JAVA SPRING DATA

Prof. Júlio Machado

julio.machado@pucrs.br

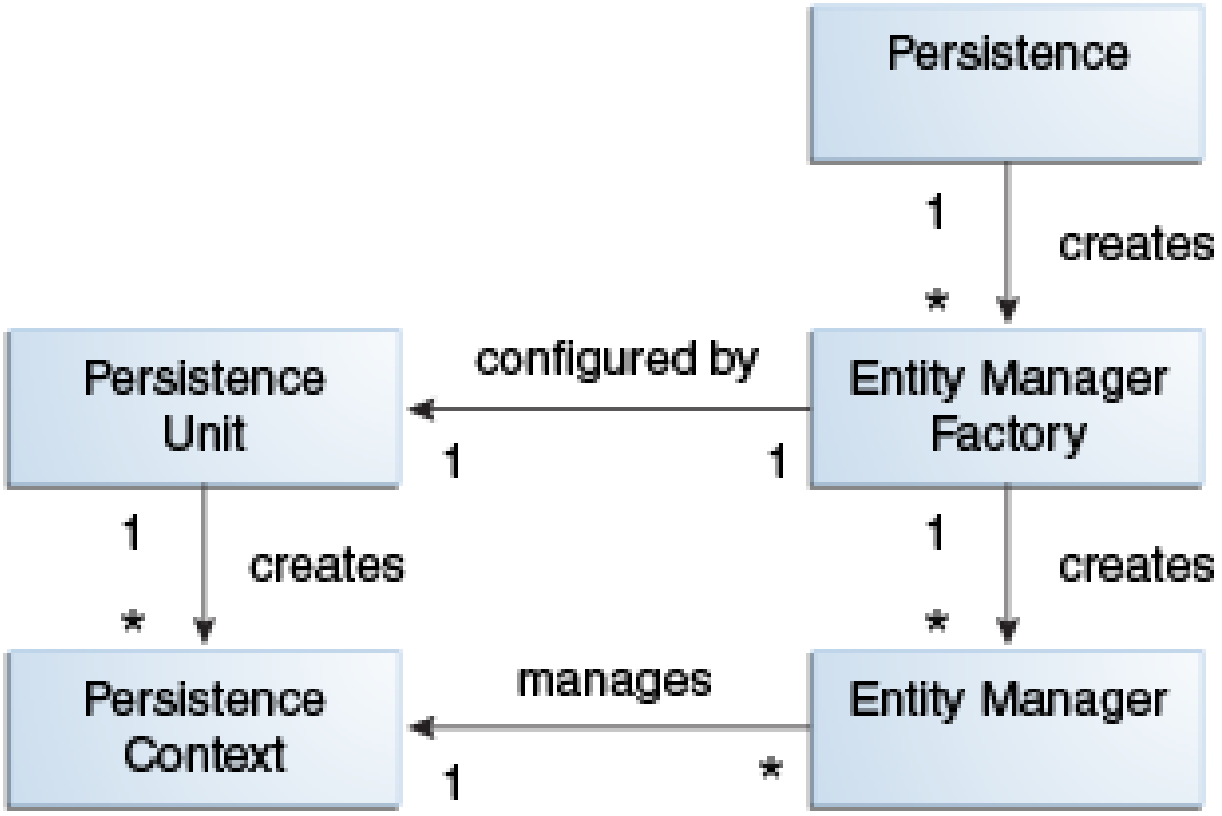
JPA

Acesso aos Dados

JPA - EntityManager

- JPA define um gerenciador de entidades ***EntityManager*** como responsável pelas operações de persistência dos objetos
  - Cada instância do *EntityManager* é associada a um “contexto de persistência” (um conjunto de entidades mapeadas para uma base de dados particular)
- O gerenciador fornece então métodos para manipular o estado dos objetos associados ao contexto
  - Criação
  - Remoção
  - Atualização
  - Busca

