

CORBA

- CORBA facilita lo sviluppo di sistemi distribuiti fornendo
 - Una infrastruttura per far comunicare oggetti in un sistema distribuito
 - Un set di servizi utili
 - Un supporto che permette ad applicazioni implementate usando vari linguaggi di interoperare

Cosa è CORBA

- Framework per costruire sistemi ad oggetti distribuiti
 - Indipendente dal linguaggio di programmazione
 - Indipendente dalla piattaforma (SO e hw)
 - Nasconde la complessità del distribuito
- Fornisce astrazioni e servizi concreti
 - Separazione tra interfaccia ed implementazione
 - Servizi per oggetti (Naming Service, etc.)
- Standard definito da Object Management Group (OMG) dal 1989 - sito www.omg.org
 - L'ultima specifica è la versione 3.0.2 di Dicembre 2002

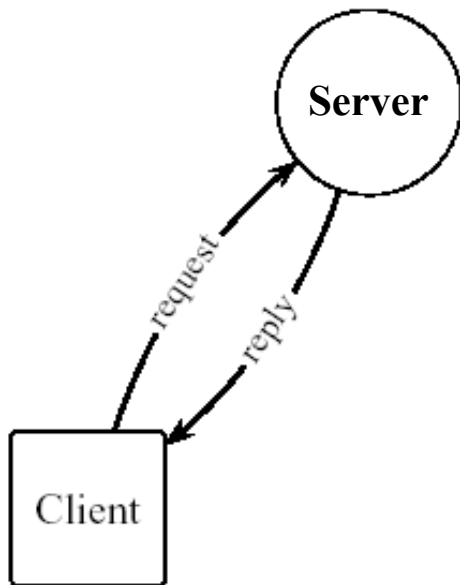
Cosa è CORBA

- ORB: Object Request Broker
 - Gestisce l'accesso remoto ad oggetti
- CORBA: Common ORB Architecture
 - Oggetti remoti sono accessibili da un programma come se fossero locali
 - **Location Transparency**: il programma non conosce la posizione degli oggetti
 - Usa il paradigma client-server

Implementazioni di ORB

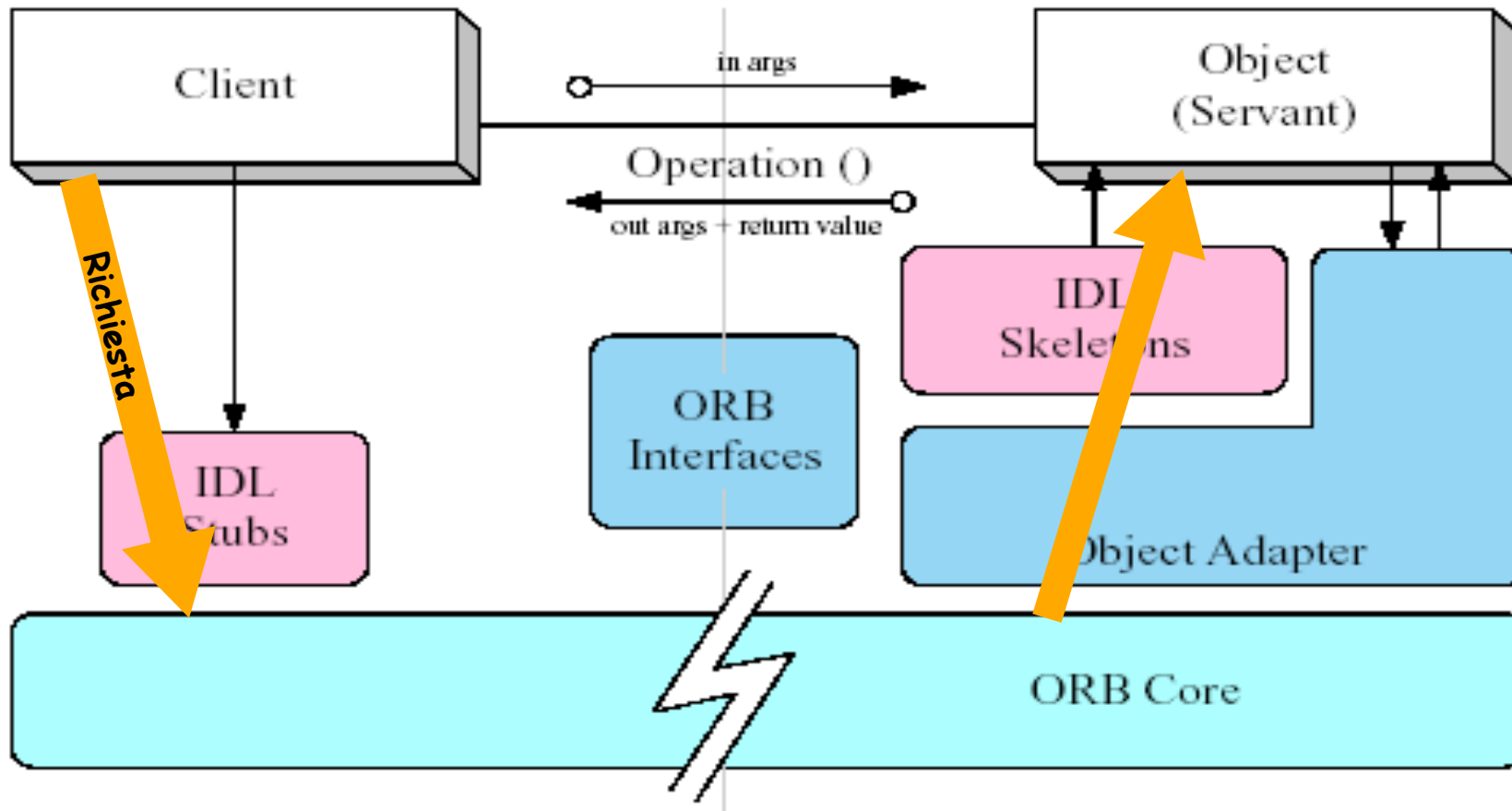
- Free
 - Orbacus [www.orbacus.com]
 - C++ e Java
 - Servizi: Naming e Event
 - JacORB [www.jacorb.org]
 - Java
 - Servizi: Naming, Event, Transaction, Concurrency, Collection
 - TAO [www.cs.wustl.edu/~schmidt/TAO.html]
 - C++
 - Ricco di Servizi
 - OmniORB [omniorb.sourceforge.net]
 - C++ e Python
 - MICO [www.mico.org]
 - C++
 - Servizi: Naming, Event, Trader, Security

