03

Acessando mais de um produto

Transcrição

Agora que aprendemos a pegar o conteúdo de uma tag do nosso XML, vamos incorporar mais informações a ele. Afinal, quando entramos em uma loja virtual é possível comprar quantos produtos quisermos. Pensando nisso, adicionaremos outro produto:

```
<?xml version="1.0" encoding="UTF-8"?>
<venda>
    <formaDePagamento>Cartão</formaDePagamento>
    <produto>
        <nome>Livro de xml</nome>
        <preco>29.90</preco>
    </produto>
        <produto>
        <nome>Livro de O.O. java</nome>
        <preco>29.90</preco>
    </produto>
</venda>
```

Também poderíamos ter outros dados na nossa venda além de `<formaDePagamento>`, como o `<endereco>` de entrega. Note que, como essas tags estão no mesmo nível, poderíamos nos perder em relação à ordem - por exemplo, com um endereço intercalando os produtos.

```
<?xml version="1.0" encoding="UTF-8"?>
<venda>
    <formaDePagamento>Cartão</formaDePagamento>
    <produto>
        <nome>Livro de xml</nome>
        <preco>29.90</preco>
    </produto>
    <endereco></endereco>
        <produto>
        <nome>Livro de O.O. java</nome>
        <preco>29.90</preco>
    </produto>
</venda>
```

Essa construção está bastante desorganizada, não? Para facilitarmos a organização, agruparemos os produtos em uma única tag `<produtos>`.

```
<?xml version="1.0" encoding="UTF-8"?>
<venda>
    <formaDePagamento>Cartão</formaDePagamento>
    <produtos>
        <produto>
            <nome>Livro de xml</nome>
            <preco>29.90</preco>
```

```
        </produto>
            <produto>
            <nome>Livro de O.O. java</nome>
            <preco>29.90</preco>
        </produto>
    </produtos>
</venda>
```

Agora queremos exibir os nomes de todos os produtos. Começaremos renomeando nossas variáveis para adequá-las ao objeto que estamos buscando.

```
public class Sistema {
    public static void main(String[] args) throws SAXException, IOException, ParserConfiguratio
        DocumentBuilderFactory fabrica = DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = fabrica.newDocumentBuilder();
        Document document = builder.parse("src/vendas.xml");

        NodeList nomeProduto = document.getElementsByTagName("nome");
        Element fdp= (Element) nomeProduto.item(0);
        String nome = fdp.getTextContent();
        System.out.println(nome);
    }

}
```

Executando o código dessa forma, teremos como retorno somente o nome do primeiro produto, "Livro de xml". Se quiséssemos pegar o próximo produto, precisaríamos mudar o argumento do método `item()` para `1`, obtendo o retorno "Livro de O.O. em java". Mas queremos todos os produtos recebidos no XML, não é? Pensando nisso, renomearemos a variável `nomeProduto` para `produtos`.

Se o retorno de `getElementsByTagName()` é como uma lista, podemos percorrê-la utilizando um `for`. Criaremos então um contador `i`. Com ela, vamos iterar até o tamanho da lista `produtos` com `getLenght()`, incrementando cada iteração com `i++`. Dentro da estrutura de repetição, colocaremos o código que pega o nome dos produtos, chamando de `produto` o item atual. Feito isso, ao invés de passarmos um índice fixo para o método `item()`, passaremos o nosso contador `i` e, por fim, executaremos o código.

```
NodeList produtos = document.getElementsByTagName("nome");

for(int i = 0;i < produtos.getLength();i++) {
    Element produto = (Element) produtos.item(i);
    String nome = produto.getTextContent();
    System.out.println(nome);

}
```

Teremos como retorno:

Livro de xml

Livro de O.O. java

Nosso código funcionou! Mas e se quiséssemos exibir o nome e o preço? Da maneira que nosso código está agora, teríamos que criar outra lista,e assim sucessivamente para cada atributo desejado. No XML, nós representamos um produto por tags, e no Java essa representação geralmente se dá por meio de uma classe. Pensando assim, ao invés de simplesmente pegarmos o valor de uma tag, criaremos uma instância de uma classe `Produto` .

Em "src", pressionaremos "Ctrl + N" criaremos um novo pacote `br.com.alura.Model` , onde armazenaremos os nossos modelos. Repetiremos o "Ctrl + N", desta vez para criarmos a classe `Produto.java` . Feito isso, voltaremos a `Sistema.java` e, dentro do laço de repetição, instanciaremos a classe `Produto` , atribuindo-a a uma variável `prod` . Essa instância receberá como argumentos o `nome` e o `preco` do produto. Sendo assim, criaremos também um double `preco` que, por enquanto, receberá outra chamada de `getTextContent()` .

```
for(int i = 0;i < produtos.getLength();i++) {
    Element produto = (Element) produtos.item(i);
    String nome = produto.getTextContent();
    double preco = produto.getTextContent();
    Produto prod = new Produto(nome, preco);

    System.out.println(nome);

}
```

Com o auxílio do Eclipse, criaremos o construtor `Produto()` que recebe os argumentos citados. Nele, faremos a atribuição dos respectivos valores com o `this` , criaremos os atributos privados `nome` e `preco` e geraremos seus getters e setters

```
package br.com.alura.Model;

public class Produto {

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public double getPreco() {
        return preco;
    }

    public void setPreco(double preco) {
        this.preco = preco;
    }

    private String nome;
    private double preco;

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

```
}
```

No `Sistema`, precisamos de uma maneira de realmente pegar o `preco`, afinal `produto.getTextContent()` também nos retornará ao nome. Para termos acesso a todas essas informações do produto, ao invés de pegarmos a tag `nome` em `getElementsByTagName()`, pegaremos `produto`. Feito isso, podemos chamar `produto.getElementsByTagName("nome").item(0).getTextContent()` para conseguirmos o nome, e `produto.getElementsByTagName("preco").item(0).getTextContent()` para o preço. Este último nos retornará um erro, já que estamos tentando associar uma string a um double. Para resolvermos esse problema, chamaremos `Double.parseDouble()`. Por fim, exibiremos o `prod` na tela com o `System.out.println()`.

```
NodeList produtos = document.getElementsByTagName("produto");

for(int i = 0;i < produtos.getLength();i++) {
    Element produto = (Element) produtos.item(i);
    String nome = produto.getElementsByTagName("nome").item(0).getTextContent();
    double preco = Double.parseDouble(produto.getElementsByTagName("preco").item(0).getTextCont
    Produto prod = new Produto(nome, preco);

    System.out.println(prod);
```

Como retorno, teremos:

```
br.com.alura.Model.Produto@4590c9c3

br.com.alura.Model.Produto@5afa04c
```

Ou seja, recebemos o endereço de memória. Isso porque, para exibirmos os valores no `System.out.println()`, precisaremos criar um método `toString()` na classe `Produto`. Sendo assim, faremos um `@Override` e sobrescreveremos o método `toString()` definido pela classe `Object`. No caso, retornaremos o nome do produto e o preço.

```
@Override
public String toString() {

    return "Nome:"+ nome + "\nPreço:"+preco+"\n";
}
```

Rodando nosso código novamente, teremos:

```
Nome:Livro de xml

Preço:29.9

Nome:Livro de O.O. java

Preço:29.9
```

Assim, conseguimos trabalhar com vários produtos recebidos em um XML de maneira relativamente fácil. No próximo vídeo trabalharemos com outras situações desse tipo.