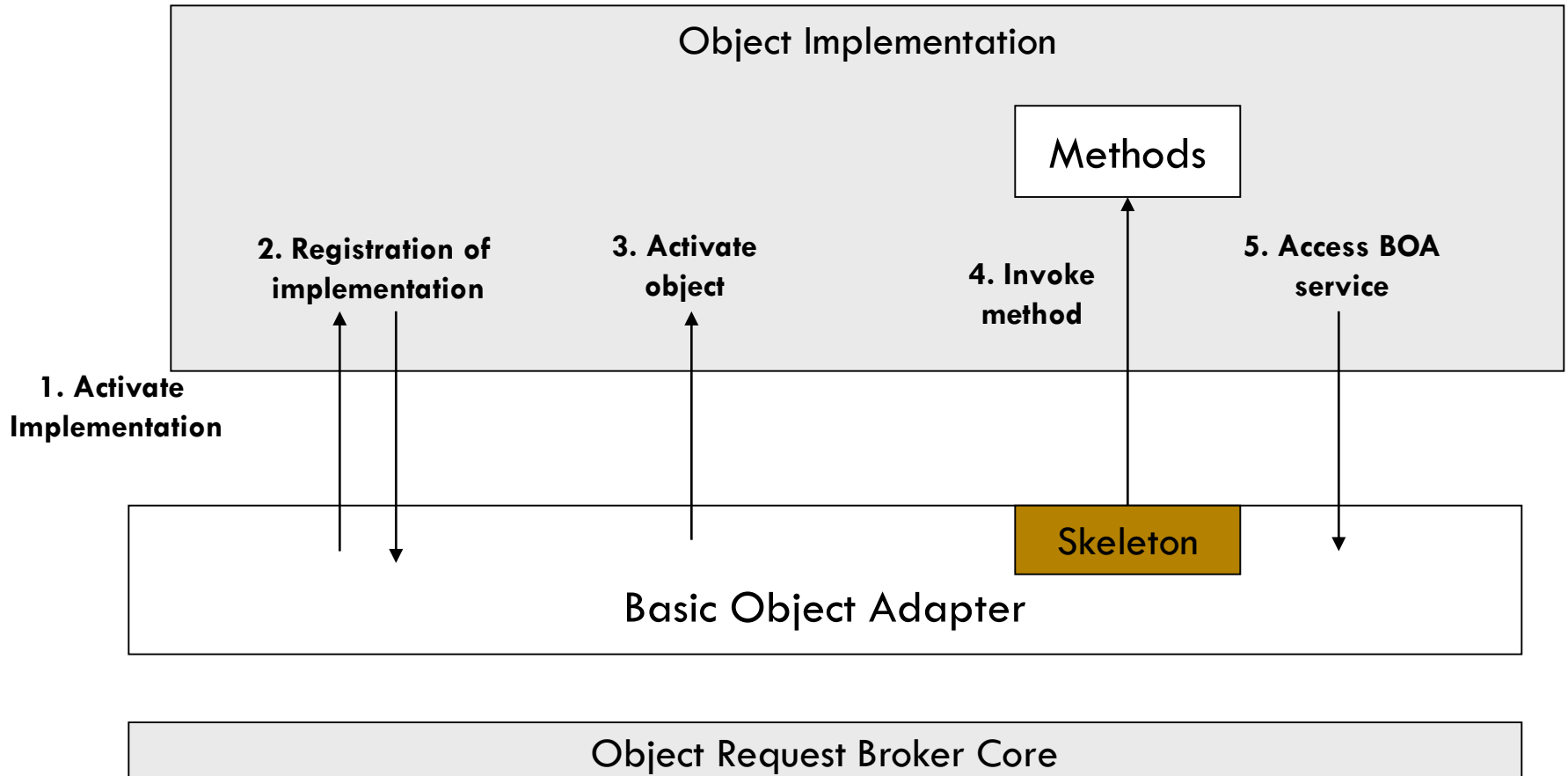
# IOR Example

- IOR:00000000000000001049444c3a5472697669616  
c3a312e3000000000001000000000000007c000102  
000000000d3135322e38312e342e31313000000048  
0000000025abacab3131303033383632313336005f  
526f6f74504f410000cafebab3bd5b8780000000  
00000000000000001000000010000002c0000000000  
010001000000004000100200001010900010100050  
1000100010109000000020001010005010001

# Object Adapter

- Different kind of object implementations -
  - objects residing in their own process and requiring activation.
  - others not requiring activation.
  - or some residing in same process as ORB.
- **OA helps the ORB to operate with different type of objects.**
  - Most widely used OA - BOA (Basic OA)
  - Recently standardized - POA (Portable OA)

# Object Implementation Invocation Scenario



# **tnameserv – Naming Service**

- Transient naming service is an implementation of the COS (Common Object Service) Naming Service Specification

**tnameserv** –ORBInitialPort *port* (900 by default)

<http://docs.oracle.com/javase/7/docs/technotes/tools/share/tnameserv.html>

# Implement the Server (2)

## 4. Implement the Server (second version)

```
import java.io.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;

public class HorlogeServer {
    public static void main(String args[]) {
        try{
            // create and initialize the ORB
            ORB orb = ORB.init(args, null);
            // create the servant and register it with the ORB
            HorlogeServant h = new HorlogeServant();
            orb.connect(h);
            // get the root naming context
            org.omg.CORBA.Object objRef =
                orb.resolve_initial_references("NameService");
            NamingContext ncRef = NamingContextHelper.narrow(objRef);
            // Bind the object reference in naming
            NameComponent nc = new NameComponent("Horloge", " ");
            NameComponent path[] = {nc};
            ncRef.rebind(path, h);
            System.out.println("Server started...");
            // wait for invocations from clients
            java.lang.Object sync = new java.lang.Object();
            synchronized(sync){
                sync.wait();
            }
        } catch(Exception e) {
            System.err.println("ERROR: " + e.getMessage());
            e.printStackTrace(System.out);
        }
    }
}
```

# Implement the Client (2)

## 5. Implement the Client (second version)

```
import java.io.*;
import java.util.*;
import org.omg.CosNaming.*;
import org.omg.CORBA.*;

public class HorlogeClient {
    public static void main(String argv[]) {
        try {
            // create and initialize the ORB
            ORB orb = ORB.init(argv, null);
            // get the root naming context
            org.omg.CORBA.Object objRef =
                orb.resolve_initial_references("NameService");
            NamingContext ncRef = NamingContextHelper.narrow(objRef);
            NameComponent nc = new NameComponent("Horloge", " ");
            // Resolve the object reference in naming
            NameComponent path[] = {nc};

            // La classe Helper fournit une fonction "narrow" pour obtenir un
            // objet sur lequel on peut invoquer les methodes
            Horloge horloge = HorlogeHelper.narrow(ncRef.resolve(path));

            // On invoque les méthodes sur l'objet distant comme s'il s'agissait
            // d'un objet local
            System.out.println(horloge.get_time_string());
            ShortHolder heure = new ShortHolder();
            ShortHolder min = new ShortHolder();
            horloge.get_time_int(heure,min);
            System.out.println(""+heure.value+":"+min.value);
        }
        catch (Exception e)
        {
            System.out.println("Une exception a été levée");
        }
    }
}
```

# IIOP- Internet Inter ORB Protocol

- It is a specialized form of GIOP (General Inter ORB Protocol) for TCP/IP networks.
- **IIOP specifies how GIOP messages will be exchanged over TCP/IP network**
- An ORB must support IIOP in order to be considered compliant with CORBA 2.0.
- It consists primarily of the specification for the IIOP IOR, which contains the host name and the port number.