JPA – JAVA PERSISTENCE API

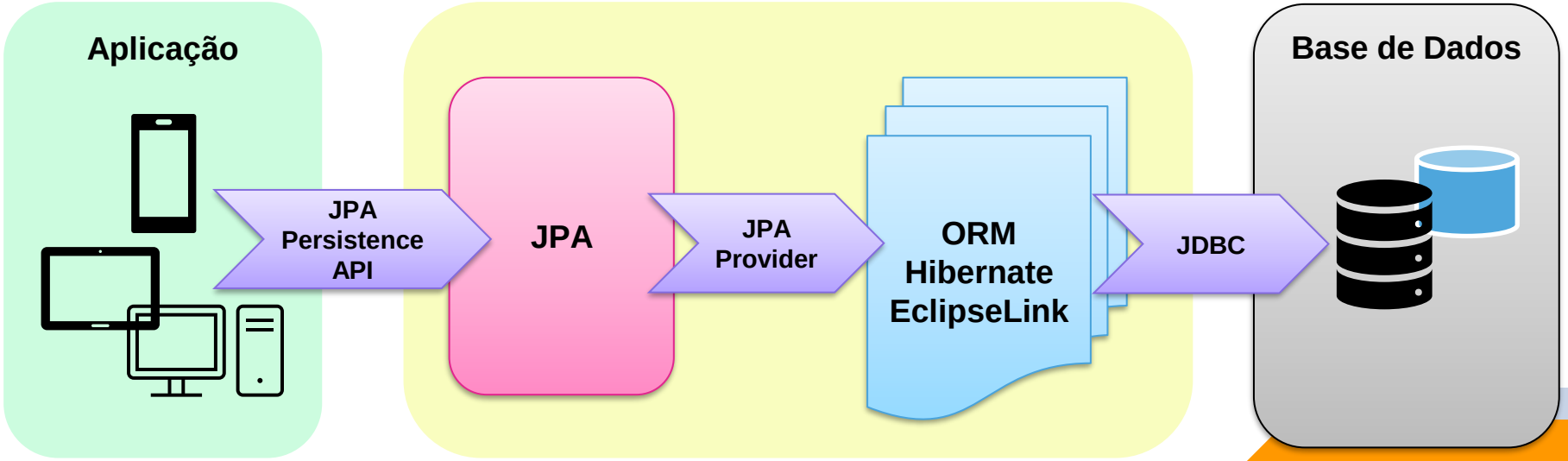
PROF. ELIANE RODRIGUES MARION SANTA ROSA
PROF. ROGÉRIO DE MORAIS
PROF. FÁTIMA

1

JPA

JPA

- JPA é apenas uma especificação: uma série de interfaces, que precisamos de uma implementação (JPA Provider) para poder utilizá-la



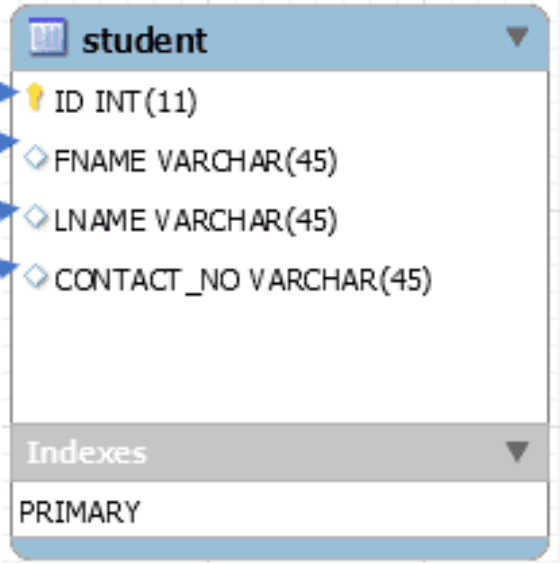
JPA

Student entity

Each **entity** represents a table in a relational database

```
@Entity(name="STUDENT")
public class Student {

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO, generator="native")
    @GenericGenerator(name = "native", strategy = "native")
    @Column(name = "ID")
    private Long studentId;

    @Column(name = "FNAME")
    private String firstName;

    @Column(name = "LNAME")
    private String lastName;

    @Column(name = "CONTACT_NO")
    private String contactNo;
```

Student table



2

Anotações JPA

Conexão com BD

- Antigamente o JAVA utilizava o JDBC (Java Database Connectivity) para conectar com o Banco de dados.
- Tudo era feito “na mão”, desde a abertura do banco, todos os métodos CRUD, até o fechamento.

