

Universidade Presbiteriana Mackenzie



Linguagem de Programação III



Aula 9: JPA e Hibernate

Prof. Ms. Leandro Pupo Natale

Tópicos da Aula

- Motivação
- Introdução: JPA, Hibernate, persistência
- Mapeamento Objeto Relacional – ORM
- Arquitetura
- Estados das instâncias
- Mecanismos de consultas
- Mapeamento de Entidades
- Ciclo de Vida das Entidades
- Carregando um objeto com dados do Banco de Dados

Motivação

- Para nos comunicarmos com o banco de dados, seja para uma consulta ou para algum tipo de alteração de dado, podemos usar a API que o Java SE disponibiliza, o **JDBC**.
- Problemas do JDBC, quando usado diretamente:
 - é que ele é muito trabalhoso além de que precisamos gerenciar muitos recursos importantes.
 - Todas as informações têm de ser passadas uma a uma.

Motivação

- Para evitar ter que fazer todas as chamadas ao JDBC e ganhar **tempo e produtividade**, utiliza-se um framework para a persistência que traz muitas **funcionalidades já implementadas**.
- O framework que utilizaremos será o Hibernate e ele será o responsável por fazer as chamadas à API do JDBC.

Persistência

- De modo geral, **persistência** significa continuar a existir, preservar, durar longo tempo ou permanecer.
- No contexto de **programação orientada a objeto**, a persistência significa a possibilidade desses objetos serem armazenados em **meio externo à aplicação**, portanto, deverá permitir que esses objetos não sejam voláteis.
- Atualmente, os bancos de dados relacionais são o meio mais utilizado para isso, porém não são os únicos.

Persistência

- Mecanismos de persistência:
 - Serialização
 - Facilidade de leitura e gravação
 - Dificuldade de consultas à base
 - Alternativas
 - Bancos de dados relacionais (JDBC)
 - Bancos de dados Orientados a objeto
 - Mapeamento Objeto relacional

