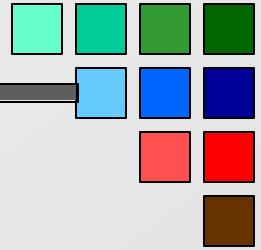
Sistemas Distribuídos

Ricardo Ribeiro dos Santos

ricrs@ec.ucdb.br

Curso de Engenharia de Computação – UCDB – Agosto/2003

Tópicos



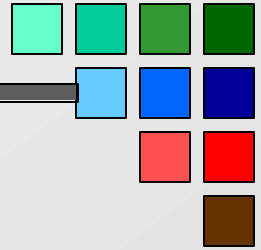
- A arquitetura CORBA

Para saber mais...

- O compreendimento e aplicação do tópico “Objetos Distribuídos” é **importante** na disciplina de SD

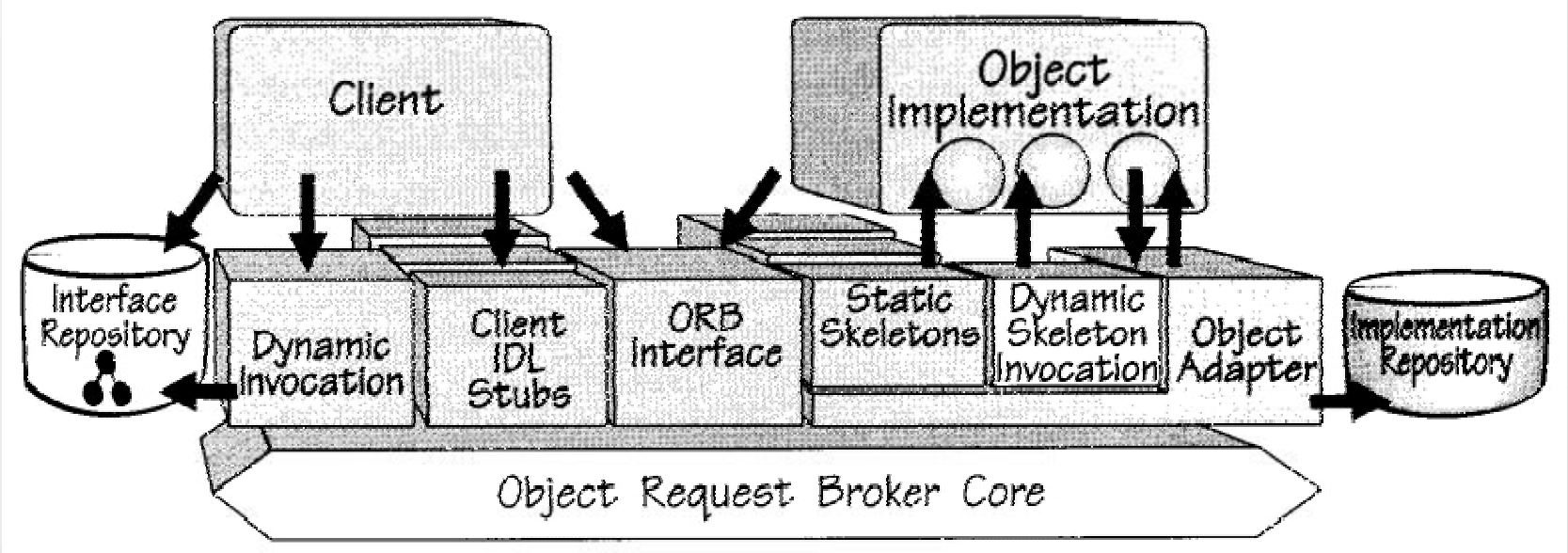
- De forma específica, outras informações sobre CORBA e implementações de aplicações baseadas em Objetos Distribuídos podem ser obtidas em:
  - ORFALI, Robert; HARKEY, Dan. Client/server programming with Java and CORBA. 2.ed New York: John wiley & sons, 1998. 1022 p.
  - HENNING, Michi; VINOSKI, Steve. Advanced CORBA programming with C++. Reading: Addison-wesley, 1999. 1038p.
  - SANTOS, Ricardo Ribeiro dos; TEIXEIRA, Mário Meireles. Desenvolvimento de aplicações distribuídas usando a arquitetura CORBA. São Carlos: ICMC/USP,2002

Introdução

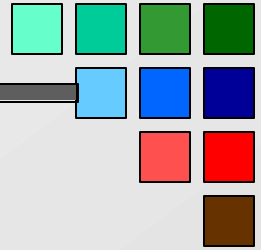


- CORBA
  - *Common Object Request Broker Architecture*
  - Solução aberta para o desenvolvimento de aplicações distribuídas em ambientes heterogêneos.
- OMG: *Object Management Group*
  - Mais de 700 membros atualmente (desde 1989).
  - Tem o objetivo de desenvolver, adotar e promover padrões para o desenvolvimento de aplicações em ambientes heterogêneos distribuídos.
  - Contribuições dos seus membros através de RFPs (*Requests for Proposals*)

Arquitetura CORBA

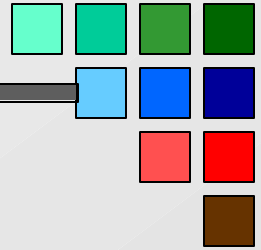


Stubs Clientes (Stubs)



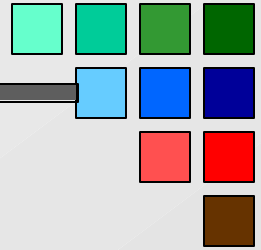
- Os *stubs clientes* são as interfaces estáticas para os serviços (objetos).
- São responsáveis por tratar dos detalhes da comunicação entre clientes e servidores.
- Para o cliente, o stub é como um *proxy local* para um objeto servidor remoto.
- O stub cliente compõe uma mensagem com a identificação do método invocado e seus parâmetros e a envia ao servidor.
- Em seguida, bloqueia-se para aguardar a resposta à solicitação feita.

Stubs Servidores (Skeletons)



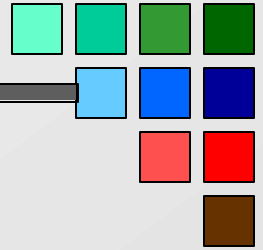
- São a parte correspondente aos stubs clientes, no ambiente servidor.
- Fornecem uma interface estática para cada serviço exportado pelo servidor.
- O *skeleton* obtém a identificação do método e os seus parâmetros da mensagem recebida e faz uma chamada local ao servidor.
- Ao ser completada a solicitação, envia uma mensagem com os resultados ao cliente.

Linguagem de Definição de Interface (IDL)



- A *Interface Definition Language* é usada para especificar a interface dos objetos servidores (dos serviços).
- A IDL é puramente *declarativa*, ou seja, não especifica nenhum detalhe de implementação.
- Através da IDL, definem-se quais os métodos disponíveis no servidor, seus parâmetros e tipos, valores de retorno, exceções, herança, eventos etc.
- A IDL estabelece uma espécie de “contrato” entre o servidor e seus clientes em potencial.

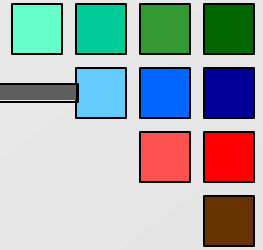
Repositório de Interfaces



- O *Interface Repository* contém uma base de dados com a definição das interfaces de todos os serviços conhecidos pelo ORB.

- É um repositório de metadados dinâmico para o ORB.

Interface de Invocação Dinâmica (DII)

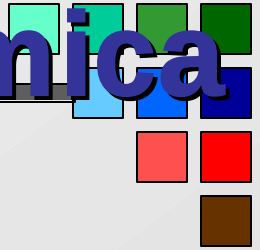


- Permite que o cliente invoque um método no servidor sem que se tenha conhecimento, em tempo de compilação, de sua interface.

- Neste tipo de interação, portanto, não se usam *stubs IDL*.

- Os dados sobre o objeto remoto são obtidos a partir da base de dados do Repositório de Interfaces.

Interface de *Skeleton* Dinâmica (DSI)

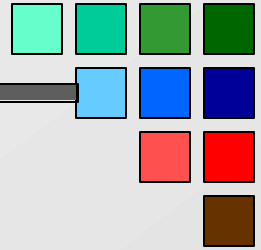


- É o correspondente à DII, no lado servidor.
- Permite que os servidores sejam escritos sem que se tenha *skeletons IDL* previamente compilados no código do programa.
- É útil na implementação de pontes entre ORBs e também para fazer a comunicação entre o CORBA e outras plataformas de computação distribuída.

Repositório de Implementação

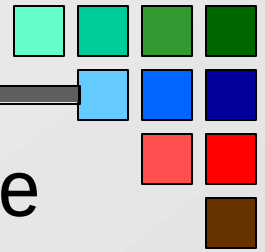
- O *Implementation Repository* é um repositório, em tempo de execução, para as classes que um servidor suporta, os objetos instanciados e suas identificações.

