



07

Divisão de responsabilidades



46%

ATIVIDADES
7 DE 9FÓRUM DO
CURSOVOLTAR
PARA
DASHBOARD

Vamos testar alguns tipos de gerenciamento de transações dentro de um servidor JEE?

1) Apenas com objetivos didáticos, abra a classe

`AutorDao` e anote-a com

```
@TransactionManagement(TransactionManagementType.CONTAINER)
```

para definirmos explicitamente que quem

controla nossas transações é o *container*.

```
@Stateless
```

```
@TransactionManagement(TransactionM
```

```
// Opcional
```

```
public class AutorDao {
```

```
...
```

```
}
```

COPIAR CÓDIGO

2) Ainda na classe `AutorDao` vamos deixar

explicito o padrão de configuração de transações

para cada método. Faça isso anotando o método

`salva()` com

```
@TransactionAttribute(TransactionAttributeType.REQUIRED) :
```

```
@TransactionAttribute(TransactionA
```

```
public void salva(Autor
```

```
autor) {
```

```
...
```

```
}
```

COPIAR CÓDIGO



116.5k xp





46%

ATIVIDADES
7 DE 9FÓRUM DO
CURSOVOLTAR
PARA
DASHBOARD

ATENÇÃO: Vale ressaltar que ambas configurações feitas são opcionais, visto que são os padrões adotados pelo *container*. Então, ao republicar e testar a aplicação, tudo deve continuar funcionando normalmente.

3) Podemos entender melhor o controle de transações implementando a divisão de responsabilidades proposta no vídeo do capítulo. Comece substituindo o

`TransactionAttributeType.REQUIRED` por `TransactionAttributeType.MANDATORY` , assim, o *container* irá verificar se há uma transação aberta, caso contrário, lançará uma *exception* do tipo

`EJBTransactionRequiredException` . Teste isso reiniciando o servidor após a alteração e tentando adicionar um novo autor (olhe no console do eclipse).

4) Agora, vamos colocar a responsabilidade de criar a transação em um outro EJB que não seja o DAO, visto que os DAOs não são o melhor lugar para isso. Crie uma classe, no pacote

`br.com.caelum.livraria.bean` , chamada `AutorService` que ficará entre o `AutorBean` e `AutorDao` . Transforme essa classe em um EJB *Stateless* e injete o EJB `AutorDao` nela.

5) Crie em `AutorService` um método `adiciona()` que recebe um **Autor** e delega para o DAO a tarefa de salvar, e um outro método chamado `todosAutores()` que também fará apenas a delegação para o DAO.



116.5k xp



@Stateless



46%

ATIVIDADES
7 DE 9FÓRUM DO
CURSOVOLTAR
PARA
DASHBOARD

```
public class AutorService {  
  
    @Inject  
    AutorDao dao;  
  
    public void adiciona(Autor  
autor) {  
        this.dao.salva(autor);  
    }  
  
    public List<Autor>  
todosAutores() {  
        return  
this.dao.todosAutores();  
    }  
}
```

COPIAR CÓDIGO

6) Precisamos agora refatorar a classe `AutorBean` para utilizar o novo EJB de serviços `AutorService`. Sendo assim, ao invés de injetarmos `AutorDao`, passaremos a injetar `AutorService`. Não podemos esquecer de refatorar os métodos que usavam o atributo `dao` para passar a usar o atributo `service`:

```
@Model  
public class AutorBean {  
    ...  
  
    @Inject  
    private AutorService  
service; // AutorDao dao; // =  
new AutorDao();  
  
    ...  
    public void cadastra() {  
        //  
this.dao.salva(autor);  
    }  
}
```



116.5k xp





46%

ATIVIDADES
7 DE 9FÓRUM DO
CURSOVOLTAR
PARA
DASHBOARD

```
this.service.adiciona(autor);
    this.autor = new
    Autor();
    }

    public List<Autor>
    getAutores() {
        // return
        this.dao.todosAutores();

        return
        this.service.todosAutores();
    }
}
```

COPIAR CÓDIGO



7) Faça o *Full Publish* da aplicação, adicione um novo autor e veja se a *exception* do teste anterior será novamente lançada. O que você acha que aconteceu?

Responda



116.5k xp





46%

ATIVIDADES
7 DE 9FÓRUM DO
CURSOVOLTAR
PARA
DASHBOARD

116.5k xp