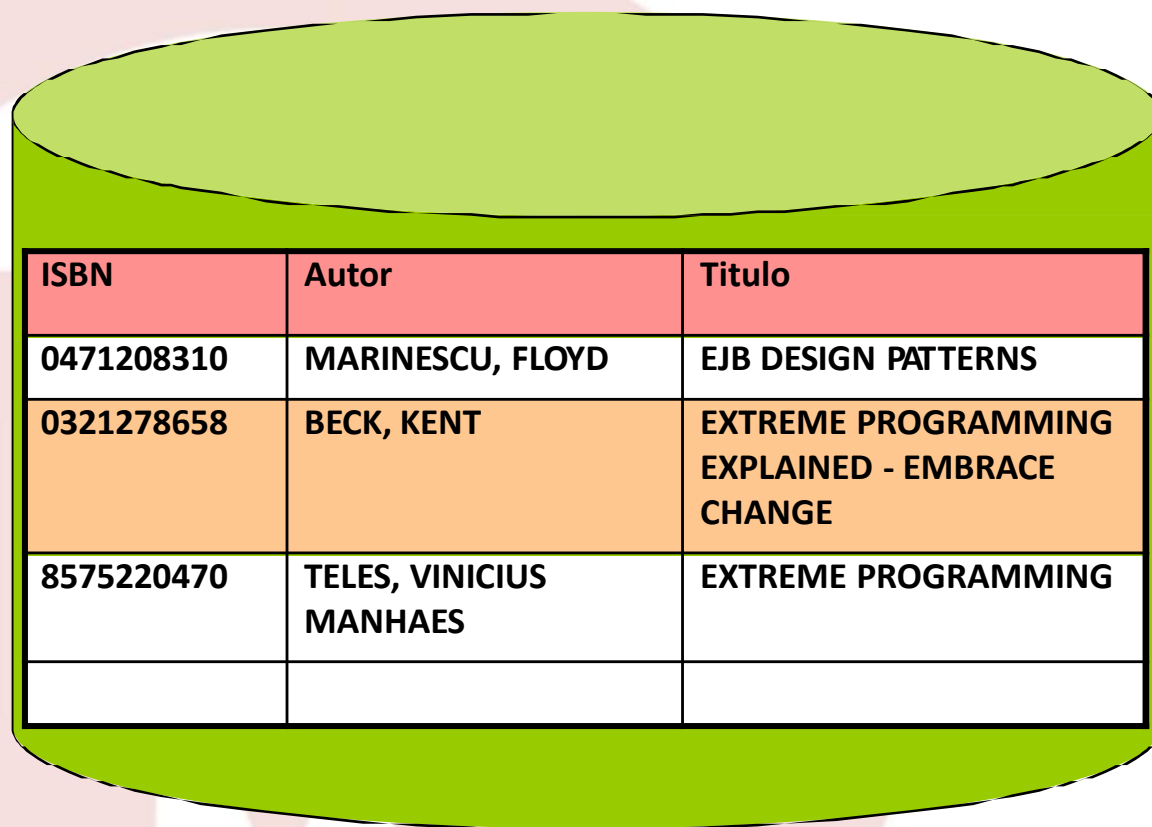


JPA – Java Persistence API

- Java Persistence API
 - É um padrão definido pelo JCP para trabalhar com persistência de dados.
 - O líder a especificação foi Gavin King, o criador do Hibernate.
 - Existem várias implementações disponíveis no mercado.
 - Hibernate, GlassFish, SAP Netweaver AS, Oracle TopLink, EclipseLink, OpenJPA, Kodo, JPOX, Amber, entre outras...

Mapeamento Objeto Relacional

- **Classes** são mapeadas a tabelas (**esquemas**)
- Instâncias (**objetos**) são mapeadas a registros (**linhas**)



ISBN	Autor	Título
0471208310	MARINESCU, FLOYD	EJB DESIGN PATTERNS
0321278658	BECK, KENT	EXTREME PROGRAMMING EXPLAINED - EMBRACE CHANGE
8575220470	TELES, VINICIUS MANHAES	EXTREME PROGRAMMING

Classe Livro

String ISBN
String autor
String titulo

instância: Livro

ISBN = 0321278658
Autor = "BECK, KENT"
Titulo = "EXTREME PROGRAMMING..."