Adaptador de Objetos



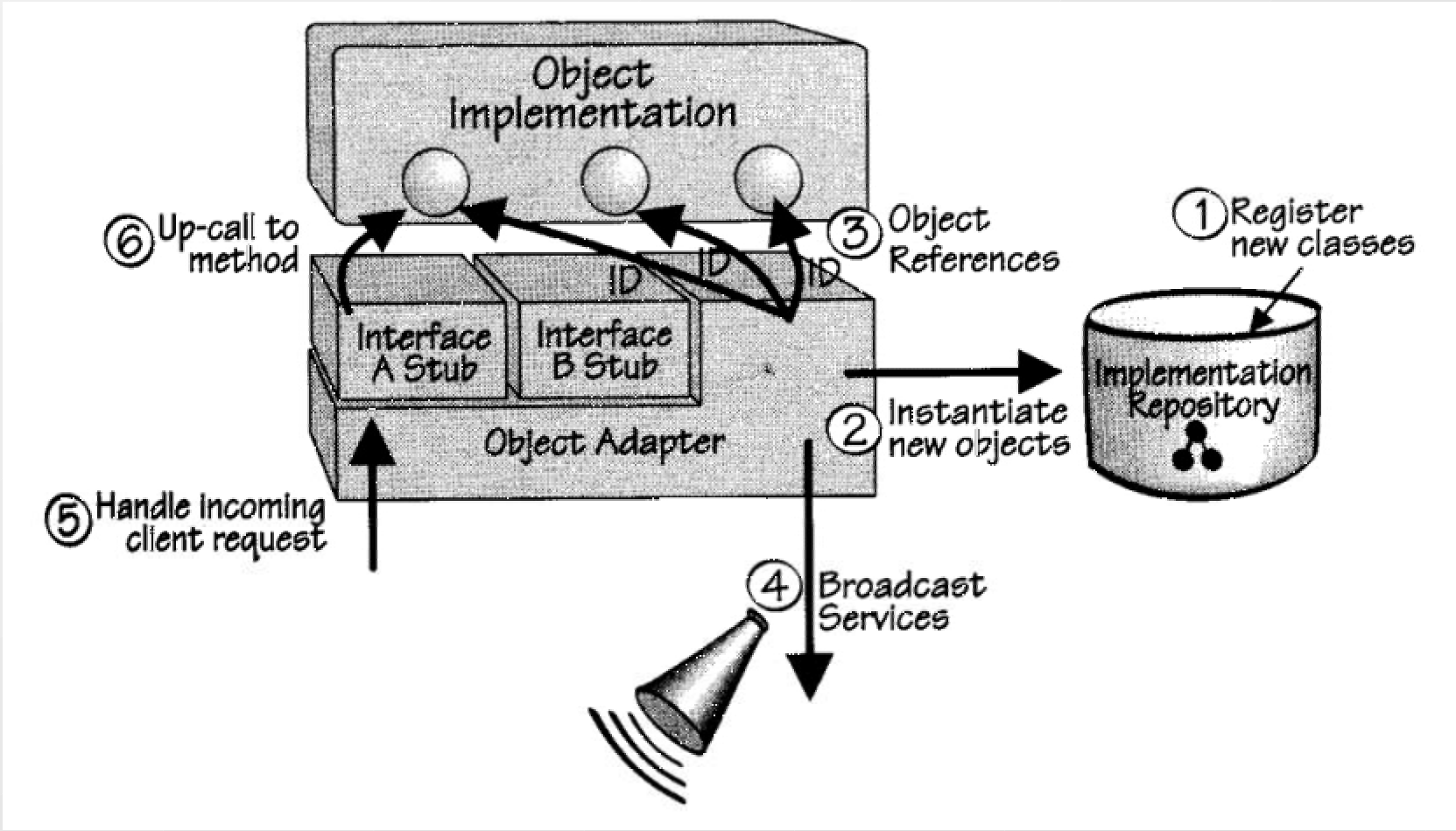
- O *Object Adapter* é a “cola” entre o ORB e as implementações dos objetos no lado servidor.
- É responsável por:
  - Registrar as classes servidoras no Repositório de Implementação;
  - Ativar (instanciar) os objetos chamados, em tempo de execução, segundo a demanda dos clientes;
  - Receber as chamadas para os objetos e repassá-las aos mesmos (invocação de métodos);
  - Gerar e gerenciar as *referências de objetos* (ORs).

Adaptador de Objetos

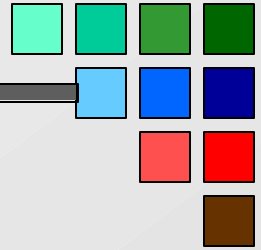


- Em resumo, um Adaptador de Objetos define como um objeto é ativado.
- O CORBA determina que todo ORB deve possuir um BOA (*Basic Object Adapter*), para garantir a portabilidade das aplicações.
- Normalmente, existe um adaptador de objetos diferente para cada linguagem suportada pelo ORB.
- Podem-se criar adaptadores para fins específicos.

Adaptador de Objetos



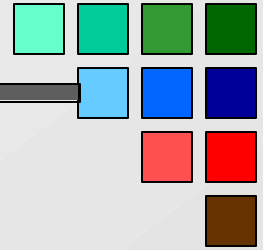
Interface do ORB



- A *ORB Interface*, presente nos lados cliente e servidor, consiste de algumas APIs para serviços locais de uso comum às aplicações.

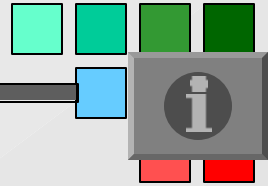
- Por exemplo, métodos para converter uma referência de objeto em uma string e vice-versa.

CORBA vs. Outras soluções para comunicação



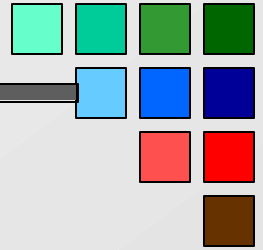
- CORBA apresenta várias vantagens em relação a outros tipos de *middleware* :
  - Inúmeros serviços agregados
  - Invocações estáticas e dinâmicas
  - *Bindings* com a maioria das linguagens (C, C++, COBOL, Java, Smalltalk, Ada...)
  - Sistema auto-descritivo (*metadados*)
  - Transparência em vários níveis
  - Serviços de segurança e transações
  - Mensagens polimórficas

CORBA vs. RPC



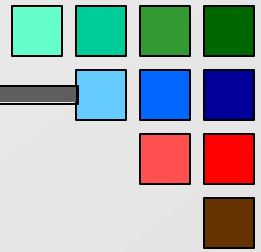
CARACTERÍSTICAS	CORBA	RPC
Binding	Através de Ref. De Objetos “Diretas”, Serviço de Nomes ou do Repositório de Implementação	Através do PortMapper
Níveis de Transparência	Invocação de chamadas, localização de aplicações e linguagem de programação.	Invocação de chamadas, localização de aplicações.
Tipos de Invocação	Chamadas bloqueantes (estilo RPC), oneway, retardo síncrono, callback e polling.	Chamadas bloqueantes e assíncronas...
Representação de Dados	CDR (Common Data Representation).	XDR (eXternal Data Representation).

Referências de Objetos (ORs)



- As *Object References* identificam, de forma única, um objeto dentro de um dado sistema CORBA.
- Uma OR pode ser um nome ou identificador, a sua representação interna fica a cargo de cada ORB.
- Os clientes obtêm ORs a partir de arquivos, serviços de diretórios, do Repositório de Interfaces ou como resultado de invocação de métodos.

Referências de Objetos

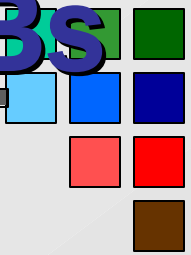


- Algumas operações relacionadas às referências de objetos:
  - *object_to_string*
  - *string_to_object*
  - *get_interface*
  - *describe*
  - *get_implementation* ...
- O CORBA 2.0 define *Interoperable ORs* (IORs) que são referências para serem usadas entre ORBs distintos.

Interoperabilidade entre ORBs

- O CORBA 1.1 garantia portabilidade das aplicações.
- O CORBA 2.0 garante a interoperabilidade entre as aplicações (entre ORBs).
- *General Inter-ORB Protocol* (GIOP)
  - Especifica uma representação comum e um conjunto de mensagens padrão para troca de informações entre ORBs, sobre um protocolo orientado à conexão.
- *Internet Inter-ORB Protocol* (IIOP)
  - Especifica como o GIOP funciona sobre uma rede TCP/IP (Internet).

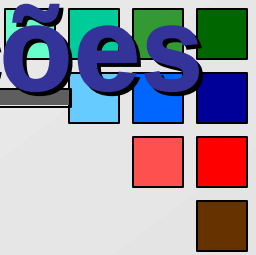
Interoperabilidade entre ORBs



- *Environment-Specific Inter-ORB Protocols* (ESIOPs)
  - Permitem a interoperabilidade com outras plataformas de computação distribuída.
  - Ex: DCE/ESIOP

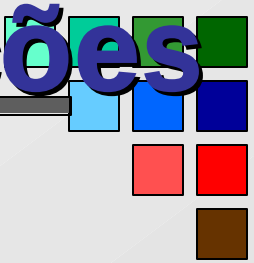
- O CORBA 2.0 também permite criar pontes entre ORBs, usando as interfaces DII e DSI. Isto viabiliza a construção de *gateways* para outras plataformas de computação distribuída.

