MÓDULO VIII

# CURSO TÉCNICO DE INFORMÁTICA

**Tielle da Silva Alexandre**

Ministério da Educação

GOVERNO FEDERAL
BRASIL
PAÍS RICO É PAÍS SEM POBREZA

Tielle da S. Alexandre
Módulo: VIII

# Projeto Final II

Disciplina do Eixo de Disciplinas do Currículo do
Curso Técnico de Informática CEFET/RJ UnED NI

Edição: CEFET/RJ UnED NI – COORDENAÇÃO DE INFORMÁTICA

Local: Estrada de Adrianópolis, 1317 – Santa Rita, Nova Iguaçu - RJ

Editora: CEFET/RJ

Ano de Publicação: 2016

CEFET/RJ

rede e-Tec Brasil

**Presidente da República**
Dilma Rousseff

**Ministro da Educação**
Cid Ferreira Gomes

**Secretário de Educação Profissional e Tecnológica**
Eliezer Moreira Pacheco

**Professora – organizadora**
André Alexandre Guimarães Couto

**Diretor Geral do CEFET/RJ**
Carlos Henrique Figueiredo Alves

**Diretora de Ensino**
Gisele Maria Ribeiro Vieira

**Coordenadora da Educação à Distância no CEFET/RJ**
Maria Esther Provenzano

**Coordenador Geral do e-Tec no CEFET/RJ**
Mauro Godinho Gonçalves

**Coordenador Geral Adjunto do e-Tec no CEFET/RJ**
Alexandre Martinez dos Santos

**Coordenadora do Curso de Informática e-Tec no CEFET/RJ**
Rosana Soares Gomes Costa

**Coordenador de Polo Nova Iguaçu do e-Tec no CEFET/RJ**
Francisco Eduardo Cirto

**Coordenador de Tutoria do e-Tec no CEFET/RJ**
Unapetinga Hélio Bomfim Vieira

**Professora Pesquisadora do e-Tec no CEFET/RJ**
Lucia Helena Dias Mendes

**Design Instrucional**
Luciana Ponce Leon Montenegro de Morais Castro

Ficha catalográfica elaborada pela Biblioteca Central do CEFET/RJ

**Apresentação do e-Tec Brasil**

Prezado estudante,

Bem vindo ao e-Tec Brasil!

Você faz parte de uma rede nacional pública de ensino, a Escola Técnica Aberta do Brasil, instituída pelo Decreto n° 6.301, de 12 de dezembro de 2007, com o objetivo de democratizar o acesso ao ensino técnico público, na modalidade a distância. O programa é resultado de uma parceria entre o Ministério da Educação, por meio das Secretarias de Educação Profissional e Tecnológica (SETEC), as universidades e escolas técnicas estaduais e federais.

A educação à distância no nosso país, de dimensões continentais e grande diversidade regional e cultural, longe de distanciar, aproxima as pessoas ao garantir acesso à educação de qualidade, e promover o fortalecimento da formação de jovens moradores de regiões distantes, geograficamente ou economicamente, dos grandes centros.

O e-Tec Brasil leva os cursos técnicos a locais distantes das instituições de ensino e para a periferia das grandes cidades, incentivando os jovens a concluir o ensino médio. Os cursos são ofertados pelas instituições públicas de ensino e o atendimento ao estudante é realizado em escolas-polo integrantes das redes públicas municipais e estaduais.

O Ministério da Educação, as instituições públicas de ensino técnico, seus servidores técnicos e professores acreditam que uma educação profissional qualificada – integradora do ensino médio e educação técnica, - é capaz de promover o cidadão com capacidades para produzir, mas também com autonomia diante das diferentes dimensões da realidade cultural, social, familiar, esportiva, política e ética.

Nós acreditamos em você!

Desejamos sucesso na sua formação profissional!

Ministério da Educação

Janeiro de 2010

Nosso contato

etecbrasil@mes.gov.br

**Indicação de ícones**


**Curiosidades:** indica informações interessantes que enriquecem o assunto.

**Interrogação:** indica perguntas frequentes do aluno em relação ao tema e respostas às mesmas.


**Você sabia? :** oferece novas informações e notícias recentes relacionadas ao tema estudado.


**Lembrete:** enfatiza algum ponto importante sobre o assunto.


**Tome nota 1:** espaço dedicado às anotações do aluno.

**Tome nota 2:** espaço também dedicado às anotações do aluno.

**Mãos a obra:** apresenta atividades em diferentes níveis de aprendizagem para que o estudante possa realizá-las e conferir o seu domínio do tema estudado.

**Bibliografia:** apresenta a bibliografia da apostila.

# SUMÁRIO


Palavra do Professor – organizador 08

Apresentação da Disciplina 09

Projeto Institucional 10

Aula 1 – Criando o projeto de uma aplicação Web 11

Aula 2 – Criando a camada *view* de uma aplicação *Web* 23

Aula 3 – Criando uma *servlet* para o cadastro de um produto 32

Aula 4 – Programando uma s*ervlet* – Método Cadastrar 42

Aula 5 – Programando uma s*ervlet* – Método Consultar 49

Aula 6 – Programando uma s*ervlet* – Método Excluir 57

Aula 7 – Programando uma s*ervlet* – Método Alterar 61

Referências Bibliográficas 66

**COM A PALAVRA, O PROFESSOR...**

Caros (as) alunos (as):

A disciplina de Projeto Final II tem por objetivo o desenvolvimento de uma aplicação *web* segundo o padrão *Model-View-Controller* (MVC). Os conhecimentos adquiridos pelos alunos em cada aula serão aplicados em atividades práticas de programação. Ao final dessa disciplina, o aluno terá desenvolvido um pequeno *software web.* Além disso, essa disciplina exige a aplicação de conhecimentos adquiridos ao longo do curso, possibilitando assim, a fixação do conteúdo.

O projeto *web* será desenvolvido utilizando a plataforma Java *Enterprise Edition* (EE) e o servidor de banco de dados MySQL. De forma incremental, o aluno irá desenvolver um *software web* contendo métodos para cadastrar, consultar, alterar e excluir (CRUD) registros do banco de dados segundo o padrão de desenvolvimento MVC.

Para que o aluno consiga acompanhar o desenvolvimento da disciplina de Projeto Final II, o aluno deve obedecer ao cronograma de estudo proposto, bem como fazer e entregar todas as atividades práticas para a correção. Portanto, a organização e o comprometimento são essenciais nesta empreitada.

O organizador.

**Apresentação da Disciplina**

**Módulo VIII – Projeto Final II**

**Carga Horária: 30 Horas**

Espera-se que o (a) cursista desenvolva as seguintes competências:

- Entender a arquitetura cliente-servidor e demais conceitos pertinentes ao desenvolvimento *web* de aplicações;
- Desenvolver uma aplicação *web* utilizando a plataforma de desenvolvimento Java *Enterprise Edition* (EE);
- Desenvolver interfaces gráficas *web;*
- Utilizar páginas Java *Server Pages* (JSP);
- Utilizar Servlets para realizar operações de inserção, atualização, exclusão e de consulta;
- Aplicar o padrão MVC no desenvolvimento de uma aplicação *web*.

**Projeto instrucional**

**Disciplina: Projeto Final II (30 horas)**

**Ementa: Construção de uma aplicação *web* usando a plataforma desenvolvimento Java EE. Desenvolvimento de interfaces gráficas *web*. Utilização do padrão MVC. Utilização de *Servlet* e JSP.**

| AULA | OBJETIVOS DE APRENDIZAGEM | MATERIAIS | CARGA HORÁRIA 30 (horas) |
|---|---|---|---|
| **1 - Criando o Projeto de uma Aplicação Web** | **Entender conceitos relacionados ao desenvolvimento de uma aplicação *web;* Instalar o ambiente de desenvolvimento para aplicações Web; Criar um projeto web no NetBeans.** | **impresso** | **4** |
| **2 - Criando a camada *view* de uma aplicação *Web*** | **Criar páginas em JSP e HTML usando o NetBeans e aplicar a reutilização de código através da diretiva *include.*** | **impresso** | **4** |
| **3 - Criando uma *servlet* para o cadastro de um produto** | **Criar uma *servlet;* Conhecer o mapeamento de uma *servlet* e detalhes de seu funcionamento e criar classes do pacote *model* e *dao*.** | **impresso** | **5** |
| **4 – Programando uma *servlet* – Método Cadastrar** | **Programar uma *servlet* que realize algumas ações e chame um método para cadastrar um produto no banco de dados; Enviar uma mensagem da *servlet* para uma página JSP.** | **impresso** | **4** |
| **5 – Programando uma *servlet* – Método Consultar** | **Construção de uma tela para consulta de produto; Programar uma *servlet* que chame um método para consultar um produto no banco de dados; Implementar o método de consulta na classe *ProdutoDAO*. Enviar o objeto recuperado para ser exibido em uma página JSP.** | **impresso** | **4** |
| **6 – Programando uma *servlet* – Método Excluir** | **Programar uma *servlet* que chame um método para excluir um produto no banco de dados e implementar o método de exclusão na classe *ProdutoDAO*.** | **impresso** | **4** |
| **7 – Programando uma *servlet* – Método Alterar** | **Programar uma *servlet* que chame um método para alterar um produto no banco de dados e implementar o método de alteração na classe *ProdutoDAO*.** | **impresso** | **4** |

Olá, caro estudante!

# Saudações Cefetianas!

Antes de iniciarmos nosso estudo sobre Tecnologia, reflita sobre essa ideia:

**"Tudo aquilo que não enfrentamos em vida acaba se tornando o nosso destino."**

Carl Jung

Bom estudo!


# Criando o Projeto de uma Aplicação *Web*

## Aula 01: Criando o Projeto de uma Aplicação Web

O Projeto Final II é disciplina que visa demonstrar o desenvolvimento *web* de uma aplicação segundo o padrão *Model-View-Controller* (MVC). A plataforma de desenvolvimento Java *Enterprise Edition* (EE) será utilizada para a programação de páginas *web* dinâmicas e o NetBeans será utilizado como ambiente de desenvolvimento.

As atividades dessa disciplina possibilitarão ao aluno a construção de uma pequena aplicação *web* possibilitando assim, a aplicação prática do conteúdo assimilado em cada aula. Em cada atividade prática, uma etapa para o desenvolvendo da aplicação será implementada. Para que consiga acompanhar o desenvolvimento da disciplina de Projeto Final II, o aluno deve obedecer ao cronograma de estudo proposto, bem como fazer e entregar todas as atividades práticas para a correção.

Nesta aula, será apresentado alguns conceitos pertinentes ao desenvolvimento *web* de aplicações, bem como será demonstrado a instalação e a configuração do ambiente de desenvolvimento no NetBeans. Ao final dessa aula, o aluno irá criar uma página padrão *web* inicial.

Através do navegador, o cliente solicita um conteúdo ao servidor. Por sua vez, o servidor recebe a solicitação do usuário, localiza o conteúdo requisitado e o envia ao usuário como resposta da solicitação (Basham, Bates e Sierra, 2008). Esse conteúdo pode ser uma página *HyperText Markup Language* (HTML), um arquivo em formato *Portable Document Format* (PDF), uma imagem entre outros tipos de conteúdos.

Na maioria das vezes, a comunicação entre o cliente e o servidor ocorre através do protocolo do Protocolo de Transferência de Hipertexto (HTTP). A estrutura do HTTP que norteia essa comunicação é composta por sequências simples de solicitações e respostas, conforme ilustrado

na Figura 1.

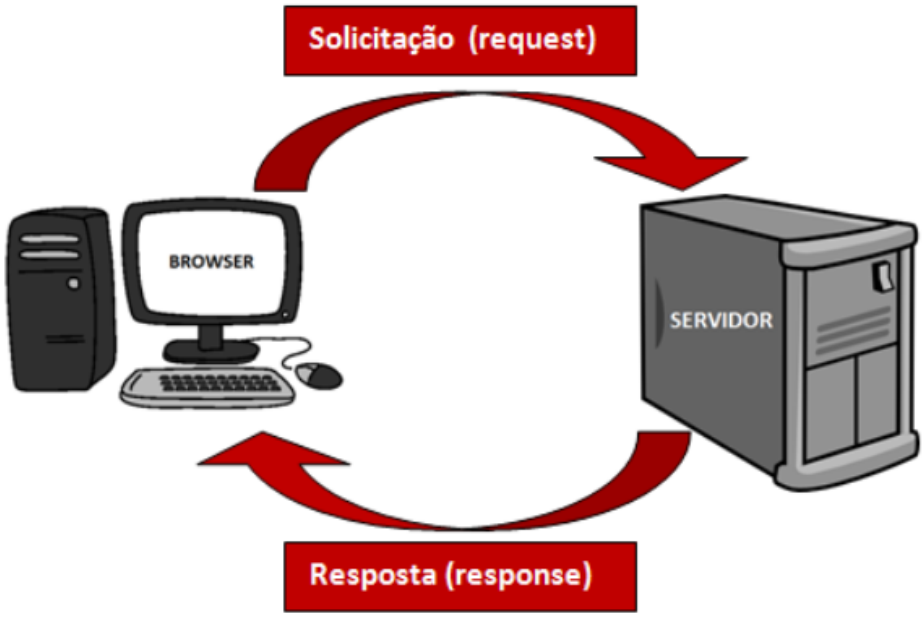



**Figura 1. Arquitetura cliente e servidor.**
Fonte: http://www.devmedia.com.br/como-funcionam-as-aplicacoes-web/25888

O protocolo HTTP possui dois métodos que serão utilizados com muita frequência: o GET e o POST. Através do método GET, os parâmetros de solicitação são anexados à *Uniform Resource Locator* (URL) na barra de endereço do navegador e o servidor simplesmente atende ao conteúdo requisitado (Basham, Bates e Sierra, 2008). É recomendado que esse método não seja utilizado com dados confidenciais, já que estes ficaram expostos na barra de endereço. Além disso, esse método não suporta um número extenso de parâmetros.

O método POST é indicado para realizar solicitações mais complexas para o servidor. Por exemplo, este método é utilizado para o envio de dados de um formulário para o servidor a fim de realizar operações de persistência no banco de dados. Os dados são enviados de forma encapsulada no corpo da solicitação. Dependendo da funcionalidade, um método do protocolo HTTP será mais adequado (Basham, Bates e Sierra, 2008).

O Java EE utiliza páginas Java *Server Pages* (JSP) e as *Servlets*

para proporcionar dinamismo à aplicação *web*. Uma página JSP possibilita a combinação de códigos em HTML e Java, sendo que o código em Java será processado por um servidor *web*. As *Servlets* são classes em Java que atendem as requisições do protocolo HTTP, portanto, as *Servlets* são responsáveis por receber uma solicitação do cliente e por devolver uma resposta.

Para serem executadas, as *Servlets* necessitam de um *Container*. O *Container* é servidor *web* que irá gerenciar os recursos usados pelas *Servlets* e será responsável por entregar as *servlets,* as solicitações (*request)* e as respostas (*response)* HTTP (Basham, Bates e Sierra, 2008). O Apache TomCat e o GlassFish são exemplos de servidor ou *Container web* (Apache, 2016).

### *Softwares* necessários e instalação

Para o desenvolvimento de uma aplicação *web* será necessário a instalação dos *softwares* indicados na Tabela 1. O Java *Development Kit* (JDK) e o servidor de banco de dados MySQL já foram instalados por serem *softwares* requisitados na disciplina de Projeto Final I.

**Tabela 2. *Softwares* necessários para o desenvolvimento de uma aplicação *web*.**

| Software | Versão |
|---|---|
| NetBeans IDE | Suporte Java EE - 8.1 |
| Java Development Kit (JDK) | 7 |
| Servidor de banco de dados MySQL | 5.5.8 |
| Servidor *web* Apache | 8.0.27 |

Para a instalação do ambiente de desenvolvimento NetBeans com suporte a tecnologia Java EE, faça o *download* do *software* acessando o seguinte endereço eletrônico: https://netbeans.org/ (NetBeans, 2016). A página que disponibiliza as versões do NetBeans para *download* é exibida na Figura 2. Note que a instalação do NetBeans já contempla duas opções de servidores *web:* o GlassFish Server e o Apache TomCat.

No momento da instalação do NetBeans IDE, o usuário pode escolher entre a instalação de um desses servidores *web.*

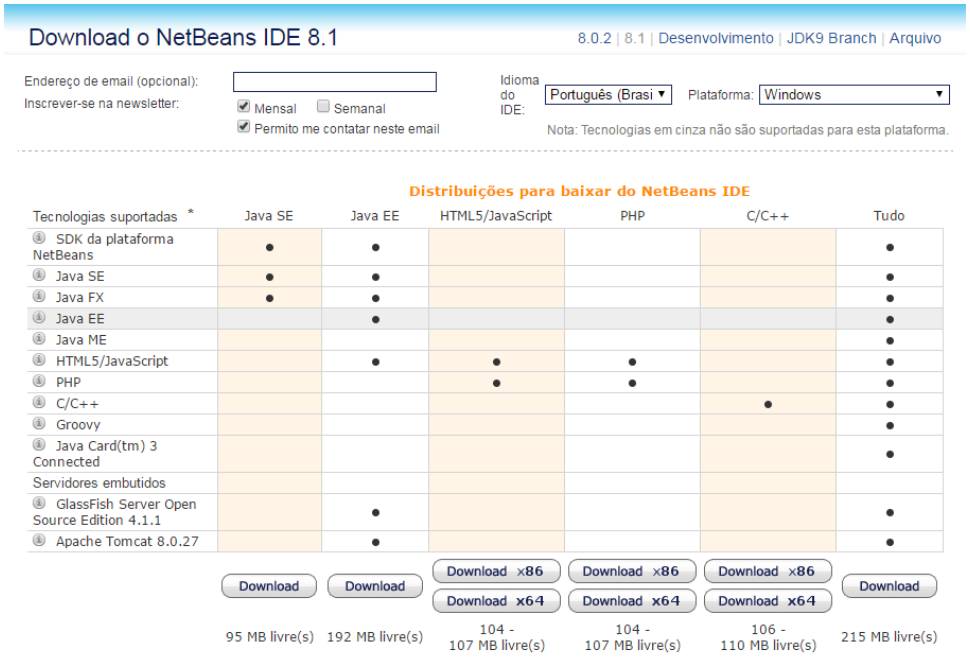



**Figura 2. Página que disponibiliza as versões do NetBeans para *download*.**

Na tela de instalação do NetBeans ilustrada na Figura 3, selecione o Apache TomCat como servidor *web* a ser instalado juntamente com o *software* do NetBeans. Após, aceite os termos de licença do NetBeans e escolha a pasta de destino para a instalação do Apache TomCat e do NetBeans.

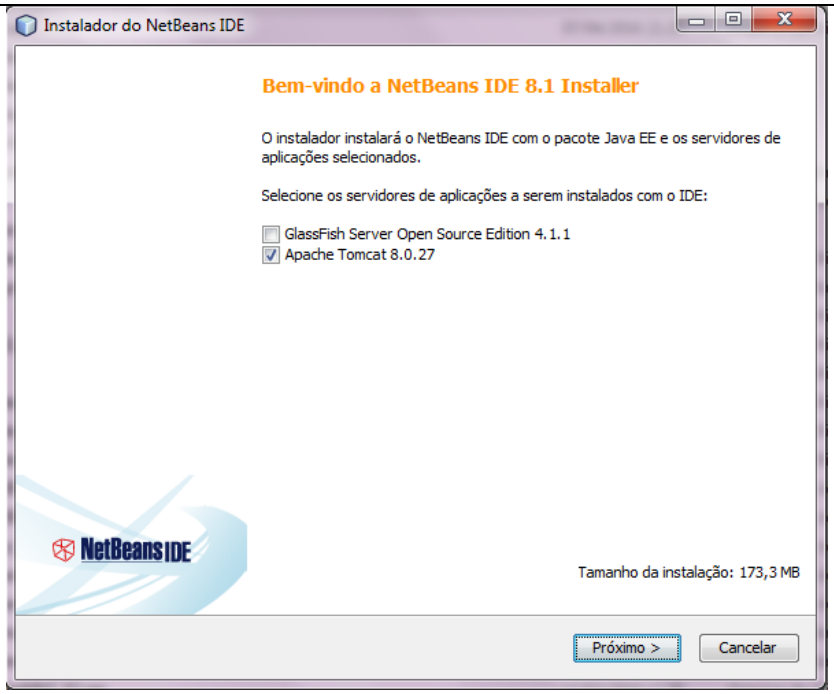



**Figura 3. Tela de instalação do NetBeans.**

## Criando um projeto *web* no NetBeans

Por fim, confirme o resumo de instalação do NetBeans e aguarde a instalação. Para criar um projeto de uma aplicação *web* no NetBeans selecione o menu *Arquivo* e a opção *Novo Projeto.* Na janela aberta, selecione em *Categorias* a opção *Java Web* e em Projetos, a opção *Aplicação Web* conforme ilustrado na Figura 4.

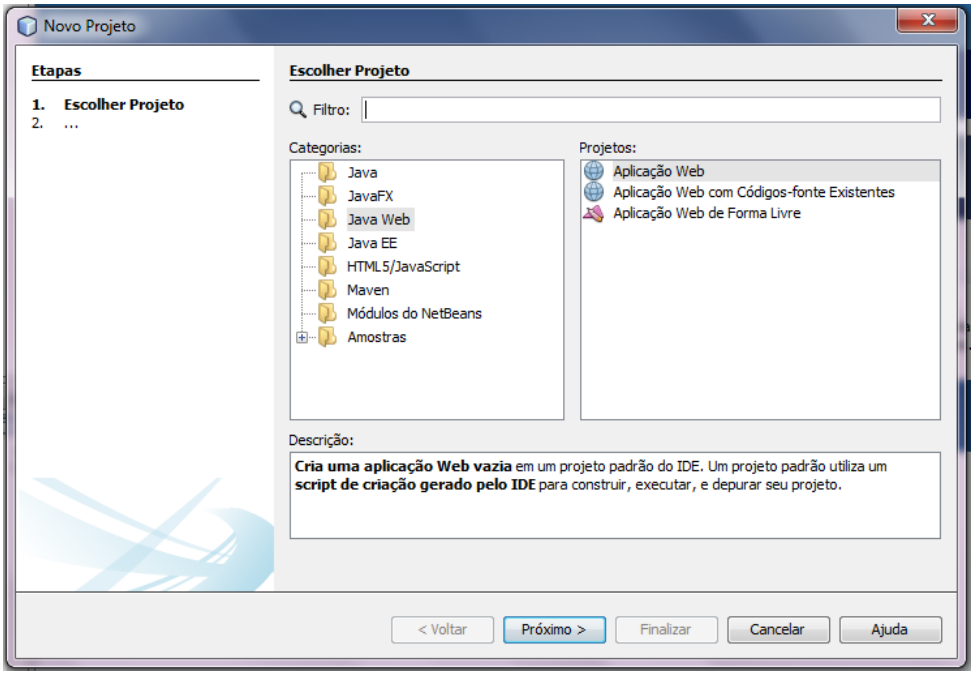



**Figura 4. Criando um projeto *web n*o NetBeans.**

Na próxima tela, informe o nome e a localização do projeto e clique no botão *Próximo*. A definição do servidor *web* e da versão do Java EE é realizada na próxima tela (Figura 5). Por padrão, o servidor *web* Apache TomCat e a versão do Java EE já instalados serão exibidos como opção nesta tela. Apenas confira as informações selecionadas e clique em *Próximo*. No botão adicionar, é possível acrescentar outras opções de servidores *web* desde que os *softwares* já estejam instalados em sua máquina. Na próxima tela de criação de um projeto *web,* é possível selecionar os *frameworks* que serão utilizados no desenvolvimento da aplicação. Neste caso, apenas clique no botão *Finalizar* para criar o projeto *web.*

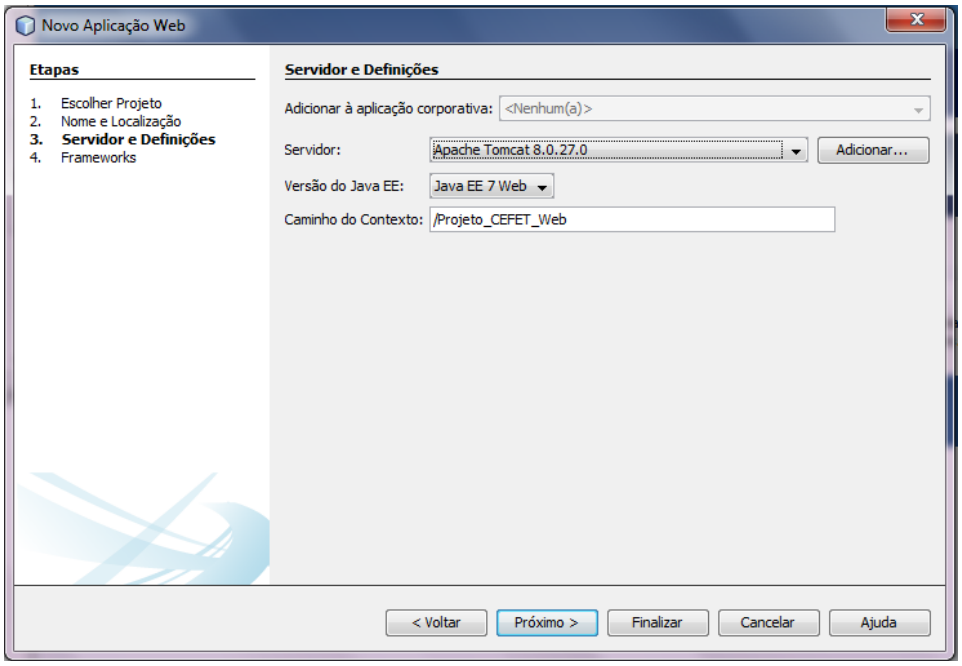



**Figura 5. Configurando o servidor *web* na criação de um projeto *web*.**

## Diretórios do projeto *web*

A estrutura de diretórios do projeto *web* criado é exibido na Figura 6. No diretório *Páginas web* ficam localizadas as páginas HTML e as páginas JSP. Essas páginas correspondem à camada *view* do padrão MVC. Note que na criação do projeto *web*, a página *index.html* foi criada automaticamente. No diretório WEB-INF conterá um arquivo denominado de *web.xml* que será responsável pela mapeamento das *servlets* possibilitando assim, a comunicação entre uma página (HTML ou JSP) e uma determinada *servlet*.

No diretório *Pacotes de Códigos – fonte* ficam localizados os pacotes e as classes em Java. É nesse diretório que será criado os pacotes *model*, *dao* e *Controller* com as suas respectivas classes do padrão MVC. Repare que os códigos em Java ficam separados das páginas de interação com o usuário (*view*). No diretório *Bibliotecas* é possível acrescentar bibliotecas que serão utilizadas em uma aplicação *web*. Observe que as bibliotecas correspondentes ao JDK e do Apache

TomCat foram acrescentadas automaticamente.

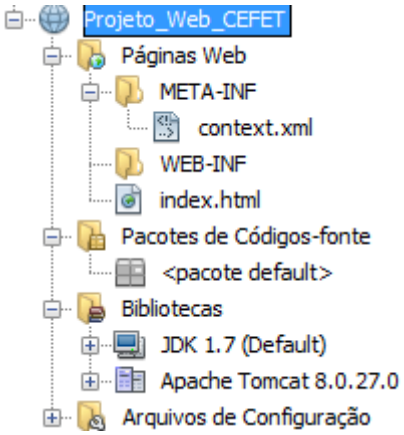



**Figura 6. Diretórios do projeto *web*.**

Para acrescentar MySQL JDBC *Driver* necessário para a comunicação com o banco de dados do MySQL, clique com o botão direito do *mouse* no diretório *Bibliotecas* e selecione a opção *Adicionar Biblioteca.* Depois escolha a opção *Driver* JDBC do MySQL e clique em *Adicionar Biblioteca.* Esse processo é ilustrado na Figura 7.

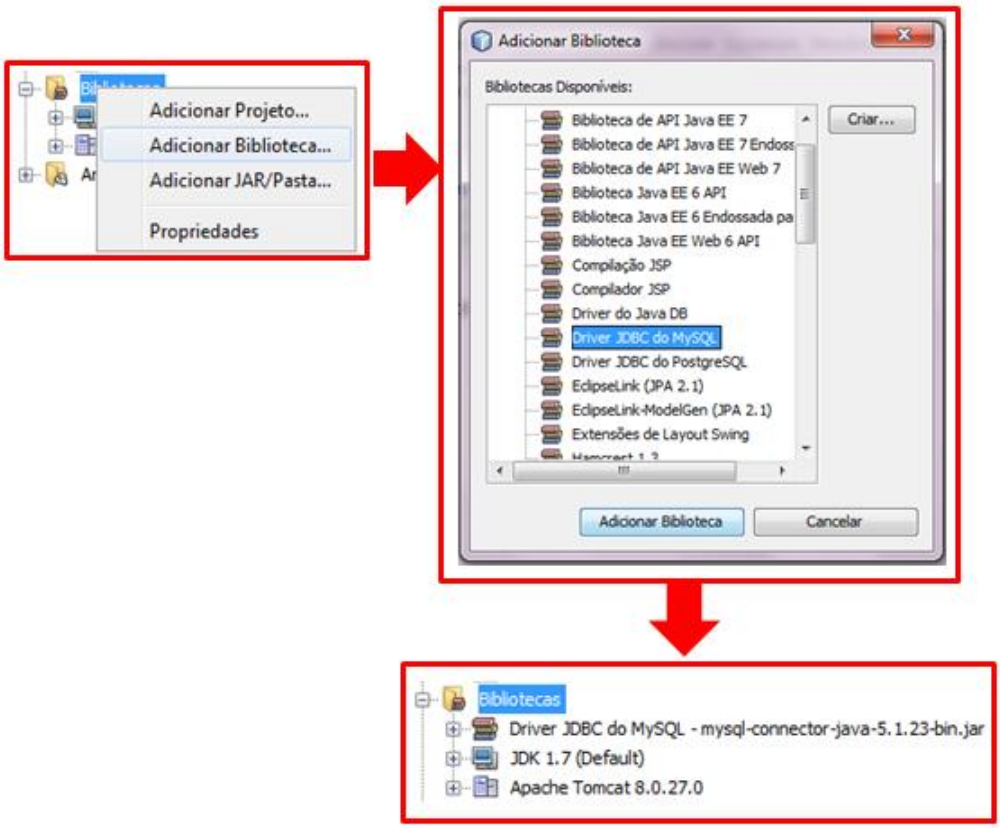



**Figura 7. Adicionando o Driver JDBC do MySQL em um projeto *web*.**

## Executando o projeto *web*

No NetBeans é possível definir o navegador que será utilizado para a execução do projeto. Para isso, clique no menu *Executar* e selecione a opção D*efinir Browser do Projeto.* Agora basta escolher, dentre os navegadores disponíveis, o navegador de sua preferência, conforme ilustrado na Figura 8.

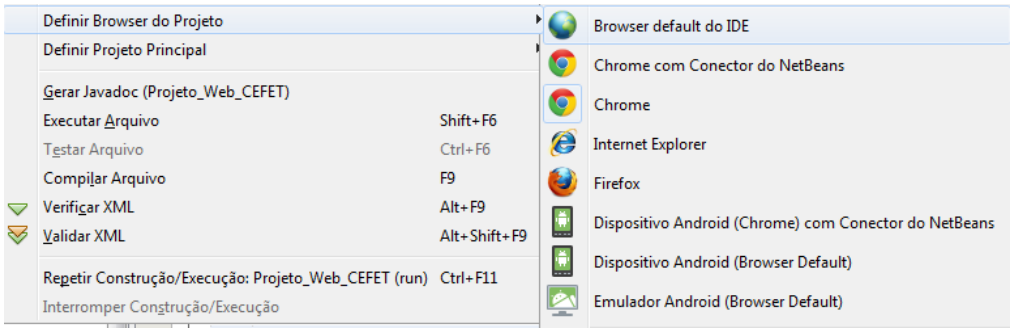



**Figura 8. Definindo o navegador para a execução do projeto *web*.**

Para executar o projeto o *web*, selecione o projeto e clique no botão *Executar Projeto* (ícone de uma seta) na barra de execução do NetBeans (Figura 9). Nesse momento, o servidor *web* (Apache TomCat) será inicializado e a página *index.html* será exibida no navegador definido.

**Figura 9. Barra de execução do NetBeans.**

Na aba de *saída* localizada na parte inferior do NetBeans é possível visualizar a execução do Apache TomCat. Nessa aba, ainda é possível reinicializar, interromper e atualizar o *status* do servidor *web* com os botões disponíveis, conforme ilustração da Figura 10.

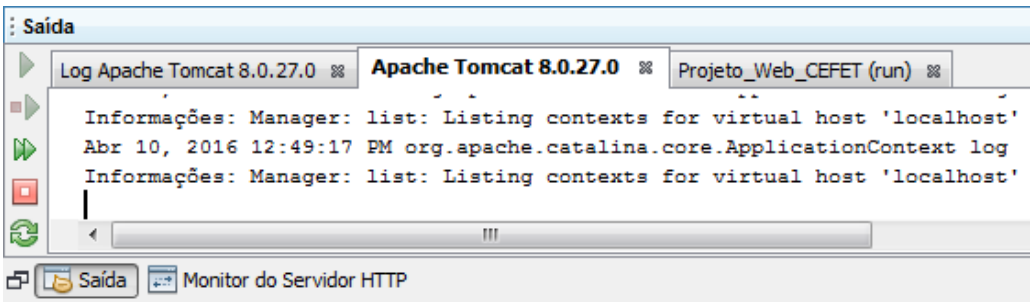
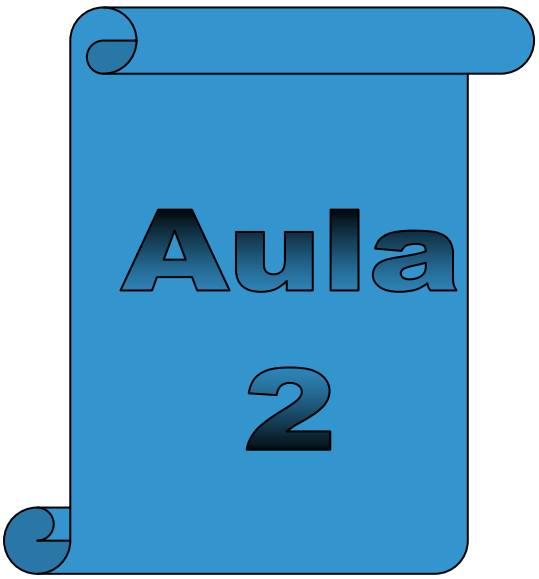



**Figura 10. Aba de saída do NetBeans.**

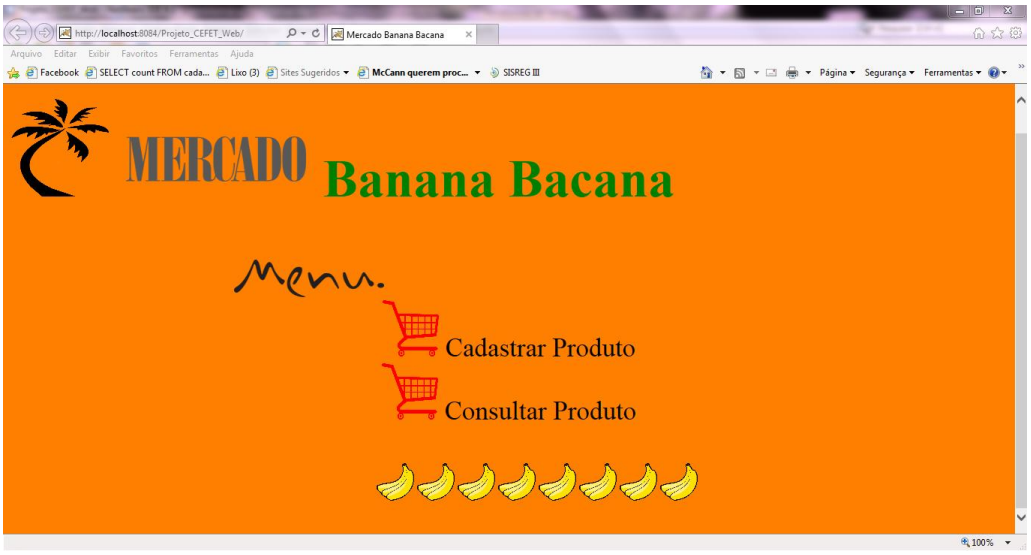



# Criando a camada *view* de uma aplicação *Web*

## Aula 02: Criando a camada *view* de uma aplicação *Web*

Nesta aula será apresentada a construção das páginas que compõem a camada *view* do padrão MVC de uma aplicação *web*. Uma página em JSP denominada de *index* conterá os *links* para as funcionalidades disponíveis na aplicação *web*. Ainda nesta aula, será construída uma pagina em JSP para o cadastro de produtos para o *Mercado Banana Bacana.* Além disso, será demonstrado, passo a passo, a construção dos elementos das páginas no NetBeans, bem como a reutilização de códigos contidos no topo e no rodapé dessas páginas.

Os *layouts* das páginas *index* e cadastro de produto podem ser visualizados, respectivamente na Figura 10 a) e na Figura 10 b). Inicialmente, a página *index* possuirá apenas dois *links*: *Cadastrar Produto* e *Consultar Produto.* A página de cadastro de produto permitirá salvar os dados de um produto no banco de dados. A imagem da seta é um *link* que direciona para a página principal (*index*). Ambas as páginas *web* serão em JSP.




**a)**


**b)**

**Figura 10. a) Página *index*. Figura 10. b) Página cadastro de produto.**

Para criar uma página JSP no NetBeans que permite a combinação de códigos em HTML e Java faça o seguinte: clique com o botão direito do *mouse* em *Páginas Web;* selecione a opção *Novo* e depois, a opção JSP (Figura 11). Na janela de configuração, informe o nome para a página em JSP a ser criada e clique em *Finalizar*. Nesta aula, foram criadas duas páginas em JSP: *index.jsp* e *cadastroProduto.jsp;*

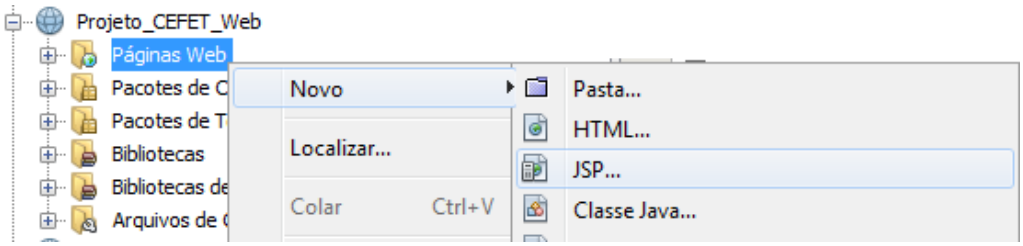



**Figura 11. Criação de uma página JSP.**

Após, crie duas pastas dentro do diretório *Páginas Web* clicando com o botão direito do *mouse* em *Páginas Web*→Novo→Pasta. Uma pasta será denominada de *Imagens* e outra pasta de *Layout_Pagina.*Na pasta *Imagens* ficarão todas as imagens que serão utilizadas nas

páginas *web.* Na pasta *Layout_Pagina* ficarão duas páginas em HTML contendo os códigos referentes ao topo e rodapé das páginas *web.* Assim, crie duas páginas em HTML na pasta *Layout_Pagina* conforme ilustrado na Figura 12.

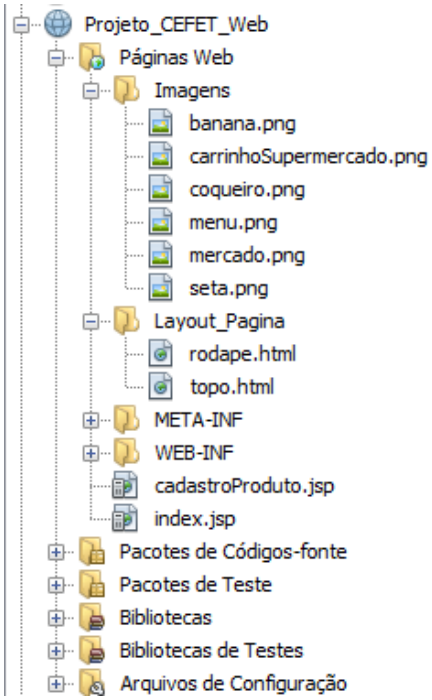



**Figura 12. Diretórios da aplicação *web*.**

Para acrescentar uma imagem na pasta *Imagens*, basta copiar a imagem desejada (Ctrl + C); selecionar a pasta *Imagens* e colar a imagem (Ctrl + V). O NetBeans disponibiliza uma paleta gráfica para inserção de elementos de uma página em HTML. Para ter acesso a essa paleta, clique no menu *Janela* do NetBeans e escolha a opção *Ferramentas do IDE* → *Paleta* (Ctrl + Shift + 8). A Figura 13 ilustra a paleta gráfica do NetBeans e os elementos que podem ser inseridos através dessa paleta. Por exemplo, se a opção *Tabela* for selecionada, o NetBeans irá gerar o código em HTML de uma tabela de acordo com os parâmetros informados pelo programador.

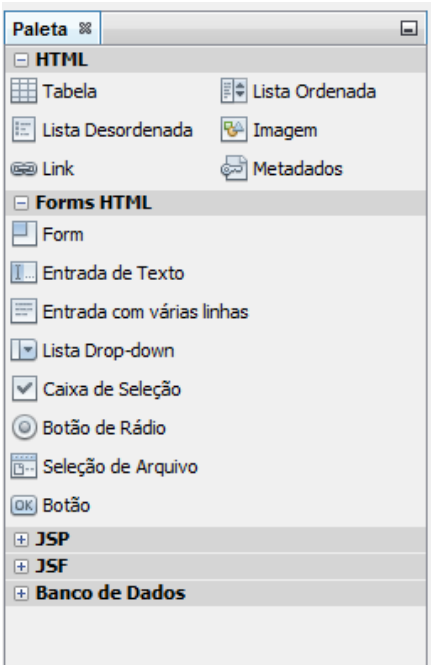



**Figura 13. Paleta gráfica do NetBeans.**

O código da página *index* é exibido na Figura 14. Essa página *web* é composta por uma tabela, imagens, *links* e títulos. O atributo *style* foi utilizado para adicionar estilos às *tags* em HTML de forma a personalizar a sua aparência. Os códigos assinalados em vermelho correspondem aos códigos do rodapé e do topo das páginas. Repare que tanto a página *index*, quanto a página cadastro de produto conservam a mesma aparência para o topo e o rodapé. Além disso, todas as páginas a serem criadas possuirão o mesmo *layout* representado pelo código assinalado.

Sendo assim, uma boa prática seria reaproveitar esse código. Para isso, basta retirar o código assinalado pelo primeiro retângulo vermelho e escrevê-lo na página *topo.html* localizada na pasta *Layout_Pagina*.O segundo retângulo em vermelho deve ser retirado e escrito na página *rodape.html.* Note que os códigos que representam cada parte da página (topo e rodapé) ficaram armazenados de forma independente do restante do código, possibilitando assim, reutilizando do código.

```
<body style="background-color: #FF7F00;">
    <table>
        <tr>
            <td>
                <h1 style="color: green; font-size: 70px; font-weight: bold">
                    <img src="Imagens/coqueiro.png" width="130" height="130" alt="coqueiro"/>
                    <img src="Imagens/mercado.png" width="250" height="100" alt="mercado"/>
                    Banana Bacana
                </h1>
            </td>
        </tr>
    </table>
    <img src="Imagens/menu.png" width="200" height="50" alt="menu" style="margin-left: 300px"/>
    <div style=" text-align: center; margin-bottom: 30px">
        <a href="cadastroProduto.jsp" style="font-size: 35px; text-decoration: none; color: #000000" >
            <img src="Imagens/carrinhoSupermercado.png" width="75" height="75" alt="carrinho"/>
            Cadastrar Produto
        </a> <br>
        <a href="#" style="font-size: 35px; text-decoration: none; color: #000000" >
            <img src="Imagens/carrinhoSupermercado.png" width="75" height="75" alt="carrinho"/>
            Consultar Produto
        </a>
    </div>
    <div style="margin-left: 490px">
        <img src="Imagens/banana.png" width="50" height="50" alt="banana"/>
        <img src="Imagens/banana.png" width="50" height="50" alt="banana"/>
        <img src="Imagens/banana.png" width="50" height="50" alt="banana"/>
        <img src="Imagens/banana.png" width="50" height="50" alt="banana"/>
        <img src="Imagens/banana.png" width="50" height="50" alt="banana"/>
    </div>
</body>
```

**Figura 14. Código da página *index* em JSP.**

O código da página *topo* é exibido na Figura 15. Para reutilizar o código da página *topo e rodape* será necessário utilizar duas diretivas. Uma diretiva é um recurso que fornece instruções especiais ao *Container* (servidor *web*) no momento da tradução da página (Basham, Bates e Sierra, 2008). As diretivas são classificadas em *page, include* e *taglib* e são inseridas em uma página JSP através do seguinte símbolo <%@ %>. Nessa aula, apenas a diretiva *include* será usada para importar uma página.

```
<html>
    <head>
        <title>Topo</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
     <table>
            <tr>
                <td>
                    <h1 style="color: green; font-size: 70px; font-weight: bold">
                        <img src="Imagens/coqueiro.png" width="130" height="130" alt="coqueiro"/>
                        <img src="Imagens/mercado.png" width="250" height="100" alt="mercado"/>
                        Banana Bacana
                    </h1>
                </td>
            </tr>
        </table>
     </body>
</html>
```

**Figura 15. Código da página *topo* em HTML.**

A diretiva *include* possibilita a inclusão de pedaços de código como títulos, imagens e barra de navegação em uma página JSP. Sendo assim, o código que se repete entre as páginas de um mesmo *layout* não precisam ser reescritas novamente, portanto, a diretiva *include* permite a reutilização de um código. Para incluir os códigos da pagina *topo* e *rodape* na página *index,* basta incluir duas diretivas *include* nos locais desejados. Acrescentado as diretivas para a inclusão das páginas *topo* e *rodape,* o código da página *index* fica conforme ilustrado na Figura 16.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Mercado Banana Bacana</title>
    </head>
    <body style="background-color: #FF7F00;">

        <%@include file="Layout_Pagina/topo.html"%>

        <img src="Imagens/menu.png" width="200" height="50" alt="menu" style="margin-left: 300px"/>
        <div style=" text-align: center; margin-bottom: 30px">
            <a href="cadastroProduto.jsp" style="font-size: 35px; text-decoration: none; color: #000000" >
                <img src="Imagens/carrinhoSupermercado.png" width="75" height="75" alt="carrinho"/>
                Cadastrar Produto
            </a> <br>
            <a href="#" style="font-size: 35px; text-decoration: none; color: #000000" >
                <img src="Imagens/carrinhoSupermercado.png" width="75" height="75" alt="carrinho"/>
                Consultar Produto
            </a>
        </div>
        <div style="margin-bottom: 40px;">

        <%@include file="Layout_Pagina/rodape.html"%>

        </div>
    </body>
</html>
```

**Figura 16. Código da página *index* com a inclusão das diretivas do tipo *include*.**

Para incluir uma tabela, basta selecionar o local desejado para a inserção e clique duas vezes com o *mouse* no botão *Tabela* da paleta gráfica do NetBeans. Na janela de configuração que será aberta, o programador pode informar os parâmetros para a geração da tabela de forma personalizada. Por exemplo, a tabela da pagina *topo* possui uma linha e uma coluna; o tamanho da borda é nulo assim como as demais opções de configuração (Figura 17). Após, clicar no botão *OK,* o código referente à tabela será criado. Na tabela da página *topo,* as *tags thead (*cabeçalho) e *tbody* foram retiradas.

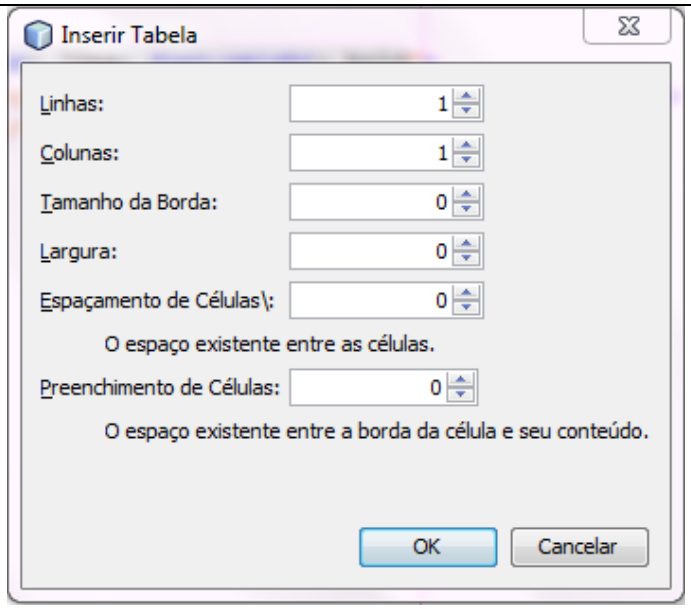



**Figura 17. Janela de configuração de uma tabela.**

Para adicionar uma imagem, selecione o local desejado para a inserção e clique duas vezes com o *mouse* no botão *Imagem* da paleta gráfica do NetBeans. Na janela de configuração, o programador informa a localização da imagem, a largura, a altura e um texto alternativo que será exibido caso a imagem não puder ser carregada (Figura 18).

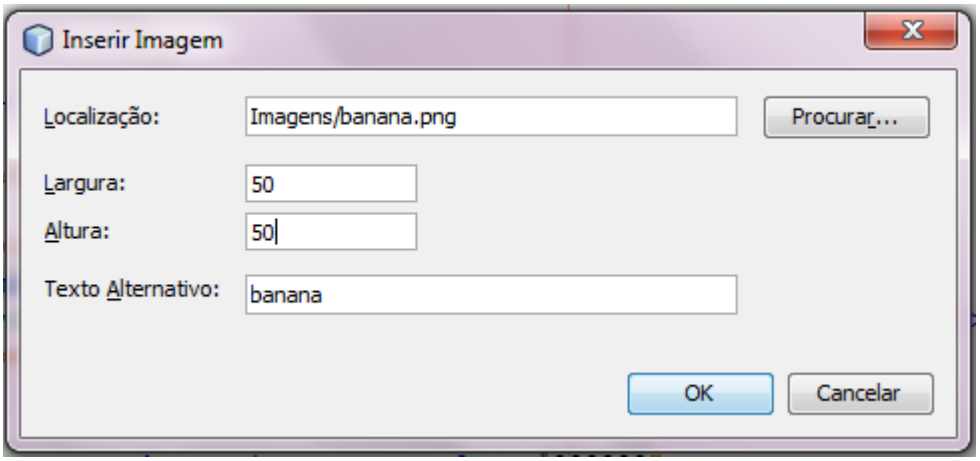



**Figura 18. Janela de configuração de uma imagem.**

Através do botão *Link* na paleta gráfica do NetBeans, uma âncora *(tag a)* pode ser inserida. O principal atributo dessa *tag* é o *href* que contém o nome da página que deve ser carregada no momento do clique na âncora. Repare que dentro da *tag* âncora (*a*) é possível acrescentar uma *tag img e que a tag div* foi utilizada para definir um estilo específico

para uma parte do código.

O código da página cadastro de produto é exibido na Figura 19. Nesta página *web* é exibido para o usuário um formulário para o cadastro de produtos. Esse formulário possui dois botões; um para salvar os dados do formulário no banco de dados e um outro botão para limpar os dados do formulário. A imagem da seta é uma âncora que direciona o usuário para a página *index.*

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Mercado Banana Bacana</title>
    </head>
    <body style="background-color: #FF7F00;">
        <%@include file="Layout_Pagina/topo.html"%>
        <form action="CadastroProdutoController.do" method="post" title="cadastroProduto">
            <table border="1" cellpadding="1" style="width: 500px">
                <thead>
                    <tr>
                        <th colspan="2">Cadastro de Produto</th>
                    </tr>
                </thead>
                <tbody>
                    <tr>
                        <td><label>Descrição:</label></td>
                        <td><input style="width: 350px;" type="text" name="descricao"/></td>
                    </tr>
                    <tr>
                        <td><label>Quantidade:</label></td>
                        <td><input style="width: 50px"type="text" name="qtd"/></td>
                    <td><label>Fornecedor:</label></td>
                        <td><select style="width: 200px" name="fornecedor" >
                                <option value="nulo">Selecione uma opção</option>
                                <option>Assai</option>
                                <option>kuhneheitz</option>
                                <option>Nova Mix</option>
                                <option>Solostocks</option>
                            </select></td>
                    </tr>
                    <tr>
                        <td><label>Preço:</label></td>
                        <td><input style="width: 125px" type="text" name="preco"/></td>
                    </tr>
                    <tr>
                        <td><label>Vendas On-line:</label></td>
                        <td><input type="radio" name="vendas" value="sim" /><label>Sim</label>
                            <input type="radio" name="vendas" value="nao" /><label>Não</label></td>
                    </tr>
                    <tr>
                        <td colspan="2" style="text-align: center">
                            <input type="reset" value="Limpar" />
                            <input type="submit" value="Salvar" /></td>
                    </tr>
                </tbody>
            </table>
        </form>
        <%@include file="Layout_Pagina/rodape.html"%>
        <a href="index.jsp"> <img src="Imagens/seta.png" width="70" height="70" alt="seta"/></a>
    </body>
</html>
```

**Figura 19. Código da página cadastro de um produto em JSP.**

Note que as diretivas *include* foram novamente usadas para inserir o topo e rodapé da página. Assim os códigos não precisam ser reescritos novamente. Todos os campos do formulário possuem preenchimento obrigatório (Figura 20). A *tag form* possui dois principais atributos: *action* e *method.* O valor do *action* será explicado nas próximas aulas e o *method* define o método que será usado para enviar os dados do formulário para o servidor. Os possíveis métodos a serem utilizados foram explicados na primeira aula (*GET* e *POST*). Por se tratar de um formulário, o método *post* foi escolhido.


**Figura 20. Formulário de cadastro de produto.**

O formulário foi inserido através da opção *Form* na paleta gráfica do NetBeans; as caixas de texto foram inseridas através do botão *Entrada de Texto*; a caixa de seleção *Fornecedor* foi inserida através do botão *Lista Drop-down*; o botão de opção *Vendas On-line* foi inserido através da opção *Botão de Rádio* e os botões *Limpar* e *Salvar* foram inseridos através da opção B*otão*.


# Criando uma *servlet* para o cadastro de um produto

## Aula 03: Criando uma *servlet* para o cadastro de um produto

Nesta aula será demonstrada a criação de uma *servlet* destinada ao cadastro de um produto do *Mercado Banana Bacana*. Além disso, será apresentado como uma *servlet* é mapeada em um projeto *web*, bem como alguns detalhes de funcionamento. Ainda nesta aula, a classe *Produto* pertencente à camada *model* será criada, assim como a classe *ProdutoDAO* do pacote *dao.*

As *servlets* possuem dois principais métodos: o *doPost()* e o *doGet()*. Quando uma aplicação *web* recebe uma solicitação para uma *servlet,* o *container* é responsável por chamar um dos métodos do *servlet*. É importante ressaltar que uma s*ervlet* não possui um método *main()* e que o *container* gerencia o clico de vida de uma *servlet* (Basham, Bates e Sierra, 2008). *As servlets* em um projeto *web* pertencem à camada *controller* do padrão MVC.

Cada *servlet* de uma aplicação *web* ficará incumbida de atender uma requisição específica do cliente. Por exemplo, para cadastrar, consultar, alterar e excluir (CRUD) um produto do *Mercado Banana Bacana* serão necessárias quatro *servlets*, ou seja, cada *servlet* é designada para executar uma funcionalidade. Normalmente, apenas um dos métodos principais de uma *servlet* (*doPost()* ou *doGet())* é utilizado para receber e responder uma solicitação HTTP. Um *servlet* é uma extensão da classe *HttpSertvlet*.

Na segunda aula, as páginas *web* pertencentes à camada *view* do padrão MVC foram criadas. Primeiramente, para implementar as classes das camadas *model*, *controller* e o *dao,* três pacotes devem ser criados no diretório *Pacotes de Códigos-fonte.* Para isso, clique com o botão direito do *mouse* na pasta deste diretório e escolha a opção Novo→Pacote Java. A Figura 21 exibe os pacotes criados. Repare que as páginas da camada *view* ficam separadas do código da lógica da aplicação.

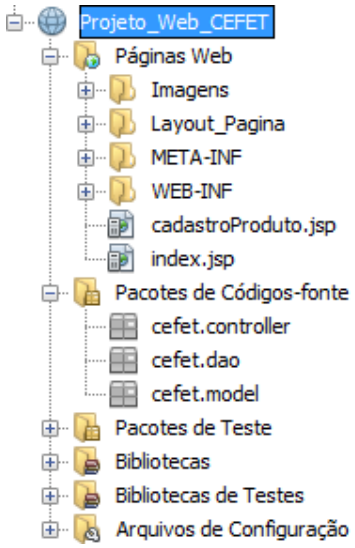



**Figura 21. Estrutura de diretórios do projeto *web*.**

Para criar uma *servlet*, clique com o botão direito do *mouse* no pacote *cefet.controller* e selecione a opção Novo→Servlet. Na primeira tela de criação de uma *servlet*, informe o nome da *servlet* no campo *Nome da Classe* e clique no botão *Próximo*. Na próxima tela, marque a opção *Adicionar informações ao descritor de implantação (web.xml)*, altere o nome do campo *Padrão(ões) de URL* acrescentado a terminação (*.do*) e clique no botão *Finalizar* (Figura 22).

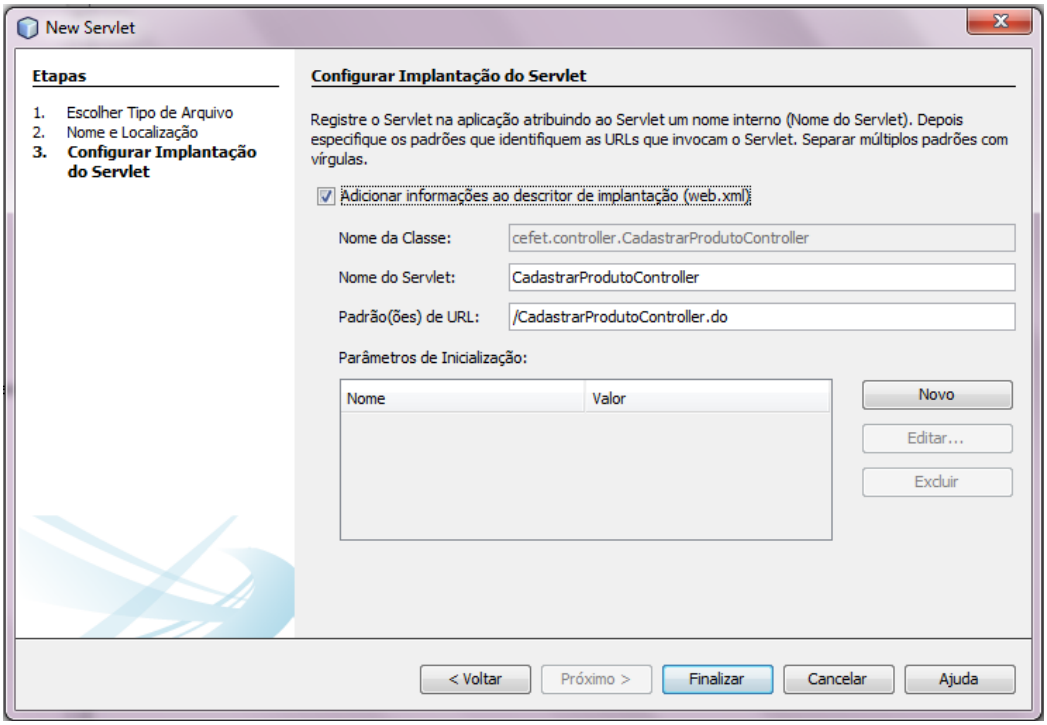
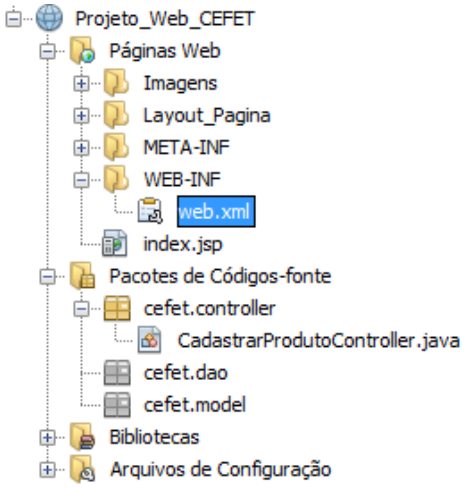



**Figura 22. Tela para a criação de uma *servlet*.**

Quando a primeira *servlet* for criada, automaticamente, um arquivo *Extensible Markup Language* (XML) chamado de *web.xml* será criado no diretório *WEB-INF.* Trata-se do descritor de implantação *(Deployment Descriptor - DD)* que conterá as instruções necessárias a execução das *servlets* e elementos que irão mapear as URLs aos *servlets.* A Figura 23 a) ilustra a localização do arquivo *web.xml* e a Figura 23 b) ilustra o código do arquivo gerado.




**a)**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
    <servlet>
        <servlet-name>CadastrarProdutoController</servlet-name>
        <servlet-class>cefet.controller.CadastrarProdutoController</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>CadastrarProdutoController</servlet-name>
        <url-pattern>/CadastrarProdutoController.do</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
</web-app>
```

**b)**

**Figura 23. a) Diretórios de um projeto *web*. Figura 23. b) Código do arquivo *web.xml*.**

Para cada aplicação *web* existirá apenas um descritor de implantação que conterá o mapeamento de todas as *servlets* utilizadas na aplicação. O elemento *servlet-mapping* identifica para o *container* qual a *servlet* que irá atender uma solicitação do cliente. O *url-pattern* localizado dentro do elemento *servlet-mapping* recebe um nome que

será utilizado pelo cliente para invocar uma *servlet* (Basham, Bates e Sierra, 2008). Note que esse nome foi definido no momento da criação da *servlet* (*CadastrarProdutoController.do)* e que o cliente não conhece o nome legítimo da classe *servlet.*

Para encontrar o nome verdadeiro da *servlet* que foi invocada pelo cliente, o *servlet-name* do elemento *servlet-mapping* deve ser idêntico ao *servlet-name* do elemento *servlet.* O *servlet-class* localizado dentro do elemento *servlet* possui o nome verdadeiro da classe *servlet* que deverá atender a solicitação do cliente. Por questões de segurança, esse nome não é visualizado pelo cliente e isso é assegurado pelo o mapeamento das *servlets.*

Esse mapeamento é gerado para cada *servlet* que for utilizada em uma aplicação *web*. Por fim, o *session-timeout* localizado no elemento *session-config* estabelece o tempo em minutos para que uma sessão seja encerrada, caso o cliente não faça nenhuma solicitação. Por padrão, o valor de *timeout* é de trinta minutos e esse pode ser modificado pelo programador.

A Figura 24 ilustra o código da classe *servlet* criada. Note que as assinaturas dos métodos *doPost()* e *doGet()* foram geradas automaticamente. O código do método *processRequest* deve ser apagado, pois esse método não será utilizado na solução desse projeto *web*. Nessa *servlet,* o método *doPost()* será implementado para realizar o cadastro de um produto no banco dados. O código desse método será explicado na próxima aula.

```
import ...6 linhas

public class CadastrarProdutoController extends HttpServlet {
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            /* TODO output your page here. You may use following sample code. */
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet CadastrarProdutoController</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Servlet CadastrarProdutoController at " + request.getContextPath() + "</h1>");
            out.println("</body>");
            out.println("</html>");
        }
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {...3 linhas }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {...3 linhas }

    @Override
    public String getServletInfo() {...3 linhas }
}
```

**Figura 24. Código da *servlet CadastrarProdutoController*.**

Para que os dados de um produto no formulário da página *cadastroProduto.jsp* sejam encaminhados para a *servlet CadastroProdutoController*, o valor do atributo *action* da *tag form* deve ser o mesmo nome estabelecido no elemento *url-pattern* do arquivo *web.xml* e o atributo *method* também da *tag form* deve ser definido como post (Figura 25).

```
<form action="CadastroProdutoController.do" method="post" title="cadastroProduto">
```

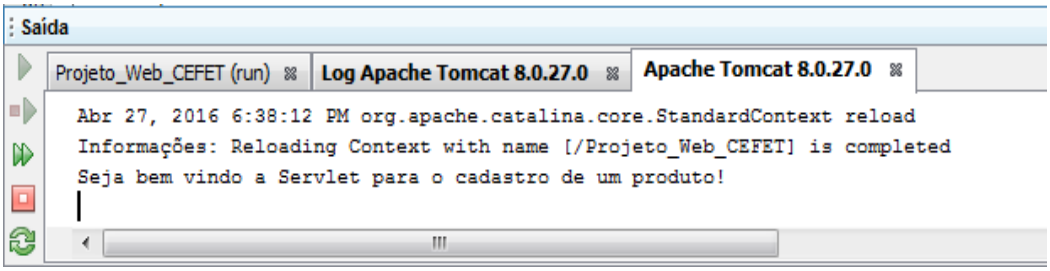
**Figura 25. *Tag form* da página *cadastroProduto.jsp*.**

Para testar a execução do mapeamento da *servlet* para o cadastro de um produto, utilize o método *out.println* para exibir uma mensagem na aba *Saida* do NetBeans (Figura 26.a). Quando o usuário acionar o botão *Salvar* na tela de cadastro de um produto, a *servlet* que irá atender essa solicitação do cliente será localizada pelo *container*. Depois da identificação da *servlet*, o *container* irá chamar o método *doPost()* ou *doGet()* desta *servlet* conforme estabelecido na solicitação. Neste caso,

o método *doPost()* será chamado e a mensagem será exibida no *console* (Figura 26.b).

```
package cefet.controller;

import java.io.IOException;
import static java.lang.System.out;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class CadastrarProdutoController extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

       out.println("Seja bem vindo a Servlet para o cadastro de um produto!");
    }

    @Override
    public String getServletInfo() {
        return "Short description";
    }
}
```

**a)**




**b)**

**Figura 26. a) Código da *servlet* para o cadastro de um produto. Figura 26. b) Aba *Saída* do NetBeans.**

Para continuar a implementação do método para o cadastro de um produto, a classe *Produto* pertencente à camada *model* deve ser criada (Figura 27). O método *cadastrarProduto()* irá chamar o método

*cadastrarProduto()* da classe *ProdutoDAO.*

```
package cefet.model;
import cefet.dao.ExceptionDAO;
import cefet.dao.ProdutoDAO;

public class Produto {
    private int idProduto;
    private String descricao;
    private int quantidade;
    private String fornecedor;
    private float preco;
    private String vendasOnLine;

    public void setIdProduto(int idProduto) {this.idProduto = idProduto;}
    public void setDescricao(String descricao) {this.descricao = descricao;}
    public void setQuantidade(int quantidade) {this.quantidade = quantidade;}
    public void setFornecedor(String fornecedor) {this.fornecedor = fornecedor;}
    public void setPreco(float preco) {this.preco = preco;}
    public void setVendasOnLine(String vendasOnLine) {this.vendasOnLine = vendasOnLine;}
    public int getIdProduto() {return idProduto;}
    public String getDescricao() {return descricao;}
    public int getQuantidade() {return quantidade;}
    public String getFornecedor() {return fornecedor;}
    public float getPreco() {return preco;}
    public String getVendasOnLine() {return vendasOnLine;}

    public void cadastrarProduto(Produto produto) throws ExceptionDAO {
        new ProdutoDAO().cadastrarProduto(produto);
    }
}
```

**Figura 27. Código da classe *Produto* pertencente à camada *model* do padrão MVC .**

Para estabelecer uma conexão com o banco de dados a classe *ConexaoBancoDeDados* foi criada. Nesta classe, o *driver* do servidor de banco de dados MySQL é especificado através do método *Class.forName()*. No método *getConnection()* da classe *DriverManager* é definido o nome do banco de dados (*mercado*), o usuário (*root*) e a senha (*banana*) para estabelecer uma conexão com o banco de dados. O método *getConnection()* dessa classe retorna a conexão estabelecida.

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class ConexaoBancoDeDados {

    public Connection getConnection() {

        Connection conn = null;

        try {
            Class.forName("com.mysql.jdbc.Driver");
        } catch (Exception e) {
            e.printStackTrace();
        }
        try {
            conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/mercado", "root", "banana");
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return conn;
    }
}
```

**Figura 28. Código da classe *ConexaoBancoDeDados* do pacote *dao* .**

No pacote *dao* deve ser criado ainda as classes *ProdutoDAO* e a *ExpectionDAO*. A classe *ProdutoDao* possuirá os métodos para cadastrar, consultar, alterar e excluir os registros de um produto no banco de dados. Nesta aula, apenas o método *cadastrarProduto()* será criado conforme o código ilustrado na Figura 29.

```java
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import cefet.model.Produto;
public class ProdutoDAO {

    public void cadastrarProduto(Produto produto) throws ExceptionDAO {
        String sql = "insert into produto (descricao,quantidade,fornecedor,preco,vendas) values (?,?,?,?,?)";
        PreparedStatement stmt = null;
        Connection connection = null;
        try {
            connection = new ConexaoBancoDeDados().getConnection();
            stmt = connection.prepareStatement(sql);
            stmt.setString(1, produto.getDescricao());
            stmt.setInt(2, produto.getQuantidade());
            stmt.setString(3, produto.getFornecedor());
            stmt.setFloat(4, produto.getPreco());
            stmt.setString(5, produto.getVendasOnLine());
            stmt.execute();
        } catch (SQLException e) {
            e.printStackTrace();
            throw new ExceptionDAO("Erro ao cadastrar produto:" + e);
        } finally {
            try {
                if (stmt != null) { stmt.close();}}
            catch (SQLException e) { e.printStackTrace();}
            try {if (connection != null) {connection.close();}
            } catch (SQLException e) {e.printStackTrace();}
        }
    }
```

**Figura 29. Código da classe *ProdutoDAO* do pacote *dao.***

Nesta classe, o método *getConnection()* é chamado para estabelecer uma conexão com o banco de dados. O comando *SQL* para a inserção de um registro no banco de dados é preparado para a execução através do método *prepareStatement()*. Após, os parâmetros necessários para a execução do banco de dados são apontados. Para cada parâmetro oculto (interrogação) do comando *SQL* deve ser apontado um valor (variável). Para qualquer erro de execução, uma exceção do tipo *ExceptionDAO* será lançada e após, a execução do método *cadastrarProduto()*, as conexões estabelecidas devem ser fechadas. O código referente à classe *ExceptionDAO* é exibido na Figura 30. Essa classe é uma extensão da classe *Exception* e é utilizada para lançar exceções específicas das classes do pacote *dao.*

```
package cefet.dao;

public class ExceptionDAO extends Exception{

    public ExceptionDAO(String mensagem){
                super(mensagem);
        }
}
```

**Figura 30. Código da classe *ExceptionDAO* do pacote *dao.***


# Programando uma *servlet* – Método cadastrar

## Aula 04: Programando uma *servlet* - Método cadastrar

Nesta aula será demonstrada a finalidade de cada código utilizado no método *doPost()* da *servlet CadastrarProdutoController*. Será explicado como os dados enviados pelo formulário na página de cadastro de produto serão recuperados pela *servlet*. Depois que os dados forem recuperados, uma verificação será feita pela *servlet* a fim de identificar se todos os campos do formulário foram preenchidos. Além disso, será apresentado como uma mensagem destinada ao usuário pode ser enviada para uma página em JSP.

Primeiramente, os dados enviados pelo formulário de cadastro de um produto serão recuperados através do método *getParameter()*. Esse método permite o acesso aos parâmetros de uma solicitação HTTP e retorna sempre um valor do tipo String oriundo do cliente. Além disso, o método *getParameter()* pode ser utilizado para recuperar os parâmetros de uma solicitação HTTP usando o método *doPost()* ou o método *doGet()*. A Figura 31 ilustra a recuperação dos dados do formulário de cadastro de um produto.

```
String descricao = request.getParameter("descricao");
String quantidade = request.getParameter("qtd");
String fornecedor = request.getParameter("fornecedor");
String preco = request.getParameter("preco");
String vendas = request.getParameter("vendas");
```

**Figura 31. Código para a recuperação dos dados oriundos do formulário.**

Quando utilizado com método *doPost()*, o método *getParameter()* recebe como parâmetro o valor do atributo *name* das *tags* do formulário. Quando utilizado com o método *doGet()*, os parâmetros de uma solicitação HTTP ficam localizados na URL após o ponto de interrogação e os parâmetros são separados pelo caractere (&) na URL, conforme ilustrado na Figura 32. Caso não exista o parâmetro informado, o valor retornado pelo método *getParameter()* será *null*.

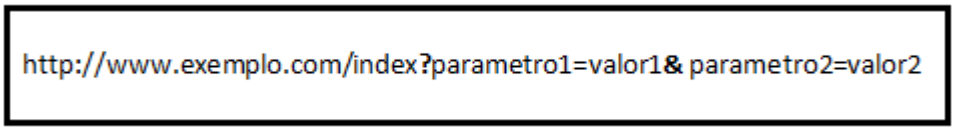



**Figura 32. Exemplo de uma URL com os parâmetros de uma solicitação.**

Depois de recuperar os parâmetros, é necessário verificar se os mesmos são diferentes de *null*. O *array* de parâmetros recebe os dados recuperados através do método *getParameter()* e o *array* de campos recebe os nomes dos campos que deveriam ser preenchidos pelo usuário. Através de um laço de repetição, todos os parâmetros são verificados. Se caso um parâmetro recuperado seja *null*, a *String mensagem* irá armazenar o nome do respectivo campo. Ao final do laço, todos os nomes dos campos não preenchidos pelo usuário estarão concatenados na variável *mensagem*. Para verificar se o campo *Fornecedor* (*tag* <*select*>) foi preenchido, o valor recuperado referente a esse campo deve ser diferente do valor da primeira opção da *tag select* (*String* nulo) (Figura 33).

```
Produto produto = null;
String mensagem = null;

String[] arrayParametros = {descricao, quantidade, fornecedor, preco, vendas};
String[] arrayCampos = {"Descricao", "Quantidade", "Fornecedor", "Preco", "Vendas On-Line"};

try {
    for (int contador = 0; contador < arrayParametros.length; contador++) {
        if (arrayParametros[contador] == null || arrayParametros[contador].length() <= 0 ||
                arrayParametros[contador].equals("nulo")) {
            if (mensagem == null) {
                mensagem = arrayCampos[contador];
            } else {
                mensagem = mensagem + " , " + arrayCampos[contador];
            }
        }

    }
```

**Figura 33. Código de verificação de parâmetros nulos.**

Caso a variável *mensagem* seja diferente de *null,* essa variável será encaminhada para a página de cadastro de produto a fim de informar ao usuário quais campos não foram preenchidos. Para isso, o método *setAttribute()* adiciona um atributo ao objeto de solicitação para que seja

enviado para uma página JSP(Basham, Bates e Sierra, 2008). O primeiro parâmetro desse método é nome do atributo e o segundo, é o valor do atributo. Para encaminhar a mensagem, um objeto do tipo *RequestDispatcher* deve ser instanciado com o nome da página JSP de destino. O método *forward()* solicita ao *container* que inicialize a página JSP enviando um objeto do tipo *resquest* e *response.*

```
if (mensagem != null) {
    mensagem = "Preencha corretamente o(s) campo(s): " + mensagem + ".";
    request.setAttribute("mensagem", mensagem);
    RequestDispatcher dispatcher = request.getRequestDispatcher("cadastroProduto.jsp");
    dispatcher.forward(request, response);
}
```

**Figura 33. Código que encaminha um atributo para uma página JSP.**

Para que a página de cadastro de um produto receba a mensagem, um código na linguagem Java deve ser inserido na página JSP de destino a fim de capturar o objeto *mensagem*. A inserção desse código é realizada através das *tags* <% %> conhecidas como *scriptlet.*

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>

    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Mercado Banana Bacana</title>
    </head>

    <body style="background-color: #FF7F00;">
        <%@include file="Layout_Pagina/topo.html"%>
        <p style="text-align: center; ">
            <%
                String mensagem = (String) request.getAttribute("mensagem");
                if (mensagem != null) {
                    out.print(mensagem);
                }
            %>
        </p>
```

**Figura 34. *Scriptlet* inserido na página de cadastro de produto.**

Através do método *getAttribute(),* o objeto *mensagem* é capturado de acordo com o nome atribuído no método *setAttribute()*. O *casting* foi necessário porque o método *getAttribute()* retorna sempre um objeto do tipo *Object*. Por isso, o programador precisa declarar um *casting* com o

tipo do objeto que de fato será retornado pelo método *getAttribute()*. É necessário certificar-se de que o objeto capturado é diferente de *null* antes de exibi-lo para o usuário. O método *out.print()* exibe o valor do objeto *mensagem* dentro da *tag* <p>.

Caso o formulário de cadastro de produto não seja preenchido corretamente pelo usuário, uma mensagem será exibida ao usuário informando os campos não preenchidos (Figura 35).

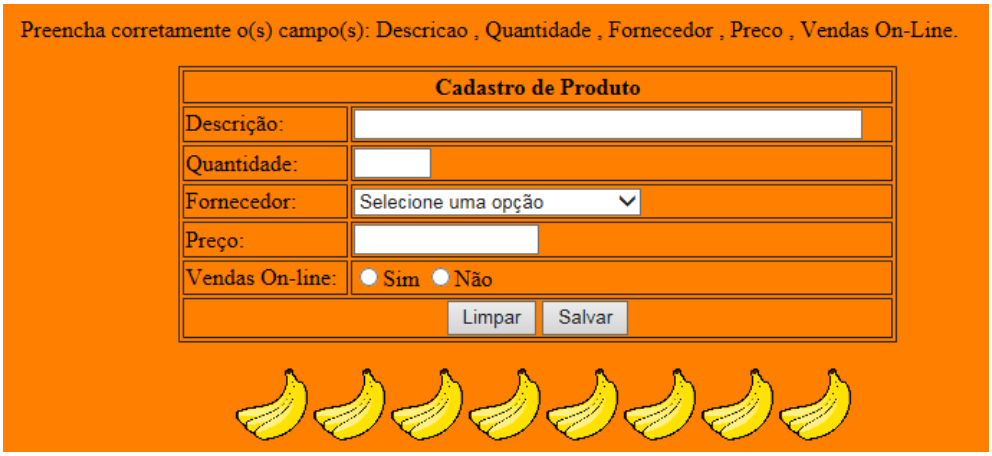



**Figura 35. Mensagem exibida ao usuário na tela de cadastro de um produto.**

Se todos os campos forem preenchidos corretamente, um objeto do tipo *Produto* será instanciado e os parâmetros recebidos serão devidamente *settados* no objeto. Após, o método *cadastrarProduto()* da classe *Produto* (pacote *model*) será chamado, enviando como parâmetro o objeto do tipo *Produto*. Caso a execução ocorra com sucesso, a tela de cadastro de um produto será exibida para o usuário através do objeto do tipo *RequestDispatcher*. Entretanto, se algum erro ocorrer durante a execução, este será capturado e exibido pelo bloco *catch* (Figura 36).

```
        else {
            produto = new Produto();
            produto.setDescricao(descricao);
            produto.setQuantidade(Integer.parseInt(quantidade));
            produto.setFornecedor(fornecedor);
            produto.setPreco(Float.parseFloat(preco));
            produto.setVendasOnLine(vendas);

            produto.cadastrarProduto(produto);
            RequestDispatcher dispatcher = request.getRequestDispatcher("cadastroProduto.jsp");
            dispatcher.forward(request, response);
        }

    } catch (ServletException | IOException | NumberFormatException | ExceptionDAO e) {
        response.getWriter().write("Erro ao cadastrar produto: " + e);
    }
  }
```

**Figura 36. Código que chama o método de cadastro de um produto.**

O código do método *doPost()* da *servlet* para o cadastro de um produto é exibido na íntegra na Figura 37.

```
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

    String descricao = request.getParameter("descricao");
    String quantidade = request.getParameter("qtd");
    String fornecedor = request.getParameter("fornecedor");
    String preco = request.getParameter("preco");
    String vendas = request.getParameter("vendas");

    Produto produto = null;
    String mensagem = null;
    String[] arrayParametros = {descricao, quantidade, fornecedor, preco, vendas};
    String[] arrayCampos = {"Descricao", "Quantidade", "Fornecedor", "Preco", "Vendas On-Line"};
    try {
        for (int contador = 0; contador < arrayParametros.length; contador++) {
            if (arrayParametros[contador] == null || arrayParametros[contador].length() <= 0 ||
                    arrayParametros[contador].equals("nulo")) {
                if (mensagem == null) {
                    mensagem = arrayCampos[contador];
                } else {
                    mensagem = mensagem + " , " + arrayCampos[contador];
                }
            }
        }
        if (mensagem != null) {
            mensagem = "Preencha corretamente o(s) campo(s): " + mensagem + ".";
            request.setAttribute("mensagem", mensagem);
            RequestDispatcher dispatcher = request.getRequestDispatcher("cadastroProduto.jsp");
            dispatcher.forward(request, response);
        }
     else {
         produto = new Produto();
         produto.setDescricao(descricao);
         produto.setQuantidade(Integer.parseInt(quantidade));
         produto.setFornecedor(fornecedor);
         produto.setPreco(Float.parseFloat(preco));
         produto.setVendasOnLine(vendas);

         produto.cadastrarProduto(produto);
         RequestDispatcher dispatcher = request.getRequestDispatcher("cadastroProduto.jsp");
         dispatcher.forward(request, response);
     }} catch (ServletException | IOException | NumberFormatException | ExceptionDAO e) {
     response.getWriter().write("Erro ao cadastrar produto: " + e);
}}
```

**Figura 37. Código do método *doPost()* da *servlet* de cadastro de um produto.**

As classes importadas na *servlet* de cadastro de um produto são ilustradas na Figura 38.

```
import cefet.dao.ExceptionDAO;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import cefet.model.Produto;
import javax.servlet.RequestDispatcher;
```

**Figura 38. Classes importadas na *servlet* de cadastro de um produto.**


# Programando uma *servlet* – Método consultar

## Aula 05: Programando uma *servlet* - Método consultar

Nesta aula será demonstrada a construção de uma tela em JSP destinada à consulta de um registro de produto no banco de dados. Caso exista um produto no banco de dados com a descrição informada pelo usuário, o registro do referido produto será recuperado e exibido em um formulário na mesma tela de consulta. Esse formulário possibilitará a alteração e a exclusão do registro recuperado. Ainda será demonstrada a construção de uma *servlet* (*BuscarProdutoController*) e a implementação do método *buscarProdutoPorDescricao()* na classe *ProdutoDAO.*

A tela de consulta de um produto por descrição é ilustrada na Figura 39. Essa tela pode ser acessada pelo usuário através de um *link* (*Consultar Produto*) na tela principal (*index.jsp*). Nesta tela, o usuário informa a descrição do produto que deseja encontrar e aciona o botão *Buscar.*


**Figura 39. Tela de consulta de um produto em JSP.**

Se um produto com a descrição informada for encontrado, os dados de um produto serão exibidos em um formulário, conforme ilustrada na Figura 40.


**Figura 40. Tela de consulta de produto exibindo um registro recuperado no banco de dados.**

Por questões de melhor visualização, o código da tela de consulta de um produto será explicado em três partes. A primeira parte do código é mostrada na Figura 41. Note que a classe *Produto* pertencente ao pacote *model* é importada através da diretiva *page*. Através da diretiva *include,* o arquivo do topo da página em HTML é incluída na página de consulta possibilitando assim, a reutilização do código.

O método *getAttribute()* permite recuperar um atributo do objeto de solicitação. Neste caso, o objeto recuperado é uma mensagem (String) que será exibida ao usuário caso os campos do formulário não sejam preenchidos corretamente. Na tag *form,* o atributo *action* recebe o nome da *servlet* (*url-pattern)* que ficará responsável por gerenciar a solicitação de uma consulta no banco de dados. Além disso, essa *servlet* será acionada através do método GET, conforme parâmetro informado no atributo *method.*

```
<%@page import="cefet.model.Produto"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Mercado Banana Bacana</title>
    </head>
    <body style="background-color: #FF7F00;">
    <%@include file="Layout_Pagina/topo.html"%>
     <p style="text-align: center;">
            <% String mensagem = (String) request.getAttribute("mensagem");
                if (mensagem != null) {
                    out.print(mensagem);
                }
            %>
        </p>
    <form action="BuscarProdutoController.do" method="get">
    <table border="1" cellspacing="1" cellpadding="1"
    style="margin-left: 450px; margin-bottom: 30px;text-align: center">
        <thead>
            <tr><th colspan="2">Consultar um produto</th></tr>
        </thead>
        <tbody>
            <tr>
                <td><label>Descrição:</label><input size="50" type="text" name="descricao" /> </td>
                <td><input type="submit" value="Buscar" name="buscar" /> </td>
            </tr>
        </tbody>
    </table>
    </form>
```

**Figura 41. Primeira parte do código da tela de consulta de produto.**

A segunda parte do código é exibida na Figura 42. Através do método *getAttribute(),* o objeto do tipo *Produto* enviado pela *servlet BuscarProdutoController* será recuperado. A partir desse objeto, os campos do formulário para exclusão e alteração serão exibidos para o usuário. Nesta tag *form*, o nome da *servlet* que irá receber os dados do formulário para a alteração é *AlterarProdutoController* e o método POST foi escolhido para submeter esses dados.

Repare que a mesma estrutura de *tags* do formulário de cadastro de produto deve ser repetida nessa página de consulta. Entretanto, o método *out.print()* será utilizado para atribuir os valores do objeto *Produto* aos respectivos campos do formulário. Sendo assim, o registro de produto recuperado no banco de dados será atribuído aos campos do formulário por meio do atributo *value* da tag *input*.

No caso da tag *select,* uma comparação deverá ser feita entre o valor do objeto recuperado (atributo) e o valor das opções disponíveis no *select* a fim de se identificar qual opção deverá receber o atributo *selected.* Note que o atributo *id* do objeto *Produto* é armazenado em um

*input* do tipo *hidden*, ou seja, esse fica invisível ao usuário.

```
<%    if(request.getAttribute("produto")!= null){
      Produto produto = (Produto)request.getAttribute("produto");
 %>
   <form action="AlterarProdutoController.do" method="post" title="alterarProduto" style="margin-left: 460px;" >
   <table border="1" cellpadding="1" style="width: 500px">
       <thead>
           <tr><th colspan="2">Alteração/Exclusão de Produto
               <input type="hidden" name="id" value="<%out.print(produto.getIdProduto());%>"/> </th></tr>
       </thead>
       <tbody>
           <tr>
               <td><label>Descrição:</label></td>
               <td><input style="width: 350px;" type="text" name="descricao"
                    value="<%out.print(produto.getDescricao());%>"/></td> </tr><tr>
               <td><label>Quantidade:</label></td>
               <td><input style="width: 50px" type="text" name="qtd"
                 value="<%out.print(produto.getQuantidade());%>"/></td>
           </tr><tr>
              <td><label>Fornecedor:</label></td>
               <td><select style="width: 150px" name="fornecedor" >
                       <option <%if(produto.getFornecedor().equals("Assai"))
                       {out.print("selected=selected");}%>>Assai</option>
                       <option <%if(produto.getFornecedor().equals("kuhneheitz"))
                       {out.print("selected=selected");}%>>kuhneheitz</option>
                       <option <%if(produto.getFornecedor().equals("Nova Mix"))
                       {out.print("selected=selected");}%>>Nova Mix</option>
                       <option <%if(produto.getFornecedor().equals("Solostocks"))
                       {out.print("selected=selected");}%>>Solostocks</option>
               </select></td>
           </tr>
```

**Figura 42. Segunda parte do código da tela de consulta de produto.**

A terceira parte do código da página de consulta de produto é exibida na Figura 43. No campo *Vendas On-line* também será necessário fazer uma comparação utilizando o método *equals()* a fim de verificar qual opção deverá receber o atributo *checked*. Através de um link (imagem de uma lixeira), a *servlet ExcluirProdutoController* será acionada pelo método GET passando como parâmetro o *id* do objeto para a exclusão.

Repare que o fechamento do comando *IF* ocorre após o fechamento da tag *form* do formulário de alteração e exclusão de produto. Por fim, um *link* que direciona o usuário para a tela principal é acrescentado. O rodapé da página (página HTML) também é inserido na página de consulta de produto através da diretiva *include*.

```
                    <tr><td><label>Preço:</label></td>
                        <td><input style="width: 125px" type="text" name="preco"
                        value="<%out.print(produto.getPreco());%>"/></td>
                    </tr><tr>
                        <td><label>Vendas On-line:</label></td>
                        <td><input type="radio" name="vendas" value="sim"
                        <%if(produto.getVendasOnLine().equals("sim"))
                        {out.print("checked=checked");}%>/>
                        <label>Sim</label>
                        <input type="radio" name="vendas" value="nao"
                        <%if(produto.getVendasOnLine().equals("nao"))
                        {out.print("checked=checked");}%>/>
                        <label>Não</label></td> </tr>
                    <tr style="vertical-align: text-bottom">
                        <td colspan="2" style="text-align: center;">
                            <input type="submit" value="Alterar" />
                            <input type="reset" value="Limpar" />  
                            <a href="ExcluirProdutoController.do?id=<%out.print(produto.getIdProduto());%>">
                                <img src="Imagens/delete.png" width="25" height="30" alt="excluir"/>
                            </a>
                        </td>
                    </tr>
                </tbody>
            </table>
        </form>
        <% } %>
    <a href="index.jsp"> <img src="Imagens/seta.png" width="70" height="70" alt="seta"/></a>
    <%@include file="Layout_Pagina/rodape.html"%>
    </body>
</html>
```

**Figura 43. Terceira parte do código da tela de consulta de produto.**

A próxima etapa é a criação da *servlet BuscarProdutoController.* O parâmetro informado para o *url-pattern* foi *BuscarProdutoController.do*. Nesta *servlet*, o método *doGet()* será implementado para gerenciar a solicitação de consulta feita pelo usuário. Lembrando que o parâmetro (descrição do produto) informado pelo usuário será enviado pela URL.

O código do método *doGet()* é ilustrada na Figura 44.

```
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    String descricao = request.getParameter("descricao");
    try {
        if(descricao.length()>0 && descricao!=null){
         Produto produto = new Produto();
         Produto result = produto.buscarProdutoPorDescricao(descricao);
         request.setAttribute("produto", result);
         RequestDispatcher dispatcher = request.getRequestDispatcher("buscarProduto.jsp");
         dispatcher.forward(request, response);
     }else{
            String mensagem = "Preencha corretamente o campo nome";
            request.setAttribute("mensagem", mensagem);
            RequestDispatcher dispatcher = request.getRequestDispatcher("buscarProduto.jsp");
            dispatcher.forward(request, response);
         }
    } catch (ExceptionDAO | ServletException | IOException e) {
         response.getWriter().write("Erro ao buscar produto: " + e);
     }
   }
```

**Figura 44. Código do método *doGet()* da *servlet BuscarProdutoController*.**

O parâmetro de consulta é recuperado pela *servlet* através do método *getParameter().* Uma verificação é realizada para comprovar se o usuário preencheu o campo *descrição* corretamente. Caso contrário, uma mensagem será exibida na tela de consulta. Após a verificação, o método *buscarProdutoPorDescricao()* da classe Produto será chamado. Esse método irá retornar um único objeto do tipo *Produto* com a descrição informada. A variável *result* irá armazenar esse objeto que será posteriormente, encaminhado para a tela de consulta de produto em JSP. Se ocorrer qualquer erro durante a execução do método *doGet()*, este será capturado pela bloco do *try/catch*.

Por fim, o código do método *buscarProdutoPorDescricao()* da classe *Produto* é exibido na Figura 45 a). Esse método chama o método *buscarProdutoPorDescricao()* da classe ProdutoDAO, cujo código é mostrado na Figura 45 b). Note que todas as colunas do registro produto são recuperadas.

```
public Produto buscarProdutoPorDescricao(String descricao) throws ExceptionDAO {
    return new ProdutoDAO().buscarProdutoPorDescricao(descricao);
}
```

**a)**

```
public Produto buscarProdutoPorDescricao(String descricao) throws ExceptionDAO {
    ResultSet rs = null;
    Connection conn = null;
    PreparedStatement stmt = null;
    Produto produto = null;
    try {String sql = "select * from produto where descricao=?";
        conn = new ConexaoBancoDeDados().getConnection();
        stmt = conn.prepareStatement(sql);
        stmt.setString(1, descricao);
        rs = stmt.executeQuery();
        if (rs != null) {
            while (rs.next()) {
                produto = new Produto();
                produto.setIdProduto(rs.getInt("id"));
                produto.setDescricao(rs.getString("descricao"));
                produto.setQuantidade(rs.getInt("quantidade"));
                produto.setFornecedor(rs.getString("fornecedor"));
                produto.setPreco(rs.getFloat("preco"));
                produto.setVendasOnLine(rs.getString("vendas"));
            }}
      } catch (SQLException e) {e.printStackTrace();
        throw new ExceptionDAO("Erro ao buscar produto: " + e);
    } finally {
        try {if (rs != null) {rs.close();}
        } catch (SQLException e) {e.printStackTrace();}
        try {if (stmt != null){stmt.close();}
        } catch (SQLException e){e.printStackTrace();}
        try {if (conn != null) {conn.close();}
        } catch (Exception e) {e.printStackTrace();}
     }return produto; }
```

**b)**

**Figura 45. a) Método *buscarProdutoPorDescricao()* da classe *Produto*. Figura 45. b) Método *buscarProdutoPorDescricao()* da classe *ProdutoDAO*.**


# Programando uma *servlet* – Método excluir

## Aula 06: Programando uma *servlet* - Método excluir

Nesta aula será demonstrada a implementação da *servlet* que recebe um parâmetro da tela de consulta em JSP e invoca um método para a exclusão de um registro de produto no banco de dados. Na tela de consulta de produto, o usuário acionará a exclusão através de um *link* representado com a imagem de uma lixeira. Após a exclusão, a tela de consulta será novamente exibida para o usuário. Ainda será demonstrada a implementação do método *excluirProduto()* na classe *ProdutoDAO* e de *Produto.*

O primeiro passo para implementar a rotina de exclusão é a criação da *servlet* chamada de *ExcluirProdutoController*. O elemento *url-pattern* do arquivo *web.xml* dessa *servlet* receberá como parâmetro a denominação: *ExcluirProdutoController.do.* Lembre-se que ao criar uma *servlet,* o arquivo *web.xml* será atualizado automaticamente, de acordo com os parâmetros informados pelo usuário no momento da criação da *servlet.*

O método de exclusão só poderá ser executado após a execução de uma rotina de consulta de produto no banco de dados. Quando o usuário clicar no *link* representado pela imagem de uma lixeira no formulário para Alteração/Exclusão de produto*,* o código identificação do produto (id) será enviado para a *servlet ExcluirProdutoController*. Quando uma *servlet* é acionada através de um *link* em uma página, o método GET será automaticamente, definido. Lembre-se que os parâmetros que serão enviados pela URL são definidos após o caractere interrogação conforme ilustrado na Figura 46.

Note que o atributo *href* da *tag* âncora (a) receberá o nome da *servlet* que deverá ser chamada para a execução do método de exclusão de um produto. O nome da *servlet* deverá ser o mesmo atribuído no elemento *url-pattern* do arquivo *web.xml.*

```
<a href="ExcluirProdutoController.do?id=<%out.print(produto.getIdProduto());%>">
```

**Figura 46. Link que aciona a exclusão de um produto.**

Após a criação da servlet, o método *doGet()* deverá ser implementado conforme o código ilustrado na Figura 47.

```
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {


    String id = request.getParameter("id");

    try {
         Produto produto = new Produto();
         produto.excluirProduto(Integer.parseInt(id));
         RequestDispatcher dispatcher = request.getRequestDispatcher("buscarProduto.jsp");
         dispatcher.forward(request, response);

    } catch (NumberFormatException | ExceptionDAO | ServletException | IOException e) {
         response.getWriter().write("Erro ao excluir produto: " + e);

    }

}
```

**Figura 47. Código do método doGet() da servlet *ExcluirProdutoController*.**

Para recuperar o parâmetro enviado pela URL, o método *getParameter()* é utilizado. Este método recebe como parâmetro o nome designado no lin*k* (id). Por padrão, o método *getParameter()* retorna um valor do tipo String. Por isso, esse deverá ser convertido para o tipo inteiro devido à exigência do método a ser invocado (*excluirProduto*()). Essa *servlet* invocará o método *excluirProduto()* da classe *Produto* (pacote *model*). A página de busca de um produto será exibida após a execução do método de exclusão.

O código do método *excluirProduto()* da classe *Produto* é exibido na Figura 48. Este método invoca o método *excluirProduto()* da classe *ProdutoDAO.*

```
public void excluirProduto(int id) throws ExceptionDAO {
    new ProdutoDAO().excluirProduto(id);
}
```

**Figura 48. Código do método excluirProduto() da classe Produto.**

Por fim, o código do método *excluirProduto()* da classe *ProdutoDAO* é exibido na Figura 49. Esse método estabelece uma conexão com o banco de dados e executa um comando em SQL para excluir um registro no banco de dados de acordo com o código de identificação de um produto. Caso um erro ocorra durante a execução do método excluir, esse será capturado pelo bloco *try/catch.*

```
public void excluirProduto(int id) throws ExceptionDAO {
    String sql = "delete from produto where id=?";
    PreparedStatement stmt = null;
    Connection connection = null;
   try {connection = new ConexaoBancoDeDados().getConnection();
        stmt = connection.prepareStatement(sql);
        stmt.setInt(1, id);
        stmt.execute();
    } catch (SQLException e) {
        e.printStackTrace();
        throw new ExceptionDAO("Erro ao excluir produto: " + e);
    } finally {
        try {if (stmt != null) {stmt.close();}
        } catch (SQLException e) {e.printStackTrace();}
        try {if (connection != null) {connection.close();}
        } catch (SQLException e) {e.printStackTrace();}
    }
}
```

**Figura 49. Código do método excluirProduto() da classe ProdutoDAO.**

Quando o *link* para a exclusão de um produto for acionado na página de consulta de um produto, o código de identificação do produto que deverá ser excluído será enviado pela URL, conforme o exemplo exibido na Figura 50.


**Figura 50. URL exibindo o parâmetro.**


# Programando uma *servlet* – Método Alterar

## Aula 07: Programando uma *servlet* - Método alterar

Nesta aula será demonstrada a implementação da *servlet* que recupera os dados de um formulário e invoca um método para a alteração de um registro no banco de dados. Assim como no método de exclusão, o método de alteração de um registro de produto é executado após a realização de uma consulta. Esse método será acionado pelo botão *Alterar* e após a alteração, a tela de consulta será novamente exibida para o usuário. Ainda será demonstrada a implementação do método *alterarProduto()* na classe *ProdutoDAO* e de *Produto.*

A primeira etapa será a criação da *servlet* chamada de *AlterarProdutoController.* O elemento *url-pattern* do arquivo *web.xml* dessa *servlet* receberá como parâmetro a denominação: *AlterarProdutoController.do.* A cada *servlet* criada, o arquivo *web.xml* é atualizado possibilitando o mapeamento das *servlets* em uma aplicação *web.* A Figura 51 mostra o código do arquivo *web.xml* após a criação de todas as *servlets* do *software* do *Mercado Banana Bacana.*

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
    <servlet>
        <servlet-name>CadastrarProdutoController</servlet-name>
        <servlet-class>cefet.controller.CadastrarProdutoController</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>BuscarProdutoController</servlet-name>
        <servlet-class>cefet.controller.BuscarProdutoController</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>AlterarProdutoController</servlet-name>
        <servlet-class>cefet.controller.AlterarProdutoController</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>ExcluirProdutoController</servlet-name>
        <servlet-class>cefet.controller.ExcluirProdutoController</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>CadastrarProdutoController</servlet-name>
        <url-pattern>/CadastroProdutoController.do</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>BuscarProdutoController</servlet-name>
        <url-pattern>/BuscarProdutoController.do</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>AlterarProdutoController</servlet-name>
        <url-pattern>/AlterarProdutoController.do</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>ExcluirProdutoController</servlet-name>
        <url-pattern>/ExcluirProdutoController.do</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>30</session-timeout>
    </session-config>
</web-app>
```

**Figura 51. Código do arquivo *web.xml*.**

O código da *tag form* do formulário de alteração na tela de consulta de produto é exibido na Figura 52. Lembre-se que o botão *Alterar* deve ser do tipo *submit* e que a *servlet AlterarProdutoController* atenderá a requisição do cliente através do método *Post.*

```
<form action="AlterarProdutoController.do" method="post" title="alterarProduto">
```

**Figura 52. Código da tag form do formulário de alteração.**

O código do método *doPost()* da *servlet AlterarProdutoController* é ilustrado na Figura 53. Esse código é bem semelhante ao código do método cadastrar um produto no banco de dados. Primeiramente, os dados do formulário serão recuperados através do método *getParameter().*

```
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    String id = request.getParameter("id");
    String descricao = request.getParameter("descricao");
    String quantidade = request.getParameter("qtd");
    String fornecedor = request.getParameter("fornecedor");
    String preco = request.getParameter("preco");
    String vendas = request.getParameter("vendas");
    Produto produto = null;
    String mensagem = null;
    String[] arrayParametros = {descricao, quantidade, fornecedor, preco, vendas};
    String[] arrayCampos = {"Descricao", "Quantidade", "Fornecedor", "Preco", "Vendas On-Line"};
    try {
        for (int contador = 0; contador < arrayParametros.length; contador++) {
            if (arrayParametros[contador] == null || arrayParametros[contador].length() <= 0) {
                if (mensagem == null) {mensagem = arrayCampos[contador];
                } else {mensagem = mensagem + " , " + arrayCampos[contador];}
            }}
        if (mensagem != null) {
            mensagem = "Preencha corretamente o(s) campo(s): " + mensagem + ".";
            request.setAttribute("mensagem", mensagem);
            RequestDispatcher dispatcher = request.getRequestDispatcher("buscarProduto.jsp");
            dispatcher.forward(request, response);
        } else {
            produto = new Produto();
            produto.setIdProduto(Integer.parseInt(id));
            produto.setDescricao(descricao);
            produto.setQuantidade(Integer.parseInt(quantidade));
            produto.setFornecedor(fornecedor);
            produto.setPreco(Float.parseFloat(preco));
            produto.setVendasOnLine(vendas);

            produto.alterarProduto(produto);
            RequestDispatcher dispatcher = request.getRequestDispatcher("buscarProduto.jsp");
            dispatcher.forward(request, response);
        }

    } catch (ServletException | IOException | NumberFormatException | ExceptionDAO e) {
        response.getWriter().write("Erro ao alterar produto: " + e);
    }
}
```

**Figura 53. Código da servlet AlterarProdutoController.**

Note que o código de identificação (id) do produto é recuperado por meio do campo *input* do tipo invisível (*hidden*) localizado no formulário de alteração na tela de consulta de produto (Figura 54). Esse campo não fica visível ao usuário, entretanto, ele é necessário para guardar o valor do id do produto para alteração. Para recuperar o valor desse campo, o atributo *name* será utilizado, assim como para recuperar os demais campos desse formulário.

```
<input type="hidden" name="id" value="<%out.print(produto.getIdProduto());%>"/> </th></tr>
```

**Figura 54. Código do campo *hidden* do formulário de alteração.**

Após a recuperação dos dados, uma verificação é realizada a fim de certificar se todos os campos do formulário foram preenchidos corretamente. Caso algum campo do formulário não seja preenchido, uma mensagem será exibida na tela de consulta de produto. Para isso, o *scriptlet* na tela de consulta (Figura 55) é usado para capturar a mensagem e exibir ao usuário.

```
<% String mensagem = (String) request.getAttribute("mensagem");
    if (mensagem != null) {
        out.print(mensagem);
    }
%>
```

**Figura 55. Código do *scriptlet* para a recuperação de uma mensagem.**

Se todos os campos do formulário forem preenchidos corretamente, um objeto do tipo *Produto* será instanciado e os dados recuperados serão devidamente *settados* no objeto. Depois, o método *alterarProduto()* da classe *Produto* (classe *model)* será invocado. O código do método *alterarProduto()* é mostrado na Figura 56.

```
public void alterarProduto(Produto produto) throws ExceptionDAO {
    new ProdutoDAO().alterarProduto(produto);
}
```

**Figura 56. Código do método *alterarProduto()* da classe *Produto*.**

Esse método irá chamar o método *alterarProduto()* da classe *ProdutoDAO.* O código desse método é mostrado na Figura 57.

```
public void alterarProduto(Produto produto) throws ExceptionDAO {
    String sql = "update produto set descricao=?,quantidade=?,fornecedor=?, preco=?,vendas=? where id=?";
    PreparedStatement stmt = null;
    Connection connection = null;
    try {
        connection = new ConexaoBancoDeDados().getConnection();
        stmt = connection.prepareStatement(sql);
        stmt.setString(1, produto.getDescricao());
        stmt.setInt(2, produto.getQuantidade());
        stmt.setString(3, produto.getFornecedor());
        stmt.setFloat(4, produto.getPreco());
        stmt.setString(5, produto.getVendasOnLine());
        stmt.setInt(6, produto.getIdProduto());
        stmt.execute();
    } catch (SQLException e) {e.printStackTrace();
        throw new ExceptionDAO("Erro ao alterar produto: " + e);
    } finally {
        try {if (stmt != null) {stmt.close();}
        } catch (SQLException e) {e.printStackTrace();}
        try {if (connection != null) {connection.close();}
        } catch (SQLException e) {e.printStackTrace();}
    }
}
```

**Figura 57. Código do método *alterarProduto()* da classe *ProdutoDAO*.**

Esse método estabelece uma conexão com o banco de dados e executa um comando em SQL que realiza a alteração de um registro de produto. Se algum erro ocorrer durante a execução desse método, este será capturado pelo bloco *try/catch.* Por fim, a estrutura do projeto *web* desenvolvido no NetBeans é exibido na Figura 58.

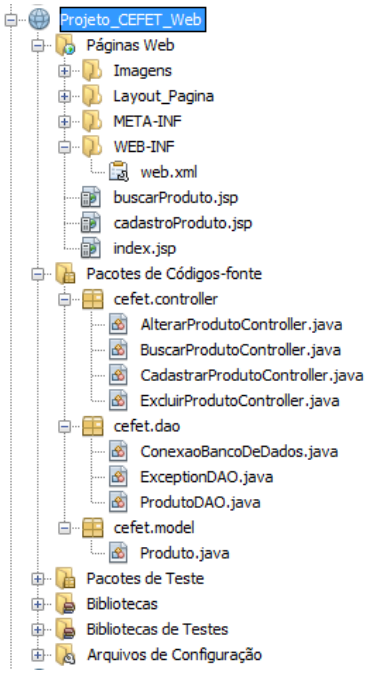



**Figura 58. Estrutura do projeto *web***

## Referências Bibliográficas

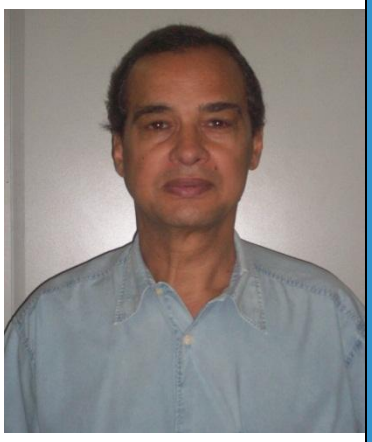
Apache TomCat. Disponível em << http://www.apache.org/ >>. Acessado em 10 de abril de 2016.

DEITEL, Harvey M; DEITEL, Paul J. Java: Como programar. 8ª Edição. 2010 - São Paulo: Prentice Hall.

NetBeans IDE. Disponível em: << https://netbeans.org/ >>. Acessado em 10 de abril de 2016.

BASHAM, B.; BATES, B. e SIERRA, K.; Use a Cabeça! Servlets & JSP. 2ª edição. 2008 – Rio de Janeiro: Editora Alta Books.

**COM A PALAVRA, O COORDENADOR GERAL ...**
Professor D. Sc. Mauro Godinho Gonçalves



A implementação de curso a distância sugere como requisito de desenvolvimento o conhecimento da tecnologia digital e das ferramentas que possibilitam a interação entre seus interlocutores.

Nesta obra sobre educação tecnológica, são discutidas as diversas possibilidades que a educação a distância oferece e apresentada a importância do mundo digital para sua implementação. Discute-se, também, as tecnologias da informação e comunicação e as várias redes sociais muito utilizadas na comunicação entre as pessoas atualmente.

O autor deste trabalho é professor com reconhecida competência nesta área, lecionando no ensino técnico, superior e pós. Assim, espero que apreciem o seu trabalho.