Desenvolvimento Web I

Aula 12 - Desenvolvendo uma livraria virtual – Parte 1

Apresentação

Até agora, você viu diversos conceitos e tecnologias para programação Web em Java. Nesta aula, você verá o desenvolvimento de uma **aplicação utilizando o conhecimento já aprendido**. Essa aplicação é um sistema para ser utilizado por uma livraria que venda seus livros pela internet.

Como atividade, você irá desenvolver uma aplicação similar àquela desenvolvida durante a aula. **Será uma aplicação de Locadora virtual de filmes, em que seus filmes podem ser locados pela internet**. Apesar de serem sistemas simplificados em relação ao que acontece no mercado, esses são bastante úteis para desenvolver e fixar o raciocínio de programação Web usando as tecnologias apresentadas.

Vídeo 01 - Apresentação

Objetivos

- Descrever os primeiros passos que devem ser realizados antes de se começar a realizar a programação das interfaces web: modelagem do negócio da aplicação.

Modelagem do negócio

O sistema a ser desenvolvido nesta aula é uma livraria on-line simplificada, porém, com recursos suficientes para demonstrar a capacidade da tecnologia de Servlets e de JSP para o desenvolvimento de programas web. Esse exemplo de sistema foi baseado no Duke's Bookstore Example, encontrado no The Java EE 5 Tutorial, o qual é fornecido gratuitamente pela em presa responsável pela linguagem Java.

Para iniciarmos o desenvolvimento desse sistema, podemos começar com as classes que representam as regras de negócio da livraria.

A primeira classe a ser definida é a Livro, a qual define as informações que descrevem um livro no sistema da livraria on-line. O código-fonte dessa classe é mostrado na Listagem 1. Note que a classe Livro possui atributos para representar um código identificador do livro (String idLivro, linha 4), título, nome dos autores, ano de publicação, preço, quantidade em estoque e uma crítica da obra (descrição, comentário sobre o livro).

```java
package livraria.negocio;

public class Livro {
    private String idLivro;
    private String titulo;
    private String autores;
    private int ano;
    private double preco;
    private int quantidade;
    private String descricao;

    public Livro() {
    }

    public String getIdLivro() {
        return idLivro;
    }

    public void setIdLivro(String idLivro) {
        this.idLivro = idLivro;
    }

    public String getTitulo() {
        return titulo;
    }

    public void setTitulo(String titulo) {
        this.titulo = titulo;
    }

    // Demais métodos get e set foram omitidos
}
```

Listagem 1 - Classe que representa um Livro

Atividade 01

---

1. Crie um projeto para o sistema de Livraria on-line e implemente a classe Livro, conforme apresentado na Listagem 1.

2. Crie um outro projeto para o sistema de Locadora virtual de filmes e modele uma classe para representar um filme.

Os livros serão escolhidos pelo usuário e acondicionados em um carrinho de compras virtual. Dessa forma, precisamos modelar também uma classe para representar esse carrinho virtual e seu conteúdo. Como um carrinho de compras

possui itens a serem comprados, com informação sobre o tipo do produto e a quantidade a ser comprada, primeiro modelamos a classe ItemCompra, como mostrado na Listagem 2. O atributo item armazena o livro escolhido e o atributo quantidade indica a quantidade desse livro a ser comprada.

```
1  package livraria.negocio;
2
3  public class ItemCompra {
4     private Livro item;
5     private int quantidade;
6
7     public ItemCompra(Livro prod) {
8        item = prod;
9        quantidade = 1;
10    }
11
12    public void incrementaQuantidade() {
13       quantidade++;
14    }
15
16    public void decrementaQuantidade() {
17       quantidade--;
18    }
19
20    public Livro getItem() {
21       return item;
22    }
23
24    public int getQuantidade() {
25       return quantidade;
26    }
27
28    public void setQuantidade(int quantity) {
29       this.quantidade = quantity;
30    }
31 }
```

Listagem 2 - Classe responsável por representar um item (produto, quantidade) do carrinho de compras.