Desenvolvimento de aplicações CORBA



- Escolher uma ferramenta/implementação CORBA
- Nessa escolha, algumas características devem ser analisadas:
  - Conformidade com a especificação;
  - Documentação disponível;
  - Mapeamento para linguagens de programação;
- Pode ser necessário verificar a viabilidade da ferramenta para o problema em questão
- Conhecer os recursos fornecidos pela IDL da especificação CORBA

Desenvolvimento de aplicações CORBA (cont.)



- O Processo de desenvolvimento é geralmente realizado em etapas:

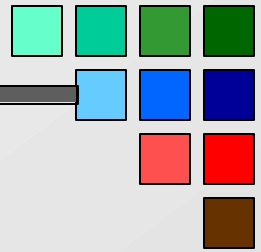
1 - Definir as interfaces (utilizando a IDL), métodos e parâmetros, que poderão ser invocados por clientes

2 - Utilizar um compilador IDL (fornecido pela implementação CORBA) para gerar os *stubs* e *skeletons*

3 - Implementar as aplicações cliente e servidor

4 - Compilar (utilizando o compilador da linguagem adotada) essas aplicações com os *stubs* e *skeletons* para gerar os arquivos executáveis

Etapas de desenvolvimento para uma aplicação CORBA



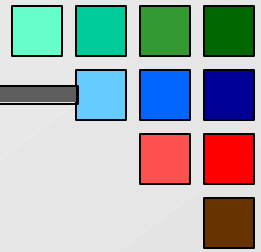
2

3

3

4

Etapas de desenvolvimento para uma aplicação CORBA



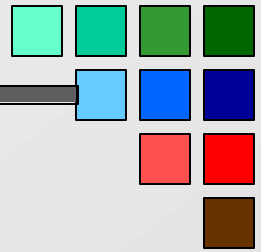
1 **Especificação em IDL**

2

3 **STUBS** **SKELETONS** 3

CLIENTE 4 **SERVIDOR**

Etapas de desenvolvimento para uma aplicação CORBA



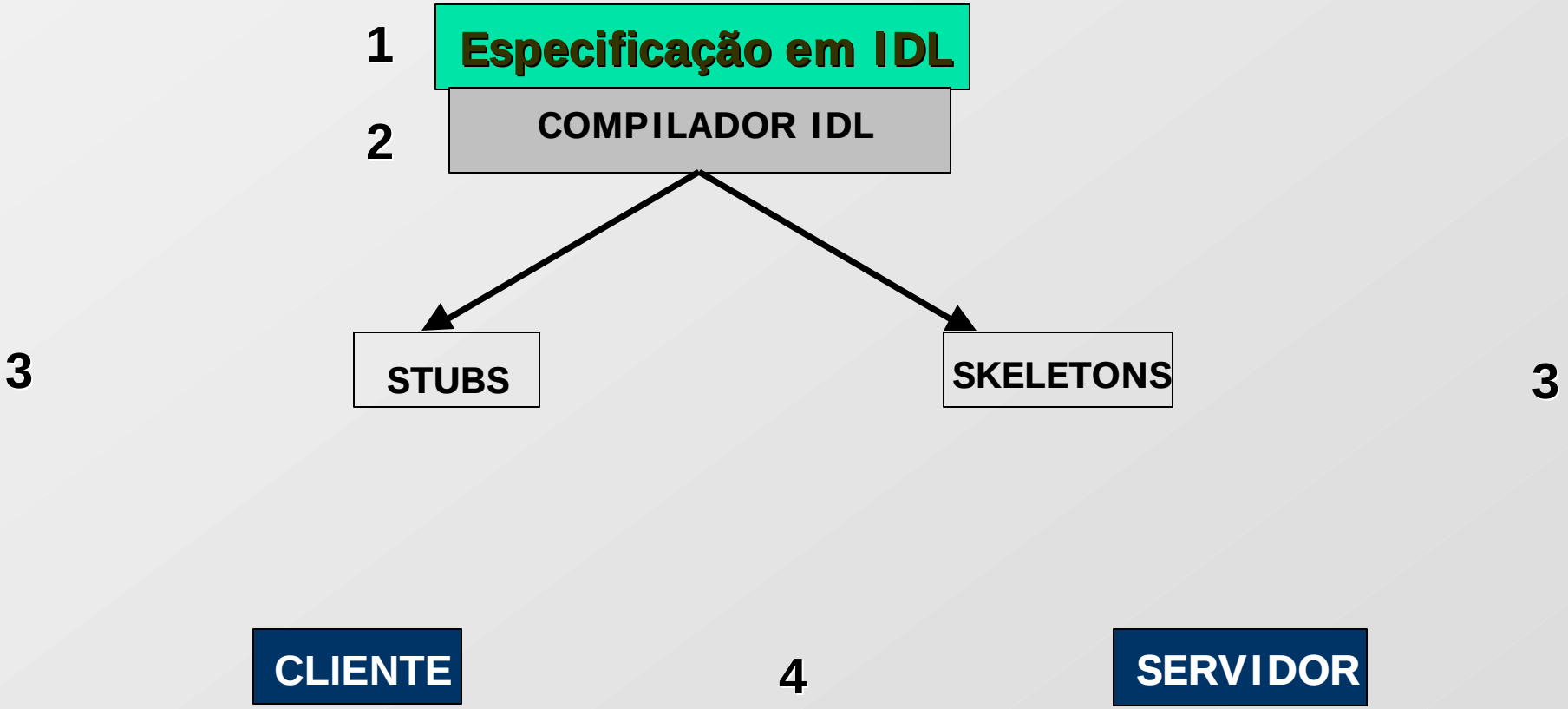
1 **Especificação em IDL**

