

# Jakarta EE Web Total

Arquitetura & Entidades da JPA



---

---

---

---

---

---

---

---

## Tópicos Abordados

- A JPA (Java Persistence API)
- Entidades
- Tipos de persistência
- Chaves primárias
  - Simples
  - Compostas
- Embeddable Classes
- EntityManager
- Ciclo de vida de uma entidade
- O arquivo persistence.xml
- Conexão com bancos de dados
  - Pool de conexões
  - Data source

---

---

---

---

---


---

---

---

## JPA – Java Persistence API

- A integração entre aplicações e bancos de dados relacionais é muito comum
- O problema é que aplicações e bancos de dados “falam línguas diferentes”



---

---

---

---

---

---

---

---

## A especificação Java EE e o ORM



- Nas primeiras versões do Java EE (chamado de J2EE), a persistência de dados era feita pelos componentes denominados *entity beans*
  - Os entity beans eram complexos e limitados
- Em paralelo, surgiu o *Hibernate*
  - Permitia usar todas as facilidades da orientação a objetos a favor do ORM
  - Open source
- A JPA foi criada com base no Hibernate e foi incorporada na plataforma a partir do Java EE 5



---

---

---

---

---

---

---

## Especificação da JPA



- A JPA é uma especificação
  - É apenas um documento com diretrizes para implementação da JPA
- As implementações da JPA são chamadas de *persistence providers*
  - EclipseLink (implementação de referência)
  - Hibernate
- Todas as implementações que seguem a especificação da JPA funcionam da mesma forma



---

---

---

---

---

---

---

## Entidades



- Em orientação a objetos, chamamos as instâncias de classes de *objetos*
- No mundo ORM, os objetos que representam dados persistidos em tabelas do banco de dados são chamados de *entidades (entities)*

ContaCorrente	
- id : Integer	
- titularConta : String	
- numConta : String	
- numAgencia : Integer	
- limite : Double	



CONTA_CORRENTE	
id	integer
titularConta	varchar(128)
numConta	varchar(10)
numAgencia	integer
limite	decimal(10, 2)



---

---

---


---

---

---

---

## Criando Entidades



- Exemplo de implementação de uma entidade

```

@Entity
public class ContaCorrente {

    @Id
    @GeneratedValue
    private Integer id;

    private String titularConta;
    private String numConta;
    private Integer numAgencia;
    private Double limite;


    // getters & setters...
}

```

@Entity define a classe como sendo uma entidade

@Id define o ID da entidade

@GeneratedValue determina que o ID deve ser gerado automaticamente




---

---

---

---


---

---


---

---

## Regras Para Uma Entidade



- Classe anotada com @Entity
- Construtor sem parâmetros (public ou protected)
- Deve implementar Serializable
  - Se for usada por interfaces remotas de EJBs
  - For usada em um JSF Bean com escopo de sessão
- Os atributos não podem ser públicos
  - Podem ser acessados externamente apenas através de getters e setters
- Deve ter um ID (chave primária)




---

---

---

---


---

---

---

---

## Tipos de Persistência



- Persistent fields*
  - As annotations são realizadas nos atributos
- Persistent properties*
  - As annotations são realizadas nos métodos getters

```

@Entity
public class ContaCorrente {

    @Id
    private Integer id;

}

```


```

@Entity
public class ContaCorrente {

    @Id
    public Integer getId() {
        return id;
    }

}

```




---

---

---

---

---

---

---

---

## Chaves Primárias em Entidades



- Toda entidade precisa ter um identificador único
  - Assim a entidade pode ser encontrada
- A chave pode ser simples ou composta
  - Simples
    - Annotation @Id
  - Composta
    - Classe específica pra representar a chave
    - Annotations @IdClass e @Id



---

---

---

---

---

---

---

## Classe de Chave Primária Composta



- A classe deve ter um construtor público sem parâmetros
- Deve implementar Serializable
- Deve implementar os métodos hashCode() e equals()

```
public class LivroPK implements Serializable {  
    private String titulo;  
    private String autor;  
  
    public LivroPK() { }  
  
    public boolean equals(Object o) { }  
  
    public int hashCode() { }  
  
    // getters & setters...  
}
```



---

---

---

---

---

---

---

## Definindo os IDs



```
@Entity  
@IdClass(LivroPK.class)  
public class Livro implements Serializable {  
  
    @Id  
    private String titulo;  
  
    @Id  
    private String autor;  
}
```

