

**FIAP GRADUAÇÃO**

# ENTERPRISE APPLICATION DEVELOPMENT

*Prof. THIAGO T. I. YAMAMOTO*

#03 - JPA API



thiagoyama



thiagoyama@gmail.com

- ♦ Introdução JPA
- ♦ Entity Manager
- ♦ Contexto de Persistência
- ♦ Estados da Entidade
- ♦ Métodos da API

Especificação independente de fabricante;

Elementos principais:

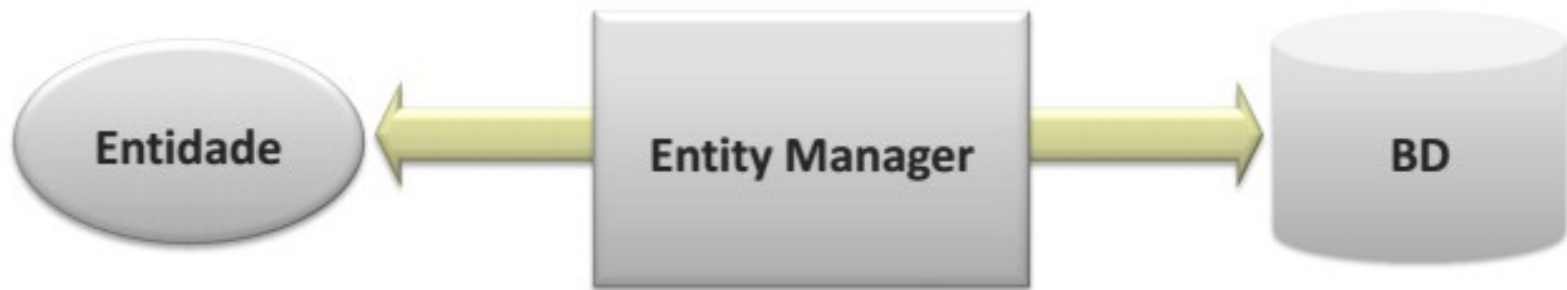
- Anotações para mapeamento O/R;
- API para persistência de entidades;
- Linguagem de consultas JPQL;

Implementação Hibernate:

- Hibernate Core;
- Hibernate Annotations;
- Hibernate Entity Manager;

# ENTITY MANAGER

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



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**:

1. A própria aplicação obtém um Entity Manager e o fecha utilizando métodos da API apropriados - *Application Managed Entity Manager*;
2. Um container instancia e gerencia o Entity Manager - *Container Managed Entity Manager*;

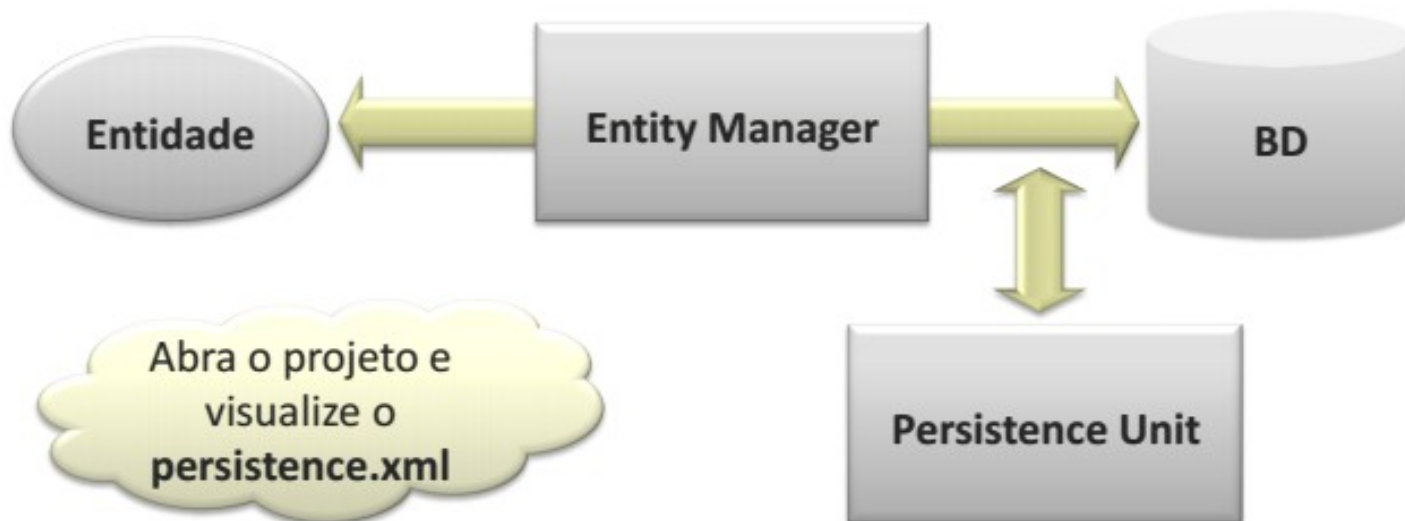
Vamos trabalhar neste material com o **Application Managed**.