2 **COMPILADOR IDL**

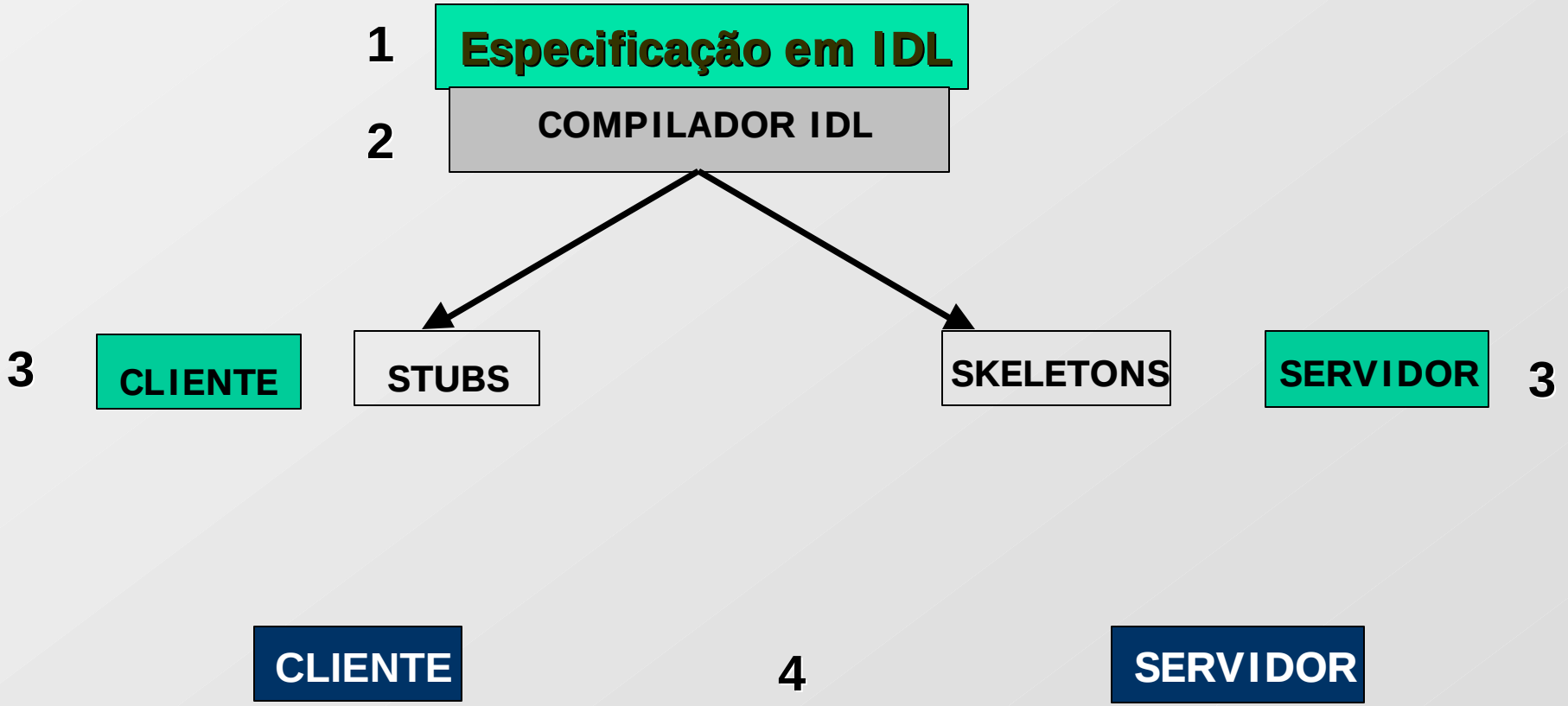
3 **STUBS** **SKELETONS** 3

CLIENTE 4 **SERVIDOR**

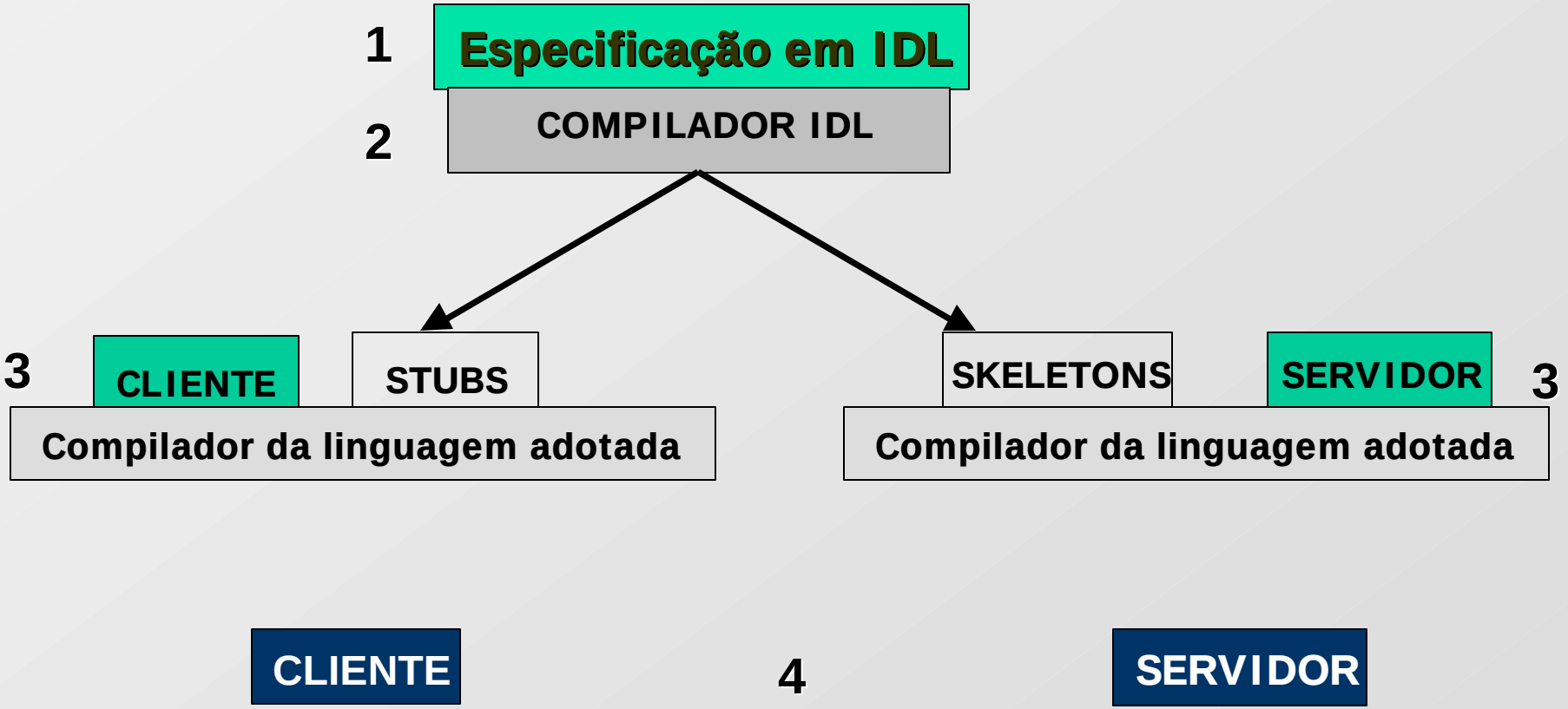
Etapas de desenvolvimento para uma aplicação CORBA



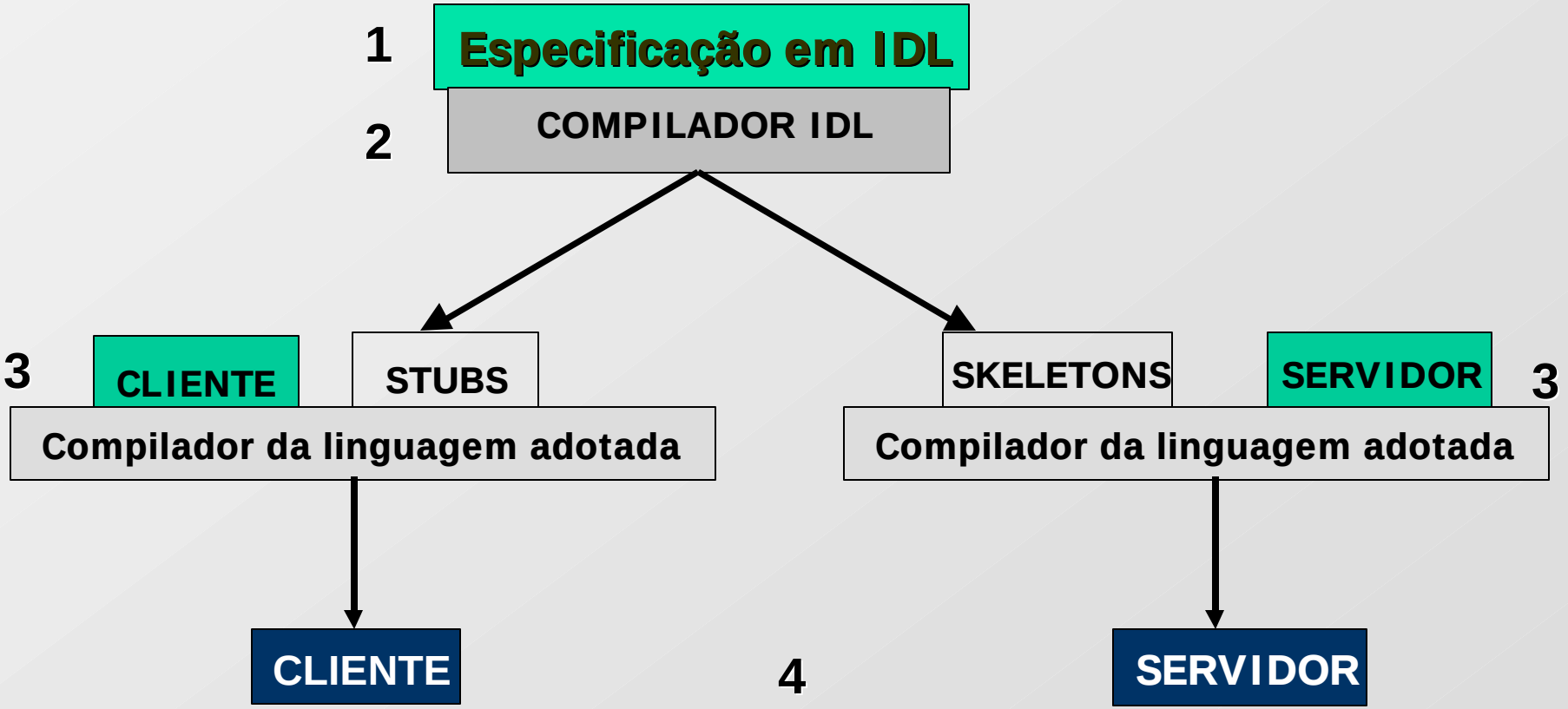
Etapas de desenvolvimento para uma aplicação CORBA

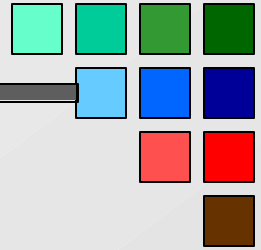


Etapas de desenvolvimento para uma aplicação CORBA



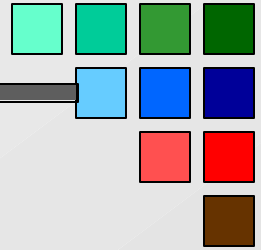
Etapas de desenvolvimento para uma aplicação CORBA





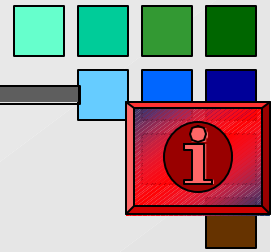
Implementações CORBA e Aplicações Exemplo

Algumas implementações CORBA



- ORBit: http://orbit-resource.sourceforge.net/
- TAO: http://www.cs.wustl.edu/~schmidt/TAO
- OmniORB: http://www.uk.research.att.com/omniORB/
- ORBacus: http://www.orbacus.com
- MiCO: http://www.mico.org/
- Visibroker: http://www.inprise.com/visibroker
- Orbix: http://www.iona.com
- idlj: disponível nas versões mais recentes do jdk

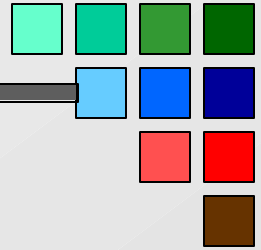
Exemplo de uma aplicação CORBA utilizando idlj



- Compilador IDL utilizado: `idlj`
- Mapeamento para Java
- `idlj` gera os *stubs*, *skeletons*, e os protótipos dos métodos que devem ser implementados
- Compilador das aplicações: `javac`
- Execução das aplicações: `java <nomeaplic>`
- Servidor de nomes: `tnameserv`

- Exemplo de uma aplicação **HelloWorld** distribuída
  - No próximo slide...

