- GRADUAÇÃO





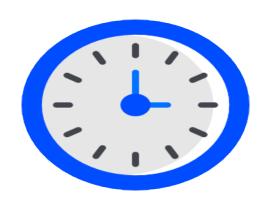
JAVA ADVANCED

#03 - AGENDA





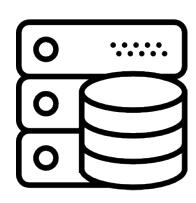
- Persistence Unit e Persistence Context
- Obter um Entity Manager
- Controlar as transações
- Estados de uma Entidade
- Métodos do Entity Manager:
 - Persist
 - Find
 - Merge
 - Refresh
 - Remove
- Métodos do ciclo de vida da Entidade



JPA – JAVA PERSISTENCE API



- Especificação independente de fabricante;
- Elementos principais:
 - Anotações para mapeamento O/R;
 - API para persistência de entidades;
 - Linguagem de consultas JPQL;
- Implementação Hibernate:
 - Hibernate Core;
 - Hibernate Annotations;
 - Hibernate Entity Manager;







ENTITY MANAGER

ENTITY MANAGER



- Unidade central para gerenciamento de entidades na JPA por meio de uma
 API padronizada;
- Responsável pela criação, atualização, remoção e consultas às entidades;
- Outras atribuições: controle de transações, gerenciamento de cache, etc...;



ENTITY MANAGER



 Uma aplicação deve ter acesso a uma instância do Entity Manager para realizar as operações de persistência;

Existem duas formas de obter-se um Entity Manager:

Application Managed Entity Manager: A própria aplicação obtém um Entity Manager
 e o fecha utilizando métodos da API apropriados;

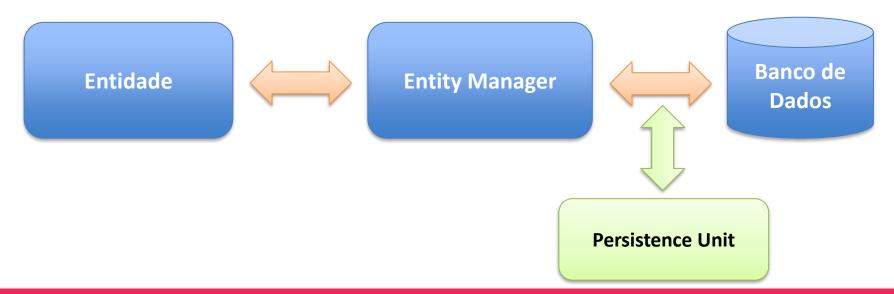
2. Container Managed Entity Manager: Um container instância e gerencia o Entity Manager;

Vamos trabalhar neste momento com o Application Managed.

PERSISTENCE UNIT



- Uma Persistence Unit define todas as configurações necessárias para que um Entity
 Manager consiga efetuar a persistência de um conjunto de entidades;
- Na Persistence Unit definimos as configurações do banco de dados, a URL de conexão, usuário, senha e etc..;
- Cada Persistence Unit é identificada por um nome dentro do arquivo persistence.xml na pasta META-INF;



PERSISTENCE CONTEXT



- É o conjunto de instâncias de entidades "visíveis" ao Entity Manager, isto é, que ele pode gerenciar;
- Cada Entity Manager mantém apenas um único Persistence Context;
- Ao fechar um contexto de persistência, todas suas instâncias de entidades associadas tornam-se
 não gerenciadas;

Entidades Gerenciadas:

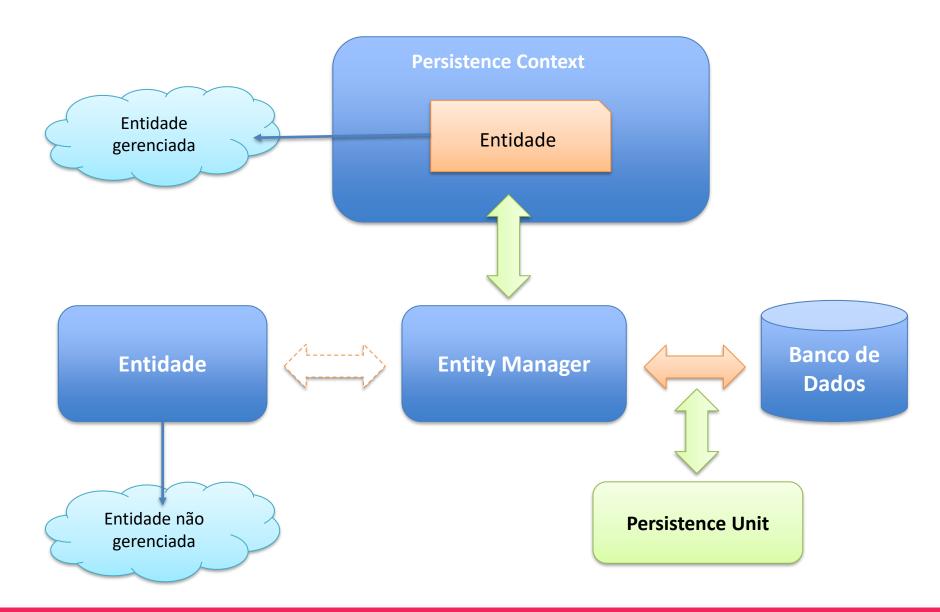
- Quando entidades estão associadas a um contexto de persistência;
- Alterações no estado das entidades são sincronizadas com o banco de dados;

Entidades Não Gerenciadas:

- Entidades não associadas a um contexto de persistência;
- Alterações nas entidades não se refletem no banco de dados;

JPA – JAVA PERSISTENCE API





INSTANCIA DE ENTITY MANAGER



 Para obter-se um Entity Manager pela aplicação é necessário primeiro uma referência a um EntityManagerFactory:

```
EntityManagerFactory f = Persistence.createEntityManagerFactory("cliente");
```

- Onde o parâmetro, no exemplo "cliente", corresponde ao nome de uma persistence unit definida no arquivo persistence.xml
- Depois é só obter o Entity Manager a partir do EntityManagerFactory:

```
EntityManager em = f.createEntityManager();
```

Para fechar o Entity Manager basta utilizar o método close:

```
em.close();
```

TRANSAÇÕES



- As transações definem quanto as entidades devem ser sincronizadas com a base de dados;
- Quando uma transação é confirmada (commit) todas as alterações realizadas nas entidades são sincronizadas com a base de dados ou, pelo contrário, podem ser descartadas (rollback);
- O Entity Manager só pode ter uma única transação ativa por vez;
- Existem dois tipos de abordagem suportadas pela JPA (definidos no persistence.xml para cada persistence unit):
 - **Resource Local**: transação nativa JDBC (mais simples, de responsabilidade da aplicação);
 - Java Transaction API (JTA): mecanismo padrão do Java EE (mais elaborado, o container Java
 EE gerencia, é transparente para a aplicação);

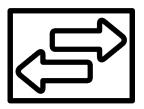
TRANSAÇÕES



- Para obter-se uma transação do Entity Manager utilizar o método getTransaction();
- Uma transação é representada pela classe EntityTransaction que contém os métodos principais:
 - begin(): delimita o início de uma transação (é obrigatório, um por vez);
 - commit(): confirma a transação (sincroniza o contexto de persistência com a base de dados);
 - rollback(): desfaz a transação;
 - isActive(): verifica se a transação encontra-se em andamento, isto é, não ocorreu commit ou rollback ainda;

Exemplo

```
EntityManager em = ...;
EntityTransaction t = em.getTransaction();
t.begin();
// métodos da JPA
t.commit(); // ou t.rollback();
```