# I PERSISTENCE UNIT

Uma **Persistence Unit** define todas as configurações necessárias para que um **Entity Manager** consiga efetuar a persistência de um conjunto de entidades;

Por exemplo, na **Persistence Unit** é onde definimos as Configurações de banco de dados que iremos acessar, a URL de conexão, etc...

Cada **Persistence Unit** é identificada por um nome dentro do arquivo **persistence.xml** na pasta **META-INF**;





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**;

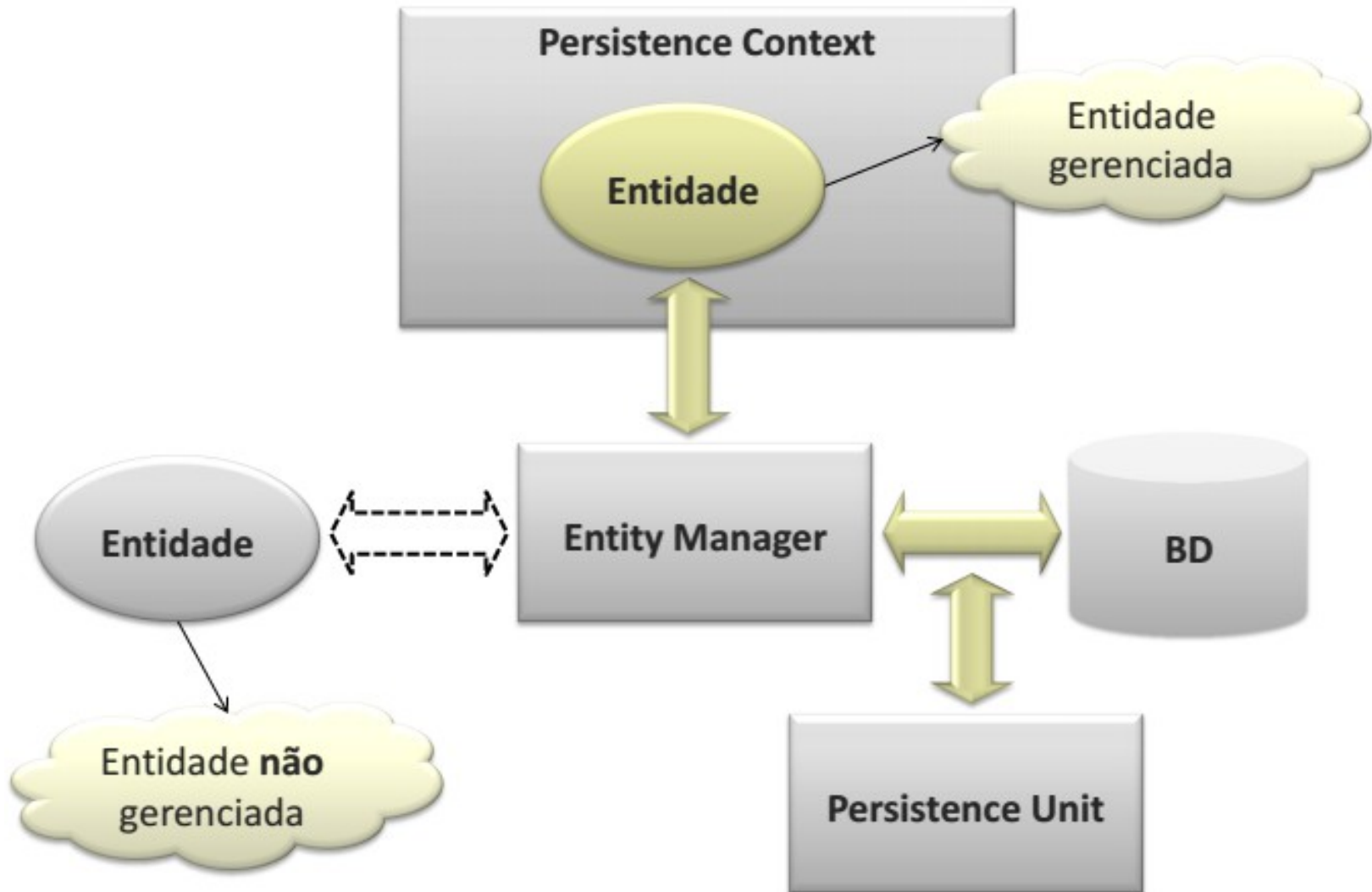
Ao fechar um contexto de persistência, todas suas instâncias de entidades associadas tornam-se não gerenciadas;

## 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 (por exemplo quando são instanciadas ou quando o Entity Manager é fechado);
- Alterações nas entidades não se refletem no banco de dados;



