01

Herança

Transcrição

Vamos dar uma olhada na classe `GerenciadorDeTransacao` :

```
@Interceptor
@Transacional
public class GerenciadorDeTransacao implements Serializable {

    private static final long serialVersionUID = -3628286010161624823L;
    private EntityManager em;

    @Inject
    public GerenciadorDeTransacao(EntityManager em) {
        this.em = em;
    }

    @AroundInvoke
    public Object executaComtransacao(InvocationContext context) {

        em.getTransaction().begin();

        try {
            Object resultado = context.proceed();
            em.getTransaction().commit();

            return resultado;
        } catch (Exception e) {
            em.getTransaction().rollback();
            throw new RuntimeException(e);
        }
    }
}
```

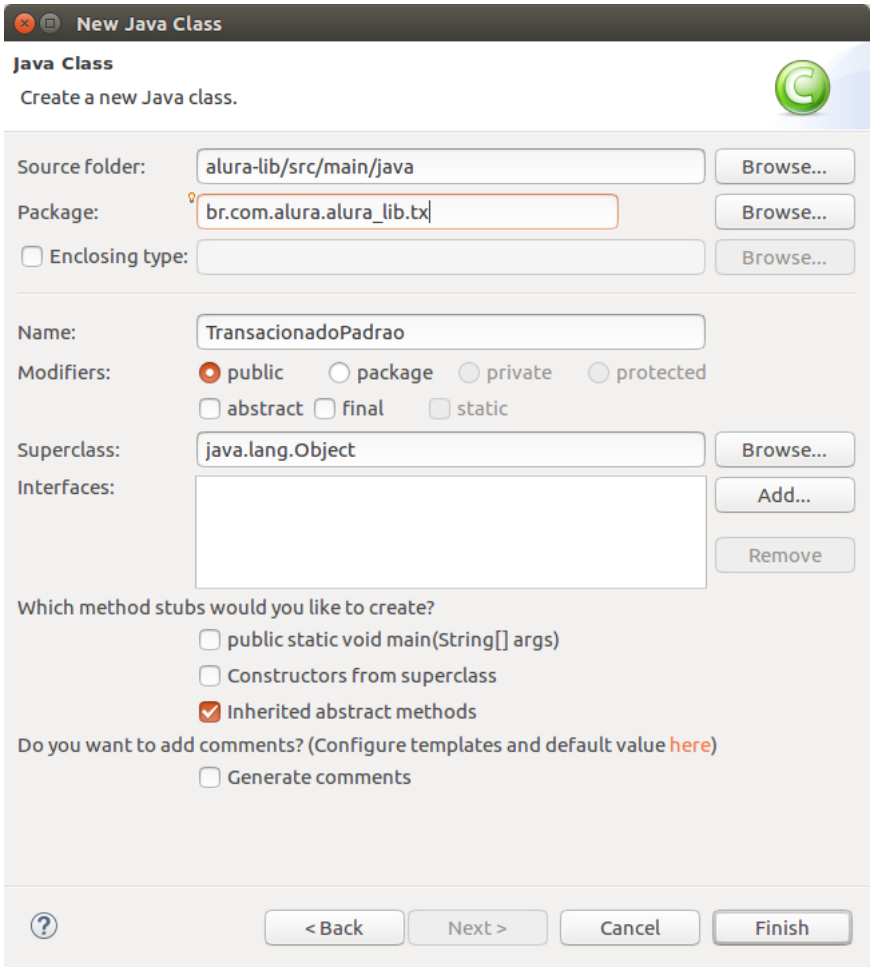
Ela é um *interceptor* que intercepta métodos que estão anotados com `@Transacional` . Ao interceptar o método. será aberta uma transação e depois, o método será invocado. Em seguida, se a operação for um sucesso a transação será *commitada*, se der errado será feito um *rollback*.

Essa implementação que criamos dentro do método `executaComtransacao()` , que irá interceptar os demais métodos, não atende todos os casos. Se um cliente tivesse a necessidade de executar algo antes da execução do método, ou depois da execução do método, seria necessário implementar um outro *interceptor*, associar o *interceptor* à anotação `@Transacional` e implementar o controle de transação desejado.

A ideia é que não seja necessário implementar um outro *interceptor*, havendo apenas um *interceptor* para fazer o controle de transação.

Queremos que o cliente seja capaz de sobrescrever o comportamento relacionado à transação. Uma forma que temos de resolver isso, é extrair o comportamento padrão para uma classe, e quem utilizar a biblioteca poderá estender da classe criada.

Vamos criar uma nova classe, chamada de `TransacionadoPadrao`:



Vamos recortar o código do método `executaComTransacao` do *interceptor*. Como vamos utilizar essa classe no *interceptor*, precisamos implementar `Serializable`.

O `EntityManager`, que está sendo recebido via injeção de dependências, utiliza o modificador de acesso `protected`, pois como vamos estender essa classe futuramente, queremos que ele esteja disponível.

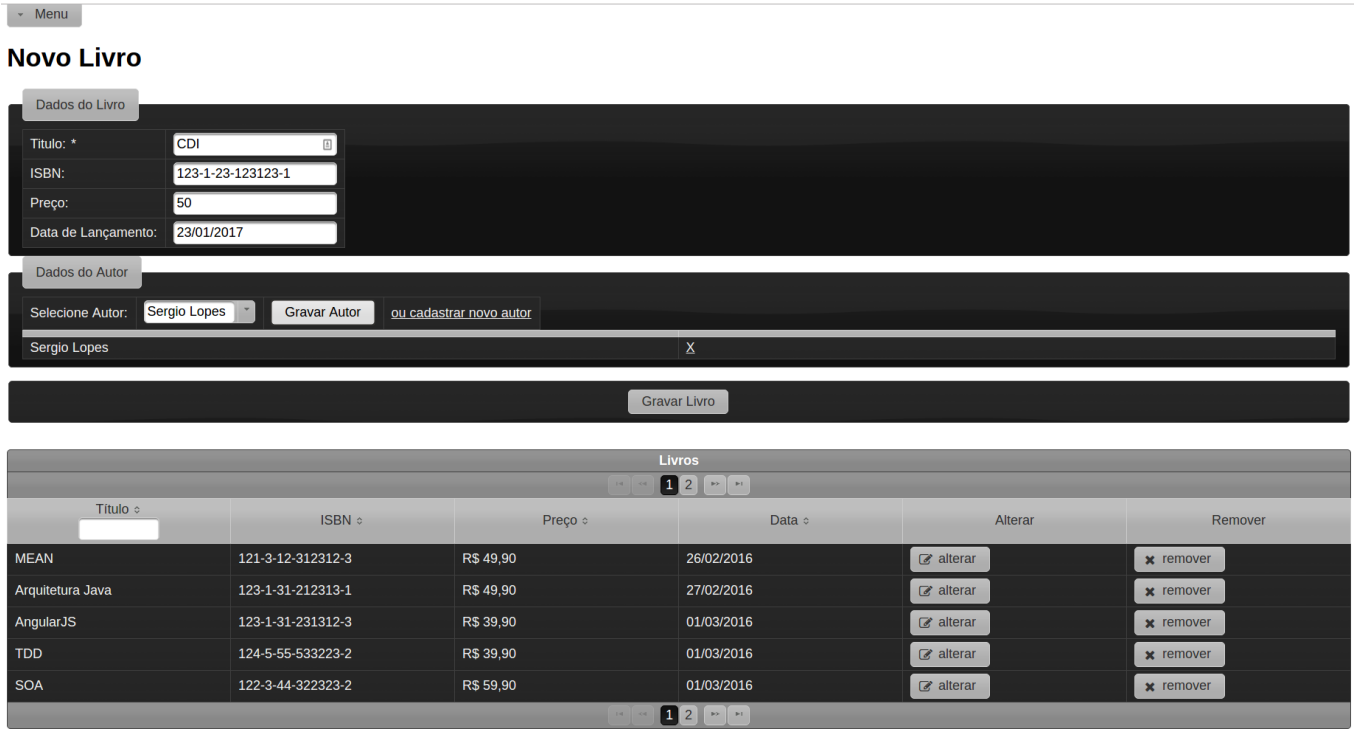
```
public class TransacionadoPadrao implements Serializable {

    private static final long serialVersionUID = 5075478157006067815L;

    @Inject
    protected EntityManager em;

    public Object executaComtransacao(InvocationContext context) {

        em.getTransaction().begin();

        try {
            Object resultado = context.proceed();
            em.getTransaction().commit();

            return resultado;
        } catch (Exception e) {
            em.getTransaction().rollback();
            throw new RuntimeException(e);
        }
```

