02

Alternativos

Transcrição

Vimos que é possível sobrescrever um comportamento utilizando o `@Specializes`. Ele é bastante útil quando temos uma herança. Mas criamos uma *interface*, e seria possível simplesmente implementar a *interface*.

Vamos substituir a herança pela implementação da *interface*. Agora que o `EntityManager` não é mais herdado, vamos receber por injeção de dependências.

A anotação `@Specializes` foi removida, pois não estamos mais utilizando herança:

```
public class MeuGerenciadorDeTransacao implements Transacionado {

    private static final long serialVersionUID = -8590951365580768798L;

    @Inject
    private EntityManager em;

    @Override
    public Object executaComtransacao(InvocationContext context) {
        // código do método
    }
}
```

Da forma como está agora, temos uma ambiguidade. Uma forma de resolver isso, é indicando que o *bean* é uma alternativa, através da anotação `@Alternative`:

```
@Alternative
public class MeuGerenciadorDeTransacao implements Transacionado {
```

Um *bean* alternativo sobrepõe a prioridade de um *bean* existente. Então o CDI irá utilizar o `MeuGerenciadorDeTransacao` por ser uma alternativa, em vez de utilizar o `TransacionadoPadrao`.

O ponto é que assim como um *interceptor*, é preciso ativar um *bean* alternativo no arquivo `beans.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://xmlns.jcp.org/xml/ns/javaee"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/bea
       version="1.2" bean-discovery-mode="all">

    <alternatives>
        <class>br.com.alura.livraria.tx.MeuGerenciadorDeTransacao</class>
    </alternatives>

    <interceptors>
        <class>br.com.alura.alura_lib.tx.GerenciadorDeTransacao</class>
    </interceptors>
</beans>
```

Ao reiniciar o servidor, e remover um livro, as mensagens do `MeuGerenciadorDeTransacao` não são exibidas no console. Mas por quê?

Como estamos trabalhando com projetos separados, *bean packages* separados, o que é declarado no no `beans.xml` como *bean* alternativo, é considerado como alternativa apenas dentro do projeto `livraria` .

Existe uma forma de resolver isso, mas não será via `beans.xml` , portanto vamos comentar as linhas referentes ao *bean* alternativo:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://xmlns.jcp.org/xml/ns/javaee"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/bea
       version="1.2" bean-discovery-mode="all">

    <!-- <alternatives>
        <class>br.com.alura.livraria.tx.MeuGerenciadorDeTransacao</class>
    </alternatives> -->

    <interceptors>
        <class>br.com.alura.alura_lib.tx.GerenciadorDeTransacao</class>
    </interceptors>
</beans>
```

Assim como com os *interceptors*, conseguimos ativar um *bean* alternativo programaticamente. E com a mesma anotação `@Priority` . A diferença é que quando ativamos o *bean* alternativo programaticamente, estamos definindo que ele estará no escopo global. O *bean* será uma alternativa para todo o projeto, inclusive as bibliotecas que o projeto utiliza.

Se estivéssemos utilizando uma alternativa a um *bean* dentro do próprio projeto, bastaria habilitá-lo no *beans.xml*.

Vamos definir que a prioridade é de aplicação:

```
@Alternative
@Priority(Interceptor.Priority.APPLICATION)
public class MeuGerenciadorDeTransacao implements Transacionado {
```

Ao reiniciar o servidor e cadastrar um livro, é possível ver que agora o *bean* alternativo está sendo utilizado. Ao observar o console, podemos ver que o `MeuGerenciadorDeTransacao` está sendo utilizado:

```
Abrindo uma transação
Antes de executar o método interceptado
Gravando livro CDI
// algumas mensagens do hibernate
Antes de commitar a transação
```

Essas são outras formas que existem para lidar com ambiguidades, além dos qualificadores. Temos o `@Typed` para restringir o *bean* e fazer com que ele atenda a um tipo específico. Vimos também o `@Specializes` , utilizando quando trabalhamos com

herança. E por fim, vimos o `@Alternative`, utilizado quando temos algo implementado na aplicação e queremos deixar de utilizar aquele *bean* e passar a utilizar um *bean* alternativo.

O `MeuGerenciadorDeTransacao` atende a `MeuGerenciadorDeTransacao` e `Transacionado`. Podemos utilizar o `@Typed` para que ele atenda somente à `Transacionado`:

```
@Alternative
@Priority(Interceptor.Priority.APPLICATION)
@Typed(Transacionado.class)
public class MeuGerenciadorDeTransacao implements Transacionado {
```