Exemplo *HelloWorld*



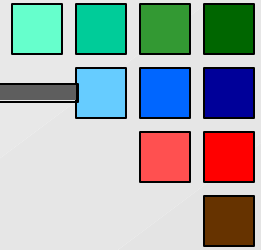
- **Exemplo Hello World**

Hello.idl

```
interface Hello
{
    void sayhello();
};
```

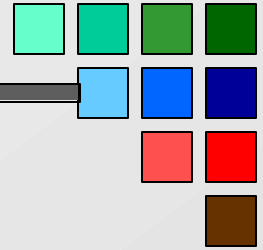
- Compilação da interface:
  - `idlj -fall Hello.idl`

Exemplo *HelloWorld*



- **Programa Cliente:** Em linhas gerais, o cliente segue o seguinte roteiro:
  - Inicializa o ORB, Adaptador de Objetos
  - Declara um objeto da classe implementada no servidor e definida no arquivo IDL
  - Obtém a Referência do Objeto Servidor
    - A partir de um serviço de nomes...
  - A partir da Referência obtida, inicializa o objeto de acordo com a classe implementada no servidor
  - Realiza requisições aos servidores

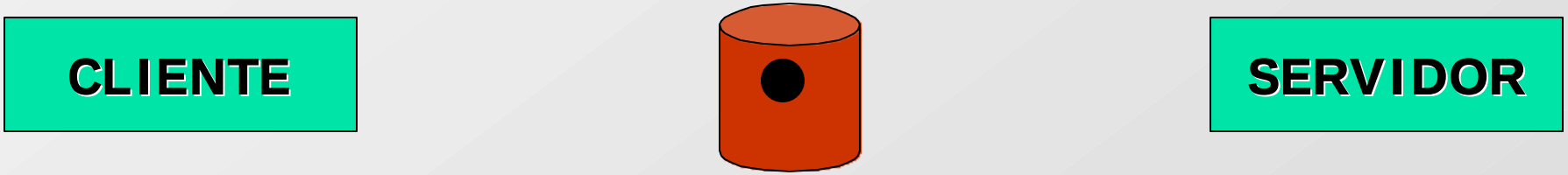
Exemplo *HelloWorld*



- **Programa Servidor:**
  - Inicializa o ORB, Adaptador de Objetos
  - Declara um objeto da classe implementada no servidor e definida no arquivo IDL
  - Cria uma referência do objeto declarado e registra essa referência em um Serviço de Nomes
  - Aguarda por requisições

