02

Serializando e deserializando objetos e coleções em XML

Muitas vezes desejamos ler um arquivo XML ou os dados de uma compra em formato JSON. Para isso podemos transformar os dados a seguir em um objeto do tipo Produto:

```
<produto>
  <nome>geladeira</nome>
  <preco>1000.0</preco>
  <descricao>geladeira duas portas</descricao>
</produto>
```

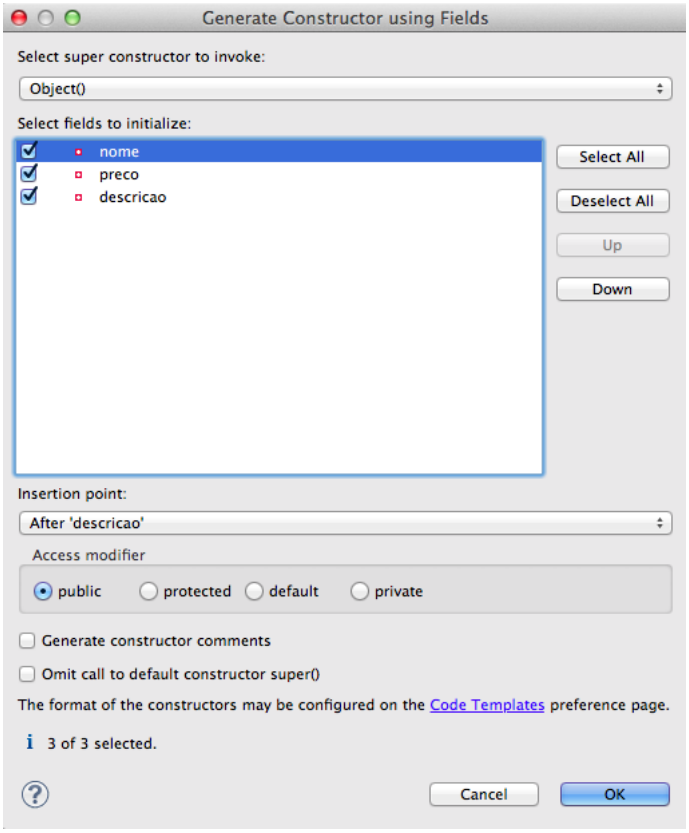
Como fazer isso em Java? Existem diversas alternativas como o JAXB, SAX, DOM etc. Nesse curso veremos o XStream que permite tanto o processo de transformar dados como esses em um objeto do tipo Produto (deserialização), como o processo inverso, dado um produto, gera o código XML ou equivalente (serialização).

Começamos criando um novo projeto no Eclipse. Baixamos a versão mais recente do "binary distribution" do XStream aqui (https://x-stream.github.io/download.html) e descompatamos o arquivo. Copiamos os arquivos lib/xstream-VERSAO.jar e o lib/xstream/xpp3_min-VERSAOc.jar para o nosso projeto, efetuamos o clique da direita em cada um desses jars, escolhemos Build, Add to Build Path.

Vamos criar nossa classe Produto:

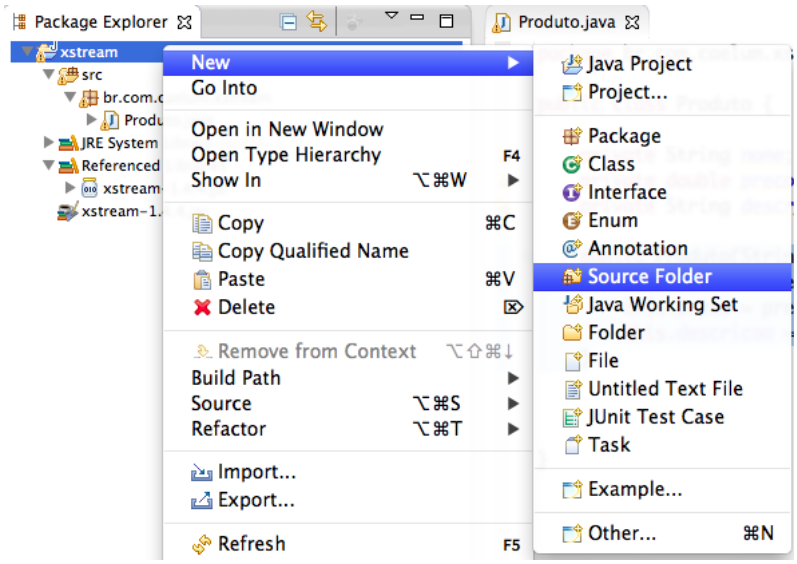
```
package br.com.caelum.xstream;

public class Produto {

    private String nome;
    private double preco;
    private String descricao;

}
```