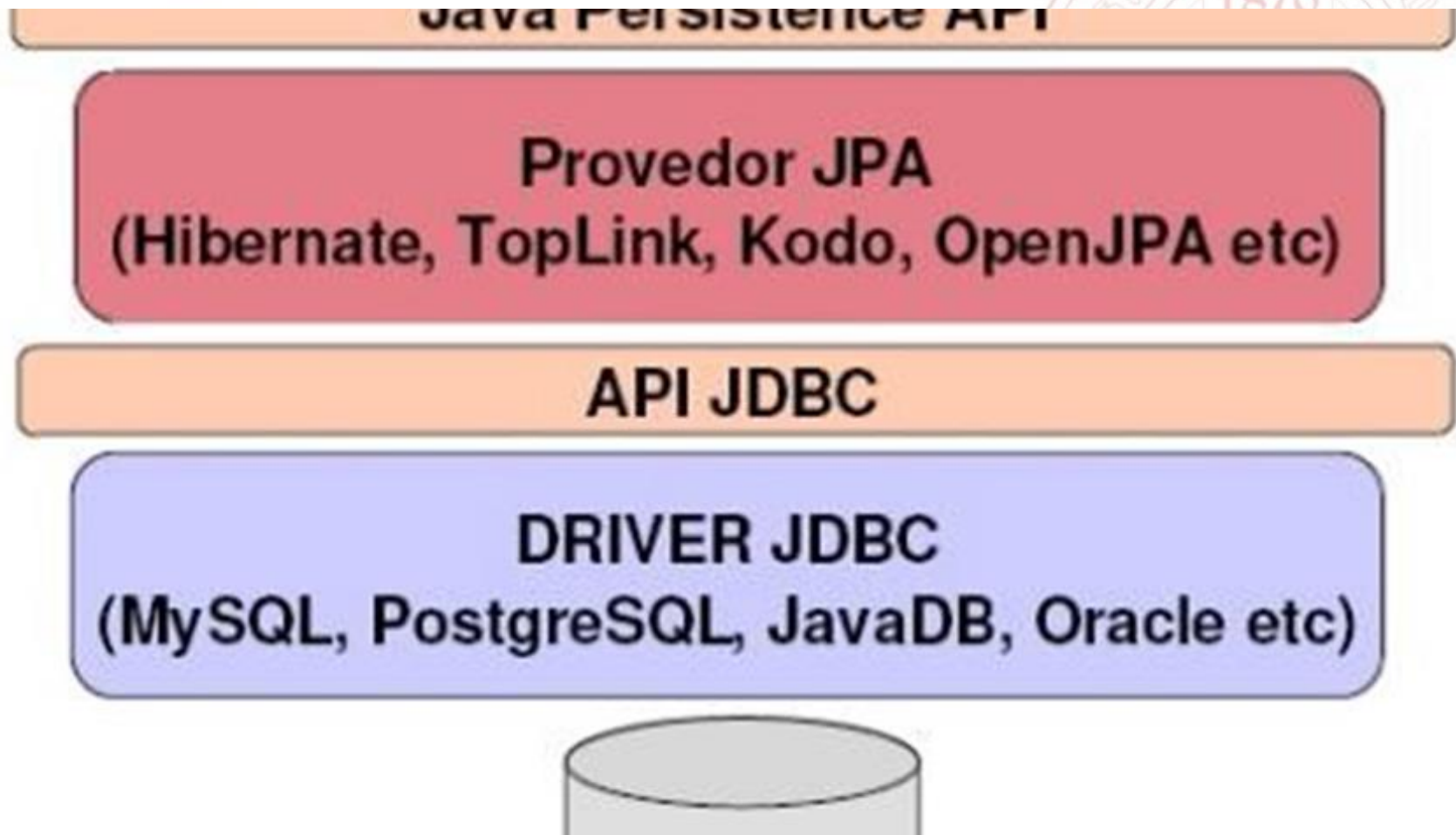
Mapeamento Objeto Relacional

- Porém quando falamos em ORM, estamos nos referindo normalmente ao processo automatizado
- Uma Solução de ORM completa normalmente oferece:
 - Uma API para realização de operações básicas de criação, leitura, atualização e remoção (CRUD) em objetos de classes persistentes
 - Uma linguagem ou API para especificar consultas sobre classes e suas propriedades
 - Um recurso para especificar meta dados de mapeamento
 - Uma técnica que permita à implementação interagir com objetos transacionais (para realizar funções de otimização)

Por que ORM? Por que Hibernate?

- Produtividade
 - Permite concentrar-se na lógica de negócio da aplicação
- Manutenabilidade
 - Menos linhas de código
 - Separação de camada de persistência da de negócios
- Performance
 - Várias otimizações
- Independência de fabricante
 - Isolamento do fabricante de banco de dados

Arquitetura



Configuração

- Para configurar o Hibernate em uma aplicação, devemos criar um arquivo chamado **persistence.xml**.
- O conteúdo desse arquivo contém informações sobre o banco de dados, como a url de conexão, usuário e senha, além de dados sobre a implementação JPA que será utilizada.
- O arquivo persistence.xml deve ser salvo em uma pasta chamada **META-INF**, que deve estar no classpath da aplicação para a IDE Eclipse, por exemplo.

Configuração

- Exemplo de configuração para o persistence.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0"
  xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">

  <persistence-unit name="livraria-pu" transaction-type="RESOURCE_LOCAL">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <properties>
      <property name="hibernate.dialect" value="org.hibernate.dialect.MySQL5InnoDBDialect"/>