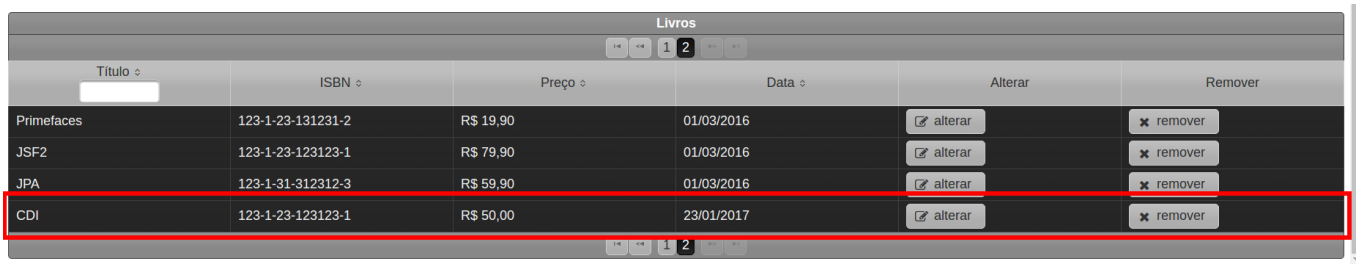
```
    }
}
```

Na classe `GerenciadorDeTransacao`, como o método não é mais responsável por gerenciar a transação, vamos mudar seu nome para `interceptar()`. Nesse *interceptor*, no lugar de injetarmos o `EntityManager`, será injetado o `TransacionadoPadrao`:

```
@Interceptor
@Transacional
public class GerenciadorDeTransacao implements Serializable {

    private static final long serialVersionUID = -3628286010161624823L;

    private TransacionadoPadrao transacionado;

    @Inject
    public GerenciadorDeTransacao(TransacionadoPadrao transacionado) {
        this.transacionado = transacionado;
    }

    @AroundInvoke
    public Object executaComtransacao(InvocationContext context) {
        return transacionado.executaComtransacao(context);
    }
}
```

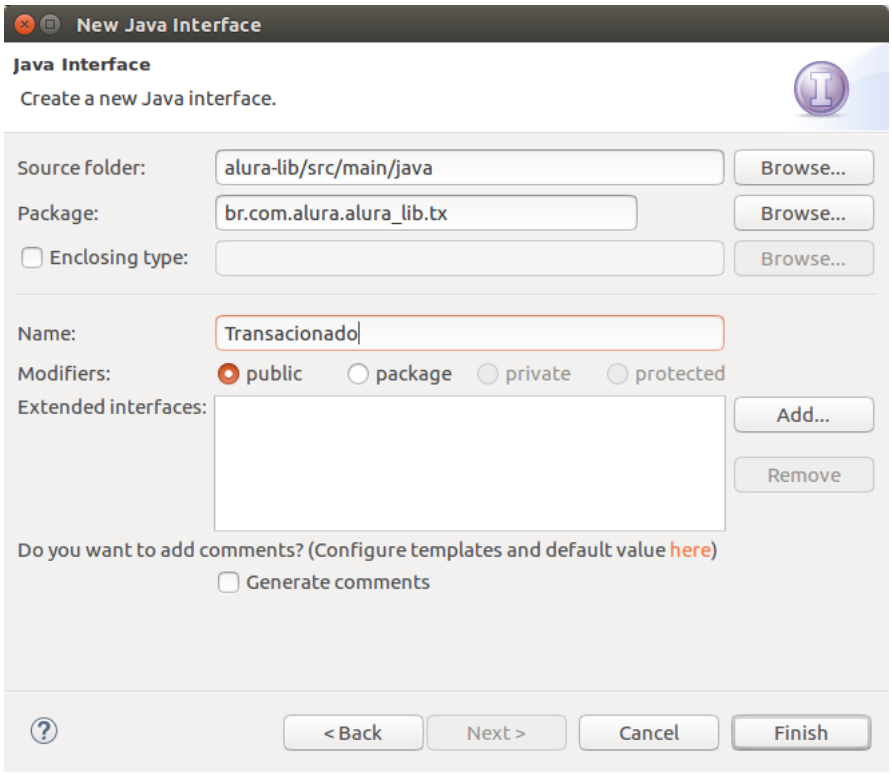
Para testar, vamos instalar o `jar` no repositório local e atualizar o projeto `livraria`. Vamos fazer um "Clean" e iniciar o Tomcat. Ao tentar gravar um livro, tudo continua funcionando:





Tudo funcionou. Agora podemos estender a classe que criamos e sobrescrever o comportamento.

