

Mapeamento Objeto-Relacional: Java Persistence API (JPA)

Fernando dos Santos
fernando.santos@udesc.br



Roteiro

- Introdução
- Mapeamento de uma entidade (classe)
- Configuração e uso da persistência



Mapeamento Objeto-Relacional

- É o mapeamento de classes e seus relacionamentos, para tabelas em um banco de dados relacional.
 - criação de objeto → insert no banco de dados
 - alteração de objeto → update no banco de dados
 - remoção de objeto → delete no banco de dados
 - associação de objetos → chaves estrangeiras no banco de dados
- No princípio, os objetos eram mapeados “a força”:
 - O desenvolvedor codificava os comandos SQL para cada classe e seus relacionamentos.
- Atualmente, o mapeamento é automático
 - O Java se encarrega de gerar os comandos SQL no banco;
 - Aumento de produtividade.



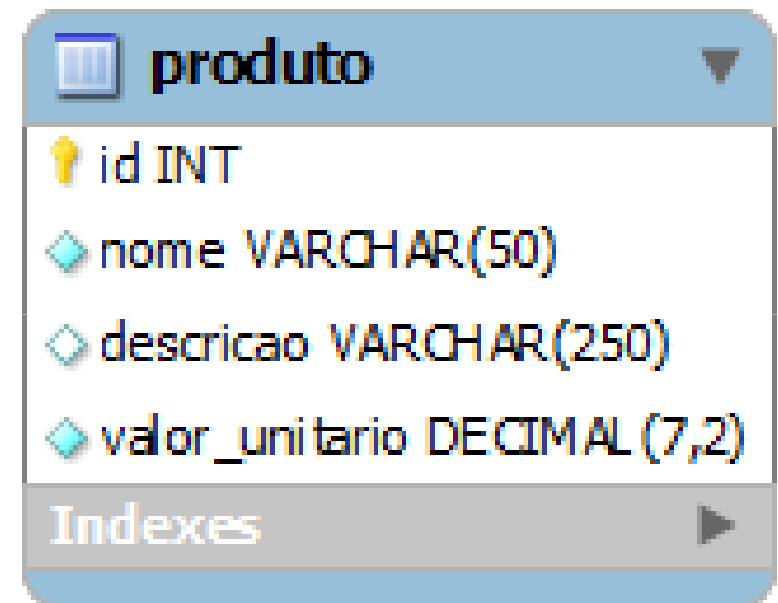
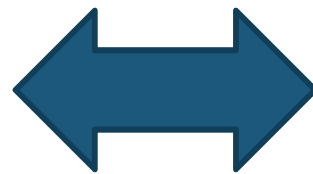
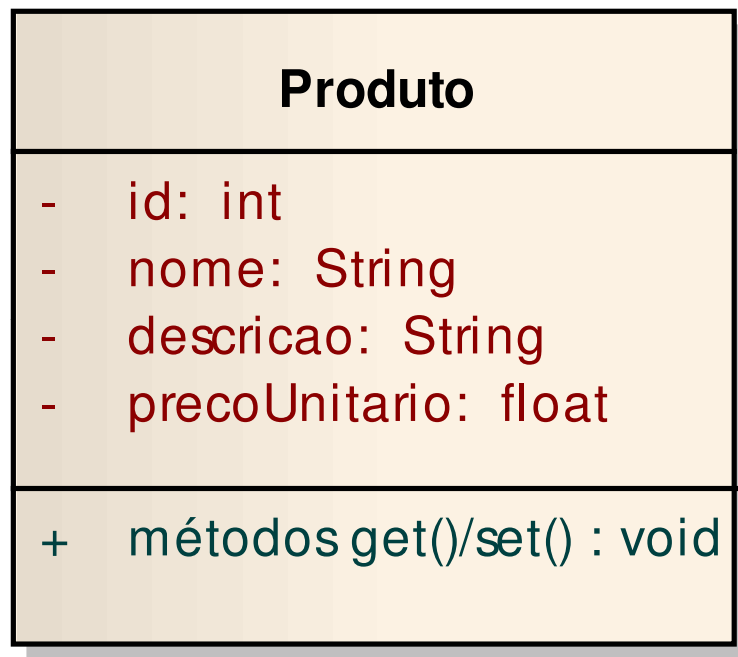
Java Persistence API (JPA)

- Biblioteca Java para realizar mapeamento objeto-relacional:
 - Metadados (anotações) para mapeamento;
 - Gerenciamento da persistência;
 - Linguagem de consulta.
- A JPA define apenas o padrão. Há várias implementações, que traduzem os objetos e seus relacionamentos em comandos SQL:
 - **Hibernate**
 - Toplink
 - ...



Entidade

- É um objeto persistente do domínio de negócio.
 - A classe da entidade representa uma tabela de banco de dados;
 - Um objeto da entidade representa uma linha na tabela.





Mapeamento de Entidades: etapa 1

- Incluir bibliotecas no projeto NetBeans.
 - **Hibernate JPA**
 - **MySQL JDBC Driver** (ou o driver de outro banco)
- Criar a classe dentro de um pacote **modelo**
- Fazer a classe realizar a interface **java.io.Serializable**

```
package modelo;  
  
import java.io.Serializable;  
  
public class Produto implements Serializable{
```

- Adicionar **anotações de mapeamento** na classe.
 - As anotações são do pacote **javax.persistence**

```
@Entity  
@Table (name="produto")  
public class Produto implements Serializable{
```



Mapeamento de Entidades: etapa 2

- Adicionar **anotações de mapeamento** nos atributos.

```
@Entity
@Table(name="produto")
public class Produto implements Serializable{
    @Id
    @GeneratedValue
    private int id;
    @Column(name="nome")
    private String nome;
    @Column(name="descricao")
    private String descricao;
    @Column(name="valor_unitario")
    private float precoUnitario;

    // métodos get() e set()
}
```



Anotações para Mapeamento de Entidades

- **@Entity**
 - Define que a classe é uma entidade persistente e será mapeada para uma tabela de banco de dados
- **@Table**
 - Define a tabela da entidade. Opcional (padrão = nome classe)
- **@Column**
 - Mapeia o atributo para uma coluna da tabela.
 - **name**: identifica o nome da coluna. Opcional (padrão = nome atributo)
- **@Id**
 - Identifica o atributo/coluna que é a chave primária da entidade.
 - Caso nome da coluna seja diferente do atributo, deve-se utilizar @Column conjuntamente para identificar a coluna.
- **@GeneratedValue**
 - Define que o valor do campo é gerado automaticamente pelo banco



@Column: parâmetros adicionais

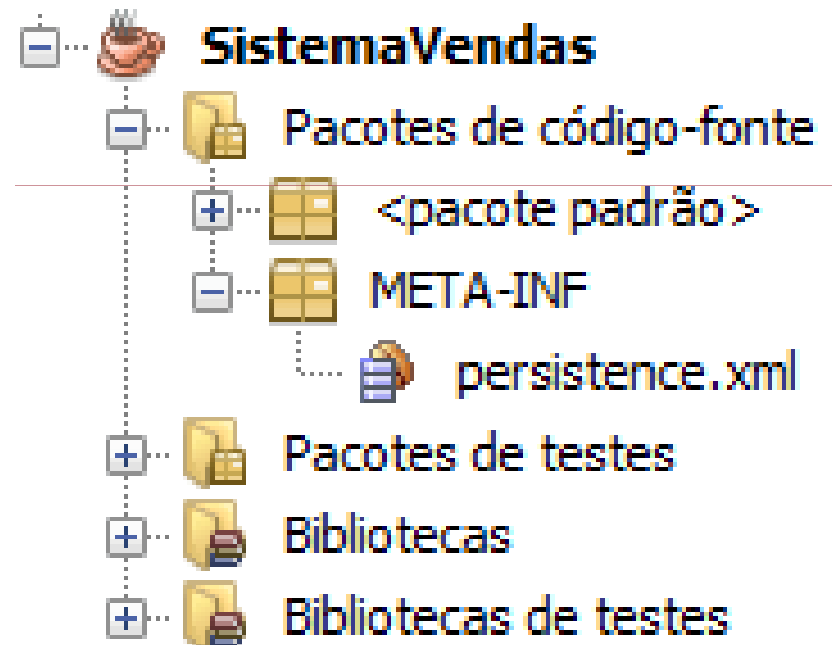
- **@Column**(name="nome", lenght=10)
 - Validação do tamanho do campo
- **@Column**(name="senha", nullable=false)
 - Validação de campo NOT NULL
- **@Column**(name="valor", precision=5, scale=2)
 - precision: quantidade de dígitos antes da vírgula
 - scale: quantidade de dígitos após a vírgula
- **@Column**(name="qualquer", insertable=false, updatable=false)
 - Restringe inserção e atualização do campo



Configuração da Persistência

Persistence Unit (Unidade de Persistência)

- A unidade de persistência define:
 - qual implementação de JPA é utilizada: Hibernate
 - parâmetros para conexão com o banco de dados
 - quais classes são mapeadas para o banco de dados
- Fica em: **META-INF/persistence.xml**





Configuração da Persistência no NetBeans

- transparências auxiliares



Uso do contexto de persistência (1)

- Operações de persistência são realizadas a partir do contexto de persistência – **Entity Manager**.
 - select / insert / update / delete
- o contexto de persistência funciona como um “cache” :
 - contém as entidades relacionadas com o banco – **gerenciadas**.
- Recuperação do contexto de persistência
 - considerando a unidade de persistência:

`<persistence-unit name="SistemaVendasPU" ... >`

- contexto de persistência – Entity Manager:

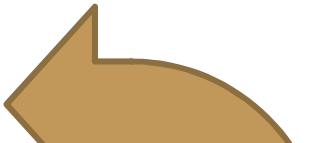
```
EntityManagerFactory emf =  
    Persistence.createEntityManagerFactory("SistemaVendasPU");  
EntityManager em = emf.createEntityManager();
```



Uso do contexto de persistência (2)

- Inserindo entidades no contexto: **persist()**

```
EntityManagerFactory emf =  
    Persistence.createEntityManagerFactory("SistemaVendasPU");  
EntityManager em = emf.createEntityManager();  
  
Produto prod1 = new Produto();  
prod1.setNome("Açucar");  
prod1.setDescricao("Açucar de cana refinado");  
prod1.setPrecoUnitario(2);  
  
em.persist(prod1);  
  
em.close();  
emf.close();
```



- Fechando o contexto para liberar recursos.



Uso do contexto de persistência (3)

- Por padrão, toda transação é finalizada com **rollback**.
 - Este é o motivo que o insert “não inseriu”.
- Para efetivar as operações, é necessário iniciar uma transação e finalizar com **commit**.

```
Produto prod1 = new Produto();  
// setar os atributos do produto...
```

```
em.getTransaction().begin();
```

```
em.persist(prod1);
```

```
em.getTransaction.commit();
```

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

```
emf.close();
```



Uso do contexto de persistência (4)

- Buscar entidades no banco via chave primária: **find()**
 - informar a classe e o valor da chave primária.
 - não é necessário transação

```
Produto prod2 = em.find(Produto.class, 1);  
System.out.println("Id: "+prod2.getId()+" Nome: "+prod2.getNome());
```

- Remoção de entidades: **remove()**
 - é necessário ter o objeto para conseguir removê-lo.

```
em.getTransaction().begin();  
Produto prod3 = em.find(Produto.class, 3);  
em.remove(prod3);  
em.getTransaction().commit();
```



Uso do contexto de persistência (5)

- **Alteração de entidades**
- **Situação 1:** alteração da entidade com o **EntityManager aberto**
 - Basta alterar os atributos do objeto. Os updates são gerados automaticamente pelo contexto de persistência ao se fazer commit.

```
em.getTransaction().begin();  
Produto prod4 = em.find(Produto.class, 4);  
prod4.setDescricao("TV de LED 40 polegadas");  
em.getTransaction().commit();
```

- **Situação 2:** alteração da entidade com diferentes EntityManagers
 - Ex: buscou a entidade por um EntityManager e vai atualizar por outro
 - É necessário utilizar o método **merge()** antes do commit()



Uso do contexto de persistência (6)

- **Alteração de entidades – Situação 2**

```
EntityManagerFactory emf1 =  
    Persistence.createEntityManagerFactory("SistemaVendasPU");  
EntityManager em1 = emf1.createEntityManager();  
Produto prod4 = em1.find(Produto.class, 4); // buscar pelo entity manager 1  
em1.close();  
emf1.close(); // fechando entity manager 1  
  
prod4.setDescricao("TV de LED 40 polegadas "); // alterando sem entity  
// manager aberto  
  
EntityManagerFactory emf2 =  
    Persistence.createEntityManagerFactory("SistemaVendasPU");  
EntityManager em2 = emf2.createEntityManager();  
em2.getTransaction().begin(); // transação no entity manager 2  
em2.merge(prod4); // necessário merge() para colocar a  
em2.getTransaction().commit(); // entidade dentro deste entity manager
```



Uso do contexto de persistência (7)

- **Alteração de entidades – comportamento do merge():**
- O comportamento do merge() baseia-se no ID da entidade:
 - **ID igual a 0 (ou nulo)**
 - **gera insert** → equivalente ao persist()
 - **ID diferente de 0 (ou não nulo)**
 - procura no banco pelo ID da entidade:
 - se existir, **gera update**
 - se não existir, **gera insert** → equivalente ao persist()



Uso do contexto de persistência (8)

- Sincronização de entidades com o banco: **refresh()**
 - para quando a entidade estiver desatualizada com o banco.

```
Usuario prod5 = em.find(Usuario.class, 5);  
System.out.println("Nome: "+prod5.getNome());  
// ocorreu alteração no banco, entidade desatualizada!  
// ex: outro sistema fez alteração no produto 5  
em.refresh(prod5); // para atualizar a entidade  
System.out.println("Nome: "+prod5.getNome());
```



Uso do contexto de persistência (9)

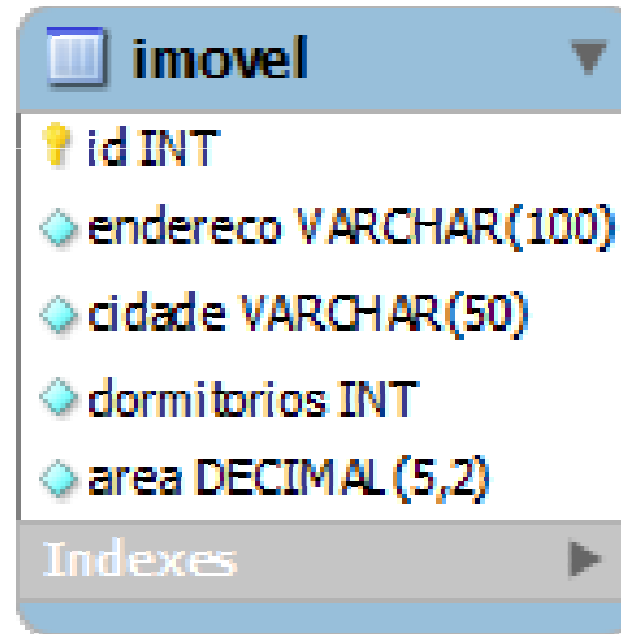
- Consulta básica: buscar todas as entidades no banco:

```
Query consulta1 = em.createQuery("select p from Produto p");  
List<Produto> produtos = consulta1.getResultList();  
for(Produto prod : produtos){  
    System.out.println("Nome: "+prod.getNome());  
}
```



Exercício

- Criar uma classe para representar a entidade abaixo.



- Criar um sistema capaz de realizar as seguintes operações:
 - cadastrar Imovel
 - alterar Imóvel
 - remover Imóvel pelo código (id)
 - listar todos os imóveis cadastrados



Bibliografia

- BAUER, Christian; KING, Gavin. **Java Persistence com Hibernate**. Rio de Janeiro: Ciência Moderna, 2007. 844 p.
- BURKE, Bill; MONSON-HAEFEL, Richard. **Enterprise JavaBeans 3.0**. 5.ed. São Paulo: Prentice Hall, 2007. 538 p.
- **The Java EE 6 Tutorial**, parte VI (Persistence)
 - <http://download.oracle.com/javaee/6/tutorial/doc/>