      <property name="hibernate.hbm2ddl.auto" value="update"/>

      <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>

      <property name="javax.persistence.jdbc.user" value="root"/>

      <property name="javax.persistence.jdbc.password" value="root"/>

      <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/livraria"/>

    </properties>
  </persistence-unit>
</persistence>
```


Configuração

- A propriedade **hibernate.dialect** permite que a aplicação escolha qual sintaxe de SQL deve ser utilizada pelo Hibernate.
- A propriedade **hibernate.hbm2ddl.auto** permite definir a política de criação de tabelas, por meio do **valor** da propriedade:
 - **create-drop**: para sobrescrever as tabelas existentes
 - **update**: atualizar as tabelas de acordo com as mudanças nas anotações sem removê-las

Mecanismos de Consultas OO

- Os **provedores de JPA** oferecem mecanismos para realizar consultas de uma maneira orientada a objetos.
- A especificação JPA 2 define dois mecanismos para realizar consultas OO:
 - **JPQL (Java Persistence Query Language)**: linguagem específica para consultas
 - **Criteria**: biblioteca Java para consultas.

Mecanismos de Consultas OO

- Vantagens da utilização dos mecanismos de consulta do JPA 2:
 - são independentes dos mecanismos específicos de consulta do banco de dados.
 - Pode-se definir uma consulta em JPQL ou Criteria e executá-la em qualquer banco de dados suportado pelo provedor JPA.

Estados de uma instância

- Estados de uma instância de objeto:
 - **Transiente:**
 - A instância não está, e nunca esteve associada a um contexto de persistência.
 - O objeto **não possui identificador** (chave primária).
 - **Persistente:**
 - A instância está associada a um contexto de persistência.
 - Possui identificador.

Ciclo de Vida das Entidades

- **New (Novo):**
 - Um objeto nesse estado não possui uma identidade (chave) e não está associado a um EntityManager. O conteúdo desse objeto não é enviado para o banco de dados. Toda instância de uma entidade que acabou de ser criada como comando new encontra-se no estado new do JPA.
- **Managed (Gerenciado):**
 - Um objeto no estado managed possui uma identidade e está associado a um Entity Manager. A cada sincronização, os dados de um objeto no estado managed são atualizados no banco de dados

Ciclo de Vida das Entidades

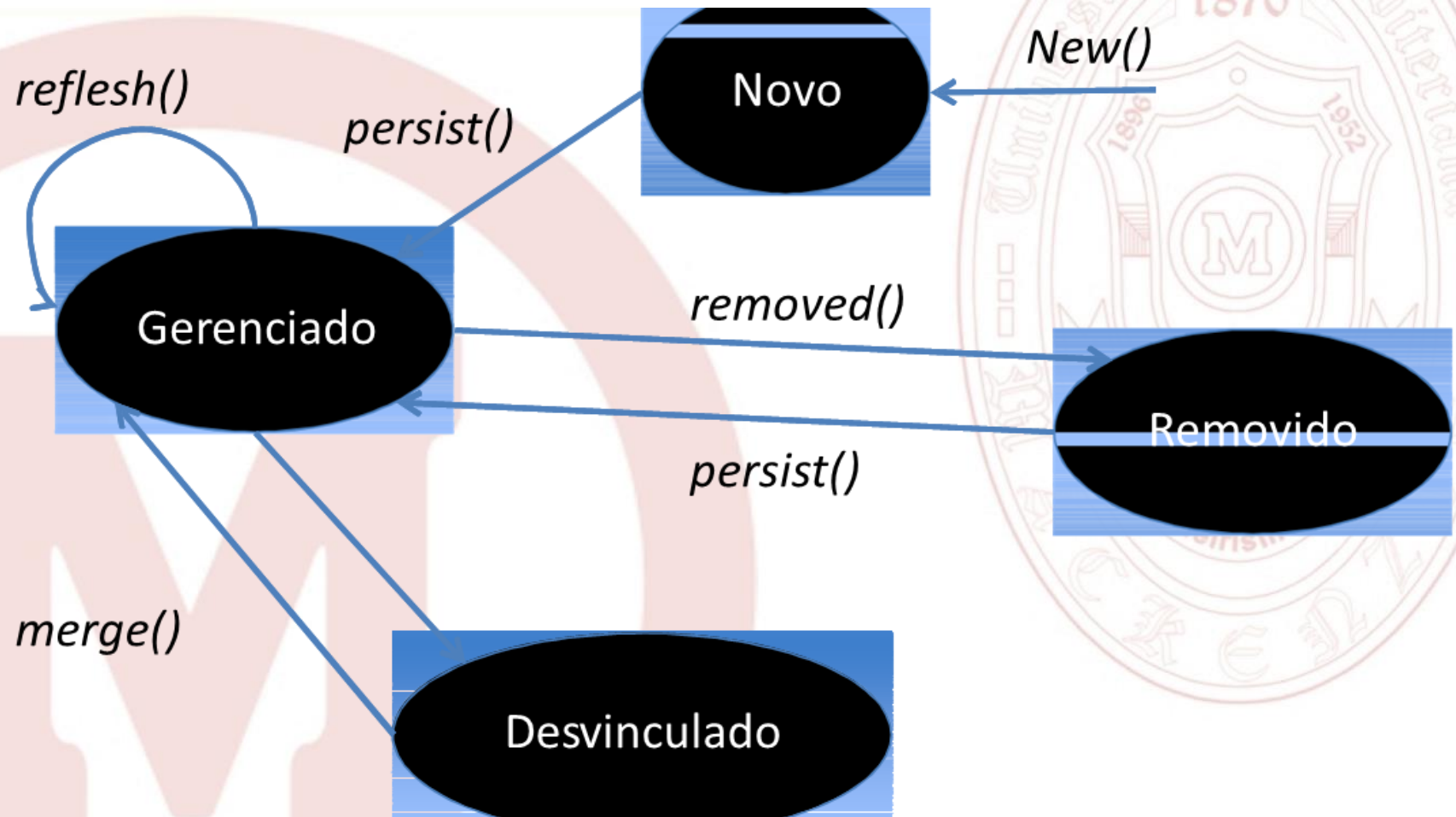
- **Detached (Desvinculado):**

- Um objeto no estado detached possui uma identidade, mas **não está associado** a um Entity Manager. Dessa forma, o conteúdo desse objeto **não é sincronizado** com o banco de dados.

- **Removed (Removido):**

- Um objeto no estado removed possui uma identidade e está associado a um Entity Manager. O conteúdo desse objeto será removido do banco de dados quando houver uma sincronização.

Ciclo de Vida das Entidades



Ciclo de Vida das Entidades

- Exemplos de transições:

- Novo para Gerenciado**