Conexão com BD

- Criaram o Hibernate => ORM (Mapeamento Objeto-Relacional), que é uma técnica que permite “espelhar” as tabelas de um banco de dados relacional em classes de uma linguagem de programação O.O..
- Hibernate foi um dos primeiros mapeamentos para o Java.
- A Oracle padronizou porque começou a surgir vários frameworks diferentes.

Conexão com BD

- Especificação JPA (Java Persistence API), foi criada pela Oracle e é um framework com base no padrão Oracle. É uma **especificação.**

JPA

- **Precisamos fazer o espelhamento do banco de dados com a classe Java.**

```java
package br.com.etechoracio.jpa.entity;

import lombok.Getter;
import lombok.Setter;

@Getter
@Setter
public class Veiculo {

    private int id;
    private String fabricante;
    private String modelo;
    private int anoFabricacao;
    private int anoModelo;
    private float preco;

}
```

```sql
CREATE TABLE TBL_VEICULO(
  ID_VEICULO BIGINT PRIMARY KEY IDENTITY,
  TX_FABRICANTE VARCHAR(30) NOT NULL,
  TX_MODELO VARCHAR(20) NOT NULL,
  NR_ANO_FABRICACAO INT NOT NULL,
  NR_ANO_MODELO INT NOT NULL,
  VLR_PRECO FLOAT NOT NULL
);
```

Respeitando as diferenças

- É importante durante este processo respeitar as diferenças das convenções dos nomes entre o mundo do banco de dados e o mundo das linguagens de programação, no nosso caso Java.

Banco de dados	**Java**
snake_case	camelCase
comum ter prefixo	não tem prefixo

	Tabela	**Classe**
Nome da Tabela	TBL_VEICULO	Veiculo
Campos da tabela	id_veiculo	id

Respeitando as diferenças

- Outro ponto importante é a relação entre os tipos de campos usados pelo banco e Java.

Tipo de Campo	Classe ou tipos primitivos Java
VARCHAR	String
INT	int, Integer, long, Long
NUMBER	int, Integer, long, Long, double, Double, Big Decimal
DATE	Date ou Calendar
TIMESTAMP	Date ou Calendar
BYTEA | BLOB	Byte[] ou byte[]
CLOB	String

Anotações

- Em Java, as tecnologias mais usadas para esse mapeamento são o JPA (Java Persistence API), que é apenas uma especificação, e o Hibernate, framework que implementa essa especificação.

- Para implementar o ORM com JPA e Hibernate, a técnica mais usada atualmente é incluir **Anotações** (Annotations).

Anotações

- @Entity

Exemplo:

```
@Entity
@Table(name="TBL_VEICULO")
public class Veiculo {
```

Nome completo (Import)	Javax.persistence.Entity
Objetivo	Indicar que a classe será usada para mapear uma tabela do banco de dados.
Onde deve ser incluída	Sobre a classe.
Atributos	Não possui
Como atuou no exemplo	Indicou para o JPA que a classe é uma Entidade ORM, ou seja, que serve para mapear uma determinada tabela do banco de dados

Anotações

- @Table

Exemplo:

```
@Entity
@Table(name="TBL_VEICULO")
public class Veiculo {
```

Nome completo (Import)	Javax.persistence.Table
Objetivo	Configurar as informações da tabela que está sendo espelhada. Se essas anotações forem omitidas, o JPA vai procurar uma tabela com o EXATO nome da classe no banco.
Onde deve ser incluída	Sobre a classe.
Atributos	name : nome da tabela no banco de dados (Obrigatório). Outros atributos: catalog, schema, indexes, uniqueConstraints.
Como atuou no exemplo	Indicou o nome da tabela no banco de dados como sendo “tbl_veiculo”.

Anotações

- @Id

Exemplo:

```
public class Veiculo {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "ID_VEICULO")
    private int id;
```

Nome completo (Import)	Javax.persistence.Id
Objetivo	indica qual atributo da classe será mapeado para a **Chave Primária** da tabela.
Onde deve ser incluída	Sobre um atributo. Obrigatório em pelo menos 1 (um) atributo.
Atributos	Não possui.
Como atuou no exemplo	Indicou que o atributo **id** está mapeado para o campo de **Chave Primária** na tabela.

Anotações

- @GeneratedValue Exemplo:

```
public class Veiculo {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "ID_VEICULO")
    private int id;
```

Nome completo (Import)	Javax.persistence.GeneratedValue
Objetivo	configura a forma de preenchimento automático do valor do campo da Chave Primária. Se não for usada, o programa deve configurar “manualmente” o valor do atributo da Chave Primária.
Onde deve ser incluída	Sobre um atributo*. Opcional.
Atributos	**strategy** (opcional): indica a estratégia para a geração do valor do atributo. Os tipos de estratégias são os valores da *enum AUTO, IDENTITY, SEQUENCE, TABLE.* **generator** (opcional, porém, obrigatório para **SEQUENCE**)
Como atuou no exemplo	indicou que o campo **id** terá seu valor preenchido automaticamente com uso da estratégia **IDENTITY**.

Os tipos de estratégias são os valores da *enum* **javax.persistence.GenerationType**:

- **AUTO**: aponta que a estratégia-padrão de preenchimento automático do banco de dados configurado será utilizada. Em alguns bancos, a Chave Primária "cresce sozinha", ou seja, possui um valor com **autoincremento**. Para outros, o JPA pegará o maior valor atualmente na tabela e usará mais 1. Se o atributo **strategy** for omitido, esta estratégia é a que será usada.

- **IDENTITY**: em alguns bancos, a Chave Primária "cresce sozinha", ou seja, possui um valor com **autoincremento**. O IDENTITY aponta que o JPA irá gerar uma instrução de insert apropriada para o banco de dados configurado para que use esse recurso no momento da criação de um novo registro.

Os tipos de estratégias são os valores da *enum* **javax.persistence.GenerationType**:

- **SEQUENCE**: alguns bancos de dados não possuem o recurso de autoincremento de valor. Assim, uma forma de fazer o valor "crescer" de forma consistente é consultar o novo valor de uma **sequence** no banco de dados. Essa opção indica e configura o uso desse recurso para a obtenção do valor que será usado na Chave Primária.

- **TABLE**: opção muito parecida com a **SEQUENCE**. A diferença é que com ela se indica uma **tabela** e não uma **sequence** de onde se pega o novo valor que será usado na Chave Primária.

Atributo generator

- **generator** (opcional, porém, obrigatório para SEQUENCE): caso tenha usado o SEQUENCE no strategy, nesse atributo deve indicar o mesmo valor que usou no name da anotação @SequenceGenerator (descrita a seguir). Caso tenha usado o TABLE no strategy, nesse atributo deve apontar o mesmo valor que usou no name da anotação @TableGenerator

Anotações

- @Column Exemplo:

```
public class Veiculo {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "ID_VEICULO")
    private int id;
```

Nome completo (Import)	Javax.persistence.Column
Objetivo	Mapeia uma coluna da tabela junto a um atributo na classe.
Onde deve ser incluída	Sobre um atributo*. Opcional.
Atributos	**name, length, precision, scale, nullabel, unique, insertable, updatable. (**Serão descritos no próximo slide).
Como atuou no exemplo	Indicou os vários atributos que estão mapeados para campos da tabela. Sobre o atributo **id,** mostrou que seu respectivo campo na tabela era **id_veiculo**;

Atributos da anotação @Column:

- **name**: (opcional): indica qual o nome do campo na tabela. Se omitido, o JPA entenderá que o campo possui exatamente o mesmo nome do atributo. O interessante desse atributo é que um campo pode ter um nome na tabela diferente de seu atributo mapeado na classe.

- **length**: (opcional): aponta o tamanho do campo na tabela. Por exemplo, para um campo **varchar**, esse campo mostraria a quantidade dos caracteres que comporta.

- **precision** (opcional): indica a precisão do campo. Esse atributo se aplica a campos numéricos.

Atributos da anotação @Column:

- **scale**: (opcional): mostra a escala do campo. Esse atributo se aplica a campos numéricos.

- **nullable**: (opcional): aponta se o campo é obrigatório (false) ou se é possível criar/atualizar um valor de um registro, deixando-o em vazio (true).

- **unique** (opcional): indica se o campo deve possuir valor único na tabela, ou seja, se é um campo com a restrição **unique** na tabela.

- **insertable** (opcional): mostra se o campo pode ter valor no momento da criação de um registro. Se esse atributo for false, um eventual valor do atributo da classe será ignorado no momento da criação de um registro.

Atributos da anotação @Column:

- **updatable**: (opcional): aponta se o campo pode ter valor no momento da atualização de um registro. Se esse atributo for false, um eventual valor do atributo da classe será ignorado no momento da atualização de um registro.