Paradigma Client-Server



- Un oggetto *client* manda una richiesta tramite un messaggio ad un oggetto *server*
- Il client non ha bisogno di sapere dove è l'oggetto server
- Il client sa che tipo di messaggio mandare, poiché conosce l'*interfaccia* dell'oggetto server

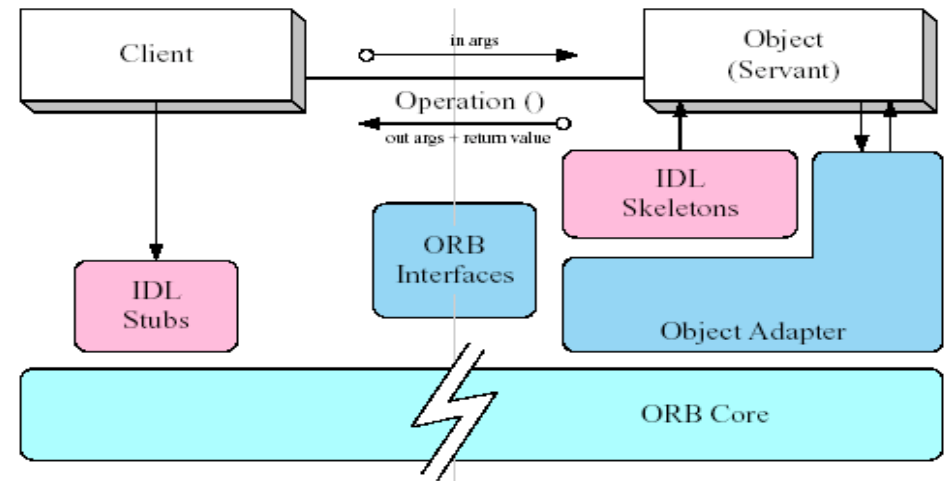
Architettura CORBA



Sviluppo di una applicazione

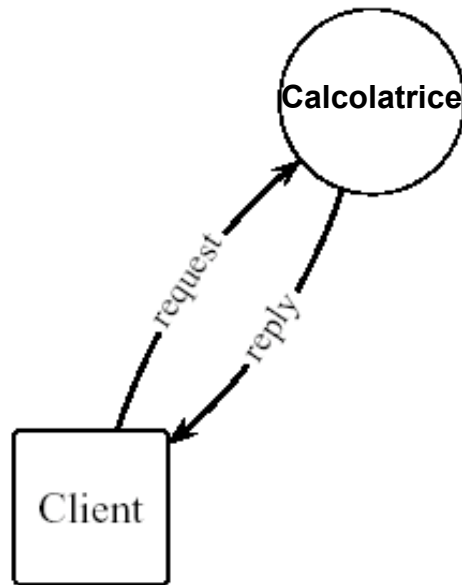
- Per progettare una applicazione CORBA bisogna basarsi sul paradigma Client-Server
- Per implementare l'applicazione occorre scrivere:

- Interfaccia IDL
- Codice Servant
- Codice Server
- Codice Client



- Deployment: mettere ogni cosa al suo posto

Applicazione: Calcolatrice



- Progettazione: oggetti client possono usare un oggetto **Calcolatrice** (servant) che è in grado di fare somme
 - I client mandano richieste tramite messaggi
- I client non hanno bisogno di sapere su quale host si trova l'oggetto **Calcolatrice**
- I client non conoscono come è implementato l'oggetto **Calcolatrice**, ma conoscono la sua interfaccia

Costruire l'applicazione

1. Implementare interfacce IDL dei servant (`Calcolatrice.idl`)
2. Generare stub e skeleton dalle interfacce IDL tramite un compilatore (che produce anche classi `Helper` e `Holder`)
 - Esempio (Java):

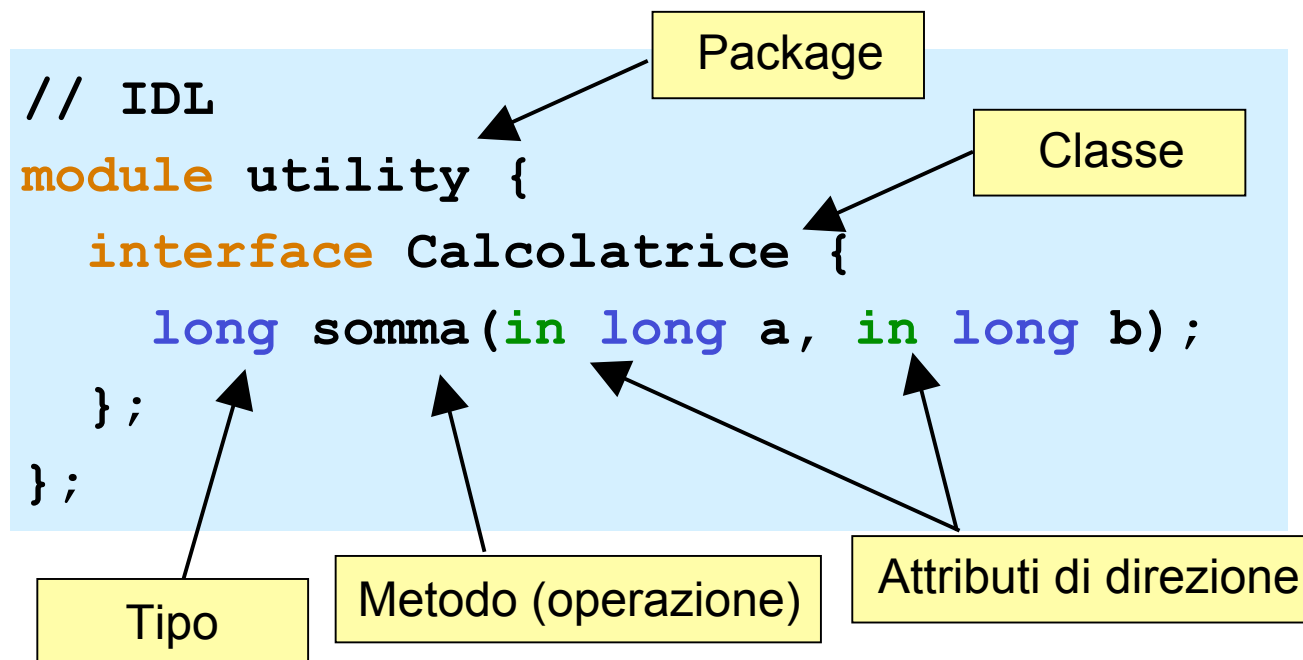
```
> idlj -fall Calcolatrice.idl
```
3. Implementare classi servant (`CalcolatriceImpl.java`)
4. Implementare server (`CalcolatriceServer.java`)
5. Compilare
 - Esempio (Java):

```
> javac CalcolatriceImpl.java  
> javac CalcolatriceServer.java
```
6. Implementare client (`CalcolatriceClient.java`)
7. Compilare
 - Esempio (Java):

```
> javac CalcolatriceClient.java
```

IDL: Calcolatrice.idl

- Tramite l'interfaccia IDL, i client conoscono quali operazioni sono disponibili su un servant e come devono essere invocate



Compilazione IDL

- Il compilatore IDL (per Java è `idlj`) riceve in ingresso `Calcolatrice.idl` e produce una directory `utility` contenente