```
@Entity
public class Editora {
    @Id
    @GeneratedValue (strategy=GenerationType.AUTO)
    private Long id;
    private String nome;

    //Getters e Setters
}
```

```
EntityManager em = emf.createEntityManager();
em.getTransaction().begin();
Editora e = new Editora();
e.setNome("EDMACK");
em.persist();
em.getTransaction().commit();
```

Ciclo de Vida das Entidades

- Exemplos de transições:
- **Gerenciado para Desvinculado:**
 - Quando não queremos mais que um objeto no estado managed seja gerenciado, podemos desvinculá-lo do seu Entity Manager tornando-o detached.
 - Dessa forma, o conteúdo desse objeto não será mais sincronizado com o banco de dados.

Ciclo de Vida das Entidades

- Exemplos de transições:
- **Gerenciado para Desvinculado:**
 - — Para tornar apenas um objeto detached, deve-se utilizar o método **detach()**:

```
Editora e = em.find(Editora.class, 1L);  
em.detach();
```

- Para tornar detached todos os objetos gerenciados por um EntityManager, deve-se usar o método **clear()**:

```
em.clear();
```


Ciclo de Vida das Entidades

- Exemplos de transições:
- **Gerenciado para Desvinculado:**
 - – Na chamada do método **close()**, todos os objetos gerenciados ou administrados por um EntityManager também passam para o estado detached.

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

Ciclo de Vida das Entidades

- Exemplos de transições:
- **Desvinculado para Gerenciado:**
 - – O estado de um objeto detached pode ser propagado para um objeto managed com a mesma identidade para que os dados sejam sincronizados com o banco de dados. Esse processo é realizado pelo método **merge()**.

```
Editoria editoraManaged = em.merge(editoraDetached);
```

Ciclo de Vida das Entidades

- Exemplos de transições:
- **Gerenciado para Removido:**
 - Quando um objeto gerenciado se torna removido, os dados correspondentes a esse objeto não são apagados do banco de dados. Porém, quando utiliza-se o método **remove()**, marca-se um objeto para ser removido do BD.
 - O conteúdo do objeto será removido do BD quando o provedor realizar uma sincronização.

```
Editora e = em.find(Editora.class, 1L);  
em.remove();
```

Ciclo de Vida das Entidades

- Exemplos de transições:
- **Gerenciado para Gerenciado:**
 - O conteúdo de um objeto no estado managed pode ficar desatualizado em relação ao BD se alguém ou alguma aplicação alterar os dados na base de dados.
 - Para atualizar um objeto managed com os dados do BD, devemos utilizar o método **refresh()**.

