01

# Desempenho do sistema

### Transcrição

Até o momento nós definimos um arquivo XML contendo nossas vendas, validamos esse arquivo usando um XSD e exibimos as suas informações no console com o código criado na classe `Sistema.java`. Vamos analisar esse código.

```
package br.com.alura.Teste;

import java.io.IOException;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

import br.com.alura.Model.Produto;

public class Sistema {
    public static void main(String[] args) throws SAXException, IOException, ParserConfiguratio
        DocumentBuilderFactory fabrica = DocumentBuilderFactory.newInstance();
        fabrica.setValidating(true);
        fabrica.setNamespaceAware(true);
        fabrica.setAttribute("http://java.sun.com/xml/jaxp/properties/schemaLanguage", "http://


        DocumentBuilder builder = fabrica.newDocumentBuilder();
        Document document = builder.parse("src/vendas.xml");

        Element venda = document.getDocumentElement();
        String moeda = venda.getAttribute("moeda");
        System.out.println(moeda);

        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).get
            Produto prod = new Produto(nome, preco);

            System.out.println(prod);
        }

    }

}
```

Nele, iniciamos uma fábrica, setamos alguns parâmetros e pedimos para o `DocumentBuilder` construir o arquivo `vendas.xml`. Após a leitura do arquivo, ele nos devolve um `Document`, a partir do qual podemos acessar qualquer elemento, como `formaDePagamento` ou `produto`. Repare, então, que o método `builder.parse()` carrega o documento inteiro na memória, e ainda assim conseguimos encontrar uma tag rapidamente. Para isso, o Java monta uma estrutura parecida com uma árvore, na qual cada tag é atribuída a um nó que, por sua vez, pode ter diversos ramos com outros nós - no caso, `venda` tem um nó `formaDePagamento` e um nó `produtos`, que por sua vez tem um nó `produto`, e assim por diante.

A partir dessa árvore, conseguimos fazer nossas consultas. E qual o problema desse processo? Como temos um arquivo com apena duas vendas, o código é executado rapidamente e não temos problema nenhum. Mas e se estivéssemos lendo um relatório de vendas anuais de um e-commerce grande? Será que esse arquivo caberia na memória de uma só vez? Mesmo que isso fosse verdadeiro, o que acontece se quisermos utilizar apenas um pedaço do arquivo, como as vendas de abril?

Ou seja, em algumas situações a abordagem que utilizamos para ler um XML acaba não sendo interessante, justamente por esse custo de memória. Nesse capítulo aprenderemos outra forma de lermos um arquivo XML sem tamanho custo de memória.