

# Workshop 01 - Livraria Web

O objetivo deste workshop é consolidar o conhecimento adquirido durante o curso. Será desenvolvido uma Livraria virtual, onde , nesse primeiro workshop, é possível consultar livros, listá-los e ver seus detalhes como autor, preço e descrição.

## Exercícios

**Exercício 1:** Criando a classe Livro

**Exercício 2:** Criando o Banco de dados

**Exercício 3:** Criando a classe LivroDao

**Exercício 4:** Criando o Bean de pesquisa

**Exercício 5:** Desenvolvendo a página inicial

**Exercício 6:** Desenvolvendo a página de resultados

**Exercício 7:** Desenvolvendo a página Livro

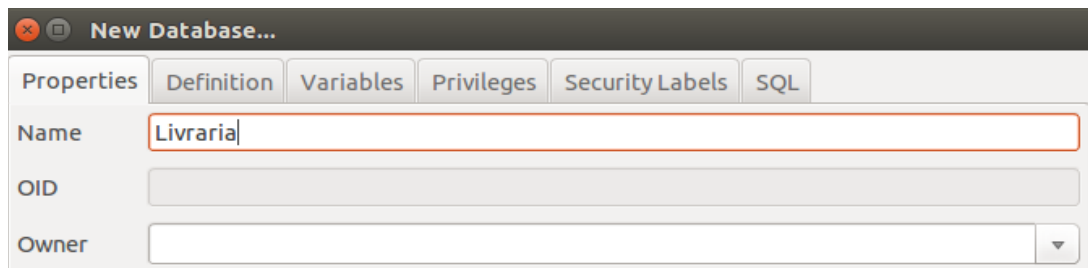
## Exercício 1 - Criando a classe Livro

1. Vamos utilizar o projeto **Livraria-web** criado no laboratório de Maven. Caso o projeto tenha sido usado, recomenda-se criar um novo, do zero, para não bagunçar a estrutura do seu projeto.
2. A classe **Livro** representa a entidade livro em nosso projeto. Obedecerá o padrão de projeto VO (Value Object) e conterá os dados de um livro. Crie a classe **Livro** (dentro do pacote **model**) assim como apresentado no UML abaixo. Não se esqueça de acrescentar os **getters** e **setters**.



## Exercício 2 - Criando o Banco de dados

1. Para o projeto funcionar, precisaremos de um banco de dados para guardar informações, como os livros que temos em estoque, os clientes cadastrados no site e os pedidos de cada cliente. Primeiro, crie um novo Banco de dados no **Postgres**. Conecte ao servidor usado, clique com o botão direito em **Databases** e em seguida, clique em **New Database...** Vamos chamar nosso novo banco de dados de Livraria, para deixar explícito o banco que estamos usando. Feito isso, clique **Ok**.



2. Agora, precisamos criar a conexão entre o banco de dados e o seu projeto. Crie a classe **FabricaConexao** assim como passado no laboratório de Banco de dados, e nessa classe, altere somente a url de destino da conexão.

```
static final String url = "jdbc:postgresql://localhost:5432/Livraria";
```

3. Crie também a classe **TestaConexao** para fazer o teste, ver se está tudo em ordem.
4. Dentro do seu Banco de dados Livraria, vamos agora criar tabelas onde vão ser inseridas e recolhidas as informações necessárias.

Primeiro, criaremos nosso **ESTOQUE** de livros.

```
CREATE TABLE estoque (
    COD_LIVRO SERIAL NOT NULL,
    TITULO VARCHAR(30) NOT NULL,
    AUTOR VARCHAR(20) NOT NULL,
    PRECO NUMERIC NOT NULL,
    IMAGEM VARCHAR(80) NOT NULL,
    DESCRICAO VARCHAR(80),
    PRIMARY KEY (COD_LIVRO));
```

```
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('GRANDE SERTAO - VEREDAS', 'ROSA, JOAO GUIMARAES', 165, 'imagens/veredas.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('QUANDO NIETZSCHE CHOROU', 'YALOM, IRVIN D.', 49.9, 'imagens/chorou.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('CASSINO ROYALE - JAMES BOND 00', 'Fleming, Ian', 29.9, 'imagens/james.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('FILOSOFIA DO TEDIO', 'Svendsen, Lars', 29.9, 'imagens/tedio.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('O CASAMENTO', 'Rodrigues, Nelson', 39.9, 'imagens/casamento.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('NEVE', 'PAMUK, ORHAN', 54, 'imagens/neve.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('VOLTA AO MUNDO EM OITENTA DIAS', 'VERNE, JULIO', 16.5, 'imagens/volta_mundo.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('CRISTOVAO COLOMBO', 'VERNE, JULIO', 16.5, 'imagens/cristovao_colombo.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('VINTE MIL LEGUAS SUBMARINAS', 'VERNE, JULIO', 14.9, 'imagens/submarinas.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('O SENHOR DOS ANEIS',
```

```
'TOLKIEN, J.R.R.', 169.9, 'imagens/senhor.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('HARRY POTTER',
'ROWLING, J.K.', 89.7, 'imagens/harry.png');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('A AVENTURAS DE PI',
'MARTEL, YANN', 23.5, 'imagens/lifeofpi.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('PARA ONDE ELA FOI?',
'FORMAN, GAYLE', 20.0, 'imagens/onde.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('O LIVRO DO CEMITERIO',
'GAILMAN, NEIL', 20.0, 'imagens/ceimiterio.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES (SANDMAN VOL 1',
'GAILMAN, NEIL', 489.0, 'imagens/sandman.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('WATCHMEN', 'MOORE,
ALAN', 37.4, 'imagens/watchmen.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('JUSTICEIRO NOIR', 'TIIER,
FRANK', 12.5, 'imagens/justiceiro.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('SUPERMAN', 'TOMASI,
PETER', 5.9, 'imagens/superman.jpg');
INSERT INTO ESTOQUE (TITULO,AUTOR,PRECO,IMAGEM) VALUES ('BATMAN', 'SNYDER,
SCOTT', 5.9, 'imagens/batman.jpeg');
```

Nesse código sql, estamos criando uma tabela **estoque** e nela estamos adicionando os livros, seus títulos, autores, preço, caminho de imagem e futuramente, uma descrição para cada livro.

### Exercício 3 - Criando a classe LivroDao

1. Primeiro iremos definir a interface GenericDao afim de mantermos um padrão para todas as classes DAO que sejam construídas em nosso projeto.

```
package dao;
```

```
import java.io.Serializable;
import java.util.Collection;
```

```
public interface Dao<E, K extends Serializable> {
```

```
    /**
     * Busca a Serializable pelo seu identificador.
     *
     * @param identificador da Serializable
     *
     * @return Serializable pesquisada
     */
```

```
    E consultar(final K id);
```

```
    /**
     * Altera a Serializable.
     *
     * @param Serializable
     */
```

```
    void alterar(final E Serializable);
```

```
/**
 * Inserir a Serializable.
 *
 * @param Serializable
 */
void salvar(final E Serializable);

/**
 * Remove a Serializable.
 *
 * @param Serializable
 */
void remover(final E Serializable);

/**
 * Lista todos os objetos da Serializable.
 *
 * @return Collection<E>
 */
Collection<E> listar();
}
```

2. Agora iremos criar a classe **LivroDao**. Essa classe será responsável pelo acesso ao banco de dados e retornar dados referentes a classe **Livro**. Crie a classe LivroDao (dentro do pacote **dao**) para acessar o banco de dados e fazer as operações de consulta, um livro e uma lista de livros. Como exemplo segue uma parte do código abaixo.

```
package dao;

import java.sql.*;
import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
import java.util.logging.Logger;

import service.FabricaConexao;
import service.Livro;

public class LivroDao implements Dao<Livro,Integer> {
    Logger LOG = Logger.getGlobal();

    private static final String OBTER_POR_ID_SQL = "SELECT AUTOR, TITULO, COD_LIVRO, IMAGEM,"
        + " PRECO, DESCRICAO FROM ESTOQUE WHERE COD_LIVRO = ?";

    private static final String CONSULTAR_SQL = "SELECT COD_LIVRO, TITULO, AUTOR, PRECO,"
        + " IMAGEM, DESCRICAO FROM ESTOQUE WHERE TITULO LIKE ?";
```

```
@Override
public Livro consultar(Integer codigo) {
    Livro livro = null;
    try (Connection conexao = FabricaConexao.getConexao();
        PreparedStatement consulta =
conexao.prepareStatement(OBTENHA_POR_ID_SQL);) {

        consulta.setInt(1, codigo);

        ResultSet resultado = consulta.executeQuery();

        if (resultado.next()) {
            livro = new Livro();
            livro.setAutor(resultado.getString("AUTOR"));
            livro.setCodigo(resultado.getInt("COD_LIVRO"));
            livro.setImagem(resultado.getString("IMAGEM"));
            livro.setPreco(resultado.getDouble("PRECO"));
            livro.setTitulo(resultado.getString("TITULO"));
            livro.setDescricao(resultado.getString("DESCRICAO"));
        }

        resultado.close();
    } catch (SQLException e) {
        LOG.severe(e.toString());
    }
    return livro;
}

public List<Livro> consultar(String titulo) {
    ArrayList<Livro> lista = new ArrayList<Livro>();
    try (Connection conexao = FabricaConexao.getConexao();
        PreparedStatement consulta =
conexao.prepareStatement(CONSULTAR_SQL);) {

        consulta.setString(1, "%" + titulo.toUpperCase() + "%");

        ResultSet resultado = consulta.executeQuery();

        while (resultado.next()) {
            Livro livro = new Livro();
            livro.setAutor(resultado.getString("AUTOR"));
            livro.setCodigo(resultado.getInt("COD_LIVRO"));
            livro.setImagem(resultado.getString("IMAGEM"));
            livro.setPreco(resultado.getDouble("PRECO"));
            livro.setTitulo(resultado.getString("TITULO"));
            livro.setDescricao(resultado.getString("DESCRICAO"));
            lista.add(livro);
        }
    }
```

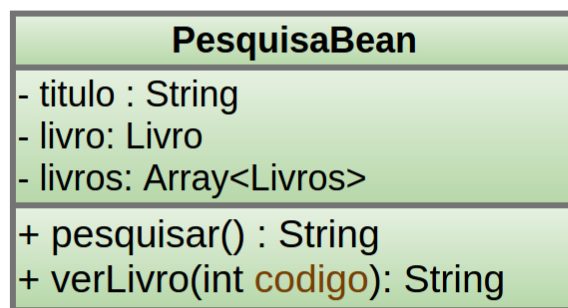
```
        resultado.close();

    } catch (SQLException e) {
        LOG.severe(e.toString());
    }
    return lista;
}
//Faça o restante do CRUD
}
```

Você deve terminar de implementar o restantes do métodos definidos na Interface Dao generica.

## Exercício 4 - Criando o Bean de pesquisa

1. Nesse passo, você irá **criar um managed bean chamado PesquisaBean** e nele faremos um método **pesquisar()**, em que iremos receber uma string contendo a propriedade **titulo** através do formulário e consultar em nossa tabela **estoque** através da classe **LivroDao**. Segue o exemplo abaixo:



```
private String titulo;

private Livro livro;
private List<Livro> livros = new ArrayList<Livro>();

public String pesquisar() {

    if (titulo == null) {
        titulo = "";
    }
    System.out.println("Pesquisa: " + titulo);

    LivroDao dao = new LivroDao();
    livros = dao.consultar(titulo);

    if (livros.size() <= 0) {
        System.out.println("Livro não foi localizado");
    }

    return "Resultado";
}
```

```
public String verLivro(Integer codigo) {  
  
    LivroDao dao = new LivroDao();  
    livro = dao.consultar(codigo);  
  
    return "Livro";  
}
```

**OBS:** consultar(titulo) e consultar(codigo) são métodos do LivroDao, mas que buscam atributos diferentes (um busca um array de livros e outro busca somente um livro).

**OBS2:** “Resultado” e “Livro” nesse caso são o nome das páginas XHTML relacionadas. No caso, estamos fazendo uma navegação dinâmica implícita. Nos próximos passos iremos criar cada uma dessas páginas.

## Exercício 5 - Desenvolvendo a página Inicial

1. Refaça os templates criados no **Laboratório de Templates**, **LayoutPadrao.xhtml**, **CabecalhoBasico.xhtml**, **ConteudoBasico.xhtml** e **Rodapebasico.xhtml**, assim como foi passado no laboratório.
2. Crie na pasta **webapp** o arquivo **Inicio.xhtml**. Essa será a página inicial da sua livraria virtual. Nela, vamos criar um menu de departamentos e um formulário de busca, onde iremos consultar os livros em estoque. Segue exemplo do código:

```
<h:body>  
<ui:composition template="/WEB-INF/template/layout.xhtml">  
    <ui:define name="content">  
        <div class="container">  
            <div class="destaque">  
                <section class="busca">  
                    <h2>Busca</h2>  
                    <h:form>  
                        <p:inputText id="search"  
                            value="#{pesquisaBean.titulo}" size="14"/>  
                        <p:commandButton icon="ui-icon-search"  
                            action="#{pesquisaBean.pesquisar}" />  
                    </h:form>  
                </section>  
  
                <section class="menu-departamentos">  
                    <h2>Departamentos</h2>  
                    <nav>  
                        <ul>  
                            <li><p:commandLink value="Livros"/>  
                                <ul>  
                                    <li><a href="#">Auto-ajuda</a></li>  
                                    <li><a href="#">Bibliografias</a></li>  
                                    <li><a href="#">Teen</a></li>  
                                    <li><a href="#">Romance</a></li>  
                                    <li><a href="#">Ficcao</a></li>  
                                    <li><a href="#">Gibis e HQs</a></li>  
                                </ul>  
                            <li><a href="#">Filmes</a></li>  
                            <li><a href="#">Games</a></li>  
                            <li><a href="#">Musica</a></li>  
                        </ul>  
                    </nav>  
                </section>  
            </div>  
        </div>  
</ui:define>  
</ui:composition>
```

```
        <li><a href="#">Acessorios</a></li>
    </ul>
</nav>
</section>
</div>
</div>
</ui:define>
</ui:composition>
</h:body>
```

**CSS:**

```
.busca,
.menu-departamentos{
    background-color: #dcdcdc;
    font-weight: bold;
    text-transform: uppercase;
    margin-right: 10px;
    width: 230px;
}

.busca h2,
.busca form,
.menu-departamentos h2{
    font-size:15px;
    font-weight:bold;
    margin: 10px;
}

.menu-departamentos li{
    background-color: white;
    margin-bottom: 1px;
    padding: 5px 10px;
}

.menu-departamentos a{
    color: #333;
    text-decoration: none;
}

.busca input{
    vertical-align: middle;
}

.busca input[type=search]{
    width: 170px;
}

.busca,
.menu-departamentos{
    float:left;
}

.menu-departamentos li ul{
    display: none;
}

.menu-departamentos li:HOVER ul{
    display: block;
}
```



```
.menu-departamentos ul ul li{
    background-color: #dcdcdc;
}

.menu-departamentos li li a:BEFORE{
    content: '\261E ';
    padding-right: 3px;
}

.menu-departamentos{
    clear: left;
    margin-top: 2px;
    padding-bottom: 10px;
}
```

Pronto, temos nossa página inicial, porem ela ainda não está funcional, precisamos da página Resultado.xhtml para onde o PesquisaBean vai redirecionar o resultado da consulta.

## Exercício 6 - Desenvolvendo a página de Resultados

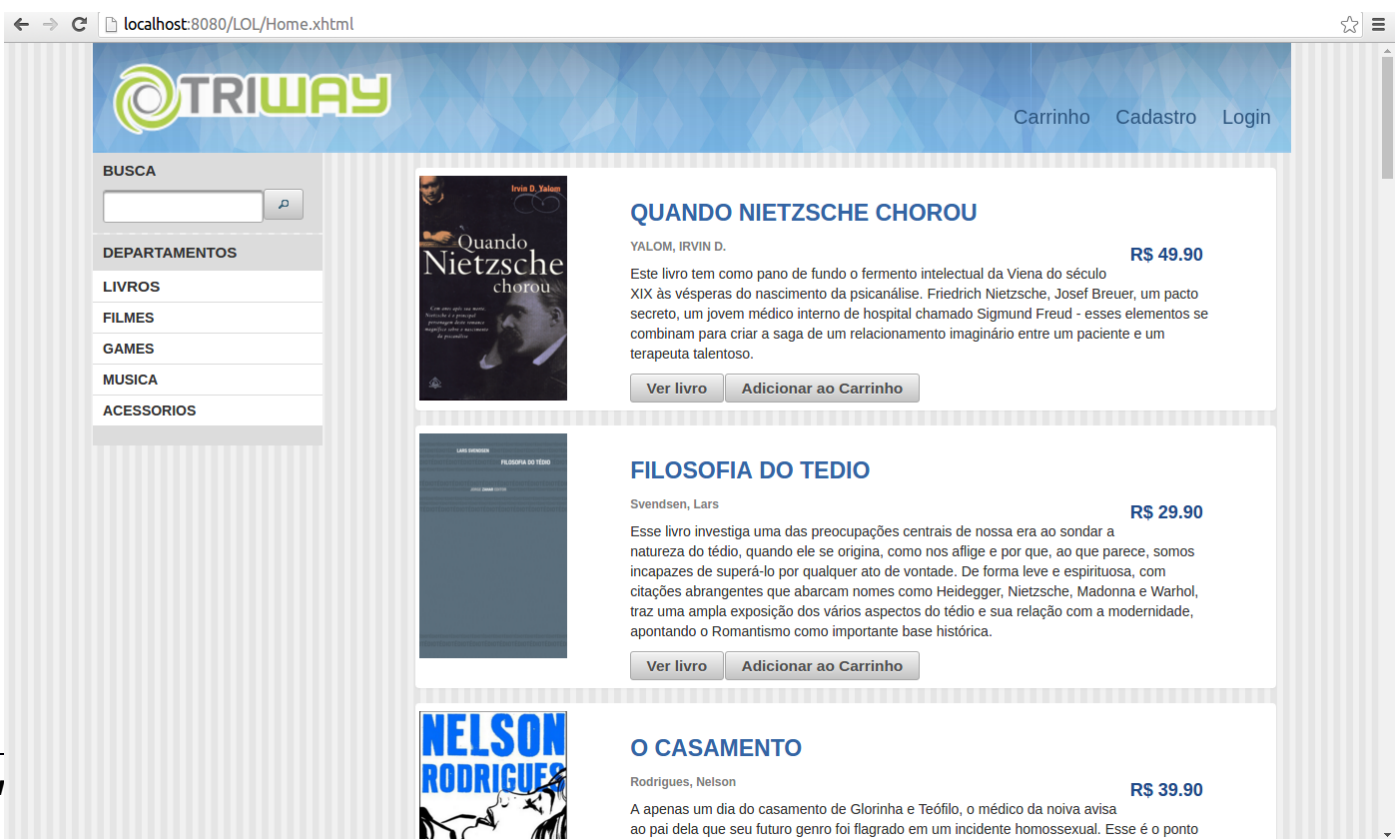
1. Na mesma pasta webapp, crie agora o arquivo **Resultado.xhtml** e usando os templates, faça com que mostre uma lista com todos os livros encontrados na nossa busca da página inicial. A section de **busca** e **departamentos** não é um template, mas pode ser usado como um. Faça com que esses elementos apareçam no **Resultado.xhtml**.

```
<div class="container">
<div class="row">
    <div class="col-md-3">
        BUSCA E DEPARTAMENTOS
    </div>
    <div class="col-md-9">
        <ui:repeat value="#{pesquisaBean.livros}" var="livro">
            <div class="panel">
                <div class="row">
                    <div class="col-md-3">
                        
                    </div>
                    <div class="col-md-8 descr-livro">
                        <h3 class="titulo">#{livro.titulo}</h3>
                        <h4 class="pull-right preco">R$ #{livro.preco}</h4>
                        <h4 class="autor">#{livro.autor}</h4>
                        <p class="descricao">#{livro.descricao}</p>
                        <h:form>
                            <p:inputText type="hidden" id="codigo"
                                value="#{livro.codigo}"/>
                            <p:commandButton value="Ver livro"
                                action="#{pesquisaBean.verLivro(livro.codigo)}"/>
                        </h:form>
                    </div>
                </div>
            </div>
        </ui:repeat>
    </div>
</div>
</div>
```

## CSS:

```
.descricao{
    margin: 10px 0;
    font-weight:100;
}
.preco{
    color: #204a87;
    font-weight: bold;
    padding-right: 10px;
}
.autor{
    font-weight: bold;
    color: gray;
}
.titulo{
    font-weight: bold;
    font-size: 23px;

    padding: 5px 0;
    margin-bottom: 5px;
    color: #3465a4;
}
#img-livro{
    width: 150px;
    margin: 2px;
    padding: 5px 1px;
    float: left;
}
.panel-body{
    margin:5px;
}
.panel{
    margin: 15px;
    position:relative;
}
```



localhost:8080/LOL/Home.xhtml

**TRIWAY** Carrinho Cadastro Login

**BUSCA**

**DEPARTAMENTOS**

- LIVROS
- FILMES
- GAMES
- MUSICA
- ACESSORIOS

**QUANDO NIETZSCHE CHOROU**

YALOM, IRVIN D. **R\$ 49.90**

Este livro tem como pano de fundo o fermento intelectual da Viena do século XIX às vésperas do nascimento da psicanálise. Friedrich Nietzsche, Josef Breuer, um pacto secreto, um jovem médico interno de hospital chamado Sigmund Freud - esses elementos se combinam para criar a saga de um relacionamento imaginário entre um paciente e um terapeuta talentoso.

**FILOSOFIA DO TEDIO**

Svendsen, Lars **R\$ 29.90**

Esse livro investiga uma das preocupações centrais de nossa era ao sondar a natureza do tédio, quando ele se origina, como nos aflije e por que, ao que parece, somos incapazes de superá-lo por qualquer ato de vontade. De forma leve e espirituosa, com citações abrangentes que abarcam nomes como Heidegger, Nietzsche, Madonna e Warhol, traz uma ampla exposição dos vários aspectos do tédio e sua relação com a modernidade, apontando o Romantismo como importante base histórica.

**NELSON RODRIGUES**

Rodrigues, Nelson **R\$ 39.90**

A apenas um dia do casamento de Glorinha e Teófilo, o médico da noiva avisa ao pai dela que seu futuro genro foi flagrado em um incidente homossexual. Esse é o ponto

## Exercício 7 - Desenvolvendo a página de Livros

1. Crie agora o arquivo **Livro.xhtml** e faça com que ele mostre para nós com detalhes o livro escolhido pelo usuário, na página de resultados. Use os templates para completar o design da página.

```
<div class="container">
<div class="row">
    <div class="col-md-3">
        BUSCA E DEPARTAMENTOS
    </div>
<div class="col-md-9">
    <div class="row">
        <div class="panel panel-default">
            <div class="panel-heading">
                <h2 class="panel-title">#{pesquisaBean.livro.titulo}</h2>
            </div>
            <div class="panel-body">
                <div class="col-md-3">
                    
                </div>
                <div class="col-md-9">
                    <dl>
                        <dt class="titulo">Nome do Produto</dt>
                        <dd class="descricao">#{pesquisaBean.livro.titulo}</dd>
                        <dt class="titulo">Autor</dt>
                        <dd class="descricao">#{pesquisaBean.livro.autor}</dd>
                        <dt class="titulo">Preço:</dt>
                        <dd class="descricao">R$ #{pesquisaBean.livro.preco}</dd>
                        <dt class="titulo">SIPNOSE</dt>
                        <dd class="descricao">#{pesquisaBean.livro.descricao}</dd>
                    </dl>
                </div>
            </div>
            <h:form>
                <p:inputText type="hidden" id="codigo"
                    value="#{pesquisaBean.livro.codigo}" />
                <p:commandButton value="Adicionar ao Carrinho"/>
            </h:form>
        </div>
    </div>
</div>
```

Página criada, teste novamente o seu formulário de busca porem, ao chegar na página de resultados, clique em **Ver Livro** e será direcionado para a página Livro.xhtml, que deverá se parecer com o exemplo abaixo. Adicionamos o botão **Adicionar ao Carrinho** mas sem ação, pois será usada futuramente, no workshop de Frameworks.