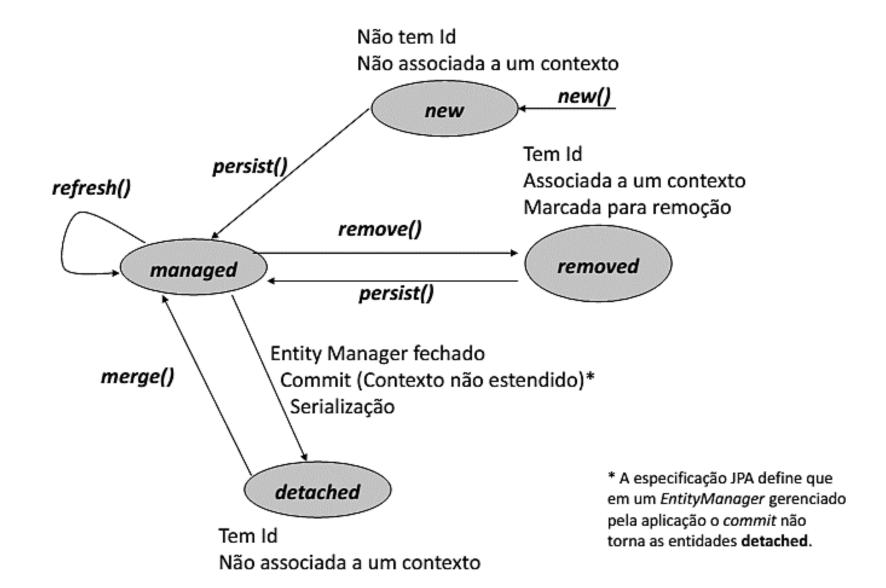




Uma entidade pode encontrar-se em um dos quatro estados abaixo:

- new: instância da entidade criada em memória mas nunca foi associada a um contexto de persistência e não possui id equivalente no banco de dados;
- managed: tem um id no banco de dados e está atualmente associada a um contexto de persistência;
- detached: tem um id no bando de dados mas não está atualmente associada ao contexto de persistência;
- removed: instância da entidade associada a um contexto de persistência mas está programada para ser removida do banco de dados;









MÉTODOS DO ENTITY MANAGER





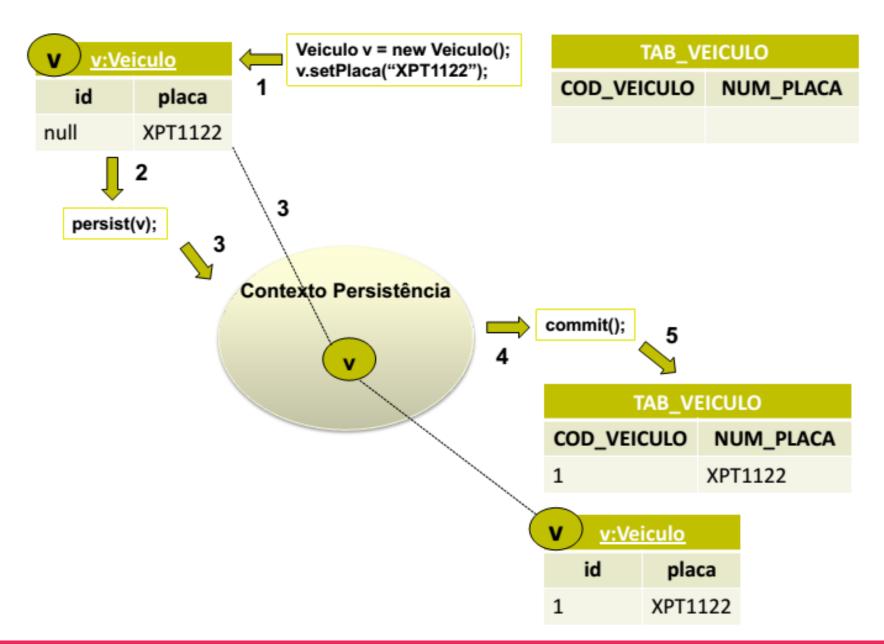
- persist (Object entity): enfileira uma nova entidade para ser inserida no banco de dados e a torna gerenciada;
 - ✓ Caso a entidade seja NEW então torna-se MANAGED;
 - ✓ Caso a entidade seja **MANAGED** ela é ignorada;
 - ✓ Caso a entidade seja **REMOVED** então torna-se **MANAGED**;
 - ✓ Caso a entidade seja DETACHED, uma IllegalArgumentException é lançada;

```
Veiculo veiculo = new Veiculo();
veiculo.setPlaca("DHZ-5678");
veiculo.setModelo("Gol");
manager.persist(veiculo);
```

