O Lado do Servidor

- Veremos um exemplo de servidor escrito em C++
- Usaremos um ORB específico, o Orbix
  - » o servidor não será portável entre ORBs
- Implementaremos a interface **`Quoter`**
  - » usando herança
  - » usando delegação

Exemplo de Servidor: os Arquivos-Fonte

- Compilação IDL:
  - » **`quoter.idl`** ⟹ **`quoter.hh`**
- Vamos escrever:
  - » **`quotimpl.h`**
    - – header file da classe implementação **`Quoter_i`**
  - » **`quotimpl.cc`**
    - – métodos da classe **`Quoter_i`**
  - » **`quotmain.cc`**
    - – mainline do servidor

Implementando a Interface **`Quoter`** com Herança

```
// arquivo quotimpl.h:
#include "quoter.hh" // gerado pelo compilador IDL

class Quoter_i : public virtual QuoterBOAImpl {
public:
    Quoter_i(const char* my_name);
    char* name(CORBA::Environment& e =
               CORBA::default_environment);
    CORBA::Long value(const char* name,
                      CORBA::Environment& e =
                      CORBA::default_environment);
```


Implementando a Interface **`Quoter`** com Herança (cont.)

```
private:
    string m_name; // classe string ANSI/ISO
    map<string,CORBA::Long> m_map; // classe map
};                                 // da STL
```

`quotimpl.cc` contém o corpo de

- **`Quoter_i::Quoter_i()`**
- **`Quoter_i::name()`**
- **`Quoter_i::value()`**

Quoter_i::name()

```
// em quotimpl.cc:

char*
Quoter_i::name(CORBA::Environment&)
{
    char* nm = CORBA::string_dup(m_name.c_str());
    if (nm == 0)
        throw CORBA::NO_MEMORY();
    return nm;
}
```


Quoter_i::value()

```
// em quotimpl.cc:

CORBA::Long
Quoter_i::value(const char* stock,
                CORBA::Environment&)
{
    map<string,CORBA::Long>::iterator iter =
        m_map.find(stock);
    if (iter == m_map.end())
        throw Stock::UnknownStock(stock);
    return (*iter).second();
}
```

Quoter_i::Quoter_i()

```
// em quotimpl.cc:

Quoter_i::Quoter_i(const char* my_name)
    : m_name(my_name)
{
    // inicializa m_map com informações lidas
    // de um arquivo ou banco de dados
    // ...

    //          (num quoter real, m_map deveria
    //           ser atualizado periodicamente)
}
```


Mainline do Servidor

```
// arquivo quotmain.cc:

#include "quoter.hh"
int main(int argc, char* argv[])
{
    CORBA::ORB_var orb_ref =
        CORBA::ORB_init(argc, argv, "Orbix");
    CORBA::BOA_var boa_ref =
        orb_ref->BOA_init(argc, argv, "Orbix_BOA");
```

Mainline do Servidor (cont.)

```
    // cria um objeto Quoter_i
    Quoter_i my_quoter("Dow Jones");

    try {
        boa_ref->impl_is_ready("DowJonesQuoter");
    }
    catch (CORBA::SystemException& e) {
        cerr << "Exceção: " << endl << &e;
        return 1;
    }
    return 0;
}
```


Mainline do Servidor: Observações (1)

- O objeto **`Quoter_i`** foi criado em memória automática
- Poderia ter sido criado no *heap*:

```
// cria um objeto Quoter_i
Quoter_ptr my_quoter_ref =
    new Quoter_i("Dow Jones");
```

  - » ou ainda:

```
// cria um objeto Quoter_i
Quoter_var my_quoter_ref =
    new Quoter_i("Dow Jones");
```

Mainline do Servidor: Observações (2)

- A string passada à função **`CORBA::BOA::impl_is_ready()`** é o **nome do servidor** (o nome com o qual o servidor foi registrado no repositório de implementações, via comando **`putit`** do Orbix)
- Essa função só retorna ao final da execução do servidor (o laço de processamento de eventos está dentro de **`impl_is_ready()`**)

Mainline do Servidor: Observações (3)

