Gerenciando entidades com o EntityManager

DSC AULA 4

EntityManager

- Vamos aprender sobre o ciclo de vida de entidades e como o EntityManager pode ser usado para realizar operações CRUD.
- O EntityManager funciona como uma ponte entre o mundo OO e Relacional
  - Ele interpreta o mapeamento objeto relacional de uma entidade e salva a entidade no banco de dados.

Principais métodos

public void persist(Object entity);	Saves (persists) an entity into the database and makes the entity managed
public <T> T merge(T entity);	Merges an entity to the EntityManager's persistence context and returns the merged entity.
public <T> T find(Class<T> entityClass, Object primaryKey);	Finds an entity instance by its primary key. This method is overloaded and comes in different forms.
public void flush();	Synchronizes the state of entities in the EntityManager's persistence context with the database.
public void remove(Object entity);	Removes an entity from the database.

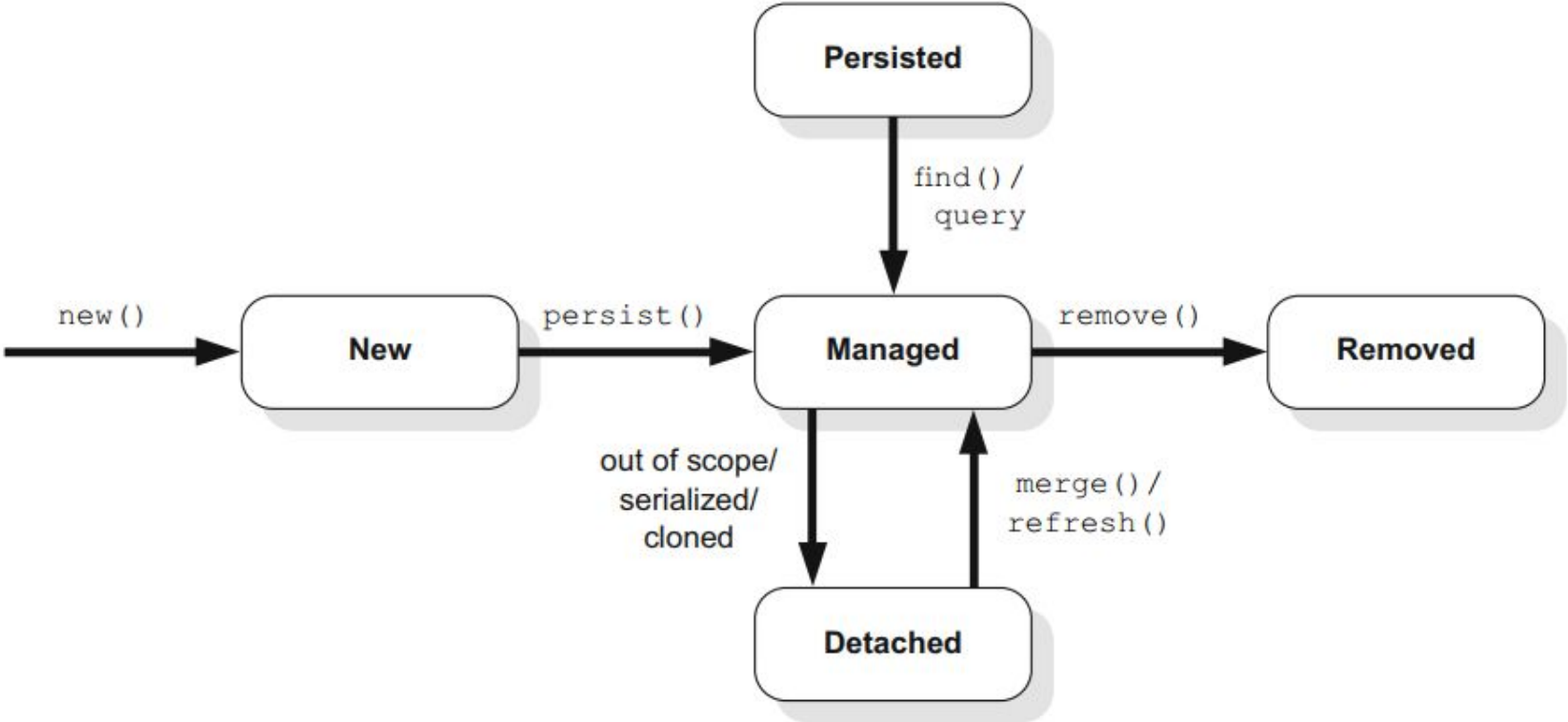
Principais métodos

public void refresh(Object entity);	Refreshes (resets) the entities from the database. This method is overloaded and comes in different forms.
public Query createQuery(String jpqlString);	Creates a dynamic query using a JPQL statement. This method is overloaded and comes in different forms.
public Query createNamedQuery(String name);	Creates a query instance based on a named query on the entity instance. This method is overloaded and comes in different forms.
public EntityTransaction getTransaction();	Retrieves a transaction object that can be used to manually start or end a transaction.
public void close();	Closes an application-managed EntityManager

O ciclo de vida de uma entidade

- O EntityManager nada sabe sobre um POJO, até que você diga ao manager para começar a tratar o POJO como uma entidade JPA.
- Uma entidade sob o controle do EntityManager é considerada gerenciada (managed).
- Por outro lado, quando um EntityManager para de gerenciar uma entidade, a entidade é dita desanexada (detached).

O ciclo de vida de uma entidade



Entidades gerenciadas (managed)

- O EntityManager assegura que os dados das entidades estejam sincronizados com o banco de dados.
  - Quando o EntityManager inicia o gerenciamento de uma entidade, ele sincroniza o estado dela com o banco de dados.
  - Além disso, o EntityManager garante que alterações feitas nos dados da entidade sejam refletidas no banco de dados.
- Uma entidade torna-se gerenciada quando ela é passada como parâmetro dos métodos persist, merge, ou refresh.
  - Quando recuperada através do método find ou através de queries, a entidade também torna-se gerenciada.

Entidades desanexada (detached)

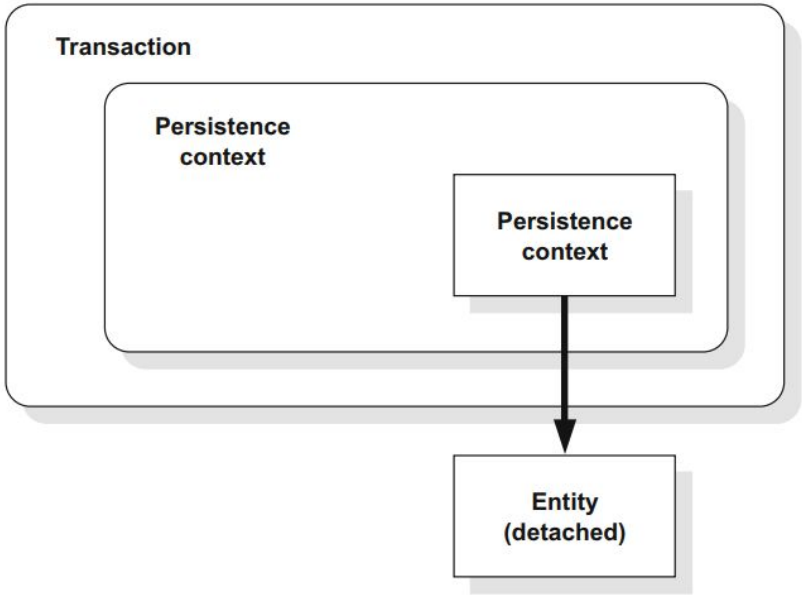
- É uma entidade que não é mais gerenciada pelo EntityManager.
  - Não há garantia de que o estado da entidade esteja em sincronia com o banco de dados.
  - Uma chamada ao método remove do EntityManager, por exemplo, torna uma entidade desanexada.
  - Ao invocar o método clean todas as entidades gerenciadas pelo EntityManager passam a ser desanexadas.
  - Instâncias clonadas ou serializadas também são desanexadas.

Contextos de persistência, escopo e EntityManager

- O escopo de persistência é a duração de tempo que um determinado conjunto de entidades permanece gerenciado.
- Um contexto de persistência é uma coleção de entidades gerenciadas por um EntityManager durante um determinado escopo de persistência.
- Há dois tipos de escopos de persistência: de transação (transaction) e estendido (extended).
    - Neste curso, veremos apenas o escopo de transação.

Escopo de transação

- Se um contexto de persistência estiver sob o escopo de transação, as entidades agregadas durante uma transação serão automaticamente desanexadas quando a transação finalizar (com sucesso ou com falha).



Operações de persistência

- Vamos realizar as operações de inserção, recuperação, atualização e deleção (CRUD) utilizando o EntityManager.
- Todas as operações de persistência devem ser invocadas dentro do escopo de uma transação.
    - Se uma transação não está presente quando um método de persistência é invocado, é lançada a transação TransactionRequiredException.
- Operações de persistência: *persist, flush, merge, refresh e remove*.

Persistindo Entidades

```
Usuario comprador = criarComprador();
EntityManager em = null;
try {
    em = emf.createEntityManager();
    EntityTransaction et = em.getTransaction();
    et.begin();
    em.persist(comprador);
    et.commit();
} finally {
    if (em != null) {
        em.close();
    }
```

O que será gerado ao persistir um Comprador

- Os dados da entidade Comprador serão inseridos em pelo menos 3 tabelas
  - TB_USUARIO
  - TB_COMPRADOR
  - TB_TELEFONE
- Além disso, há ainda a entidade CartaoCredito, cujos dados são inseridos na tabela
  - TB_CARTAO_CREDITO

INSERT’S gerados pelo EclipseLink

- INSERT INTO TB_CARTAO_CREDITO (TXT_BANDEIRA, DT_EXPIRACAO, TXT_NUMERO) VALUES (?, ?, ?)

- INSERT INTO TB_USUARIO (TXT_CPF, DT_NASCIMENTO, TXT_EMAIL, TXT_LOGIN, TXT_NOME, TXT_SENHA, END_TXT_BAIRRO, END_TXT_CEP, END_TXT_CIDADE, END_TXT_COMPLEMENTO, END_TXT_ESTADO, END_TXT_LOGRADOURO, END_NUMERO, DISC_USUARIO) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

INSERT’s gerados pelo EclipseLink

- INSERT INTO TB_COMPRADOR (ID_CARTAO_CREDITO, ID_USUARIO) VALUES (?, ?)

- INSERT INTO TB_TELEFONE (ID_USUARIO, TXT_NUM_TELEFONE) VALUES (?, ?)

- INSERT INTO TB_TELEFONE (ID_USUARIO, TXT_NUM_TELEFONE) VALUES (?, ?)

Recuperando entidades pelo Id

- Observe que, se uma instância de entidade com a chave informada não existir no banco de dados, o método find vai retornar *null*.
- Se o seu provedor de persistência suporta caching e a entidade já existe na cache, então o EntityManager retorna a instância em *cache* da entidade em vez de recuperá-la a partir do banco de dados.

```
Comprador comprador = em.find(Comprador.class, 9L);
```

Lendo entidades relacionadas

- O EntityManager normalmente carrega todos os dados de instância da entidade quando uma entidade é recuperada a partir do banco de dados.
- Como já vimos, existem dois tipos de fetch: FetchType.EAGER e FetchType.LAZY.
    - Lazy permite que uma entidade só seja carregada no momento em que for necessária.

```
@Entity
@Table(name="TB_COMPRADOR")
@DiscriminatorValue(value = "C")
@PrimaryKeyJoinColumn(name="ID_USUARIO", referencedColumnName = "ID")
public class Comprador extends Usuario implements Serializable {
    @OneToOne(fetch = FetchType.LAZY, cascade = CascadeType.ALL, optional = false)
    @JoinColumn(name = "ID_CARTAO_CREDITO", referencedColumnName = "ID")
    private CartaoCredito cartaoCredito;
```

Comportamento padrão para relacionamentos

Relationship type	Default fetch behavior	Number of entities retrieved
One-to-one	`EAGER`	Single entity retrieved
One-to-many	`LAZY`	Collection of entities retrieved
Many-to-one	`EAGER`	Single entity retrieved
Many-to-many	`LAZY`	Collection of entities retrieved

Atualizando entidades

- Dentro de uma transação, não é necessário chamar explicitamente métodos para atualizar objetos que estejam gerenciados.
- Ao realizar o commit na transação, o provedor de persistência irá atualizar os objetos modificados.

```
et.begin();
Comprador comprador = em.find(Comprador.class, 1L);
comprador.setEmail("teste@gmail.com");
comprador.addTelefone("(81)990908787");
et.commit();
```

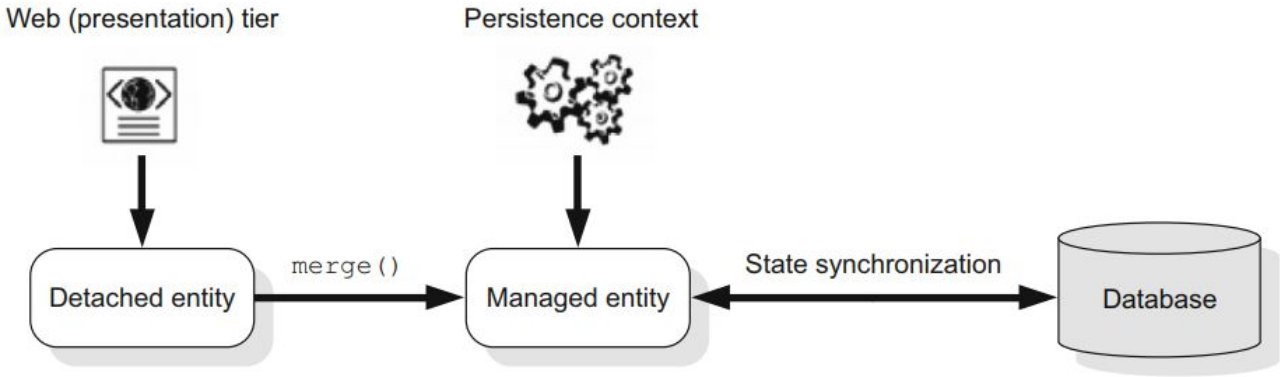
Atualizando entidades

- Caso o objeto não seja gerenciado, é necessário chamar o método merge para que ele passe a ser gerenciado e, assim, a instância possa ser atualizada na base de dados.

```
et.begin();
/**
 * Considere que o comprador foi recuperador por outra
 * instância de entity manager.
 */
comprador.setEmail("teste@gmail.com");
comprador.addTelefone("(81)990908787");
comprador = em.merge(comprador);
et.commit();
```

Atualizando entidades

- É comum em aplicações web, uma instância de *entitymanager* recuperar a entidade e outra atualizar.
  - Nesses casos, o método *merge* é indispensável



Atualizando entidades

- Observe, ainda o exemplo abaixo
  - Tanto o comprador quanto o cartão de crédito serão atualizados através do método *merge* porque o mapeamento de cartaoCredito em comprador possui CascadeType.ALL

```
et.begin();
/**
 * Considere que o comprador foi recuperador por outra
 * instância de entity manager.
 */
comprador.setEmail("teste@gmail.com");
comprador.addTelefone("(81)990908787");
CartaoCredito cartaoCredito = comprador.getCartaoCredito();
cartaoCredito.setBandeira("MASTER");
comprador = em.merge(comprador);
et.commit();
```

Atualizando entidades

- CascadeType.ALL significa que todas as operações realizadas sobre o comprador serão realizadas sobre cartão de crédito.
  - persist(comprador), merge(comprador), remove(comprador), refresh(comprador) também serão propagadas para a instância da entidade CartaoCredito associada.

```
@PrimaryKeyJoinColumn(name = "ID_USUARIO", referencedColumnName = "ID_USUARIO")
public class Comprador extends Usuario implements Serializable {

    @OneToOne(fetch = FetchType.LAZY, cascade = CascadeType.ALL, optional = true)
    @JoinColumn(name = "ID_CARTAO_CREDITO", referencedColumnName = "ID_CARTAO_CREDITO")
    private CartaoCredito cartaoCredito;
```

Efeito de vários valores de CascadeType

CascadeType.MERGE	Somente operações EntityManager.merge são propagadas para entidades relacionadas.
CascadeType.PERSIST	Somente operações EntityManager.persist são propagadas para entidades relacionadas.
CascadeType.REFRESH	Somente operações EntityManager.refresh são propagadas para entidades relacionadas.
CascadeType.REMOVE	Somente operações EntityManager.remove são propagadas para entidades relacionadas.
CascadeType.ALL	Todas as operações são propagadas para entidades relacionadas.

Removendo entidades

- Para remover entidades, utilize o método *remove* do EntityManager.
- Dê atenção especial aos relacionamentos, mais especificamente ao CascadeType.
- Para ocorrer a remoção de uma entidade relacionada, é necessário *CascadeType.ALL* ou *CascadeType.REMOVE*.

Removendo entidades

- O exemplo de código a seguir resulta na remoção uma instância de categoria do banco de dados

```
et.begin();
Categoria categoria = em.find(Categoria.class, 2L);
em.remove(categoria);
et.commit();
```

Para Baixar

- https://svn.riouxsvn.com/dsc-ifpe/exemplo_07/
- Verificar classes de teste VendedorTeste e CompradorTeste