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

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

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();
```

Como o processo de obtenção de um EntityManagerFactory é custoso, faz-se necessário o uso do padrão de projeto Singleton para garantir que a instanciação ocorrerá apenas uma única vez;

Uma vez obtido um Entity Manager os métodos da JPA podem ser acionados;

Para fechar o Entity Manager basta utilizar o método close (lembre-se que todas as suas entidades gerenciadas passarão a ser não gerenciadas):

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

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);

Vamos utilizar neste material o **Resource Local**;

Para obter-se uma transação do Entity Manager utilizar o método **getTransaction()**

Uma transação representada pela classe **EntityTransaction** que contém os métodos principais:

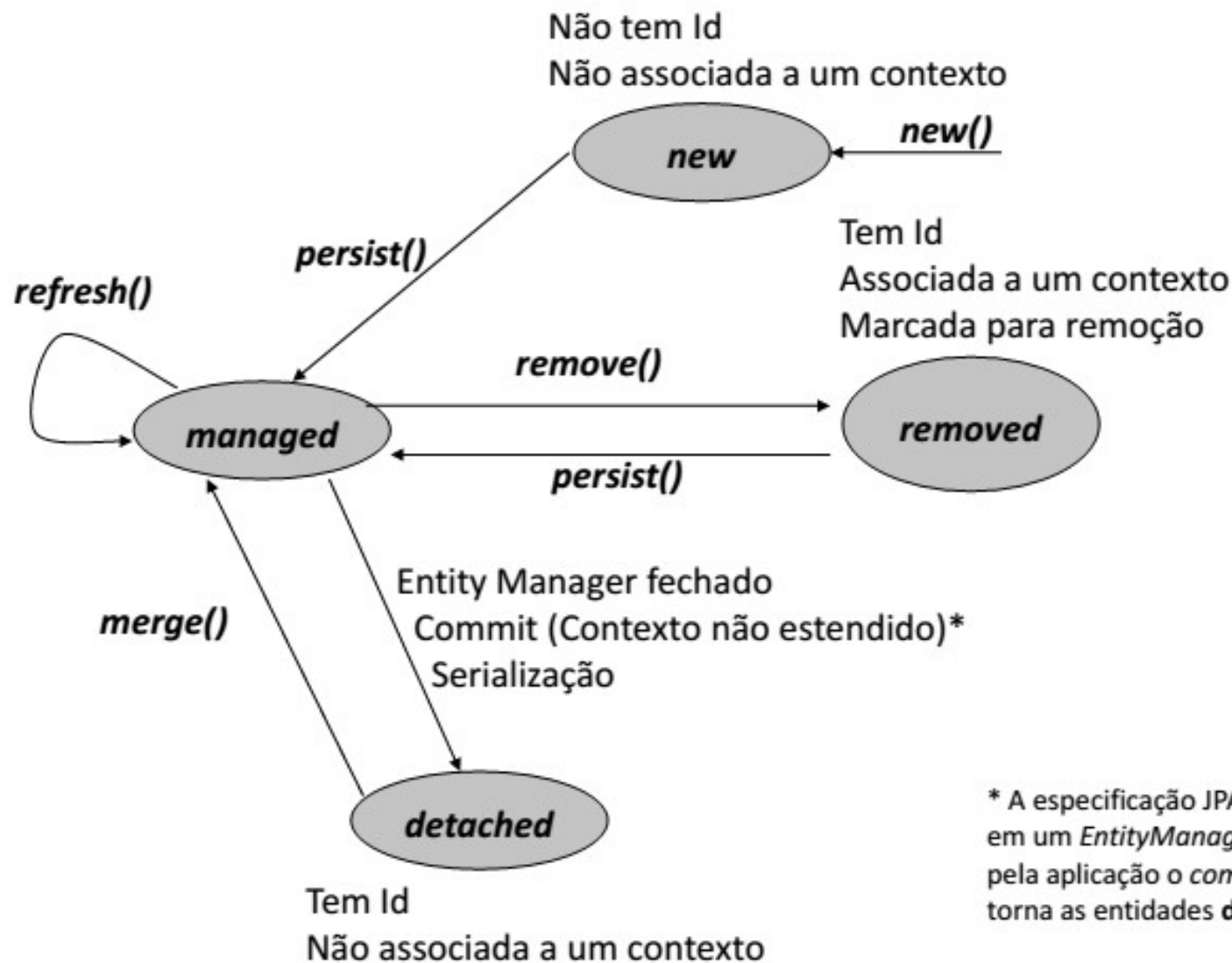
- **begin()**: delimita o início de uma transação (é obrigatório, um por vez)
- **commit()**: confirma a transação (sincroniza o contexto de persistência com a base de dados)
- **rollback()**: desfaz a transação
- **isActive()**: verifica se a transação encontra-se em andamento, isto é, não ocorreu commit 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 (mas já o foi um dia);
- **removed:** instância da entidade associada a um contexto de persistência mas está programada para ser removida do banco de dados;



# ENTITY MANAGER MÉTODOS



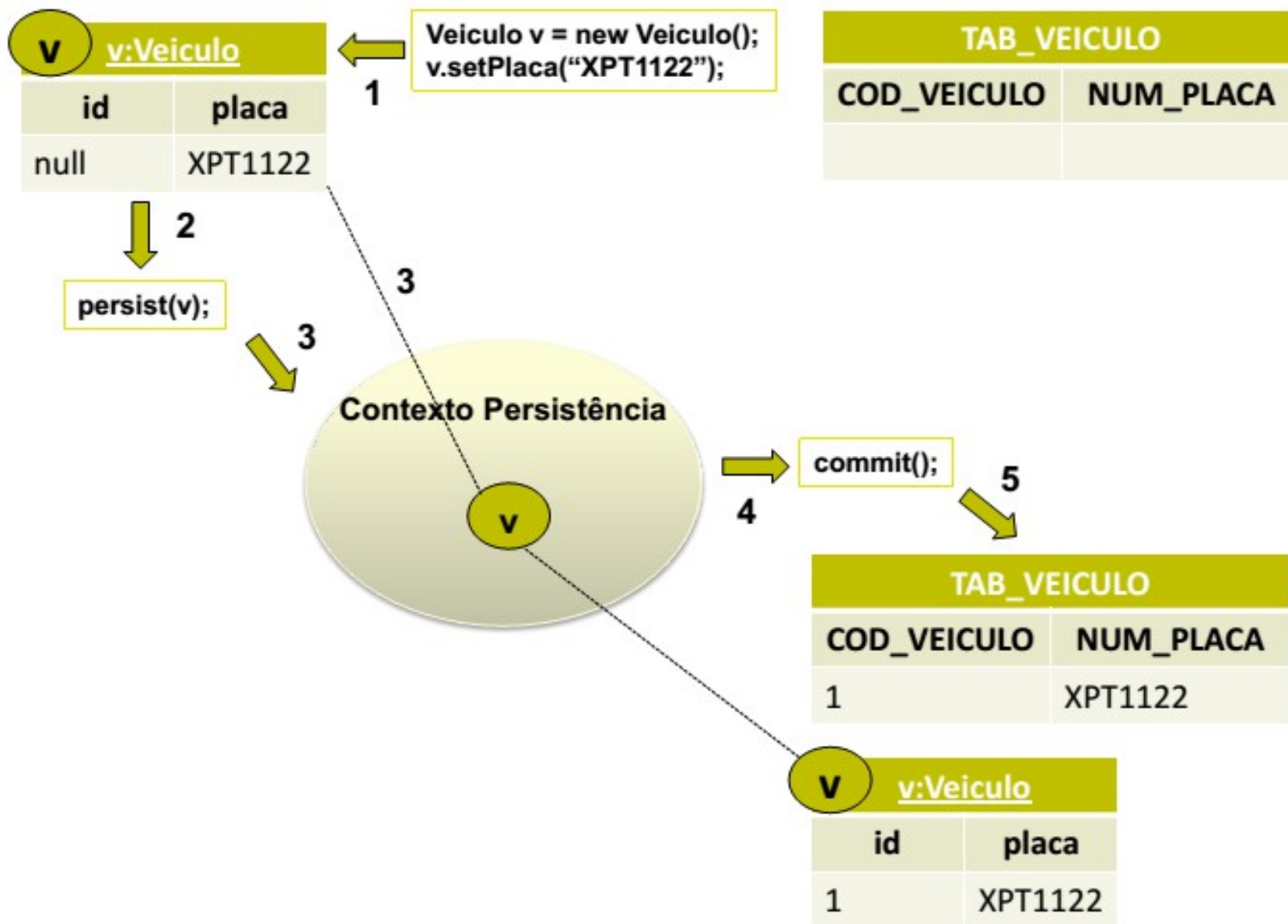
**persist (Object entity):** enfileira uma nova entidade para ser inserida no banco de dados e a torna gerenciada;

- 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

Exemplo:

```
VeiculoEntity veiculo = new VeiculoEntity();  
veiculo.setPlaca("DHZ-5678");  
veiculo.setModelo("Fusca");
```

```
manager.persist(veiculo);
```

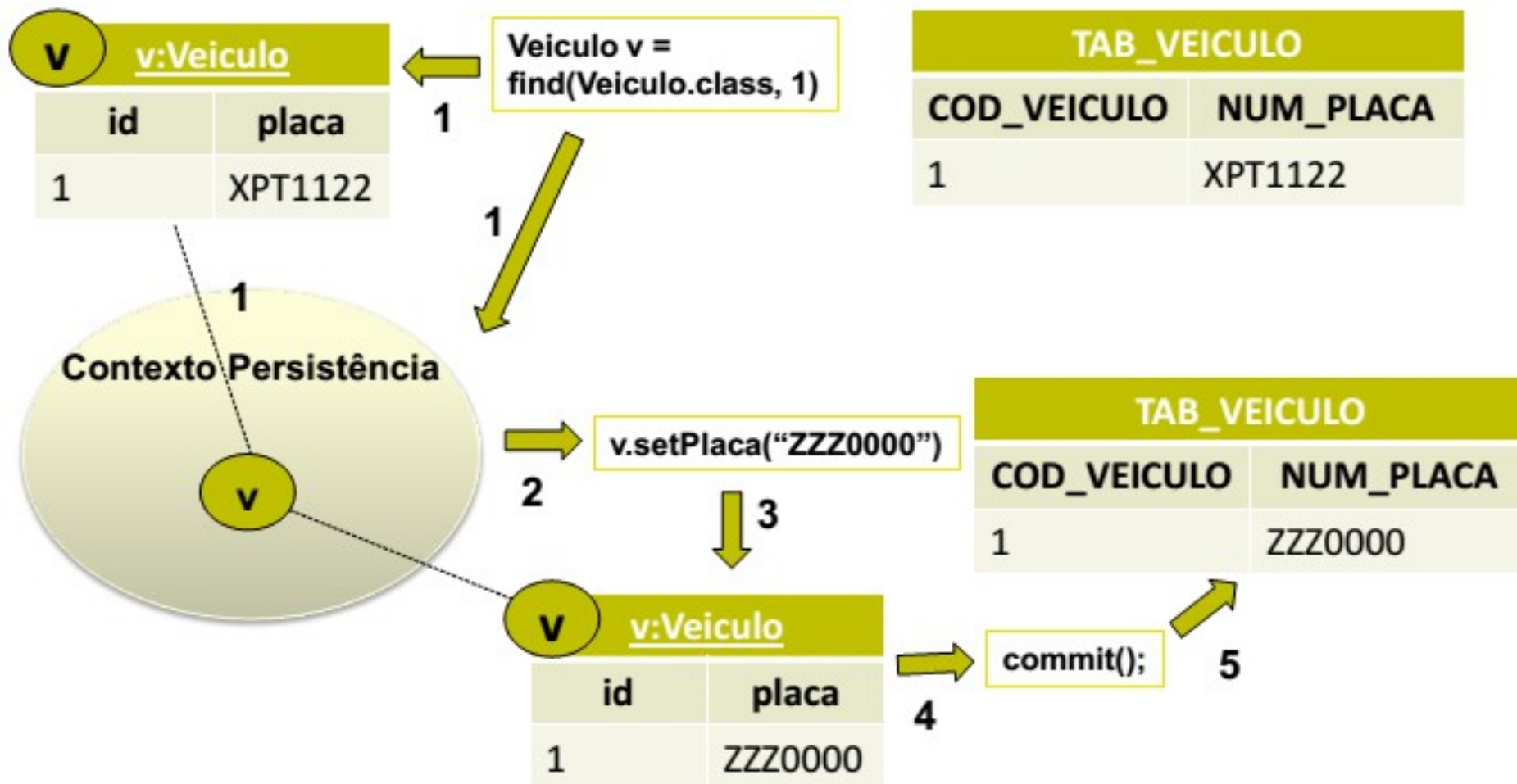


`<T> T find (Class<T> classeEntidade, Object PK)`: realiza uma busca por meio da chave primária da entidade.

- Retorna uma instância **MANAGED** caso seja localizada ou *null* caso contrário

### Exemplo:

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



<T> T merge (T entidade): retorna uma cópia gerenciada de uma entidade não gerenciada