	Java vers. 1.3	Java vers. 1.4
Stub	<code>_CalcolatriceStub.java</code>	
Skeleton	<code>_CalcolatriceImplBase.java</code>	<code>CalcolatricePOA.java</code>
Helper	<code>CalcolatriceHelper.java</code>	
Holder	<code>CalcolatriceHolder.java</code>	
Interface	<code>Calcolatrice.java</code>	
Signature	<code>CalcolatriceOperations.java</code>	

- La J2SE contiene `idlj` e la libreria `org.omg.CORBA`

Servant: CalcolatriceImpl.java

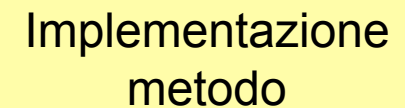
Java vers. 1.3

```
package server;
import utility.*;
public class CalcolatriceImpl extends _CalcolatriceImplBase {
    public CalcolatriceImpl() {
        super(); }
    public int somma(int a, int b) {
        return a + b; }
}
```

Skeleton



Implementazione
metodo



Java vers. 1.4

```
package server;
import utility.*;
public class CalcolatriceImpl extends CalcolatricePOA {
    private ORB orb;
    public void setORB(ORB orb_val) {
        orb = orb_val; }
    public int somma(int a, int b) {
        return a + b; }
}
```

Server: `CalcolatriceServer.java`

Operazioni principali del server (Java vers. 1.3):

- Inizializza ORB con il numero della porta usata

```
org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);
```

- Crea oggetto servant
- Registra oggetto servant nell'ORB

```
orb.connect(calc);
```

- Pubblica nome oggetto su Naming Server, file oppure URL
- Attende invocazioni

Server: CalcolatriceServer.java

```
// Java
package server;
import utility.*;
import java.io.*;
public class CalcolatriceServer {
    public static void main(String args[]) {
        org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);
        // crea un oggetto Calcolatrice
        CalcolatriceImpl calc = new CalcolatriceImpl();
        orb.connect(calc);
        // pubblica oggetto (su Naming Server)
        // attende invocazioni
        Object sync = new Object();
        synchronized (sync) {
            sync.wait();
        }
    }
}
```

Inizializza ORB

Registra nell'ORB

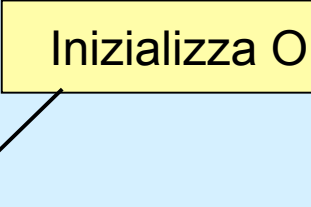
Client

Operazioni principali del client:

- Inizializza ORB
- Ottiene riferimento (detto IOR) ad oggetto servant
 - Dalla chiamata ad un altro oggetto
 - Da un Naming Service
 - Da file oppure URL
- Effettua un cast (Narrowing) del riferimento
 - Dalla stringa contenente l'IOR si passa ad un tipo `Object`
`org.omg.CORBA.Object obj = orb.string_to_object(ior);`
 - Dal tipo `Object` si passa al tipo del servant, tramite la classe `Helper`
`Calcolatrice calc = CalcolatriceHelper.narrow(obj);`

Client

```
// Java
package client;
import utility.*;
import java.io.*;
public class CalcolatriceClient {
    public static void main(String args[]) {
        org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);
        // ottiene riferimento ad oggetto remoto obj
        // narrowing
        Calcolatrice calc = CalcolatriceHelper.narrow(obj);
        int a, b;
        // input a, b
        // invoca metodo remoto
        System.out.println("a + b = " + calc.somma(a, b));
    }
}
```



Deployment

- Lato server

- Creare directory **utility**
- Creare directory **server** con servant e server
- Attivare Naming Service ed eseguire

```
> tnameserv -ORBInitialPort 1050
```

```
> java server.CalcolatriceServer -ORBInitialPort 1050
```