Com relação ao carrinho de compras, podemos modelá-lo por meio da classe CarrinhoCompras, mostrada na Listagem 3.

Pode-se dizer que um carrinho de compras possui uma lista ou um conjunto de itens (ItemCompra) a serem comprados. Esse pensamento está correto. Entretanto, o conteúdo do carrinho de compras foi modelado como um mapeamento (interface

Map, linha 6) de String em objetos do tipo ItemCompra. O motivo disso será discutido adiante.

Vídeo 02 - Coleções

O uso de <String, ItemCompra> (Java Generics) é opcional, mas fortemente recomendado, já que garante que o tipo da chave do mapeamento seja String - mesmo tipo do atributo **idLivro**, que é o identificador de um livro - e que o valor associado a cada chave seja do tipo ItemCompra.

Importante!

Para maior esclarecimento sobre esse assunto, leia a referência sobre Java Generics indicada nas leituras complementares.

Decidimos utilizar um Map porque as operações de adicionar ou de remover um livro do carrinho requerem a busca no carrinho pelo objeto ItemCompra relacionado àquele livro. Isso pode ser visto nos métodos adicionar() e remover(), definidos, respectivamente, nas linhas 12 e 22. Sobre o primeiro método, ele recebe o livro escolhido pelo usuário (representado pelo parâmetro book) e realiza a busca para verificar se já existe um objeto ItemCompra para o livro recebido como parâmetro.

Observe, na linha 13, o uso do método containsKey() existente na classe Map (variável items). Esse método retorna true, se existir, no mapeamento, uma entrada que mapeie o código do livro (book.getIdLivro()) no objeto ItemCompra associado a esse produto. Se esse for o caso, isso quer dizer que o cliente já tinha selecionado esse livro antes, então, simplesmente, recuperamos o objeto ItemCompra (método items.get(book.getIdLivro()) associado ao código do livro sendo comprado e incrementamos sua quantidade no carrinho de compras. Caso contrário, criamos

um novo objeto ItemCompra (linha 17) e o adicionamos no mapeamento (linha 18). Note que mapeamos o objeto utilizando o método put() do mapeamento (linha 18), em que mapeamos o código do livro (book.getIdLivro()) no objeto criado (novoItem).

O uso do mapeamento na implementação do carrinho de compras facilita e torna mais eficiente a implementação das operações do carrinho, as quais poderiam ser feitas também por meio de listas ou conjuntos.

```
package livraria.negocio;

import java.util.*;

public class CarrinhoCompras {
   Map<String, ItemCompra> itens;

   public CarrinhoCompras() {
      itens = new HashMap<String, ItemCompra>();
   }

   public synchronized void adicionar(Livro book) {
      if (itens.containsKey(book.getIdLivro())) {
         ItemCompra item = itens.get(book.getIdLivro());
         item.incrementaQuantidade();
      } else {
         ItemCompra novoItem = new ItemCompra(book);
         itens.put(book.getIdLivro(), novoItem);
      }
   }

   public synchronized void remover(String idLivro) {
      if (itens.containsKey(idLivro)) {
         ItemCompra item = itens.get(idLivro);
         item.decrementaQuantidade();
         if (item.getQuantidade() <= 0) {
            itens.remove(idLivro);
         }

      }
   }

   public synchronized List<ItemCompra> getItens() {
      List<ItemCompra> resultado = new ArrayList<ItemCompra>();
      resultado.addAll(this.itens.values());
      return resultado;
   }

   protected void finalize() throws Throwable {
      itens.clear();
   }

   public synchronized int getNumeroItens() {
      int numeroItens = 0;
      for (ItemCompra item : getItens()) {
         numeroItens += item.getQuantidade();
      }
      return numeroItens;
   }

   public synchronized double getTotal() {
```

```
52        double total = 0.0;
53        for (ItemCompra item : getItens()) {
54            Livro book = item.getItem();
55            total = total + (item.getQuantidade() * book.getPreco());
56        }
57        return total;
58    }
59
60    public void limpar() {
61        itens.clear();
62    }
63 }
```

