
Prog. Orientada a Objetos e Mapeamento Objeto-Relacional – IMD0104

Aula 04 – Mapeamento Objeto Relacional (MOR)

João Carlos Xavier Júnior

jcxavier@imd.ufrn.br

Mapeamento Objeto Relacional

- Também conhecida com:
 - ❖ Object Relational Mapping (ORM).
 - ❖ ou
 - ❖ Gateway-based Object Persistence (GOP).

Persistência automática e transparente de objetos de um aplicativo Java para tabelas em um banco de dados relacional, utilizando meta-dados que descrevem o mapeamento entre os objetos e o banco de dados. Em essência, transforma dados de uma representação para a outra.

Hibernate in Action

Componentes de uma solução ORM

- ❑ API para efetivação de operações CRUD (Create, Read, Update e Delete).
- ❑ Linguagem para **construção de consultas** referentes às classes e suas propriedades.
- ❑ Mecanismo de especificação dos meta-dados de mapeamento.
- ❑ Técnicas de interação com o SGBDR, incluindo:
 - ❖ Associações recuperadas sob demanda (*lazy loading*);
 - ❖ Etc.

Por que utilizar ORM?

- ❑ Produtividade:
 - ❖ Elimina a maior parte do código de infra-estrutura (BD e tabelas).
- ❑ Manutenibilidade:
 - ❖ Menos linhas de código, menos manutenção;
 - ❖ Alterações no BD e nas tabelas de forma mais fácil.
- ❑ Independência de fornecedor de BD.

Java Persistence API (JPA)

Java Persistence API (JPA)

- ❑ **Especificação** para gerenciamento de persistência.
- ❑ **Interface** que define um **padrão para o mapeamento** entre **objetos java** e **bancos de dados**.
- ❑ Diversos **frameworks de mapeamento** objeto/relacional implementam a JPA.
- ❑ Gerencia o desenvolvimento de entidades do Modelo Relacional usando a plataforma **Java SE** e **Java EE**.

Camadas de Persistência em Java

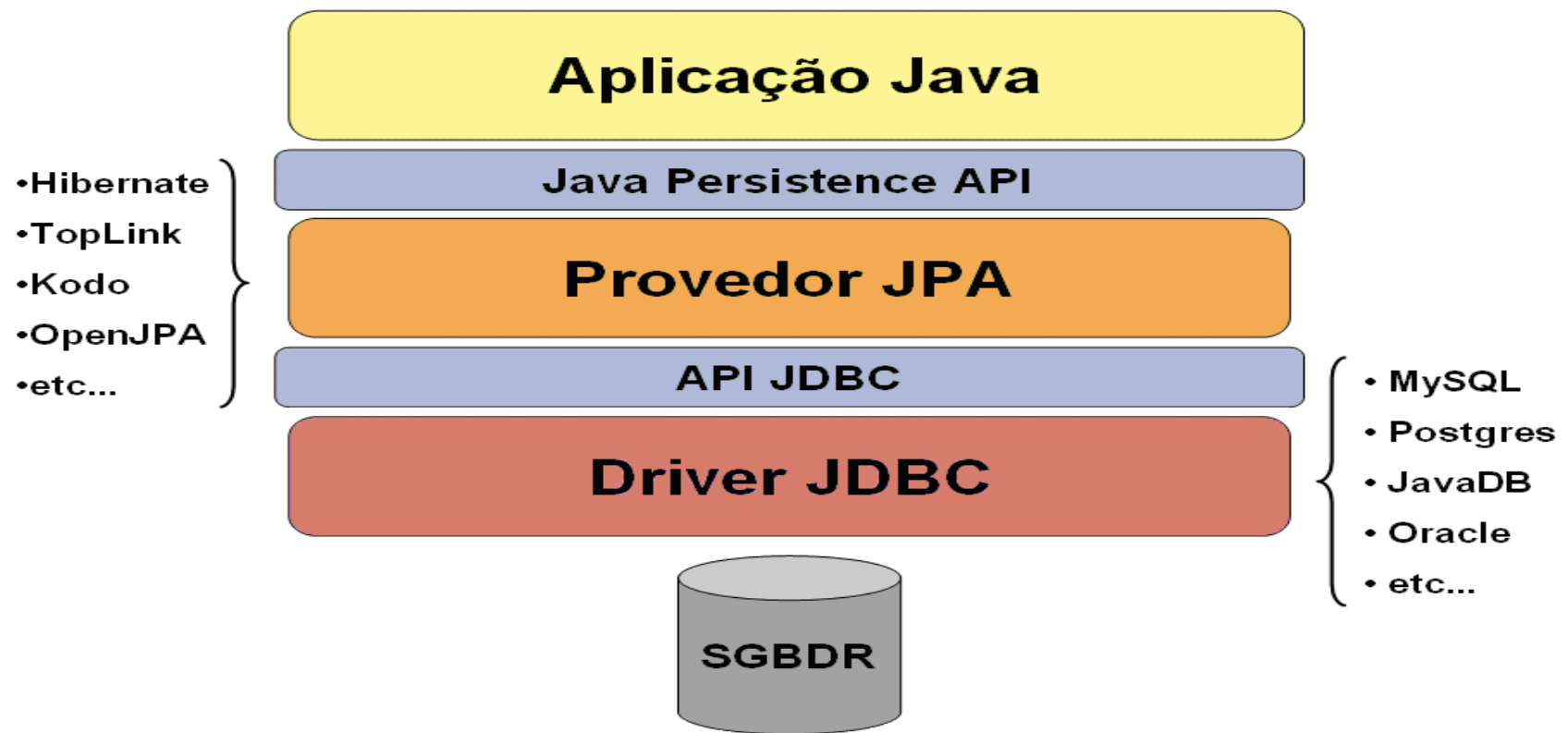
- ❑ Implementações de camadas de persistência:
 - ❖ **Hibernate**: framework que permite a persistência transparente de objetos em bases de dados utilizando JDBC e o mapeamento de classes para XML.
 - ❖ **Castor**: um framework de ligação de dados (data binding). Ele propoe a ligação entre objetos Java, documentos XML e dados SQL.
 - ❖ **Object-Relational Java Bridge (OJB)**: um framework do grupo **Apache** para prover uma implementação *open-source* para mapeamento de objetos.

Camadas de Persistência em Java

- ❑ Implementações de camadas de persistência:
 - ❖ **TopLink**: framework para mapeamento objeto-relacional desenvolvido pela [Oracle](#), que permite armazenar objetos Java em bancos de dados relacionais ou converter objetos Java em documentos XML.
 - ❖ **OpenJPA**: um framework de persistência open-source da Apache.
 - ❖ **Kodo**: framework que pode usar tanto as especificações JPA quanto JDO (Java Data Objects).

Java Persistence API

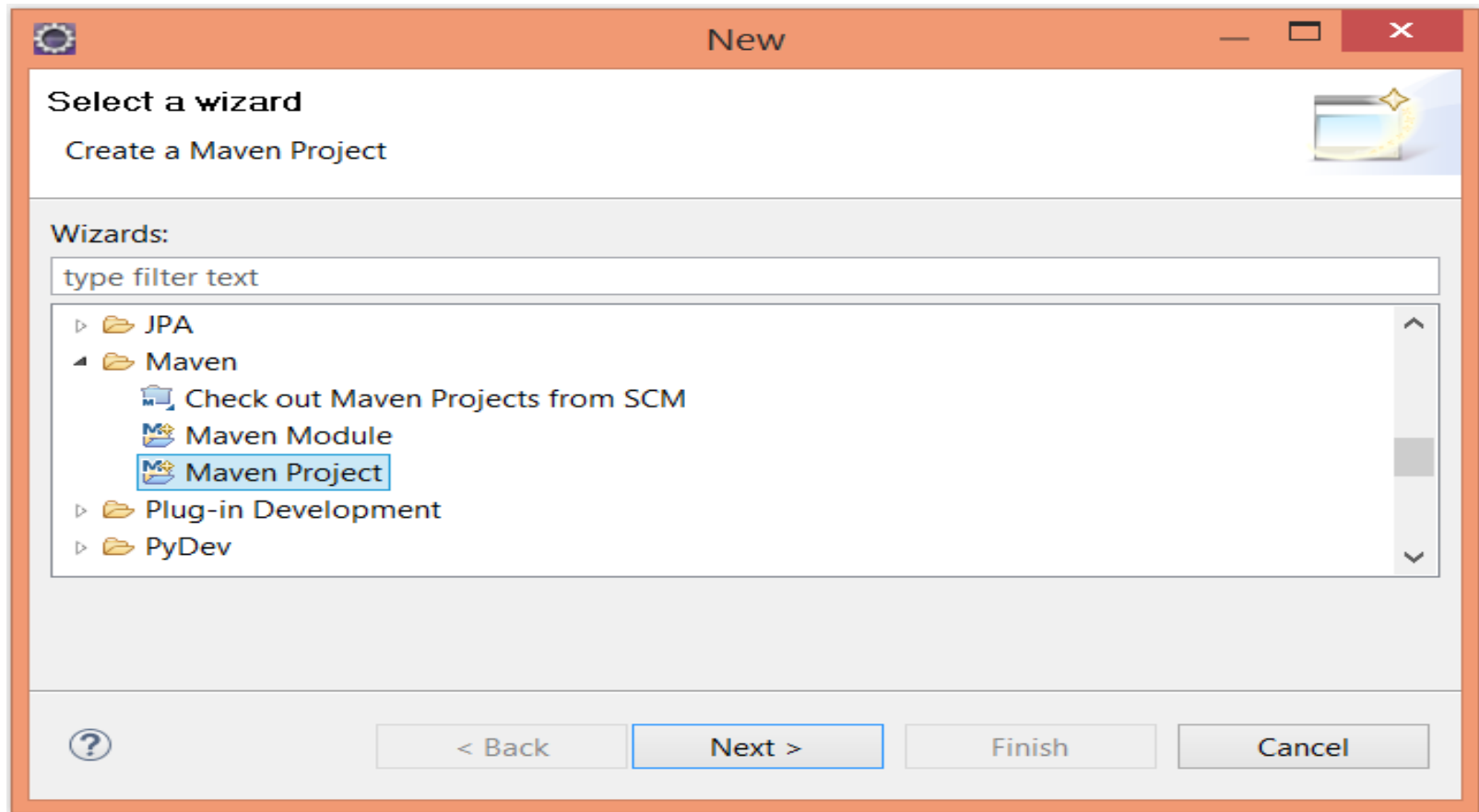
- ❑ JPA depende da JDBC e de um framework de persistência.



