INF1406 – 2016.1

Exercício 4: Negociação de Ações

Este trabalho está dividido em três partes apenas para facilitar o seu desenvolvimento. Sugiro implementar cada parte gradualmente, só passando à próxima quando terminar a anterior. Só é necessário entregar um trabalho, desde que contenha os itens das 3 partes.

A implementação de CORBA sugerida em JAVA é o JacORB, versão 3.1 ou maior. Você pode obtê-lo em http://www.jacorb.org/download.html.

Outras implementações incluem:

Java: a própria JDK inclui uma implementação. Note que ela adere a uma versão antiga do padrão CORBA e não recebe suporte e atualizações há muitos anos. Por isso, a sugestão de uso em Java é o JacORB.

LUA: http://www.tecgraf.puc-rio.br/~maia/oil/

C++: http://www.cs.wustl.edu/~schmidt/TAO.html

C#: http://iiop-net.sourceforge.net/

Note que as implementações de Lua e C# não aderem totalmente às especificações de CORBA, portanto podem ser um pouco diferentes do esperado.

O uso de Java é obrigatório mas não será exigida a utilização de mais de uma linguagem, você pode utilizar qualquer combinação que desejar (apenas uma ou várias).

Parte 1: CORBA básico

Nesse exercício, o objetivo é implementar a aplicação distribuída **StockMarket** conforme apresentada em aula. A IDL que define o servidor StockServer é a seguinte:

```
// StockMarket.idl
// O módulo StockMarket consiste das definições
// úteis para desenvolvimento de aplicações
// que lidam com mercado de ações.
module StockMarket {

   // Exceção para definir o uso de um símbolo desconhecido
   exception UnknownSymbol {
     string symbol;
   };

   // O tipo StockSymbol é usado para representar os
   // identificadores das ações
   typedef string StockSymbol;

   // Um StockSymbolList é uma sequência de StockSymbols.
   typedef sequence<StockSymbol> StockSymbolList;

   // A interface StockServer é a interface que fornece
   // as informações sobre as ações do mercado.
   interface StockServer {

      // getStockValue() retorna o valor de uma
      // determinada ação identificada por um
      // StockSymbol fornecido como parâmetro de entrada.
      // Se o StockSymbol dado for desconhecido, a exceção
      // UnknownSymbol deve ser lançada.
      float getStockValue(in StockSymbol symbol) raises
(UnknownSymbol);

      // getStockSymbols() retorna uma sequência com todos os
      // StockSymbols conhecidos pelo servidor do mercado de
      // ações.
      StockSymbolList getStockSymbols();
   };
};
```

Sua solução deve contemplar:

- A compilação do arquivo IDL para gerar as classes (stubs, skeletons, helper, etc) usadas na solução.

- O servant do Objeto CORBA que implementa a interface IDL StockMarket::StockServer. Decida se você usará a estratégia de implementação por herança ou por delegação.

- O servidor que cria o servant e o exporta para o ORB, deixando-o pronto para receber as requisições provenientes de um cliente. Como não usaremos um serviço de nomes para a etapa de publicação, você pode salvar o IOR do objeto em um arquivo, assumindo que o cliente tem acesso a esse mesmo arquivo.

- O cliente que faz chamadas remotas ao objeto CORBA que implementa a interface StockMarket::StockServer. Para recuperar a referência ao objeto CORBA, você pode assumir que o cliente tem acesso ao arquivo onde está o IOR do objeto criado pelo servidor. Ele deve exibir no console todas as ações disponíveis no servidor, com seus valores.

Parte 2: ValueTypes

1. O objetivo desse exercício é experimentar o uso de valuetypes de CORBA. **Acrescente à interface StockServer uma nova operação (invente um nome) que retorne uma sequência com as informações sobre todas as ações disponíveis.** O tipo que contém as informações sobre uma ação deve ser definido como:

```
// O tipo StockInfo possui o nome e o
// respectivo valor de uma ação.
valuetype StockInfo {
  public StockSymbol name;
  public float value;
  // Operação que formata o nome e o valor
  string toString();
};
```

Altere o código do seu cliente para usar a nova operação de StockServer para recuperar todas as informações de ações. Será necessário fornecer a implementação do tipo StockInfo onde o tipo for utilizado (cliente e servidor) e sua fábrica onde o tipo for recebido (cliente).

PERGUNTA 1:
Repare que no cliente será possível imprimir os dados das ações de duas formas:

1. Através dos campos name e value
2. Através da operação _toString()

Tente explicar qual a diferença entre essas duas formas, se houver.

PERGUNTA 2:
Para um CLIENTE que quer imprimir todas as ações disponíveis e seus preços, qual a diferença entre usar esse novo método getStockInfo, ou usar os métodos da parte 1? Qual das duas formas você considera melhor?

Parte 3: Integração com outros serviços

Nesse exercício, o objetivo é demonstrar a integração entre servidores, que ocorre a partir de uma requisição de um cliente.

