FIAP GRADUAÇÃO

FIAP

ENTERPRISE APPLICATION DEVELOPMENT

Prof. Me. Thiago T. I. Yamamoto

#02 – JPA INTRODUÇÃO

 *profthiagoy@fiap.com.br*  *thiagoyama*

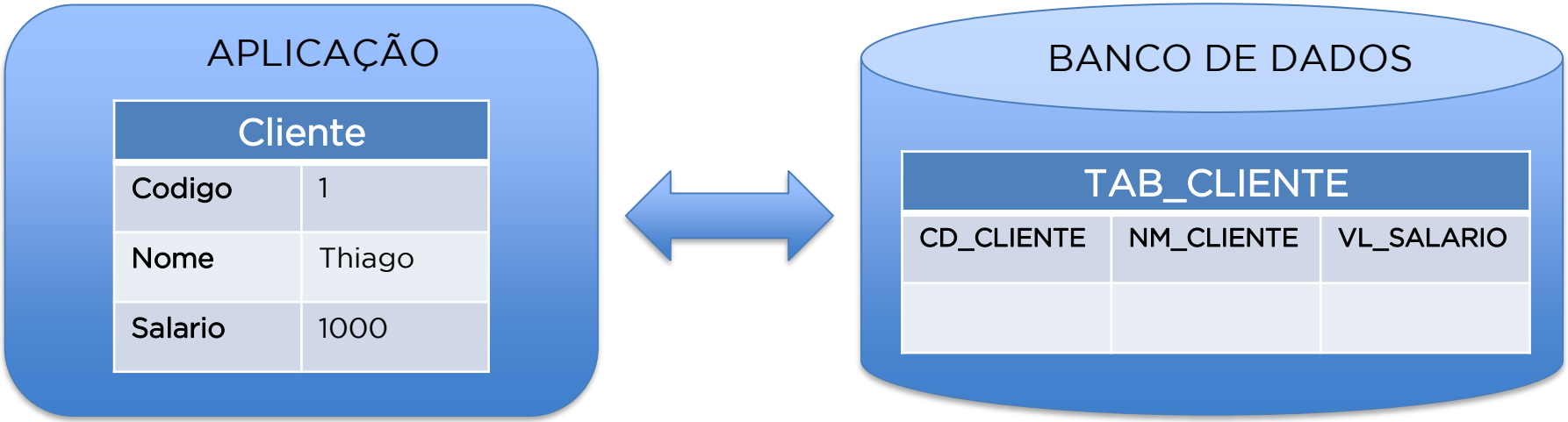
TRAJETÓRIA

- JPA Intodução

#02 - AGENDA

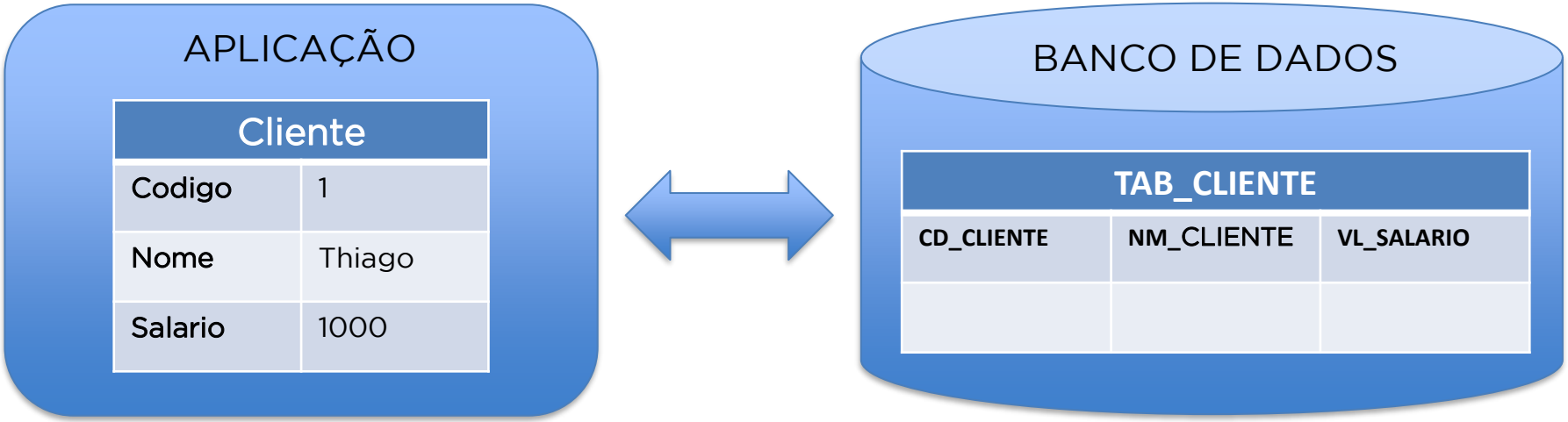
- Mapeamento Objeto-Relacional
- Anotações Java
- APIs de Persistência
- JPA – Java Persistence API e criação do Projeto
- Mapeamento de Entidades com anotações:
  - Entity e Id
  - Table
  - GeneratedValue e SequenceGenerator
  - Column
  - Transient
  - Temporal
  - Lob
  - Enumerated

PERSISTÊNCIA EM UMA APLICAÇÃO




```
String sql = "INSERT INTO TAB_CLIENTE (CD_CLIENTE, NM_CLIENTE, VL_SALARIO)
VALUES (?, ?, ?)";
PreparedStatement ps = conn.prepareStatement(sql);
ps.setInt(1, cliente.getCodigo());
ps.setString(2, cliente.getNome());
ps.setFloat(3, cliente.getSalario());
//...
```

MAPEAMENTO OBJETO-RELACIONAL



Mapeamento:
Cliente > TAB_CLIENTE
codigo > CD_CLIENTE
nome > NM_CLIENTE
salario > VL_SALARIO

- É possível **mapear a classe Cliente** para representar a tabela **TAB_CLIENTE** do banco de dados através de **anotações** ou arquivo xml;

ANOTAÇÕES JAVA

ANOTAÇÕES JAVA

FIAP

- São textos inseridos **diretamente no código fonte,** que **expressam informações complementares** sobre uma classe, seus métodos, atributos e etc..;
- Tais informações podem ser acessadas via **API Reflection,** por elementos fora do código fonte, por exemplo, **APIs** de **persistência**;
- Disponíveis no **Java 5** (JSR-175);
- Pode-se **criar** novas anotações a qualquer momento, sendo um processo bastante simples;
- Alternativa aos **descritores de deployment (XML);**

ANOTAÇÕES JAVA

- Objetos são instanciados a partir de **classes anotadas**;
- Processador **reconhece as anotações** encapsuladas nos objeto;
- Os **resultados** são produzidos a partir das informações contidas nas anotações.

ANOTAÇÕES JAVA - SINTAXE

- Podem ser inseridos antes da declaração de **pacotes**, **classes**, **interfaces**, **métodos** ou **atributos**;
- Iniciam com um **"@";**
- Uma anotação tem efeito sobre o **próximo elemento** na sequência de sua declaração;
- **Mais de uma anotação** pode ser aplicada a um mesmo elemento do código (classe, método, propriedade, etc...);
- Podem ter parâmetros na forma (param1="valor", param2="valor", ...);

```
@Override
@SuppressWarnings("all")
public String toString() {
   return "Thiago";
}
```

ANOTAÇÕES JAVA NATIVAS

Algumas anotações são **nativas**, isto é, já vem com o **JDK**:

- @Override - Indica que o método anotado sobrescreve um método da superclasse;
- @Deprecated - Indica que um método está obsoleto e não deve ser utilizado;
- @SuppressWarnings(tipoAlerta) - Desativa os alertas onde *tipoAlerta* pode ser “all”, “ cast ”, “null”, etc...;

ANOTAÇÕES JAVA - CRIAÇÃO

FIAP

- Utiliza a palavra **@interface;**
- **Métodos** definem os parâmetros aceitos pela anotação;
- Parâmetros possuem tipos de dados restritos (**String, Class, Enumeration, Annotation e Arrays desses tipos**);
- Parâmetros **podem** ter **valores default**;

Exemplo Anotação:

```
public @interface Mensagem {
  String texto() default "vazio";
}
```

Exemplo Utilização:

```
@Mensagem(texto="Alo Classe")
public class Teste {
  @Mensagem(texto="Alo Metodo")
  public void teste() { }
}
```

ANOTAÇÕES JAVA – META ANOTAÇÕES

Anotações para criar anotações:

- @Retention - Indica por quanto tempo a anotação será mantida:
  - **RetentionPolicy.SOURCE** - Nível código fonte;
  - **RetentionPolicy.CLASS** - Nível compilador;
  - **RetentionPolicy.RUNTIME** - Nível JVM;
- @Target - Indica o escopo da anotação:
  - **ElementType.PACKAGE** - Pacote;
  - **ElementType.TYPE** - Classe ou Interface;
  - **ElementType.CONSTRUCTOR** - Método Construtor;
  - **ElementType.FIELD** - Atributo;
  - **ElementType.METHOD** - Método;
  - **ElementType.PARAMETER** - Parâmetro de método;

ANOTAÇÕES JAVA - EXEMPLO

FIAP

Exemplo Anotação:

```
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.TYPE, ElementType.METHOD})
public @interface Mensagem {
    String texto() default "vazio";
}
```

Exemplo Utilização:

```
@Mensagem(texto="Alo Classe")
public class Teste {
    @Mensagem(texto="Alo Metodo")
    public void teste() { }
}
```

ACESSANDO AS ANOTAÇÕES – API REFLECTION

FIAP

- **API Reflection** é utilizado por vários **Frameworks** para recuperar informações de um objeto como **anotações, métodos** e **atributos**, em tempo de execução;

- Recuperar anotação de **classe**:

```
Mensagem m = obj.getClass().getAnnotation(Mensagem.class);
```

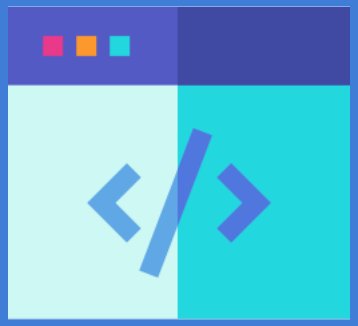
- Recuperar anotação de **método**:

```
Method[] metodos = obj.getClass().getDeclaredMethods();
for (int i = 0; i < metodos.length; i++)
        System.out.println(metodos[i].getAnnotation(Mensagem.class));
```

- Recuperar anotação de **propriedades**:

```
Field[] att = obj.getClass().getDeclaredFields();
for (int i = 0; i < att.length; i++)
        System.out.println(att[i].getAnnotation(Mensagem.class));
```

CODAR!



Escreva uma **classe** que tenha um **método** capaz de receber como parâmetro um objeto e gerar código SQL automaticamente capaz de **selecionar todos os registro de uma tabela**.

Crie uma anotação **@Tabela** que possua um parâmetro **nome** indicando o nome da tabela na qual a classe será mapeada.

Via **API Reflection** gerar automaticamente o código SQL necessário.

Exemplo:

```
@Tabela(nome="TAB_ALUNO")
public class Aluno { }
```

Irá gerar o **SQL** (impresso no console do eclipse):

```
SELECT * FROM TAB_ALUNO
```

APIs DE PERSISTÊNCIA

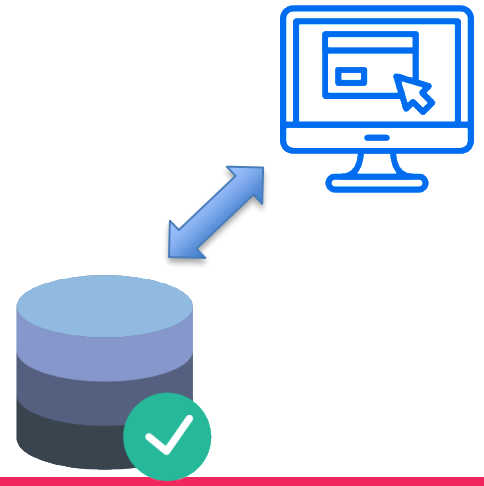
APIs DE PERSISTÊNCIA

FIAP

- 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 objeto em declarações SQL** (**INSERT, UPDATE, DELETE, SELECT, ...**).

FRAMEWORKS DE ORM

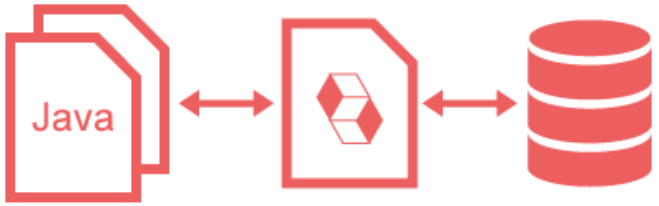
- **Object Relational Mapper (ORM) ou Mapeamento objeto relacional** são **frameworks** que permitem que **os objetos sejam mapeados** para o **modelo relacional** dos bancos de dados;
- O **ORM** possui **métodos básicos** que realizam a interação entre a **aplicação e o banco de dados**, se responsabilizando 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;
  - Phyton : Django, SQLAlchemyl;



JPA – JAVA PERSISTENCE API

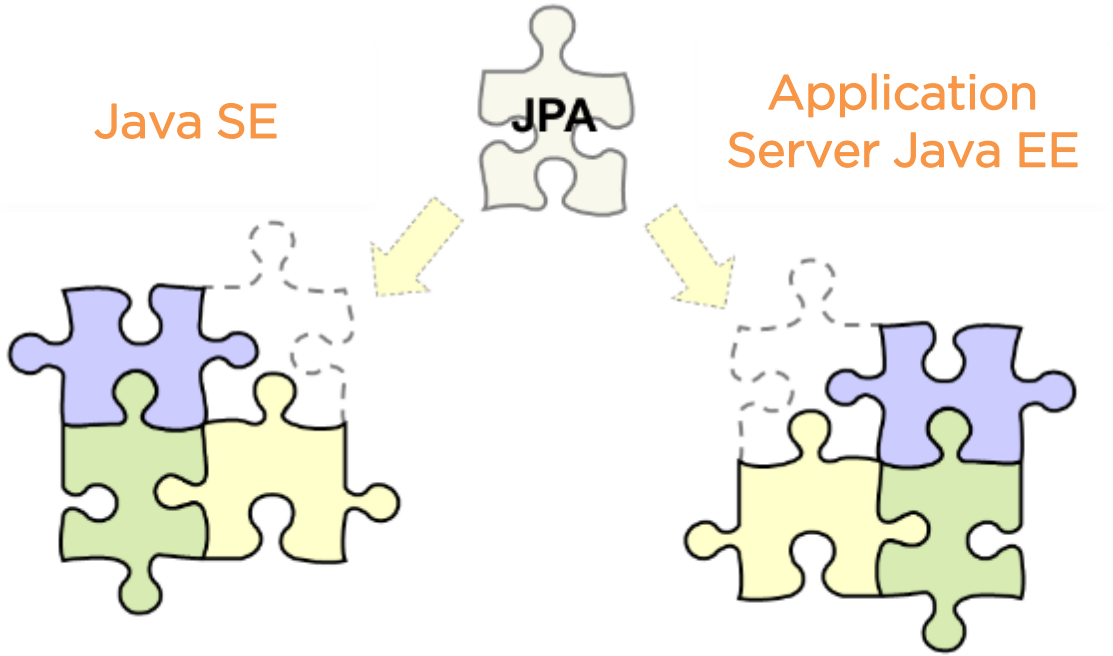
- **JPA** define uma interface comum para **persistência de dados** do **Java EE**;
- Oferece uma **especificação** padrão para **mapeamento objeto-relacional** para objetos Java simples, através de **anotações**;
- Pode ser utilizado de forma *standalone*, com **Java SE**;

- Exemplos de implementações:
  - Hibernate
    - http://www.hibernate.org/
  - Toplink
    - https://www.oracle.com/technetwork/middleware/toplink/overview/index.html



JPA – ONDE USAR?

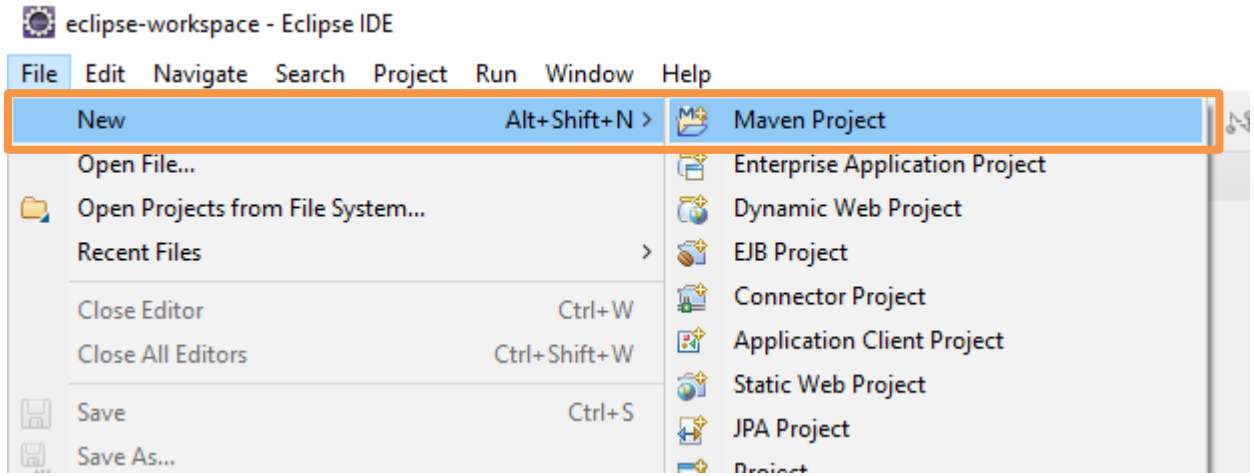
- É possível **utilizar o JPA** em todos os **projetos Java,** em conjunto com outros **frameworks Java**, do **Java EE** ou **não**;
- Dessa forma, o **JPA** se encaixa em **projetos web** com JSP, Servlets, JSF, Spring MVC.. É utilizado também em projetos Desktop, Console e etc..



CRIANDO UM PROJETO JAVA COM JPA

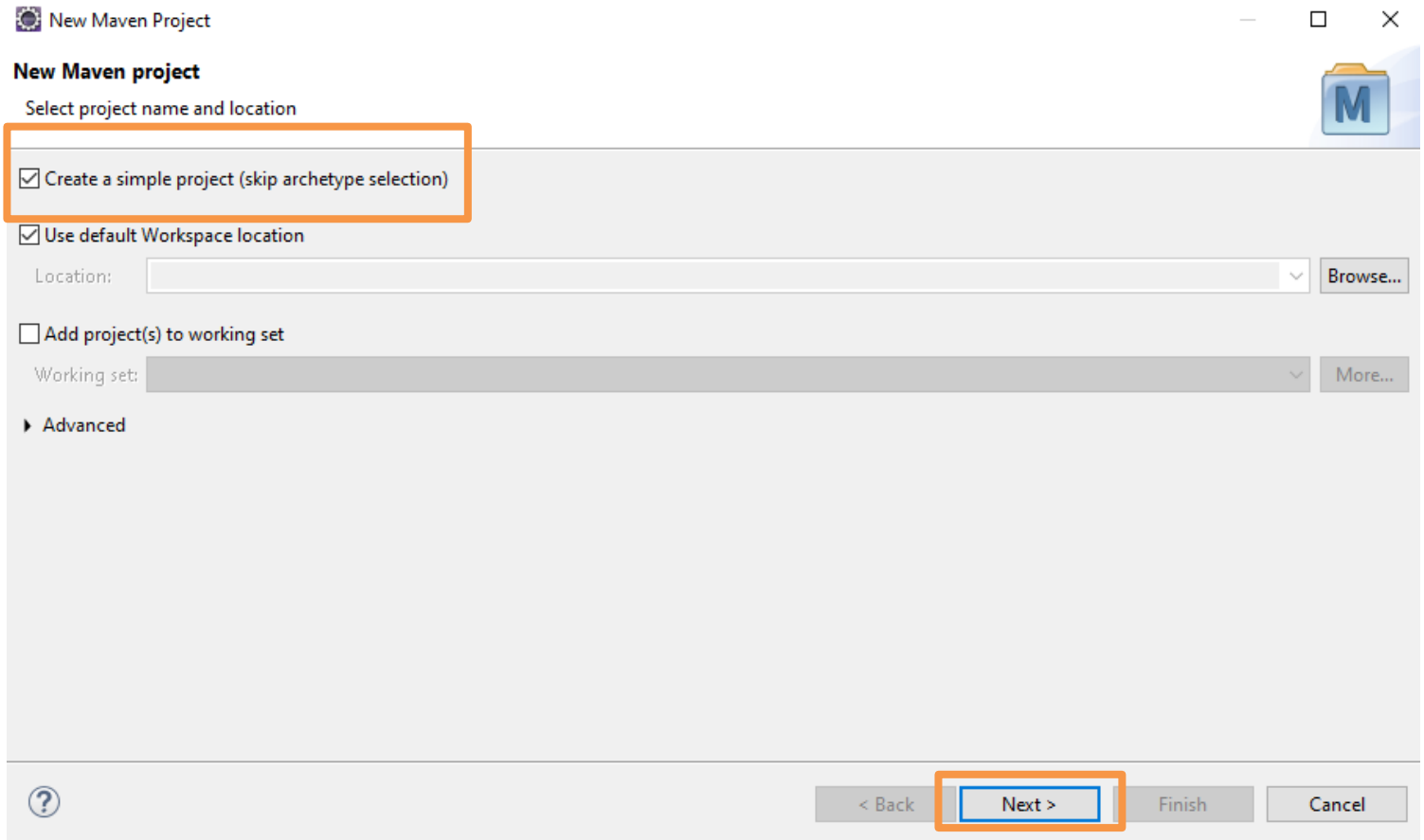
CRIANDO O PROJETO

- Crie um **Maven Project**;



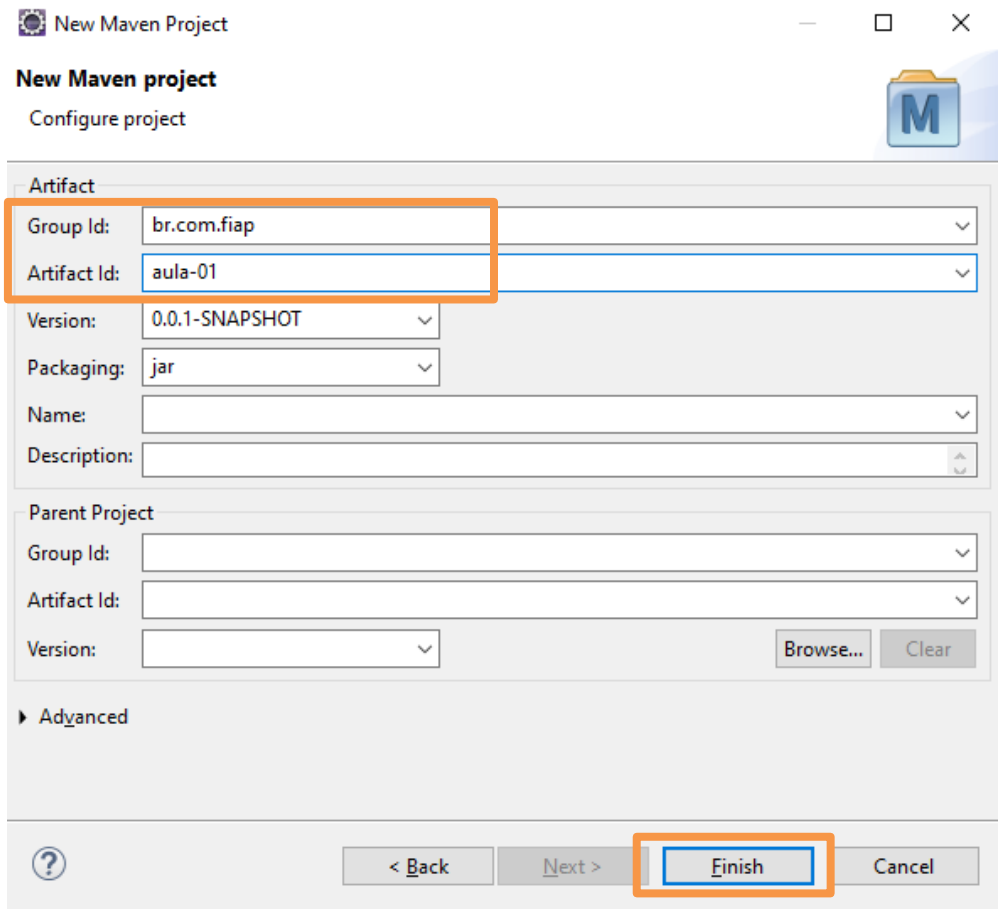
CRIANDO O PROJETO

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



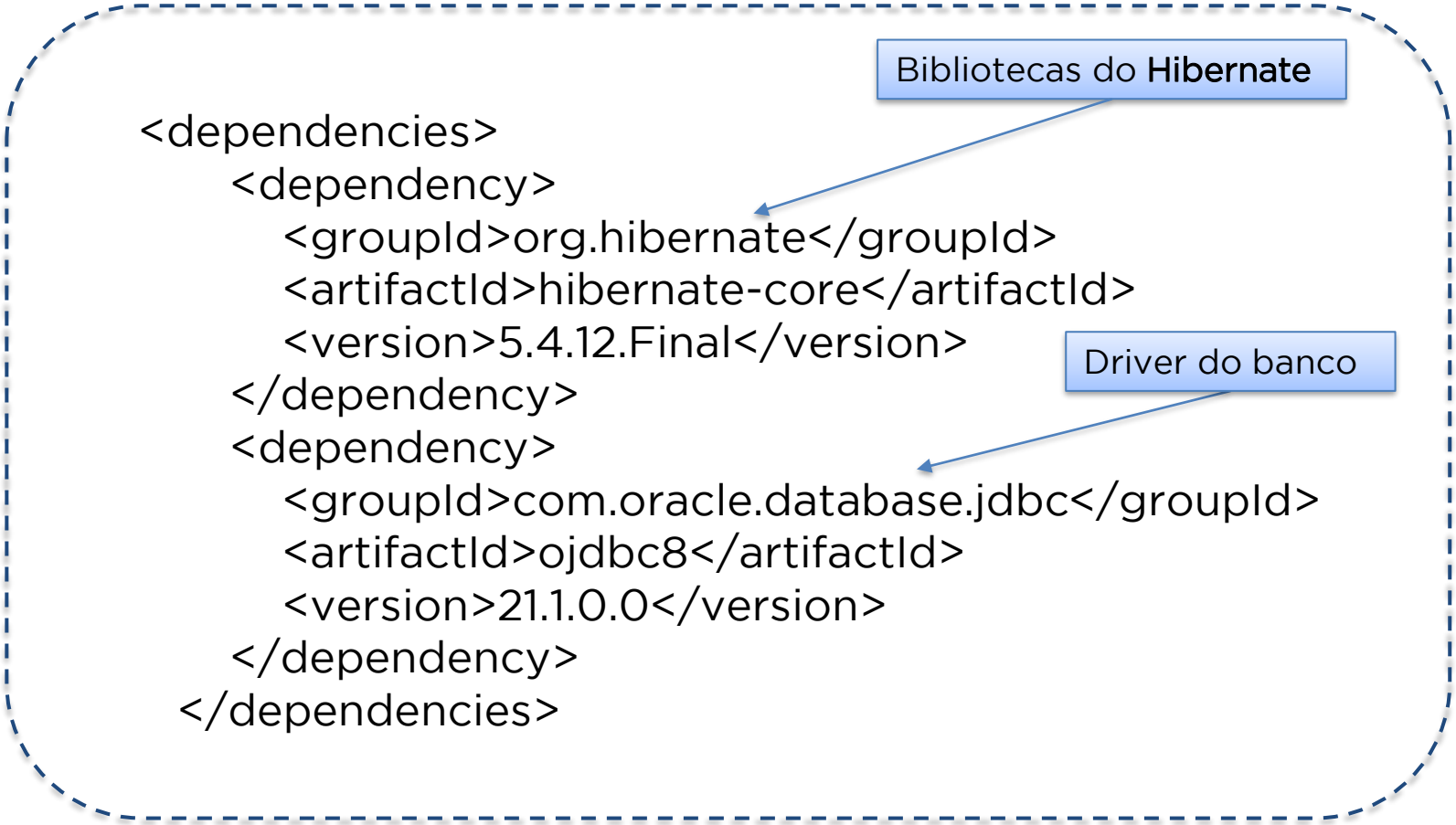
CRIANDO O PROJETO

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



POM.XML

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

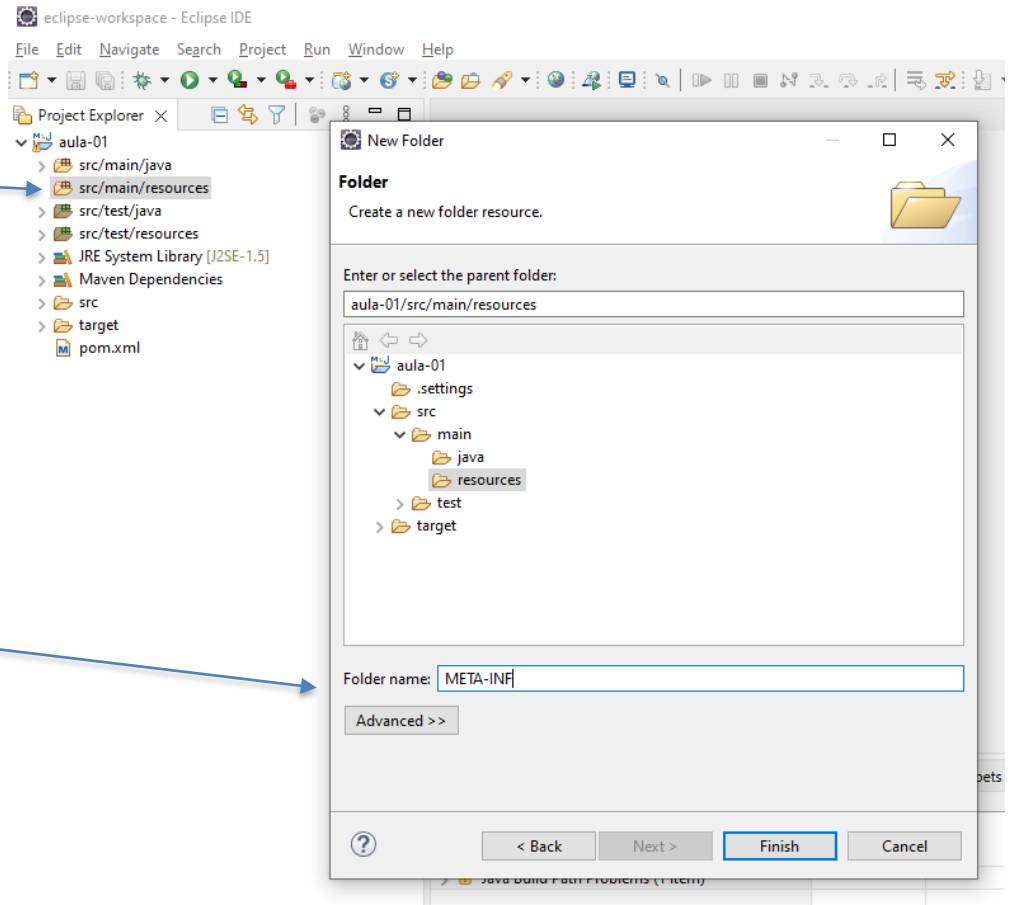


META-INF

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

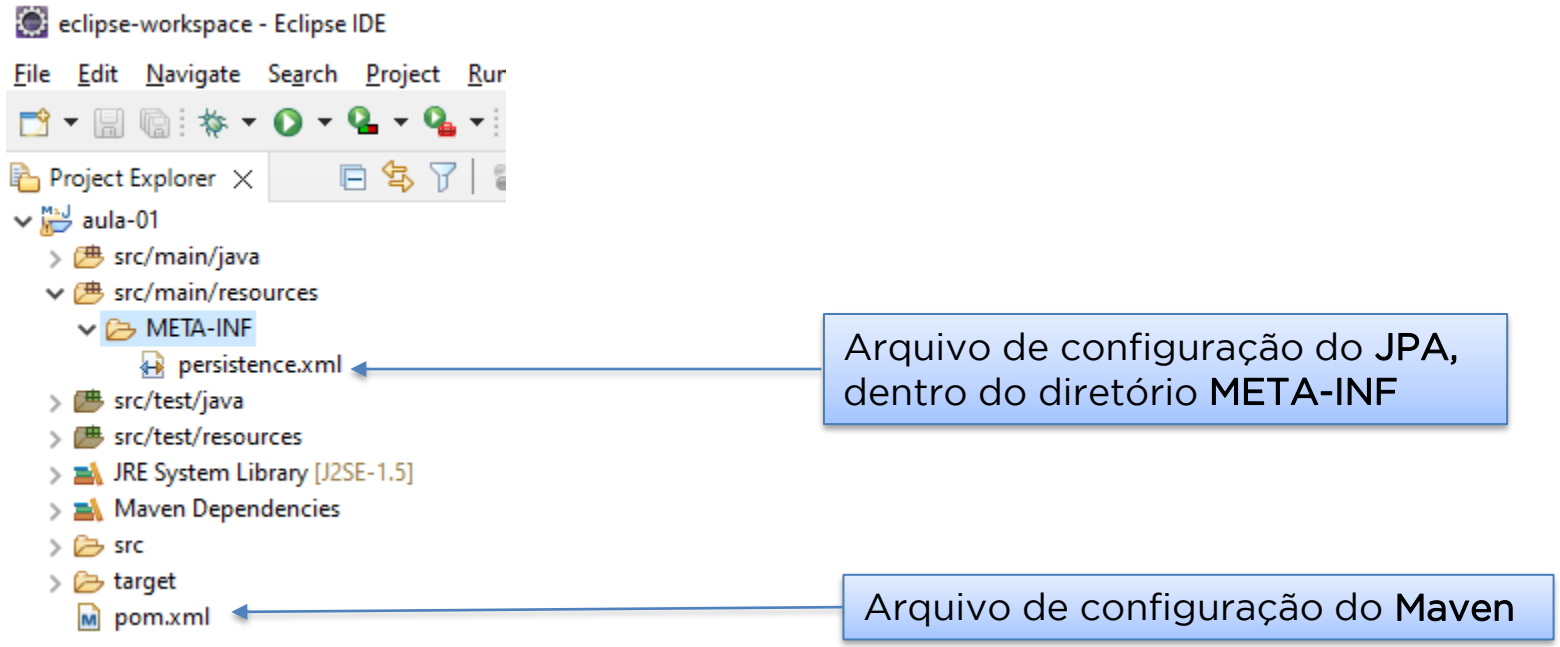
Clique com o botão direito do mouse na pasta e escolha “New” > “Other..” e procure por **Folder**

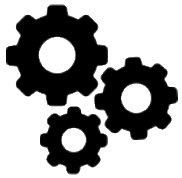
Defina o nome do diretório, deve ser exatamente **META-INF**



PERSISTENCE.XML

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





PERSISTENCE.XML

- 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

ENTIDADE

FIAP

- 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 da**dos, 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**;

@ENTITY

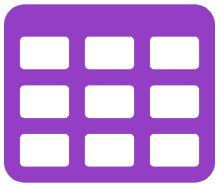
- As anotações da JPA situam-se no pacote **javax.persistence**;
- A anotação **@Entity** especifica que uma classe é uma entidade;
- O **nome da tabela** da entidade será o **mesmo da classe** com a anotação **@Entity.**

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

@TABLE

- 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
}
```



@ID

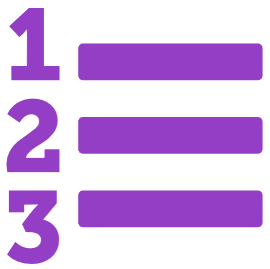
- Define o atributo que representa a **chave primária;**
- As únicas anotações obrigatórias para uma entidade são o **@Entity** e **@Id**;

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

@GENERATEDVALUE

FIAP

- 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:
  - **GeneratorType.SEQUENCE**: baseado em *sequence*;
  - **GeneratorType.IDENTITY**: campos identidade, auto *increment*;

@SEQUENCEGENERATOR

- 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;

MAPEAMENTO SEQUENCE - EXEMPLO

```
@Entity
@SequeceGenerator(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
}
```

@COLUMN

- 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;

@COLUMN

```
@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
}
```

@TRANSIENT

- Indica que o atributo **não** será um **campo na tabela**;

```
@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;
}
```

@TEMPORAL

- 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;
}
```

@LOB

- Permite **mapear** objetos de **grande dimensão** 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;
}
```

ENUM

- Propriedades que possuem **valores fixos**, por exemplo, gênero (MASCULINO, FEMININO), dia da semana (SEGUNDA, TERÇA, ...)
- 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

@ENUMERATED

- 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;

FIAP

"Aquele que não luta pelo futuro que quer, deve aceitar o futuro que vier"