Listagem 3 - Classe que representa um carrinho de compras virtual

O método remover(), na linha 22, só vai fazer algo caso exista uma entrada no mapeamento para o código do livro passado como parâmetro. Se esse for o caso, basta recuperar o objeto ItemCompra mapeado (linha 24) e decrementar, em uma unidade, a quantidade de livros a ser comprada (método decrementaQuantidade()).

Vídeo 03 - Mapas

Por fim, caso a quantidade de livros restante a ser comprada seja menor ou igual a zero, isso quer dizer que nenhum livro daquele tipo será comprado. Sendo assim, removemos do mapeamento a entrada relativa a esse livro (linha 27), fazendo com que esse item saia do carrinho de compras.

É interessante destacar alguns outros métodos, como o getItens(), da linha 33, o qual retorna o conteúdo do carrinho de compras, ou seja, todos os itens a serem comprados. Para isso, criamos uma nova lista de objetos ItemCompra e adicionamos nessa lista todo os itens existentes no carrinho de compras. Isso é feito por meio do método addAll(), o qual recebe uma coleção de objetos (conteúdo do carrinho) e adiciona todos esses elementos na lista criada.

Outro método do carrinho de compras é o getTotal(), linha 51, o qual percorre todos os elementos do carrinho de compras e contabiliza o valor total a ser gasto com a compra desses produtos. Para isso, o método leva em conta a quantidade de

livros a serem comprados e o preço unitário. A variável total é utilizada como acumulador (linha 55) durante o cálculo do valor total a ser gasto na compra dos livros selecionados.

Por fim, vale a pena também ressaltar que o método limpar(), definido na linha 60, tem como objetivo remover todos os itens selecionados do carrinho de compra. Isso é feito acionando-se o método clear() da interface Map, que vai eliminar todas as entradas existentes no mapeamento.

Atividade 02

1. Implemente o carrinho de compras da livraria virtual, conforme mostrado até este ponto da aula. Crie um programa para testar as funcionalidades do carrinho de compras. Execute o programa e verifique que o comportamento da implementação está correto.
2. Implemente um carrinho de compras para a locadora virtual. Crie também um programa para testar as funcionalidades do carrinho de compras. Execute o programa e verifique se o comportamento da implementação está correto.

Modelagem do armazenamento dos livros e do sistema

Para modelar o armazenamento dos livros disponíveis na livraria virtual, iremos utilizar uma abordagem em memória. Isso porque o foco do curso é ensinar as tecnologias web. Entretanto, é fortemente recomendado para os alunos que concluíram o módulo Banco de Dados, como atividade opcional, que usem banco de dados ao invés da abordagem em memória. Entretanto, essa implementação em banco de dados não pode interferir seu rendimento no curso, se possível, sendo realizada apenas ao final deste módulo.

Utilizaremos uma classe chamada Livraria para representar tanto as operações na livraria como o meio de armazenamento.

A primeira parte da implementação dessa classe é mostrada na Listagem 4.1 e é uma simplificação do que um sistema desenvolvido para o mercado pode possuir. A nossa implementação possui uma lista de objetos do tipo Livro (linha 18), representando o estoque existente na livraria virtual.

