05

Criando uma classe de validação e utilizando objetos opcionais

Vamos fazer uma refatoração para adequar nosso código a algumas boas práticas: ao invés de deixarmos a lógica de validação no controller, vamos extraí-la para uma classe apropriada, desacoplando o código.

```
package validadores;
public class ValidadorDeProduto {
    @Inject private ProdutoDAO produtoDAO;
    public boolean temErros(Form<Produto> formulario) {
        Produto produto = formulario.get();
        if (produto.getPreco() < 0.0) {
            formulario.reject(new ValidationError("preco", "Preço tem que ser maior ou igual a :
        }
        if (produtoDAO.comCodigo(produto.getCodigo()) != null) {
            formulario.reject(new ValidationError("codigo", "Já existe um produto com este códi
        }
        return formulario.hasErrors();
    }
}
```

Com o validador pronto, basta injetá-lo no controller e substituir a lógica anterior por uma chamada ao método `temErros()`.

```
@Inject private ValidadorDeProduto validadorDeProduto;
public Result salvaNovoProduto() {
    Form<Produto> formulario = formularios.form(Produto.class).bindFromRequest();
    if (validadorDeProduto.temErros(formulario)) {
        return badRequest(...);
    }
    // ...
}
```

Agora que extraimos nosso validador, podemos adequar nosso código a outra boa prática já que estamos usando Java8: os objetos opcionais, ou *Optional*. Ao invés de retornar um **Produto** ou objeto nulo, o *DAO* agora retorna um *possível produto*.

```
public Optional<Produto> comCodigo(String codigo) {
    Produto produto = produtos
        .where()
        .eq("codigo", codigo)
        .findUnique();
    return Optional.ofNullable(produto);
}
```

E portanto não comparamos mais com nulo, e sim perguntamos ao possível **Produto** se ele existe!

```
public boolean temErros(Form<Produto> formulario) {
    // ...
```

```
        if (produtoDAO.comCodigo(produto.getCodigo()).isPresent()) { ... }
        // ...
    }
```