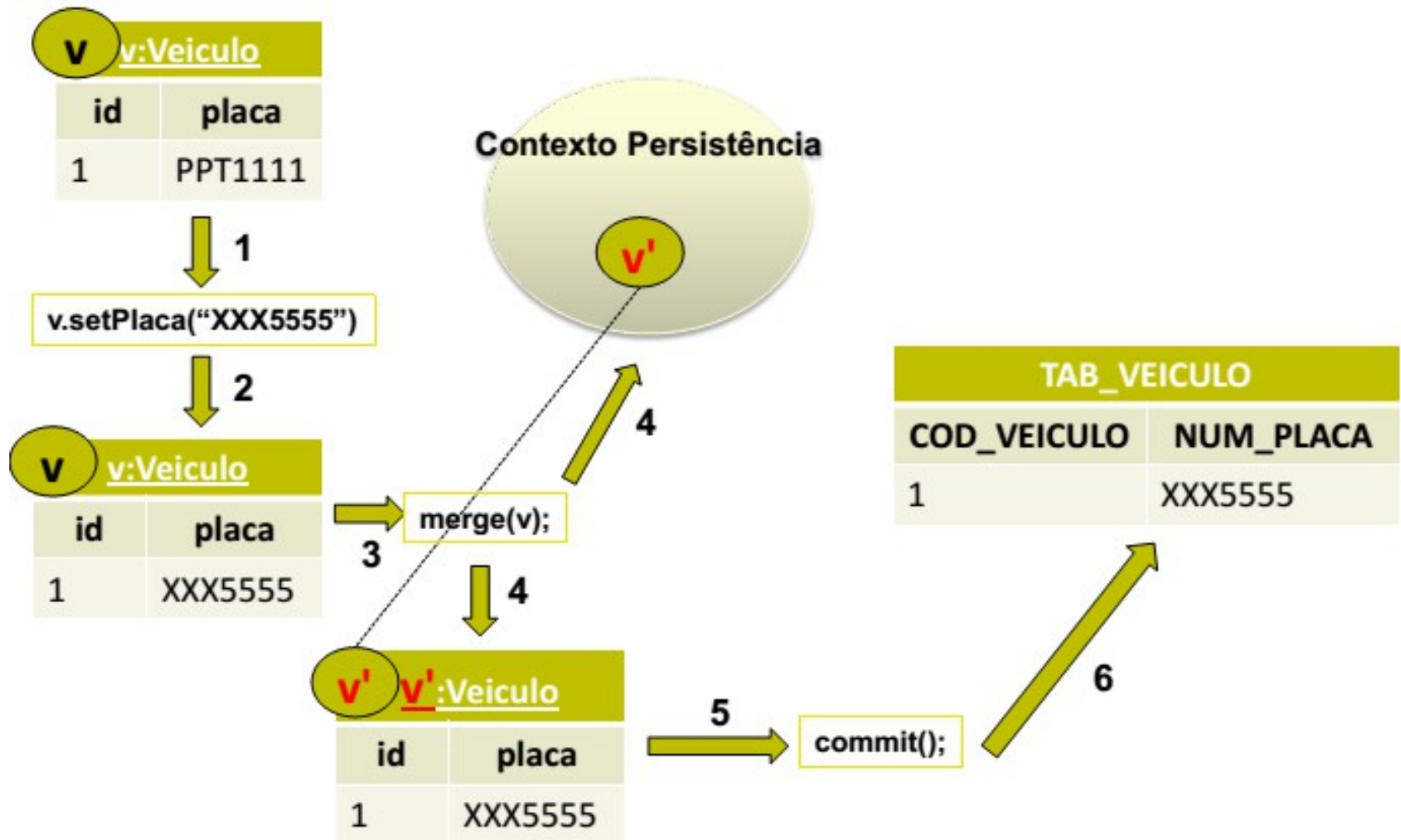
- Caso a entidade seja **DETACHED**, seu estado é copiado para uma instância **MANAGED** com a mesma identidade ou uma nova cópia **MANAGED** da entidade é criada
- Caso a entidade seja **NEW**, uma nova entidade **MANAGED** é criada com o estado copiado da entidade original
- Caso a entidade seja **MANAGED** ela é ignorada
- Caso a entidade seja **REMOVED**, uma **IllegalArgumentException** é lançada

**Exemplo:**

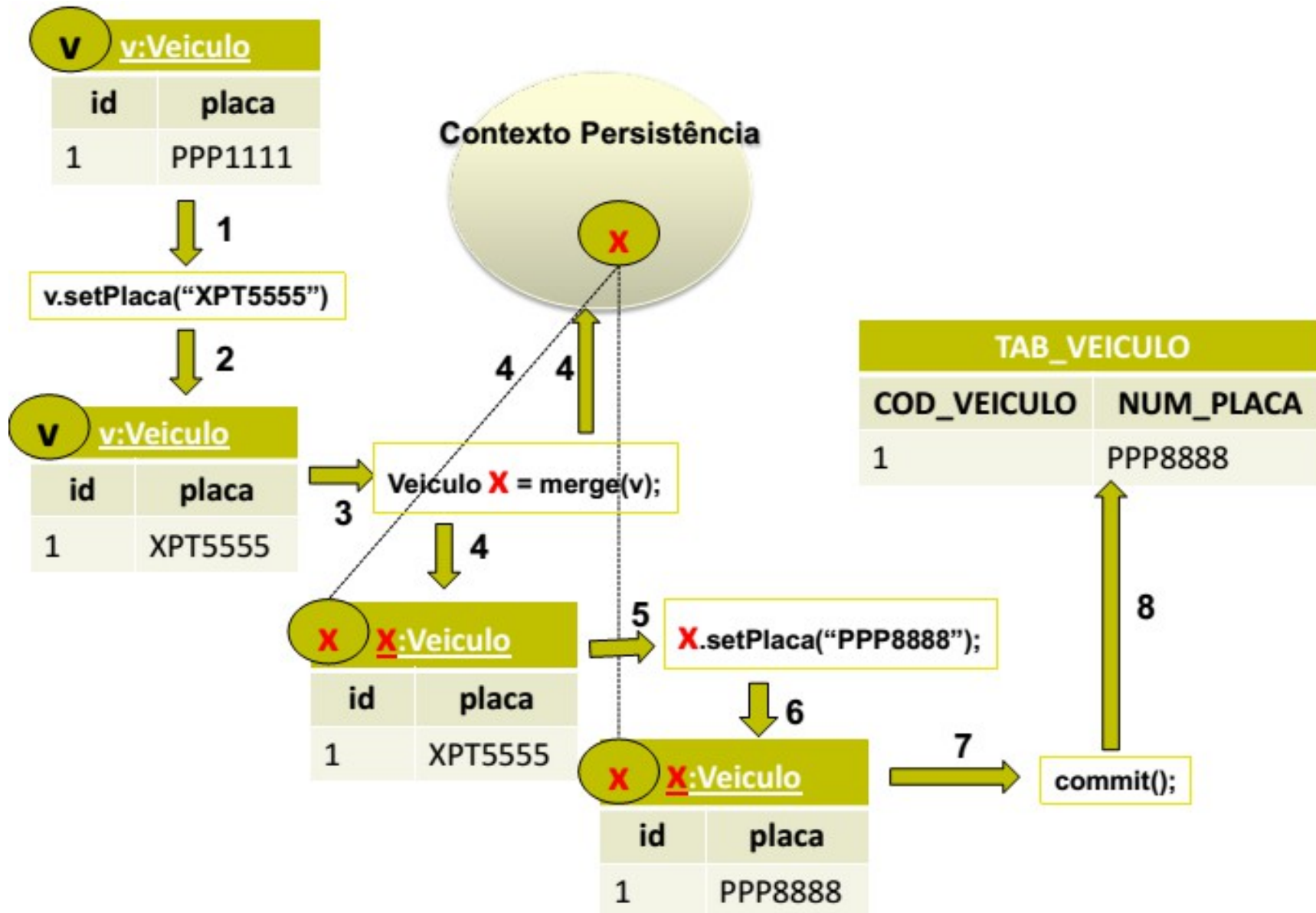
```
VeiculoEntity v2 = manager.merge(veiculo);
```

```
v2.setPlaca("DHZ-5678");  
v2.setModelo("Fusca");
```

# MERGE



# MERGE



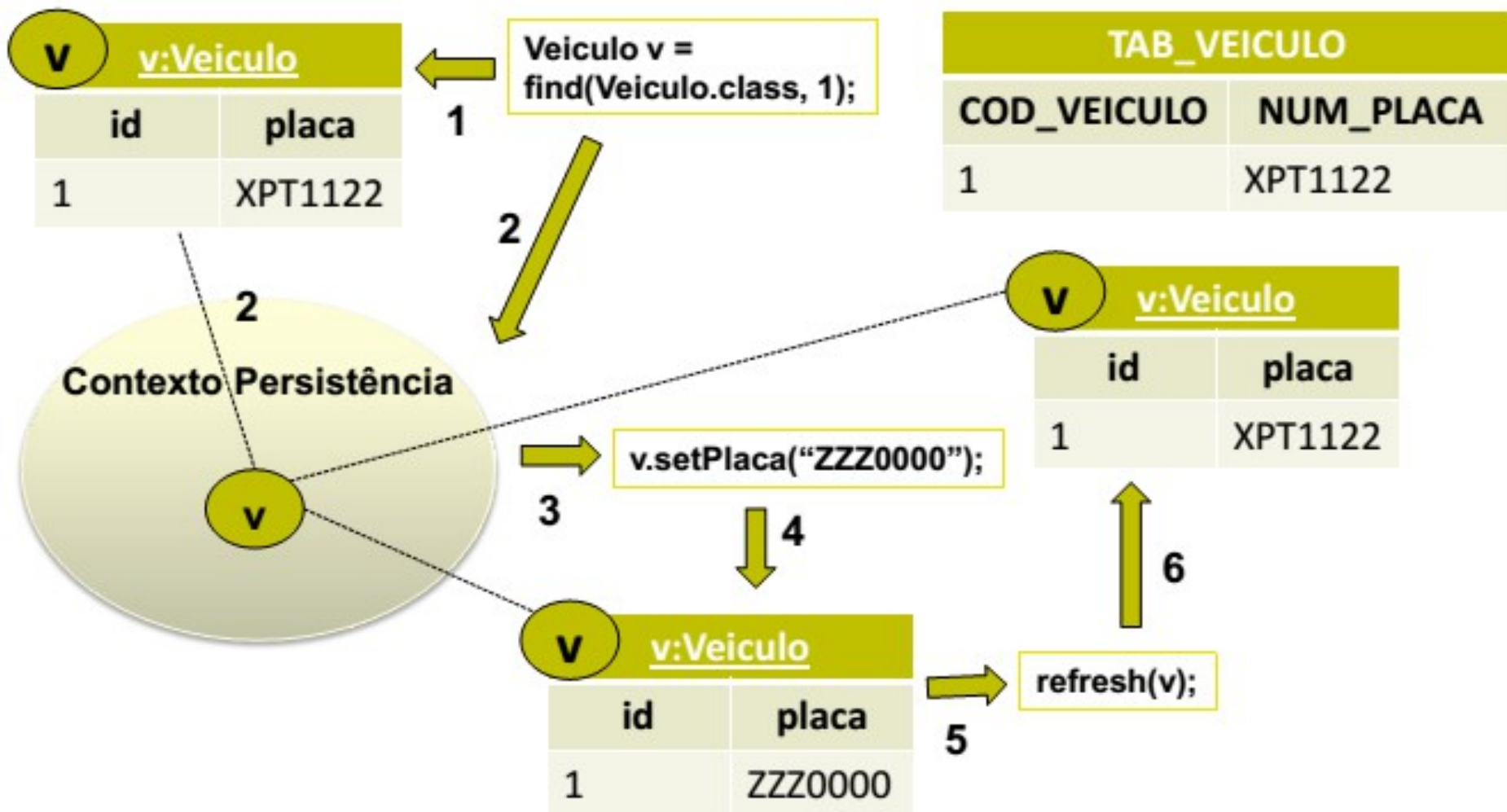
- **refresh(Object entidade)**: certifica que o estado da entidade encontra-se sincronizado com a base de dados;
- Caso a entidade seja **NEW** ela é ignorada;
- Caso a entidade seja **MANAGED**, seu estado é atualizado com a base de dados;
- Caso a entidade seja **REMOVED**, ela é ignorada;
- Caso a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

## Exemplo:

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



# REFRESH



**remove (Object entidade):** Identifica uma entidade como **REMOVED** que será excluída do banco de dados após uma instrução **commit** ou **flush**

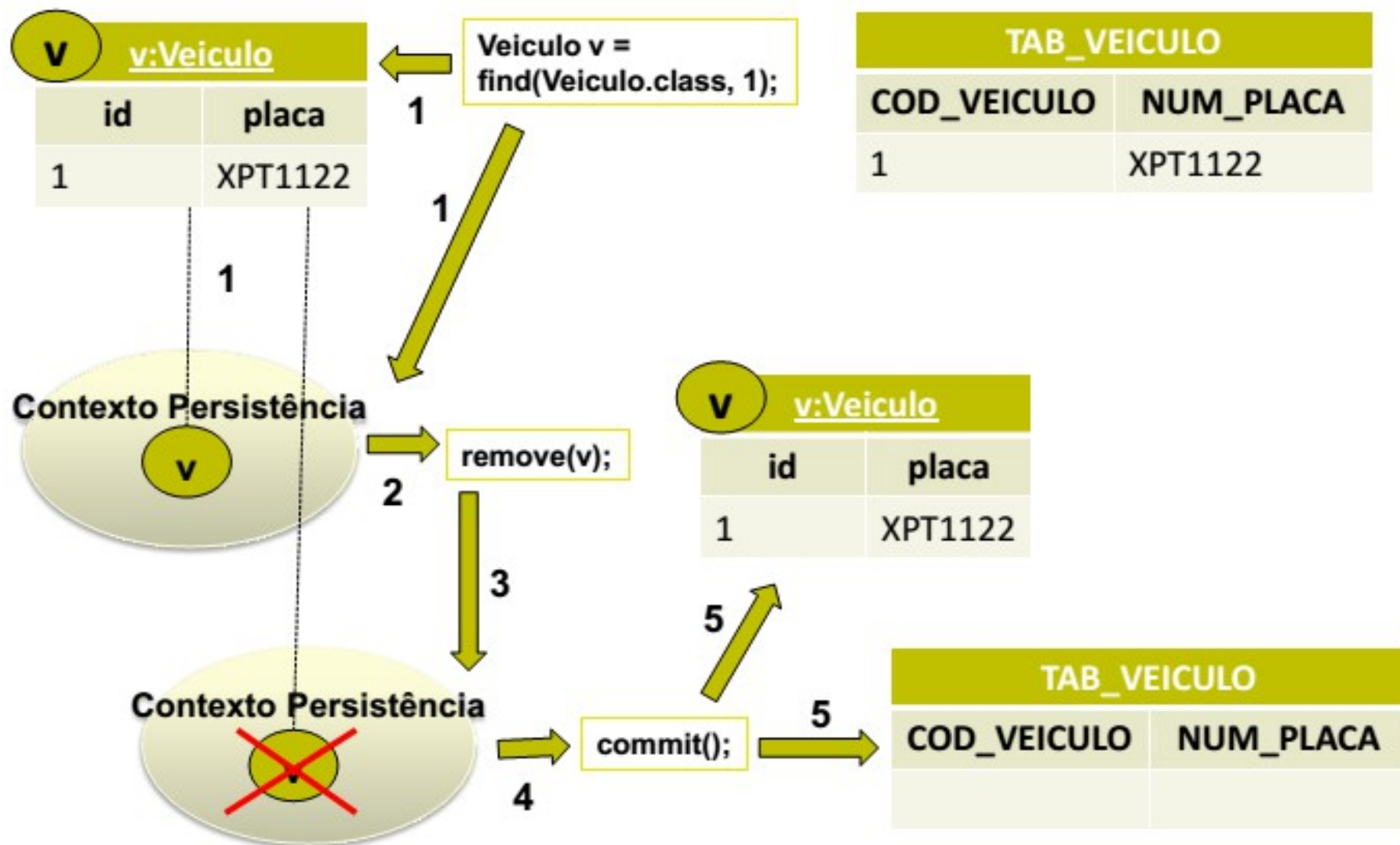
- Caso a entidade seja **NEW**, ela é ignorada;
- Caso a entidade seja **MANAGED**, ela torna-se **REMOVED**.
- Caso a entidade seja **REMOVED** ela é ignorada;
- Caso a entidade seja **DETACHED**, uma **IllegalArgumentException** é lançada;

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

**Exemplo:**

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

# REMOVE



- **clear()**: torna todas as entidades do persistence context DETACHED;
- **close()**: fecha o EntityManager;
- **isOpen()**: verifica se o EntityManager está aberto;
- **flush()**: sincroniza todo o persistence context com a base de dados;
- **contains(Object obj)**: verifica se o objeto obj pertence ao contexto de persistência;

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

Copyright © 2013 - 2017 Prof. Me. Thiago T. I. Yamamoto

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

*“Aprender é a única coisa que a mente nunca se cansa, nunca tem medo e nunca se arrepende”*