02

Refatorações, referências, duplicidades e ciclos

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-2.zip).

Já sabemos que o XStream é capaz de serializar e deserializar uma coleção de itens. Se pensarmos que sempre serializamos um único objeto raiz e a partir dele navegamos em seus objetos filho para serem serializados, algumas situações engraçadas podem acontecer. Qual o resultado da serialização de uma compra com duas geladeiras, onde usamos a mesma referência para a geladeira?

```
private Compra compraDuasGeladeirasIguais() {
    Produto geladeira = geladeira();

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

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

Seria o resultado a serialização da mesma geladeira duas vezes, como no código esperado abaixo?

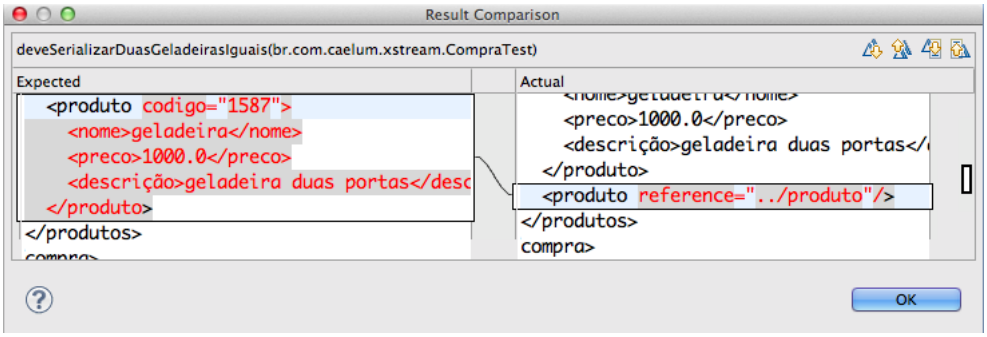
```
String resultadoEsperado = "<compra>\n"
        + "  <id>15</id>\n"
        + "  <produtos>\n"
        + "    <produto codigo=\"1587\">\n"
        + "      <nome>geladeira</nome>\n"
        + "      <preco>1000.0</preco>\n"
        + "      <descrição>geladeira duas portas</descrição>\n"
        + "    </produto>\n"
        + "    <produto codigo=\"1587\">\n"
        + "      <nome>geladeira</nome>\n"
        + "      <preco>1000.0</preco>\n"
        + "      <descrição>geladeira duas portas</descrição>\n"
        + "    </produto>\n"
        + "  </produtos>\n"
        + "</compra>";
```

Portanto escrevemos o código de teste para este xml e a compra acima:

```
@Test
public void deveSerializarDuasGeladeirasIguais() {
    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=\"1587\">\n"
                + "      <nome>geladeira</nome>\n"
                + "      <preco>1000.0</preco>\n"
                + "      <descrição>geladeira duas portas</descrição>\n"
                + "    </produto>\n"
                + "  </produtos>\n"
                + "</compra>";

        Compra compra = compraDuasGeladeirasIguais();

        XStream xstream = xstreamParaCompraEProduto();

        String xmlGerado = xstream.toXML(compra);

        assertEquals(resultadoEsperado, xmlGerado);
    }
```

Mas ao rodar este código, o teste falha. Se efetuarmos o clique duplo, percebemos as diferenças entre o xml esperado e o gerado:



Acontece que quando o XStream detecta que um objeto que já foi serializado está sendo serializado novamente, ele adiciona uma referência para aquele objeto através de um XPath relativo. Com isso ele evita a duplicação da informação no xml gerado, o que é ótimo para ferramentas que fazem serialização e deserialização de dados. Portanto o xml com a referência é:

```
<compra>
  <id>15</id>
  <produtos>
    <produto codigo="1587">
      <nome>geladeira</nome>
      <preco>1000.0</preco>
      <descrição>geladeira duas portas</descrição>
    </produto>
    <produto reference="../produto"/>
  </produtos>
</compra>
```

Podemos alterar o modo XSTREAM_RELATIVE_REFERENCES para uma referência absoluta. Para isso basta configurarmos o XStream para utilizar o tipo de referência baseado em XPath absoluto:

```
XStream xstream = xstreamParaCompraEProduto();
xstream.setMode(XStream.XPATH_ABSOLUTE_REFERENCES);
```

O que resulta em uma pequena modificação no xml:

```
<compra>
  <id>15</id>
  <produtos>
    <produto codigo="1587">
      <nome>geladeira</nome>
      <preco>1000.0</preco>
      <descrição>geladeira duas portas</descrição>
    </produto>
    <produto reference="/compra/produtos/produto"/>
  </produtos>
</compra>
```

O XStream fornece também uma maneira de criar identificadores únicos (IDs) ao invés usar XPath:

```
XStream xstream = xstreamParaCompraEProduto();
xstream.setMode(XStream.ID_REFERENCES);
```

```
<compra id="1">
  <id>15</id>
  <produtos id="2">
    <produto id="3" codigo="1587">
      <nome>geladeira</nome>
      <preco>1000.0</preco>
      <descrição>geladeira duas portas</descrição>
    </produto>
    <produto reference="3"/>
  </produtos>
</compra>
```

Todas essas variações são úteis no momento em que pensamos que desejamos serializar e deserializar. Mas talvez o uso do XStream seja para enviar ou receber um XML específico, e nesse caso queremos a duplicação do XML relativo a geladeira. Para isso basta dizer ao XStream que desejamos NO_REFERENCES:

```
XStream xstream = xstreamParaCompraEProduto();
xstream.setMode(XStream.NO_REFERENCES);
```

E o resultado era o nosso esperado XML:

```
<compra>
  <id>15</id>
  <produtos>
```

```
    <produto codigo="1587">
      <nome>geladeira</nome>
      <preco>1000.0</preco>
      <descrição>geladeira duas portas</descrição>
    </produto>
    <produto codigo="1587">
      <nome>geladeira</nome>
      <preco>1000.0</preco>
      <descrição>geladeira duas portas</descrição>
    </produto>
  </produtos>
</compra>
```

Mas o que aconteceria se houvesse um ciclo em nossos objetos e estamos sempre serializando tudo? Vamos criar a classe Categoria, que terá subcategorias do tipo Categoria:

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

public class Categoria {

    private Categoria pai;
    private String nome;

    public Categoria(Categoria pai, String nome) {
        this.pai = pai;
        this.nome = nome;
    }

    public void setPai(Categoria pai) {
        this.pai = pai;
    }

}
```

Agora criamos diversas categorias, fechando um ciclo:

```
Categoria esporte = new Categoria(null, "esporte");
Categoria futebol = new Categoria(esporte, "futebol");
Categoria geral = new Categoria(futebol, "geral");
esporte.setPai(geral); // fechou o ciclo
```

E tentamos serializar com o XStream com referências baseadas em ID:

```
XStream xstream = new XStream();
xstream.setMode(XStream.ID_REFERENCES);
xstream.aliasType("categoria", Categoria.class);

assertEquals("oqueacontecera?", xstream.toXML(esporte));
```

O resultado é o esperado, um XML que referencia o primeiro elemento na hora do loop:

```
<categoria id="1">
  <pai id="2">
    <pai id="3">
      <pai reference="1"/>
      <nome>futebol</nome>
    </pai>
    <nome>geral</nome>
  </pai>
  <nome>esporte</nome>
</categoria>
```

Portanto atualizamos nosso teste para refletir o que já deveríamos esperar:

```
@Test
public void deveSerializarUmCiclo() {
    Categoria esporte = new Categoria(null, "esporte");
    Categoria futebol = new Categoria(esporte, "futebol");
    Categoria geral = new Categoria(futebol, "geral");
    esporte.setPai(geral); // fechou o ciclo

    XStream xstream = new XStream();
    xstream.setMode(XStream.ID_REFERENCES);
    xstream.aliasType("categoria", Categoria.class);

    String xmlEsperado = "<categoria id=\"1\">\n" +
            "  <pai id=\"2\">\n" +
            "    <pai id=\"3\">\n" +
            "      <pai reference=\"1\"/>\n" +
            "      <nome>futebol</nome>\n" +
            "    </pai>\n" +
            "    <nome>geral</nome>\n" +
            "  </pai>\n" +
            "  <nome>esporte</nome>\n" +
            "</categoria>";
    assertEquals(xmlEsperado, xstream.toXML(esporte));
}
```

Mas o que aconteceria se eu escolhesse o modo sem referências? Rodamos o teste com NO_REFERENCES:

```
xstream.setMode(XStream.NO_REFERENCES);
```

E o resultado é uma exception: CircularReferenceException. É impossível que ao serializar um esporte, ele tente serializar o seu pai, geral, e ao tentar serializar o geral, serialize o futebol, e ao serializar o futebol serialize o esporte e continue assim eternamente. Para evitar essa serialização infinita, o XStream joga uma Exception dizendo que não é possível serializar um ciclo através de NO_REFERENCES.

Esta não é uma limitação da biblioteca, mas sim uma limitação do conceito de serializar cópias sem utilizar referências. Por isso, sempre que podemos ter um loop infinito é importante utilizar algum tipo de referência. Anotamos o teste onde esperamos que seja jogada a Exception:

```
    @Test(expected=CircularReferenceException.class)
```