02

Mapeando objeto para um XML

Transcrição

No último vídeo aprendemos a mapear um arquivo XML para um objeto, e agora tentaremos gerar um XML a partir de um objeto `Venda`. Nesse processo, continuaremos utilizando a nossa classe `MapeaXMLDireto`, pois ainda estamos realizando o mapeamento de um XML, mas separaremos nossas ações em dois métodos: `xmlParaObjeto()` e `objetoParaXml()`, com ambos recebendo um objeto `jaxbContext` como argumento. O método `xmlParaObjeto()` receberá grande parte da lógica que criamos no vídeo anterior.

```
public class MapeiaXMLDireto {
    public static void main(String[] args) throws Exception {
        JAXBContext jaxbContext = JAXBContext.newInstance(Venda.class);

        xmlParaObjeto(jaxbContext);
        objetoParaXml(jaxbContext);
    }

    private static void objetoParaXml(JAXBContext jaxbContext) throws Exception {

    }

    private static void xmlParaObjeto(JAXBContext jaxbContext) throws Exception {
        Unmarshaller unmarshaller = jaxbContext.createUnmarshaller();

        Venda venda = (Venda) unmarshaller.unmarshall(new File("src/vendas.xml"));
        System.out.println(venda);
    }
}
```

No método `objetoParaXml()`, utilizaremos o objeto `jaxbContext` para chamarmos o método `createMarshaller()`, que criará uma classe capaz de mapear um objeto para um XML. Associaremos o retorno a uma variável local `marshaller` e, a partir dela, chamaremos o método `marshall()`, passando como argumento o objeto com o qual estamos trabalhando, `Venda`.

Após instanciarmos a `Venda`, chamaremos o método `setFormaDePagamento()` com o texto "Crediario". Também criaremos uma lista de produtos recebendo uma instância de `ArrayList<>`, e adicionaremos alguns produtos a ela com `produtos.add()`. Aproveitaremos o momento de criação do produto para passarmos o nome e o preço. Por fim, chamaremos `venda.setProdutos()` e passaremos nossos `produtos` ao objeto `venda` que criamos anteriormente.

Passaremos a `venda` como primeiro argumento do método `marshal()`. Como segundo argumento, ele pede um "writer", que é basicamente um local para escrever o resultado da conversão. Poderíamos escrever um arquivo, mas optaremos por imprimir no console. Para isso, criaremos uma instância de `StringWriter` e a passaremos como argumento do método. Por fim, faremos um `System.out.println()` de `writer.toString()`.

```
private static void objetoParaXml(JAXBContext jaxbContext) throws Exception {
    Marshaller marshaller = jaxbContext.createMarshaller();
    Venda venda = new Venda();
    venda.setFormaDePagamento("Crediario");
```

```
        List<Produto> produtos = new ArrayList<>();
        produtos.add(new Produto("Livro de java", 59.90));
        produtos.add(new Produto("Livro de  xml", 59.90));
        produtos.add(new Produto("Livro de O.O.", 59.90));
        venda.setProdutos(produtos);
        StringWriter writer = new StringWriter();

        marshaller.marshal(venda, writer);
        System.out.println(writer.toString());

    }
```

Executando nosso código, teremos como retorno o arquivo XML gerado, com as tags `<venda>`, `<formaDePagamento>` (com o valor "Crediario") e `<produtos>`, além de cada `<produto>` individual criado e as tags de seus respectivos atributos (`<nome>` e `<preco>`). Assim, conseguimos mapear uma classe para um arquivo XML de maneira relativamente simples.

Parabéns! Você concluiu o curso de Java e XML da Alura. Esperamos que você tenha gostado e, se tiver qualquer dúvida, não deixe de fazer uma postagem lá no fórum. Bons estudos e até a próxima!