Adicionamos o construtor com os 3 campos, usando CTRL+3, Generate Constructor Using Fields:

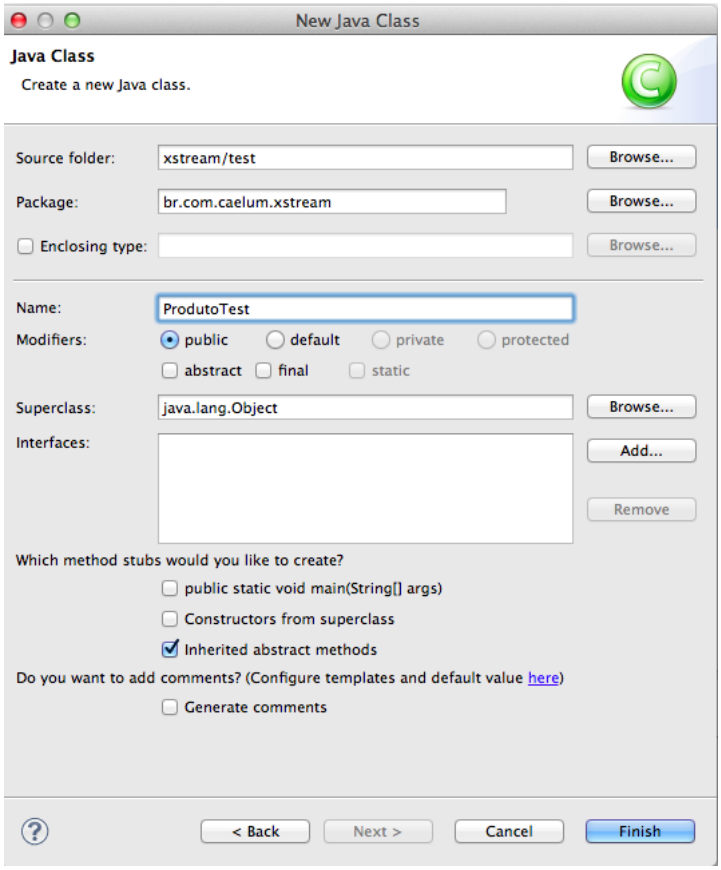



```
public Produto(String nome, double preco, String descricao) {
        this.nome = nome;
        this.preco = preco;
        this.descricao = descricao;
    }
```

Vamos criar nosso primeiro teste de serialização, onde tentamos gerar um xml para um produto. Para isso criaremos um novo diretório de código fonte Java, chamado test. Clique da direita no nome do projeto, New, Source Folder:



O nome do diretório será test. Crie o diretório. Criamos uma nova classe com CTRL+N, Class, onde o pacote será novamente br.com.caelum.xstream mas o nome da classe agora é ProdutoTest e o source folder é test:




```
package br.com.caelum.xstream;

public class ProdutoTest {

}
```

