

**FIA/P GRADUAÇÃO**



# JAVA ADVANCED

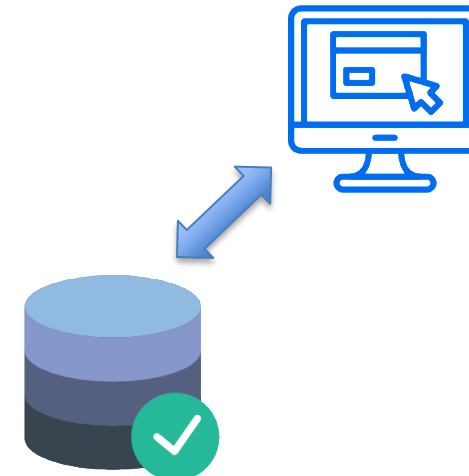


# APIs DE PERSISTÊNCIA

- Em um projeto de **software real** **não** é necessário criar as suas próprias **anotações** para a **persistência de objetos**;
- Existem **APIs** prontas que lidam com o problema;
- Tais APIs são responsáveis, entre outras coisas, pela **transformação dos objetos em declarações SQL** (**INSERT, UPDATE, DELETE, SELECT, ...**).



- **Object Relational Mapper (ORM)** ou **Mapeamento objeto relacional** são frameworks que permitem que **os objetos sejam mapeados** para o **modelo relacional** dos bancos de dados;
- O **ORM** possui **métodos básicos** que realizam a interação entre a **aplicação e o banco de dados**, se responsabilizando por tarefas como o **CRUD**, dessa forma, o desenvolvedor não precisa se preocupar com os **comandos SQL**;
- **ORM não** é exclusivo da **linguagem Java**, está presente em diversas linguagens de programação, como por exemplo:
  - **Java** : JPA, Hibernate, Spring Data;
  - **.NET**: NHibernate, Entity Framework, Dapper;
  - **PHP**: Doctrine 2, ReadBeanPHP, EloquentORM;
  - **Python** : Django, SQLAlchemy;



- JPA define uma interface comum para **persistência de dados** do **Java EE**;
- Oferece uma **especificação** padrão para **mapeamento objeto-relacional** para objetos Java simples, através de **anotações**;
- Pode ser utilizado de forma *standalone*, com **Java SE**;

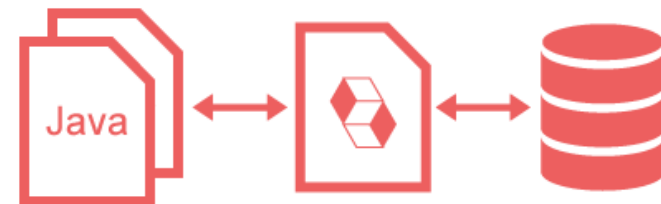
- Exemplos de implementações:

- Hibernate

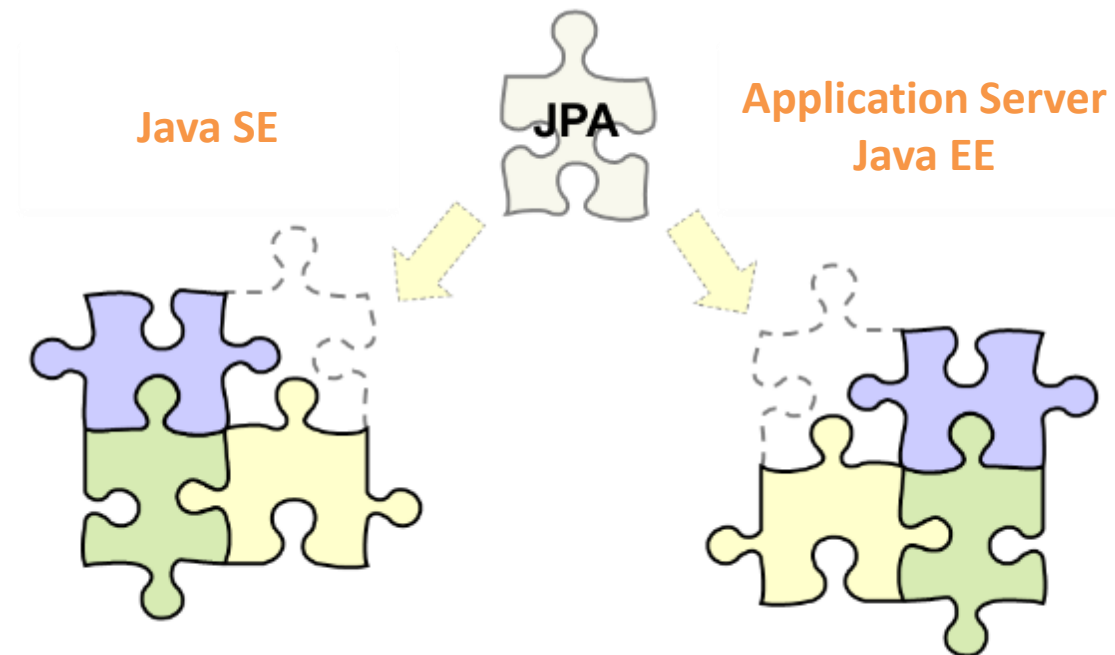
- <http://www.hibernate.org/>

- Toplink

- <https://www.oracle.com/technetwork/middleware/toplink/overview/index.html>



- É possível **utilizar o JPA** em todos os **projetos Java**, em conjunto com outros **frameworks Java**, do **Java EE** ou **não**;
- Dessa forma, o **JPA** se encaixa em **projetos web** com JSP, Servlets, JSF, Spring MVC. É utilizado também em projetos Desktop, Console e etc.

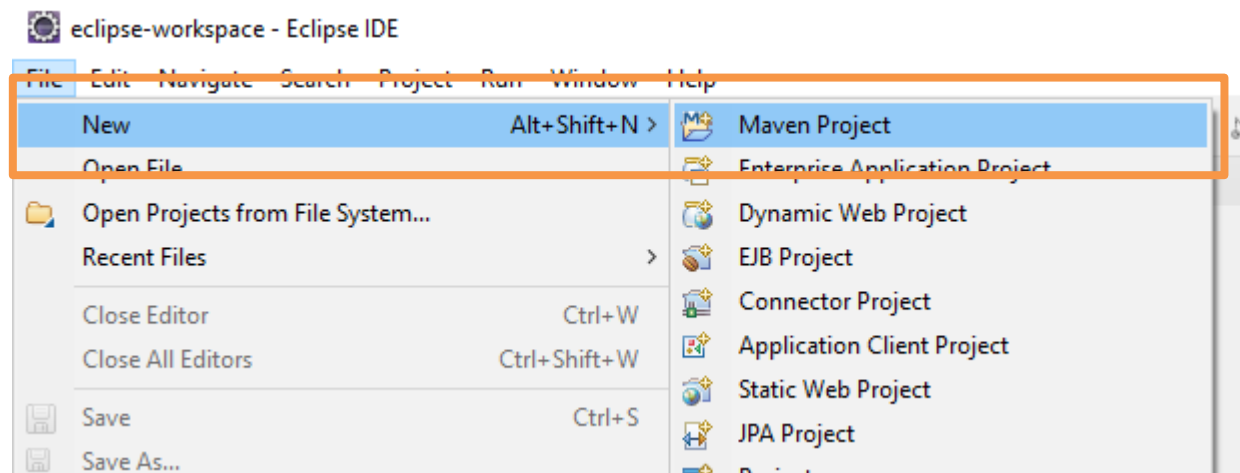




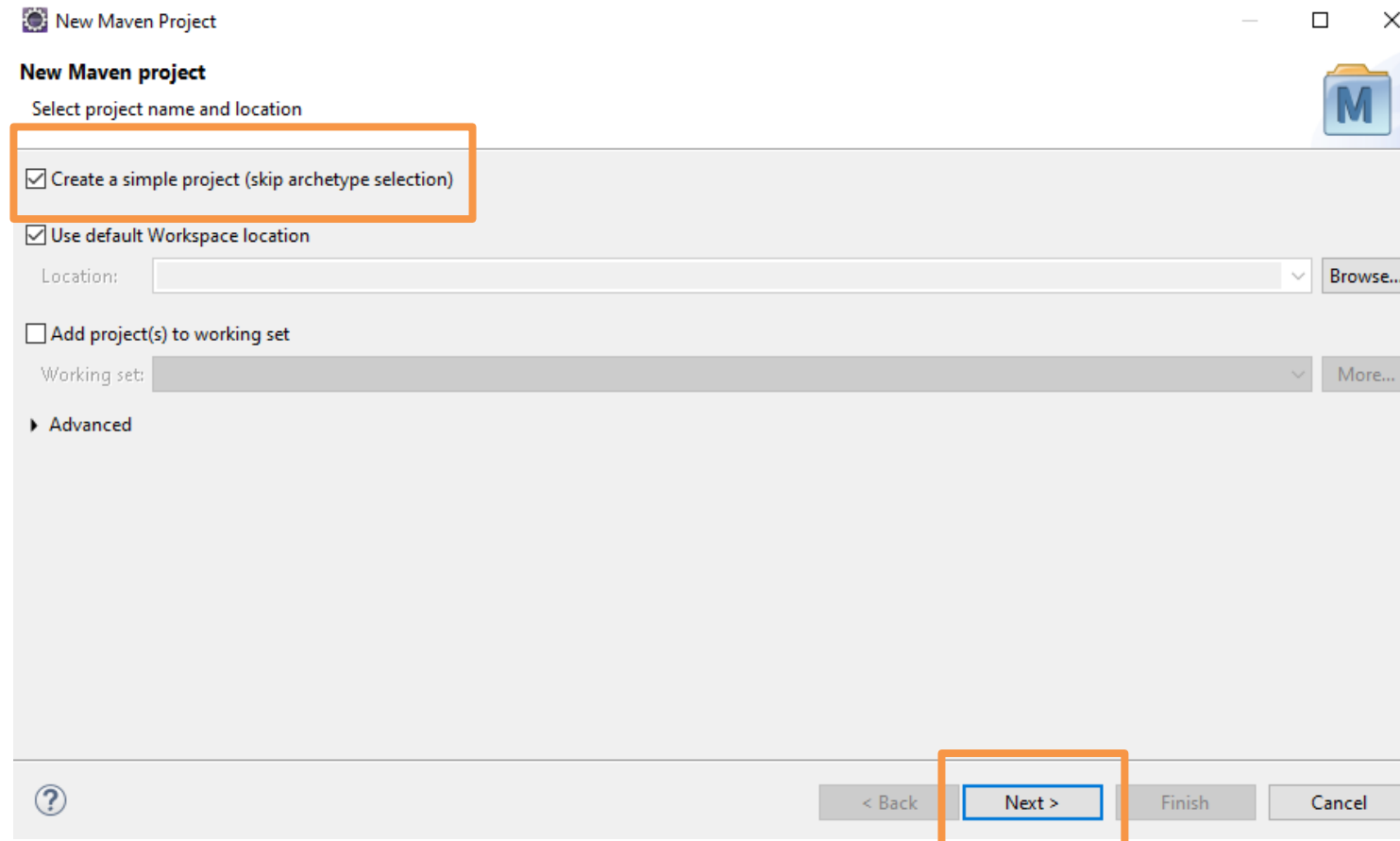
# CRIANDO UM PROJETO JAVA COM JPA



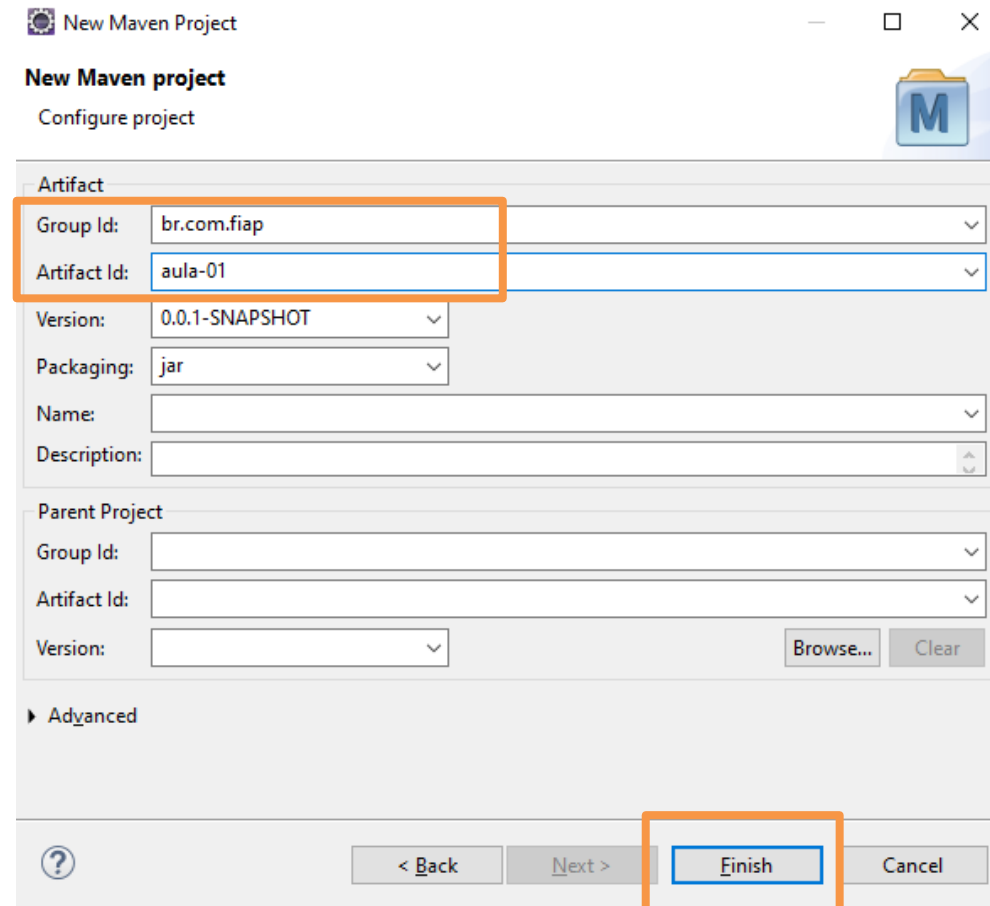
- Crie um **Maven Project**;



- Marque o **checkbox** para criar um projeto simples e clique em **Next**.



- Defina o **Group Id** e o **Artifact Id** e finalize o processo clicando em **Finish**.



The screenshot shows the 'New Maven Project' dialog box. The 'Artifact' section is expanded, and the 'Group Id' and 'Artifact Id' fields are highlighted with an orange box. The 'Group Id' is 'br.com.fiap' and the 'Artifact Id' is 'aula-01'. The 'Version' is '0.0.1-SNAPSHOT' and the 'Packaging' is 'jar'. The 'Parent Project' section is also visible, with 'Group Id', 'Artifact Id', and 'Version' fields. The 'Advanced' section is collapsed. The 'Finish' button at the bottom is highlighted with an orange box.

- No arquivo **pom.xml** podemos configurar as **dependências** (bibliotecas) do projeto;

```
<dependencies>
  <dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.4.12.Final</version>
  </dependency>
  <dependency>
    <groupId>com.oracle.database.jdbc</groupId>
    <artifactId>ojdbc8</artifactId>
    <version>21.1.0.0</version>
  </dependency>
</dependencies>
```

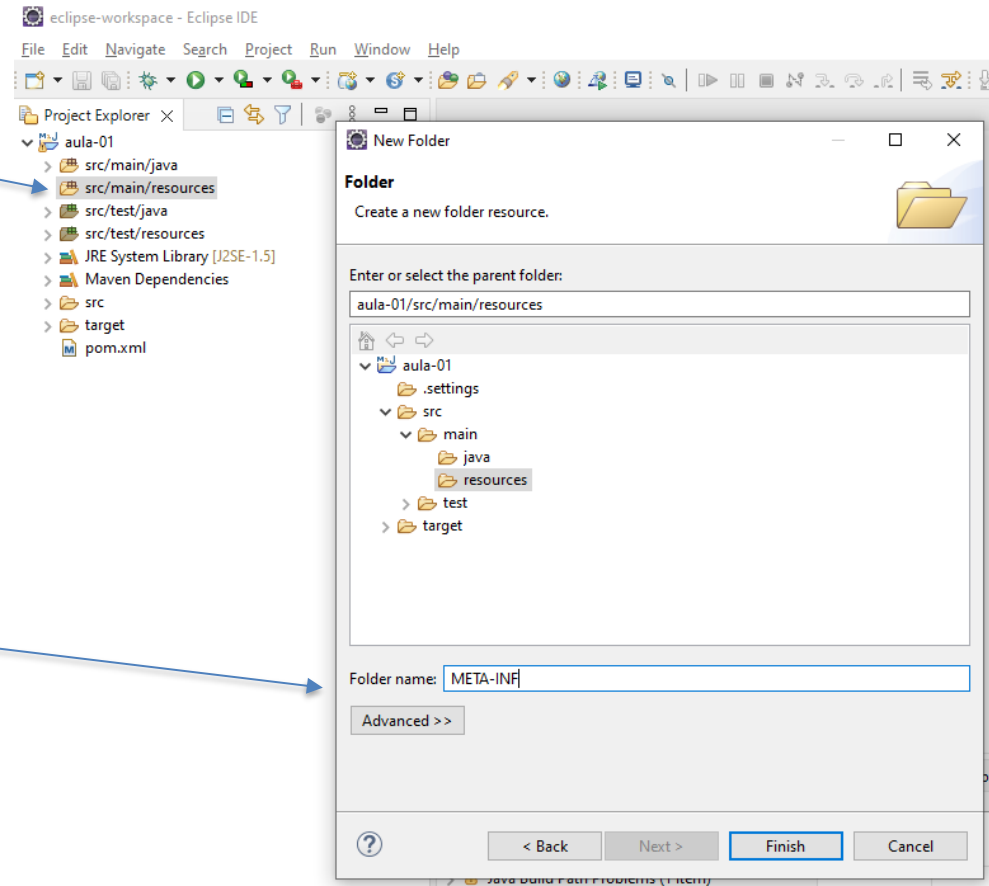
Bibliotecas do **Hibernate**

Driver do banco

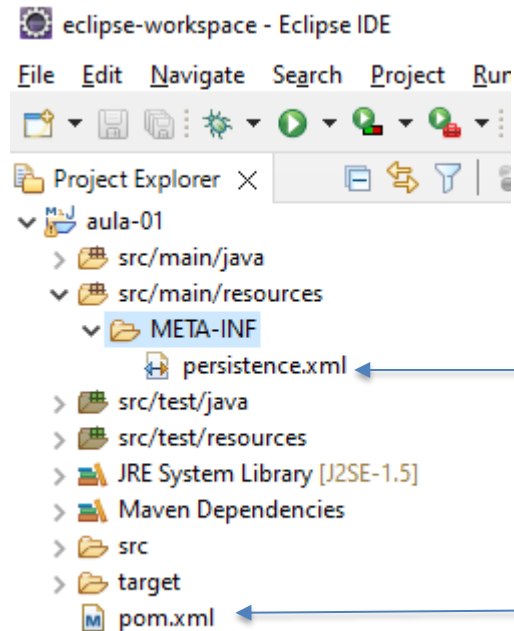
- Na pasta **src/main/resources** crie uma pasta chamada **META-INF**. Atenção, é um **Folder** e não **Source Folder**.

Clique com o botão direito do mouse na pasta e escolha “New” > “Other..” e procure por **Folder**

Defina o nome do diretório, deve ser exatamente **META-INF**

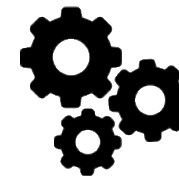


- Precisamos de um arquivo de **configuração** para o **JPA**, esse arquivo deve ficar dentro do diretório **que acabamos de criar**;



Arquivo de configuração do **JPA**, dentro do diretório **META-INF**

Arquivo de configuração do **Maven**



- Neste arquivo definimos a **URL**, **usuário** e **senha** para conectar com o **banco de dados**, o **driver** (jdbc) que será utilizado;
- Podemos configurar também se vamos **criar** o banco, **atualizar** ou só **validar** de acordo com as entidades (classes) do sistema;

```
<persistence-unit name="oracle" transaction-type="RESOURCE_LOCAL">
  <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
  <properties>
    <property name="hibernate.show_sql" value="true" />
    <property name="hibernate.hbm2ddl.auto" value="update" />
    <property name="hibernate.dialect" value="org.hibernate.dialect.Oracle12cDialect" />
    <property name="javax.persistence.jdbc.driver" value="oracle.jdbc.OracleDriver" />
    <property name="javax.persistence.jdbc.user" value="" />
    <property name="javax.persistence.jdbc.password" value="" />
    <property name="javax.persistence.jdbc.url" value="" />
  </properties>
</persistence-unit>
```



# ENTIDADES



- Representam as unidades de **persistência**;
- Correspondem a **classes simples** (POJO) cujo estado pode ser persistido;
- Permitem o acesso aos dados por meio de **métodos gets e sets**;
- Possuem, obrigatoriamente, um **identificador único**;
- Recomendável que implementem a interface **Serializable**;
- Tais classes são **mapeadas** para o **banco de dados** por meio de anotações;
- São como **espelhos do banco de dados**, isto é, uma instância é criada ou alterada primeiramente em memória e posteriormente **atualizada** no **banco de dados**;
- São gerenciadas por um mecanismo de persistência denominado **Entity Manager**;

- As anotações da JPA situam-se no pacote **javax.persistence**;
- A anotação **@Entity** especifica que uma classe é uma entidade;
- O **nome da tabela** da entidade será o **mesmo da classe** com a anotação **@Entity**.

**@Entity**

```
public class Cliente {  
    private int id;  
    private String nome;  
    // métodos get e set  
}
```

- É possível alterar o **nome** da **tabela** associada a entidade através do atributo **name** da *annotation* **@Table**.

```
@Entity
@Table(name="TB_CLIENTE")
public class Cliente {
    private int id;
    private String nome;
    // métodos get e set
}
```



- Define o atributo que representa a **chave primária**;
- As únicas anotações **obrigatórias** para uma entidade são o **@Entity** e **@Id**;

```
@Entity
```

```
@Table(name="TB_CLIENTE")
```

```
public class Cliente {
```

```
    @Id
```

```
    private int id;
```

```
    private String nome;
```

```
    // métodos get e set
```

```
}
```



- Especifica a **estratégia de geração de valores** automáticos para os atributos;
- Normalmente utilizado em conjunto com o atributo anotado com **@Id**;
- Parâmetros:
  - **generator**: nome do gerador de chaves;
  - **strategy**: indica o tipo de estratégia utilizada;
- Tipos mais comuns de **estratégias** de geração:
  - **GenerationType.SEQUENCE**: baseado em *sequence*;
  - **GenerationType.IDENTITY**: campos identidade, *auto increment*;



- Define um gerador de chave primária baseado em *sequence* de **banco de dados**;
- Possui uma associação com o **@GeneratedValue** definido com a estratégia **GenerationType.SEQUENCE**;
- Parâmetros:
  - **name**: nome a ser referenciado pelo @GeneratedValue;
  - **sequenceName**: nome da *sequence* de banco de dados;
  - **allocationSize**: incremento;

**@Entity**

**@SequenceGenerator(name="cliente",  
sequenceName="TB\_SQ\_CLIENTE",allocationSize=1)**

**@Table(name="TB\_CLIENTE")**

public class Cliente {

**@Id**

**@GeneratedValue(strategy=GenerationType.SEQUENCE,  
generator="cliente")**

private int id;

private String nome;

// métodos get e set

}

- Especifica a **coluna** da tabela associada ao **atributo** da entidade;
- Caso não definido, assume-se que a coluna terá o mesmo **nome do atributo**;
- Alguns parâmetros:
  - **Name** - nome da coluna;
  - **Nullable** (default true) - não permite valores nulos;
  - **Insertable** (default true) - atributo utilizado em operações de INSERT;
  - **Updatable** (default true) - atributo utilizado em operações de UPDATE;
  - **Length** (default 255) – atributo utilizado para definir o tamanho do campo, aplicado somente para Strings;



**@Entity**

**@Table(name="TB\_CLIENTE")**

public class Cliente {

**@Id**

**@Column(name="CD\_CLIENTE")**

private int id;

**@Column(name="NM\_CLIENTE", nullable=false)**

private String nome;

// métodos get e set

}

- Indica que o atributo **não** será um **campo na tabela** (Banco de Dados);

**@Entity**

**@Table(name="TB\_CLIENTE")**

public class Cliente {

**@Id**

**@Column(name="CD\_CLIENTE")**

private int id;

**@Column(name="NM\_CLIENTE", nullable=false)**

private String nome;

**@Transient**

private int chaveAcesso;

}

- Especifica o tipo de dado a ser armazenado em propriedades do tipo **Date** e **Calendar**, através dos parâmetros:
  - **TemporalType.TIMESTAMP**: data e hora;
  - **TemporalType.DATE**: somente data;
  - **TemporalType.TIME**: somente hora;



**@Entity**

```
public class Cliente {  
  
    ...  
  
    @Column(name="DT_NASCIMENTO")  
    @Temporal(value=TemporalType.DATE)  
    private Calendar dataNascimento;  
  
}
```

- Permite **mapear** objetos de **grande dimensão** (LOB – Large Object) para a base de dados. Exemplos: imagens, documentos de texto, planilhas, etc..;
- Os bancos de dados oferecem um tipo de dado para tais objetos. Exemplo: **BLOB** no Oracle;
- No objeto, o atributo mapeado normalmente é do tipo **byte[]** (array);

**@Entity**

```
public class Noticia {
```

```
...
```

```
    @Column(name="FL_IMAGEM")
```

```
    @Lob
```

```
    private byte[] imagem;
```

```
}
```



- Propriedades que possuem **valores fixos**, por exemplo, gênero (MASCULINO, FEMININO, etc), dia da semana (SEGUNDA, TERÇA, ...), meses do ano, etc.
- O índice associado ao valor depende da **sequência** que foi declarado no **Enum**;

```
public enum Tipo {  
    OURO, PRATA, BRONZE  
}
```

No exemplo acima, será gravado no banco de dados os valores: OURO = 0, PRATA = 1 e BRONZE = 2

- É possível configurar o valor que será **gravado no banco** para um atributo do tipo **enum**, a **posição** ou o **nome** da constante, através dos parâmetros:
  - **EnumType.ORDINAL** – posição da constante;
  - **EnumType.STRING** – nome da constante;

**@Entity**

```
public class Cliente {
```

```
    ...
```

```
    @Enumerated(EnumType.STRING)
```

```
    private Tipo tipo;
```

```
}
```

# VOCÊ APRENDEU...

---



- Conceito de **ORM** e **JPA**;
- Criar **anotações** Java;
- Criar e configurar um **projeto** com **JPA/Hibernate**;
- **Mapear** uma entidade com as anotações:
  - @Entity e @Id;
  - @SequenceGenerator e @GeneratedValue;
  - @Table, @Column, @Transient, @Temporal;
  - @Lob e @Enumerated;

# CODAR!

---



2) Crie um programa para realizar procedimentos bancários de uma conta. Para tanto, crie as classes **Conta**, **ContaCorrente**, **ContaInvestimento** e **ContaPoupança**. Crie ainda um *enum* **TipoConta** e uma *exception* **SaldoInsuficienteException**. A saída deve ser fornecida numa classe **View**, contendo:

- Instanciar uma conta corrente;
- Instanciar uma conta poupança;
- Chamar o método retirar da conta corrente;
- Criar uma lista de conta corrente;
- Adicionar 3 contas;
- Exibir os saldos dessas 3 contas.

Utilize a anotação *@Override* caso necessário.



**Copyright © 2024 – 2034**  
**Prof. Dr. Marcel Stefan Wagner**

Todos direitos reservados. Reprodução ou divulgação total ou parcial deste documento é expressamente proibido sem o consentimento formal, por escrito, do Professor (autor).

Agradecimentos: Prof. Me Gustavo Torres Custódio | Prof. Me. Thiago T. I. Yamamoto

*“Se a vida não ficar mais fácil, trate de ficar mais forte.”*