Configurando o Ambiente Projeto Maven

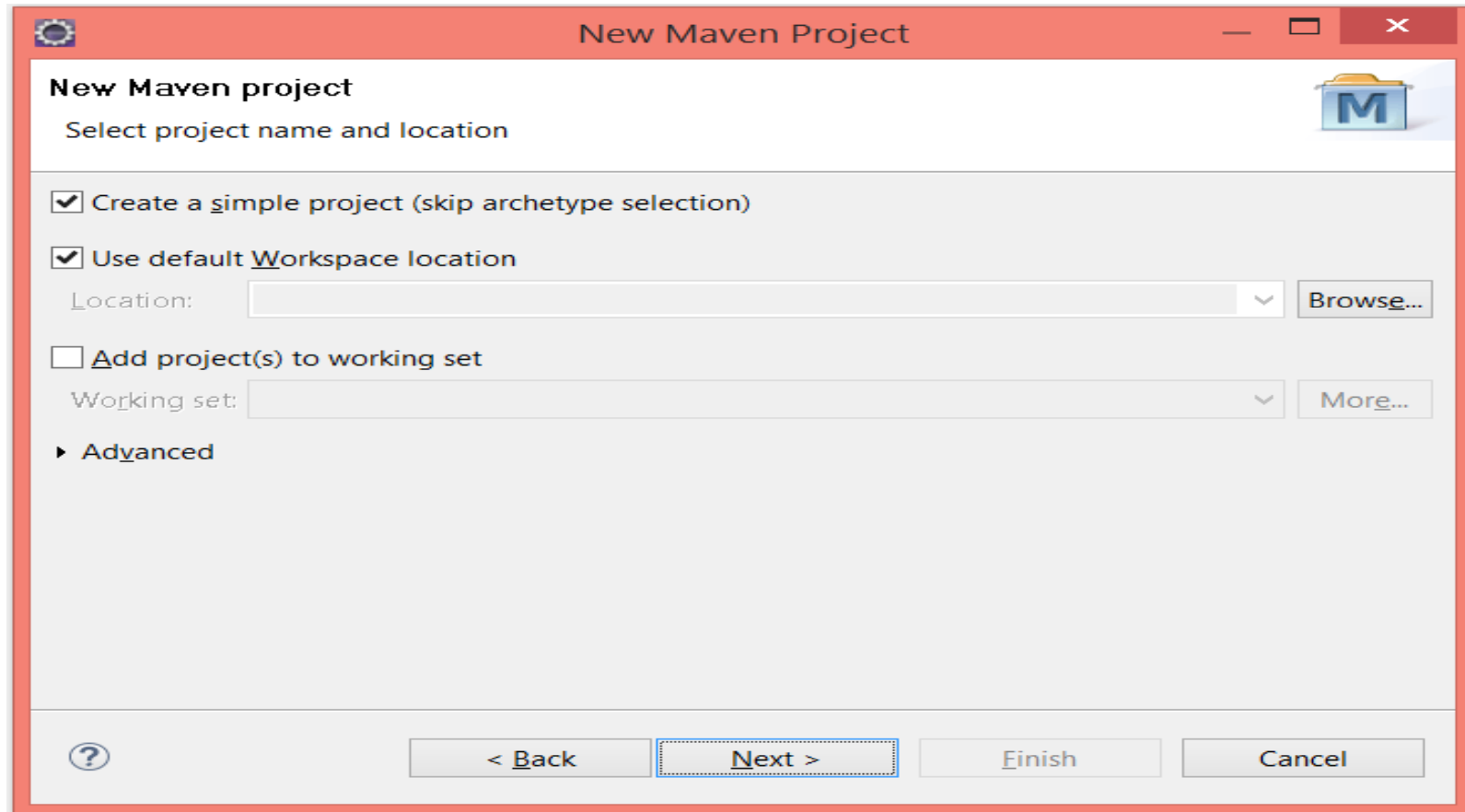
Projeto Maven

❑ Criando um novo Projeto Maven:



Projeto Maven

❑ Criando um novo Projeto Maven:



Projeto Maven

❑ Criando um novo Projeto Maven:

New Maven Project
Configure project

Artifact

Group Id:

Artifact Id:

Version:

Packaging:

Name:

Description:

Parent Project

Group Id:

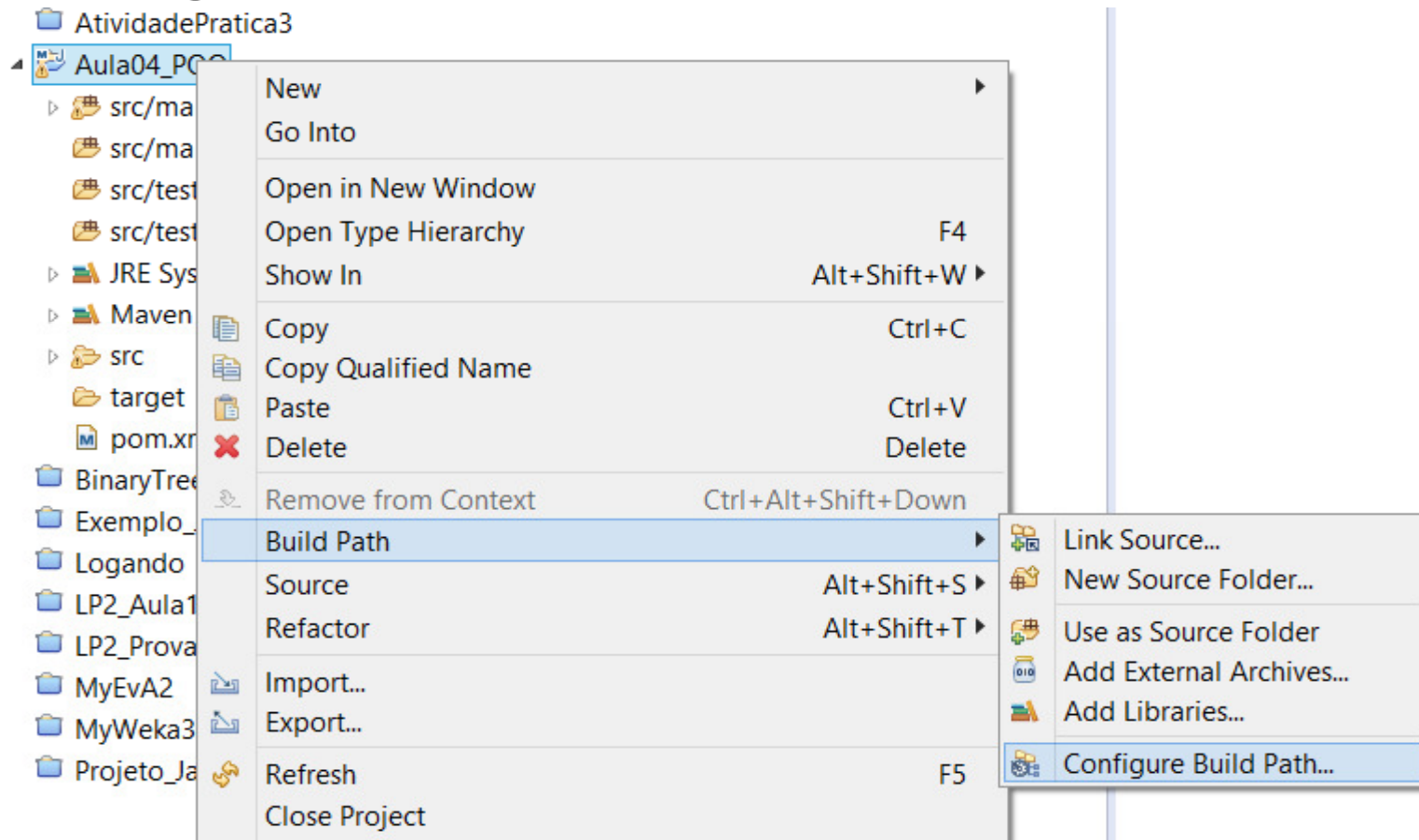
Artifact Id:

Version:

► **Advanced**

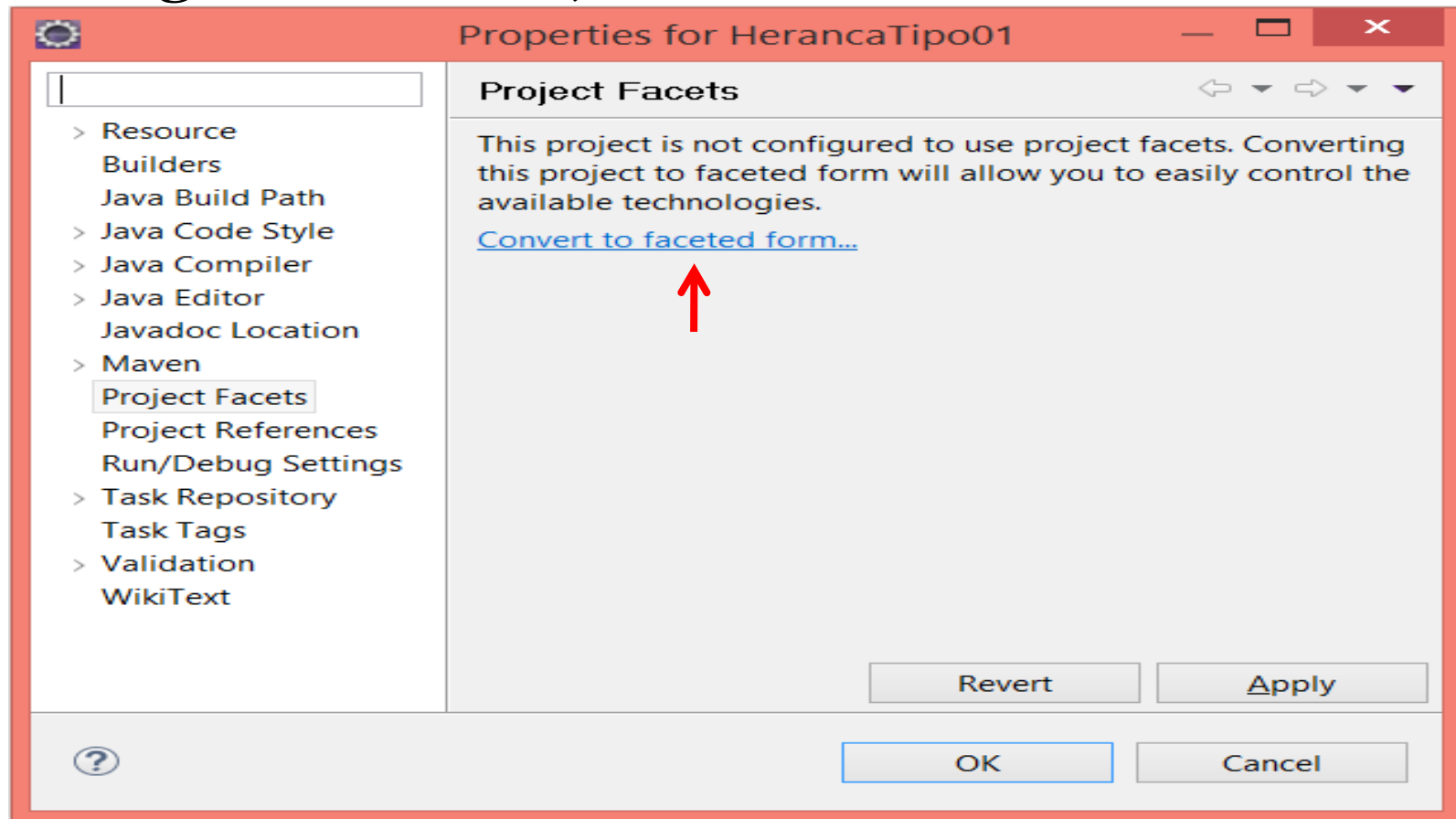
Projeto Maven

❑ Configurando o Projeto Maven:



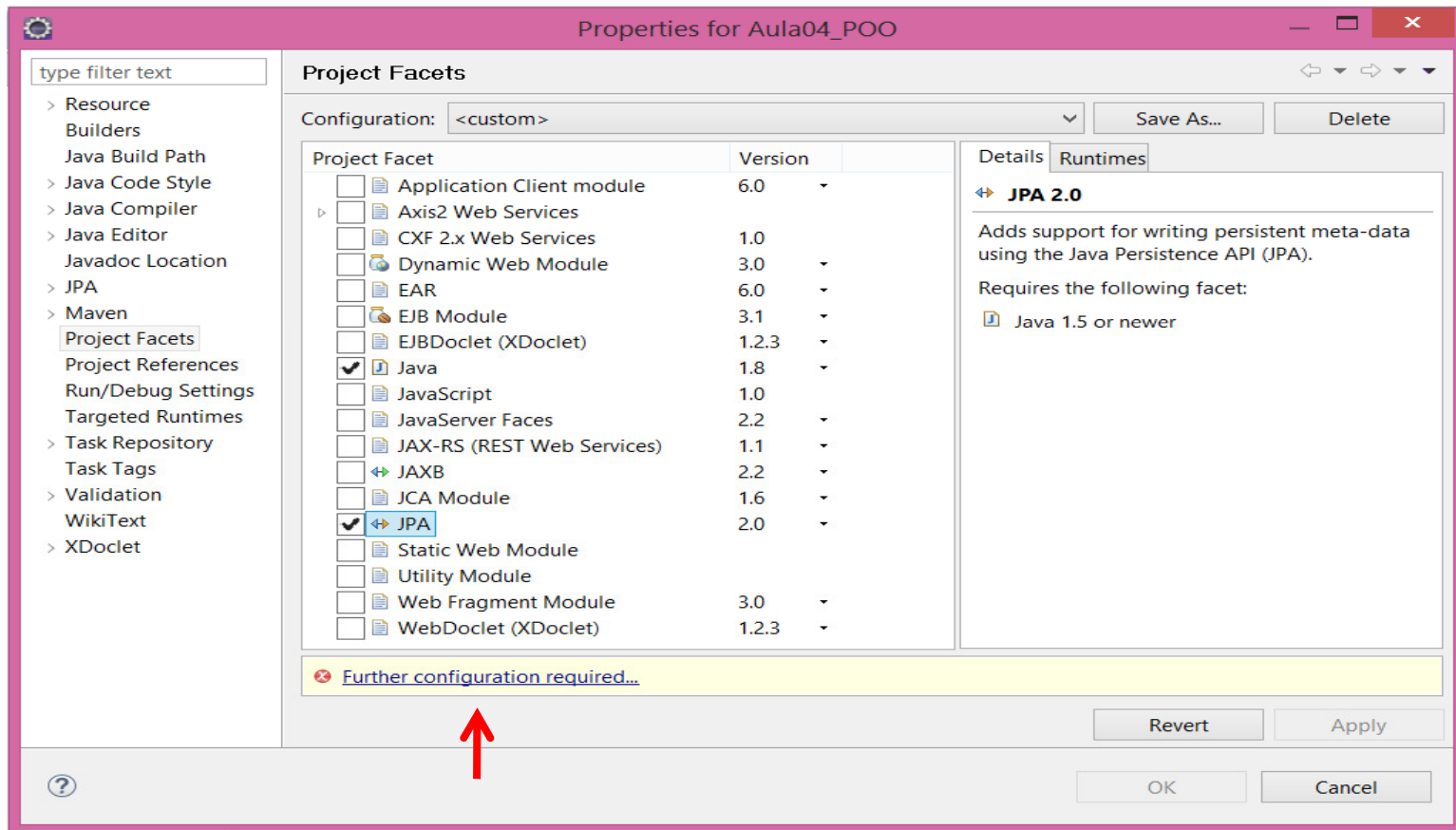
Projeto Maven

❑ Configurando o Projeto Maven:



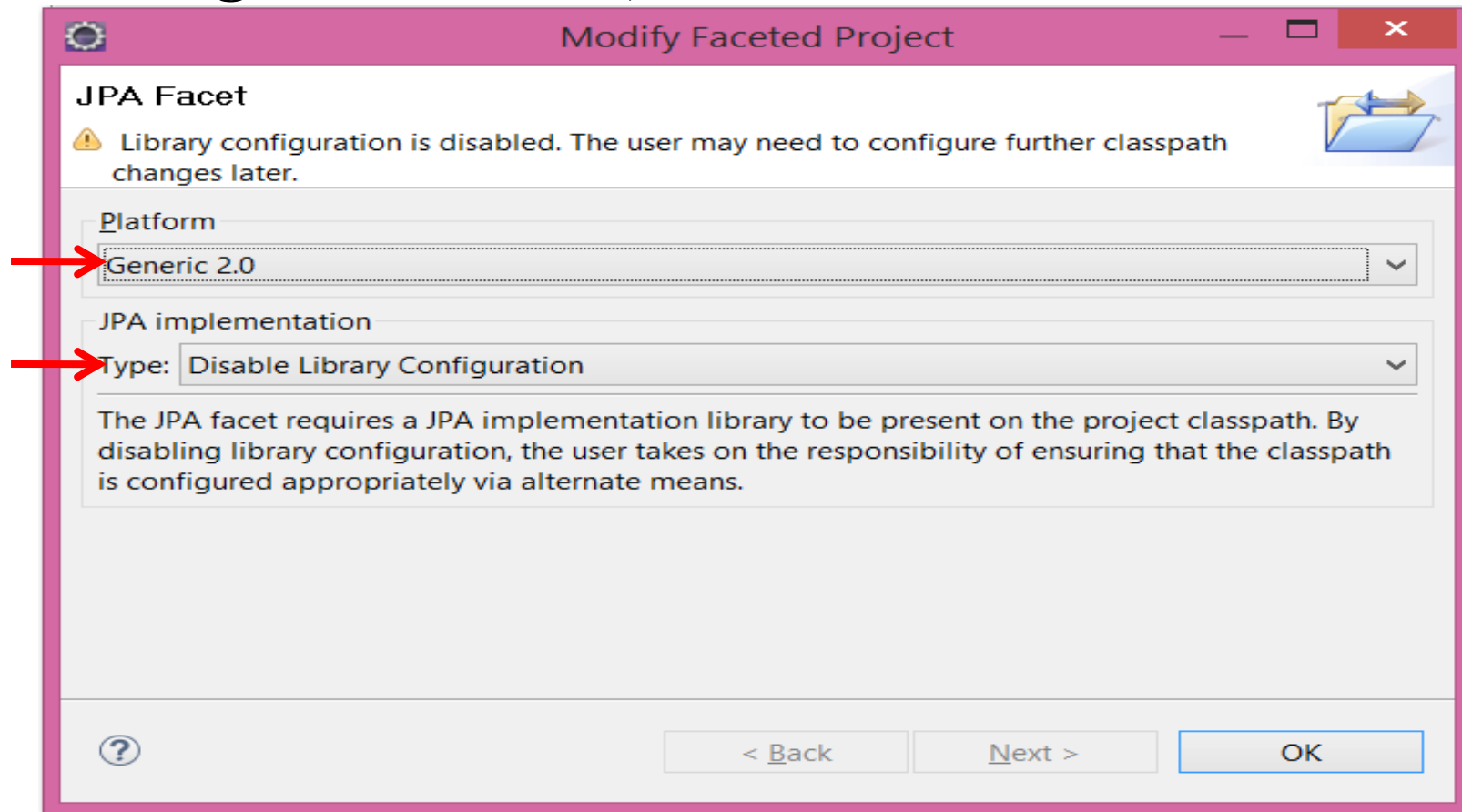
Projeto Maven

❑ Configurando o Projeto Maven:



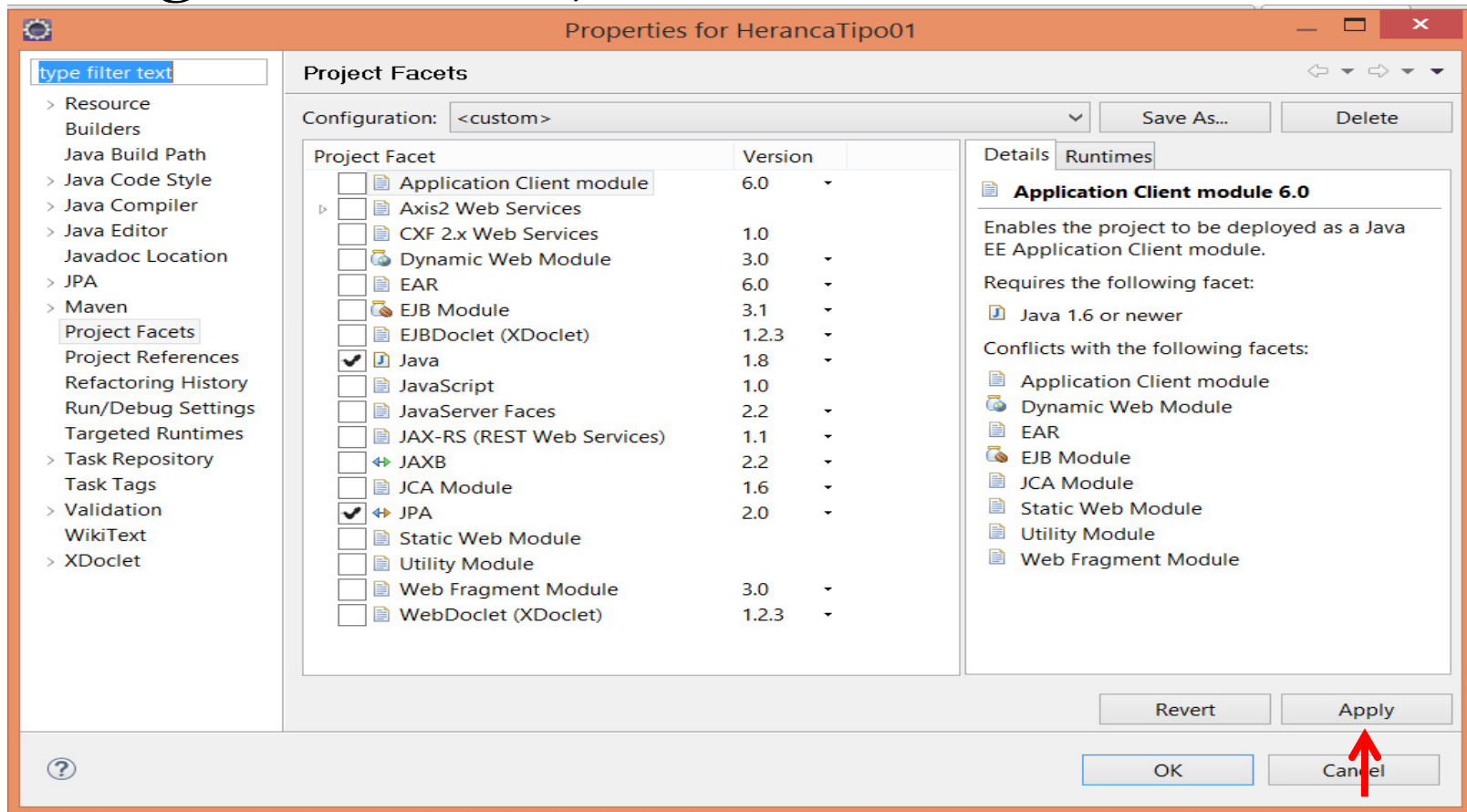
Projeto Maven

❑ Configurando o Projeto Maven:



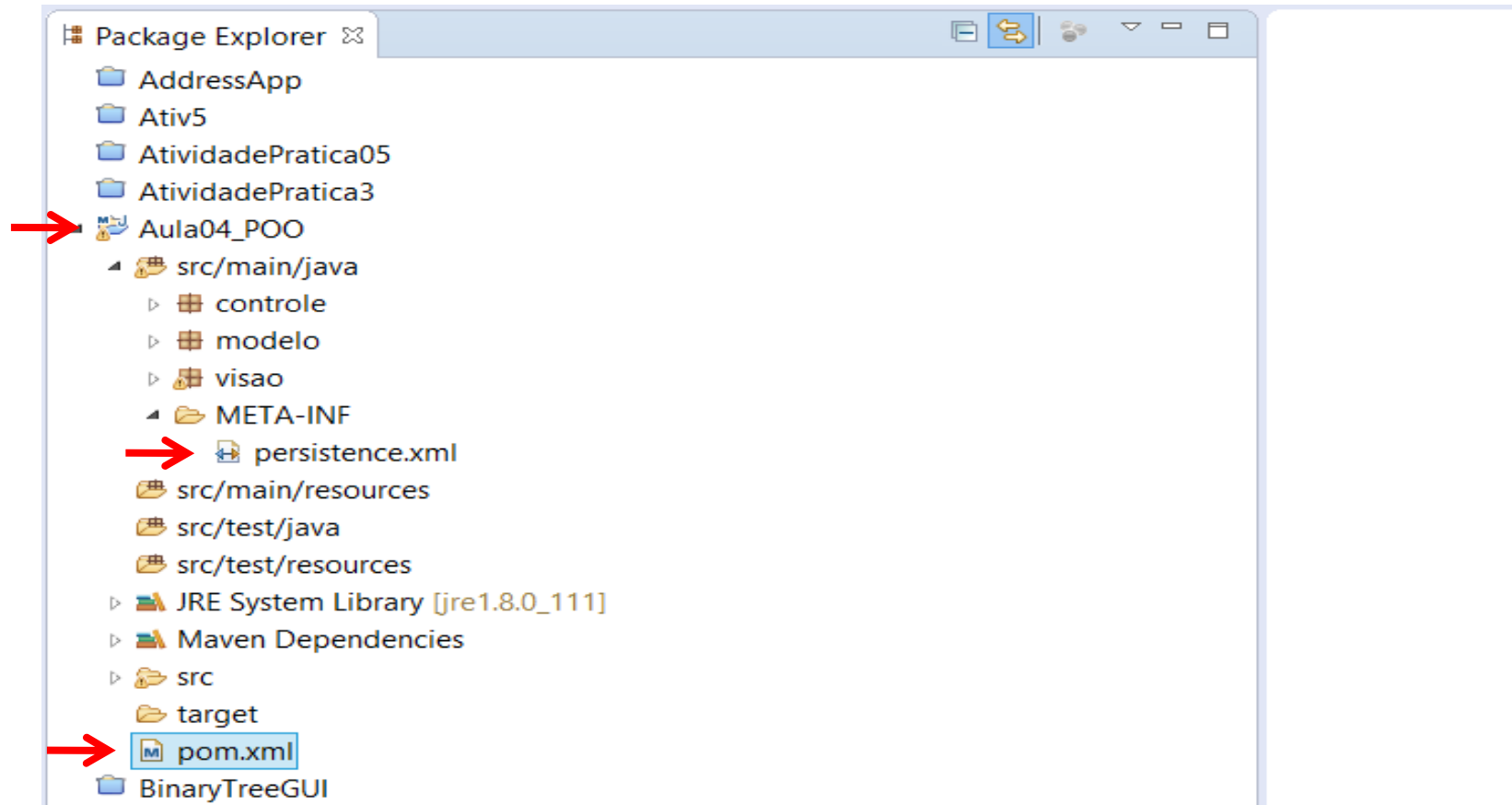
Projeto Maven

❑ Configurando o Projeto Maven:



Projeto Maven

❑ Projeto Maven criado:



Projeto Maven

❑ Pom.xml:

```
1<project xmlns="http://maven.apache.org/POM/4.0.0"
2  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
4
5   <modelVersion>4.0.0</modelVersion>
6   <groupId>br.ufrn.imd</groupId>
7   <artifactId>Aula04</artifactId>
8   <version>0.0.1-SNAPSHOT</version>
9
10  <dependencies>
11
12    <dependency>
13      <groupId>org.hibernate.javax.persistence</groupId>
14      <artifactId>hibernate-jpa-2.1-api</artifactId>
15      <version>1.0.0.Final</version>
16    </dependency>
17
18    <dependency>
19      <groupId>postgresql</groupId>
20      <artifactId>postgresql</artifactId>
21      <version>9.1-901.jdbc4</version>
22    </dependency>
23
24    <!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-entitymanager -->
25    <dependency>
26      <groupId>org.hibernate</groupId>
27      <artifactId>hibernate-entitymanager</artifactId>
28      <version>4.2.5.Final</version>
29    </dependency>
30  </dependencies>
31
32
33</project>
```

Projeto Maven

❑ Persistence.xml:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <persistence version="2.0"
3     xmlns="http://java.sun.com/xml/ns/persistence"
4     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5     xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
6
7     → <persistence-unit name="Aula04" transaction-type="RESOURCE_LOCAL">
8         <provider>org.hibernate.ejb.HibernatePersistence</provider>
9
10    → <class>modelo.Contato</class>
11
12    → <properties>
13        <property name="hibernate.connection.driver_class" value="org.postgresql.Driver"/>
14        <property name="hibernate.connection.url" value="jdbc:postgresql://localhost:5432/bd_poo04"/>
15        <property name="hibernate.connection.username" value="jcx1"/>
16        <property name="hibernate.connection.password" value="jcx1"/>
17        <property name="hibernate.hbm2ddl.auto" value="update"/>
18        <!-- <property name="hibernate.hbm2ddl.auto" value="create"/> -->
19    </properties>
20
21 </persistence-unit>
22
23 </persistence>
```

JPA Annotations

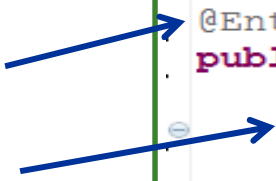
JPA Annotations

- ❑ Anotações para mapeamento OO-ER:
 - ❖ **@Entity**: informa que classe é uma **entidade**.
 - ❖ **@Id**: informa o atributo na classe que será utilizado como **chave primária**.

```
+ import java.io.Serializable;
@Entity
public class Contato implements Serializable{
    @Id
    private Long id;

    private String nome;
    private String telefone;
    private Date dataNascimento;
    private int idade;

    public Long getId() {
        return id;
    }
}
```



JPA Annotations

❑ Anotações para mapeamento OO-ER:

- ❖ **@Table**: informa o nome da **tabela** no banco de dados.
- ❖ **@Column**: informa os nomes das **colunas** de uma referida tabela.

```
import java.io.Serializable;

@Entity
@Table(name="contato")
public class Contato implements Serializable{

    @Id
    private Long id;

    @Column(nullable=false)
    private String nome;

    @Column(length=14)
    private String telefone;
    private Date dataNascimento;
    private int idade;
```


JPA Annotations

- ❑ Anotações para mapeamento OO-ER:
 - ❖ **@GeneratedValue**: anotação utilizada para geração automática de **chaves** (incrementais).
 - A anotação **@GeneratedValue** possui um atributo chamado **strategy**, que define a estratégia de geração de valores incrementados.
 - **@GeneratedValue(strategy=GenerationType.AUTO)** é a mais comum, permite pegar a estratégia preferida do BD.
 - As estratégias preferidas são **IDENTITY** (MySQL, SQLite e MsSQL) e **SEQUENCE** (Oracle e PostgreSQL).

JPA Annotations

- ❑ Anotações para mapeamento OO-ER:
 - ❖ **@SequenceGenerator**: anotação que possui um atributo **name** do qual é referenciado em **@GeneratedValue**, através do atributo **generator**.
 - O nome da sequence gerada no PostgreSQL, por exemplo, seria a colocada no atributo **sequenceName**, de **@SequenceGenerator**.

JPA Annotations

□ Exemplo:

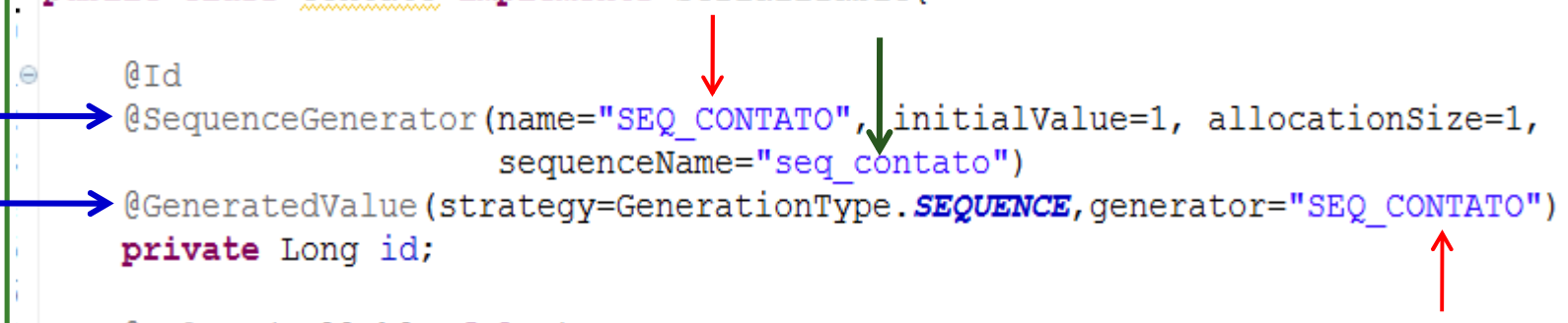
```
import java.io.Serializable;

@Entity
@Table(name="contato")
public class Contato implements Serializable{

    @Id
    @SequenceGenerator(name="SEQ_CONTATO", initialValue=1, allocationSize=1,
                      sequenceName="seq_contato")
    @GeneratedValue(strategy=GenerationType.SEQUENCE, generator="SEQ_CONTATO")
    private Long id;

    @Column(nullable=false)
    private String nome;

    @Column(length=14)
    private String telefone;
    private Date dataNascimento;
    private int idade;
```



JPA Annotations

❑ Resultado:

```
@Id
@SequenceGenerator(name="SEQ_CONTATO", initialValue=1, allocationSize=1,
                  sequenceName="minha_seq")
@GeneratedValue(strategy=GenerationType.SEQUENCE, generator="SEQ_CONTATO")
private Long id;
```



JPA Annotations

- ❑ Anotações para mapeamento OO-ER:
 - ❖ **@Temporal**: anotação para campos ou propriedades persistentes do tipo **java.util.Date** e **java.util.Calendar**.
 - O atributo **value** suporta três tipos de valores (**Date**, **Time** e **TimeStamp**).

```
@Temporal(value=TemporalType.DATE)  
private Date onlyDate;
```

```
@Temporal(value=TemporalType.TIME)  
private Date onlyTime;
```

```
@Temporal(value=TemporalType.TIMESTAMP)  
private Date dateAndTime;
```

JPA Annotations

□ Exemplo:

```
@Entity
@Table(name="contato")
public class Contato implements Serializable{

    @Id
    @SequenceGenerator(name="SEQ_CONTATO", initialValue=1, allocationSize=1,
                      sequenceName="seq_contato")
    @GeneratedValue(strategy=GenerationType.SEQUENCE, generator="SEQ_CONTATO")
    private Long id;

    @Column(nullable=false)
    private String nome;

    @Column(length=14)
    private String telefone;

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

JPA Annotations

□ Anotações para mapeamento OO-ER:

❖ **@Transient**: permite que um determinado **atributo** possa ser manipulado na aplicação, mas jamais seja persistido no BD.

- Muito usado em situações de **campo calculado**.

```
@Column(nullable=false)
private String nome;
```

```
@Column(length=14)
private String telefone;
```

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

```
@Transient
private int idade;
```

EntityManagerFactory

e

EntityManager

Provedor de Persistência

❑ **EntityManagerFactory** (`javax.persistence`)

- ❖ Fábrica de EntityManagers.
- ❖ **Apenas uma instância por aplicação.**
- ❖ Definida em uma unidade de persistência.
- ❖ EntityManagerFactory ligada a uma unidade de persistência (one-to-one).

❑ **EntityManager** (`javax.persistence`):

- ❖ Entity Managers configurados para **persistir** ou **gerenciar** tipos específicos de objetos, **ler** e **escreve-los** numa base de dados.
- ❖ É implementado por um Provedor de Persistência (persistence provider).

EntityManager

- ❑ **Métodos Comuns** em CRUD's:
 - ❖ **Merge**: atualiza um objeto.
 - ❖ **Persist**: salva um novo objeto.
 - ❖ **Remove**: remove um objeto.
 - ❖ **Find**: recupera um objeto através do *id*.

Controle Transacional

- Dentre os possíveis controles:
 - ❖ `EntityManager.getTransaction().begin();`
 - ❖ `EntityManager.getTransaction().commit();`
 - ❖ `EntityManager.getTransaction().rollback();`

Prática

- ❑ Criar uma classe Contato com os seguintes atributos:
 - ❖ int id;
 - ❖ String nome;
 - ❖ String telefone;
 - ❖ Date dataNascimento.
- ❑ Criar uma classe DAO para manipular os objetos
- ❑ Criar uma classe de conexão que instancie adequadamente EntityManagerFactory e EntityManager
- ❑ Criar uma classe (principal) para manipular objetos do tipo Contato

Associações entre Classes e Tabelas

Relacionamentos

❑ Associação (um para um):

❖ @OneToOne:

- Indica um relacionamento um para um;
- Usado para indicar qual tabela mapeia as informações de relacionamento.

❖ @JoinColumn:

- Elemento opcional que permite definir informações sobre a chave estrangeira.

Relacionamentos

❑ Um-para-um:

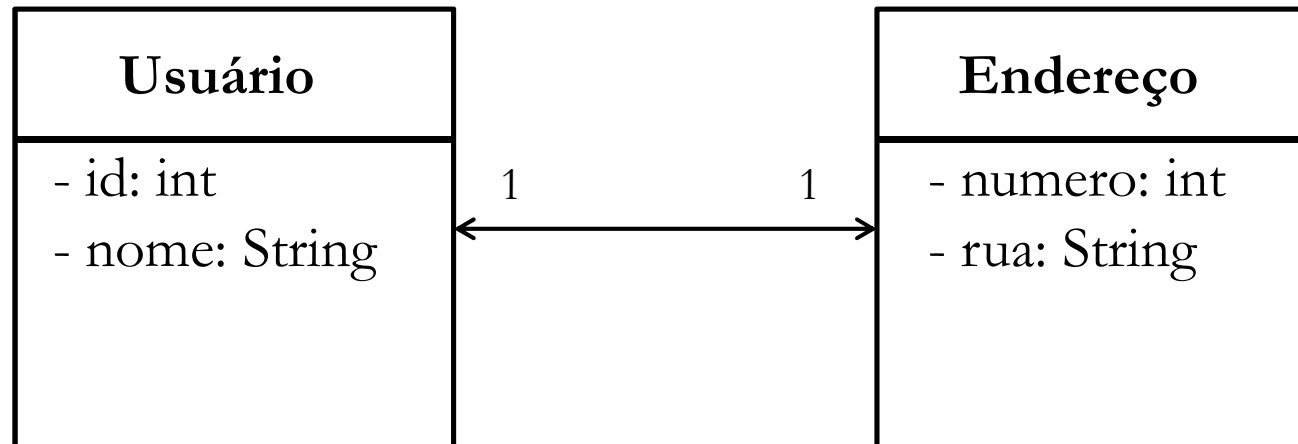
❖ Parâmetros da anotação @OneToOne:

- **cascade**: define ações automatizadas no relacionamento (ex.: ao apagar um Endereco, apagar também um Usuario).
- **fetch**: o valor padrão é **EAGER**. Ou seja, ao carregar o Endereco já será feita a consulta relacionada ao Usuario de modo automático (**join automático**).

❖ @JoinColumn(name="id_usu", nullable=false): define qual é a coluna mapeada para fazer a união na consulta. É indicado o **nome** da coluna, que não pode ser **nulo**.

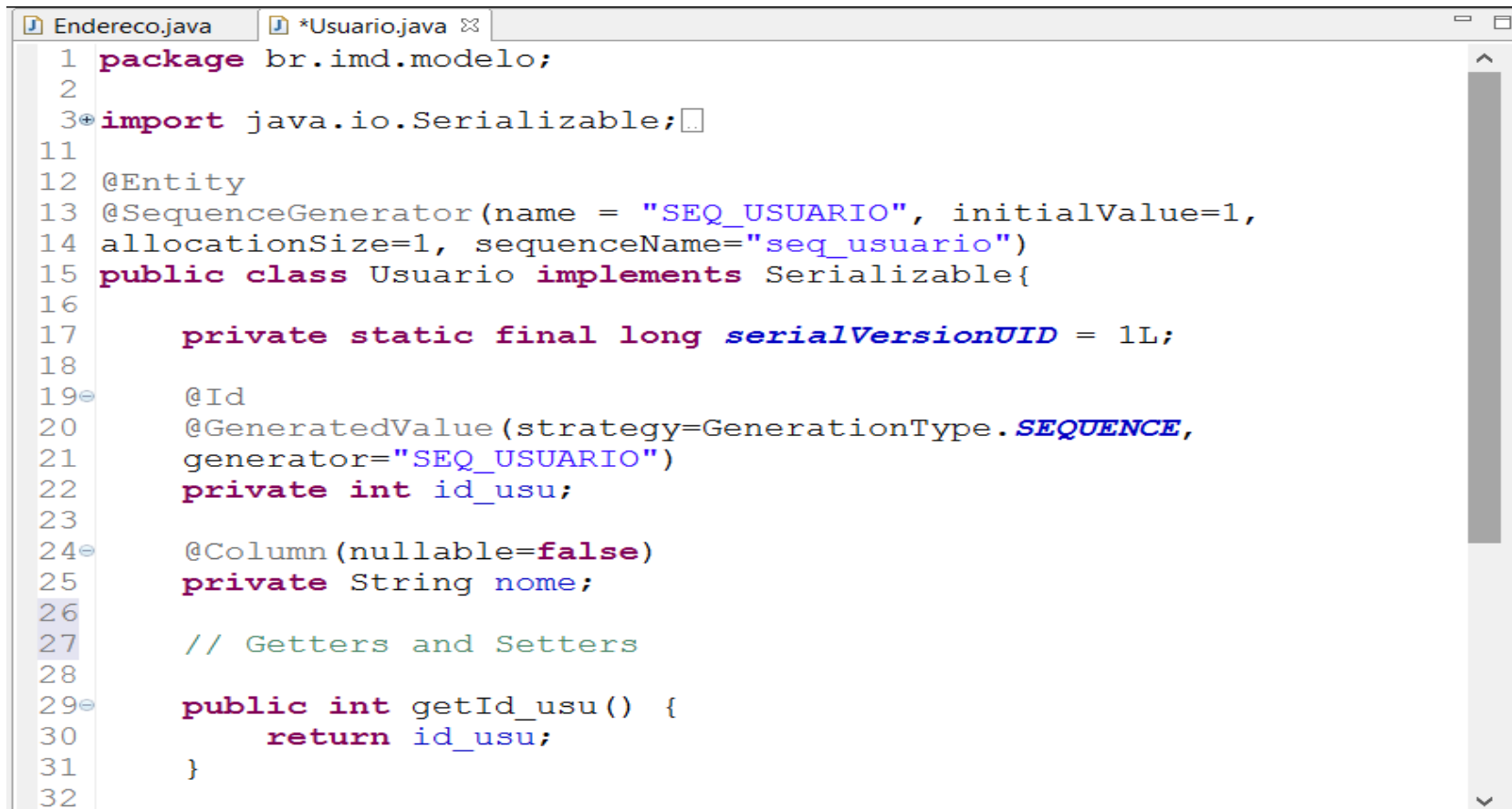
Relacionamentos

❑ Associação (um-para-um):



Relacionamentos

❑ Exemplo (1:1):

A screenshot of a Java IDE window showing the code for a file named 'Endereco.java'. The code defines a 'Usuario' entity with a sequence generator, a static serial version UID, and fields for 'id_usu' and 'nome'. It includes JPA annotations like @Entity, @SequenceGenerator, @Id, @GeneratedValue, and @Column. The code is as follows:

```
1 package br.imd.modelo;
2
3 import java.io.Serializable;
4
11
12 @Entity
13 @SequenceGenerator(name = "SEQ_USUARIO", initialValue=1,
14 allocationSize=1, sequenceName="seq_usuario")
15 public class Usuario implements Serializable{
16
17     private static final long serialVersionUID = 1L;
18
19     @Id
20     @GeneratedValue(strategy=GenerationType.SEQUENCE,
21 generator="SEQ_USUARIO")
22     private int id_usu;
23
24     @Column(nullable=false)
25     private String nome;
26
27     // Getters and Setters
28
29     public int getId_usu() {
30         return id_usu;
31     }
32 }
```

Relacionamentos

Exemplo (1:1):

```
Endereco.java
1 package br.imd.modelo;
2
3 import java.io.Serializable;
14
15 @Entity
16 @SequenceGenerator(name = "SEQ_ENDERECO", initialValue=1,
17 allocationSize=1, sequenceName="seq_endereco")
18 public class Endereco implements Serializable{
19
20     private static final long serialVersionUID = 1L;
21
22     @Id
23     @GeneratedValue(strategy=GenerationType.SEQUENCE,
24 generator="SEQ_ENDERECO")
25     private int id_end;
26
27     private int numero;
28     private String rua;
29
30     @OneToOne(cascade=CascadeType.ALL, fetch=FetchType.LAZY)
31     @JoinColumn(name="id_usu", nullable=false)
32     private Usuario user;
33
34     // Getters and Setters
35 }
```

Relacionamentos

❑ Um-para-um bidirecional:

- ❖ Um **relacionamento bidirecional** se faz presente quando duas classes de um mapeamento contêm referências mútuas.
- ❖ Permite **consultar** em **qualquer lado** do relacionamento.
- ❖ O parâmetro “**mappedBy**” indica quem é o *owner* desse relacionamento, o lado mais forte.

Relacionamentos

❑ Um-para-um bidirecional:

❖ Alteração em Usuario:

```
@Entity
@SequenceGenerator(name="SEQ_USUARIO", initialValue=1,
allocationSize=1, sequenceName="seq_usuario")
public class Usuario implements Serializable {

    @Id
    @GeneratedValue(strategy=GenerationType.SEQUENCE, generator="SEQ_USUARIO")
    private int id;

    @Column(nullable=false)
    private String nome;

    @OneToOne(mappedBy = "user")
    private Endereco end;

    // Getters and Setters
}
```

Relacionamentos

❑ Um-para-um bidirecional:

❖ Consulta:

```
Usuario user = manager.find(Usuario.class, 1);  
System.out.println(user.getNome());  
System.out.println(user.getEnd().getRua());  
  
System.out.println("\n");  
  
Endereco end = manager.find(Endereco.class, 3);  
System.out.println(end.getUser().getNome());  
System.out.println(end.getRua());  
System.out.println("Dados encontrados!!!");
```

Associações

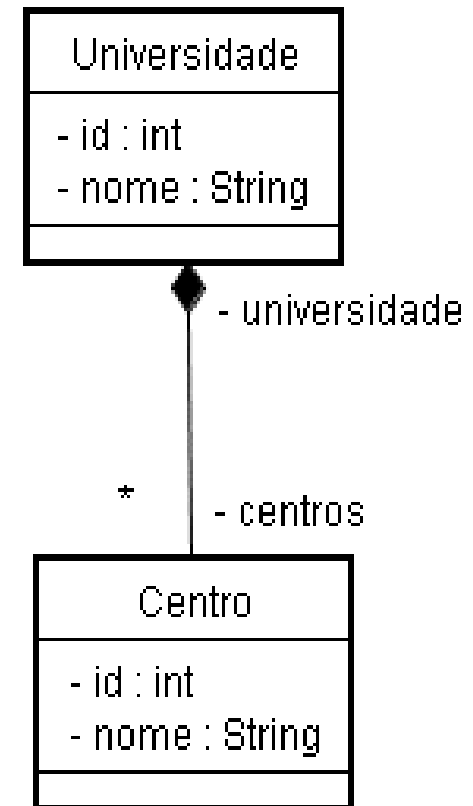
❑ Um-para-muitos:

- ❖ Relacionamentos **um para muitos** bidirecionais são análogos ao **um para um**, fazendo o uso de `@OneToMany` (no lugar de `@OneToOne`).

Associações

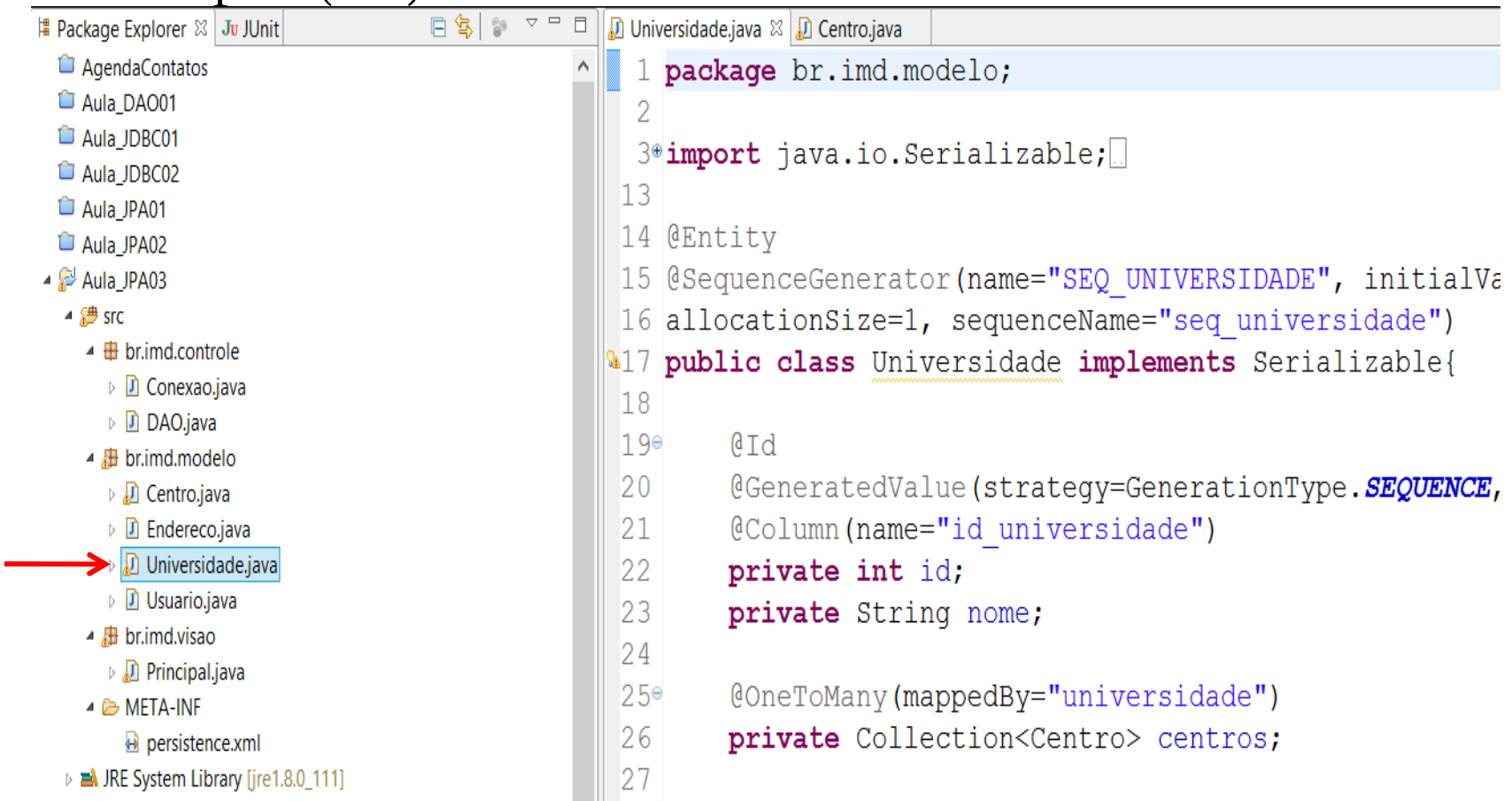
❑ Um-para-muitos:

- ❖ Domínio: uma universidade pode estar associada a vários centros.
- ❖ A classe Universidade deve ter um atributo que é coleção de centros.
- ❖ Anotação `@OneToMany` na classe Universidade.



Associações

Exemplo (1:n):



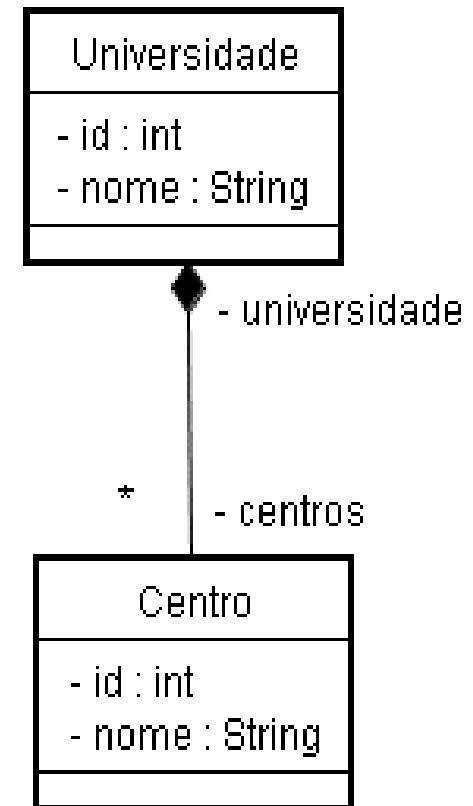
The screenshot shows an IDE with a Package Explorer on the left and a code editor on the right. The Package Explorer displays a project structure with a red arrow pointing to the `Universidade.java` file. The code editor shows the following Java code:

```
1 package br.imd.modelo;
2
3 import java.io.Serializable;
4
13
14 @Entity
15 @SequenceGenerator(name="SEQ_UNIVERSIDADE", initialValue=
16 allocationSize=1, sequenceName="seq_universidade")
17 public class Universidade implements Serializable{
18
19     @Id
20     @GeneratedValue(strategy=GenerationType.SEQUENCE,
21     @Column(name="id_universidade")
22     private int id;
23     private String nome;
24
25     @OneToMany(mappedBy="universidade")
26     private Collection<Centro> centros;
27
```


Associações

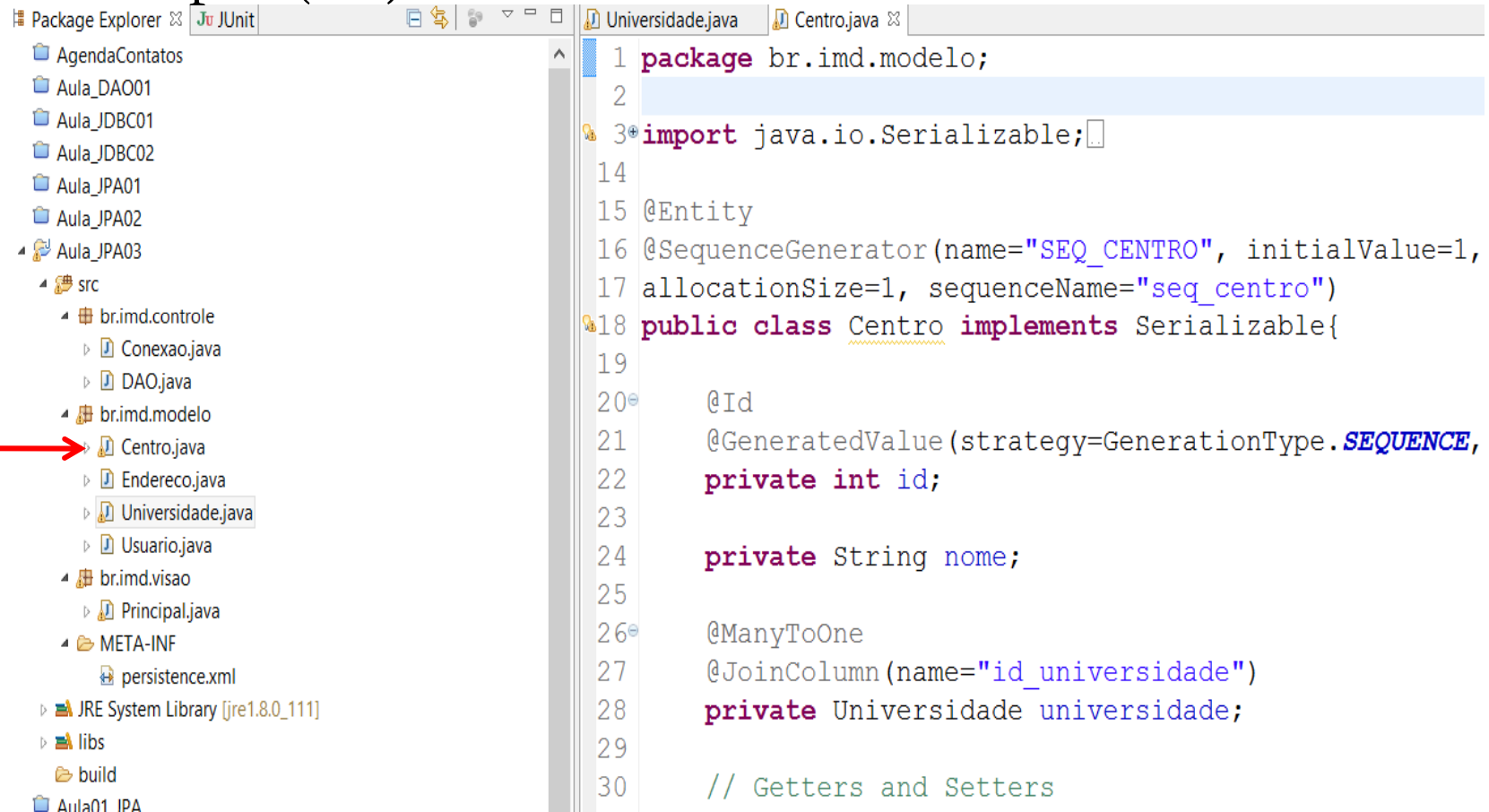
❑ Muitos-para-um:

- ❖ Domínio: vários centros podem estar associados a uma única universidade.
- ❖ A classe Centro deve ter um atributo do tipo Universidade.
- ❖ Anotação `@ManyToOne` na classe Centro.



Associações

Exemplo (n:1):



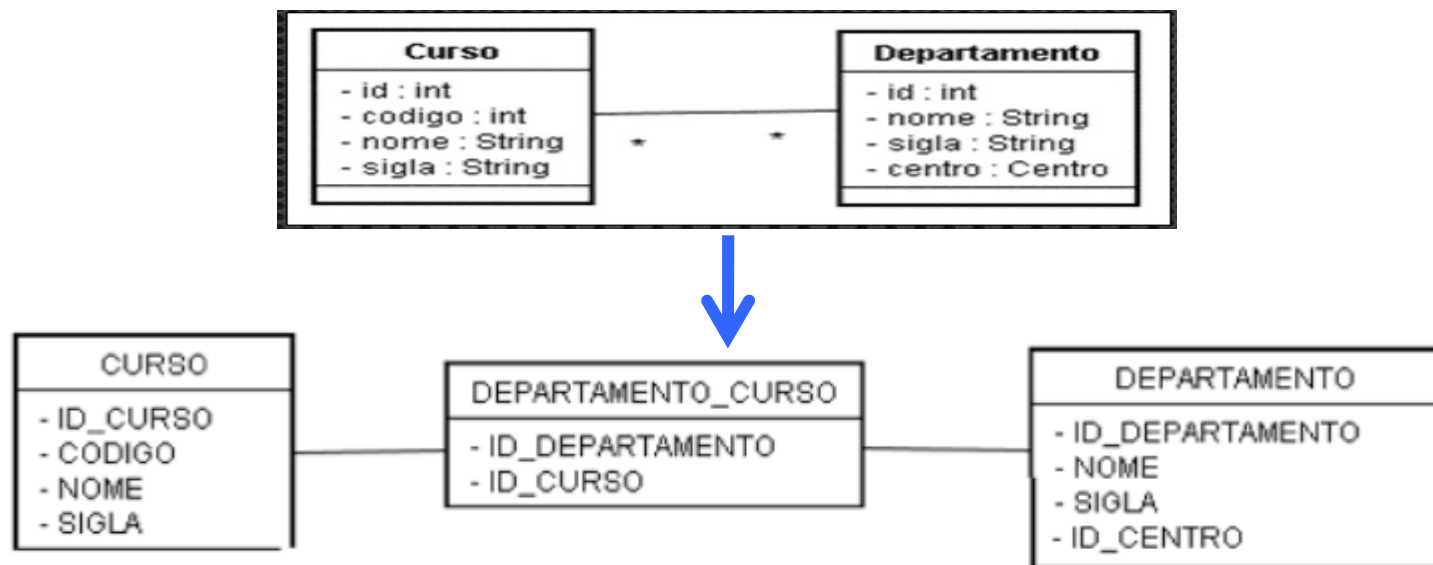
The screenshot displays an IDE interface with a Package Explorer on the left and a code editor on the right. The Package Explorer shows a project structure with a package `br.imd.modelo` containing classes `Centro.java`, `Endereco.java`, `Universidade.java`, and `Usuario.java`. A red arrow points to `Centro.java`. The code editor shows the following Java code for `Centro.java`:

```
1 package br.imd.modelo;
2
3 import java.io.Serializable;
4
14
15 @Entity
16 @SequenceGenerator(name="SEQ_CENTRO", initialValue=1,
17 allocationSize=1, sequenceName="seq_centro")
18 public class Centro implements Serializable{
19
20     @Id
21     @GeneratedValue(strategy=GenerationType.SEQUENCE,
22     private int id;
23
24     private String nome;
25
26     @ManyToOne
27     @JoinColumn(name="id_universidade")
28     private Universidade universidade;
29
30     // Getters and Setters
```

Associações

❑ Muitos-para-Muitos:

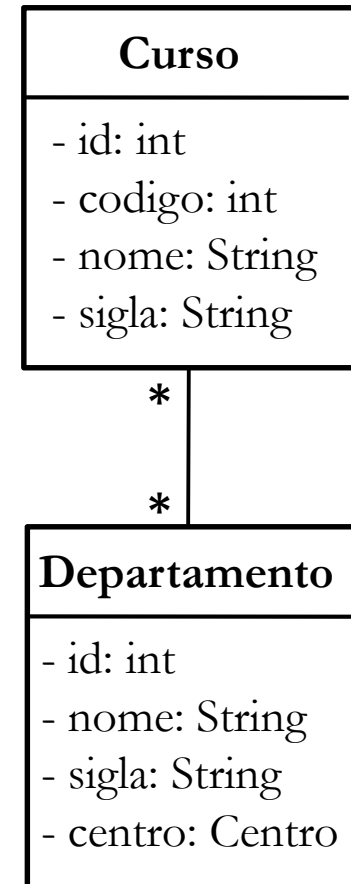
- ❖ Em relacionamentos **muitos-para-muitos** (ou n-para-m), convencionou-se criar uma tabela intermediária que armazene pares de chaves, identificando os dois lados do relacionamento.



Associações

❑ Muitos-para-muitos:

- ❖ Domínio: vários cursos podem estar associados a vários departamentos.
- ❖ A classe **Curso** deve ter um atributo que é coleção de departamentos.
- ❖ A classe **Departamento** deve ter um atributo que é coleção de cursos.
- ❖ Anotação **@ManyToMany**.



Associações

❑ Muitos-para-muitos:

❖ **@JoinTable**: informa qual tabela intermediária receberá as chaves primárias de ambas as relações (**Curso** x **Departamento**).

- **name**: nome da tabela intermediária;
- **joinColumns**: determina qual a chave-primária referente ao lado esquerdo da relação (**Curso**);
- **inverseJoinColumns**: determina qual a chave-primária referente ao lado direito da relação (**Departamento**).

Associações

❑ Muitos-para-muitos:

```
@Entity
@SequenceGenerator(name="SEQ_CURSO", initialValue=1,
allocationSize=1, sequenceName="seq_curso")
public class Curso implements Serializable {
    @Id
    @GeneratedValue(strategy=GenerationType.SEQUENCE, generator="SEQ_CURSO")
    @Column(name="id_curso")
    private int id;

    private int codigo;
    private String nome;
    private String sigla;

    @ManyToMany(fetch=FetchType.LAZY)
    @JoinTable(name="departamento_curso",
        joinColumns = @JoinColumn(name="id_curso"),
        inverseJoinColumns = @JoinColumn(name="id_departamento"))
    private Collection<Departamento> departamentos;

    // Getters and Setters
}
```

Associações

❑ Muitos-para-muitos:

```
@Entity
@SequenceGenerator(name="SEQ_DEPARTAMENTO", initialValue=1,
allocationSize=1, sequenceName="seq_departamento")
public class Departamento implements Serializable{
    @Id
    @GeneratedValue(strategy=GenerationType.SEQUENCE, generator="SEQ_DEPARTAMENTO")
    @Column(name="id_departamento")
    private int id;
    private String nome;
    private String sigla;

    @ManyToMany(fetch=FetchType.LAZY)
    @JoinTable(name="departamento_curso",
        joinColumns = @JoinColumn(name="id_departamento"),
        inverseJoinColumns = @JoinColumn(name="id_curso"))
    private Collection<Curso> cursos;

    @ManyToOne
    @JoinColumn(name="id_centro")
    private Centro centro;
```

Associações

❑ Muitos-para-muitos:

```
@Entity
@SequenceGenerator(name="SEQ_CENTRO", initialValue=1,
allocationSize=1, sequenceName="seq_centro")
public class Centro implements Serializable{

    @Id
    @GeneratedValue(strategy=GenerationType.SEQUENCE, generator="SEQ_CENTRO")
    private int id;

    private String nome;

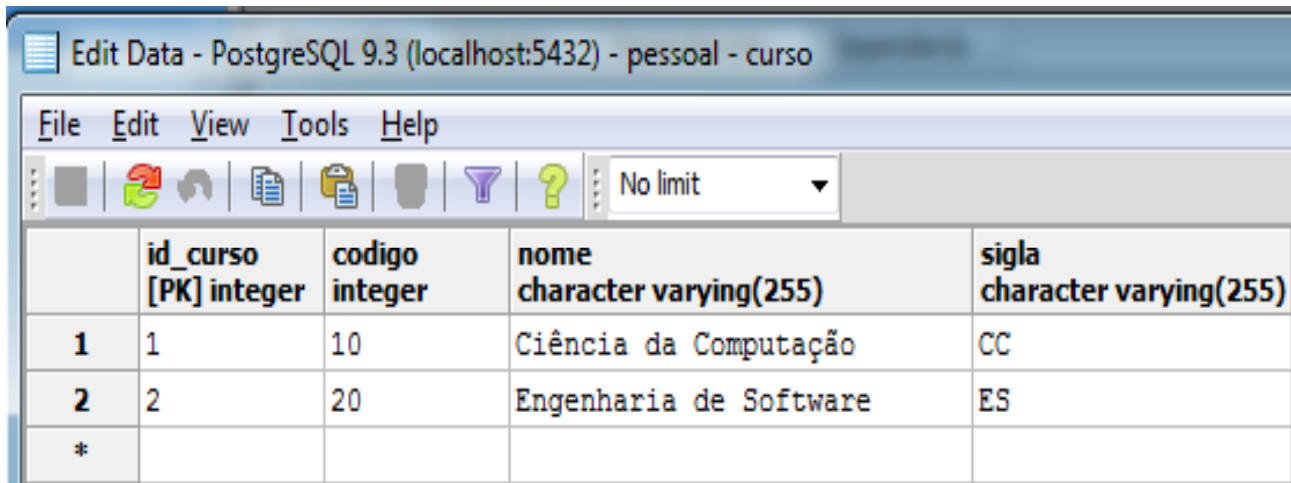
    @ManyToOne
    @JoinColumn(name="id_universidade")
    private Universidade universidade;

    @OneToMany(mappedBy="centro")
    private Collection<Departamento> departamentos;

    // Getters and Setters
```


Associações

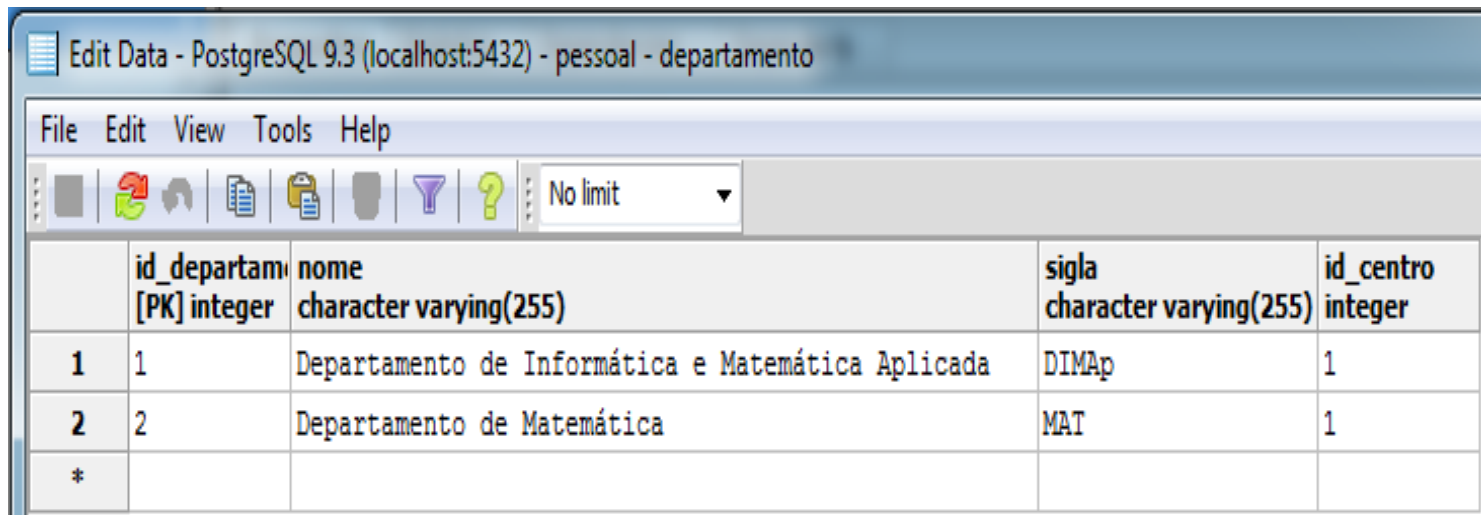
- ❑ Muitos-para-muitos:
 - ❖ Tabelas geradas: **Curso**



	id_curso [PK] integer	codigo integer	nome character varying(255)	sigla character varying(255)
1	1	10	Ciência da Computação	CC
2	2	20	Engenharia de Software	ES
*				

Associações

- ❑ Muitos-para-muitos:
 - ❖ Tabelas geradas: **Departamento**



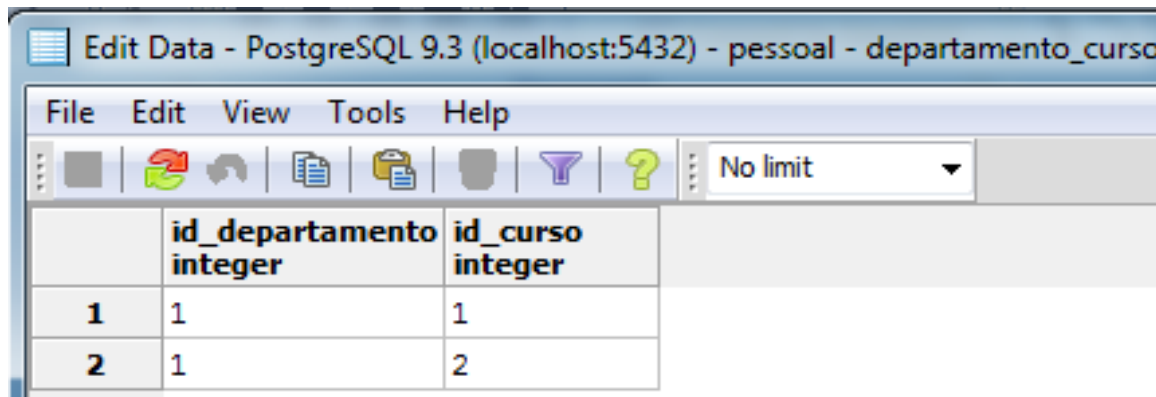
The screenshot shows a PostgreSQL data editor window titled "Edit Data - PostgreSQL 9.3 (localhost:5432) - pessoal - departamento". The window has a menu bar (File, Edit, View, Tools, Help) and a toolbar with various icons. A dropdown menu is set to "No limit". The table data is as follows:

	id_departam [PK] integer	nome character varying(255)	sigla character varying(255)	id_centro integer
1	1	Departamento de Informática e Matemática Aplicada	DIMAp	1
2	2	Departamento de Matemática	MAT	1
*				

Associações

❑ Muitos-para-muitos:

❖ Tabelas geradas: **Departamento_Curso**



	id_departamento integer	id_curso integer
1	1	1
2	1	2

Dúvidas...

