04

Mãos na massa: Formatando datas

Chegou a hora de você pôr em prática o que foi visto na aula. Para isso, execute os passos listados abaixo.

1) Agora que o livro terá data de lançamento, na classe `Produto`, crie o atributo `dataLancamento`, do tipo `Calendar` e adicione a anotação `@DateTimeFormat`, para que o Spring consiga converter os valores de texto para `Calendar`. Além disso, gere o *getter* e *setter* desse atributo:

```
@Entity
public class Produto {

    @DateTimeFormat
    private Calendar dataLancamento;

    public Calendar getDataLancamento() {
        return dataLancamento;
    }

    public void setDataLancamento(Calendar dataLancamento) {
        this.dataLancamento = dataLancamento;
    }

    // restante do código omitido

}
```

2) No formulário **form.jsp**, logo após a `div` das páginas, crie mais uma `div`:

```
<div>
    <label>Data de Lançamento</label>
    <input name="dataLancamento" type="text" />
    <form:errors path="dataLancamento" />
</div>
```

3) Para não ter que configurar o formato da data por anotações, configure-o através do `AppWebConfiguration`, criando o método abaixo:

```
@Bean
public FormattingConversionService mvcConversionService() {
    DefaultFormattingConversionService conversionService =
        new DefaultFormattingConversionService();
    DateFormatterRegistrar registrar = new DateFormatterRegistrar();
    registrar.setFormatter(new DateFormatter("dd/MM/yyyy"));
    registrar.registerFormatters(conversionService);

    return conversionService;
}
```

4) E para não perder as informações se um campo estiver inválido, troque as *tags* do formulário no form.jsp de `<input type="text" name="nome" />` para `<form:input path="nome" />`, por exemplo. Ele ficará com a seguinte estrutura:

```
<form:form action="${s:mvcUrl('PC#grava').build() }" method="post"
    commandName="produto">

    <div>
        <label>Título</label>
        <form:input path="titulo" />
        <form:errors path="titulo" />
    </div>
    <div>
        <label>Descrição</label>
        <form:textarea path="descricao" rows="10" cols="20"/>
        <form:errors path="descricao" />
    </div>name
    <div>
        <label>Páginas</label>
        <form:input path="paginas" />
        <form:errors path="paginas" />
    </div>
    <div>
        <label>Data de lançamento</label>
        <form:input path="dataLancamento" />
        <form:errors path="dataLancamento" />
    </div>
    <c:forEach items="${tipos}" var="tipoPreco" varStatus="status">
        <div>
            <label>${tipoPreco}</label>
            <form:input path="precos[${status.index}].valor" />
            <form:hidden path="precos[${status.index}].tipo"
                value="${tipoPreco}" />
        </div>
    </c:forEach>
    <button type="submit">Cadastrar</button>
</form:form>
```

5) Para isso funcionar, faça o método `form()` do `ProdutoController` receber um `Produto`:

```
@RequestMapping("form")
public ModelAndView form(Produto produto) {
    ModelAndView modelAndView = new ModelAndView("produtos/form");
    modelAndView.addObject("tipos", TipoPreco.values());

    return modelAndView;
}
```

6) O método `gravar`, também do `ProdutoController` irá apresentar um erro, pois dentro dele há uma chamada para o método `form`. Então passe um `Produto` para ele:

```
@RequestMapping(method=RequestMethod.POST)
public ModelAndView grava(@Valid Produto produto, BindingResult result,
        RedirectAttributes redirectAttributes) {
```

```
        if(result.hasErrors()) {
            return form(produto);
        }

        produtoDao.gravar(produto);

        redirectAttributes.addFlashAttribute("sucesso", "Produto cadastrado com sucesso!");

        return new ModelAndView("redirect:produtos");
    }
```