- O servidor das partes 1 e 2 deve agora implementar duas interfaces: **StockServer e StockExchange.** Ou seja, ele agora deve ter dois objetos remotos. Chamaremos esse programa servidor agora de **StockSeller.** A interface StockServer é definida pela interface IDL StockMarket::StockServer. A interface StockExchange é definida pela interface IDL StockMarket::StockExchange.

```
// A interface StockExchange é a interface que permite
// a compra de ações.
interface StockExchange {
  // Simula a venda de uma ação.
  // Se o StockSymbol for conhecido, após a compra o
  // valor dessa ação é aumentado em 10%, todas as
  // impressoras conhecidas são notificadas e true é
  // retornado. Se o StockSymbol dado for desconhecido,
  // a exceção UnknownSymbol é lançada. Se houver
  // qualquer outro erro relacionado à venda da ação,
  // é retornado false. Erros relacionados à
  // comunicação com impressoras podem ser simplesmente
  // impressos em tela e true deve ser retornado.
  boolean buyStock(in StockSymbol symbol) raises
(UnknownSymbol);

  // Registra a existência de uma impressora,
  // conectando-a ao StockSeller.
  boolean connectPrinter(in ExchangePrinter printer);
};
```

- Criar um outro servidor chamado **StockLogger** que implementa a interface **ExchangePrinter** descrita a seguir. Você deve criar duas implementações dessa interface: uma classe **DisplayExchangePrinter**, cuja implementacão da função **print** direciona a saída para a console, e uma **FileExchangePrinter**, cuja implementação da função **print** direciona a saída para um arquivo qualquer.

```
// A interface ExchangePrinter é a interface que
// representa uma impressora de negociações de ações.
interface ExchangePrinter {
  // Imprime que houve uma negociação da ação indicada.
  // A saída utilizada para impressão não é
  // especificada. Exemplos de saídas: tela, arquivo.
  void print(in StockSymbol symbol);
};
```

- Você precisará de um novo servidor para o **StockLogger**. Assim como no caso do servidor do StockSeller, você pode usar um arquivo para guardar o IOR dos objetos remotos desse novo servidor. Note que você passará a ter, portanto, dois servidores. Um para o StockSeller e outro para o StockLogger, pois são programas diferentes, possivelmente rodando em máquinas diferentes.

- A impressora, ao inicializar, deve fazer a conexão de AMBOS os seus objetos remotos ExchangePrinter no StockSeller, através do objeto StockExchange. Como o IOR desses objetos estão salvos em arquivo, você pode usá-los como parâmetro do seu programa. Assim como no exercício anterior, você pode assumir que clientes têm acesso aos arquivos onde estão os IORs dos objetos criados pelos servidores.

- Altere a aplicação cliente para que use também o método buyStock do StockSeller. Ao chamar esse método, espera-se que as impressoras automaticamente imprimam a negociação da ação.

- Quando executar tudo, execute no mínimo dois processos StockLogger e verifique se todas as saídas foram impressas corretamente ao comprar uma ação. Se possível, execute-os em máquinas diferentes.

Seu trabalho deve contemplar:

- Três projetos Eclipse Java (ou um script responsável por compilar corretamente o código e outro responsável por executar o código, para cada programa): StockSeller, StockLogger e Client.
- O código-fonte da sua solução. A linguagem utilizada deve ser Java, compatível com a JVM 1.6, para ao menos um dos programas.
- Qualquer biblioteca que utilizar (o que inclui o JacORB).
- Respostas para as duas perguntas feitas na Parte 2.

Observações:

- A JVM a ser utilizada deve ser a 1.6.
- O trabalho deve ser feito individualmente ou em duplas.
- O prazo MÁXIMO de entrega do trabalho será no dia 20/06/16, com apresentações nas aulas subsequentes. NÃO ATRASEM POIS NÃO DEVIDO AO FIM DO PERÍODO HAVERÁ NOVA DATA DE APRESENTAÇÃO. APÓS AS DATAS ESPECIFICADAS O ALUNO FICARÁ COM ZERO! Quem quiser entregar e apresentar antes, poderá fazê-lo, basta agendar por email.
- Você pode enviar o seu trabalho e as respostas das duas perguntas por email para caeaugusto@gmail.com, caugusto@inf.puc-rio.br ou levar no dia da apresentação.
- A apresentação será feita durante o horário de aula, na sala de aula. Caso não possa comparecer, você deve enviar o trabalho antes e apresentar antes.
- Durante a apresentação, serão feitas perguntas para avaliar o seu entendimento do problema, da solução, dos erros de concorrência e suas soluções. A nota será baseada nas suas respostas e no fato dos seus programas funcionarem como especificado ou não.

PONTO EXTRA (2 PONTOS):

- Implemente seus servidores usando ao menos 2 linguagens diferentes (por exemplo, StockSeller e Client em Java e StockLogger em C++). Lembre-se de que você será responsável por fazer seus programas rodarem em minha máquina, portanto teste seu programa antes em uma

máquina Windows 10 “vazia” ou crie binários completos sem dependências externas.