Como a nossa implementação é em memória, toda vez que o servidor é reiniciado, o estoque deve ser carregado na memória. Isso é feito pelo método popularLivros(), definido na linha 20. Esse método, executado logo quando o objeto Livraria é criado (linha 17), tem como objetivo criar alguns livros e inseri-los no estoque.

```
1  package livraria.negocio;
2
3  import java.util.ArrayList;
4  import java.util.Collection;
5  import java.util.Collections;
6  import java.util.Iterator;
7  import java.util.List;
8
9  import livraria.negocio.excecoes.CompraException;
10 import livraria.negocio.excecoes.LivroNaoEncontradoException;
11
12 public class Livraria {
13    private List<Livro> estoqueLivros;
14
15    public Livraria() {
16       estoqueLivros = new ArrayList<Livro>();
17       popularLivros();
18    }
19
20    private void popularLivros() {
21       Livro livro = new Livro();
22       livro.setIdLivro("0596005407");
23       livro.setAno(2008);
24       livro.setTitulo("Head First Servlets and JSP");
25       livro.setDescricao("Livro sobre Servlets e JSP.");
26       livro.setAutores("Bryan Basham, Kathy Sierra, Bert Bates");
27       livro.setQuantidade(10);
28       livro.setPreco(200.5);
29       estoqueLivros.add(livro);
30       livro = new Livro();
31       livro.setIdLivro("9788573935721");
32       livro.setAno(2007);
33       livro.setTitulo("Desenvolvendo Aplicações Web com JSP, Servlets, "
34       + "JavaServer Faces, Hibernate, EJB 3 Persistence e Ajax");
35       livro.setDescricao(
36       "Livro sobre tecnologias usadas na programação Java para Web.");
37       livro.setAutores("Edson Gonçalves");
38       livro.setQuantidade(10);
39       livro.setPreco(110.9);
40       estoqueLivros.add(livro);
41    }
42 }
```

Listagem 4.1 - Implementação da classe Livraria – Parte 1

Além disso, a classe Livraria deve possuir métodos de consulta aos livros disponíveis. Esses métodos são vistos na Listagem 4.2. O método getLivros() retorna todos os livros disponíveis.

Note que a implementação desse método faz uso da operação Collections.unmodifiableList().

Essa operação pega uma lista como parâmetro e retorna uma lista com os mesmos elementos, porém, seus métodos de modificação da lista (adicionar e remover) irão levantar uma exceção se executados. Isso é opcional, mas garante que apenas a lista original, guardada pela classe Livraria, poderá ser modificada. Essa garantia nos permite saber que apenas métodos da classe Livraria podem adicionar ou remover livros da lista, evitando que essas operações sejam executadas “sem querer” por outras partes do código que tiveram acesso a ela pelo método getLivros().

Vídeo 04 - Mapas Exemplo

Outro método de busca disponível é o getLivro(), o qual recebe um código e retorna o livro relacionado àquele código (linhas 48 a 50) ou levanta a exceção LivroNaoEncontradoException (linhas 53 a 56), caso não exista no estoque nenhum livro com o código passado. Essa exceção foi definida por nós e seu código é visto mais adiante, na Listagem 5.

Essa operação pega uma lista como parâmetro e retorna uma lista com os mesmos elementos, porém, seus métodos de modificação da lista (adicionar e remover) irão levantar uma exceção se executados. Isso é opcional, mas garante que apenas a lista original, guardada pela classe Livraria, poderá ser modificada. Essa garantia nos permite saber que apenas métodos da classe Livraria podem adicionar ou remover livros da lista, evitando que essas operações sejam executadas “sem querer” por outras partes do código que tiveram acesso a ela pelo método getLivros().

Outro método de busca disponível é o getLivro(), o qual recebe um código e retorna o livro relacionado àquele código (linhas 50 a 52) ou levanta a exceção LivroNaoEncontradoException (linhas 55 a 58), caso não exista no estoque nenhum livro com o código passado. Essa exceção foi definida por nós e seu código é visto mais adiante, na Listagem 5.

```
1  public List<Livro> getLivros() {
2      return Collections.unmodifiableList(estoqueLivros);
3  }
4
5  public Livro getLivro(String idLivro) throws LivroNaoEncontradoException {
6      Livro livroProcurado = null;
7      for (Livro book : estoqueLivros) {
8          if (book.getIdLivro().equals(idLivro)) {
9              livroProcurado = book;
10         }
11     }
12
13     if (livroProcurado == null) {
14         throw new LivroNaoEncontradoException(
15         "Não foi possível encontrar o livro: " + idLivro);
16     }
17
18     return livroProcurado;
19 }
```

Listagem 4.2 - Implementação da classe Livraria – Parte 2

Por fim, a classe Livraria possui métodos relacionados à compra de livros. O primeiro deles é o método comprarLivros(), que recebe como parâmetro um carrinho de compras contendo os livros selecionados pelo usuário. Esse método é apresentado na Listagem 4.3, linha 63. Basicamente, os produtos contidos no carrinho são analisados (linhas 67 a 73) um por um, e, para cada livro, o método comprarLivro() é executado. Esse método é mostrado na linha 76 e, dado o código de um livro e a quantidade a ser comprada, é responsável por:

- Verificar se o livro existe no estoque (linhas 79 a 83).
- Pegar a quantidade de livros em estoque (linha 83).

- Verificar se o estoque é suficiente para atender a quantidade de livros a ser comprada (linha 87), lançando uma exceção, caso o estoque seja insuficiente (linhas 91 e 92). Esse caso pode ocorrer quando vários usuários tentam comprar o mesmo livro ao mesmo tempo, numa quantidade de estoque satisfatória para cada cliente individualmente, mas superior ao estoque no total dos vários usuários.

- Debitar do estoque a quantidade vendida do livro (linha 88 e 89).

```
1  public void comprarLivros(CarrinhoCompras carrinho) throws CompraException {
2     Collection<ItemCompra> items = carrinho.getItens();
3     Iterator<ItemCompra> i = items.iterator();
4
5     while (i.hasNext()) {
6        ItemCompra item = (ItemCompra) i.next();
7        Livro livro = (Livro) item.getItem();
8        String id = livro.getIdLivro();
9        int quantity = item.getQuantidade();
10       comprarLivro(id, quantity);
11    }
12 }
13
14 public void comprarLivro(String idLivro, int qtdComprada) throws CompraException {
15       Livro livroSelecionado;
16       try {
17          livroSelecionado = getLivro(idLivro);
18       } catch (LivroNaoEncontradoException e) {
19          throw new CompraException(e.getMessage());
20       }
21
22       int qtdEstoque = livroSelecionado.getQuantidade();
23
24       if ((qtdEstoque - qtdComprada) >= 0) {
25          int novaQtd = qtdEstoque - qtdComprada;
26          livroSelecionado.setQuantidade(novaQtd);
27       } else {
28          throw new CompraException("Livro " + idLivro
29          + " sem estoque suficiente.");
30       }
31    }
32
33    public void fechar() {
34       // liberaria conexões de banco de dados, se usasse
35 }
```

Listagem 4.3 - Implementação da classe Livraria – Parte 3

As listagens 5 e 6 mostram o código das exceções criadas para representar possíveis erros durante a execução do sistema. O uso delas é feito nos métodos já mostrados da classe Livraria..

```
1  package livraria.negocio.excecoes;
2
3  public class LivroNaoEncontradoException extends Exception {
4
5     private static final long serialVersionUID = 1L;
6
7     public LivroNaoEncontradoException() {
8     }
9
10    public LivroNaoEncontradoException(String msg) {
11       super(msg);
12    }
13
14 }
```

Listagem 5 - Código da exceção criada para o erro de livro não encontrado

```
1  package livraria.negocio.excecoes;
2
3  public class CompraException extends Exception {
4
5     private static final long serialVersionUID = 1L;
6
7     public CompraException() {
8     }
9
10    public CompraException(String msg) {
11       super(msg);
12    }
13
14 }
```

Listagem 6 - Código da exceção criada para representar um erro na compra do livro

Atividade 03

1. Implemente a classe Livraria, conforme apresentado, bem como as exceções LivroNaoEncontradoException e CompraException. Crie um programa para testar as funcionalidades da classe Livraria. Execute o programa e verifique se o comportamento da implementação está correto.

2. Adicione mais cinco livros distintos sobre programação no método popularLivros() da classe Livraria.

3. Crie uma classe chamada Locadora para implementar as operações de locar filmes e para armazenar os filmes disponíveis, de forma semelhante à classe Livraria. Crie um programa para testar as funcionalidades da classe criada. Execute o programa e verifique que o comportamento da implementação está correto.

Conclusão

Nesta aula, paramos por aqui! Na próxima aula continuaremos com a implementação da Livraria virtual, focando na parte web da aplicação. Até lá!

Leitura Complementar

Para complementar seu entendimento sobre a programação em Java, analise os métodos disponibilizados pelas classes envolvidas no desenvolvimento da Livraria virtual. Isso pode ser feito observando-se a API do Java:

JavaTM Platform Standard Edition, v 6.0 – API Specifications. Disponível em: <http://download.oracle.com/javase/6/docs/api/>. Acesso em: 28 ago. 2012.

Em especial, observe as classes Map, HashMap e Collections, encontradas no pacote java.util.

Além disso, leia sobre programação genérica em Java (Java Generics):

UM PASSEIO pelo Java 5: generics: parte 1. Disponível em: <http://www.devmedia.com.br/articles/viewcomp.asp?comp=3001>. Acesso em: 28 ago. 2012.

UM PASSEIO pelo Java 5: generics: parte 2. Disponível em: <http://www.devmedia.com.br/articles/viewcomp.asp?comp=3061>. Acesso em: 28 ago. 2012.

JavaTM Platform Standard Edition, v 6.0: generics. Disponível em: <http://download.oracle.com/javase/1.5.0/docs/guide/language/generics.html>. Acesso em: 28 ago. 2012.

Resumo

Nesta aula, você praticou a modelagem das classes de negócio de uma aplicação web: a Livraria Virtual. Você criou, observou e praticou a criação de classes que representam entidades (coisas) do negócio, ou seja, de uma livraria virtual: livros, carrinho de compras, a livraria em si e os erros que podem acontecer nas operações do negócio (exceções). Na próxima aula, continuaremos com o desenvolvimento das camadas necessárias para que a aplicação rode no ambiente da web. Até lá!

Autoavaliação

Descreva os passos que você seguiu para realizar a modelagem de negócio das aplicações Livraria virtual e Locadora virtual.

Referências

AHMED, K. Z.; UMRYSH, C. E. **Desenvolvendo aplicações comerciais em Java com Java J2EE e UML**. Rio de Janeiro: Ciência Moderna, 2003. 324 p.

BASHAM, Bryan; SIERRA, Kathy; BATES, Bert. **Head First Servlets and JSP**: passing the Sun Certified Web Component Developer Exam (SCWCD). 2. ed.: O'Reilly Media, 2008.

CATTELL, Rick; INSCORE, Jim. **J2EE:** criando aplicações comerciais. Rio de Janeiro: Campus, 2001.

HALL, Marty. **More Servlets and JavaServer Pages (JSP)**. New Jersey: Prentice Hall PTR (InformIT), Sun Microsystems Core Series, 2001. 752 p. Disponível em: <http://pdf.moreservlets.com/>. Acesso em: 17 ago. 2012.

HALL, Marty; BROWN, Larry. **Core Servlets and JavaServer Pages (JSP)**. 2. ed. New Jersey: Prentice Hall PTR (InformIT), Sun Microsystems Core Series, 2003. 736 p. (Core Technologies, 1). Disponível em: <http://pdf.coreservlets.com/>. Acesso em: 28 ago. 2012.

HYPERTEXT **Transfer Protocol** -- HTTP/1.1: methods definition. Disponível em: <http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html>. Acesso em: 28 ago. 2012.

J2EE Tutorial. **Java EE Reference at a Glance.** Disponível em: . Acesso em: 17 ago. 2012.