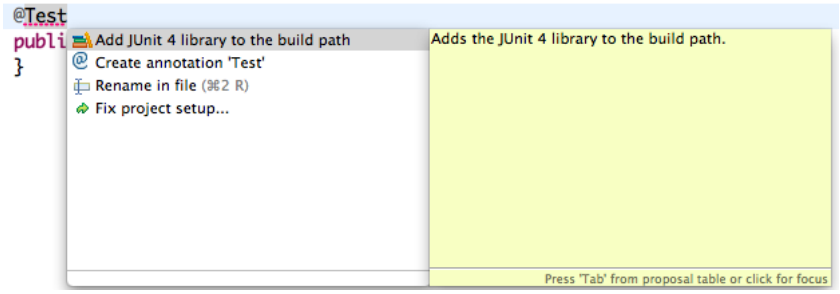
Vamos adicionar um caso de teste, onde queremos que o xml seja aquele que mostramos anteriormente. Por isso o nome do método será "deve gerar o xml com o nome preco e descricao adequados":

```
public void deveGerarOXmlComONomePrecoEDescricaoAdequados() {
}
```

Para criar testes com o JUnit a partir da versão 4, devemos anotar o método com @Test:

```
@Test
public void deveGerarOXmlComONomePrecoEDescricaoAdequados() {
}
```

Mas o JUnit não está em nosso projeto. Com o cursor na linha do @Test, apertamos o atalho CTRL+1, e escolhemos adicionar o JUnit ao projeto:



Vamos ao teste, primeiro esperamos um xml exatamente igual ao que mostramos no começo desse curso:

```
String resultadoEsperado = "<produto>\n" +
        "  <nome>geladeira</nome>\n" +
        "  <preco>1000.0</preco>\n" +
        "  <descricao>geladeira duas portas</descricao>\n" +
        "</produto>";
```

Criamos o nosso produto:

```
Produto geladeira = new Produto("geladeira", 1000, "geladeira duas portas");
```

E pedimos ao XStream, em sua versão mais simples e sem nenhuma configuração, gerar o xml:

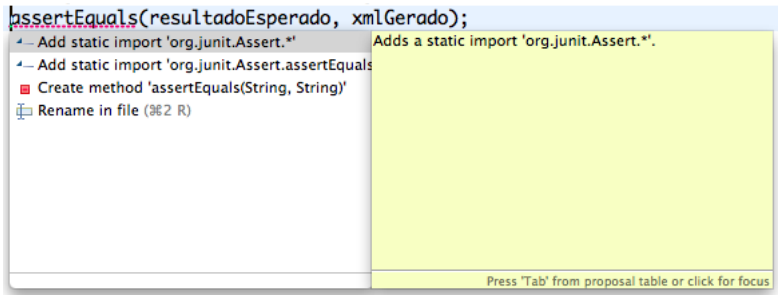
```
XStream xstream = new XStream();
String xmlGerado = xstream.toXML(geladeira);
```

Não esqueça de importar a classe XStream utilizando o atalho CTRL+SHIFT+O.

