



Java Avançado

Exercícios Propostos

Manipulação de Dados nos Formatos XML e JSON

Carlos Tosin

Histórico

Data	Observação
13/05/2011	Criação do documento

Sobre a Softblue

Fundada em 2003 na cidade de Curitiba-PR por André Milani e Carlos Tosin, inicialmente para dedicar-se à construção de soluções para web, a **Softblue** rapidamente expandiu seus negócios para a realização de cursos e treinamentos, devido ao fato de seus sócios serem altamente especializados em determinadas áreas de TI, certificados nas tecnologias em que atuam e autores de livros de informática sobre os mesmos temas.

Atualmente, a **Softblue** disponibiliza cursos e treinamentos on-line, permitindo que pessoas do Brasil inteiro possam realizá-los, estudando e aprendendo novas tecnologias, aprimorando seus conhecimentos para o mercado de trabalho, de acordo com sua disponibilidade de horários.

Exercícios Propostos

Exercício 1

Crie um mecanismo que armazena o estado de um objeto (valores dos atributos) em um arquivo no formato XML. Implemente também o mecanismo inverso, que a partir de um arquivo, reconstroi o objeto com os dados lidos do arquivo.

É importante que estas funcionalidades de gravação e leitura possam ser utilizadas para objetos de quaisquer classes. Para facilitar a codificação, considere que as classes cujos objetos que serão gravados têm apenas atributos cujos tipos são *String*, *int*, *double* ou *boolean*.

A título de exemplo, considere a classe *curso.javaavancado.Produto*, definida da seguinte forma:

Produto
- nome : String - categoria : int - valor : double - vendido : boolean

Se você atribuir os valores “*Produto 1*”, “*2*”, “*100.5*” e “*true*” para os atributos *nome*, *categoria*, *valor* e *vendido*, respectivamente, o XML gerado deve ter o seguinte conteúdo:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<curso.javaavancado.Produto>
  <nome>Produto 1</nome>
  <categoria>2</categoria>
  <valor>100.5</valor>
  <vendido>false</vendido>
</curso.javaavancado.Produto>
```

Perceba que a tag raiz do documento é o nome da classe, enquanto as outras tags correspondem ao nome de cada atributo (o valor do atributo é adicionado em forma de texto, dentro da tag).

O acesso às funcionalidades implementadas deve ser feito através das seguintes classes e métodos (estáticos), que deverão ser criadas por você:

```
void XMLWriter.write(Object obj, OutputStream os) throws Exception;
Object XMLReader.read(InputStream is) throws Exception;
```

Dica 1: Para que este mecanismo funcione é preciso utilizar a Reflection API. Alguns métodos úteis que podem ser utilizados são os seguintes:

- **Class.getDeclaredFields():** Retorna um array com os atributos (*Field*) declarados pela classe.
- **Field.setAccessible():** Permite ler/alterar o valor de um atributo, mesmo que ele esteja marcado como *private*.
- **Field.get(Object):** Lê o valor do atributo do objeto desejado.
- **Field.set(Object, Object):** Atribui um valor ao atributo no objeto desejado.
- **Field.getType():** Retorna o tipo de dado do atributo. Lembre-se de que os dados lidos do XML estão em formato *String*, e devem ser convertidos para o tipo correto antes de serem armazenados no atributo do objeto.

Dica 2: a utilização da API DOM pode facilitar o processo de geração e leitura do XML.

Exercício 2

O XML é um formato bastante utilizado no tráfego de informações. Neste exercício, ele será utilizado como formato de troca de dados numa arquitetura cliente-servidor.

Crie uma calculadora capaz de somar, subtrair, multiplicar e dividir dois números. O detalhe é que as operações desta calculadora devem ser acessadas por clientes remotos, utilizando sockets. E os dados que irão trafegar pelo socket, tanto na requisição como na resposta, devem ser formatados em XML.

Imagine que um cliente quer executar a soma dos valores 10 e 20. O XML que deve ser enviado ao servidor deve ter o seguinte formato:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<request>
  <op>SOMAR</op>
  <valor1>10.0</valor1>
  <valor2>20.0</valor2>
</request>
```

