UNIP
UNIVERSIDADE PAULISTA

Aplicações de Linguagem de Programação Orientada a Objetos

MÓDULO 8

Professor: Me. Pablo I. Gandulfo
Data: 01/08/2020
Versão: 1.1

Professor: Me. Pablo I. Gandulfo
Data: 01/08/2020
Versão: 1.1

Hibernate

- Mapeamento Objeto-Relacional
- Hibernate
- JPA
  - Arquitetura
  - Principais Conceitos
- Passos Envolvidos no Mapeamento Objeto-Relacional
  - Herança
  - Referência Um-pra-um
  - Referência Um-pra-muitos
  - Referência Muitos-pra-muitos
- Exercícios

Mapeamento Objeto-Relacional ou ORM - *Object/Relational Mapping*

- Com o objetivo de persistir dados, normalmente uma aplicação faz uso de um mecanismo de Mapeamento Objeto-Relacional

- Determinadas classes do sistema denominadas **persistentes** são modeladas especificamente para referenciar (consultar ou armazenar) dados em um SGBD, representando as tabelas do banco de dados (além de relações, *sequences* e outros objetos do BD)

- **Um ORM consegue traduzir** operações de **consulta**, **inclusão**, **alteração** e **exclusão** dessas classes persistentes **em comandos SQL**, de forma automática

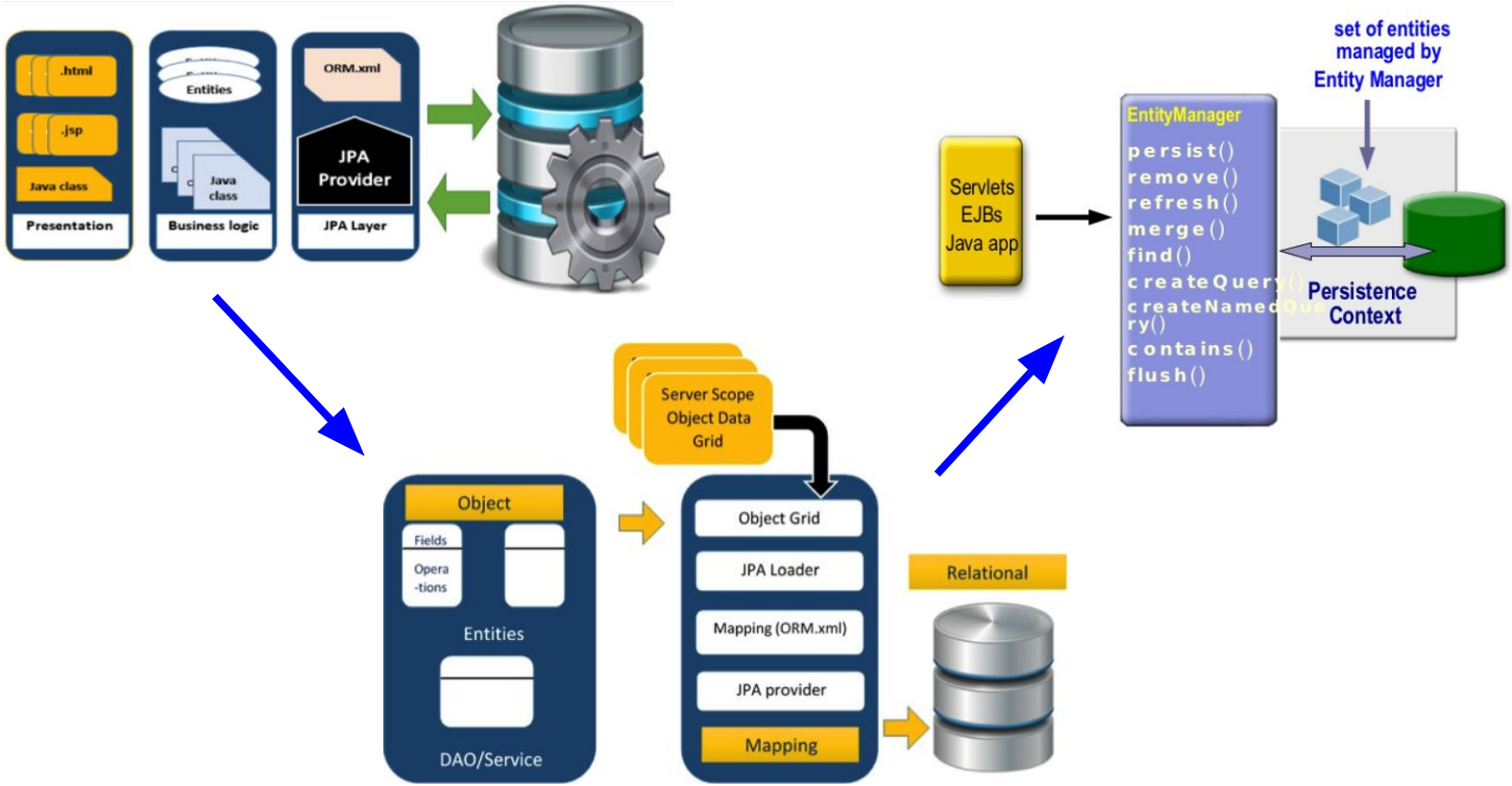
Hibernate

- O Hibernate é um dos ORM’s mais populares do mercado

- Ele também implementa a especificação JPA, um padrão definido para os ORM’s seguirem, uniformizando a maneira como são utilizados

- Existem outros ORM’s no mercado:
  - EclipseLink (Oracle TopLink)
  - Apache OpenJPA
  - Batoo JPA
  - DataNuclues Access Platform

JPA - Arquitetura



JPA – Principais Conceitos

- *Persistent Unit*: define configurações de acesso ao BD e outros parâmetros envolvidos
- *Entity Manager*: responsável por executar operações de consulta, inclusão, alteração e exclusão de entidades (*Entity*) e manter o ciclo de vida desses objetos
- *Entity*: uma classe persistente que, através de anotações no código ou configurações num arquivo xml, define como o seu mapeamento se dará aos objetos de banco de dados
  - *Persistent Field*: cada campo persistente é mapeado para uma ou mais colunas da tabela do banco de dados
  - *Relationship*: define uma relação entre classes que é mapeada em uma ou mais relações de tabelas do banco de dados, considerando a cardinalidade de um pra um (1:1), um pra muitos (1:N), muitos pra um (N:1) e muitos pra muitos (N:N)

Hibernate e JPA – Exemplo

- Arquivo Script SQL - Aula 8.txt => script da tabela de BD utilizada
- Bibliotecas necessárias => disponíveis na pasta \Fontes\Aula 8\lib
- Arquivo de configurações \META-INF\persistence.xml => define o Persistent Unit de nome ‘ExemploJPA’
- Classe hibernate.Aluno => classe persistente que representa a tabela Aluno
- Classe ExemploJPA => classe que executa operações de consulta, inclusão, alteração e exclusão de Aluno

Mapeamento entre O. O. e BD

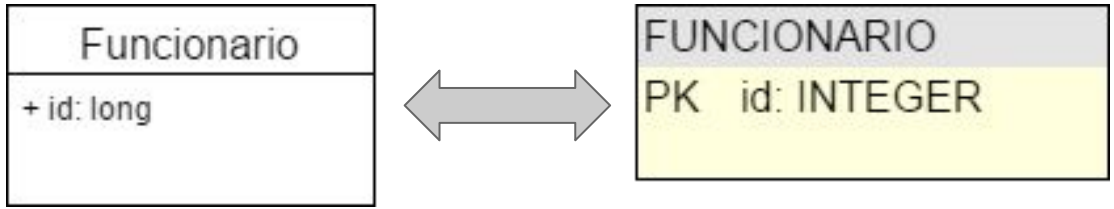
- Algumas considerações devem ser feitas ao realizar essa “tradução”:
  - O modelo de classes precisa ser mapeado para o modelo relacional, ou vice-versa, dependendo do caso
  - Nem todas as relações são facilmente traduzidas (ex.: Herança), e nem todas as propriedades são diretamente convertidas em colunas
  - Algumas regras de normalização de BD devem ser revisadas em detrimento de regras ou considerações ao especificar as classes
  - Alguns conceitos devem ser considerados no mapeamento, como a identificação única de registros através de chaves primárias das tabelas

Passos Envolvidos no Mapeamento Objeto-Relacional

- Realize estes passos para mapear o modelo de classes no modelo de Banco de Dados:
  1. Definir a identidade única dos objetos e registros
  2. Mapear classes e relações em tabelas
  3. Mapear propriedades em colunas

Definir a Identidade Única dos Objetos e Registros

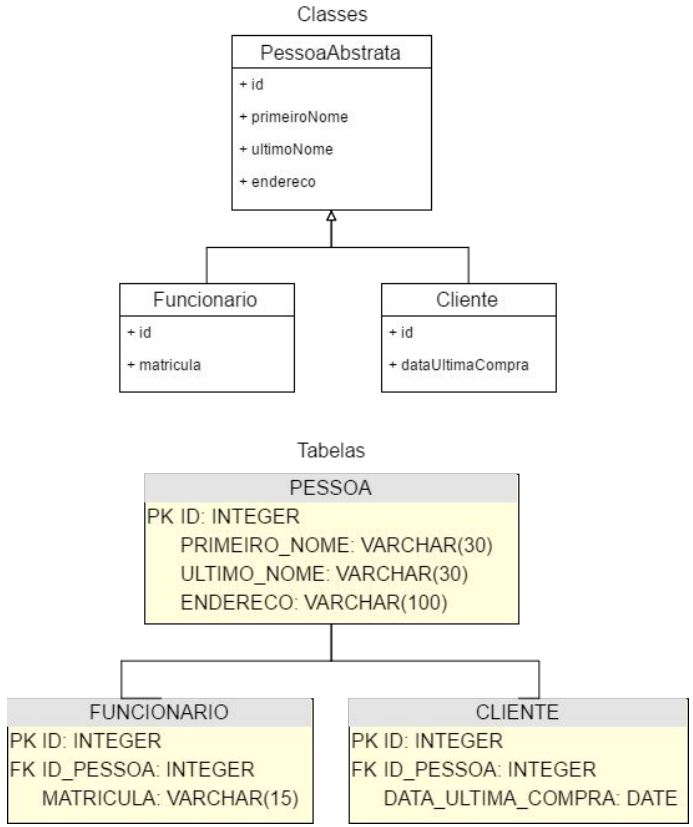
- Uma boa prática adotada em modelos relacionais é de definir uma chave primária para as tabelas que não dependa de regras de negócio. Portanto, utilizar uma única coluna, normalmente de nome **id**, do tipo inteiro longo.
- Além de evitar refatoração de chaves primárias ao alterar colunas significativas da tabela, facilita o mapeamento da chave no modelo de classes, tanto quanto para estabelecer relações entre essas entidades



Mapear Classes e Relações em Tabelas

- Para uma classe que não possui relações de herança ou de referência, o mapeamento é feito diretamente numa tabela do BD
- Para mapear relações de herança ou referência entre classes, as seguintes abordagens podem ser utilizadas:
  - **Herança**: uma tabela para cada classe
  - **Herança**: uma tabela para cada classe concreta
  - **Herança**: uma tabela para a hierarquia inteira das classes
  - **Referência Um-pra-um**: uma tabela para cada classe
  - **Referência Um-pra-muitos**: uma tabela para as duas classes
  - **Referência Um-pra-muitos**: uma tabela para cada classe
  - **Referência Muitos-pra-muitos**: uma tabela adicional para a relação

Relação de Herança: Uma Tabela para Cada Classe

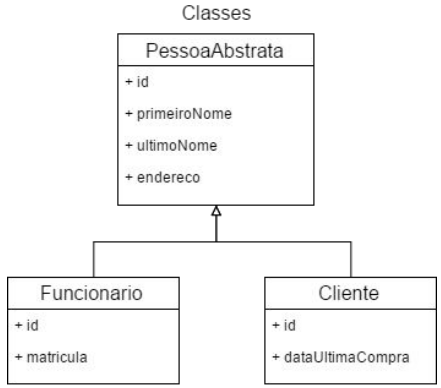


Relação de Herança: Uma Tabela para Cada Classe

- Vantagens:
  - Esta abordagem é a que mais se aproxima dos conceitos de orientação a objetos
  - Suporta polimorfismo
  - Manutenção é mais fácil porque não há duplicação

- Desvantagens:
  - Existem muitas tabelas para manter
  - Demora mais para ler e escrever nas tabelas
  - Ler dados diretamente das tabelas é mais difícil, já que requer utilizar a relação entre elas

Relação de Herança: Uma Tabela para Cada Classe Concreta



Tabelas

FUNCIONARIO
PK ID: INTEGER
PRIMEIRO_NOME: VARCHAR(30)
ULTIMO_NOME: VARCHAR(30)
ENDERECO: VARCHAR(100)
MATRICULA: VARCHAR(15)

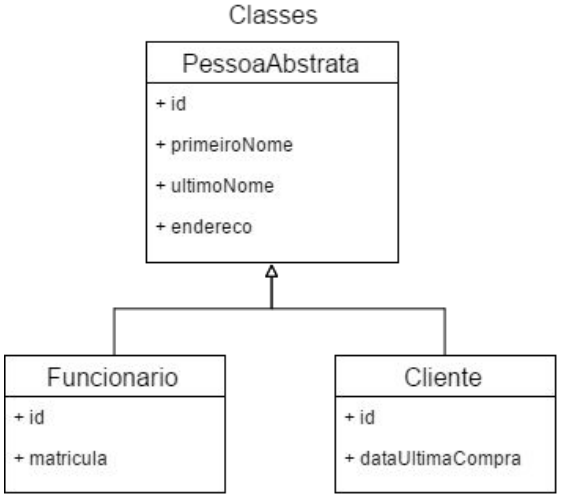
CLIENTE
PK ID: INTEGER
PRIMEIRO_NOME: VARCHAR(30)
ULTIMO_NOME: VARCHAR(30)
ENDERECO: VARCHAR(100)
DATA_ULTIMA_COMPRA: DATE

Relação de Herança: Uma Tabela para Cada Classe Concreta

- Vantagens:
  - Ler dados diretamente das tabelas é fácil

- Desvantagens:
  - Se a classe ancestral for modificada, todas as tabelas também serão
  - Definições de dados estão duplicadas

Relação de Herança: Uma Tabela para a Hierarquia Inteira das Classes

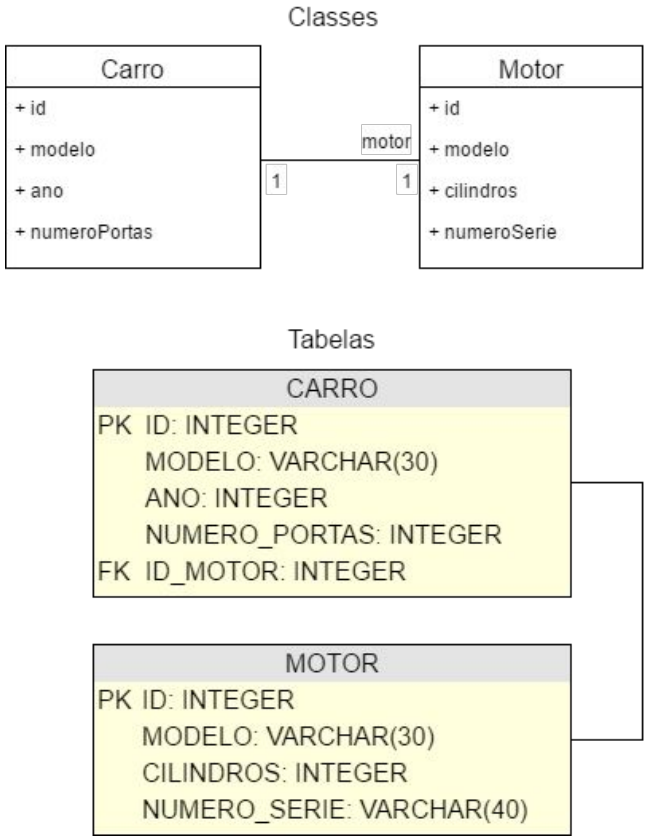


Relação de Herança: Uma Tabela para a Hierarquia Inteira das Classes

- Vantagens:
  - O modelo é fácil de entender
  - Polimorfismo é suportado
  - Ler dados diretamente das tabelas é fácil

- Desvantagens:
  - A tabela pode ficar extremamente grande em termos de colunas e dados
  - É mais difícil impor regras particulares a uma classe filha específica
  - Manutenções em uma das classes podem ser mais trabalhosas, já que podem afetar colunas e dados das outras classes

Relação de Referência Um-Pra-Um: Uma Tabela para Cada Classe

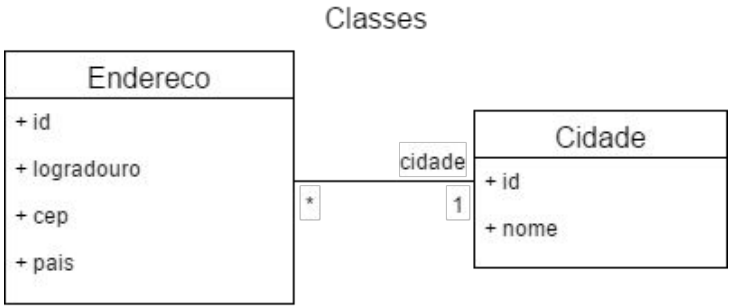


Relação de Referência Um-Pra-Um: Uma Tabela para Cada Classe

- A classe Carro e a classe Motor tem uma relação do tipo um-pra-um, sendo que cada classe foi mapeada numa tabela separada

- O ID do Carro pode estar na tabela do Motor como uma FK, ou o ID do Motor pode estar na tabela Carro como uma FK

Relação de Referência Um-Pra-Muitos: Uma Tabela para as Duas Classes



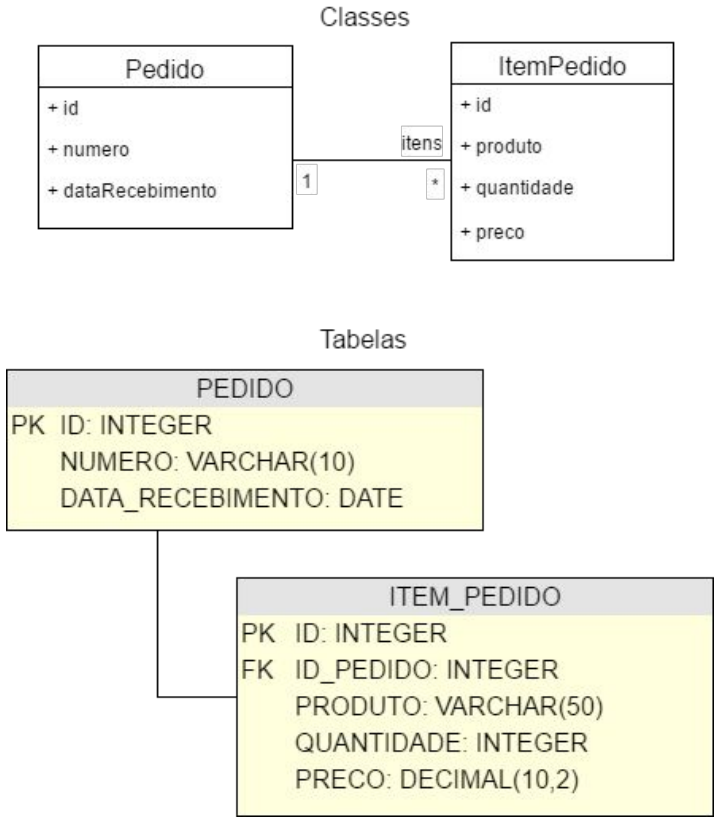
Tabelas

ENDERECO
PK ID: INTEGER
LOGRADOURO: VARCHAR(80)
CEP: VARCHAR(8)
PAIS: VARCHAR(50)
ID_CIDADE: INTEGER
NOME_CIDADE: VARCHAR(60)

Relação de Referência Um-Pra-Muitos: Uma Tabela para as Duas Classes

- Exemplo:
  - Uma classe Endereço possui uma Cidade
  - É possível colocar a Cidade numa classe separada, para uma melhor validação dos possíveis valores que ela pode receber
  - Ao mapear para a tabela relacional, serão combinadas as classes Endereço e Cidade numa só tabela

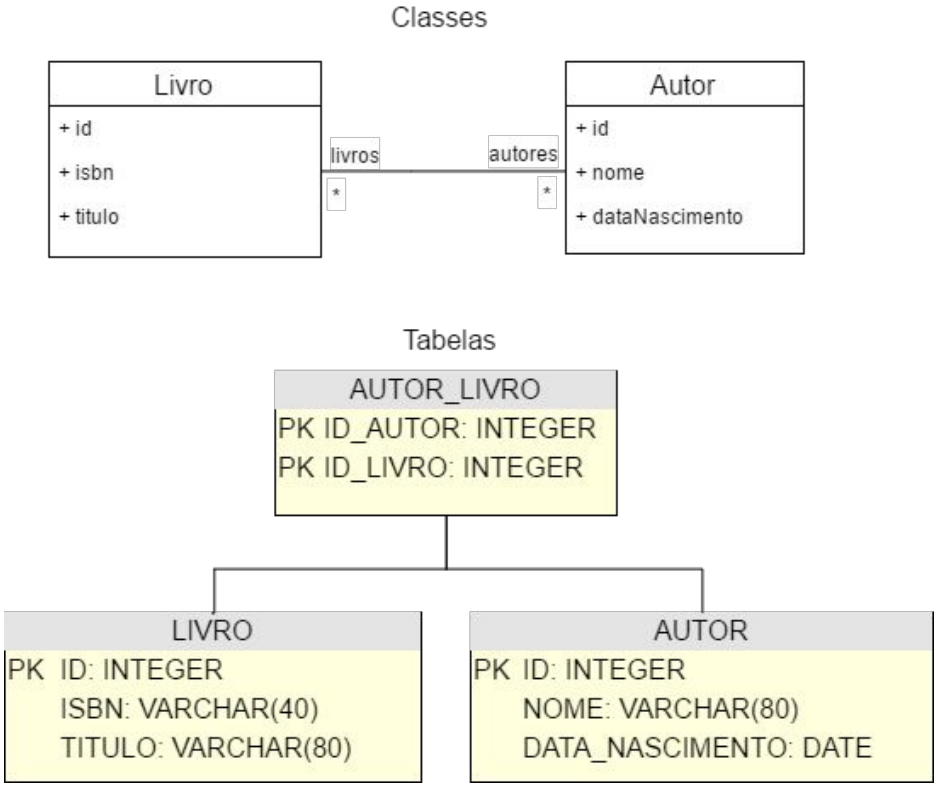
Relação de Referência Um-Pra-Muitos: Uma Tabela para Cada Classe



Relação de Referência Um-Pra-Muitos: Uma Tabela para Cada Classe

- Existe uma referência à uma coleção de objetos de uma classe (muitos registros) a partir do objeto de outra classe (um registro)
- Num sistema de pedidos de compra, um pedido possui um ou vários itens (de pedido de compra)
- O campo ID do Pedido é adicionado na tabela de Item de Pedido

Relação de Referência Muitos-pra-Muitos: Uma Tabela Adicional para a Relação



Relação de Referência Muitos-pra-Muitos: Uma Tabela Adicional para a Relação

- Autor e Livro possuem uma relação muitos-pra-muitos

- Nesses casos, é possível criar uma tabela de junção contendo o ID do Autor e o ID do Livro

Mapear Propriedades em Colunas

Tipo JAVA	Tipo SQL	Tipo JAVA	Tipo SQL
boolean	BIT	String	CHAR
byte	TINYINT		VARCHAR
short	SMALLINT		LONGVARCHAR
int	INTEGER	byte[]	BINARY
long	BIGINT		VARBINARY
float	REAL		LONGVARBINARY
double	FLOAT	java.sql.Date	DATE
	DOUBLE	java.sql.Time	TIME
java.math.BigDecimal	NUMERIC	java.sql.Timestamp	TIMESTAMP
	DECIMAL		

Exercício 1

- Crie a interface genérica DAO e a implementação dessa interface que permita realizar as seguintes operações na tabela Funcionario ou Celular **através de JPA**:
  - Object consultar(int id)
  - boolean incluir(Object o)
  - boolean alterar(Object o)
  - boolean excluir(int id)

- Confirme que o mesmo código que aciona os métodos e ilustra o uso desse DAO desenvolvido na aula anterior em JDBC é aplicável para esta versão do DAO em JPA, sem modificar qualquer linha de código