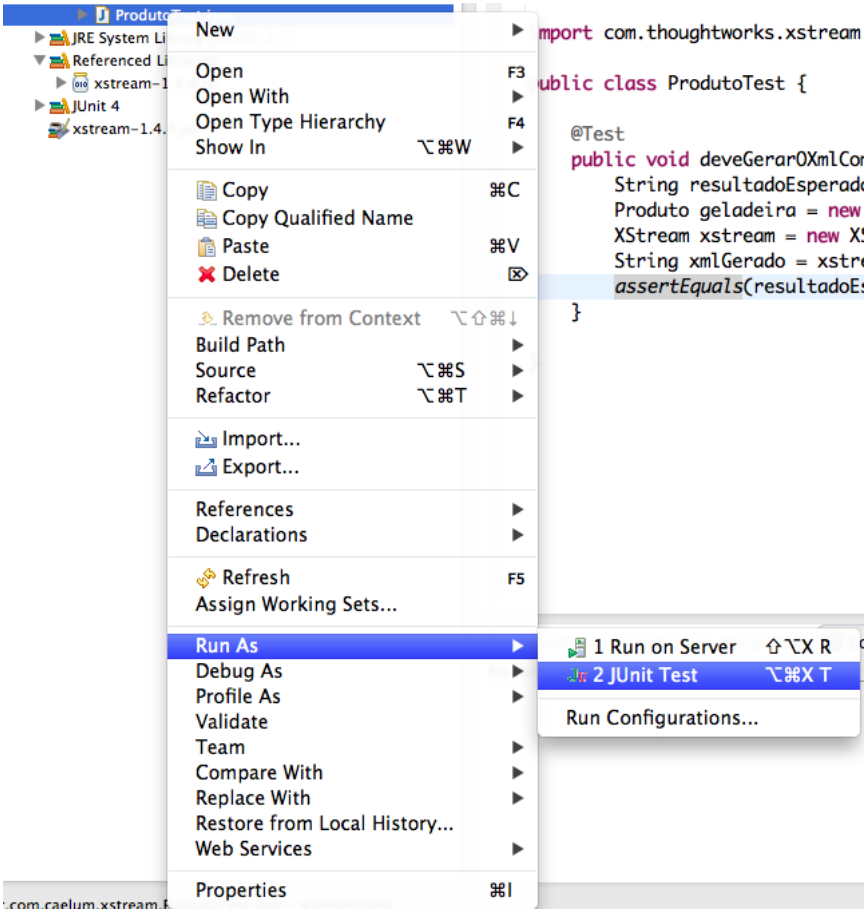
Mas na verdade gostaríamos de conferir que o valor do xmlGerado é exatamente igual o resultadoEsperado, portanto garantimos a igualdade, isto é, assertEquals:

```
assertEquals(resultadoEsperado, xmlGerado);
```

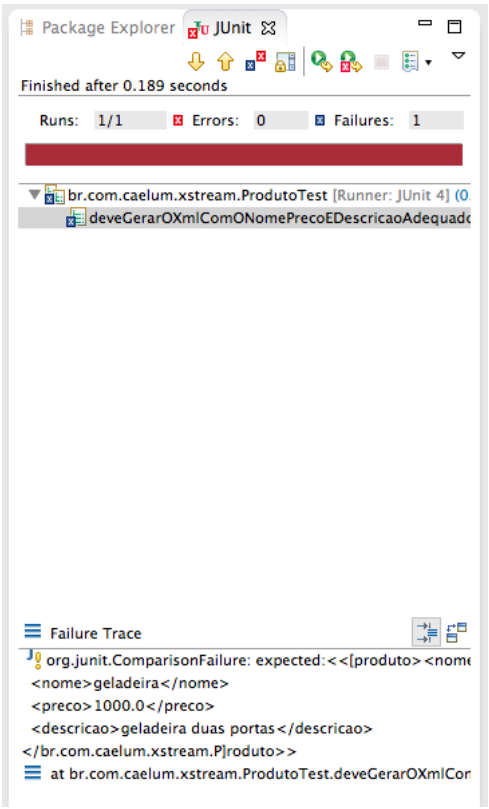
Com o cursor na palavra assertEquals, o atalho CTRL+1 permitirá importar estaticamente o método assertEquals.



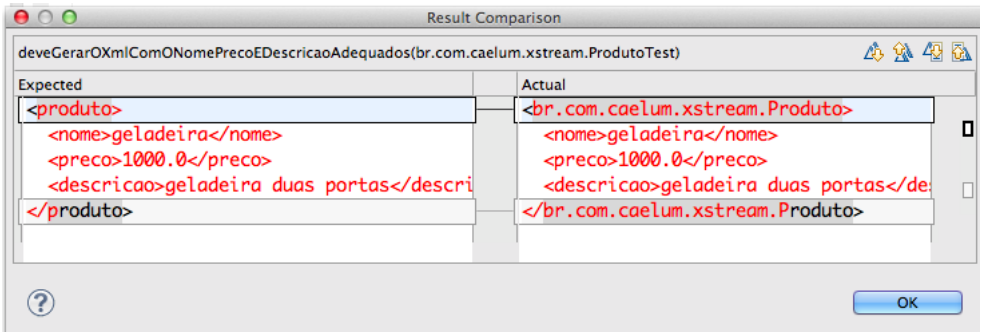
Agora rodamos o teste clicando da direita no arquivo ProdutoTest, Run As, JUnit Test.



