JPA ANNOTATIONS

CRIANDO ANOTAÇÕES

```java
@Target({ElementType.FIELD, ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
public @interface Coluna {
    String nome();
    boolean obrigatorio();
    int tamanho() default 255;
}
```

@interface - Palavra reservada para definir uma classe como anotação.
Definir parâmetros de configurações da anotação:
Ex: String nome();
boolean obrigatorio();
int tamanho() default 255; // default - definir padrão quando uso opcional

@Retention - Anotação para classe anotação que determina até quando estará presente.
Valores: SOURCE - até .java
CLASS - até .class
RUNTIME - até jvm

@Target - Escopo da anotação, definindo onde a anotação pode ser utilizada.
Valores: TYPE - classe
FIELD - atributos
METHOD - metodos

ANOTAÇÕES DE CLASSE

```java
@Entity
@Table(name = "TAB_CLIENTE")
@SequenceGenerator(name = "cliente", sequenceName = "SQ_TAB_CLIENTE", allocationSize = 1)
public class Cliente {}
```

@Entity - Defini a classe java como uma entidade no banco de dados.

@Table - Indica qual será o nome da tabela no banco de dados.

@SequenceGenerator - Define que a tabela terá um SEQUENCE para chave-primária da tabela.
name - nome do SequenceGenerator que será usado no atributo que
corresponde a chave-primária
sequenceName - nome da SEQUENCE no banco de dados
allocationSize - incremento para a SEQUENCE

ANOTAÇÕES DE ATRIBUTOS

```java
@Id
@Column(name = "cd_cliente")
@GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "cliente")
private int codigo;

@Column(name = "nm_cliente", nullable = false, length = 100)
private String nome;
```

@Id **-** Indica o atributo como chave-primário no banco de dados.

@Column **-** Define o nome da coluna no banco de dados.
Obs.: não é necessário passar o parâmetro 'nullable' por já estar anotado com @Id

@GeneratedValue **-** Indica o atributo que receberá o SequenceGenerator definido na classe.
strategy – indica como a persistência deve atribuir a chave-primária
generator – SEQUENCE no java, definido no 'name' da SequenceGenerator

OUTRAS ANOTAÇÕES DE ATRIBUTOS

@Temporal **-** Utilizado com atributos do tipo Calendar ou Date para datas no banco de dados.
Ex: @Temporal(TemporalType.DATE)

Valores: DATE – somente a data
TIME – somente as horas e minutos
TIMESTAMP – data e horas

@CreationTimestamp **-** Gravar a data e hora de cadastro no banco de dados de forma automática.

@Enumerated **-** Utilizado com atributos do tipo Enum.
Ex: @Enumerated(EnumType.STRING)

Valores: ORDINAL – armazena a posição do elemento da enum
STRING – armazena o texto do elemento da enum

@Lob **-** Mapear um campo BLOB no oracle para armazenar arquivos no banco de dados.
O atributo deve ser um array de byte.
Ex: @Lob
Private byte[] foto;

@Transient **-** Anotação que informa atributo que não será mapeado no banco de dados.

ANOTAÇÕES DE MÉTODOS

@PrePersist **-** Utilizado para executar o método antes de persistir no banco de dados.

ANOTAÇÕES DE MAPEAMENTOS ENTRE ENTIDADES

Carro.java *(entidade que recebe a FK)*

```java
@OneToOne(cascade = {CascadeType.PERSIST, CascadeType.MERGE}, fetch = FetchType.LAZY)
@JoinColumn(name = "cd_piloto", nullable = false)
private Piloto piloto;
```

Piloto.java

```java
@OneToOne(mappedBy = "piloto", cascade = CascadeType.PERSIST)
private Carro carro;
```

@OneToOne - Refere-se a uma relação 1:1 entre entidades, tendo o relacionamento como Unidirecional ao colocar esta anotação na entidade que recebe a FK, tornando-a "dona" da relação, ou o relacionamento como Bidirecional ao colocar esta anotação também na outra entidade do relacionamento.

- cascade: propriedade que propaga em cascata uma ação no banco de dados para o Relacionamento com CascadeType.
  Valores: - PERSIST: ação de cadastrar (insert)
  - MERGE: ação de atualizar (update)
  - REMOVE: ação de deletar (delete)
  - ALL: permite propagar todas as ações anteriores

- fetch: determina como o relacionamento será carregado (select) com FetchType.
  Valores: - LAZY: carrega o necessário da entidade, e a entidade relacionada quando chamar o atributo relacionado
  - EAGER: carrega todas as informações incluindo entidades relacionadas

- mappedBy: utiliza o nome do atributo FK que foi mapeado inicialmente no 1:1 Unidirecional, transformando o relacionamento em bidirecional.

@JoinColumn - Informa a coluna no banco de dados da outra entidade do relacionamento que será utilizada para fazer o relacionamento.

Carro.java *(entidade que recebe a FK)*

```java
@ManyToOne
@JoinColumn(name = "cd_equipe", nullable = false)
private Equipe equipe;
```

Equipe.java

```java
@OneToMany(mappedBy = "equipe", cascade = CascadeType.ALL)
private List<Carro> carros;
```

@ManyToOne - Refere-se ao mapeamento de uma relação N:1
Ex: muitos carros pertencem a uma equipe

@OneToMany - Refere-se ao mapeamento de uma relação 1:N
Ex: uma equipe possui muitos carros

Equipe.java *(escolher um dos lados do relacionamento para configurar o mapeamento)*

```java
@ManyToMany(cascade = CascadeType.PERSIST)
@JoinTable(name = "T_EQUIPE_CORRIDA",
           joinColumns = @JoinColumn(name = "cd_equipe"),
           inverseJoinColumns = @JoinColumn(name = "cd_corrida"))
private List<Corrida> corridas;
```

Corrida.java

```java
@ManyToMany(mappedBy = "corridas")
private List<Equipe> equipes;
```

@ManyToMany **-** Refere-se a um mapeamento N:N

@JoinTable **-** Anotação que cria a tabela associativa entre as duas entidades relacionadas.
- joinColumns: configura a coluna que armazena a PK/FK da classe que está utilizando para fazer o mapeamento.
- inverseJoinColumns: configura a coluna que armazena a PK/FK da classe do outro lado do relacionamento (classe que está sendo mapeada)