Carrinho de Compras

Se entrarmos no site da Casa do Código, selecionarmos um livro e clicar em comprar, veremos que o livro é adicionado ao carrinho de compras. Veremos também que o carrinho de compras é uma lista com todos os livros selecionados. Esta lista exibe os preços de cada livro, a quantidade e o total da soma dos preços.

Faremos o mesmo em nosso projeto. Crie uma nova pasta chamada `carrinho` no diretório `webapp/WEB-INF/views/`. Baixe os arquivos que serão usados nessa aula através deste link: https://s3.amazonaws.com/caelum-online-public/spring-mvc-1-criando-aplicacoes-web/carrinho-com-imagem.zip. Descompacte o zip baixado e extrai-o. Copie o arquivo `itens.jsp` que extraiu de dentro do zip e cole-o na pasta `carrinho` do seu projeto.

O primeiro ajuste que faremos é fazer com que a página de `itens.jsp` exiba a quantidade de produtos no carrinho do mesmo jeito que fizemos na página de detalhes. Encontre a seguinte linha de código no arquivo `itens.jsp`:

```
<li><a href="/cart" rel="nofollow">Carrinho</a></li>
```

E modifique para:

```
<li><a href="${s:mvcUrl('CCC#itens').build()}" rel="nofollow">Carrinho
```

Adicione o taglib "tags" em itens.jsp:

```
<%@ taglib uri="http://www.springframework.org/tags" prefix="s" %>
```

Observação: Note que estamos criando uma nova url. Esta url criada aponta para o método `itens` na classe `CarrinhoComprasController` que ainda não existe, mas que criaremos em instantes. Este método será o responsável por exibir esta página.

O próximo passo será a criação da tabela que exibirá nossa lista de produtos. Procure as seguintes linhas no HTML da página `itens.jsp`:

```
<body>
    <tr>
        <td class="cart-img-col">
            <img src="http://cdn.shopify.com/s/files/1/0155/7645/produc
        </td>
        <td class="item-title">TÍTULO DO LIVRO AQUI</td>
        <td class="numeric-cell">R$ 59,90</td>
        <td class="quantity-input-cell">
            <input type="number" min="0" readonly="readonly" id="update
        </td>
        <td class="numeric-cell">R$ 59,90</td>
        <td class="remove-item">
            <a href="/cart/change?218748921802387812&quantity=0">
                excluir
            </a>
        </td>
    </tr>
</body>
```

Perceba que pela estrutura da tabela, cada produto será representado por uma linha. Sendo assim, envolveremos a tag `tr` com a tag `forEach` da JSTL percorrendo a lista de itens presente em nosso carrinho. Até aqui teremos o seguinte código:

```
<c:forEach items="${carrinhoCompras.itens }" var="item">
    <tr>
        <td class="cart-img-col">
            <img src="http://cdn.shopify.com/s/files/1/0155/7645/produc
        </td>
        <td class="item-title">TÍTULO DO LIVRO AQUI</td>
        <td class="numeric-cell">R$ 59,90</td>
        <td class="quantity-input-cell">
            <input type="number" min="0" readonly="readonly" id="update
        </td>
        <td class="numeric-cell">R$ 59,90</td>
        <td class="remove-item">
            <a href="/cart/change?218748921802387812&quantity=0">
                excluir
            </a>
        </td>
    </tr>
</c:forEach>
```

Antes de continuarmos, vamos implementar um novo método na classe `CarrinhoCompras`. Veja que apesar de estarmos fazendo `${carrinhoCompras.itens }` a classe `CarrinhoCompras` não tem esse método. Outra observação a ser feita é que o carrinho de compras guarda um objeto do tipo `Map` e nesse ponto da aplicação, não queremos um `Map` mas sim uma lista. Veja o método a seguir:

```
public Collection<CarrinhoItem> getItens() {
    return itens.keySet();
}
```

O método anterior deve ser colocado na classe `CarrinhoCompras`. Perceba que ele retornar uma `Collection` que funciona como uma lista. Note também que o retorno do método captura as chaves do `Map` e as retorna. Se você se lembrar, vai perceber que as chaves desse `Map` são objetos da classe `CarrinhoItem` que possuem as informações sobre os produtos adicionados ao carrinho.