```
Editora e = em.find(Editora.class, 1L);  
em.refresh(e);
```

Criando um Banco

- O Hibernate é capaz de **gerar as tabelas** do banco para a nossa aplicação.
- Faz-se isso com as anotações colocadas nas classes e as informações presentes no persistence.xml.
- As tabelas são geradas através do método da classe Persistence, o **createEntityManagerFactory(String persistenceUnit)** da classe Persistence.
 - – persistenceUnit permite escolher, pelo nome, uma unidade de persistência definida no persistence.xml.

Criando um Banco

- Definindo uma unidade de persistência
- Exemplo de geração de tabela:

```
public EntityManagerFactory emf =  
    Persistence.createEntityManagerFactory("Atividade10PU");
```

Entity Beans

- As regras de negócio de uma aplicação EJB são implementadas nos Session Beans.
- Já os dados da aplicação que devem ser persistidos são armazenados em objetos chamados **Entity Beans**.

Entity Beans

- Session Beans modelam tarefas realizadas pelo sistema
- Entity Beans modelam as entidades
 - Representam dados
 - Stateful Session Beans não representam dados!
 - Independente da forma de armazenamento
- Dois termos distintos
 - Entity Bean Instance: os dados na memória (instância da classe do Entity Bean)
 - Entity Bean Data: os dados fisicamente armazenados no banco

Entity Beans

- Entity Beans
 - têm ciclo de vida independente da aplicação cliente
 - têm identidade e estado visível para o cliente
 - são concorrentes
- Exemplos de Entity Beans que podem formar uma aplicação:
 - clientes
 - produtos
 - pedidos
 - funcionários
 - fornecedores

Mapeamento de Entidades

- Seguindo os padrões da JPA cada entidade DEVE atender os requisitos:
 - Ser anotada coma anotação **@Entity**.
 - Deve ter um construtor sem argumentos, público ou protegido.
 - Não pode ser declarada **final**. Nenhum método ou variável de instância pode ser declarada como **final**.
 - Variáveis de instância persistentes não devem ser declaradas públicas e só podem ser acessadas pelos métodos da classe.

Mapeamento de Entidades

- Um dos principais objetivos dos frameworks ORM é estabelecer o mapeamento entre os conceitos do modelo orientado a objetos e os conceitos do modelo relacional.
- Este mapeamento pode ser definido através de **xml**
- ou de maneira mais prática com **anotações Java**.
- As anotações Java de mapeamento do JPA estão no pacote **javax.persistence**.

Anotação @Entity

- É a principal anotação do JPA. Ela deve aparecer antes do nome de uma classe e deve ser definida em todas as classes que terão objetos persistidos no banco de dados.
- As **classes** anotadas com **@Entity** são mapeadas para
- **tabelas**.
 - Por convenção, as tabelas possuem os mesmos nomes das classes.
 - Pode-se alterar esse comportamento utilizando a anotação
- **@Table**.

Anotação @Entity

- Os **atributos** declarados em uma classe anotada ou
- **POJOs** (Plain-Old-Java-Objects) ou Java Beans, com
- @Entity são mapeados para **colunas na tabela**
- correspondente à classe.
 - Por convenção, as colunas também possuem os mesmos nomes dos atributos.
 - Pode-se alterar esse padrão utilizando para isso a anotação
- **@Column.**

Anotação @Id

- É utilizada para indicar **qual atributo** de uma classe anotada com @Entity será mapeado para a **chave primária da tabela** correspondente à classe.
- Geralmente o atributo anotado com @Id é do tipo Long.

Anotação @GeneratedValue

- Geralmente vem acompanhado da anotação @Id.
- Serve para indicar que o **valor de um atributo** que compõe uma **chave primária deve ser gerado** pelo banco no momento em que um novo registro é inserido.