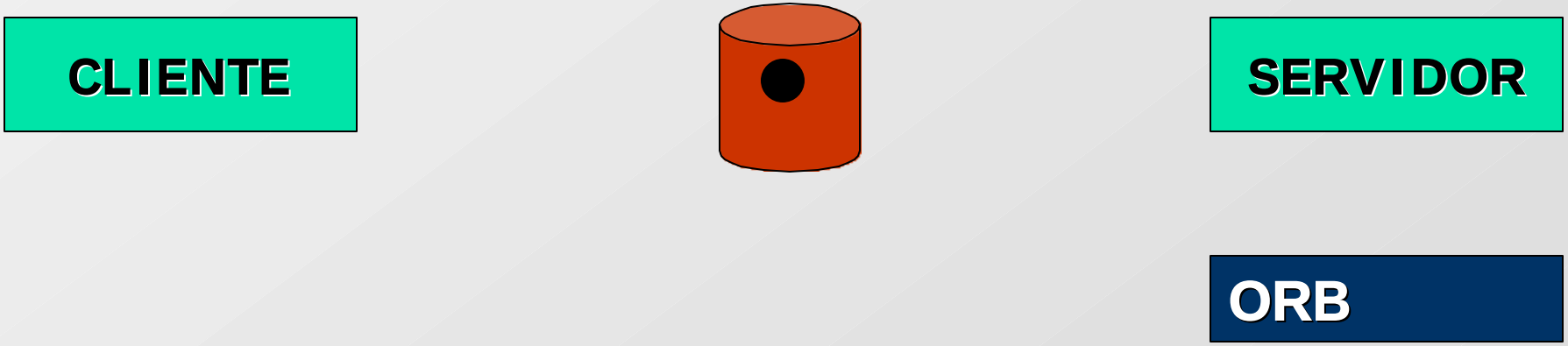
Exemplo *HelloWorld*

- Utilizando *binding* através de um arquivo compartilhado



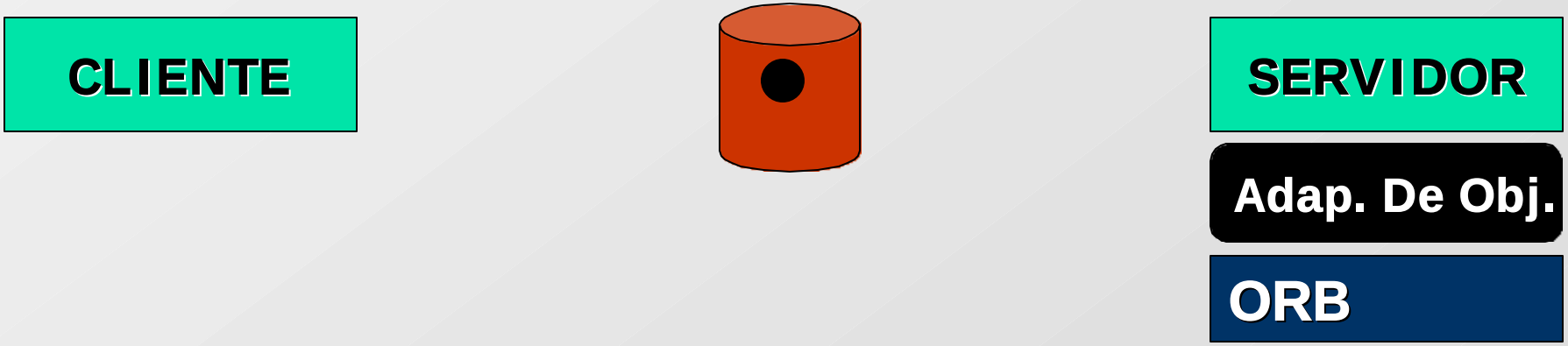
Exemplo *HelloWorld*

- Utilizando *binding* através de um arquivo compartilhado



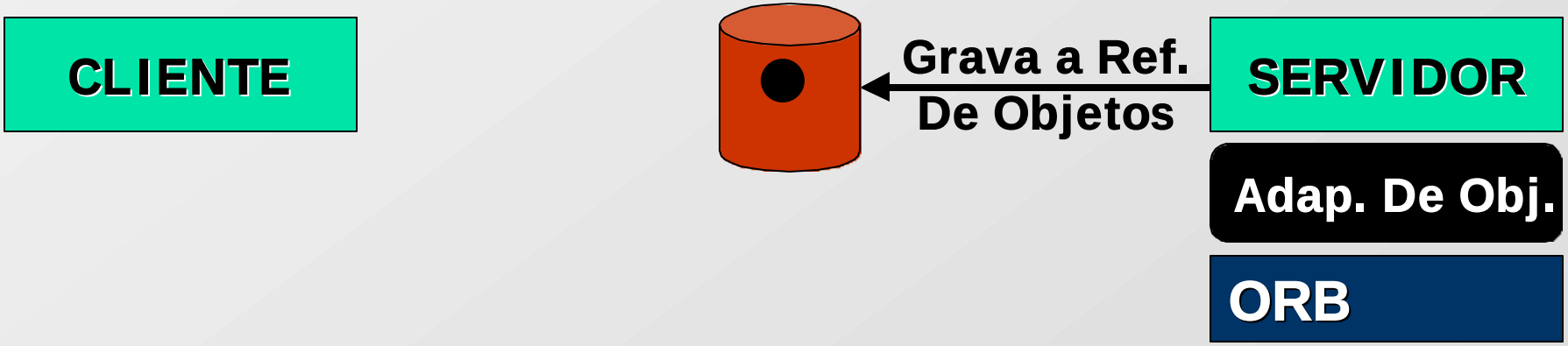
Exemplo *HelloWorld*

- Utilizando *binding* através de um arquivo compartilhado



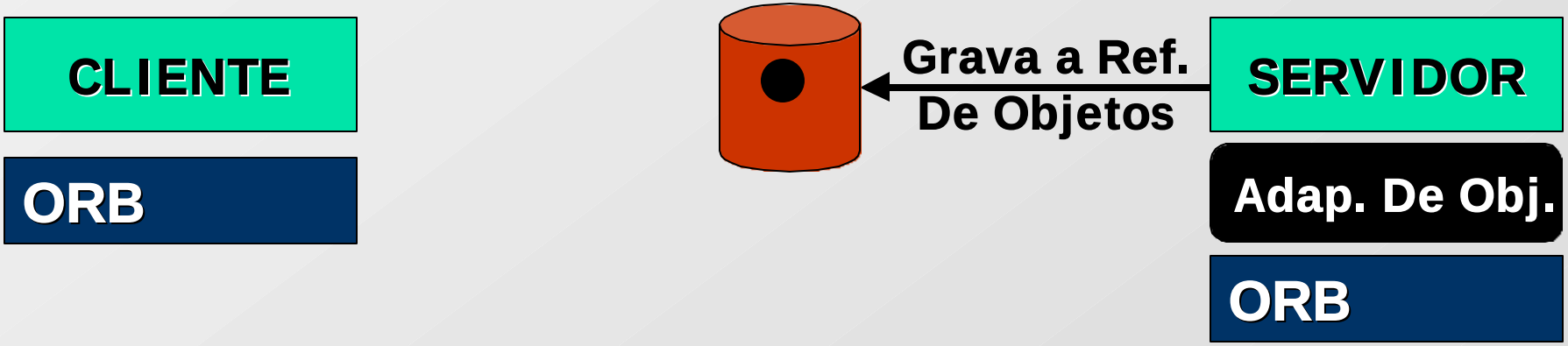
Exemplo *HelloWorld*

- Utilizando *binding* através de um arquivo compartilhado



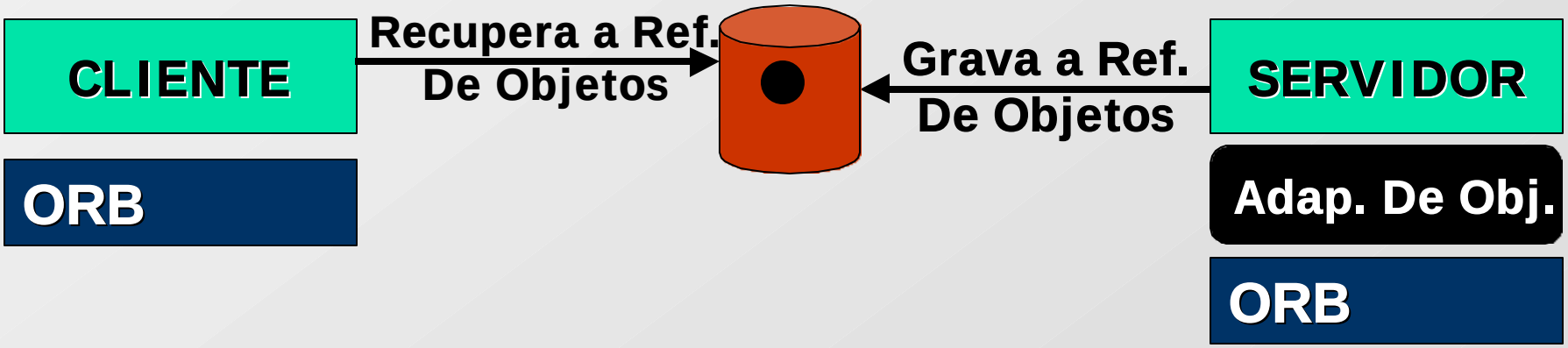
Exemplo *HelloWorld*

- Utilizando *binding* através de um arquivo compartilhado



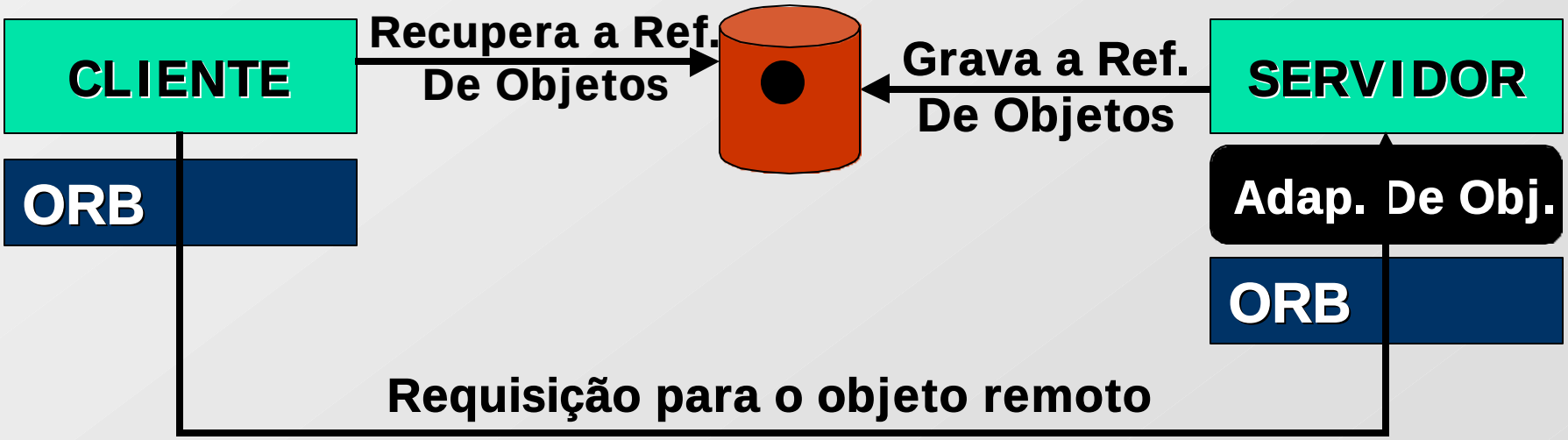
Exemplo *HelloWorld*

- Utilizando *binding* através de um arquivo compartilhado

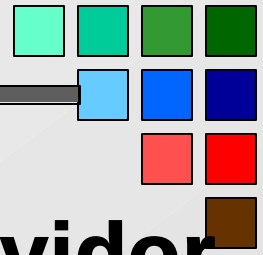


Exemplo *HelloWorld*

- Utilizando *binding* através de um arquivo compartilhado



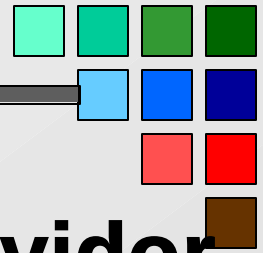
Exemplo *HelloWorld*



- **Exemplo Hello World - Aplicação Servidor**

```
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import org.omg.PortableServer.POA;
import java.util.Properties;


class HelloImpl extends HelloPOA {
  private ORB orb;

  public void setORB(ORB orb_val) {
     orb = orb_val;
  }

// implementação do método sayhello()
  public String sayhello() {
     return "\nHello world !!\n";
  }
}
```

Exemplo *HelloWorld*

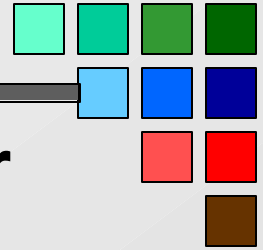


- **Exemplo Hello World - Aplicação Servidor**

```
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import org.omg.PortableServer.POA;
import java.util.Properties;


class HelloImpl extends HelloPOA {
  private ORB orb;

  public void setORB(ORB orb_val) {
     orb = orb_val;
  }

// implementação do método sayhello()
  public String sayhello() {
     return "\nHello world !!\n";
  }
}
```

Implementação do método say_hello()

Exemplo *HelloWorld*

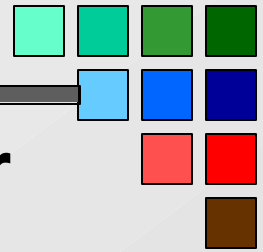


- **Exemplo Hello World - Aplicação Servidor**

```
public class HelloServer {
  public static void main(String args[]) {
  try{

      // Criação e inicialização do ORB
      ORB orb = ORB.init(args, null);

      // obtém a referência do rootpoa e ativa o POAManager
      POA rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
      rootpoa.the_POAManager().activate();

      // cria o servant e registra com o ORB
      HelloImpl helloImpl = new HelloImpl();
      helloImpl.setORB(orb);

      // obtém referência do objeto servant
      org.omg.CORBA.Object ref = rootpoa.servant_to_reference(helloImpl);
      Hello href = HelloHelper.narrow(ref);
      org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
```

Converte o objeto helloImpl
para o formato a ser gravado no SN

Exemplo *HelloWorld*



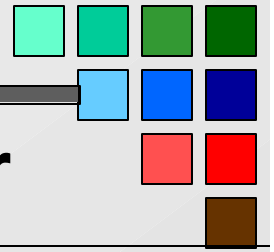
- **Exemplo Hello World - Aplicação Servidor**

```
public class HelloServer {
  public static void main(String args[]) {
  try{

      // Criação e inicialização do ORB
      ORB orb = ORB.init(args, null);

      // obtém a referência do rootpoa e ativa o POAManager
      POA rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
      rootpoa.the_POAManager().activate();

      // cria o servant e registra com o ORB
      HelloImpl helloImpl = new HelloImpl();
      helloImpl.setORB(orb);

      // obtém referência do objeto servant
      org.omg.CORBA.Object ref = rootpoa.servant_to_reference(helloImpl);
      Hello href = HelloHelper.narrow(ref);
      org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
```

Inicializa o ORB

Converte o objeto helloImpl
para o formato a ser gravado no SN

Exemplo *HelloWorld*



- **Exemplo Hello World - Aplicação Servidor**

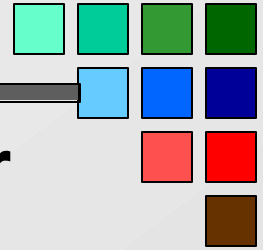
```
public class HelloServer {
  public static void main(String args[]) {
  try{

     // Criação e inicialização do ORB
     ORB orb = ORB.init(args, null);

     // obtém a referência do rootpoa e ativa o POAManager
     POA rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
     rootpoa.the_POAManager().activate();

     // cria o servant e registra com o ORB
     HelloImpl helloImpl = new HelloImpl();
     helloImpl.setORB(orb);

     // obtém referência do objeto servant
     org.omg.CORBA.Object ref = rootpoa.servant_to_reference(helloImpl);
     Hello href = HelloHelper.narrow(ref);
     org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
```

