02

Customizando o processo com conversores próprios

Esse capítulo da continuidade ao projeto terminado anteriormente. Se você ainda não o possui configurado no Eclipse, baixe e importe (https://s3.amazonaws.com/caelum-online-public/XSTREAM/xstream-fim-do-4.zip) o projeto agora mesmo.

Vamos tentar formatar o preço dos produtos para o formato brasileiro ("R$ 1.000,00"). Formatar um número para o padrão brasileiro de moeda em Java é bem simples:

```
Locale brasil = new Locale("pt","br");
NumberFormat formatador = NumberFormat.getCurrencyInstance(brasil);
String valorEmReais = formatador.format(1000.0);
```

Vamos na classe ProdutoTest e no nosso teste original deveGerarOXmlComONomePrecoEDescricaoAdequados vamos dizer os XStream que queremos utilizar um conversor novo para converter na classe Produto o campo preco:

```
xstream.registerLocalConverter(Produto.class, "preco", new PrecoConverter());
```

Claro, o PrecoConverter ainda não existe. Criamos a classe PrecoConverter utilizando o CTRL+1:

```
public class PrecoConverter implements Converter {

    @Override
    public boolean canConvert(Class arg0) {
        return false;
    }

    @Override
    public void marshal(Object arg0, HierarchicalStreamWriter arg1,
            MarshallingContext arg2) {

    }

    @Override
    public Object unmarshal(HierarchicalStreamReader arg0,
            UnmarshallingContext arg1) {
        return null;
    }

}
```

Note que um conversor tem por padrão 3 métodos: um que diz se ele é capaz de converter objetos de determinado tipo; outro que serializa o objeto em uma String e um último que deserializa essa String para objeto. Somos capazes de converter um double para o formato brasileiro, portanto:

```
@Override
public boolean canConvert(Class type) {
```

```
    return type.isAssignableFrom(Double.class);
}
```

Note que o XStream utilizará Double.class e não double.class.

Na hora de serializar (marshal) temos o valor a ser serializado, o writer para onde escrever, e o contexto de serialização. Queremos pegar o valor e fazer um cast para double:

```
Double valor = (Double) value;
```

Fazemos agora a formatação:

```
Double valor = (Double) value;
Locale brasil = new Locale("pt","br");
NumberFormat formatador = NumberFormat.getCurrencyInstance(brasil);
String valorEmReais = formatador.format(valor);
```

E agora escrevemos o valor em reais no nosso output, nossa saída:

```
Double valor = (Double) value;
Locale brasil = new Locale("pt","br");
NumberFormat formatador = NumberFormat.getCurrencyInstance(brasil);
String valorEmReais = formatador.format(valor);
writer.setValue(valorEmReais);
```

Já na hora de deserializar, fazemos o caminho contrário, recebendo o reader e o contexto, somos responsáveis por devolver o valor do preço:

```
@Override
public Object unmarshal(HierarchicalStreamReader reader,
        UnmarshallingContext context) {
    String valor = reader.getValue();
    return // le formatado
}
```

Criamos o formatador, como antes:

```
Locale brasil = new Locale("pt","br");
NumberFormat formatador = NumberFormat.getCurrencyInstance(brasil);
```

E usamos ele para ler o valor formatado:

```
String valor = reader.getValue();
Locale brasil = new Locale("pt","br");
NumberFormat formatador = NumberFormat.getCurrencyInstance(brasil);
try {
    return formatador.parse(valor);
} catch (ParseException e) {
```

```
        throw new ConversionException(e);
    }
```

Note como é simples criar um conversor qualquer que cria um nó com um valor simples. Alteramos nosso teste para o preço esperado ser o formatado em moeda brasileira:

```
String resultadoEsperado = "<produto codigo=\"1587\">\n" +
        "  <nome>geladeira</nome>\n" +
        "  <preco>R$ 1.000,00</preco>\n" +
        "  <descrição>geladeira duas portas</descrição>\n" +
        "</produto>";
```

Rodamos nosso teste e temos sucesso. Mas em casos simples como este onde transformaremos um objeto ou tipo primitivo em uma única String e vice-versa, podemos implementar um SingleValueConverter:

```
public class PrecoSimplesConverter implements SingleValueConverter {
    @Override
    public boolean canConvert(Class type) {
        return type.isAssignableFrom(Double.class);
    }
}
```

E adicionamos os dois métodos que tem o mesmo código de nosso converter anterior, mas agora sem usar o writer ou o reader:

```
@Override
public Object fromString(String valor) {
    Locale brasil = new Locale("pt","br");
    NumberFormat formatador = NumberFormat.getCurrencyInstance(brasil);
    try {
        return formatador.parse(valor);
    } catch (ParseException e) {
        throw new ConversionException(e);
    }
}

@Override
public String toString(Object value) {
    Double valor = (Double) value;
    Locale brasil = new Locale("pt","br");
    NumberFormat formatador = NumberFormat.getCurrencyInstance(brasil);
    String valorEmReais = formatador.format(valor);
    return valorEmReais;
}
```

Conversores que implementam SingleValueConverter são muito práticos para configurações simples como nesse caso do preço:

```
xstream.registerLocalConverter(Produto.class, "preco", new PrecoSimplesConverter());
```