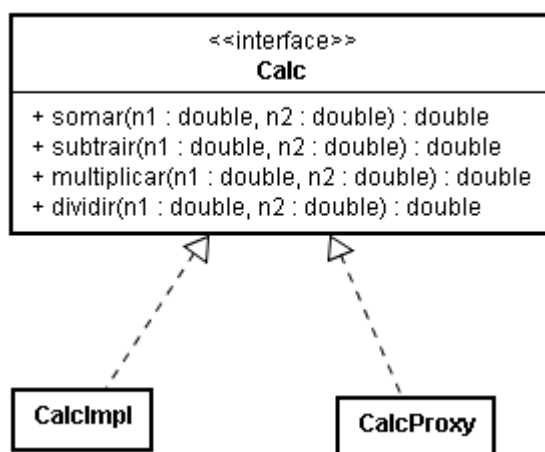
Depois de efetuada a operação, o servidor irá responder também no formato XML, que deve ter o seguinte formato:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<response>
  <result>30.0</result>
</response>
```

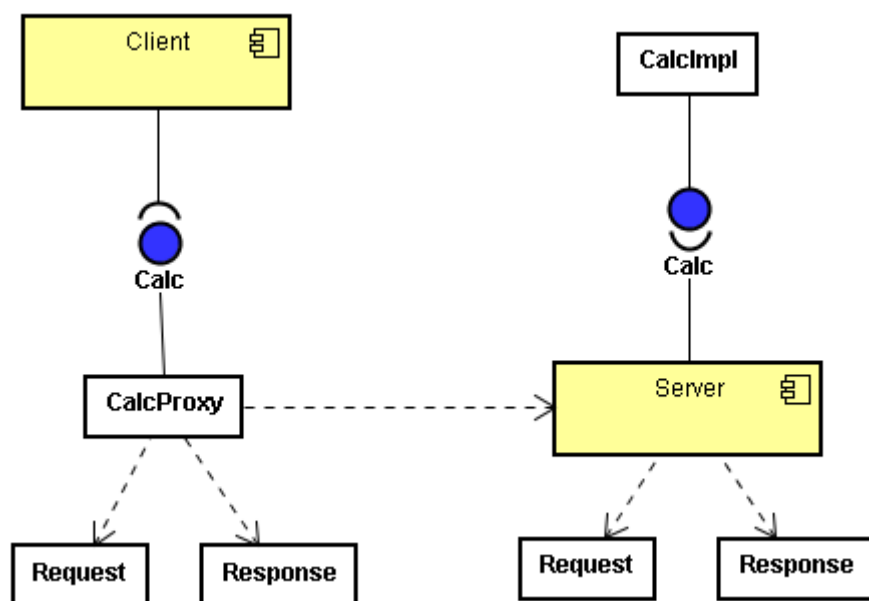
Com as informações acima você já terá condições de implementar a solução completa, de acordo com o que foi solicitado. Mas se você quer um desafio maior, pode tentar implementar a mesma solução de uma forma mais “elegante” e profissional, como explicada abaixo.

O RMI é uma API do Java que permite executar chamadas remotas a métodos. Uma característica interessante desta API é que o código cliente chama o método remoto como se estivesse fazendo uma chamada local. O RMI permite este tipo de situação devido ao conceito de *stub*, que faz o papel de um *proxy*: todas as chamadas que ele recebe são enviadas ao servidor, o qual devolve uma resposta para o stub, que por sua vez retorna a informação para quem o chamou. A ideia é que você implemente este stub (proxy), que encapsulará todas as chamadas remotas e permitirá que o cliente chame a operação matemática como se a chamada fosse local.

Para implementar um proxy você precisará criar uma interface *Calc* e duas classes que implementam esta interface. Observe:



A classe *CalcImpl* é utilizada pelo servidor, e contém a real implementação das operações de soma, subtração, multiplicação e divisão. Já a classe *CalcProxy* é utilizada pelo cliente, e centraliza toda a lógica do acesso remoto. A figura seguinte mostra um diagrama que representa a arquitetura da solução em linhas gerais:



A visão do cliente é a interface *Calc*, onde ele vai invocar as operações necessárias. Quando isto for feito, o *CalcProxy* vai montar um objeto *Request* (que representa a requisição), convertê-lo para XML e enviar os dados ao servidor.

O servidor, ao receber a requisição, cria também um objeto *Request* com base no XML recebido. Ele analisa então a requisição para descobrir qual operação chamar e também quais valores fornecer para o método. Depois disso, o servidor chama o método em *CalcImpl*, que vai efetuar a operação. Com o resultado obtido, o servidor cria agora um objeto *Response* (que representa a resposta à requisição), converte o objeto para XML e devolve a informação para o cliente.

Quando o proxy recebe a resposta à requisição, ele também cria um objeto *Response* com base no XML recebido. Dentro deste objeto estará a resposta da operação, que é retornada então para quem fez a chamada ao método.