

PROGRAMAÇÃO PARA WEB I

JPA

Profa. Silvia Bertagnolli

O QUE É ORM?

Mapeamento Objeto Relacional (ORM – Object-Relational Mapping) consiste em converter objetos para o modelo relacional

Modelo Orientado a Objetos	Modelo Relacional
Classe	Tabela
Objeto	Linha ou tupla
Atributo	Coluna
Método	Não possui equivalente
Associação/composição	Chave estrangeira

O QUE É ORM?

Com ORM é possível abstrair o modelo de classes e os códigos SQL/JDBC

A maioria dos comandos SQL são gerados pelos metadados, logo usa menos linhas de código

A migração entre bancos é mais fácil e rápida

JPA

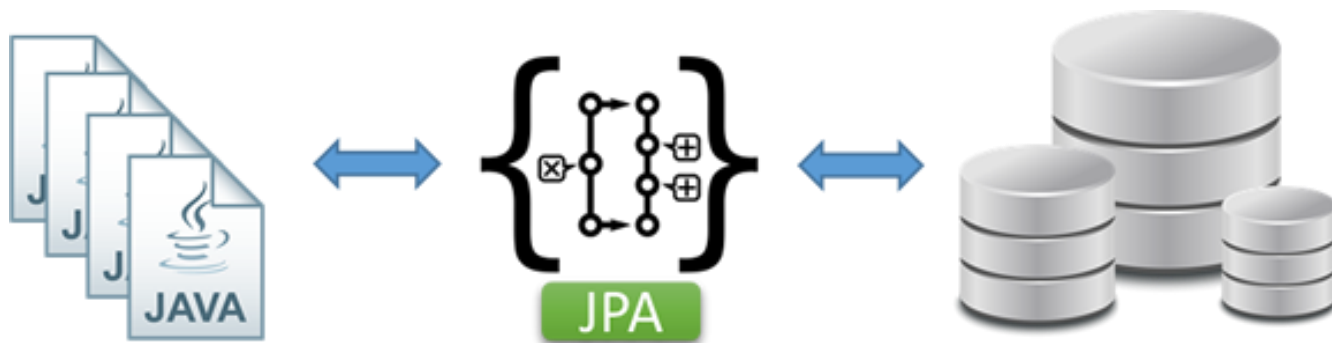
A JPA (Java Persistence API) ou Jakarta Persistence é uma especificação que fornece uma API (conjunto de classes e interfaces) para trabalhar com ORM

Existem implementações da especificação, por exemplo:
Hibernate e EclipseLink*

Anotações são o recurso básico para usar essa especificação

A Persistência de dados é baseada em classes chamadas entidades - antigos POJOs

ARQUITETURA COM JPA



Fonte = <http://www.devmedia.com.br/guia/guia-de-referencia-jpa-java-persistence-api/38173>

NÃO ESQUECER:

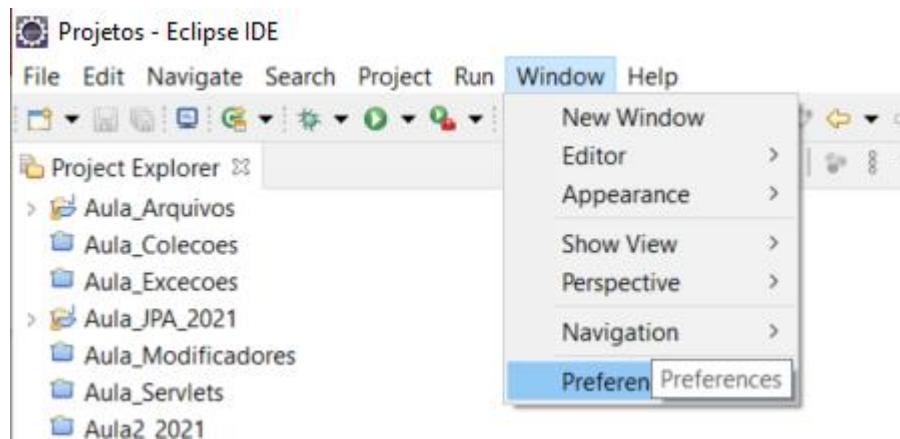
**Para conectar com
um banco de dados
é necessário
criá-lo antes de
conectar o código
Java**

ELEMENTOS DO PROJETO

PASSOS

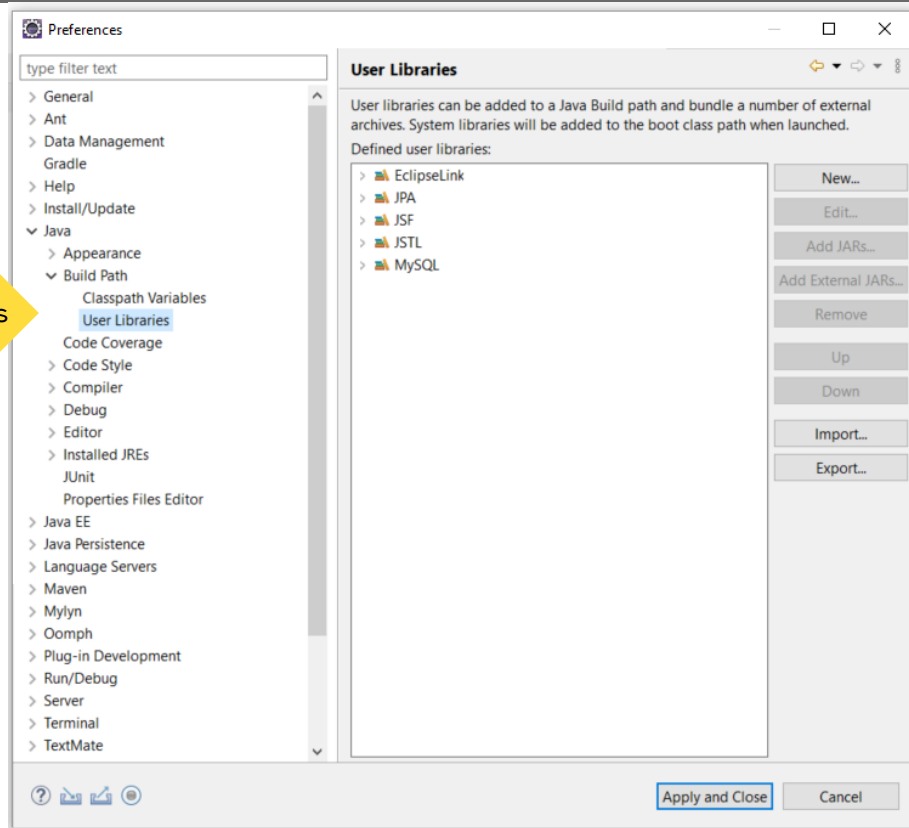
- 1º - Criando User Library
- 2º - Criando o projeto
- 3º - Configurando o ambiente
- 4º - Criando a entidade e registrar no arquivo de persistência
- 5º - Usar as entidades

1º - CRIANDO USER LIBRARY

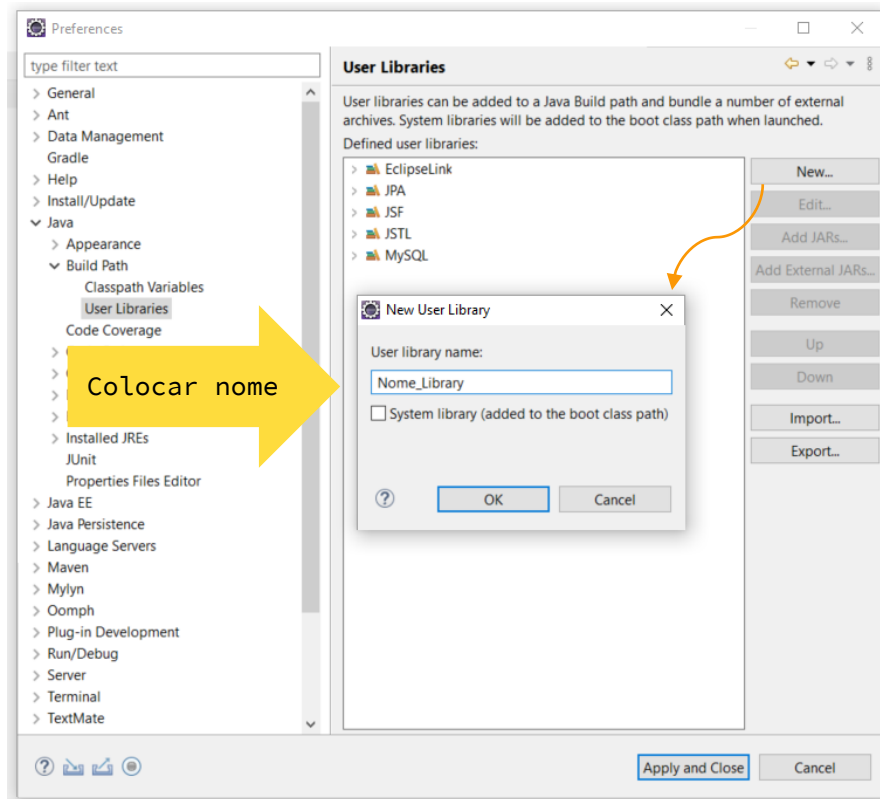


1º - CRIANDO USER LIBRARY

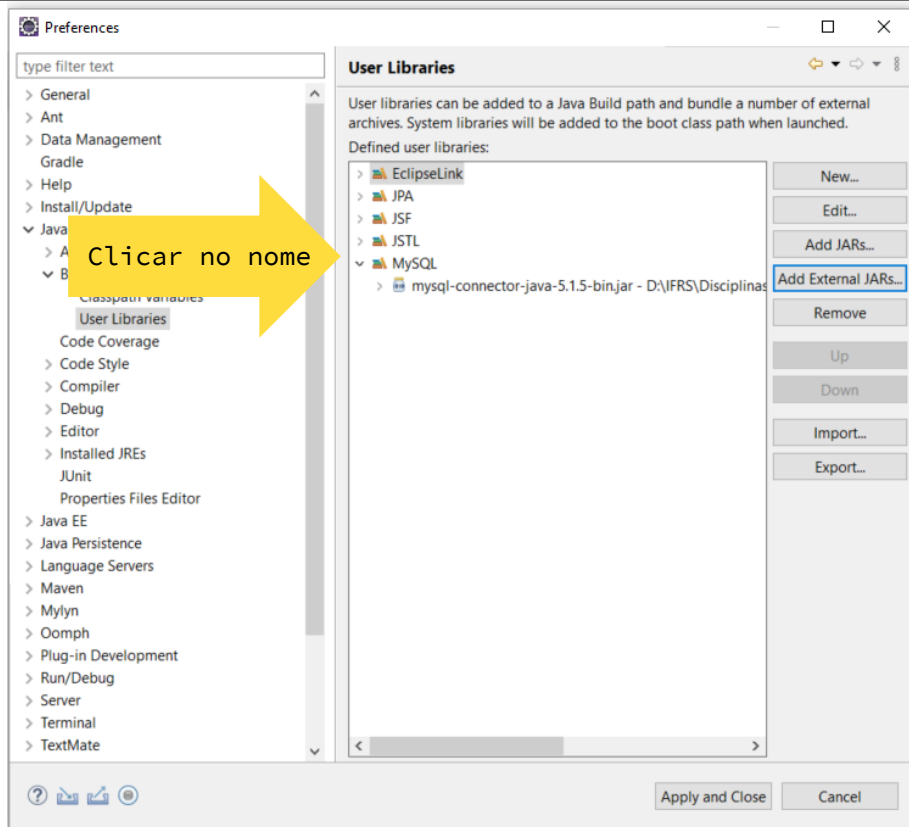
User Libraries



1º - CRIANDO USER LIBRARY



1º - CRIANDO USER LIBRARY



Clicar no nome

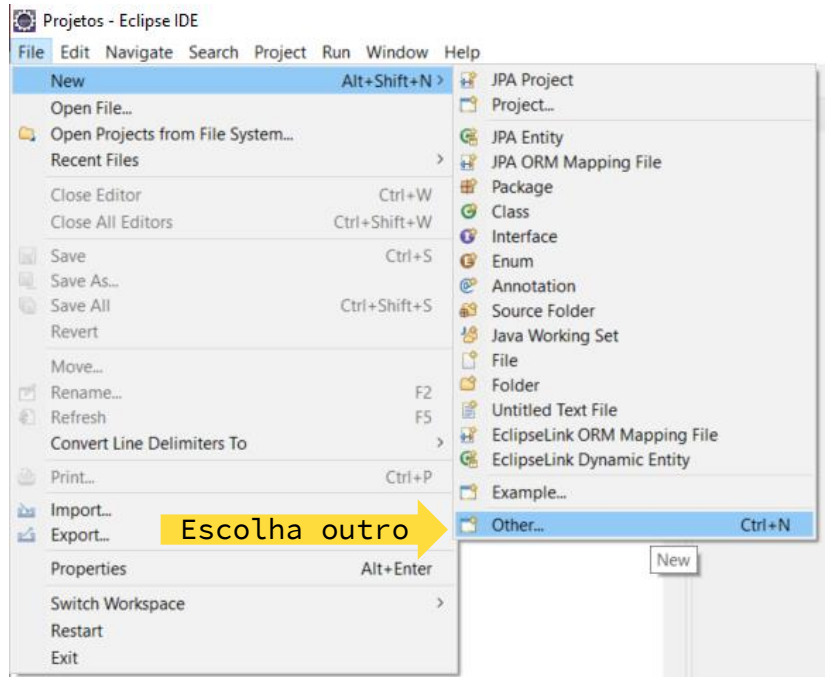
Escolher o arquivo
.jar e vincular

Criar 2 constantes
para bibliotecas:

1) EclipseLink
adicionar
eclipseLink.jar e
javax.persistence_2.1.
0.v201304241213.jar

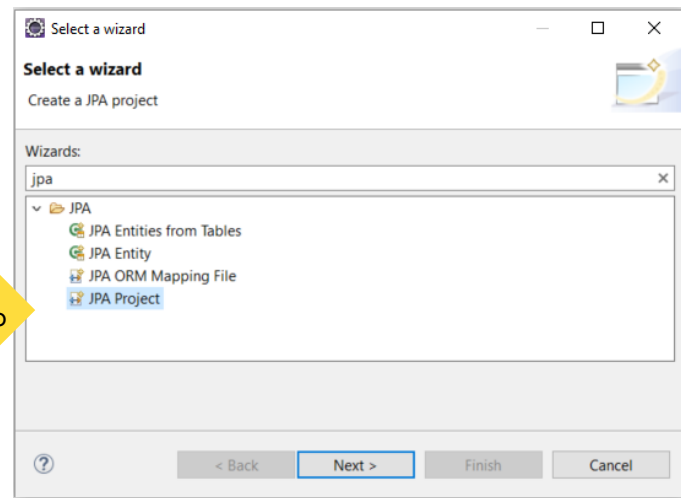
2) MySQL adicionar
mysql-connector-java-
5.1.5-bin.jar

2º CRIANDO O PROJETO



Informe jpa


Escolha esse tipo de projeto



Nome do proj.

Manter demais
configurações

Vá no Next

 **New JPA Project** — □ ×

JPA Project
Configure JPA project settings.

Project name:

Project location
☒ Use default location
Location:

Target runtime


JPA version

Configuration

A general starting point for a JPA application.

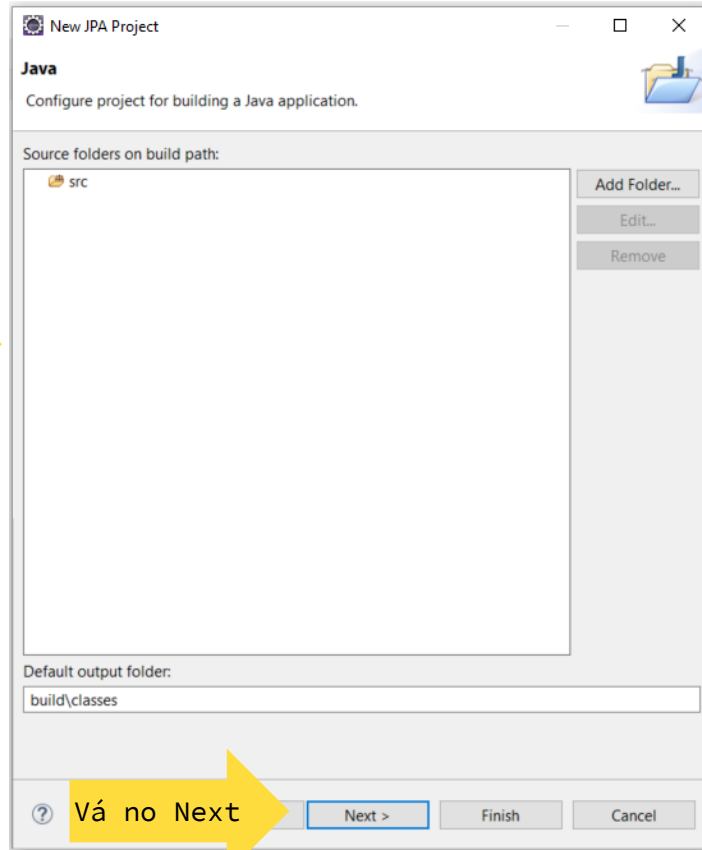
EAR membership
☐ Add project to an EAR
EAR project name:

Working sets
☐ Add project to working sets
Working sets:



2º CRIANDO O PROJETO

Manter demais
configurações



Informar User Library:
EclipseLink e MySQL

Aqui dá pra criar uma
conexão com a sua base

New JPA Project

JPA Facet

Configure JPA settings.

Platform
Generic 2.2

JPA implementation
Type: User Library

☒ EclipseLink
☐ JPA
☐ JSF

☐ Include libraries with this application

Connection
<None>

[Add connection...](#)
[Connect](#)

☐ Add driver library to build path

Driver:

☐ Override default catalog from connection

Catalog:

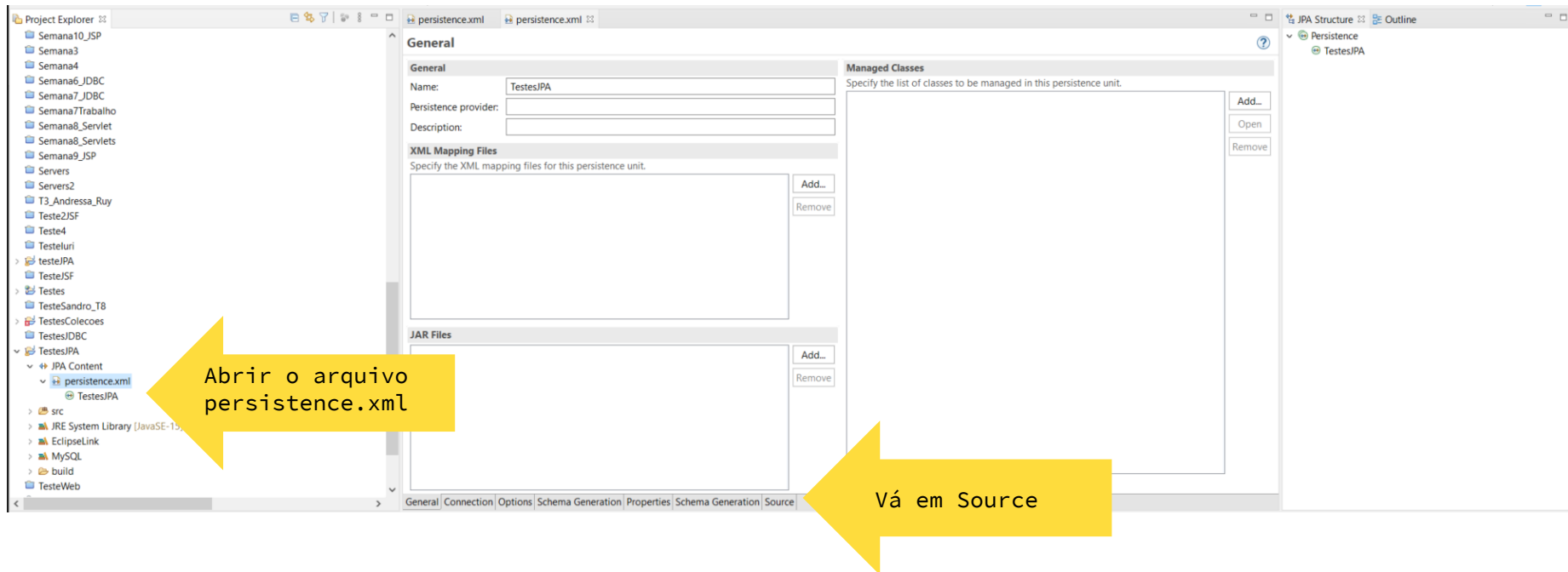
☐ Override default schema from connection

Schema:

☐ Persistent class management

? < Back Finalizar Finish Cancel

3º CONFIGURANDO O AMBIENTE



3º CONFIGURANDO O AMBIENTE

O persistence.xml é um arquivo de configuração padrão da JPA

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.2" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd">
  <persistence-unit name="testeJPA" transaction-type="RESOURCE_LOCAL">
    <class>classes.Usuario</class>
    <properties>
      <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/bd"/>
      <property name="javax.persistence.jdbc.user" value="root"/>
      <property name="javax.persistence.jdbc.password" value=""/>
      <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>
      <!-- <property name="javax.persistence.schema-generation.database.action" value="drop-and-create"/> -->
    </properties>
  </persistence-unit>
</persistence>
```



Adicionar esse conteúdo no arquivo
Copiar o conteúdo do arquivo persistence.xml do Moodle

3º CONFIGURANDO O AMBIENTE

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.2" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd">
  <persistence-unit name="testeJPA" transaction-type="RESOURCE_LOCAL">
    <class>classes.Usuario</class>
    <properties>
      <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/bd"/>
      <property name="javax.persistence.jdbc.user" value="root"/>
      <property name="javax.persistence.jdbc.password" value=""/>
      <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>
      <!-- <property name="javax.persistence.schema-generation.database.action" value="drop-and-create"/> -->
    </properties>
  </persistence-unit>
</persistence>
```

Unidade de persistência

Ao criar a unidade de persistência definimos:

- As propriedades da conexão
- As entidades que farão parte do sistema

3º CONFIGURANDO O AMBIENTE

- `javax.persistence.jdbc.url`: descrição da URL de conexão com o banco de dados
- `javax.persistence.jdbc.user`: nome do usuário do banco de dados
- `javax.persistence.jdbc.password`: senha do usuário do banco de dados
- `javax.persistence.jdbc.driver`: nome completo da classe do driver JDBC

3º CONFIGURANDO O AMBIENTE

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.2" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd">
  <persistence-unit name="testeJPA" transaction="JTA">
    <class>classes.Usuario</class>
    <properties>
      <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/bd"/>
      <property name="javax.persistence.jdbc.user" value="root"/>
      <property name="javax.persistence.jdbc.password" value=""/>
      <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>
      <!-- <property name="javax.persistence.schema-generation.database.action" value="drop-and-create"/> -->
    </properties>
  </persistence-unit>
</persistence>
```

Incluir classes que serão persistidas

ESTRATÉGIAS PARA GERAR TABELAS

`javax.persistence.schema-generation.database.action:`

- `create` - criar as tabelas para as entidades quando implantadas em um servidor
- `drop-and-create` - apagar as tabelas existentes e recriar quando a aplicação for implantada
- `nenhum` - não criar nada

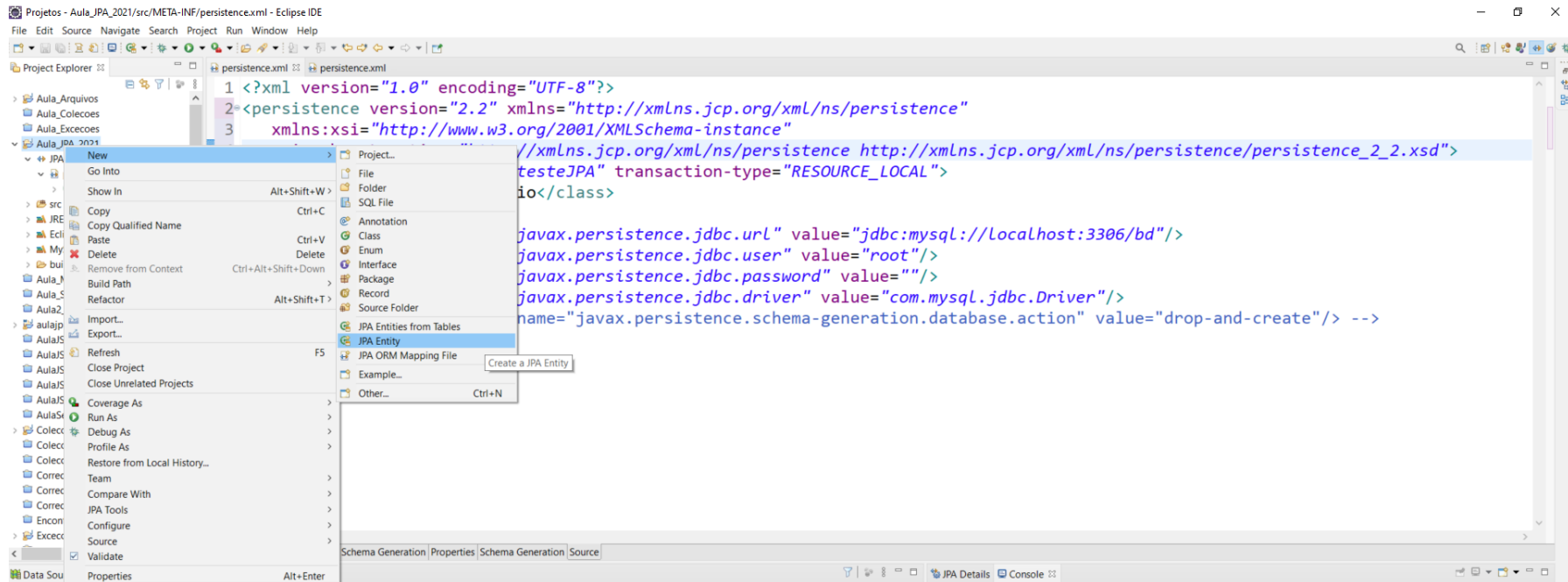
HABILITANDO GERAÇÃO AUTOMÁTICA

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.2" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd">
<persistence-unit name="testeJPA" transaction-type="RESOURCE_LOCAL">
<class>classes.Usuario</class>
<class>enums.Usuario2</class>
<class>datas.Usuario3</class>
<class>colecoes.Usuario4</class>
<properties>
<property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/testejpa2022"/>
<property name="javax.persistence.jdbc.user" value="root"/>
<property name="javax.persistence.jdbc.password" value=""/>
<property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>
<property name="javax.persistence.schema-generation.database.action" value="drop-and-create"/>
</properties>
</persistence-unit>
</persistence>
```

NÃO ESQUECER:

**Todas as classes
que devem ser
persistidas devem
constar na lista
de classes de
Entidade**

4º - CRIANDO A ENTIDADE



4º - CRIANDO A ENTIDADE

New JPA Entity

Entity class
Create a new JPA entity. Only JPA enabled projects may be selected.

Project: TestesJPA
Source folder: /Testes/JPA/src Browse...
Java package: entidades Browse...
Class name: Usuario
Superclass: Browse...

Inheritance
☒ Entity
☐ Mapped superclass
☐ Inheritance:
XML entity mappings
☐ Add to entity mappings in XML
Mapping file: META-INF/orm.xml Browse...

Next > Finish Cancel

Vá no Next

New JPA Entity

Entity Properties
Set entity name, table name, fields, and access type.

Entity name: Usuario
Table name
☒ Use default
Table name: Usuario

Entity fields

Key	Name	Type

Add... Edit... Remove

Access type
☒ Field
☐ Property

Finish Cancel

Concluir

4º - CRIANDO A ENTIDADE (CÓDIGO GERADO)

```
package entidades;

import java.io.Serializable;

/**
 * Entity implementation class for Entity: Usuario
 *
 */
@Entity
public class Usuario implements Serializable {

    private static final long serialVersionUID = 1L;

    public Usuario() {
        super();
    }

}
```

ENTIDADES

CRIANDO UMA ENTIDADE JPA

Uma entidade JPA é uma classe Java que recebe anotações e que possui seus campos persistidos em um banco de dados pela API JPA

Geralmente as entidades JPA são classes POJOs e, portanto, não necessitam estender nenhuma classe ou implementar nenhuma interface específica

CÓDIGO GERADO

```
package entidades;

import java.io.Serializable;

/**
 * Entity implementation class for Entity: Usuario
 *
 */
@Entity
public class Usuario implements Serializable {

    private static final long serialVersionUID = 1L;

    public Usuario() {
        super();
    }

}
```

ANOTAÇÕES

Para definir uma classe como entidade, basta adicionar a anotação **@Entity** antes da declaração do nome da classe

Para definir a chave primária devemos usar a anotação **@Id**

Observações:

Toda entidade **deve** ter um construtor sem parâmetros

Toda entidade **deve** ter uma chave primária, que identifica de forma única cada uma das suas instâncias, por isso o código anterior está com erro

ENTIDADE USUÁRIO

```
package classes;  
import java.io.Serializable;  
import javax.persistence.*;
```

@Entity

```
public class Usuario implements Serializable {  
    private static final long serialVersionUID = 1L;
```

@Id

```
    private Long id;  
    public Long getId() { return id; }  
    public void setId(Long id) { this.id = id; }  
    //outros métodos  
}
```


DECLARANDO ATRIBUTOS E CRIANDO COLUNAS

Note que a classe usuário só tem um atributo

Agora, vamos complementar a classe usuário como pede abaixo, sabendo que o nome da tabela é user:

Atributo	Coluna	Descrição
idUsuario	idUsuario	Chave primária e deve ser gerada (auto incremento)
Identificador	identificador	String, tamanho 20, não pode ser nulo,
senha	senha	String, tamanho 10, não pode ser nulo

@TABLE

Por padrão, os nomes das tabelas são criadas com letra maiúscula

Se você desejar trocar o nome para iniciar com letra minúscula, por exemplo, deve usar @Table

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

CHAVE PRIMÁRIA

```
//...
```

```
@Entity
```

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

```
public class Usuario implements Serializable {
```

```
    private static final long serialVersionUID = 1L;
```

```
    @Id
```

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

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private Long id;
```

CHAVE PRIMÁRIA: ESTRATÉGIAS DE GERAÇÃO DA CHAVE

GenerationType.AUTO: o JPA escolhe a a melhor estratégia

GenerationType.IDENTITY: força que o JPA utilize colunas com valores auto incrementáveis; usada somente com SGBDs que possuem suporte a autoincremento (ideal para o MySQL)

GenerationType.SEQUENCE: indica que deve ser usada uma *sequence* do banco de dados para a geração da chave primária da entidade – Oracle utiliza essa abordagem (ideal para o Oracle)

ANOTAÇÃO @COLUMN

name – Nome da coluna na tabela – valor é uma String

columnDefinition – Recebe uma String com o tipo que será usado pela coluna equivalente na tabela do banco de dados

length – usado para tipos de colunas que possuem um valor variável de largura do campo, por exemplo, varchar

No MySQL se nenhum valor for definido o campo é criado em seu tamanho máximo de caracteres, que no caso é 255

Cada banco de dados possui um limite máximo para cada tipo de dado

ANOTAÇÃO @COLUMN

nullable – Recebe um valor booleano cujo o padrão é true, caso não declarado. Se false, este campo é obrigatório (equivale à restrição not null)

A anotação @Id possui uma anotação dependente, chamada de @javax.persistence.GeneratedValue. Esta anotação é utilizada para gerar as chaves primárias

A anotação @GeneratedValue possui um atributo chamado strategy, que define a estratégia de geração de valores incrementados

ANOTAÇÃO @COLUMN - EXEMPLOS

```
//...
```

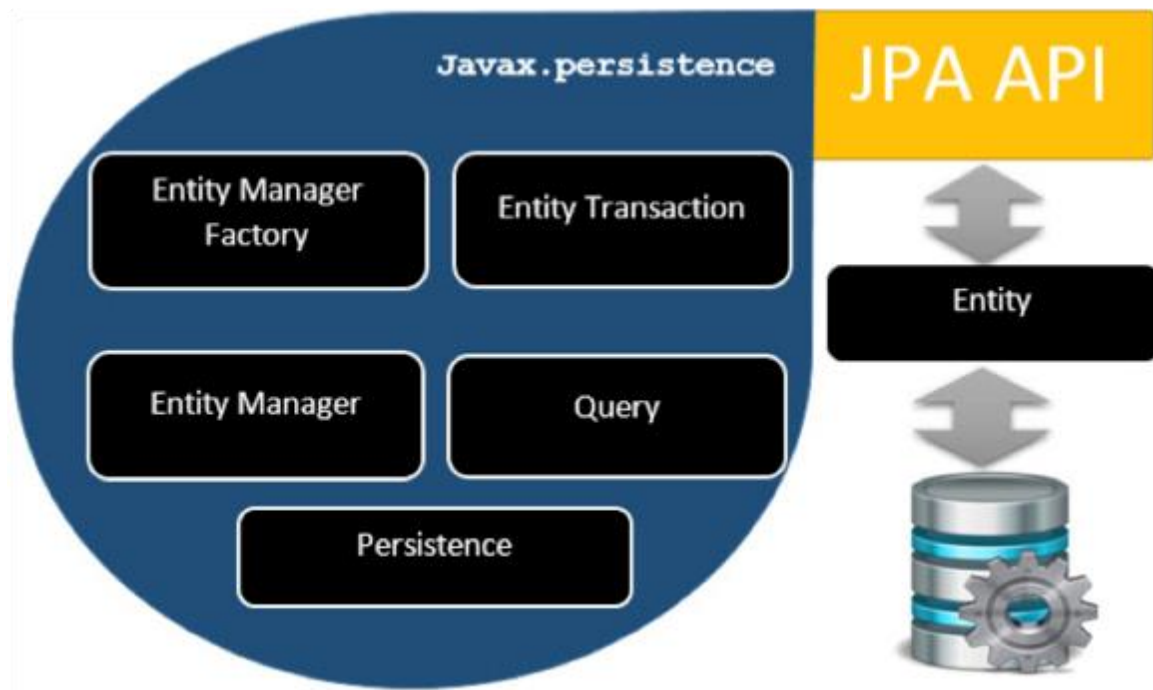
```
@Column(name="identifier", nullable=false,  
         columnDefinition = "text", length = 20)  
private String identificador;
```

```
@Column(name="password", nullable=false,  
         columnDefinition = "text", length = 10)  
private String senha;
```

NÃO ESQUECER:

Você só precisa
definir os
detalhes físicos
se gerar as
tabelas com o
recurso de schema
generation do JPA

API JPA



RESUMO DA API

Classe/Interface	Descrição
EntityManagerFactory	Esta é uma classe de fábrica de EntityManager. Ele cria e gerencia várias instâncias EntityManager
EntityManager	É uma interface que gerencia as operações de persistência de objetos
Entity	Entidades são os objetos de persistência
EntityTransaction	Para cada EntityManager, as operações são mantidas pela classe EntityTransaction
Persistence	Esta classe contém métodos estáticos para obter instâncias de EntityManagerFactory
Query	Essa interface é implementado por cada fornecedor JPA para obter objetos relacionais que atendem aos critérios

EXEMPLO 1

```
import javax.persistence.*;
import classes.Usuario;

public class TesteJPA_0 {
    public static void main(String[] args) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("testeJPA");
        EntityManager em = emf.createEntityManager();
        Usuario user = new Usuario("sbertagnolli2", "123456");
        em.getTransaction().begin();
        em.persist(user);
        System.out.println("Usuário salvo com sucesso! ");
        em.getTransaction().commit();
        em.close();
        emf.close();
    }
}
```




Fábrica dos gerenciadores de entidade (Entity Manager)

EXEMPLO 1

```
import javax.persistence.*;
import classes.Usuario;

public class TesteJPA_0 {
    public static void main(String[] args) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("testeJPA");
        EntityManager em = emf.createEntityManager();
        Usuario user = new Usuario("sbertagnolli2", "123456");
        em.getTransaction().begin();
        em.persist(user);
        System.out.println("Usuário salvo com sucesso! " );
        em.getTransaction().commit();
        em.close();
        emf.close();
    }
}
```




Nome da unidade de persistência deve ser o mesmo definido no arquivo persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.2" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd">
  <persistence-unit name="testeJPA" transaction-type="RESOURCE_LOCAL">
    <class>classes.Usuario</class>
    <properties>
      <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/bd"/>
    </properties>
  </persistence-unit>
</persistence>
```

EXEMPLO 1

```
import javax.persistence.*;
import classes.Usuario;

public class TesteJPA_0 {
    public static void main(String[] args) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("testeJPA");
        EntityManager em = emf.createEntityManager();
        Usuario user = new Usuario("sberta colli2", "123456");
        em.getTransaction().begin();
        em.persist(user);
        System.out.println("Usuário salvo com sucesso! ");
        em.getTransaction().commit();
        em.close();
        emf.close();
    }
}
```




Entity Manager gerencia as entidades -
possibilita usar os métodos de
persistir, pesquisar e excluir objetos
do banco de dados

EXEMPLO 1

```
import javax.persistence.*;
import classes.Usuario;

public class TesteJPA_0 {
    public static void main(String[] args) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("testeJPA");
        EntityManager em = emf.createEntityManager();
        Usuario user = new Usuario("sbertagnolli2", "123456");
        em.getTransaction().begin();
        em.persist(user);
        System.out.println("Usuário salvo com sucesso! ");
        em.getTransaction().commit();
        em.close();
        emf.close();
    }
}
```

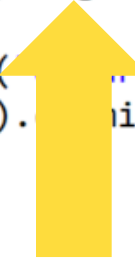


Cria objeto que será armazenado no BD

EXEMPLO 1

```
import javax.persistence.*;
import classes.Usuario;

public class TesteJPA_0 {
    public static void main(String[] args) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("testeJPA");
        EntityManager em = emf.createEntityManager();
        Usuario user = new Usuario("sbertagnolli2", "123456");
        em.getTransaction().begin();
        em.persist(user);
        System.out.println("Usuario salvo com sucesso! ");
        em.getTransaction().commit();
        em.close();
        emf.close();
    }
}
```



Inicia uma nova transação

EXEMPLO 1

```
import javax.persistence.*;
import classes.Usuario;


public class TesteJPA_0 {
    public static void main(String[] args) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("testeJPA");
        EntityManager em = emf.createEntityManager();
        Usuario user = new Usuario("sbertagnolli2", "123456");
        em.getTransaction().begin();
        em.persist(user);
        System.out.println("Usuário salvo com sucesso! " );
        em.getTransaction().commit();
        em.close();
        emf.close();
    }
}
```

Salva (persiste) o objeto
no banco - sem usar SQL

EXEMPLO 1

```
import javax.persistence.*;
import classes.Usuario;

public class TesteJPA_0 {
    public static void main(String[] args) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("testeJPA");
        EntityManager em = emf.createEntityManager();
        Usuario user = new Usuario("sbertagnolli2", "123456");
        em.getTransaction().begin();
        em.persist(user);
        System.out.println("Usuário salvo com sucesso! ");
        em.getTransaction().commit();
        em.close();
        emf.close();
    }
}
```



Conclui a transação dando commit no BD
Isso efetiva a inserção do objeto no BD

EXEMPLO 1

```
import javax.persistence.*;
import classes.Usuario;

public class TesteJPA_0 {
    public static void main(String[] args) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("testeJPA");
        EntityManager em = emf.createEntityManager();
        Usuario user = new Usuario("sbertagnolli2", "123456");
        em.getTransaction().begin();
        em.persist(user);
        System.out.println("Usuário salvo com sucesso! ");
        em.getTransaction().commit();
        em.close();
        emf.close();
    }
}
```



Fecha o EntityManagerFactory e o
EntityManager

JPAUTIL

JPAUTIL

A classe JPAUtil é criada usando o padrão de projeto Singleton

Essa classe permite que através da fábrica EntityManagerFactory seja possível obter uma nova instância de EntityManager caso não tenha exista nenhuma aberta, ou retornará a que já está aberta

PADRÃO DE PROJETO: SINGLETON

Singleton
<u>- singleton : Singleton</u>
<u>- Singleton()</u> <u>+ getInstance() : Singleton</u>

JPAUTIL

```
import javax.persistence.*;

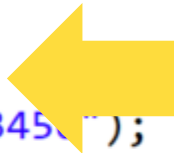
public class JPAUtil {
    private static EntityManagerFactory emf;
    public static EntityManager getEntityManager() {
        if(emf == null)
            emf = Persistence.createEntityManagerFactory("testeJPA");
        return emf.createEntityManager();
    }
    public void fechaEntityManager(){
        emf.close();
    }
}
```



Retorna um EntityManager
vinculado com a unidade de
persistência testeJPA

EXEMPLO 2

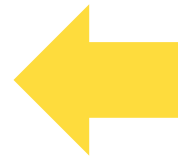
```
public class Exemplo2 {  
  
    public static void main(String[] args) {  
        EntityManager em = JPAUtil.getEntityManager();  
        Usuario user = new Usuario("sbertagnolli2", "123456789");  
        em.getTransaction().begin();  
        em.persist(user);  
        System.out.println("Usuário salvo com sucesso! " );  
        em.getTransaction().commit();  
        em.close();  
        JPAUtil.close();  
    }  
}
```



Usa a classe
para enviar
e obter
dados do
banco

EXEMPLO 3 – TRAT. DE EXCEÇÕES

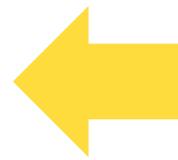
```
EntityManager em = JPAUtil.getEntityManager();
try {
    Usuario user = new Usuario("sbertagnolli2", "123456");
    em.getTransaction().begin();
    em.persist(user);
    System.out.println("Usuário salvo com sucesso! ");
    em.getTransaction().commit();
} catch (RuntimeException e) {
    if (em.getTransaction().isActive()) {
        em.getTransaction().rollback();
    }
} finally {
    em.close();
    JPAUtil.close();
}
```



Faz rollback se ocorreu exceção

EXEMPLO 4 – PESQUISANDO OBJETOS

```
EntityManager em = JPAUtil.getEntityManager();
try {
    em.getTransaction().begin();
    Usuario usuario = em.find(Usuario.class, 1L);
    em.remove(usuario);
    System.out.println("objeto excluído com sucesso");
    em.getTransaction().commit();
} catch (RuntimeException e) {
    if (em.getTransaction().isActive())
        em.getTransaction().rollback();
} finally {
    if (em != null) {
        em.close();
        JPAUtil.close();
    }
}
```



Pesquisa objeto
que tem a chave
primária igual a
1L (long)


EXEMPLO 4 – EXCLUINDO OBJETOS

```
EntityManager em = JPAUtil.getEntityManager();
try {
    em.getTransaction().begin();
    Usuario usuario = em.find(Usuario.class, 1L);
    em.remove(usuario);
    System.out.println("objeto excluído com sucesso");
    em.getTransaction().commit();
} catch (RuntimeException e) {
    if (em.getTransaction().isActive())
        em.getTransaction().rollback();
} finally {
    if(em != null) {
        em.close();
        JPAUtil.close();
    }
}
```

Remove o objeto que foi pesquisado na linha anterior

EXEMPLO 5 – MODIFICANDO OBJETOS

```
EntityManager em = JPAUtil.getEntityManager();
try {
    em.getTransaction().begin();
    Usuario usuario = em.find(Usuario.class, 2L);
    usuario.setIdentificador("novoid");
    System.out.println("objeto alterado com sucesso");
    em.getTransaction().commit();
} catch (RuntimeException e) {
    if (em.getTransaction().isActive())
        em.getTransaction().rollback();
} finally {
    if (em != null) {
        em.close();
        JPAUtil.close();
    }
}
```



Modifica o identificador
do objeto que tem a PK
igual a 2

EXEMPLO 6 – CONSULTANDO COM SQL

```
EntityManager em = JPAUtil.getEntityManager();
TypedQuery<Usuario> query =
    em.createQuery("SELECT obj FROM Usuario obj", Usuario.class);
List<Usuario> usuarios= query.getResultList();
for (Usuario usuario : usuarios) {
    System.out.println(usuario.toString());
}
em.close();
JPAUtil.close();
```



Seleciona todos os objetos da tabela usuário e coloca em uma List

EXEMPLOS - PACKAGE TESTES

ENUMERAÇÕES

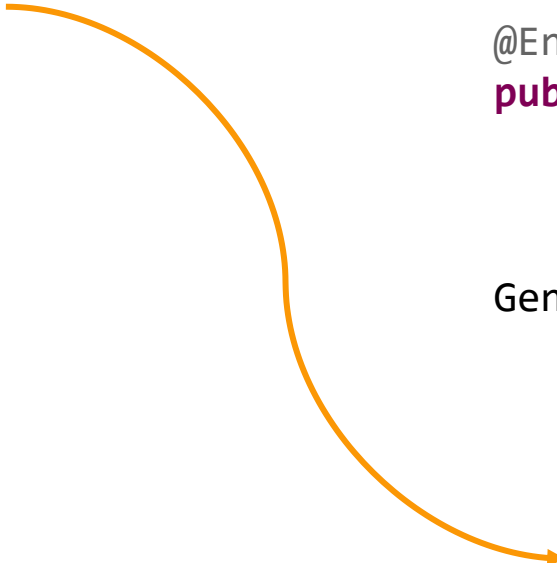
@ENUMERATED

@Enumerated que permite associar valores de uma enumeração Java (enum) a um atributo

Este tipo de anotação é bem útil para mapear atributos que têm um conjunto restrito de valores que podem ser especificados em um enum, como é o caso dos dias da semana

@ENUMERATED

```
public enum Perfil {  
    ADM, ALUNO, PROFESSOR;  
}
```



```
@Entity  
public class Usuario implements Serializable {  
    private static final long serialVersionUID = 1L;  
    @Id  
    @GeneratedValue(strategy =  
        GenerationType.IDENTITY)  
    private Long idUsuario;  
    private String identificador;  
    private String senha;  
  
    @Enumerated(EnumType.STRING)  
    private Perfil perfil;
```

No BD...

Salva o valor da enum associado ao objeto que foi persistido

```
Usuario2 user = new Usuario2("fulano", "123456", Perfil.ALUNO);  
em.getTransaction().begin();  
em.persist(user);  
System.out.println("Usuário salvo com sucesso! ");  
em.getTransaction().commit();
```

+ Opções



IDUSUARIO

data_cadastro

IDENTIFICADOR

PERFIL

SENHA



Edita



Copiar



Apagar

1

2021-11-29

fulano

ALUNO

123456

USANDO ENUM (STRING)

@Enumerated(EnumType.STRING)

Grava o valor da constante da enum no banco de dados

Exemplo: ADM, ALUNO ou PROFESSOR

Vantagem: dá para trocar a ordem dos enums a qualquer momento e sua aplicação continuará funcionando

Desvantagem: não dá para alterar os nomes das constantes da enum

USANDO ENUM (ORDINAL)

@Enumerated(EnumType.ORDINAL)

Grava a ordem do enum no banco de dados

Exemplo: 1 - ADM; 2 - ALUNO ou 3 - PROFESSOR

Vantagem: possibilita renomear as constantes da enum em qualquer momento

Desvantagem: não é possível mudar a ordem das constantes da enum

EXEMPLO - PACKAGE ENUMS

TEMPORAL

@TEMPORAL

Quando são usadas as classes “`java.util.Date`”, e “`java.util.Calendar`” deve-se explicitamente utilizar a anotação `@Temporal`

Essa anotação pode ser usada para:

- Só Data - `@Temporal(TemporalType.DATE)`
- Só Hora - `@Temporal(TemporalType.TIME)`
- Data e Hora - `@Temporal(TemporalType.TIMESTAMP)`

@TEMPORAL (EXEMPLOS)

```
//Somente Data
```

```
@Temporal(TemporalType.DATE)  
private Date dataCadastro;
```

```
// Somente hora
```

```
@Temporal(TemporalType.TIME)  
private Date horaAtual;
```

```
// Data e hora
```

```
@Temporal(TemporalType.TIMESTAMP)  
private java.util.Date dataHora;
```


USANDO DATAS

```
@Temporal(TemporalType.DATE)  
private Calendar dataCadastro;
```

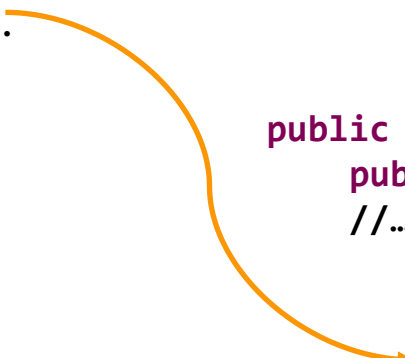
OU:

```
@Temporal(TemporalType.DATE)  
private java.util.Date dataCadastro;
```

EXEMPLO

@Entity

```
public class Usuario implements Serializable {  
    private static final long serialVersionUID = 1L;  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long idUsuario;  
  
    @Temporal(TemporalType.DATE)  
    @Column(name = "data_cadastro", nullable = false)  
    private Date dataCadastro;  
    //...
```



```
public class TesteJPA_1 {  
    public static void main(String[] args) {  
        //...  
        Usuario user = new Usuario("fulano", "123456");  
        user.setPerfil(Perfil.ALUNO);  
        user.setDataCadastro(new Date());  
    }  
}
```

No BD...

+ Opções



IDUSUARIO

data_cadastro

IDENTIFICADOR

PERFIL

SENHA



Edita



Copiar



Apagar

1

2021-11-29

fulano

ALUNO

123456

LOCALDATE, LOCALTIME E LOCALDATETIME

Quando são usadas as classes “`java.time.LocalDate`”, “`java.time.Localtime`”, “`java.time.LocalDateTime`” não é necessário definir a anotação `@Temporal`

Podemos usar assim:

```
@Column(name = "data_cad", nullable = false)  
private LocalDate dataCadastro;
```

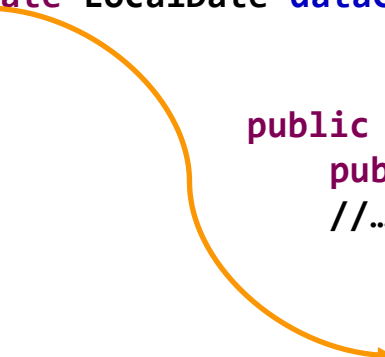
Obs.: `LocalDate` é equivalente a `TemporalType.DATE`,
`LocalDateTime` equivale a `TemporalType.TIMESTAMP` e `LocalTime` é `TemporalType.TIME`

EXEMPLO

@Entity




```
public class Usuario implements Serializable {  
    private static final long serialVersionUID = 1L;  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long idUsuario;  
  
    @Column(name = "data_cadastro", nullable = false)  
    private LocalDate dataCadastro;  
    //...
```

```
public class TesteJPA_1 {  
    public static void main(String[] args) {  
        //...  
        Usuario user = new Usuario("fulano", "123456", emails);  
        user.setPerfil(Perfil.ALUNO);  
        user.setDataCadastro(LocalDate.now());  
    }  
}
```



No BD...

Gera a coluna como sendo do tipo BLOB

+ Opções				IDUSUARIO	data_cad	IDENTIFICADOR	PERFIL	SENHA
<input type="checkbox"/>	 Editar	 Copiar	 Apagar	1	[BLOB - 44 Bytes]	fulano	ALUNO	123456

EXEMPLO - PACKAGE DATAS

COLEÇÕES

@ELEMENTCOLLECTION

@ElementCollection é usada quando queremos armazenar em uma coleção (List ou Set) objetos do tipo String ou classes invólucro (Character, Integer, Long, Float ...)

Por exemplo, um usuário tem vários e-mails que são armazenados em um conjunto

USANDO @ELEMENTCOLLECTION

@Entity

```
public class Usuario implements Serializable {
```

```
...
```

```
    @ElementCollection
```

```
    @CollectionTable(name="usuario_tem_emails")
```

```
    private Set<String> emails;
```

Se o nome da tabela não é informada o JPA tenta achar a tabela `usuario_emails`
Nome classe + `_` + nome atributo

`@CollectionTable` serve para definir qual tabela vai conter a lista de emails

No BD...

Salva o valor da enum associado ao objeto que foi persistido

```
Set<String> emails = new HashSet<String>();  
emails.add("fulano1@mail.com");  
emails.add("fulano2@mail.com");  
Usuario user = new Usuario("fulano", "123456", emails);
```

```
SELECT * FROM `usuario_tem_emails`
```

☐ Mostrar tudo | Número de registros: 25 ▼

+ Opções



☐ Edita ☐ Copiar ☐ Apagar

	IDUSUARIO	data_cadastro	IDENTIFICADOR	PERFIL	SENHA
1	2021-11-29	fulano	ALUNO	123456	

+ Opções

Usuario_IDUSUARIO	EMAILS
1	fulano1@mail.com
1	fulano2@mail.com

Liga cada um dos e-mails com o Usuário correspondente - usa FK

EXEMPLO - PACKAGE COLEÇÕES

EXERCÍCIOS

**Fazer os
exercícios 10 e
11 da lista
disponível no
Moodle!**