Agora, exibiremos as informações referentes aos produtos do carrinho de compras no HTML da página `itens.jsp`. Na linha onde há:

```
<td class="item-title">TÍTULO DO LIVRO AQUI</td>
```

Substitua por:

```
<td class="item-title">${item.produto.titulo}</td>
```

Apesar da linha anterior funcionar para o título do produto, não funcionaria para o preço. O preço dentro da classe produto está sendo representado por uma lista e não queremos uma lista de preços neste ponto da aplicação. Queremos simplesmente o preço selecionado pelo usuário, um único preço. Por hora faremos apenas `item.preco` e depois implementaremos este método para que funcione. Sendo assim onde há:

```
<td class="numeric-cell">R$ 59,90</td>
```

Substitua por:

```
<td class="numeric-cell">${item.preco}</td>
```

Agora exibiremos a quantidade de cada item adicionado no carrinho de compras. Encontre e observe a seguinte linha do HTML:

```
<input type="number" min="0" readonly="readonly" id="updates_4082273665
```

O atributo `id` e `name` estão com alguns números malucos. Estes números são usados no site da Casa do Código. Não precisamos destes números. Modificaremos estes atributos para ter o texto `quantidade` e no valor usaremos o método `getQuantidade` da classe `CarrinhoCompras` que recebe um `item` e retorna a quantidade daquele item no carrinho. A linha anterior modificada fica assim:

```
<input type="number" min="0" readonly="readonly" id="quantidade" name="
```

A seguinte linha diz respeito ao preço novamente, mas desta vez não é o valor unitário do produto. É o total da soma dos preços daquele produto específico. Criaremos um método pra isso em instantes, mas já vamos deixar o HTML pronto por hora. Onde esta assim:

```
<td class="numeric-cell">R$ 59,90</td>
```

Deixe assim:

```
<td class="numeric-cell">${carrinhoCompras.getTotal(item)}</td>
```

O método `getTotal` que usamos neste HTML não existe ainda. Iremos implementá-lo em instantes. O próximo passo será modificar o link de remoção de produtos do carrinho. Atualmente ele se encontra dessa forma:

```
<td>
    <a href="/cart/change?218748921802387812&quantity=0">
        <img src="/excluir.png" alt="Excluir" title="Excluir" />
    </a>
</td>
```

Em vez de um simples link para remoção, usaremos um formulário com o atributo `method` configurado como `post`. Substituiremos o link da tag `a` e a tag `img` pela tag `input` configurando o atributo `type` com o valor `image`. As modificações propostas ficam assim:

```
<td>
    <form action="" method="post">
        <input type="image" src="/excluir.png" alt="Excluir" title="Exc
    </form>
</td>
```

Por último e não menos importante temos o rodapé desta tabela que lista nossos produtos no carrinho. Ela contém o seguinte código:

```
<tfoot>
    <tr>
        <td colspan="3">
            <input type="submit" class="checkout" name="checkout" value
        </td>
        <td class="quantity-input-cell">
            <input type="submit" class="update-cart" disabled="disabled
        </td>
        <td class="numeric-cell">R$ 59,90</td>
        <td></td>
    </tr>
</tfoot>
```

Só precisaremos modificar as duas `td`s centrais. Remover a que tem o `name=update` por se tratar de algo especifico da Casa do Código e fazer com que a que tem o valor `R$ 59,90` passe a exibir o total calculado pelo carrinho de compras. Esta parte do código modificada deve ficar parecida com o código abaixo:

```
<tfoot>
    <tr>
        <td colspan="3">
            <input type="submit" class="checkout" name="checkout" value
        </td>
        <td class="numeric-cell">${carrinhoCompras.total}</td>
        <td></td>
    </tr>
</tfoot>
```

Na classe `CarrinhoItem`, precisaremos criar todos os métodos que usamos na tabela de lista para que a tabela possa funcionar corretamente o mostrar a lista dos produtos no carrinho. Começaremos pelo método que retorna o preço de um produto específico. O método será chamado de `getPreco` e retornará um objeto do tipo `BigDecimal` e este método simplesmente usará o objeto produto para retornar o preço escolhido pelo usuário através do objeto `tipoPreco`, veja o código:

```
public BigDecimal getPreco(){
    return produto.precoPara(tipoPreco);
}
```