```
@Entity
public class Editora {
    @Id
    @GeneratedValue (strategy=GenerationType.AUTO)
    private Long id;
    private String nome;

    //Getters e Setters
}
```


Anotação @Temporal

- Utilizada para atributos do tipo **Calendar** ou **Date**.
- Por padrão, tanto data quanto hora são armazenados no BD.
- Com a anotação **@Temporal**, pode-se mandar persistir somente a data ou somente a hora para definir precisão:
DATE, TIME e TIMESTAMP

```
@Entity
public class Livro {
    @Temporal(TemporalType.DATE)
    private Calendar publicacao;

    //...
}
```

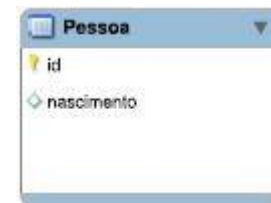
Anotação @Transient

- A instância de um dado transiente não está associada a um contexto de persistência. O objeto não possui identificador (chave primária).
- Assim, caso não se deseje que alguns atributos de um determinado grupo de objetos sejam persistidos no banco de dados, aplica-se o modificador **transient** ou a anotação **@Transient**.

Anotação @Transient

- Exemplo: o atributo idade com a anotação
- **@Transient** para que essa informação não seja armazenada no BD. A idade de uma pessoa pode ser deduzida a partir de sua data de nascimento, que já está armazenada no banco.

```
@Entity
public class Pessoa {
    @Temporal(TemporalType.DATE)
    private Calendar nascimento;
    @Transient
    private int idade;
    //...
}
```



Manipulando Entidades

- **Persistindo um objeto:** para armazenar as informações de um objeto no banco de dados , utiliza-se o método `persist()` do `EntityManager`.

```
EntityManager em = emf.createEntityManager();  
Aluno a = new Aluno();  
a.setNome("Vitor");  
em.persist(a);
```

Manipulando Entidades

- **Buscando um objeto:** para obter um objeto que contenha as informações do banco de dados, utiliza-se os métodos `find()` ou `getReference()` do `EntityManager`.

```
EntityManager em = emf.createEntityManager();  
Aluno a1 = em.find(Aluno.class, 1L);  
Aluno a2 = em.getReference(Aluno.class, 2L);
```

- Há uma diferença entre esses métodos de busca.
 - O método `find()` recupera os dados desejados imediatamente.
 - Já o método `getReference()` posterga essa tarefa até a primeira chamada de um método `get` no objeto desejado.

Manipulando Entidades

- **Removendo um objeto:** para remover um registro correspondente a um objeto, utiliza-se o método `remove()` do `EntityManager`.

```
EntityManager em = emf.createEntityManager();  
Aluno a = em.find(Aluno.class, 1L);  
em.remove(a);
```

- **Atualizando um objeto:** para alterar os dados de um registro correspondente a um objeto, utiliza-se os próprios métodos `setters` desse objeto.

```
EntityManager em = emf.createEntityManager();  
Aluno a = em.find(Aluno.class, 1L);  
a.setNome("Vitor");
```

Manipulando Entidades

- **Listando um objeto:** para obter uma listagem com todos os objetos referentes os registros de uma tabela, pode-se utilizar a linguagem de consulta do JPA, a **JPQL**.
- Vantagens da JPQL:
 - Muito parecida com o SQL
 - Mesma sintaxe para BD diferentes

```
EntityManager em = emf.createEntityManager();  
Query query = em.createQuery("SELECT e FROM Aluno e");  
List<Aluno> alunos = query.getResultList();
```

- O resultado dessa consulta é uma lista com todos as instâncias da entidade Aluno que foram persistidas.

Sincronização

- **Transações:** as modificações realizadas nos objetos administrados por um EntityManager são mantidas em memória.
- Em certos momentos, é necessário sincronizar os dados da memória com os dados do BD.
- Essa sincronização deve ser realizada através de uma transação JPA criada pelo EntityManager que administra os objetos que deseja-se sincronizar.

Sincronização

- Uma sincronização consiste em propagar para o banco de dados as modificações, remoções e inserções de entidades realizadas em memória através de um `EntityManager`.
- Quando houver uma sincronização, as modificações realizadas no estado dos objetos managed são propagadas para o banco de dados, assim como os registros referentes aos objetos em estado removed são apagados do banco de dados.

Sincronização

- De acordo com a especificação, uma sincronização só pode ocorrer se uma transação estiver ativa.
- Cada Entity Manager possui uma **única transação** associada. Para recuperar a transação associada a um Entity Manager, utilizamos o método **getTransaction()**.
- **Transações:** Para abrir uma transação utilizamos o método **begin()**.

```
EntityManager em = emf.createEntityManager();  
em.getTransaction().begin();
```


Sincronização

- Com a transação aberta, pode-se sincronizar os dados com o banco através dos métodos:
 - **flush()** (parcialmente) ou
 - **commit()** (definitivamente).

