02

Relacionamentos, deserialização e refatoração

Esse capítulo continua onde paramos com o capítulo anterior. Caso você não possua o projeto pronto, pode baixá-lo e importá-lo no Eclipse a partir deste arquivo (https://s3.amazonaws.com/caelum-online-public/XSTREAM/xstream-fim-do-1.zip).

Mas nem todo sistema é composto por um único objeto. É normal existir relacionamentos e coleções de objetos. No nosso caso, uma compra contém diversos produtos:

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

import java.util.ArrayList;
import java.util.List;

public class Compra {

    private int id;
    private List<Produto> produtos = new ArrayList<>();

    public Compra(int id, List<Produto> produtos){
        this.id = id;
        this.produtos = produtos;
    }

}
```

E criaremos agora nosso CompraTest com o teste deveSerializarCadaUmDosProdutosDeUmaCompra que garante que a compra traz todos os produtos em seu xml.

A configuração do XStream é a que vimos até agora, mas qual será o resultado de serializar uma compra? Queremos que o xml seja uma lista de produtos e o id da compra:

```
<compra>
  <id>15</id>
  <produtos>
    <produto codigo="1587">
      <nome>geladeira</nome>
      <preco>1000.0</preco>
      <descrição>geladeira duas portas</descrição>
    </produto>
    <produto codigo="1588">
      <nome>ferro de passar</nome>
      <preco>100.0</preco>
      <descrição>ferro com vaporizador</descrição>
    </produto>
  </produtos>
</compra>
```

Para isso definiremos nosso xml esperado dentro do método:

```
public class CompraTest {

    @Test
    public void deveSerializarCadaUmDosProdutosDeUmaCompra() {

        String resultadoEsperado = "<compra>\n"+
            "  <id>15</id>\n"+
            "  <produtos>\n"+
            "    <produto codigo=\"1587\">\n"+
            "      <nome>geladeira</nome>\n"+
            "      <preco>1000.0</preco>\n"+
            "      <descrição>geladeira duas portas</descrição>\n"+
            "    </produto>\n"+
            "    <produto codigo=\"1588\">\n"+
            "      <nome>ferro de passar</nome>\n"+
            "      <preco>100.0</preco>\n"+
            "      <descrição>ferro com vaporizador</descrição>\n"+
            "    </produto>\n"+
            "  </produtos>\n"+
            "</compra>";
    }
}
```

Primeiro criamos a compra:

```
Produto geladeira = new Produto("geladeira", 1000, "geladeira duas portas", 1587);
Produto ferro = new Produto("ferro de passar", 100, "ferro com vaporizador", 1588);
List<Produto> produtos = new ArrayList<Produto>();
produtos.add(geladeira);
produtos.add(ferro);

Compra compra = new Compra(15, produtos);
```

Depois configuramos o xstream com as mesmas configurações de antes, além de um alias para a compra:

```
XStream xstream = new XStream();
xstream.alias("compra", Compra.class);
xstream.alias("produto", Produto.class);
xstream.aliasField("descrição", Produto.class, "descricao");
xstream.useAttributeFor(Produto.class, "codigo");
```

E então adicionamos nosso assertEquals, mesmo que sabendo que ele não funcionará:

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

assertEquals(resultadoEsperado, xmlGerado);
```

Ficamos com uma classe de teste completa:

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

```
import static org.junit.Assert.assertEquals;

import java.util.Arrays;

import org.junit.Test;

import com.thoughtworks.xstream.XStream;

public class CompraTest {

    @Test
    public void deveSerializarCadaUmDosProdutosDeUmaCompra() {
        String resultadoEsperado = "nao sei...";

        Produto geladeira = new Produto("geladeira", 1000, "geladeira duas portas", 1587);
        Produto ferro = new Produto("ferro de passar", 100, "ferro com vaporizador", 1588);
        List<Produto> produtos = new ArrayList<Produto>();
        produtos.add(geladeira);
        produtos.add(ferro);

        Compra compra = new Compra(15, produtos);

        XStream xstream = new XStream();
        xstream.alias("compra", Compra.class);
        xstream.alias("produto", Produto.class);
        xstream.aliasField("descrição", Produto.class, "descricao");
        xstream.useAttributeFor(Produto.class, "codigo");

        String xmlGerado = xstream.toXML(compra);

        assertEquals(resultadoEsperado, xmlGerado);
    }

}
```

Rodamos o teste e verificamos que o mesmo passa! O XStream já gerou o código de uma lista de produtos.

```
<compra>
  <id>15</id>
  <produtos>
    <produto codigo="1587">
      <nome>geladeira</nome>
      <preco>1000.0</preco>
      <descrição>geladeira duas portas</descrição>
    </produto>
    <produto codigo="1588">
      <nome>ferro de passar</nome>
      <preco>100.0</preco>
      <descrição>ferro com vaporizador</descrição>
    </produto>
  </produtos>
</compra>
```