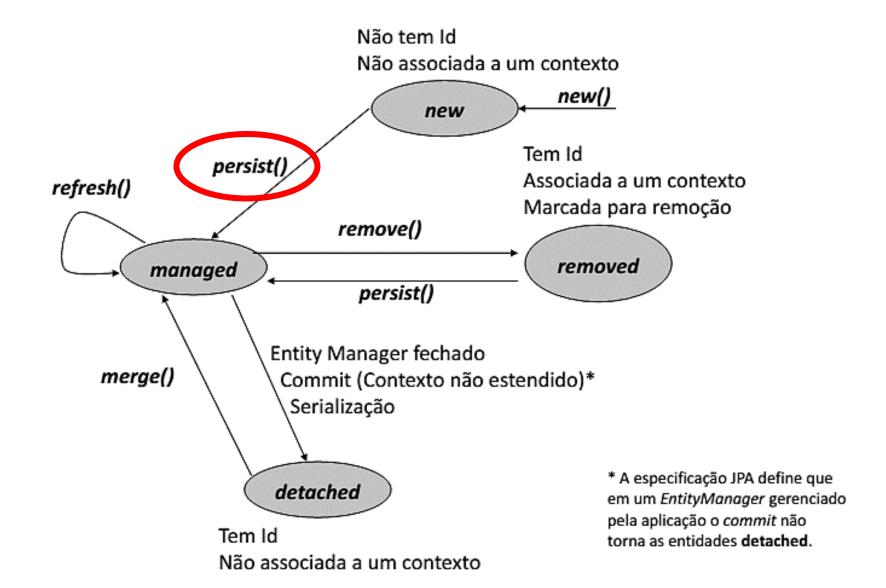


PERSIST













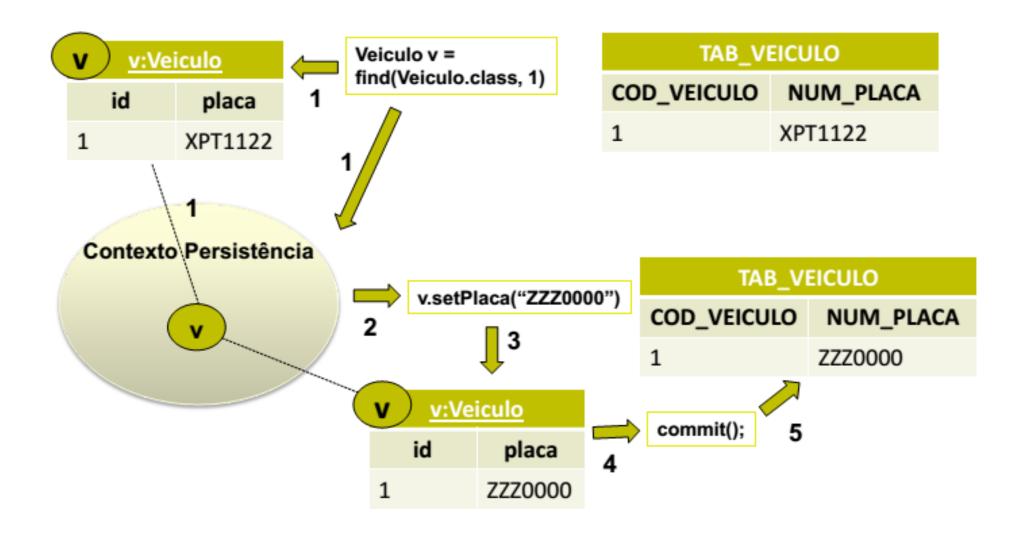
- <T> T find (Class<T> classeEntidade, Object PK): realiza uma busca por meio da chave primária da entidade;
 - ✓ Retorna uma instância MANAGED caso seja localizada ou *null* caso contrário;

```
// Busca veiculo com id igual a 10
Veiculo veiculo = manager.find(Veiculo.class, 10);
```

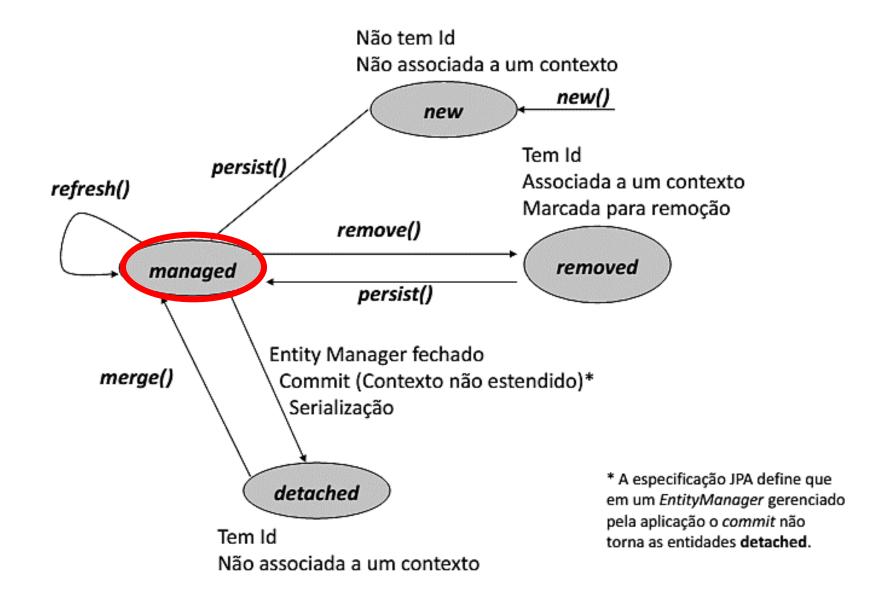
















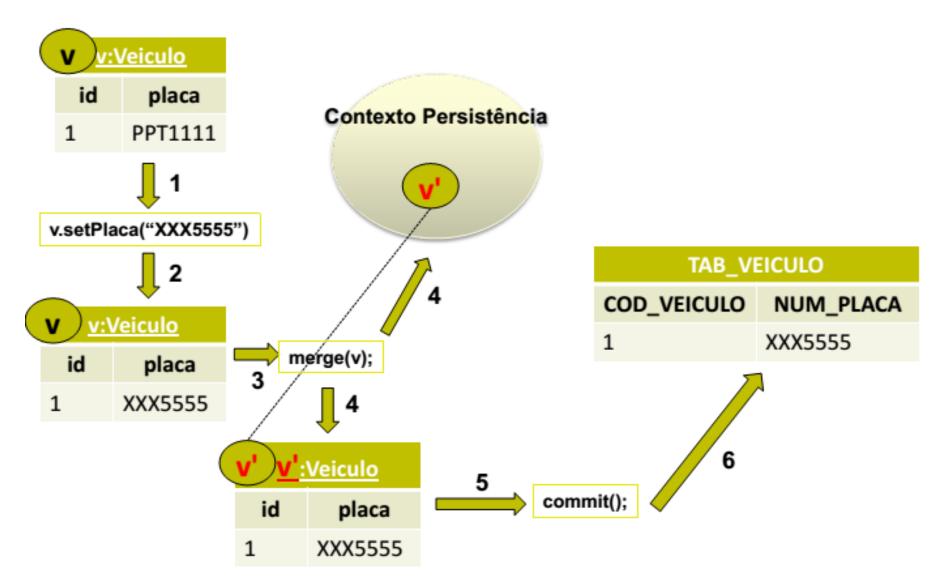
- <T> T merge (T entidade): retorna uma cópia gerenciada de uma entidade não gerenciada;
 - ✓ Caso a entidade seja **DETACHED**, seu estado é copiado para uma instância **MANAGED** com a mesma identidade ou uma nova cópia **MANAGED** da entidade é criada;
 - ✓ Caso a entidade seja NEW, uma nova entidade MANAGED é criada com o estado copiado da entidade original;
 - ✓ Caso a entidade seja **MANAGED** ela é ignorada;
 - ✓ Caso a entidade seja REMOVED, uma IllegalArgumentException é lançada;

```
Veiculo v2 = manager.merge(veiculo);
v2.setPlaca("DHZ-5678");
v2.setModelo("Fusca");
```



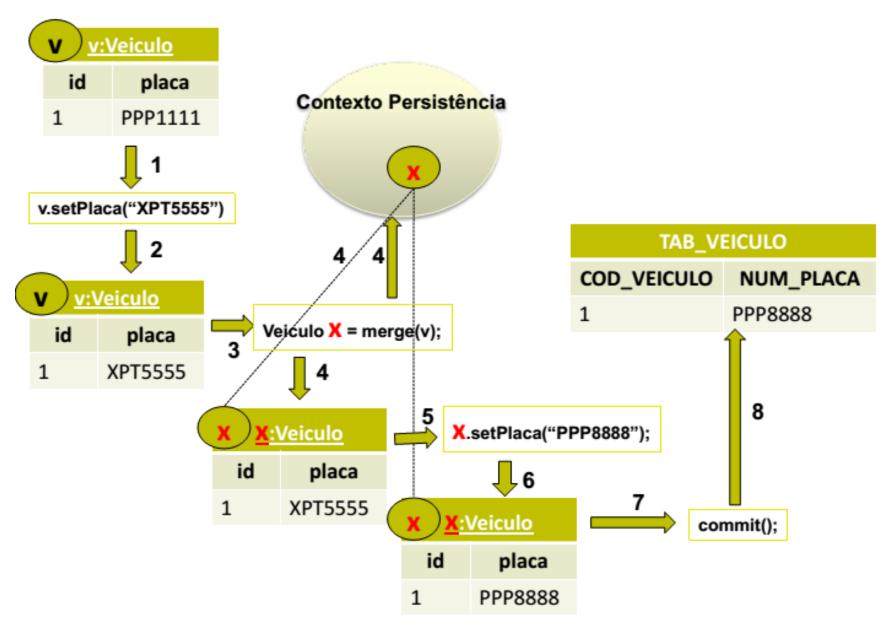
MERGE



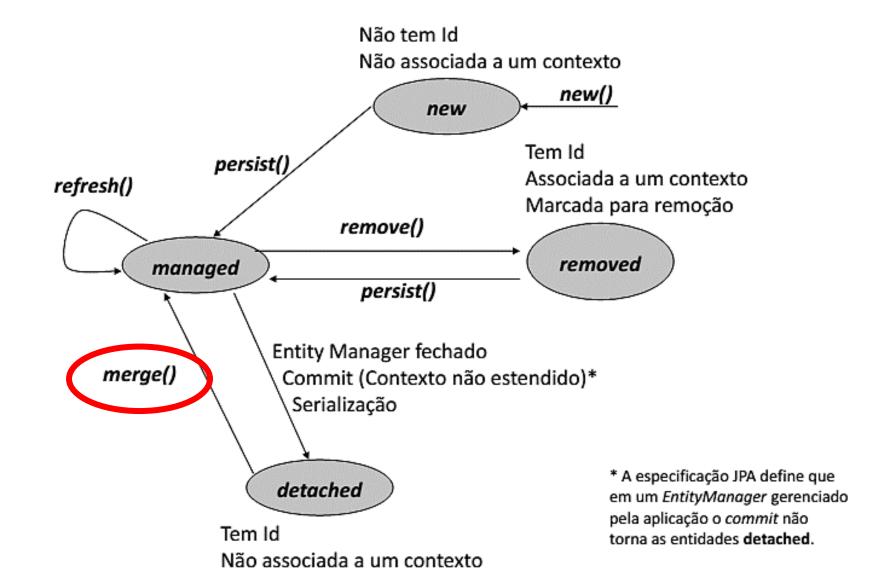














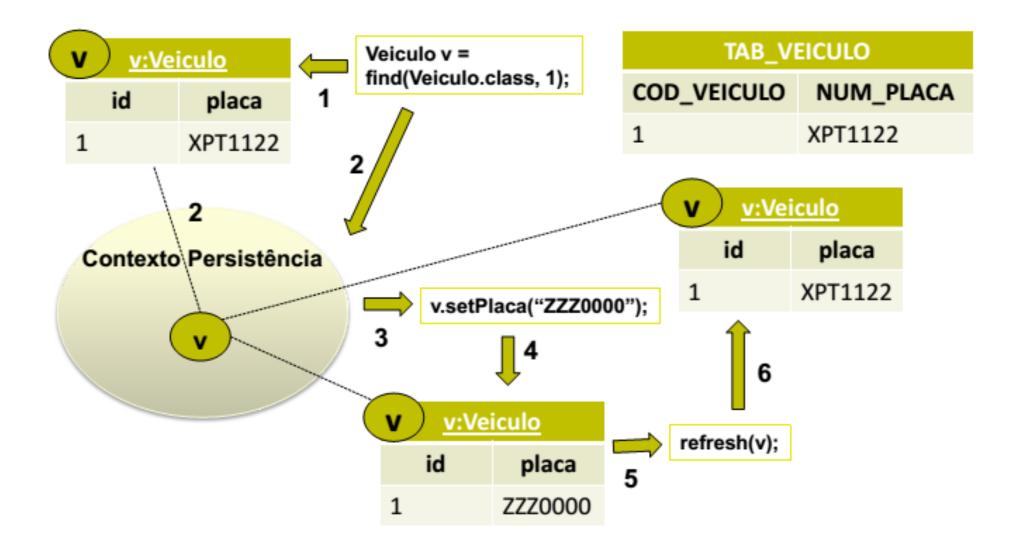


- refresh(Object entidade): certifica que o estado da entidade encontra-se sincronizado com a base de dados;
 - ✓ Caso a entidade seja NEW ela é ignorada;
 - ✓ Caso a entidade seja MANAGED, seu estado é atualizado com a base de dados;
 - ✓ Caso a entidade seja **REMOVED**, ela é ignorada;
 - ✓ Caso a entidade seja DETACHED, uma IllegalArgumentException é lançada;

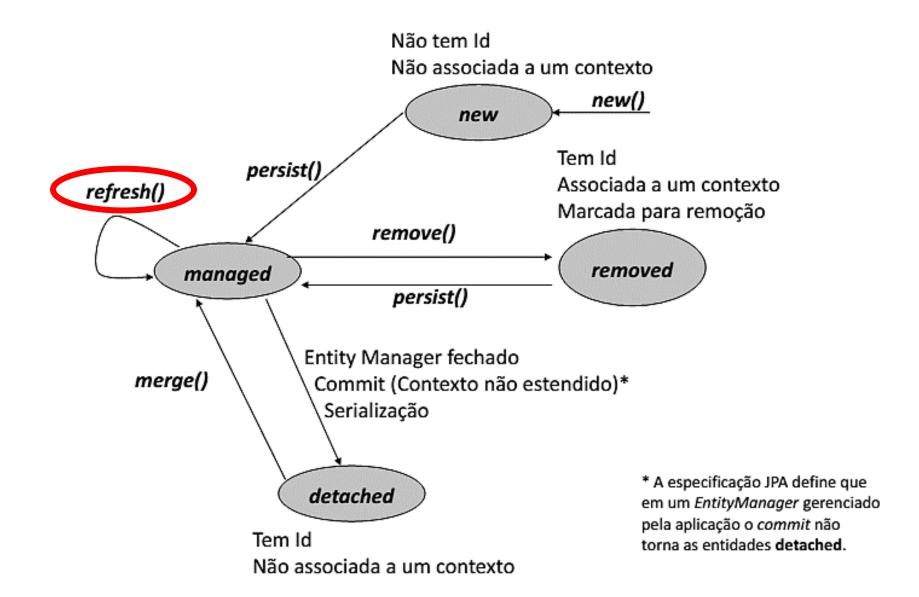
```
Veiculo veiculo = new Veiculo();
veiculo = manager.find(Veiculo.class, 1);
veiculo.setPlaca("DHZ-5678");
veiculo.setModelo("Fusca");
manager.refresh(veiculo);
```















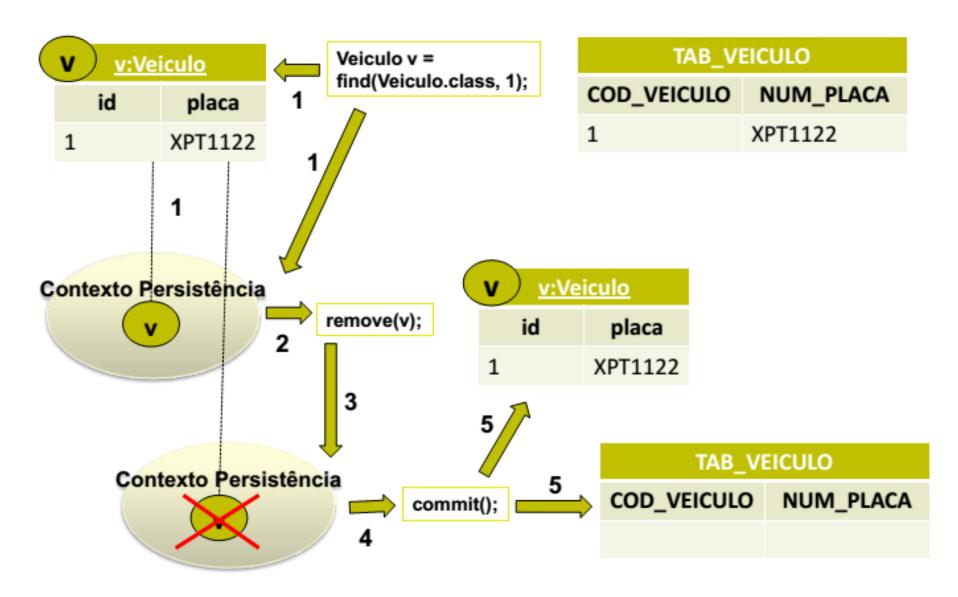
- remove(Object entidade): marca uma entidade como REMOVED que será excluída do banco de dados após uma instrução commit;
 - ✓ Caso a entidade seja NEW, ela é ignorada;
 - ✓ Caso a entidade seja **MANAGED**, ela torna-se **REMOVED**;
 - ✓ Caso a entidade seja **REMOVED** ela é ignorada;
 - ✓ Caso a entidade seja **DETACHED**, uma **IllegalArgumentException** é lançada;

