

## MAPEAMENTO ÚNICO (TABELA ÚNICA PARA TODA A HIERARQUIA)

No mapeamento único, todas as subclasses de uma hierarquia de herança são armazenadas em uma única tabela no banco de dados. Essa tabela contém colunas para todos os atributos das superclasses e subclasses, além de uma coluna discriminadora que identifica a classe específica de cada registro.

### Características:

- Uma tabela no banco de dados armazena todas as informações de Pessoa, Professor e Aluno.
- Um atributo discriminador (geralmente chamado dtype ou algo similar) é usado para distinguir qual tipo de objeto está representado em cada linha.

### Benefícios:

- Simplicidade na estrutura do banco de dados.
- Melhor desempenho em consultas que retornam todas as instâncias da hierarquia, já que os dados estão em uma única tabela.

### Desvantagens:

- Pode haver muitas colunas com valores nulos, pois subclasses diferentes podem ter atributos que não se aplicam a outras.
- Alterações na hierarquia podem exigir mudanças significativas na tabela.

### Exemplo com as Classes:

- Pessoa é a classe base.
- Professor e Aluno são subclasses.
- Todos os atributos (como nome, idade, salário de Professor ou matrícula de Aluno) estarão em uma única tabela chamada, por exemplo, Pessoa.

## IMPLEMENTAÇÃO DO CÓDIGO DO MAPEAMENTO ÚNICO

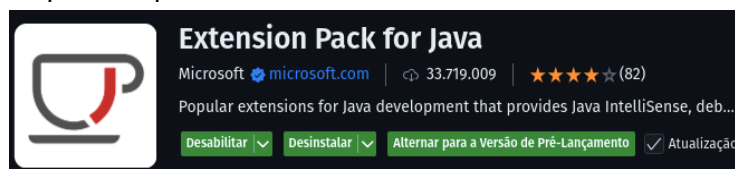
### Passo 1: Configurando o ambiente no GitHub Codespaces

No GitHub crie um novo repositório (ou pasta) chamado aulaJPA, depois inicie um Codespace.

- No repositório, clique na aba Codespaces e crie um novo Codespace.

### Passo 2: Instale a extensão do Java (Extension Pack for Java)

No Codespace/VS Code procure pela extensão **Extension Pack for Java** e faça a instalação



### Passo 3: Crie um projeto Maven

No explorador do Codespace/VS Code, procure no lado esquerdo pelo **“Maven”**, e clique em **New Project...**

- Selecione na lista o projeto **“maven-archetype-quickstart”**;

- Selecione a versão **1.4**;
- Escreva o nome do group Id “**com.portal**”, e pressione enter;
- Escreva o nome do projeto “**aula\_jpa\_tabunica**”;
- Pesquise e selecione a pasta criada no passo 1;
- No terminal escreva a palavra **1.0-SNAPSHOT**, e depois pressione enter;
- No terminal escreva a letra **Y**, e depois pressione enter;
- Pressione qualquer tecla para fechar o terminal.

#### Passo 4: Edite o projeto Maven criado

Feche o terminal, depois clique com o botão direito na pasta do projeto, e selecione “**Abrir no Terminal Integrado**”.

Obs.: Utilize o código anexado à aula para realizar as próximas tarefas.

- Exclua a pasta **test**, que está dentro da pasta **src**;
- Para esse exemplo, exclua a pasta **com**, contida dentro da pasta **java**;
- Crie a pasta **dominio** dentro da pasta **java**, e importe os arquivos **Pessoa.java**, **Professor.java** e **Aluno.java**.

Código do arquivo **Pessoa.java**

```
package dominio;

import java.io.Serializable;
import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.Inheritance;
import javax.persistence.InheritanceType;
import javax.persistence.DiscriminatorType;
import javax.persistence.DiscriminatorColumn;
import javax.persistence.DiscriminatorValue;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToOne;

@Entity
//identifica que apenas UMA tabela será criada no banco de dados.
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "tipo", length = 1, discriminatorType = DiscriminatorType.STRING)
public class Pessoa{

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private long idPessoa;
    private String nome;
    private String cpf;

    public Pessoa() {
        this("", "");
    }
    public Pessoa(String nome, String cpf) {
        setNome(nome);
        setCpf(cpf);
    }

    public long getIdPessoa() {
        return idPessoa;
    }
    public void setIdPessoa(long idPessoa) {
        this.idPessoa = idPessoa;
    }
}
```

```

    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getCpf() {
        return cpf;
    }
    public void setCpf(String cpf) {
        this.cpf = cpf;
    }

    @Override
    public String toString() {
        return "Pessoa [idPessoa=" + idPessoa + ", nome=" + nome + ", CPF=" + cpf + "];"
    }
}

```

### Código do arquivo **Professor.java**

```

package dominio;

import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;

@Entity //entidade de domínio
@DiscriminatorValue("P")
public class Professor extends Pessoa {

    private int matriculaProfessor;

    public Professor() {
        this("", "", 0);
    }
    public Professor(String nome, String cpf, int matriculaProfessor) {
        super(nome, cpf);
        setMatriculaProfessor(matriculaProfessor);
    }

    public int getMatriculaProfessor() {
        return this.matriculaProfessor;
    }

    public void setMatriculaProfessor(int matriculaProfessor) {
        this.matriculaProfessor = matriculaProfessor;
    }

    @Override
    public String toString() {
        return "Professor [idPessoa= " + super.getIdPessoa() + ", nome= " + super.getNome() + ", matriculaProfessor= " + getMatriculaProfessor() + "];"
    }

    public void imprimeDados() {
        System.out.println("Nome: " + super.getNome());
        System.out.println("CPF: " + super.getCpf());
        System.out.println("Matricula Professor: " + getMatriculaProfessor());
    }
}

```

## Código do arquivo **Aluno.java**

```
package dominio;

import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;

@Entity //entidade de domínio
@DiscriminatorValue("A")
public class Aluno extends Pessoa {

    private int matriculaAluno;

    public Aluno() {
        this("", "", 0);
    }
    public Aluno(String nome, String cpf, int matriculaAluno) {
        super(nome, cpf);
        setMatriculaAluno(matriculaAluno);
    }

    public int getMatriculaAluno() {
        return this.matriculaAluno;
    }
    public void setMatriculaAluno(int matriculaAluno) {
        this.matriculaAluno = matriculaAluno;
    }

    @Override
    public String toString() {
        return "Aluno [idPessoa= " + super.getIdPessoa() + ", nome= " + super.getNome() + ", matriculaAluno= " + getMatriculaAluno() + "];"
    }
    public void imprimeDados() {
        System.out.println("Nome: " + super.getNome());
        System.out.println("CPF: " + super.getCpf());
        System.out.println("Matricula Aluno: " + getMatriculaAluno());
    }
}
```

- Agora, crie a pasta **aplicativo**, dentro da pasta **java**, e importe o arquivo **Principal.java**;

## Código do arquivo **Principal.java**

```
package aplicativo;

import dominio.Pessoa;
import dominio.Professor;
import dominio.Aluno;
import java.util.ArrayList;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.Query;

public class Principal {

    public static void main(String[] args) {
        //Instancia o EntityManagerFactory com as configurações de persistencia
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("aula-jpa");
        //Intancia o EntityManager
        EntityManager em = emf.createEntityManager();
    }
}
```

```

Professor professor1 = new Professor("Rafael", "XXX.XXX.XXX-XX", 0001);
Aluno aluno1 = new Aluno("Miguel", "XXX.XXX.XXX-XX", 0001);

Professor professor2 = new Professor("Gabriel", "XXX.XXX.XXX-XX", 0001);
Aluno aluno2 = new Aluno("Uriel", "XXX.XXX.XXX-XX", 0001);

em.getTransaction().begin();// iniciar transação com banco de dados

em.persist(professor1);
em.persist(aluno1);
em.persist(professor2);
em.persist(aluno2);

//consulta em jpql
Query consultaP = em.createQuery("select professor from Professor professor");
ArrayList<Professor> listaP = (ArrayList<Professor>) consultaP.getResultList();

//consulta em jpql
Query consultaA = em.createQuery("select aluno from Aluno aluno");
ArrayList<Aluno> listaA = (ArrayList<Aluno>) consultaA.getResultList();

em.getTransaction().commit();//confirmar as alterações realizadas
emf.close();
em.close();

for(Professor objP: listaP) {
    System.out.println(objP);
}
for(Aluno objA: listaA) {
    System.out.println(objA);
}
}
}

```

- Importe a pasta **resources/META-INF** para a pasta **main**;

Código do arquivo **persistence.xml** que está dentro da pasta **main/resources/META-INF**:

```

<?xml version="1.0" encoding="UTF-8"?>
<persistence 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_1.xsd"
    version="2.1">

    <persistence-unit name="aula-jpa" transaction-type="RESOURCE_LOCAL">
        <properties>
            <property name="javax.persistence.jdbc.url"
                value="jdbc:postgresql://localhost:5432/bd_jpa_tabunica" />

            <property name="javax.persistence.jdbc.driver" value="org.postgresql.Driver" />
            <property name="javax.persistence.jdbc.user" value="root" />
            <property name="javax.persistence.jdbc.password" value="root" />

            <property name="hibernate.hbm2ddl.auto" value="update" />

            <!-- https://docs.jboss.org/hibernate/orm/5.4/javadocs/org/hibernate/dialect/package-summary.html -->
            <property name="hibernate.dialect" value="org.hibernate.dialect.PostgreSQLDialect" />
        </properties>
    </persistence-unit>
</persistence>

```

- Substitua o código do arquivo **pom.xml** pelo código do arquivo **pom.xml** do projeto anexado.

## Código do arquivo **pom.xml**

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.portal</groupId>
  <artifactId>aulajpa</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <properties>
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
  </properties>

  <dependencies>
    <!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
    <dependency>
      <groupId>org.hibernate</groupId>
      <artifactId>hibernate-core</artifactId>
      <version>6.1.7.Final</version>
      <type>pom</type>
    </dependency>

    <!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-entitymanager -->
    <dependency>
      <groupId>org.hibernate</groupId>
      <artifactId>hibernate-entitymanager</artifactId>
      <version>5.6.15.Final</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/org.postgresql/postgresql -->
    <dependency>
      <groupId>org.postgresql</groupId>
      <artifactId>postgresql</artifactId>
      <version>42.7.4</version>
    </dependency>
  </dependencies>
</project>
```

## Passo 5: Configurando o banco de dados PostgreSQL no Docker

- Crie um arquivo **docker-compose.yml** no diretório raiz do projeto.

```
version: '3.8'
services:
  postgres:
    image: postgres:15
    container_name: postgres_tabunica
    restart: always
    environment:
      POSTGRES_USER: root
      POSTGRES_PASSWORD: root
      POSTGRES_DB: bd_jpa_tabunica
    ports:
      - "5432:5432"
    volumes:
      - postgres-data:/var/lib/postgresql/data

volumes:
  postgres-data:
```

- Inicie o contêiner do PostgreSQL:

```
docker-compose up -d
```

- Verifique se o banco está rodando:

```
docker ps
```

### Passo 6: Compile e execute o projeto:

- Compile o projeto:

```
mvn clean compile
```

- Execute a classe principal:

```
mvn exec:java -Dexec.mainClass="aplicativo.Principal"
```

Comando para acessar o banco PostgreSQL

```
docker exec -it postgres_tabunica psql -U root -d bd_jpa_tabunica  
  
SELECT * FROM sua_tabela;
```

### Passo 6: Salve o Projeto no GitHub

No terminal do Codespaces, compile e execute os seguinte comandos:

- O comando “**git add .**” adiciona todas as alterações feitas nos arquivos ao *staging area* (área de preparação). O ponto (.) significa que todas as alterações no diretório atual serão incluídas. Se quiser adicionar apenas um arquivo específico, substitua o “.” pelo nome do arquivo. Segue o comando:

```
git add .
```

- O comando “**git commit -m "mensagem"**” cria um commit com as alterações que foram adicionadas ao *staging area*. A flag *-m* permite que uma mensagem descritiva seja adicionada ao commit. A mensagem deve ser clara e objetiva, explicando o que foi alterado. Por exemplo:

```
git commit -m "Aula JPA tabunica"
```

- O comando “**git push origin main**” envia os commits do seu repositório local para o repositório remoto no GitHub. O *origin* é o nome padrão do repositório remoto, e *main* é o nome do *branch* principal.

```
git push origin main
```