08

Criando nosso primeiro formulário

Transcrição

Vamos criar mais um HTML para o cadastro. Começamos criando uma pasta para separar os arquivos, `New > Folder` e nomeamos de `livro`. Dentro dessa pasta, vamos criar um HTML `New > HTML` e vamos nomea-lo de `form.html`. Clicamos em Next, e procuramos uma opção chamada **New Facelet Composition Page**. Facelets é o framework que cuida dos templates do JSF, e clique em `Finish`. Vamos deixar apenas a estrutura abaixo:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://xmlns.jcp.org/jsf/html"
    xmlns:f="http://xmlns.jcp.org/jsf/core">

</html>
```

Nesse curso, não estudaremos as coisas básicas do JSF, iremos detalhar apenas o que é um mais avançado ou um pouco diferente do padrão do JSF.

Vamos começar criando o form, e dentro dele colocamos um `Label` e um `InputText` para a entrada do texto, conforme abaixo:

```
<!-- Mantém o código de abertura do html acima -->
    <h:form>
        <div>
            <h:outputLabel value="" />
            <h:inputText />
        </div>
    </h:form>
<!-- Fecha o html abaixo -->
```

Alguns campos que são importantes quando tratamos de cadastro de livro, são eles: Título, Descrição, Número de Páginas e o Preço.

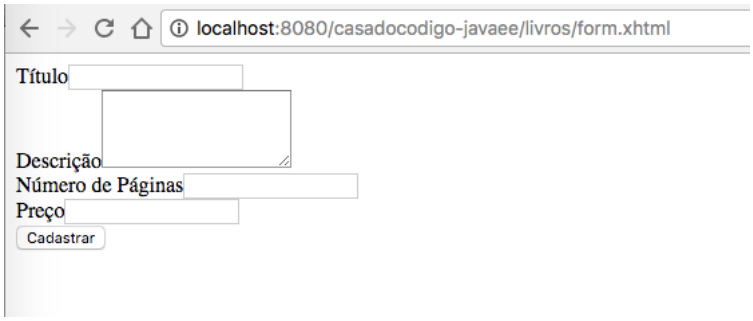
Vamos duplicar os dados, selecionando o que queremos copiar, e pressionando (Alt + Ctrl + Up / Down). Se no seu sistema operacional não funcionar, basta copiar e colar os campos.

Nomeamos cada um dos `<h:outputLabel />` com os quatro campos que citamos acima. E ao final de cada formulário, sempre temos um botão de confirmação. Assim, vamos colocar um botão com `<h:commandoButton />` e colocar o value dele para *Cadastrar* conforme abaixo:

```
        <div>
            <h:outputLabel value="Título" />
            <h:inputText />
        </div>
        <div>
```

```
            <h:outputLabel value="Descrição"/>
            <h:inputTextarea />
        </div>
        <div>
            <h:outputLabel value="Número de Páginas"/>
            <h:inputText />
        </div>
        <div>
            <h:outputLabel value="Preço"/>
            <h:inputText  />
        </div>
        <h:commandButton value="Cadastrar" />
```

Além disso, precisamos renomear nossa página para o padrão do JSF, assim, feche o arquivo `form.html` e renomei-o para `form.xhtml` . Como o XHTML já é reconhecido pelo servidor como arquivo JSF, já podemos subir o servidor e realizar o primeiro teste. Quando acessamos `http://localhost:8080/casadocodigo/livro/form.xhtml` temos o seguinte resultado.



Toda tela jsf, em geral possui uma classe Java por trás que chamamos de BackBean. Por isso, vamos criar uma classe, clicando em cima da pasta *src/main/java* pressione `Ctrl + N` , escolha a opção `Class` e dê o nome de `AdminLivrosBean` . Vamos colocá-la no pacote `br.com.casadocodigo.loja.beans` .

Dentro dessa classe, criaremos o método salvar da seguinte forma:

```
public class AdminLivrosBean {

    private Livro livro;

    public void salvar() {
        System.out.println("Livro salvo com Sucesso!");
    }

}
```

Os dados do livro no JSF são associados diretamente aos atributos do bean, assim vamos criar um atributo em nossa classe, como sendo `private Livro livro;` . Nesse ponto, recebemos um erro, pois a classe Livro ainda não existe. Vamos criá-la conforme abaixo, e já gerar os *getters* e *setters* da classe com o atalho `Ctrl + 3 > ggas` .