JPA - EntityManager



JPA - EntityManager

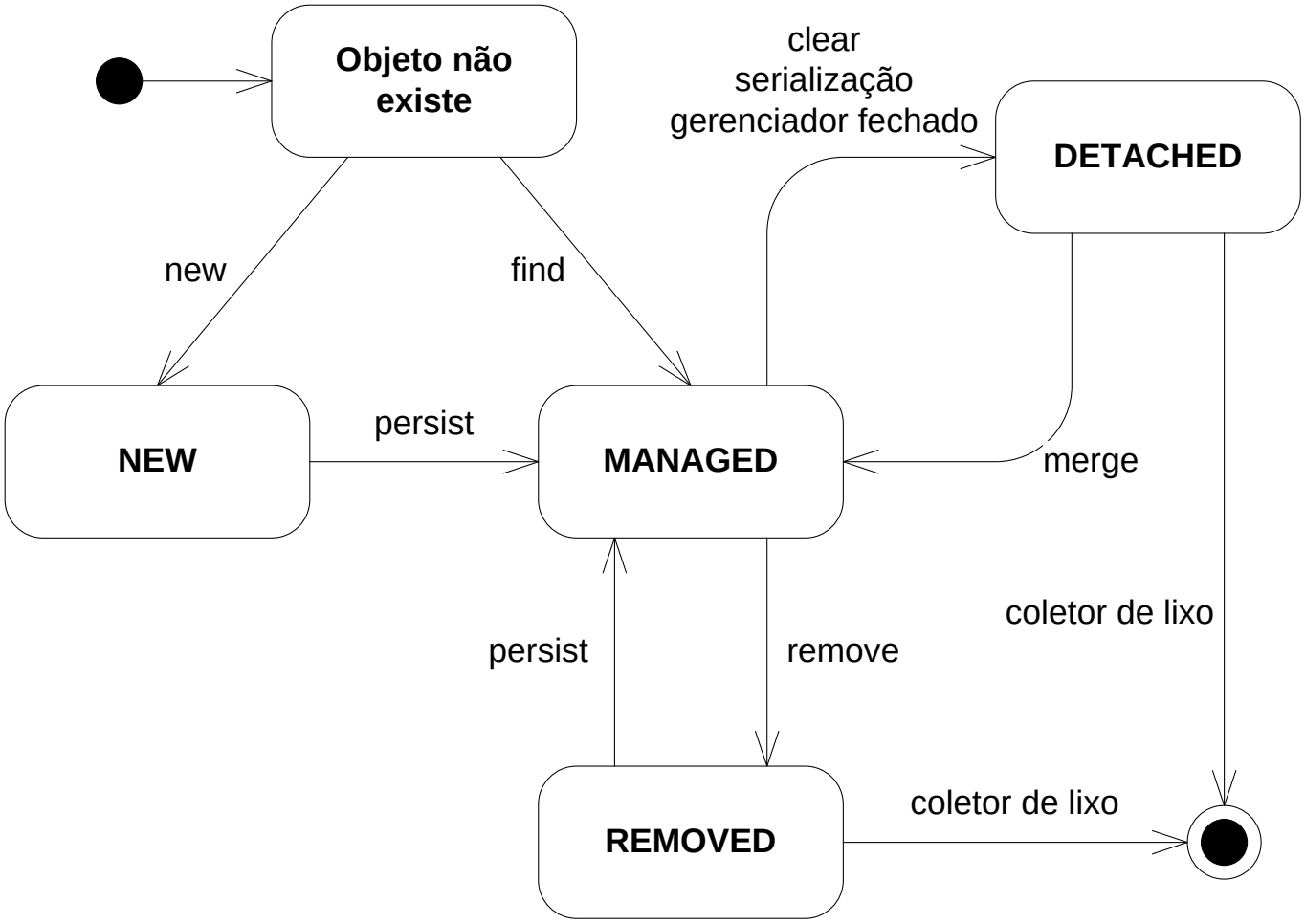
- Gerenciamento pelo contêiner
  - Injeção de dependência via anotação *@PersistenceContext*
  - O contêiner propaga a contexto de persistência para todos os componentes que utilizam a instância do *EntityManager* dentro de uma mesma transação JTA
  - Ex.:

```
@PersistenceContext
private EntityManager em;
```

JPA - Operações de Persistência

- Ciclo de vida das entidades:
  - **NEW** – são entidades que ainda não possuem uma identidade persistente e não estão ainda associadas a um contexto de persistência
  - **MANAGED** – são entidades que possuem uma identidade persistente e estão associadas a um contexto de persistência
  - **DETACHED** – são entidades que possuem uma identidade persistente e não estão atualmente associadas a um contexto de persistência
  - **REMOVED** – são entidades que possuem uma identidade persistente, estão associadas a um contexto de persistência e estão marcadas para remoção da base de dados

JPA - Operações de Persistência



JPA - Operações de Persistência

- Método **find**():
  - Busca imediatamente uma entidade com base na chave primária
    - Retorna um objeto MANAGED
  - Ex.:

```
Editora editora =
  entityManager.find(Editora.class, codigo);
```

  - Não requer transação ativa (operação de consulta)