```
EntityManager em = emf.createEntityManager();  
em.getTransaction().begin();  
em.getTransaction().commit();
```

Sincronização

- **commit()** :
 - – Para confirmar uma transação, devemos usar o método commit(). Quando esse método é invocado, ocorre uma sincronização com o banco de dados e a transação é finalizada.
- **flush()**
 - – Com uma transação ativa, também podemos disparar uma sincronização através do método flush().

Sincronização

- **flush()**
 - Apesar dos dados serem enviados para o banco de dados, eles **não ficarão visíveis para outras transações**.
 - Esses dados serão considerados apenas nas consultas efetuadas dentro da própria transação.
 - Diversas chamadas ao método flush() podem ser efetuadas dentro de uma mesma transação.
 - Toda modificação, remoção ou inserção realizada no banco de dados devido às chamadas ao método flush() podem ser desfeitas através do método **rollback()**.
 - Uma chamada a esse método também finaliza a transação.

Carregando um Objeto com os Dados do Banco de Dados

- Como os Entity Managers administram as instâncias das entidades, eles são responsáveis pelo carregamento do estado dos objetos.
- Há dois **modos de carregar um objeto** com os dados obtidos de um BD:
 - **LAZY**: o provedor posterga ao máximo a busca dos dados no banco de dados.
 - **EAGER**: o provedor busca imediatamente os dados no banco de dados.

Carregando um Objeto com os Dados do Banco de Dados

- Métodos **find()** e **getReference()**
 - Os 2 métodos permitem que a aplicação obtenha instâncias das entidades a partir das identidades dos objetos.
 - Diferenças:
 - **find()** tem comportamento **EAGER**
 - **getReference()** tem comportamento **LAZY**

Hibernate

- Hibernate é uma ferramenta de ORM, **Object Relational Mapping** (Mapeamento objeto-relacional) para Java e .NET (NHibernate)
- Oferece serviços para consulta e recuperação de dados
- Objetivo: liberar o desenvolvedor de grande parte da programação relacionada a persistência de dados
- Suporte aos principais SGDBs

Hibernate

- Ajuda a persistir objetos Java em um banco de dados relacional.
- O trabalho do desenvolvedor é definir como os objetos são mapeados nas tabelas do banco e o Hibernate faz todo o acesso ao banco, gerando inclusive os comandos SQL necessários.

Hibernate

- Projeto Hibernate
 - Mantido pelo grupo JBoss
 - Projeto Opensource
 - Adotado por muitos projetos no mundo inteiro
 - Versão 3 foi base para a especificação de persistência da JPA
 - Hibernate implementa a JPA.





Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0)

Você tem a liberdade de:



Compartilhar — copiar, distribuir e transmitir a obra.



Remixar — criar obras derivadas.



Sob as seguintes condições:



Atribuição — Você deve creditar a obra da forma especificada pelo autor ou licenciante (mas não de maneira que sugira que estes concedem qualquer aval a você ou ao seu uso da obra).



Compartilhamento pela mesma licença — Se você alterar, transformar ou criar em cima desta obra, você poderá distribuir a obra resultante apenas sob a mesma licença, ou sob uma licença similar à presente.

Este é um resumo amigável da [Licença Jurídica](#) (a licença integral)

Advertência



Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0)

You are free:



to Share — to copy, distribute and transmit the work



to Remix — to adapt the work



Under the following conditions:



Attribution — You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).



Share Alike — If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

This is a human-readable summary of the [Legal Code \(the full license\)](#).

[Disclaimer](#)

Imagens

- <http://www.digitalprank.org/wp-content/uploads/2008/03/ejb.png>
- <http://sandersconsulting.com/Portals/58319/images/checklist.jpg>
- <http://vip.cs.utsa.edu/classes/cs4393f2006/lectures/images/overview-j2eeArchitecture.gif>
- <http://www.dave-woods.co.uk/wp-content/uploads/2010/02/screens.gif>
- http://www.ishopping.pk/product_images/y/423/iMac-Lion_45051_std.png
- <http://www.wegotserverd.com/wp-content/uploads/2011/08/MacMiniServer.jpg>

Obrigado

Prof. Ms. Leandro Pupo Natale

leandro.natale@mackenzie.br