Para excluir entidades desacopladas primeiro deve-se torná-la **MANAGED**, por exemplo, utilizando o método **find**;

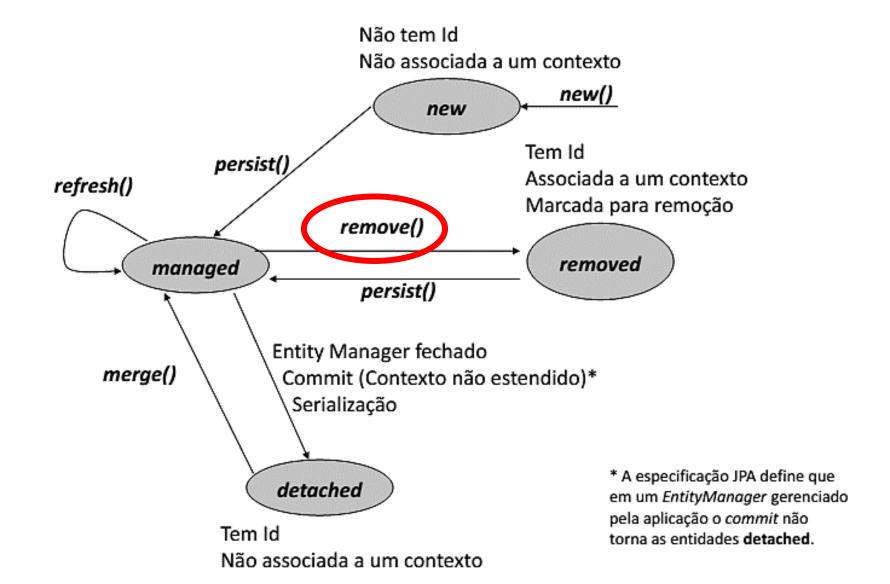
Veiculo veiculo = manager.find(Veiculo.class, 10);
manager.remove(veiculo);

REMOVE







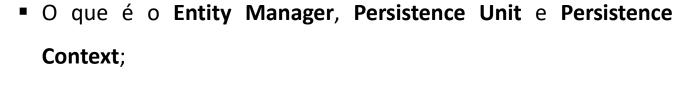


MÉTODOS DO CICLO DE VIDA



- Pode-se realizar ações em cada fase do ciclo de vida de uma entidade;
- Para tanto, basta utilizar as anotações abaixo:
 - ✓ @PrePersist
 - ✓ @PostPersist
 - ✓ @PreRemove
 - ✓ @PostRemove
 - ✓ @PreUpdate
 - ✓ @PostUpdate
 - ✓ @PostLoad

VOCÊ APRENDEU...



- Obter uma instância do Entity Manager e a controlar as transações;
- Sobre os estados da entidade e os métodos:
 - Persist;
 - Merge;
 - Find;
 - Refresh;
 - Remove;
- Métodos do ciclo de vida da entidade;





Copyright © 2024 – 2034 Prof. Dr. Marcel Stefan Wagner

Todos direitos reservados. Reprodução ou divulgação total ou parcial deste documento é expressamente proíbido sem o consentimento formal, por escrito, do Professor (autor).

Agradecimentos: Prof. Me Gustavo Torres Custódio | Prof. Me. Thiago T. I. Yamamoto