03

Criando serviço de pagamento

Transcrição

Vamos melhorar o nosso código do método `finalizar()`, ele neste momento está verboso, essa era uma característica das APIs do java EE, mas o pessoal tem percebido que dá para se pensar em *APIs fluentes* com encadeamento de métodos. Do `target()` podemos pedir o `request()` que dele, por sua vez, podemos fazer o `post()`. Tudo isso sendo atribuído ao `response`. Sempre tomando cuidado, claro, com a legibilidade do código. No fim teremos algo assim:

```
public void finalizar() {

    //...

    Pagamento pagamento = new Pagamento(getTotal());
    String target = "http://book-payment.herokuapp.com/payment";
    Client client = ClientBuilder.newClient();
    String response = client.target(target).request().post(Entity.json(pagamento), String.class

    System.out.println(response);

}
```

Ainda não temos código específico para a finalização do Carrinho. O que fizemos acima não é uma lógica de negócio. Podemos, então, fazer o encapsulamento desse código extraindo um método `pagar()`:

```
public void finalizar(Usuario usuario) {

    //...

    String response = pagar();

    System.out.println(response);
}

private String pagar() {

    //todo o código
}
```

Porém, o simples fato de termos um método novo fará com que as coisas funcionem. Criaremos uma Classe que será exclusiva para esse tipo de serviço. Chamaremos de "PagamentoGateway" no pacote "service":

```
package br.com.casadocodigo.loja.service;

public class PagamentoGateway {

}
```

Retiramos o método `pagar()` do CarrinhoCompra e o colocamos nessa nova Classe. Ele não pode ser um método privado e receberá o BigDecimal com valor `total` tendo como retorno o `response`:

```
public class PagamentoGateway {

    public String pagar(BigDecimal total) {
        Pagamento pagamento = new Pagamento(total);
        //código
        return response;
    }
}
```

Mas podemos pegar o `return` direto da chamada do objeto:

```
public class PagamentoGateway {

    public String pagar(BigDecimal total) {
        Pagamento pagamento = new Pagamento(total);
        //código
        return client.target(target).request().post(Entity.json(pagamento), String.class);
    }
}
```

Vamos colocar o serviço de pagamento para ser injetado no CarrinhoCompra:

```
@Inject
private PagamentoGateway pagamentoGateway
```

E dentro do método `finalizar()`:

```
public void finalizar(Usuario usuario) {
    //código

    String response = pagamentoGateway.pagar(getTotal());
}
```

Não podemos nos esquecer de indicar o serialize na Classe:

```
public class PagamentoGateway implements Serializable {

    private static final long serialVersionUID = 1L;

    //...
}
```

Perceba que agora a lógica do nosso negócio ficou bem mais simplificada. Tudo deve funcionar normalmente, pois apenas criamos uma nova Classe com a responsabilidade do pagamento e separamos os códigos de serviço.