No `GerenciadorDeTransacao` estamos trabalhando diretamente com a implementação: o `TransacionadoPadrao`. Para evitar utilizar a implementação diretamente, vamos criar uma *interface*:



O método da interface terá a mesma assinatura do método `executaComTransacao`. Como todas as classes que implementarem `Transacionado` precisarão implementar a *interface* `Serializable`, vamos estender dessa *interface*. Dessa forma, todas as classes que implementarem `Transacionado` já podem ser serializadas:

```
public interface Transacionado extends Serializable {

    public Object executaComtransacao(InvocationContext context);
}
```

Na classe `TransacionadoPadrao`, não é mais necessário implementar `Serializable`, e vamos passar a implementar `Transacionado`.

```
public class TransacionadoPadrao implements Transacionado {
```

Como já temos o método `executaComTransacao()` na classe, não precisamos fazer mais nada.

Agora vamos instalar a biblioteca no repositório local, e atualizar o projeto `livraria`. Vamos dar um "Clean" no servidor e em seguida reiniciá-lo.

Ao acessar o sistema e tentar remover algum livro da lista, tudo continua funcionando.

Mas perceba que no nosso `GerenciadorDeTransacao`, não fizemos alterações e continuamos recebendo o `TransacionadoPadrao`:

```
@Inject
public GerenciadorDeTransacao(TransacionadoPadrao transacionado) {
    this.transacionado = transacionado;
}
```

Mas em tese, não queremos injetar um `TransacionadoPadrao`, e sim um `Transacionado`. Pois queremos trabalhar com a interface, e não com a implementação.

Podemos definir na classe `TransacionadoPadrao`, que ela deve atender apenas por um tipo específico, que é a *interface* `Transacionado`. Para isso devemos utilizar a anotação `@Typed`:

```
@Typed(Transacionado.class)
public class TransacionadoPadrao implements Transacionado {
```

Vamos instalar novamente o `jar`, atualizar o projeto `livraria`, e tentar subir a aplicação, fazendo o "Clean" e iniciando o Tomcat.

Dessa vez ocorre um erro:

```
WELD-001408: Unsatisfied dependencies for type TransacionadoPadrao with qualifiers @Default
```

O erro indica que não existe ninguém para suprir a dependência do `TransacionadoPadrao`. Isso ocorreu justamente porque foi utilizada a anotação `@Typed` indicando que o `TransacionadoPadrao` só irá atender quando a dependência for um `Transacionado`.

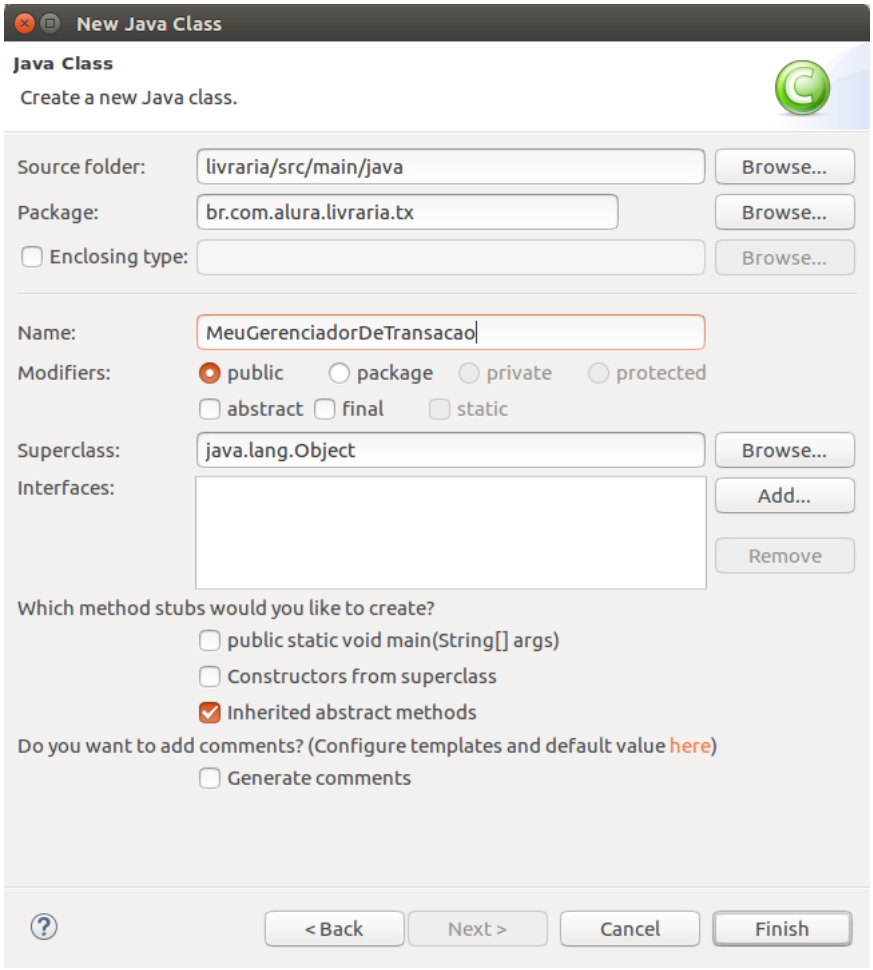
Na classe `GerenciadorDeTransacao`, em vez de receber um `TransacionadoPadrao`, vamos receber um `Transacionado`.

```
private Transacionado transacionado;

@Inject
public GerenciadorDeTransacao(Transacionado transacionado) {
    this.transacionado = transacionado;
}
```