Inicializa o ORB

Inicializa o Adaptador de Objetos

Converte o objeto helloImpl para o formato a ser gravado no SN

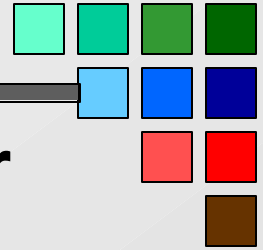
Exemplo *HelloWorld*



- **Exemplo Hello World - Aplicação Servidor**

```
// Especifica o servidor de nomes(INS)
    NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

    // Especifica uma referência de objeto
    String name = "Hello";
    NameComponent path[] = ncRef.to_name( name );
    ncRef.rebind(path, href);

    System.out.println("HelloServer pronto e esperando...");

    // Aguarda a invocação de clientes
    orb.run();
  }catch (Exception e){ System.err.println("ERROR: " + e); e.printStackTrace(System.out);}
  System.out.println("HelloServer Terminando ...");
 }
}
```

Exemplo *HelloWorld*

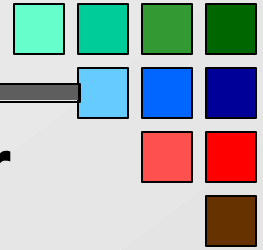


- **Exemplo Hello World - Aplicação Servidor**

```
// Especifica o servidor de nomes(INS)
    NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

    // Especifica uma referência de objeto
    String name = "Hello";
    NameComponent path[] = ncRef.to_name( name );
    ncRef.rebind(path, href);

    System.out.println("HelloServer pronto e esperando...");

    // Aguarda a invocação de clientes
    orb.run();
  }catch (Exception e){ System.err.println("ERROR: " + e); e.printStackTrace(System.out);}
  System.out.println("HelloServer Terminando ...");
 }
}
```

Grava a Ref. Ao objeto helloImpl no SN

Exemplo *HelloWorld*



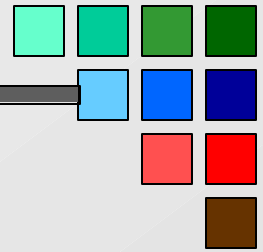
- **Exemplo Hello World - Aplicação Servidor**

```
// Especifica o servidor de nomes(INS)
    NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

    // Especifica uma referência de objeto
    String name = "Hello";
    NameComponent path[] = ncRef.to_name( name );
    ncRef.rebind(path, href);

    System.out.println("HelloServer pronto e esperando...");

    // Aguarda a invocação de clientes
    orb.run();
  }catch (Exception e){ System.err.println("ERROR: " + e); e.printStackTrace(System.out);}
  System.out.println("HelloServer Terminando ...");
 }
}
```

Grava a Ref. Ao objeto helloImpl no SN

Servidor fica aguardando por requisições de clientes

Exemplo *HelloWorld*



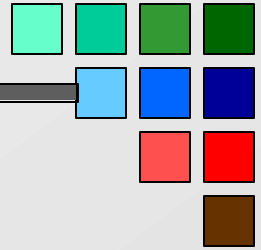
- **Exemplo Hello World - Aplicação Cliente**

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

public class HelloClient
{
  static Hello helloImpl;
  public static void main(String args[])
  {
    try{
      // Cria e inicializa o ORB
      ORB orb = ORB.init(args, null);

      //Procura pelo servidor de nomes
      org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
      NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
      String name = "Hello";
      helloImpl = HelloHelper.narrow(ncRef.resolve_str(name));

      System.out.println(helloImpl.sayhello());
    }catch (Exception e) {  System.out.println("ERROR : " + e) ;  e.printStackTrace(System.out);  }
  }
}
```

Inicializa o ORB

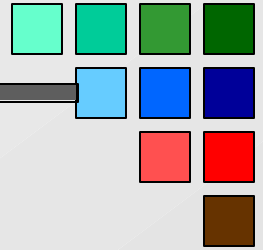
Obtém a Ref. do Objeto remoto a partir do SN

Exemplo *HelloWorld*



- Compilação do cliente e servidor:
  - **`javac *.java`**
- Passos para execução:
  - 1) Executar o serviço de nomes:
    - **`tnameserv -ORBInitialPort XXX`**
      - XXX é o número da porta utilizada para serviço de nomes
  - 2) Executar o Servidor
    - **`start java HelloServer -ORBInitialPort XXX`**
  - 3) Executar o cliente
    - **`java HelloClient -ORBInitialPort XXX`**

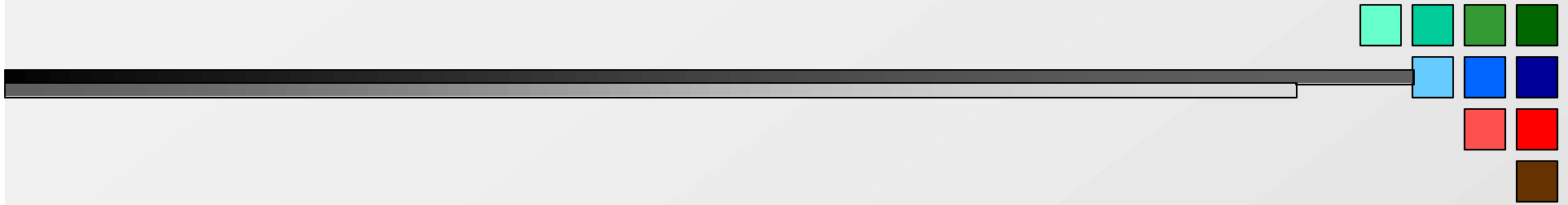
Exemplo Agenda



- **Exemplo Agenda - Interface IDL**

```
interface Agenda
{
   void insereItem(in string nome, in string fone);
   string consultaNome(in string fone);
};
```

- **Como ficaria a implementação do cliente e do servidor???**

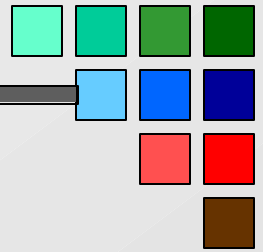


Mais detalhes sobre a IDL...

A IDL

- A IDL é utilizada para descrever as interfaces dos objetos servidores, definindo suas funcionalidades;
- A OMG CORBA IDL é um padrão ISO e ANSI (X3H7);
- A IDL não é uma linguagem de programação. Ao contrário, ela permite que as interfaces sejam definidas independentemente das linguagens utilizadas para implementá-las;
- A definição das interfaces entre componentes é um dos aspectos mais importantes do projeto de um sistema distribuído;

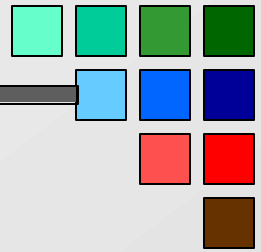
A IDL (cont.)



- Com o uso da IDL, é possível especificar:
  - Os atributos de uma interface;
  - Seus métodos, parâmetros de entrada/saída, tipos e valores de retorno;
  - As interfaces das quais herda;
  - Exceções e Eventos.

- A gramática da IDL é um subconjunto da linguagem C++ com mais algumas palavras-chave para suportar conceitos de sistemas distribuídos;

A IDL (cont.)



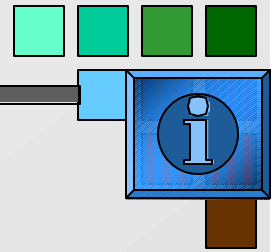
- Com o uso da IDL, é possível especificar:
  - C
  - S
    e
  - A
  - E

tipos

Deve haver um mapeamento de IDL para uma linguagem de programação, a fim de que *stubs*, *skeletons* e servidor sejam implementados nessa linguagem

- A gramática da IDL é um subconjunto da linguagem C++ com mais algumas palavras-chave para suportar conceitos de sistemas distribuídos;

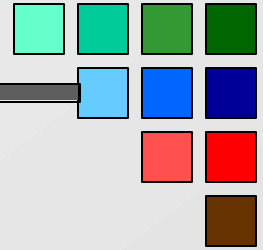
A IDL (cont.)



- As descrições das interfaces dos objetos em IDL são geralmente usadas para gerar os *stubs* e *skeletons*;

- As descrições em IDL podem também ser armazenadas no Repositório de Interfaces, que é o repositório de *meta-dados* de um sistema CORBA.

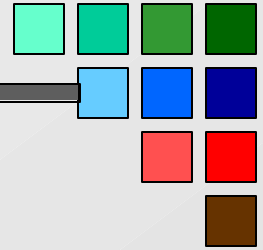
A IDL - Interface e *Module*



- Interface
  - Define um conjunto de métodos (*operations*) que um cliente pode invocar em um objeto;
  - Uma interface pode, além disso:
    - possuir atributos, constantes, tipos pré-definidos;
    - declarar exceções;
    - ser derivada de uma ou mais interfaces (herança).

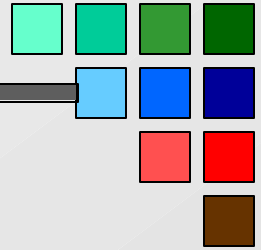
- Módulo (*Module*)
  - É um *namespace* que agrupa um conjunto de descrições de classes (interfaces). Um módulo define um escopo;

Exemplo de IDL - *Module* e Interface



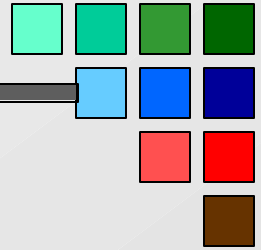
```
module Banco
{