O método `precoPara` não existe na classe `Produto`. Crie-o. Neste método, precisaremos filtrar de todos os preços, o preco escolhido e logo após encontra-lo retornar seu valor através do método `getvalor()`. Observe o código abaixo:

```
public BigDecimal precoPara(TipoPreco tipoPreco) {
    return precos.stream().filter(preco -> preco.getTipo().equals(tipoP
}
```

Outro método que precisamos implementar é o que soma o total de um produto específico e recebe um `CarrinhoItem`. O método fará o seguinte cálculo: Quantidade de vezes que aquele item foi adicionado no carrinho vezes o seu preço. Na classe `CarrinhoCompras` teremos então um método chamado `getTotal` que recebe um `CarrinhoItem` e chama o método `getTotal` daquele item. Passando para este método a quantidade de vezes que o produto foi adicionado ao carrinho. Até aqui teremos:

```
public BigDecimal getTotal(CarrinhoItem item){
    return item.getTotal(getQuantidade(item));
}
```

E na classe `CarrinhoItem` teremos:

```
public BigDecimal getTotal(int quantidade) {
    return this.getPreco().multiply(new BigDecimal(quantidade));
}
```

Note como é simples, estamos recuperando o preço do produto e usando o método `multiply` da classe `BigDecimal` e passando para este método um novo `BigDecimal` pois o valor recebido por parâmetro é do tipo `int`.

Agora precisamos mostrar o valor total do carrinho, até aqui só calculamos o total de um produto específico. Para calcular o total do carrinho, só precisamos fazer um laço na classe `CarrinhoCompras` que a cada iteração chame o método `getTotal` criado anteriormente. Passando como parâmetro o produto atual e guardando o resultado de cada iteração em uma variável. Este método também se chamará `getTotal`, mas não receberá parâmetros. Vejamos no código como fica:

```
public BigDecimal getTotal(){
    BigDecimal total = BigDecimal.ZERO;
    for (CarrinhoItem item : itens.keySet()) {
        total = total.add(getTotal(item));
    }
    return total;
}
```

Para podermos testar, vamos ajustar outros dois detalhes que são: O `CarrinhoComprasController` não tem nenhum método mapeando a página de itens do carrinho e vamos fazer com que ao adicionar um produto, o usuário seja redirecionado para a página de itens do carrinho.

Para o primeiro ponto, precisaremos de um novo método. Chamaremos este método de `itens` e o mapeamento será o mesmo do controller que é `/carrinho`. Neste caso especificaremos apenas o método `HTTP` que será `GET`. Este método apenas cria um objeto `ModelAndView` direcionando a requisição para a página `itens.jsp`. Vejamos em código:

```
@RequestMapping(method=RequestMethod.GET)
public ModelAndView itens(){
    return new ModelAndView("/carrinho/itens");
}
```

Observação: Lembre-se que esta adição e a próxima modificação é para ser feita na classe `CarrinhoComprasController`.

E para o ajuste do segundo ponto, precisamos apenas alterar o método `add` para que em vez de `redirect:/produtos` este faça `redirect:/carrinho`.

```
@RequestMapping("/add")
public ModelAndView add(Integer produtoId, TipoPreco tipo){
    ModelAndView modelAndView = new ModelAndView("redirect:/carrinho");
    CarrinhoItem carrinhoItem = criaItem(produtoId, tipo);
    carrinho.add(carrinhoItem);
    return modelAndView;
}
```

Se iniciarmos o servidor agora, teremos um erro do tipo `NotSerializableException` informando que o objeto da classe `CarrinhoCompras` não é serializável. Isso acontece, porque o servidor ao verificar que um objeto está em escopo de sessão, ele tenta salvar este objeto em arquivo. Para que ele possa sempre salvar a sessão e recuperar este objeto. Para resolvermos isto, basta fazer com que a classe `CarrinhoCompras` implemente a interface `Serializable`.

```
@Component
@Scope(value=WebApplicationContext.SCOPE_SESSION)
public class CarrinhoCompras implements Serializable{

    private static final long serialVersionUID = 1L;

}
```

Perceba que a interface `Serializable` não adiciona nenhum método, mas sim um atributo estático. Agora podemos testar! Faça todos os passos de uma compra normal, visite a lista de produtos, adicione alguns ao carrinho de compras e clique em `Finalizar Compra`. Verifique se tudo está funcionando corretamente e prossiga para a próxima aula, onde continuaremos a desenvolver nosso carrinho de compras.