@IdClass define o nome da classe da chave composta

Cada uma das chaves recebe a anotação @Id



---

---

---

---

---

---

---

## Embeddable Classes



- Existem situações onde você pode desejar agrupar em uma classe, mas sem criar uma tabela no BD pra isso
- Para estes casos você pode usar as Embeddable Classes

```
@Embeddable
public class Endereco implements Serializable {
    private String rua;
    private Integer numero;
    // getters & setters...
}
```

Substitui a annotation @Entity



## Embeddable Classes



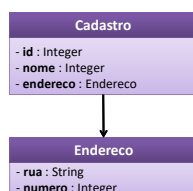
- Embeddable classes* podem ser definidas como propriedades de entidades

```
@Entity
public class Cadastro implements Serializable {
    @Id
    private Integer id;
    private String nome;
    @Embedded
    private Endereco endereco;
    // getters & setters...
}
```

O @Embedded indica uma embeddable class



## Mapeamento em Embeddable Classes



CADASTRO	
id	integer
nome	varchar
rua	varchar
numero	integer

Embeddable classes são mapeadas para a mesma tabela da entidade



## EntityManager



- A interface `EntityManager` é o ponto de entrada para o uso da JPA pelo programador
- Possui os métodos para interagir com entidades

Método	Descrição
<code>persist()</code>	Cria uma entidade
<code>merge()</code>	Atualiza uma entidade
<code>remove()</code>	Exclui uma entidade
<code>find()</code>	Busca uma entidade com base no seu ID
<code>createQuery()</code>	Permite procurar entidades de acordos com os critérios desejados



---

---

---

---

---

---

---

## EntityManager



- A instância de um `EntityManager` pode ser injetada em componentes do Java EE via CDI

```
@Stateless
public class MyBean {
    @PersistenceContext
    private EntityManager em;
    //...
}
```

`@PersistenceContext`  
injeta uma instância  
de `EntityManager`



---

---

---

---

---

---

---

## Ciclo de Vida de Uma Entidade



- Uma entidade pode assumir diversos estados durante o seu ciclo de vida
  - *new* (nova)
  - *managed* (gerenciada)
  - *detached* (não-gerenciada)
- Quando a entidade está no estado *gerenciada*, qualquer alteração nela é refletida no banco de dados