Vamos instalar novamente o `jar`, atualizar o projeto `livraria`, e tentar subir a aplicação, fazendo o "Clean" e iniciando o Tomcat. Ao tentar cadastrar um livro, e também ao tentar remover o livro cadastrado, tudo funcionou.

Agora nós injetamos `Transacionado`, que é uma *interface*. Se por algum motivo for preciso alterar a implementação, não será mais necessário alterar a classe `GerenciadorDeTransacao`.

Agora vamos ver como ficaria a situação de poder sobrescrever aquele comportamento no projeto `livraria`. Para isso, será criada uma nova classe:



Dentro da classe `MeuGerenciadorDeTransacao`, vamos estender a classe `TransacionadoPadrao`:

```
public class MeuGerenciadorDeTransacao extends TransacionadoPadrao {

    private static final long serialVersionUID = -8590951365580768798L;

}
```

Vamos sobrescrever o método `executaComTransacao`. O método possui a mesma implementação do método da classe pai, apenas com a adição de alguns `System.out`, para imprimir no console as operações que estão sendo realizadas.

```
public class MeuGerenciadorDeTransacao extends TransacionadoPadrao {

    private static final long serialVersionUID = -8590951365580768798L;

    @Override
    public Object executaComtransacao(InvocationContext context) {

        System.out.println("Abrindo uma transação");
        em.getTransaction().begin();

        try {
            System.out.println("Ante sd eexecutar o método interceptado");
            Object resultado = context.proceed();
```

```
            System.out.println("Antes de commitar a transação");
            em.getTransaction().commit();

            return resultado;
        } catch (Exception e) {

            System.out.println("Antes de efetuar o rollback da transação");
            em.getTransaction().rollback();

            throw new RuntimeException(e);
        }
    }
}
```

Após dar um "Clean" e tentar iniciar o Tomcat, um erro ocorre:

```
WELD-001409: Ambiguous dependencies for type Transacionado with qualifiers @Default
  at injection point [BackedAnnotatedParameter] Parameter 1 of [BackedAnnotatedConstructor] @Inject
  at br.com.alura.alura_lib.tx.GerenciadorDeTransacao.<init>(GerenciadorDeTransacao.java:21)
  Possible dependencies:
  - Managed Bean [class br.com.alura.livraria.tx.MeuGerenciadorDeTransacao] with qualifiers [@Any @I
  - Managed Bean [class br.com.alura.alura_lib.tx.TransacionadoPadrao] with qualifiers [@Any @Defaul
```

Existem duas classes que atendem ao `Transacionado`: `TransacionadoPadrao` e `MeuGerenciadorDeTransacao`. Mas nesse caso queremos que `MeuGerenciadorDeTransacao` seja injetado. Precisamos indicar que `MeuGerenciadorDeTransacao` é uma especialização de `TransacionadoPadrao` e que ele deve utilizar essa especialização. Para isso, existe a anotação `@Specializes`:

```
@Specializes
public class MeuGerenciadorDeTransacao extends TransacionadoPadrao {
```

Ao iniciar o Tomcat, acessar o sistema e tentar cadastrar um livro, é possível ver no console as mensagens:

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

Por meio da utilização de herança foi possível utilizar nossa implementação, e sobrescrevemos a forma como a transação está sendo realizada.