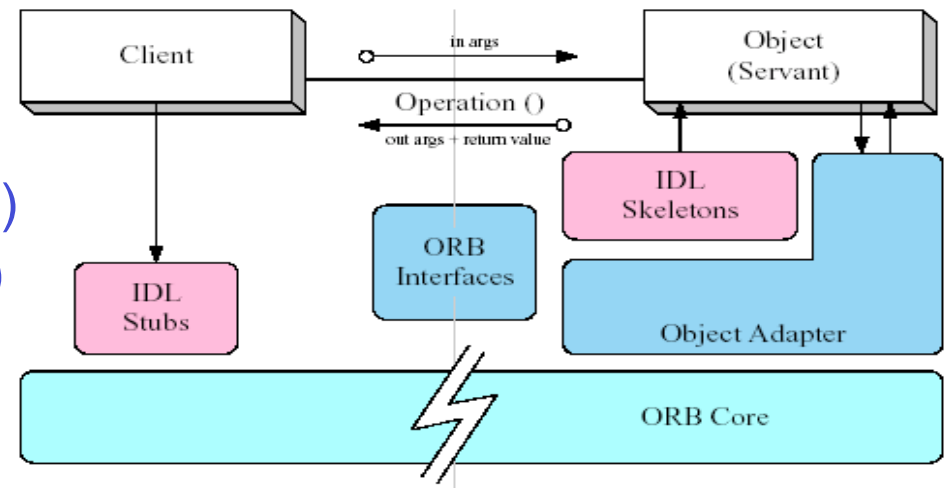
- Lato client

- Creare directory **utility**
- Creare directory **client**
- Eseguire

```
> java client.CalcolatriceClient -ORBInitialPort 1050
```

Componenti CORBA

- Il programmatore di un'applicazione CORBA deve provvedere ad implementare
 - Client e Server
 - Oggetti (Interfacce e Servant)
- L'architettura CORBA è composta da
 - Object Request Broker (ORB)
 - Stub e Skeleton
 - Dynamic Invocation Interface (DII)
 - Dynamic Skeleton Interface (DSI)
 - Object Adapter
 - Interface Repository



Oggetti

- Un oggetto è caratterizzato da:
 - Riferimento
 - Interfaccia
 - Implementazione (servant)
- Riferimento ad un oggetto (IOR)
 - Un handle che identifica un oggetto
 - Permette all'ORB di localizzare l'oggetto
 - Riferisce un singolo oggetto
 - Un oggetto può avere più riferimenti
 - E' analogo ad un puntatore in C++, e ad un riferimento Java

Interfaccia

- Costituisce il *contratto* offerto da un oggetto ai client
 - I client devono usare l'interfaccia IDL per specificare l'operazione da compiere su un oggetto
- Definisce
 - Un tipo IDL
 - Ciò che è implementato da un oggetto remoto
- Può contenere
 - Dichiarazioni di eccezioni
 - Definizioni di costanti
 - Operazioni (metodi)
 - Attributi (campi)
- Supporta ereditarietà multipla

Servant

- Implementa ed esegue operazioni
- Rappresenta uno o più oggetti CORBA
 - Non vi è un rapporto 1-1 tra oggetti CORBA e servant
- Fornisce un target per un oggetto CORBA
- Può essere istanziato su richiesta
- Vive dentro un processo server
- L'implementazione è basata su
 - Ereditarietà (il servant è ereditato da una classe CORBA)
 - Delega (un oggetto CORBA chiama metodi del servant)

Object Request Broker (ORB)

- E' responsabile del trasferimento di richieste di operazioni e dei valori di ritorno
 - Mantiene l'Interface Repository (IR)
 - Gestisce riferimenti agli oggetti
 - Localizza oggetto
 - Location transparency: il client non conosce dove l'oggetto usato risiede
 - Attiva server (se necessario)
 - Trasmette argomenti
 - Attiva servant (se necessario)
 - Aspetta il completamento dell'operazione
 - Ritorna parametri out/inout e valore di ritorno
 - Ritorna una eccezione se la richiesta fallisce

Funzionalità dell'ORB

- Client e Server devono riferire un ORB, per ottenere un riferimento usano il metodo statico

```
orb = org.omg.CORBA.ORB.init()
```

- Un riferimento ad un servant (detto Interoperable Object Reference IOR) può essere convertito in una stringa (forma standard), e dalla stringa si può passare all'IOR, attraverso i metodi dell'ORB

```
String object_to_string(in Object obj);  
Object string_to_object(in String str);
```

- Un IOR può essere compreso da qualunque ORB conforme alle specifiche OMG

Stub e Skeleton

- Stub
 - Opera da collante tra client e ORB
 - Ne viene creato uno per ogni servant
 - Trasmette gli argomenti dell'invocazione, tramite la classe Holder
- Skeleton
 - Opera da collante tra ORB e servant
 - Trasmette gli argomenti dell'invocazione
- Sono generati dalla compilazione di una definizione IDL con un compilatore IDL per il linguaggio target

IDL

- Tramite IDL, un servant dice ai client quali operazioni sono disponibili e come devono essere invocate
- Una interfaccia è mappata con un linguaggio target tramite un *compilatore IDL*
 - OMG ha standardizzato la mappatura tra IDL ed i linguaggi: C, C++, Java, COBOL, Smalltalk, Ada, Lisp, Python
 - Esistono compilatori IDL anche per: Eiffel, Modula3, Perl, Tcl, Objective-C