```
public class Livro {

    private String titulo;
    private String descricao;
    private BigDecimal preco;
```

```
    private Integer numeroPaginas;

    // getters e setters aqui!

    @Override
    public String toString() {
        return "Livro [titulo=" + titulo + ", descricao=" + descricao + ", preco=" + preco + ",
                + numeroPaginas + "]";
    }

}
```

Já criamos também o `toString()` do Livro, com o atalho `Ctrl + 3` e na caixa de busca, digite `toString`. Selecionamos a opção `Generate toString()` e clicamos em selecionar todos os campos, logo depois em Finish.

Vá para a classe `AdminLivrosBean`, e também crie os getters e setters do Livro, com o atalho `Ctrl + 3 > ggas` como já estamos acostumados.

Uma vez que temos a classe `Livro` e também `AdminLivrosBean`, vamos relacioná-los em nossa tela pelos atributos `value` do xhtml utilizando a *ExpressionLanguage* do JSF.

```
<div>
    <h:outputLabel value="Título" />
    <h:inputText value="#{adminLivrosBean.livro.titulo}" /> <!-- NOVIDADE NO VALUE AQUI -->
</div>
<!-- REPITA O PROCESSO PARA OS DEMAIS INPUTS -->
```

Para vermos os resultados na tela, basta ir na view *Servers* abrir na setinha do servidor para que possamos ver nosso projeto. Clicamos em cima do projeto com o botão direito vamos na opção `Full Publish` que reenviará os nossos arquivos novos para o servidor. Assim, podemos testá-los na tela.

Ao acessar a aplicação no navegador, recebemos o erro: `Target Unreacheable, identifier 'adminLivrosBean' resolved to null`. Pois o JSF ainda não exerga nosso Bean. Para que o JSF veja nosso Bean, temos que utilizar o **CDI**. Que veio no *JavaEE 6* e ficou mais poderoso no *JavaEE 7*.

Assim, para que o CDI libere nosso Bean para a tela, utilizamos a annotation `@Named` em cima do Bean. Agora o JSF conseguirá ver nosso Bean, com o nome *default* `adminLivrosBean`, ou seja, apenas a primeira letra ficou minúscula. Vamos realizar o Full Publish novamente e poderemos testar.

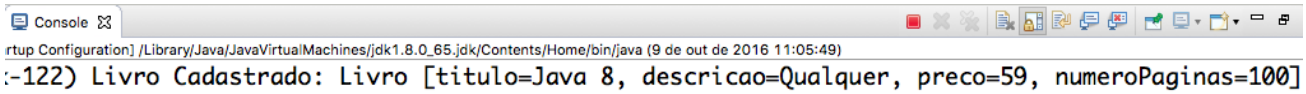
Ao preencher o formulário e tentar cadastrar, recebemos o seguinte erro: `... value="#{adminLivrosBean.livro.titulo}": Target Unreacheable, identifier 'adminLivrosBean' resolved to null`. O que aconteceu agora é que, ao tentar pegar o título, o livro veio nulo. Por isso, vamos fazer `private Livro livro = new Livro()` lá no nosso Bean, e dessa forma evitamos receber esse null. Realizamos `Full Publish` novamente, e ao preencher os dados e clicar em *Cadastrar*, dessa vez os dados sumiram e não obtivemos erros.

Isso ocorreu pois nosso Bean agora está sendo gerenciado pelo CDI, e o CDI por default possui um tempo de vida muito curto, no caso, sempre que o JSF precisa do Bean, ele cria uma nova instância e entrega ao JSF. Porém, para que os dados fiquem vivos durante toda a requisição do usuário, precisamos que o CDI deixe ele vivo durante todo o Request, para isso, usamos a annotation `@RequestScoped`. Fique atento para utilizar o `@RequestScoped` correto, do pacote do CDI, que é `javax.enterprise.context.RequestScoped`. Ao realizar o Full Publish novamente, podemos testar o cadastro.

Dessa vez, os valores permaneceram, porém não obtivemos nenhum resultado no console, como era esperado. Pois precisamos "ligar" o botão Cadastrar com o método salvar do nosso Bean. Fazemos isso com o atributo `action` do `commandButton`, ficando assim:

```
<h:commandButton value="Cadastrar" action="#{adminLivrosBean.salvar}" />
```

Finalmente, realizamos *Full Publish* e ao preencher o form e clicar em Cadastrar, vemos no Console do Eclipse o resultado esperado.



Fizemos uso do CDI e começamos com algo simples, mas vamos fazer de fato nosso cadastro funcionar.