# EJB:

Este documento foi feito para relembrar alguns conceitos do EJB.

## Anotations:

@Stateless: Essa anotação serve para dizer que essa classe será gerenciada pelo EJB. Diferente da anotação Singleton, onde apenas uma instância é criada, essa anotação pode criar diversas instâncias dependendo da necessidade e do valor máximo inserido no pool de conexões.

@PostConstruct : Essa anotação serve para que seja chamado esse método logo após a criação da instancia da classe. Essa função de chamar após a criação da instância também é chamada de callback.

Singleton : Cria apenas uma instancia da classe (por padrão é lazy, ou seja, somente é criada quando é utilizada.

Startup: Essa anotação faz com que seja criada uma intância da classe logo ao iniciar a aplicação.Com isso a classe deixa de ser lazy e começa a ser Eager.

EntityManager: A EntityManager é a interface principal da especificação JPA. Ela serve para gerenciar os dados armazenados pelos sistemas.

PersistenceContext: Como o EJB Container administrará o **JPA**, é preciso usar uma anotação especifica do mundo **EJB** para fazer a "injeção" do EntityManager, nesse caso, precisamos utilizar @PersistenceContext

## Transações JTA

 JTA significa Java Transaction API que é um padrão JavaEE que se preocupe com o gerenciamento da transação dentro de um servidor JavaEE.

## Gerenciamento das transações

O JTA, então, é a forma padrão de gerenciar a transação dentro do servidor JavaEE e já funciona sem nenhuma configuração. Este padrão se chama CONTAINER MANAGED TRANSACTION (CMT).

Podemos deixar a nossa intenção explicita e configurar o gerenciamento pelo container. Para tal existe a anotação @TransactionManagement que define o tipo de gerenciamento da transação, no nosso caso *CONTAINER*:

@Stateless

@TransactionManagement(TransactionManagementType.CONTAINER) //opcional

public class AutorDao {

### TransactionAttributeType.REQUIRED

Significa que o JTA garante uma transação rodando quando o método é chamado. Se não tiver nenhuma transação, uma nova é aberta. Caso já tenha uma rodando, a atual será utilizada. De qualquer forma, sempre é preciso ter uma transação (REQUIRED).

### TransactionAttributeType.MANDATORY

MANDATORY significa obrigatório. Nesse caso, o container verifica se já existe uma transação rodando, caso contrário, joga uma exceção. Ou seja, quem faz a chamada deve abrir uma transação.

### TransactionAttributeType.REQUIRES\_NEW

O REQUIRES\_NEW indica que sempre deve ter uma nova transação rodando. Caso já exista, a transação atual será suspensa para abrir uma nova. Caso não tenha nenhuma rodando, será criada uma nova transação.

### TransactionAttributeType.NEVER

Indica que jamais deve haver uma transação em execução. Isso pode ser útil para métodos que obrigatoriamente devem ser executados sem contexto transacional.

## Gerenciamento da transação programaticamente

Para o EJB Container aceitar o gerenciamento da transação programaticamente, é preciso reconfigurar o padrão. Ou seja, ao invés de usar CONTAINER na anotaçãoTransactionManagement usaremos BEAN, porque o Session Bean vai gerenciar a transação (também é chamado BEAN MANAGED TRANSACTION). Assim também podemos apagar a anotação @TransactionAttribute que não faz mais sentido.

Para realmente gerenciar a transação, é preciso injetar um objeto com este papel. Para este propósito existe a interface UserTransaction do JTA. Basta injetar o objeto através da anotação @Inject:

@Inject UserTransaction tx;

UserTransaction possui os métodos clássicos relacionados com o gerenciamento da transação como begin(), commit() e rollback().

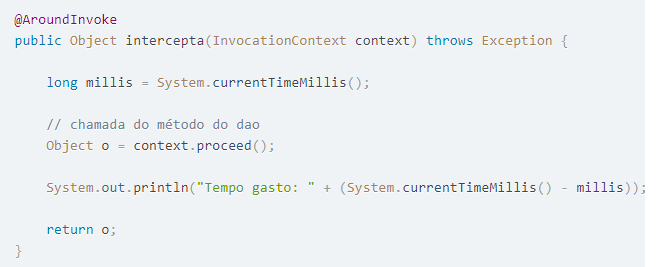
## Interceptadores

Na maioria das aplicações é preciso fazer um monitoramento, principalmente quando a aplicação não está muito madura ainda. Por exemplo, pode ser necessário monitorar o tempo de acesso ao banco de dados.

Para que possamos ter o interceptador, primeiramente devemos anotar a classe do interceptador com @AroundInvoke.

O método que irá fazer essa interceptação deve por default retornar um object e deve receber um InvocationContext. Esse invocationContext terão os valores da classe onde estamos querendo interceptar.

Exemplo de Interceptador:



Quando usamos o método context.proceed(), estamos mandando que o EJB de segmento ao método que foi interceptado e após isso será novamente retornado para o Interceptor.

Caso quiséssemos pegar o nome da classe interceptada e o método, poderíamos fazer desta forma:

String metodo = context.getMethod().getName();

String nomeClasse = context.getTarget().getClass().getSimpleName();

Na classe que desejamos interceptar, basta colocar a anotation @Interceptor passando qual interceptador queremos utilizar.

Exemplo da classe interceptada:

