01

Stereotypes

Transcrição

Na última aula vimos como implementar a autorização utilizando um recurso do CDI, que são os *observers*, que é uma implementação do [padrão de projeto observer (https://en.wikipedia.org/wiki/Observer_pattern)](https://en.wikipedia.org/wiki/Observer_pattern).

Vamos agora dar uma olhada nas nossas classes de `bean`s, no pacote `br.com.alura.livraria.bean`. Se você observar algumas classes como `AutorBean` e `LoginBean`, verá que estas têm algo em comum: ambas têm as anotações `@Named` e `@RequestScoped`. Já se olharmos o `LivroBean` e o `VendasBean`, temos as anotações `@Named` e `@ViewScoped`. Por fim, o `TemaBean` possui as anotações `@Named` e `@SessionScoped`.

Estamos repetindo o código em nossos `bean`s e isso não é bom, porque sempre que for necessário realizar uma alteração, precisaremos alterar em vários lugares do sistema. Por isso, o CDI tem uma anotação que representa o `@Named` e o `@RequestScoped`. Se pensarmos na responsabilidade da classe `AutorBean` dentro do padrão MVC, a classe representa a regra de negócio, então, se encaixará bem na camada de *model*. Por esse motivo a anotação é `@Model`:

```
@Model
public class AutorBean implements Serializable {
```

Ao olhar o código da anotação `@Model`, é possível perceber que ela agrupa outras anotações:

```
@Named
@RequestScoped
@Documented
@Stereotype // esteriótipo
@Target({ TYPE, METHOD, FIELD })
@Retention(RUNTIME)
public @interface Model {
}
```

Dentro da anotação, existe as anotações `@Named` e `@RequestScoped`. Para agrupar anotações, existe um recurso do CDI chamado **estereótipo**, e para isto existe a anotação `@Stereotype`. Então vamos substituir as anotações do `AutorBean` e do `LoginBean` pela anotação `@Model`:

```
@Model
public class AutorBean implements Serializable {
```

```
@Model
public class LoginBean implements Serializable {
```

Agora seria interessante fazer o mesmo para o `LivroBean`. Algo como uma anotação `@ViewModel`, por exemplo. Mas não temos isso disponível. Porém nada nos impede de criar. Na biblioteca, dentro de `br.com.alura.alura_lib.jsf.annotation`, vamos criar a anotação `@ViewModel`.

A anotação será um `@Stereotype` e agrupará a anotações `@Named` e `@ViewScoped`.

```
@Stereotype
@Named
@ViewScoped
@Target({ ElementType.TYPE, ElementType.METHOD, ElementType.FIELD })
@Retention(RetentionPolicy.RUNTIME)
public @interface ViewModel {

}
```

Agora vamos instalar a biblioteca no repositório local e em seguida atualizar o projeto `livraria`.

Na classe `LivroBean`, vamos substituir as anotação `@Named` e `@ViewScoped` por `@ViewModel`:

```
@ViewModel
public class LivroBean implements Serializable {
```

O mesmo para o `VendasBean`:

```
@ViewModel
public class VendasBean implements Serializable{
```

Ao subir o Tomcat, tudo continua funcionando.

Falta agora criarmos o `@SessionModel`, para agrupar as anotações `@Named` e `@SessionScoped`, utilizadas na classe `TemaBean`:

```
@Named
@SessionScoped
public class TemaBean implements Serializable {
```

Na biblioteca, no pacote `br.com.alura.alura_lib.jsf.annotation`, vamos criar a anotação `@SessionModel`. A única diferença em relação à `@ViewModel` é que aqui utilizamos `@SessionScoped` (do pacote `javax.enterprise.context`) no lugar de `@ViewScoped`:

```
@Stereotype
@Named
@SessionScoped
@Target({ ElementType.TYPE, ElementType.METHOD, ElementType.FIELD })
@Retention(RetentionPolicy.RUNTIME)
public @interface SessionModel {

}
```

Vamos instalar a biblioteca no repositório local, novamente, e atualizar o projeto `livraria`. Em seguida, é possível utilizar a anotação na classe `TemaBean`:

```
@SessionModel
public class TemaBean implements Serializable {
```

Ao testar e verificar se as informações são mantidas durante a sessão, alterando o tema, é possível verificar que tudo continua funcionando. Mais adiante veremos como resolver os débitos técnicos que ficaram no `alura-lib`.