- O Orbix oferece outras funções (proprietárias) de processamento de eventos: **`CORBA::BOA::processNextEvent()`** e **`CORBA::BOA::processEvents()`**
- No caso do Orbix, pode -se omitir as chamadas às funções **`CORBA::ORB_init()`** e **`CORBA::ORB::BOA_init()`**
  - » ...e neste caso o servidor fica ainda menos portável

Mainline do Servidor (ainda mais específica para o Orbix)

```
int main()
{
    Quoter_i my_quoter("Dow Jones");
    try {
        CORBA::Orbix.impl_is_ready("DowJonesQuoter");
    }
    catch (CORBA::SystemException& e) {
        cerr << "Exceção: " << endl << &e;
        return 1;
    }
    return 0;
}
```


Implementando a Interface **`Quoter`** com Delegação

```
// arquivo quotimpl.h:
#include "quoter.hh" // gerado pelo compilador IDL

class Quoter_i {
public:
    Quoter_i(const char* my_name);
    char* name(CORBA::Environment& e =
               CORBA::default_environment);
    CORBA::Long value(const char* name,
                      CORBA::Environment& e =
                      CORBA::default_environment);
```

Implementando a Interface **`Quoter`** com Delegação (cont.)

```
private:
    string m_name; // classe string ANSI/ISO
  class
    map<string,CORBA::Long> m_map; // classe map
};                                 // da STL

// a chamada de macro abaixo gera a definição da
// classe TIE_Quoter(Quoter_i)
DEF_TIE_Quoter(Quoter_i)

// as macros DEF_TIE_... são específicas do Orbix
```


Delegação: Mainline do Servidor

```
int main()
{
    Quoter_i my_quoter_i("Dow Jones");
    TIE_Quoter(Quoter_i) my_quoter(&my_quoter_i);
    try {
        CORBA::Orbix.impl_is_ready("DowJonesQuoter");
    }
    catch (CORBA::SystemException& e) {
        cerr << "Exceção: " << endl << &e;
        return 1;
    }
    return 0;
}
```

Delegação: Mainline do Servidor (Observações)

- Notar que o **`Quoter_i`** e o *tie* foram criados em memória automática
- Poderiam ter sido criados no *heap*:

```
// primeiro cria um objeto Quoter_i
Quoter_i* p_quoter_i =
    new Quoter_i("Dow Jones");
// depois cria um tie para o Quoter_i:
Quoter_ptr quoter_ref =
    new TIE_Quoter(Quoter_i)(p_quoter_i);
```


Note Que...

- neste exemplo não tivemos que nos preocupar com ativação de objetos
  - » pois nosso servidor implementa um só objeto
  - » e este objeto é criado (ativado) na mainline do servidor
- Isto só funciona para servidores que implementam "poucos" objetos, todos eles ativados na mainline do servidor
  - » "poucos" = "que cabem todos na memória"

No Caso Geral...

- um servidor pode implementar muitos objetos
  - » e guardar num banco de dados (ou em *flat files*) o estado de cada objeto
- Tais objetos não podem estar todos ativos ao mesmo tempo (não cabem na memória)
  - » eles tem que ser ativados/desativados dinamicamente

Orbix Loaders

- No Orbix, ativação de objetos é feita por *loaders*
- Um *loader* é chamado quando uma nova *object reference,* referenciando um objeto implementado num servidor, surge no espaço de endereçamento desse servidor

Orbix Loaders (cont.)

- Um *loader* é chamado caso não esteja ativo
  - » o objeto alvo de uma requisiçao ao servidor
  - » um objeto local referenciado por algum argumento de entrada de uma requisição ao servidor
  - » um objeto local referenciado por algum argumento de saída (ou valor de retorno) de uma requisição que o servidor fez a outro servidor
  - » o objeto referenciado pelo valor de retorno de **`CORBA::ORB::string_to_object()`**, ou da função **`_bind()`** do Orbix

Orbix Loaders (cont.)

- Uma *loader class* definida pelo usuário deve ser derivada de **`CORBA::LoaderClass`**
- Em tempo de execução, o loader é instalado mediante a criação de uma instância da nova *loader class*
- O *loader* pode redefinir certas funções definidas em **`CORBA::LoaderClass`**:
  - » **`load()`**
  - » **`save()`**