IDL

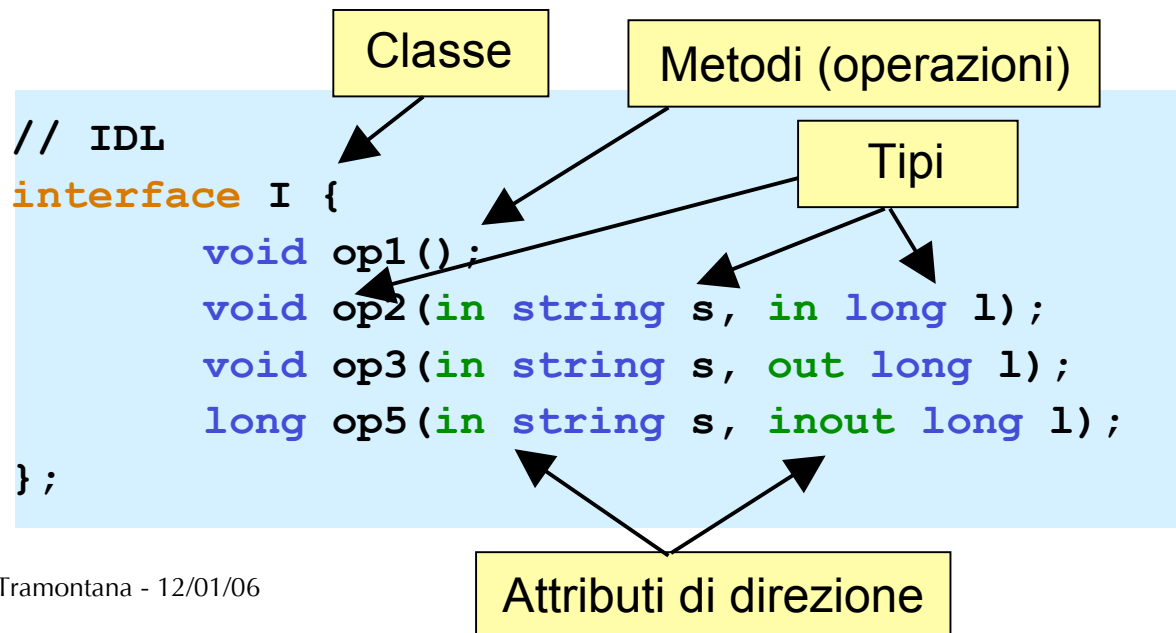
- Ha una sintassi C-like
- Un compilatore IDL riceve in ingresso un codice IDL e genera:
 - stub, skeleton, helper e holder
- Definisce:
 - Interfacce di oggetti
 - Tipi che possono essere trasmessi
 - Costanti
- E' un puro linguaggio dichiarativo orientato agli oggetti

IDL

- IDL supporta
 - Moduli
 - Interfacce
 - Operazioni
 - Attributi
 - Ereditarietà
 - Array, struct, enum, union
 - Costanti
 - Eccezioni
- IDL **non** supporta
 - Puntatori
 - Costruttori o distruttori
 - Overloading ed overriding delle operazioni
 - Costrutti di controllo

IDL: Operazioni

- Parametri possono essere **in**, **out** o **inout**
- Singolo valore di ritorno o **void**
- Parametri e risultati sono passati per valore (tuttavia, non si passano oggetti ma i loro riferimenti)
- Non si può avere overloading di metodi



IDL: Corrispondenza di Tipi

Tipo IDL	Tipo Java	Dimensione
boolean	boolean	Non specificato
char	char	8 bit
octet	byte	8 bit
string	java.lang.String	variabile
short	short	16 bit
long	int	32 bit
float	float	Vedi IEEE
...

IDL: Eccezioni

Sono dichiarate con la clausola **raises**

```
// IDL
module M {
    interface I {
        exception NotPermitted { string reason; };
        exception NoSuchFile {};
        void deleteFile(in string name) raises
            (NotPermitted, NoSuchFile);
    };
};
```

IDL: Oneway

- Le operazioni possono essere dichiarate **oneway**
- I messaggi possono non arrivare
- I client non si bloccano
- Non vi sono: valori di ritorno e parametri **out** e **inout**
- Non vi sono eccezioni definite dall'utente

```
// IDL
interface I {
    oneway void eventHint(in any evt);
};
```

IDL: Ereditarietà

- Le interfacce possono derivare da qualunque numero di altre interfacce
- Operazioni ed attributi non possono essere ridefiniti

```
// IDL
interface A {
    void opA();
};
interface B {
    void opB();
};
interface C : A, B {
    void opC(); // OK
    void opA(); // Errore: si scontra con opA ereditata
};
```