---

---

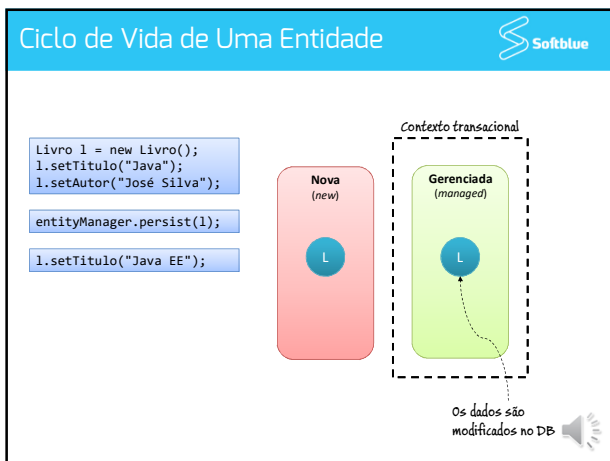
---

---

---

---

---




---

---

---

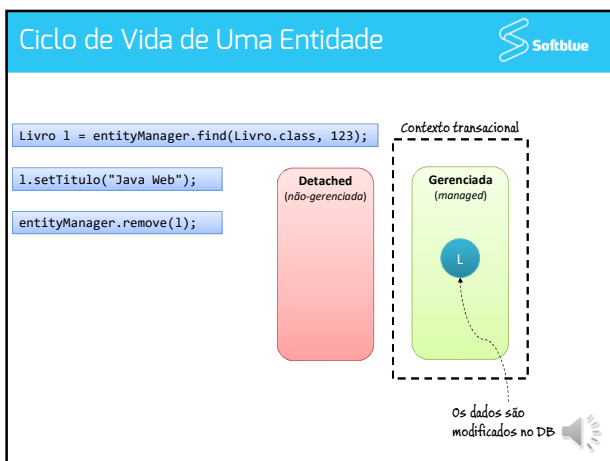
---

---

---

---

---




---

---

---

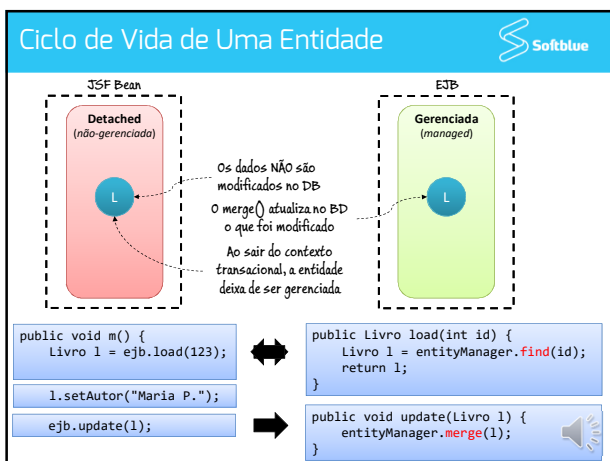
---

---

---

---

---




---

---

---

---

---

---

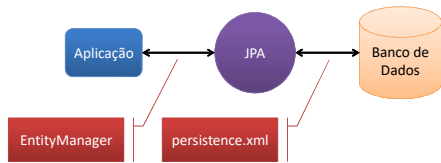
---

---

## O arquivo *persistence.xml*



- A JPA tem a responsabilidade de se comunicar com o banco de dados e fazer o gerenciamento das entidades



---

---

---

---

---

---

---

## O arquivo *persistence.xml*



- O arquivo *persistence.xml* define como a JPA se comunica com o banco de dados
  - Normalmente é utilizada uma *data source*

```
<persistence>
  <persistence-unit name="appPU">
    <jta-data-source>jdbc/appds</jta-data-source>
  </persistence-unit>
</persistence>
```



---

---

---

---

---

---

---

## Conexões com o Banco de Dados



- Em aplicações web, conexões com o banco de dados são normalmente gerenciadas pelo próprio servidor
  - Abrir conexões é um processo considerado caro computacionalmente
  - Existe um limite de conexões simultâneas que podem ficar ativas no banco de dados



---

---

---

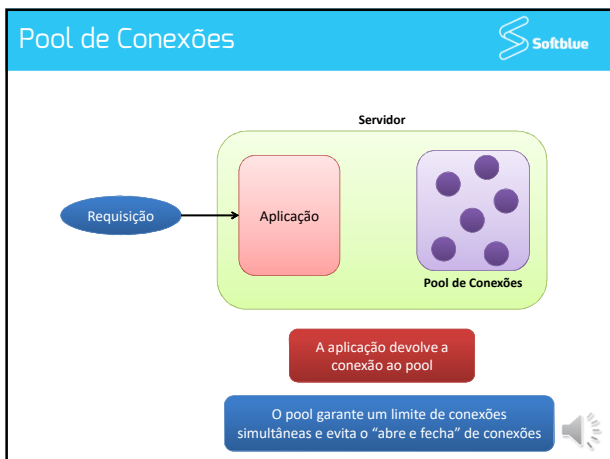
---

---

---

---





---

---

---

---

---

---

---

- ### Pool de Conexões
- A configuração do pool de conexões varia de um servidor para outro
    - Wildfly, GlassFish, etc.
  - É necessário consultar a documentação do servidor utilizado

---

---

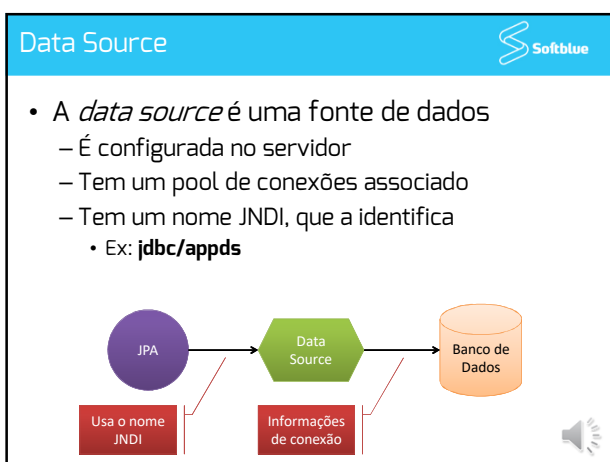
---

---

---

---

---



---

---

---

---

---

---

---



---

---

---

---

---

---

---