JPA - Operações de Persistência

- Método **getReference**():
  - Busca quando necessário (por exemplo, ao acessar uma propriedade get) a entidade com base na chave primária
    - Retorna um objeto MANAGED
  - Ex.:

```
Editora editora =
  entityManager.getReference(Editora.class,
  codigo);
String nome = editora.getNome();
```

  - Não requer transação ativa (operação de consulta)

JPA - Operações de Persistência

- Método **persist**():
  - Armazena uma nova entidade na base de dados
    - Se a entidade já é MANAGED a operação não tem efeito
    - É propagada de acordo com a configuração de *cascade*
  - Deve ser realizado em uma transação ativa
    - A operação no BD somente é completada na realização de um *commit*
  - Ex.:

```
Autor autor = new Autor();
autor.setPrimeiroNome(“Julio”);
autor.setUltimoNome(“Machado”);
entityManager.persist(autor);
```

JPA - Operações de Persistência

- Método **remove**():
  - Remove uma entidade da base de dados com base na chave primária
    - Entidade a ser removida deve ser MANAGED e passa a ser REMOVED
    - Removido da base de dados quando a transação é completada ou operação *flush()*
    - É propagada de acordo com a configuração de *cascade*
  - Deve ser realizado em uma transação ativa
    - A operação no BD somente é completada na realização de um *commit*
  - Ex.:

```
Editora editora =
  entityManager.find(Editora.class, codigo);
entityManager.remove(editora);
```

JPA - Operações de Persistência

- Método **flush**():
  - Realiza atualização do banco de dados e do contexto de persistência
  - Deve ser realizado em uma transação ativa
  - É propagada de acordo com a configuração de *cascade*
  - Diversos usos:
    - Permitir que uma busca seja realizada sobre dados mais recentes
    - Inserir objetos e obter valor atual de chaves primárias auto-geradas
    - Permitir um tratamento mais fino sobre exceções e erros
    - Evitar erros de integridade referencial
  - Ex.:

```
Autor autor = new Autor();
autor.setPrimeiroNome(“Julio”);
autor.setUltimoNome(“Machado”);
entityManager.persist(autor);
entityManager.flush();
int id = autor.getCodigo();
```

JPA - Operações de Persistência

- Método **refresh**():
  - Atualiza uma entidade do contexto com as informações do banco de dados
    - Entidade a ser atualizada deve ser MANAGED
  - Ex.:

```
Editora editora =
  entityManager.find(Editora.class, codigo);
entityManager.refresh(editora);
```

JPA - Operações de Persistência

- Método **merge**():
  - Permite que o estado de uma entidade DETACHED seja propagado para a respectiva entidade MANAGED
    - Útil para reintroduzir entidades no contexto de persistência, por exemplo se a entidade foi serializada, clonada ou cuja fonte é outro EntityManager
    - Qualquer alteração sobre a entidade agora MANAGED será persistida no banco de dados quando a transação realizar commit ou operação flush()
  - Deve ser realizado em uma transação ativa
  - Ex.:

```
editora = entityManager.merge(editora);
editora.setNome(“Novo Nome”);
```

JPA - Operações de Persistência

- Método **detach**():
  - Desanexa um determinado objeto do contexto de persistência
  - Ex.:

```
Editora editora =
  entityManager.find(Editora.class, codigo);
entityManager.detach(editora);
```

JPA - Operações de Persistência

- Método **clear**():
  - Desanexa todos os objetos no contexto de persistência do **Entity Manager**
  - Ex.:

```
entityManager.clear();
```

JPA - Operações de Persistência

- Método **close**():
  - Fecha e libera os recursos utilizados pelo **Entity Manager**
  - Utilizar somente quando o gerenciamento do objeto for realizado pela própria aplicação
  - Ex.:

```
entityManager.close();
```

JPA - Operações de Consulta

- JPA fornece diversos meios para a consulta de entidades:
  - Jakarta Persistence Query Language (JPQL)
    - Linguagem textual baseada em SQL
  - Criteria API
    - API para a construção de objetos de consulta
  - Native Queries
    - Consulta em linguagem nativa SQL

JPA - Operações de Consulta

- Objetos de consultas:
  - *Query* – sem verificação de tipo nos resultados
  - *TypedQuery* – com verificação de tipo nos resultados
    - Obtido pela passagem do tipo na obtenção da consulta via *createQuery()* ou *createNamedQuery()*
- Fornecem método para configuração de parâmetros:
  - Parâmetros JPQL seguem formato nomeado “:nome” ou posicional “?1”
  - Método *setParameter(parametro,valor)*
- Fornecem métodos para execução das consultas:
  - *getResultList()* – retorna uma lista como o resultado da consulta
  - *getSingleResult()* – retorna um único valor como resultado da consulta
  - *getResultStream()* – retorna um cursor para a base de dados
  - *executeUpdate()* – para comandos de alteração de dados, retorna número de linhas

JPA - JPQL

- Método **createQuery**():
  - Criar consultas dinâmicas (definidas “inline”) com base na linguagem JPQL
  - Ex.:

```
Query consulta = entityManager.createQuery("select
  umlivro from Livro umlivro where
  umlivro.editora.codigo = :cod");
consulta.setParameter(“cod”, codigo);
List<Livro> resultado = consulta.getResultList();
```

```
TypedQuery<Livro> consulta =
  entityManager.createQuery("select umlivro from Livro
  umlivro where umlivro.editora.codigo = :cod",
  Livro.class);
consulta.setParameter(“cod”, codigo);
List<Livro> resultado = consulta.getResultList();
```

JPA - JPQL

- Método **createNamedQuery**():
  - Criar consultas estáticas (definidas nos “metadados”) com base na linguagem JPQL
  - Consultas são definidas com a notação @NamedQuery ou @NameQueries na entidade
    - Nome deve ser único dentro da unidade de persistência
  - Ex.:

```
@Entity
@NamedQuery(name="todosLivros",query="select l from
  Livro l")
public class Livro implements Serializable{...}

Query consulta =
  entityManager.createNamedQuery("todosLivros");
List<Livro> resultado = consulta.getResultList();
```

JPA - JPQL

- Seleção:
  - Ex.: Todos os livros

```
TypedQuery<Livro> consulta =
 entityManager.createQuery("select umlivro
 from Livro umlivro", Livro.class);
List<Livro> resultado =
 consulta.getResultList();
```

JPA - JPQL

- Seleção com projeção:
  - Ex.: Todos os títulos de livros

```
TypedQuery<String> consulta =
 entityManager.createQuery("select
 umlivro.titulo from Livro umlivro",
 String.class);
List<String> resultado =
 consulta.getResultList();
```

JPA - JPQL

- Seleção com projeção:
  - Uma projeção pode resultar em dados que não correspondem a nenhum objeto existente
  - Resultado da consulta é um arranjo de Object contendo os valores projetados
  - Ex.: Todos os títulos de livros com o nome de suas editoras

```
Query consulta = entityManager.createQuery("select
 umlivro.titulo, umlivro.editora.nome from Livro
 umlivro");
List<Object[]> resultado = consulta.getResultList();
for(Object[] dados : resultado){
 String livro = dados[0].toString();
 String editora = dados[1].toString();
}
```

JPA - JPQL

- Seleção com projeção:
  - Utiliza-se o operador *new* para projetar o resultado em um objeto de uma classe existente
  - Ex.: Todos os títulos de livros com o nome de suas editoras

```
public class LivroEditora {
 public LivroEditora(String l, String e){...}
 ...
}

Query consulta = entityManager.createQuery("select new
  dados.LivroEditora(umlivro.titulo, umlivro.editora.nome)
  from Livro umlivro");
List<LivroEditora> resultado = consulta.getResultList();
for(LivroEditora dados : resultado){
 String livro = dados.getNomeLivro();
 String editora = dados.getNomeEditora();
}
```

JPA - JPQL

- Filtragem:
  - Operadores de comparação: <, >, <=, >=, =, <>, is null, is not null, between...and..., not between...and..., like, not like
  - Operadores lógicos: and, or, not
  - Operadores de conjunto: member of, not member of, is empty, is not empty, exists, not exists, in, not in, all, any, some
  - Ex.: Todos os livros de uma determinada editora

```
TypedQuery<Livro> consulta =
  entityManager.createQuery("select umlivro from
  Livro umlivro where umlivro.editora.codigo =
  :cod",Livro.class);
consulta.setParameter(“cod”, codigo);
List<Livro> resultado = consulta.getResultList();
```

JPA - JPQL

- Agregação:
  - Operadores de agregação: avg, sum, min, max, count
  - Ex.: A quantidade de livros

```
TypedQuery<Long> consulta =
 entityManager.createQuery("select count(umlivro)
 from Livro umlivro",Long.class);
Long resultado = consulta.getSingleResult();
```

JPA - JPQL

- Ordenação:
  - Operador order by, com opções de asc ou desc
  - Ex.: Todos os livros ordenados pelo nome

```
TypedQuery<Livro> consulta =
 entityManager.createQuery("select umlivro from
 Livro umlivro order by umLivro.nome
 asc",Livro.class);
List<Livro> resultado = consulta.getResultList();
```

JPA - JPQL

- Junção:
  - Operador join
  - Suporta versões com outer join
  - Suporta execução com busca imediata das entidades relacionadas via operador fetch no caso de uma configuração lazy no relacionamento
  - Ex.: Todos os livros e seu autores relacionados em uma única consulta

```
TypedQuery<Livro> consulta =
 entityManager.createQuery("select umlivro from
 Livro umlivro left join fetch
 umLivro.autores",Livro.class);
List<Livro> resultado = consulta.getResultList();
```

JPA - JPQL

- Paginação:
  - Métodos de configuração do objeto de consulta:
    - *setFirstResult()*
    - *setMaxResults()*
  - Ex.: Todos os livros ordenados pelo nome com paginação

```
TypedQuery<Livro> consulta =
 entityManager.createQuery("select umlivro from
 Livro umlivro order by umLivro.nome
 asc",Livro.class);
consulta.setFirstResult(10);
Consulta.setMaxResults(20);
List<Livro> resultado = consulta.getResultList();
```

JPA - JPQL

- Operações em lote:
  - Operadores update e delete
  - Alterações e remoções realizadas em um único comando
  - Ex.: Remover todos os livros de uma determinada editora

```
Query consulta = entityManager.createQuery("delete
  Livro umlivro where umlivro.editora.codigo =
  :cod",Livro.class);
consulta.setParameter(“cod”, codigo);
consulta.executeUpdate();
```

JPA - Criteria

- Conjunto de classes e interfaces para a definição de objetos de consulta
  - Uma árvore representando a estrutura a partir de uma fonte raiz “mais ampla” até elementos “mais específicos”
- Fortemente tipado
  - Suporte ao conceito de metamodelo
  - Modelo baseado em strings é fracamente tipado
- Portável
  - Consultas na API Criteria são independes do modelo da fonte de dados

JPA - Criteria

- Objetos básicos:
  - *CriteriaBuilder*
    - Objeto responsável pela montagem das estruturas das consultas
    - Obtido via EntityManagerFactory ou EntityManager através do método *getCriteriaBuilder()*
  - *CriteriaQuery*
    - Objeto de consulta
    - Obtido via CriteriaBuilder através do método *createQuery()*
      - O parâmetro do método deve indicar o tipo de classe associado ao resultado da consulta tipada que será construída
- Ex.:

```
CriteriaBuilder cb =
 entityManager.getCriteriaBuilder();
CriteriaQuery<Livro> cq =
 cb.createQuery(Livro.class);
```

JPA - Criteria

- Fonte da consulta:
  - *Root*
    - Objeto que define a raiz da consulta, ou seja, por onde começa a navegação sobre as entidades
    - Obtido via CriteriaQuery através do método *from()*
    - Pode existir múltiplos *from()*
- Ex.:

```
CriteriaBuilder cb =
 entityManager.getCriteriaBuilder();
CriteriaQuery<Livro> cq =
 cb.createQuery(Livro.class);
Root<Livro> umLivro = cq.from(Livro.class);
```

JPA - Criteria

- Seleção:
  - Método *select()*
  - Projeção do resultado da consulta

- Ex.: Todos os livros

```
CriteriaBuilder cb =
 entityManager.getCriteriaBuilder();
CriteriaQuery<Livro> cq =
 cb.createQuery(Livro.class);
Root<Livro> umLivro = cq.from(Livro.class);
cq.select(umLivro);
TypeQuery<Livro> consulta =
 entityManager.createQuery(cq);
List<Livro> resultado =
 consulta.getResultList();
```

JPA - Criteria

- Seleção:
  - Reescrevendo a consulta com uma outra estrutura mais próxima da linguagem SQL

- Ex.: Todos os livros

```
CriteriaBuilder cb =
  entityManager.getCriteriaBuilder();
CriteriaQuery<Livro> cq =
  cb.createQuery(Livro.class);
TypeQuery<Livro> consulta =
  entityManager.createQuery(
    cq.select(
      cq.from(Livro.class)
    )
);
List<Livro> resultado = consulta.getResultList();
```

JPA - Criteria

- Seleção:
  - Método *select()* não é necessário se a projeção é a própria entidade fonte da consulta

- Ex.: Todos os livros

```
CriteriaBuilder cb =
 entityManager.getCriteriaBuilder();
CriteriaQuery<Livro> cq =
 cb.createQuery(Livro.class);
TypeQuery<Livro> consulta =
 entityManager.createQuery(cq);
List<Livro> resultado =
 consulta.getResultList();
```

JPA - Criteria

- Seleção com projeção:
  - Método *select()* com *get()* sobre o dado desejado

- Ex.: Todos os títulos de livros

```
CriteriaBuilder cb =
 entityManager.getCriteriaBuilder();
CriteriaQuery<String> cq =
 cb.createQuery(String.class);
Root<Livro> umLivro = cq.from(Livro.class);
cq.select(umLivro.get("titulo"));
TypeQuery<String> consulta =
 entityManager.createQuery(cq);
List<String> resultado =
 consulta.getResultList();
```

JPA - Criteria

- Seleção com projeção:
  - Reescrevendo a consulta com uma outra estrutura mais próxima das linguagens SQL

- Ex.: Todos os títulos de livros

```
CriteriaBuilder cb =
  entityManager.getCriteriaBuilder();
CriteriaQuery<Livro> cq =
  cb.createQuery(Livro.class);
TypeQuery<String> consulta =
  entityManager.createQuery(
    cq.select(
      cq.from(Livro.class).get("titulo")
    )
);
List<String> resultado = consulta.getResultList();
```

JPA - Criteria

- Metamodelo:
  - Consultas fortemente tipadas necessitam de um metamodelo que descreve os dados (nome e tipos)
  - O metamodelo é representado em tempo de execução por objetos genéricos do tipo *EntityType<T>*
  - Um metamodelo está associado à classe da entidade
    - Estático – criado em tempo de desenvolvimento (usualmente através de uma ferramenta automatizada) via anotação *@StaticMetamodel*
      - Nome possui um "_" no final
      - Atributos correspondem aos campos ou propriedades persistentes da entidade relacionada
    - Dinâmico – obtido em tempo de execução via método *getModel()* do contexto raiz da consulta (objeto *Root*), ou via método *entity()* do objeto *MetaModel* obtido a partir do EntityManager via método *getMetaModel()*
      - Este tipo de uso é desaconselhado

JPA - Criteria

- Exemplo “estático”:

```
@Entity
public class Pet {
  @Id
  protected Long id;
  protected String name;
  protected String color;
  @ManyToOne
  protected Set<Owner> owners;
  ...
}

@StaticMetamodel(Pet.class)
public class Pet_ {
  public static volatile SingularAttribute<Pet, Long> id;
  public static volatile SingularAttribute<Pet, String> name;
  public static volatile SingularAttribute<Pet, String> color;
  public static volatile SetAttribute<Pet, Owner> owners;
}
```

JPA - Criteria

- Exemplo “dinâmico”:

```
CriteriaBuilder cb =
 entityManager.getCriteriaBuilder();
CriteriaQuery cq =
 cb.createQuery(Pet.class);
Root<Pet> pet = cq.from(Pet.class);
EntityType<Pet> Pet_ = pet.getModel();

Metamodel m = entityManager.getMetamodel();
EntityType<Pet> Pet_ = m.entity(Pet.class);
```

JPA - Criteria

- Seleção com projeção:
  - Método *select()* com *get()* sobre o dado desejado via metamodelo

- Ex.: Todos os títulos de livros

```
CriteriaBuilder cb =
 entityManager.getCriteriaBuilder();
CriteriaQuery<String> cq =
 cb.createQuery(String.class);
Root<Livro> umLivro = cq.from(Livro.class);
cq.select(umLivro.get(Livro_.titulo));
TypeQuery<String> consulta =
 entityManager.createQuery(cq);
List<String> resultado =
 consulta.getResultList();
```

JPA - Criteria

- Seleção com projeção:
  - Uma projeção pode resultar em dados que não correspondem a nenhum objeto existente
  - Opção 1 (array de object)
  - Método *array()* de CriteriaBuilder com *get()* sobre vários dados desejados
  - Resultado da consulta é um arranjo de Object contendo os valores projetados

JPA - Criteria

- Ex.: Todos os títulos de livros com o seu código

```
CriteriaBuilder cb =
  entityManager.getCriteriaBuilder();
CriteriaQuery<Object[]> cq =
  cb.createQuery(Object[].class);
Root<Livro> umLivro = cq.from(Livro.class);
cq.select(cb.array(umLivro.get(Livro_.titulo),
  umLivro.get(Livro_.codigo)));
TypeQuery<Object[]> consulta =
  entityManager.createQuery(cq);
List<Object[]> resultado = consulta.getResultList();
for(Object[] dados : resultado){
  String livro = (String)dados[0];
  Integer codigo = (Integer)dados[1];
}
```

JPA - Criteria

- Seleção com projeção:
  - Uma projeção pode resultar em dados que não correspondem a nenhum objeto existente
  - Opção 2 (array de object)
  - Método *multiselect()* de CriteriaQuery com *get()* sobre vários dados desejados
  - Resultado da consulta é um arranjo de Object contendo os valores projetados

JPA - Criteria

- Ex.: Todos os títulos de livros com o seu código

```
CriteriaBuilder cb =
  entityManager.getCriteriaBuilder();
CriteriaQuery<Object[]> cq =
  cb.createQuery(Object[].class);
Root<Livro> umLivro = cq.from(Livro.class);
cq.multiSelect(umLivro.get(Livro_.titulo),
  umLivro.get(Livro_.codigo));
TypeQuery<Object[]> consulta =
  entityManager.createQuery(cq);
List<Object[]> resultado = consulta.getResultList();
for(Object[] dados : resultado){
  String livro = (String)dados[0];
  Integer codigo = (Integer)dados[1];
}
```

JPA - Criteria

- Seleção com projeção:
  - Uma projeção pode resultar em dados que não correspondem a nenhum objeto existente
  - Opção 3 (classe wrapper que conterá o resultado)
  - Método *construct()* de CriteriaBuilder com *get()* sobre vários dados desejados
  - Resultado da consulta é uma coleção de objetos wrapper contendo os valores projetados

JPA - Criteria

- Ex.: Todos os títulos de livros com o seu código

```
public class LivroWrapper {
 public LivroWrapper(String t, Integer
 i){...}
 ...
}
```

JPA - Criteria

- Ex.: Todos os títulos de livros com o seu código

```
CriteriaBuilder cb =
  entityManager.getCriteriaBuilder();
CriteriaQuery<LivroWrapper> cq =
  cb.createQuery(LivroWrapper.class);
Root<Livro> umLivro = cq.from(Livro.class);
cq.select(cb.construct(LivroWrapper.class,
  umLivro.get(Livro_.titulo),
  umLivro.get(Livro_.codigo)));
TypeQuery<LivroWrapper> consulta =
  entityManager.createQuery(cq);
List<LivroWrapper> resultado =
  consulta.getResultList();
for(LivroWrapper dados : resultado){
  String livro = dados.getTitulo();
  Integer codigo = dados.getCodigo();
}
```

JPA - Criteria

- Seleção com projeção:
  - Uma projeção pode resultar em dados que não correspondem a nenhum objeto existente
  - Opção 4 (objeto Tuple)
  - Método *createTupleQuery()* de CriteriaBuilder para obter uma consulta que retornará objetos Tuple
  - Método *multiselect()* de CriteriaQeury com *get()* sobre vários dados desejados
  - Resultado da consulta é uma coleção de objetos Tuple contendo os valores projetados

JPA - Criteria

- Ex.: Todos os títulos de livros com o seu código

```
CriteriaBuilder cb =
  entityManager.getCriteriaBuilder();
CriteriaQuery<Tuple> cq = cb.createTupleQuery();
Root<Livro> umLivro = cq.from(Livro.class);
cq.multiselect(umLivro.get(Livro_.titulo),
  umLivro.get(Livro_.codigo));
TypeQuery<Tuple> consulta =
  entityManager.createQuery(cq);
List<Tuple> resultado = consulta.getResultList();
for(Tuple dados : resultado){
  String livro =
  tuple.get(umLivro.get(Livro_.titulo));
  Integer codigo =
  tuple.get(umLivro.get(Livro_.codigo));
}
```

JPA - Criteria

- Filtragem:
  - Método *where()* trabalha sobre um *Expression* ou uma lista de *Predicate*
  - Diferentes métodos sobre *CriteriaBuilder* retornam *Expression* ou *Predicate*
  - Métodos de comparação: equal(), notEqual(), gt(), ge(), lt(), le(), between(), isNull(), isNotNull(), like(), notLike(), in()
  - Métodos lógicos: and(), or(), not()
  - Operadores de conjunto: isMember(), isNotMember(), isEmpty(), isNotEmpty()
  - Parâmetros de uma consulta são objetos *ParameterExpression* especificados via método *parameter()* de *CriteriaBuilder*

JPA - Criteria

- Ex.: Livro de um determinado código

```
CriteriaBuilder cb =
 entityManager.getCriteriaBuilder();
CriteriaQuery<Livro> cq =
 cb.createQuery(Livro.class);
Root<Livro> umLivro = cq.from(Livro.class);
cq.select(umLivro);
ParameterExpression<Integer> codParam =
 cb.parameter(Integer.class);
cq.where(cb.equal(umLivro.get(Livro_.codigo),
 codParam));
TypeQuery<Livro> consulta =
 entityManager.createQuery(cq);
consulta.setParameter(codParam,1);
Livro resultado = consulta.getSingleResult();
```

JPA - Criteria

- Reescrevendo a consulta com uma sintaxe mais próxima das linguagens SQL
- Ex.: Livro de um determinado código

```
CriteriaBuilder cb =
  entityManager.getCriteriaBuilder();
CriteriaQuery<Livro> cq = cb.createQuery(Livro.class);
Root<Livro> umLivro = cq.from(Livro.class);
TypeQuery<String> consulta = entityManager.createQuery(
   cq.select(umLivro)
   .where(
      cb.equals(umLivro.get(Livro_.codigo),
  cb.parameter(Integer.class,"cod"))
    )
);
consulta.setParameter("cod",1);
Livro resultado = consulta.getSingleResult();
```

JPA - Criteria

- Ordenação:
  - Método *orderBy()* de *CriteriaQuery*
  - Opções de ordenação via métodos *asc()* e *desc()* de *CriteriaBuilder*

- Ex.: Todos os livros ordenados pelo nome

```
CriteriaBuilder cb =
  entityManager.getCriteriaBuilder();
CriteriaQuery<Livro> cq =
  cb.createQuery(Livro.class);
Root<Livro> umLivro = cq.from(Livro.class);
cq.select(umLivro);
cq.orderBy(cb.asc(umLivro.get(Livro_.titulo)));
TypeQuery<Livro> consulta =
  entityManager.createQuery(cq);
List<Livro> resultado = consulta.getResultList();
```

JPA - Criteria

- Agregação:
  - Métodos de agregação: avg(), sum(), min(), max(), count()

- Ex.: A quantidade de livros

```
CriteriaBuilder cb =
 entityManager.getCriteriaBuilder();
CriteriaQuery<Long> cq =
 cb.createQuery(Long.class);
Root<Livro> umLivro = cq.from(Livro.class);
cq.select(cb.count(umLivro));
TypeQuery<Long> consulta =
 entityManager.createQuery(cq);
Long resultado = consulta.getSingleResult();
```

JPA - Criteria

- Produto cartesiano:
  - Múltiplos *from()*

- Ex.: todas editoras com seus livros

```
CriteriaBuilder cb =
  entityManager.getCriteriaBuilder();
CriteriaQuery<Editora> cq =
  cb.createQuery(Editora.class);
Root<Livro> umLivro = cq.from(Livro.class);
Root<Editora> umaEditora = cq.from(Editora.class);
cq.select(umaEditora);
cq.where(cb.equals(umaEditora,umLivro.get(Livro_.edi
  tora)));
TypedQuery<Editora> consulta =
  entityManager.createQuery(cq);
List<Editora> resultado = consulta.getResultList();
```

JPA - Criteria

- Junção:
  - Método *join()* de objetos resultantes de *from()* indica a navegação entre entidades associadas via um campo ou propriedade

- Ex.: todos livros de uma determinada editora

```
CriteriaBuilder cb = entityManager.getCriteriaBuilder();
CriteriaQuery<Livro> cq = cb.createQuery(Livro.class);
Root<Livro> umLivro = cq.from(Livro.class);
Join<Livro,Editora> umaEditora =
  umLivro.join(Livro_.editora);
cq.select(umLivro);
cq.where(cb.equals(umaEditora.get(Editora_.nome),
  cb.parameter(String.class,”nome”)));
TypeQuery<Livro> consulta = entityManager.createQuery(cq);
consulta.setParameter(“nome”,”xyz”);
List<Livro> resultado = consulta.getResultList();
```

JPA - Criteria

- Junção:
  - Suporta execução com busca imediata das entidades relacionadas via método *fetch()* de objetos resultantes de *from()*

- Ex.: Todos os livros e seu autores relacionados em uma única consulta

```
CriteriaBuilder cb =
  entityManager.getCriteriaBuilder();
CriteriaQuery<Livro> cq =
  cb.createQuery(Livro.class);
Root<Livro> umLivro = cq.from(Livro.class);
umLivro.fetch("autores", JoinType.LEFT);
cq.select(umLivro);
TypeQuery<Livro> consulta =
  entityManager.createQuery(cq);
List<Livro> resultado = consulta.getResultList();
```

JPA - Nativo

- Provedores JPA suportam a utilização de consultas na linguagem SQL nativa do banco de dados sendo acessado
- Método ***createNativeQuery()***
- Ex.:

```
NativeQuery<Livro> consulta =
  entityManager.createNativeQuery("select * from
  Livro",Livro.class);
List<Livro> resultado = consulta.getResultList();
```