    interface Conta{
        op1();
        op2();
    };
    interface Cliente{
        op1();
        op2();
        op3();
    };
```

A IDL - Operações



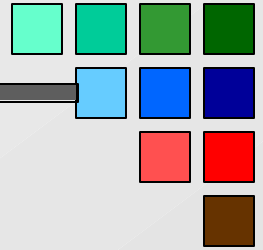
- Operações
  - São o equivalente em CORBA a um método;
  - A IDL define a assinatura de uma operação, ou seja, seus parâmetros, tipos e valor de retorno;
  - É possível especificar se um parâmetro é de entrada (*in*), saída (*out*) ou ambos (*inout*).

A IDL - Tipos de Dados



- Tipos de dados
  - Básicos: *short*, *long* (32 bits), *long long*, *unsigned long*, *float*, *double*, *long double*, *fixed<x, y>*, *char*, *wchar*, *boolean*, *octet*, *any*;
  - Construídos: *string*, *wstring*, *array*, *sequence*, *struct*, *union*, *enum;*
  - Deve haver um mapeamento entre a IDL e cada linguagem suportada pelo ORB.

A IDL - Operações e Parâmetros

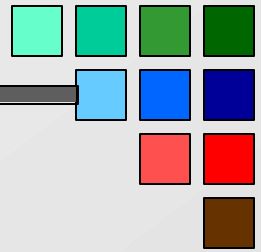


- Cada parâmetro de um método deve ter um nome e sua direção especificada: *in*, *out*, *inout*;
- Todas as operações devem ter um tipo de retorno (mesmo que seja *void*);
- Todas as definições feitas em uma interface IDL são *public*. Conceitos como *private* e *protected* têm a ver com a implementação da interface (em C++, Java).

Exemplo de IDL - Operações, Atributos e Tipos de Dados

```
typedef float Preco;
struct Lugar {
   char fila;
   unsigned long assento;
};

interface Bilheteria {
   readonly attribute string nomeCinema;
   readonly attribute char ultimaFila;
   . . .
   Preco obtemPreco (in Lugar lugarRes, in string dataRes);
   Reserva fazReserva (in Lugar lugarRes, in string dataRes,
                       in string numCartao);
};
```

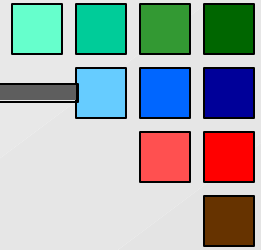
IDL - Operações *oneway*



- Operações são, por “padrão”, definidas como bloqueantes (*two-way*);
- Uma operação pode ser definida na IDL como *oneway*, a fim de não provocar o bloqueio do cliente durante a chamada;
  - Envio pelo “melhor esforço” (*best effort*);
- A especificação CORBA determina que qualquer requisição a um objeto deve ser executada no máximo uma vez (semântica *at-most-once*).

```
interface Bilheteria {
   // Envia comentarios sobre as instalacoes do cinema
   oneway void fazComentario (in string coment);
};
```

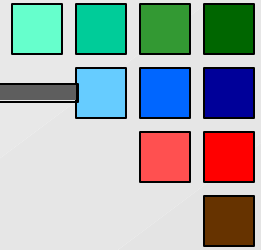
IDL - Exceções



- Uma operação IDL pode gerar uma exceção para indicar a ocorrência de um erro.

```
interface Bilheteria {
   exception LugarInexistente { Lugar lugarRes; };
   exception DataInvalida { Data dataRes; };

   Preco obtemPreco (in Lugar lugarRes, in string dataRes)
                  raises (LugarInexistente, DataInvalida);
};
```

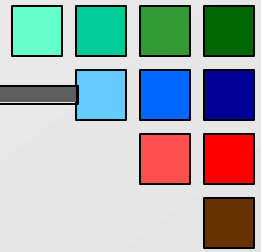
IDL - Herança



- É possível definir herança (simples ou múltipla) entre as interfaces presentes em um módulo.

```
interface Reserva {
   readonly attribute Data dataRes;
   readonly attribute Lugares assentos;
   void cancel();
};

interface ReservaGrupo : Reserva {
   readonly attribute string responsavel;
   readonly attribute string nomeGrupo;
};
```

IDL - Tipos Construídos



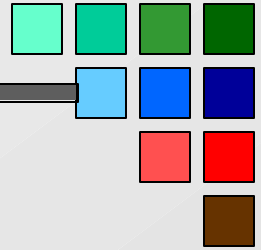
- Os tipos: *struct*, *union* e *enum*, apresentam as mesmas funcionalidades dos tipos existentes (com o mesmo nome) em linguagens como C++.

```
struct CartaoCred {
   string num;
   string dataVal;
};
```

```
union Token switch (long) {
   case 1 : long l;
   case 2 : float f;
   default : string str;
};
```

```
enum tipoAssento (plateia, balcao1, balcao2);
```

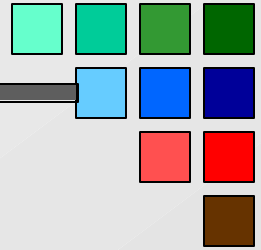
IDL - *Arrays*



- A IDL permite a definição de *arrays* multidimensionais de tamanho fixo. Definições de *array* devem ser escritas usando-se *typedef*.

```
struct Lugar {
   char fila;
   unsigned long assento;
};

typedef Reserva Reservas [100];

typedef Lugar ConjuntoLugares [10] [20];
```

IDL - *Sequences*



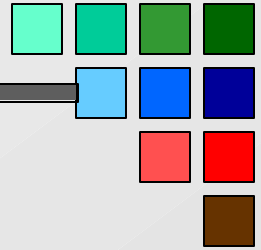
- Uma seqüência é uma lista de elementos de tamanho variável, de qualquer tipo IDL.

```
struct Lugar {
   char fila;
   unsigned long assento;
};
typedef sequence<Lugar> Lugares;
```

```
struct no{
   long id;
   sequence<no>  next;
};
```

- É recomendável que seqüências e *arrays*, quando usado como parâmetro, atributo ou valor de retorno sejam definidos usando-se *typedef*.

IDL - *Strings*

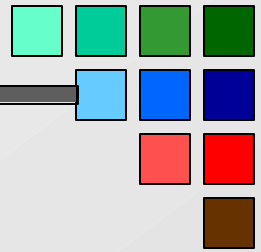


- Uma *string* em IDL é uma lista de *char* e pode conter qualquer caractere ASCII-Latin, exceto NUL;
- Uma *string* pode ou não ter um tamanho máximo (*bounded*).

```
typedef string<10> Data;
 Data dataNasc;
```

```
interface Cadastro{
  void inserir (in string nome, in string endereco);
  string ret_nome (in string nome_pessoa);
}
```

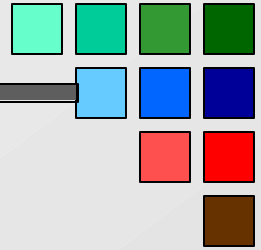
IDL - Constantes



- Uma constante em IDL pode ser definida assim:

```
interface Bilheteria {
   const long tamNomeFilme = 60;

   typedef string<tamNomeFilme> nomeFilme;

   . . .
};
```

IDL - *Typedef*



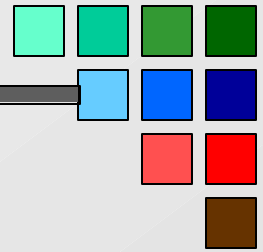
- Uma declaração *typedef* pode ser usada para definir um nome mais simples ou significativo para um tipo básico ou definido pelo usuário.

```
typedef string<10> Data;

void op1(in Data d);
```

é equivalente a:

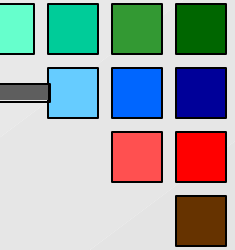
```
void op1(in string<10> d);
```

IDL - *Any*



- Utilizado para representar qualquer outro tipo IDL, inclusive tipos construídos e outro tipo *Any*;
- Há situações em que não é possível especificar, em tempo de compilação, qual(is) o(s) tipo(s) de dado(s) que devem ser transmitidos entre cliente e servidor;
- As características do tipo *Any* são similares ao void * (C/C++);
- É auto-descritivo (é possível saber que tipo de dado está contido em uma variável *Any*).

A IDL e o Rep. de Interfaces



- **IDL + Repositório de Interfaces** permite:
  - Que os componentes do sistema “descubram-se” em tempo de execução e possam interoperar (Invocação Dinâmica-DII);
  - Que os clientes sejam escritos sem se preocupar com os servidores que são utilizados;
  - Comunicação com sistemas mais antigos, não necessariamente orientados a objetos;
  - Comunicação com sistemas que modificam a interface constantemente.