Repare que o teste falha e o resultado é uma barra vermelha:



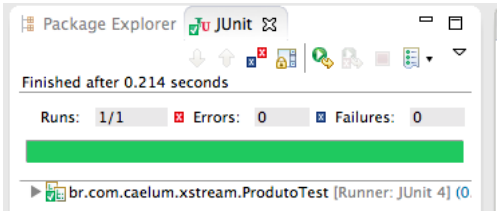
Ao efetuarmos um clique duplo sobre a mensagem "org.junit.ComparisonFailure", o Eclipse mostra o erro na comparação:



Repare que o XStream gerou a tag chamada "br.com.caelum.xstream.Produto", e não a tag "produto" como desejávamos. Como dizer para o XStream efetuar a troca do nome de uma classe? A configuração é bem simples, basta fazer um alias:

```
XStream xstream = new XStream();
xstream.alias("produto", Produto.class);

String xmlGerado = xstream.toXML(geladeira);
```

Agora o resultado é verde:



Ficamos então com o código de teste que define o resultado que queremos chegar (resultadoEsperado), cria o produto (produto), cria e configura o XStream (xstream) e efetua a conversão (xmlGerado). Por fim, ele confirma que o resultado esperado é igual ao xml gerado (assertEquals):

```
package br.com.caelum.xstream;

import static org.junit.Assert.*;

import org.junit.Test;

import com.thoughtworks.xstream.XStream;

public class ProdutoTest {

    @Test
    public void deveGerarOXmlComONomePrecoEDescricaoAdequados() {
        String resultadoEsperado = "<produto>\n" +
                "  <nome>geladeira</nome>\n" +
                "  <preco>1000.0</preco>\n" +
                "  <descricao>geladeira duas portas</descricao>\n" +
                "</produto>";

        Produto geladeira = new Produto("geladeira", 1000, "geladeira duas portas");

        XStream xstream = new XStream();
        xstream.alias("produto", Produto.class);
```

```
        String xmlGerado = xstream.toXML(geladeira);

        assertEquals(resultadoEsperado, xmlGerado);
    }

}
```

Da mesma maneira que fizemos um alias para uma classe, podemos fazê-lo para um campo. Vamos alterar nossa "descricao" para descrição:

```
String resultadoEsperado = "<produto>\n" +
        "  <nome>geladeira</nome>\n" +
        "  <preco>1000.0</preco>\n" +
        "  <descrição>geladeira duas portas</descrição>\n" +
        "</produto>";
```

Para isso configuramos o xstream, dizendo para executar o alias de um campo, usando a palavra descrição para a classe Produto, campo descricao:

```
xstream.aliasField("descrição", Produto.class, "descricao");
```

Rodamos o teste e tudo continua verde. Mas como serializar atributos xml? Imagine que o código de um produto é representado através de um atributo:

```
String resultadoEsperado = "<produto codigo=\"1587\">\n" +
        "  <nome>geladeira</nome>\n" +
        "  <preco>1000.0</preco>\n" +
        "  <descrição>geladeira duas portas</descrição>\n" +
        "</produto>";
```

Vamos adicionar o campo codigo no Produto, e adicionamos esse valor no construtor do Produto:

```
package br.com.caelum.xstream;

public class Produto {

    private String nome;
    private double preco;
    private String descricao;
    private int codigo;

    public Produto(String nome, double preco, String descricao, int codigo) {
        this.nome = nome;
        this.preco = preco;
        this.descricao = descricao;
        this.codigo = codigo;
    }

}
```

Ao construirmos nossa geladeira, passamos o código 1587:

```
Produto geladeira = new Produto("geladeira", 1000, "geladeira duas portas", 1587);
```

Para configurar o XStream, vamos dizer para ele utilizar um atributo para a classe Produto, campo codigo:

```
xstream.useAttributeFor(Produto.class, "codigo");
```

Rodamos o teste novamente e tudo continua verde.