Mas e o processo de deserialização? Se eu tenho um XML em minhas mãos, como recuperar uma Compra ou um produto? Faremos o processo inverso: dado o mesmo xml anterior, queremos gerar uma compra com dois produtos:

```
@Test
public void deveGerarUmaCompraComOsProdutosAdequados() {
    String xmlDeOrigem = "<compra>\n"+
            "  <id>15</id>\n"+
            "  <produtos>\n"+
            "    <produto codigo=\"1587\">\n"+
            "      <nome>geladeira</nome>\n"+
            "      <preco>1000.0</preco>\n"+
            "      <descrição>geladeira duas portas</descrição>\n"+
            "    </produto>\n"+
            "    <produto codigo=\"1588\">\n"+
            "      <nome>ferro de passar</nome>\n"+
            "      <preco>100.0</preco>\n"+
            "      <descrição>ferro com vaporizador</descrição>\n"+
            "    </produto>\n"+
            "  </produtos>\n"+
            "</compra>";
        XStream xstream = new XStream();
        xstream.alias("compra", Compra.class);
        xstream.alias("produto", Produto.class);
        xstream.aliasField("descrição", Produto.class, "descricao");
        xstream.useAttributeFor(Produto.class, "codigo");
}
```

Ao invés de serializar uma compra, deserializaremos o xml, falando pro **xstream** gerar um objeto **a partir** do **XML**:

```
Compra compraResultado = (Compra) xstream.fromXML(xmlDeOrigem);
```

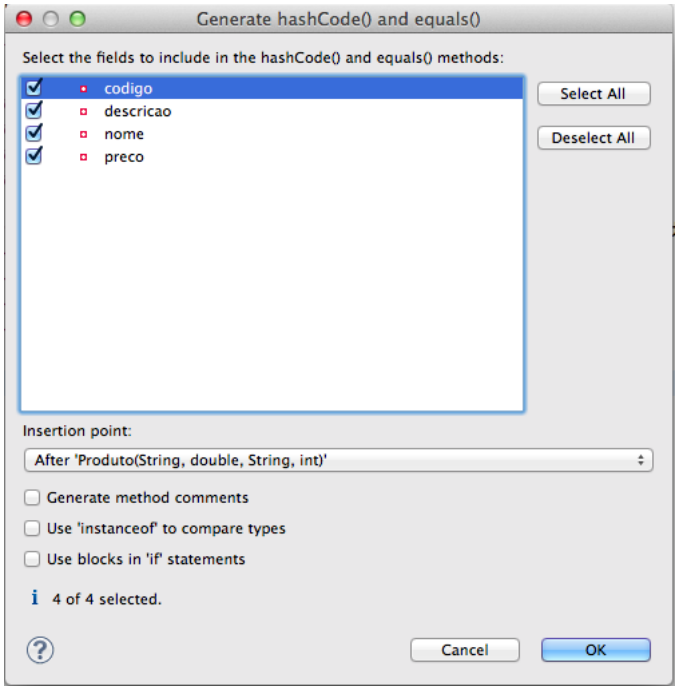
Se rodarmos o teste como está, mesmo sem assert, o programa falha. Ele não encontra a dependência do XStream padrão para deserializar conteúdo xml. É o jar do xmlpull. Copiamos o jar do diretório lib/xstream e adicionamos em nosso projeto.

Agora precisamos comparar essa compra com a compra que criamos no teste anterior:

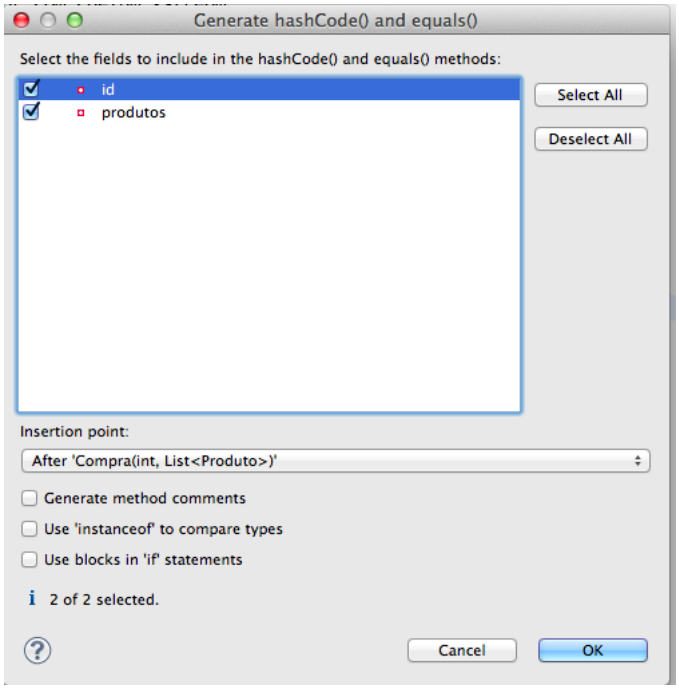
```
Produto geladeira = new Produto("geladeira", 1000,
        "geladeira duas portas", 1587);
Produto ferro = new Produto("ferro de passar", 100,
        "ferro com vaporizador", 1588);
List<Produto> produtos = new ArrayList<Produto>();
produtos.add(geladeira);
produtos.add(ferro);

Compra compraEsperada = new Compra(15, produtos);
assertEquals(compraEsperada, compraResultado);
```