IDL: Attributi

```
// IDL
interface VolumeControl {
    attribute float level;
    readonly attribute string name;
};
```

Interfacce CORBA

- La Dynamic Invocation Interface (DII)
 - Fornisce un mezzo per invocare dinamicamente oggetti che non erano conosciuti a design-time
 - Il client ottiene l'interfaccia di un oggetto e costruisce l'invocazione
- La Dynamic Skeleton Interface (DSI)
 - Permette all'ORB di recapitare chiamate ad oggetti che non hanno uno skeleton, poiché a design-time non era stato definito il tipo
- L'Interface Repository (IFR)
 - Contiene informazioni su tipi IDL (interfacce)
 - Fornisce informazioni sui tipi, necessarie per formulare richieste che usano la DII
- L'ORB interface fornisce accesso ai servizi Naming Server, Trader Server, etc.

Object Adapter e Policy

- Fornisce l'API usata dai server per registrare le implementazioni dei loro oggetti
- E' lo strato tra l'ORB Core e lo skeleton IDL
- Fornisce policy per specificare QoS, ovvero per la gestione delle risorse
 - Lifespan definisce se gli oggetti creati devono essere *Transient* (durare solo per la vita del server) o *Persistent*
 - *ServantRetention* definisce la durata dell'attivazione di un oggetto che può essere pari alla durata di un metodo (*Non-Retain*) o più lunga (*Retain*)
 - *IdAssignment* definisce se il sistema o l'utente forniscono l'*ObjectId*
 - *Request* definisce la mappatura tra oggetti e servant

Interoperabilità ORB

- Gli ORB necessitano di un linguaggio comune per interoperare
 - General Inter ORB Protocol (GIOP) specifica
 - Rappresentazione dati comune, come IDL è rappresentato a basso livello
 - Formato dei messaggi scambiati (request, reply, etc.)
 - Protocollo di scambio messaggi (gestione connessioni, etc.)
 - IIOP è una mappatura del GIOP per TCP/IP
 - Specifica come messaggi GIOP vengono scambiati su TCP/IP

IOR

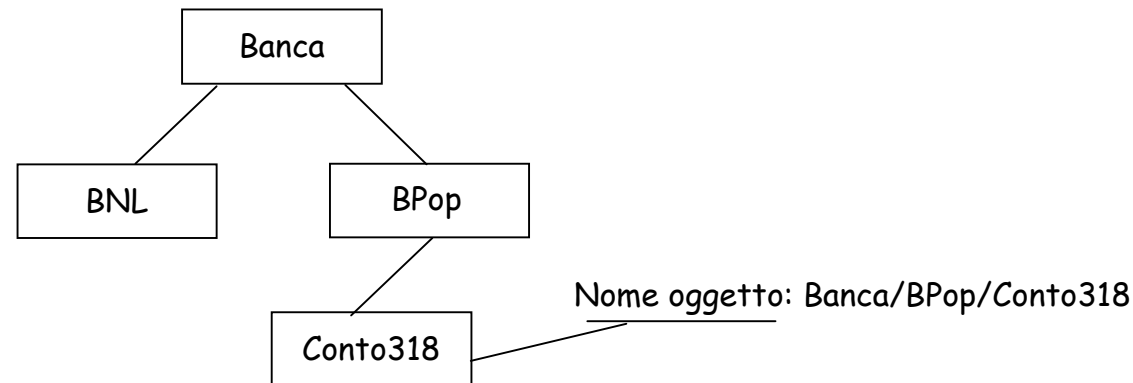
- I diversi ORB hanno bisogno di identificare gli oggetti, i loro tipi, i protocolli supportati ed i servizi disponibili
- IOR sono introdotti per questi compiti
- IOR contengono un identificatore di tipo ed il protocollo supportato
- Sono creati dai riferimenti agli oggetti
- Possono essere trasformati in stringhe (è possibile fare l'operazione inversa)

Servizi CORBA

- Sono stati specificati 15 servizi che estendono e completano le funzionalità dell'ORB e forniscono supporto per qualunque applicazione distribuita
 - Naming Service
 - Permette di far conoscere gli oggetti di una applicazione
 - Concurrency Control Service
 - Fornisce primitive di lock, unlock, etc.
 - Event Service
 - Supporta comunicazioni asincrone molti a molti
 - Transaction Service
 - Fornisce supporto per le transazioni (commit, abort, etc.)
 - Licensing Service
 - Security Service
 - Trader Service
 - Etc.

Naming Service

- Utile per conservare riferimenti ad oggetti
- Ha una struttura simile a quella di un file system



Naming Service

- **Concetti:**
 - *Name binding* (nome, oggetto) associa un nome con un oggetto
 - *Naming context* denota un set di name bindings con nomi unici
 - I name binding sono sempre relativi ad un naming context
 - Un nome è una sequenza di componenti, ad es:
Banca/BPop/Conto318
 - Banca e BPop sono Naming context dell'oggetto chiamato Conto318

Naming Service

- Operazioni
 - Bind: associa un nome ad un riferimento di un oggetto
 - Rebind: sostituisce il riferimento ad un oggetto ad un nome già presente
 - Bind Context: associa un nome ad un naming context
 - Rebind Context
 - Unbind
 - Resolve: trova un riferimento dato un nome

Naming Service

- Uso

- Ottenere riferimento al servizio

```
Org.omg.CORBA.Object objRef =  
    orb.resolve_initial_references("NameService");
```

- Narrow

```
NamingContext ncRef = NamingContextHelper.Narrow(objRef);
```

- Bind del server

```
NameComponent nc = new NameComponent("Calc", "");  
NameComponent path[] = {nc};  
ncRef.rebind(path, calc);
```

- Risoluzione del client

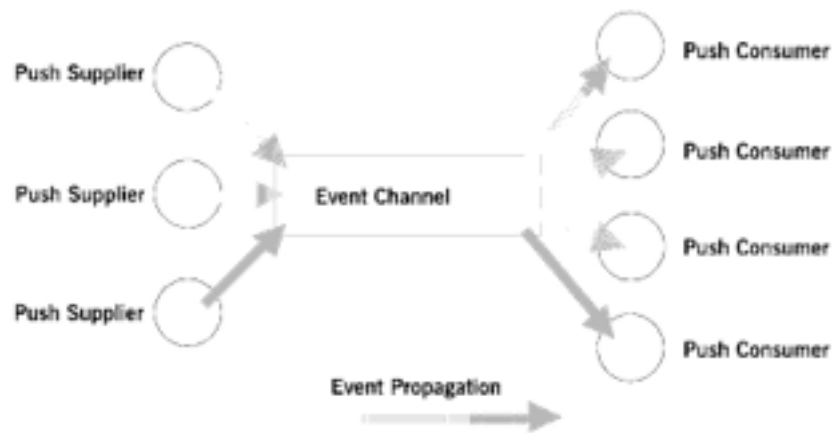
```
NameComponent nc = new NameComponent("Calc", "");  
NameComponent path[] = {nc};  
Calcolatrice calc = CalcolatriceHelper.narrow(ncRef.resolve(path));
```

Paradigma Publish-Subscribe

- Un sistema in cui una entità (*publisher*) può pubblicare un messaggio o evento che è fatto pervenire agli interessati (*subscriber*)
 - Uno a molti
 - One-way (il publisher non aspetta risposte)
 - Asynchronous (il publisher non si blocca mentre il messaggio viene inviato ai subscriber)
- I publisher sono detti anche supplier
- I subscriber sono detti anche consumer

Event Service

- Esistono due modelli
 - Push: i supplier fanno partire la comunicazione inviando messaggi ai consumer
 - Pull: i consumer fanno partire la comunicazione richiedendo messaggi ai supplier



Event Service

- **Modello Push**
 - Il supplier si connette al proxy del consumer invocando `connect_push_supplier()`
 - Un consumer si connette al proxy del supplier invocando `connect_push_consumer()`
 - Un evento è pubblicato dal supplier invocando `push()` sul proxy del consumer
- Nel modello Pull si hanno oggetti ed invocazioni analoghe

Consumer Pull (con JacORB)

```
import org.omg.CosEventChannelAdmin.*;
...
// risolve il riferimento iniziale al servizio
EventChannel ecs = EventChannelHelper.narrow( nc.resolve(
    nc.to_name("eventchannel") ) );
// ottiene un riferimento per ricavare i proxy
ConsumerAdmin ca = ecs.for_consumers();
// ottiene il proxy
ProxyPullSupplier pps = ca.obtain_pull_supplier();
...
// connette il consumer al channel
pps.connect_pull_consumer( (PullConsumer) pullConsumer );
...
// riceve l'evento (b è un boolean che indica se l'evento
// è disponibile)
Any received = pps.try_pull(b);
```

Conclusioni

- CORBA
 - Riduce la complessità e le difficoltà dello sviluppo di sistemi distribuiti
 - Permette l'interoperabilità tra oggetti implementati in linguaggi diversi
 - Fornisce un framework a cui agganciare oggetti che possono essere scoperti a run-time
- CORBA non affronta
 - Load Balancing
 - Scalabilità
 - Mobilità del codice delle classi