Rodamos o teste e... falha. As compras são distintas? Acontece que não sobrescrevemos o método equals. Vamos na classe Produto e CTRL+3, Generate hashcode and equals:



Geramos o equals também para a classe Compra:



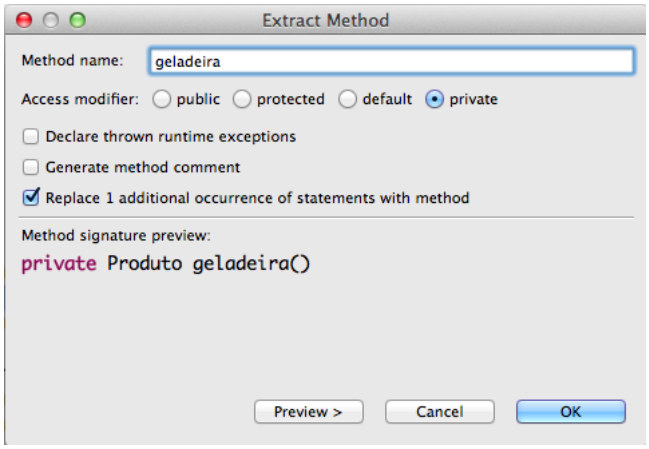
Rodamos novamente o teste, e agora o equals funciona, uma vez que estamos comparando os valores internos de nossas compras!

Continuando o curso, refatoraremos nossos testes para evitar duplicidade de código, veremos como trabalhar melhor as coleções, customizar o XML, criar conversores, evitar certas tags e muito mais.

Vamos refatorar nosso CompraTest para facilitar os testes a seguir. Primeiro extraímos o método que gera a geladeira, selecionando a parte da direita da definição da variável e escolhendo Refactor, Extract Method:

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

O nome do método será **geladeira**:



Extraímos agora o método **ferro** que gera a variável ferro:

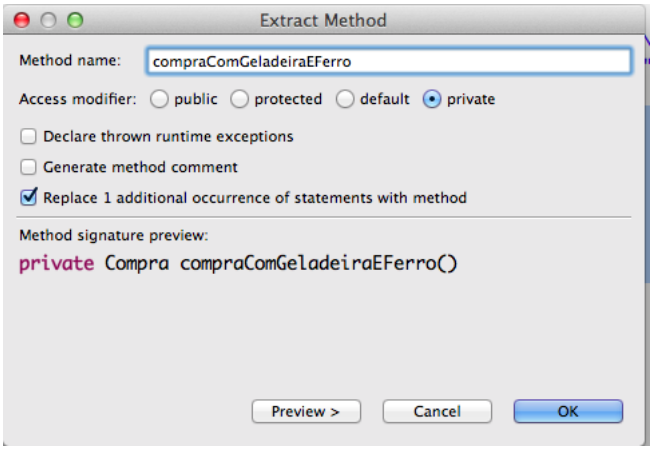
```
private Produto ferro() {
    return new Produto("ferro de passar", 100,
            "ferro com vaporizador", 1588);
}
```

Extraímos um método que cria uma Compra com ferro e geladeira, selecionamos a criação de todas essas variáveis:

```
Produto geladeira = geladeira();
Produto ferro = ferro();
List<Produto> produtos = new ArrayList<Produto>();
produtos.add(geladeira);
produtos.add(ferro);

Compra compra = new Compra(15, produtos);
```

E extraímos o método **compraComGeladeiraEFerro**:



Por fim, extraímos um método para a configuração do XStream para as classes Produto e Compra:

```
XStream xstream = new XStream();
xstream.alias("compra", Compra.class);
xstream.alias("produto", Produto.class);
xstream.aliasField("descrição", Produto.class, "descricao");
xstream.useAttributeFor(Produto.class, "codigo");
```

Chamaremos esse método de **xstreamParaCompraEProduto**: