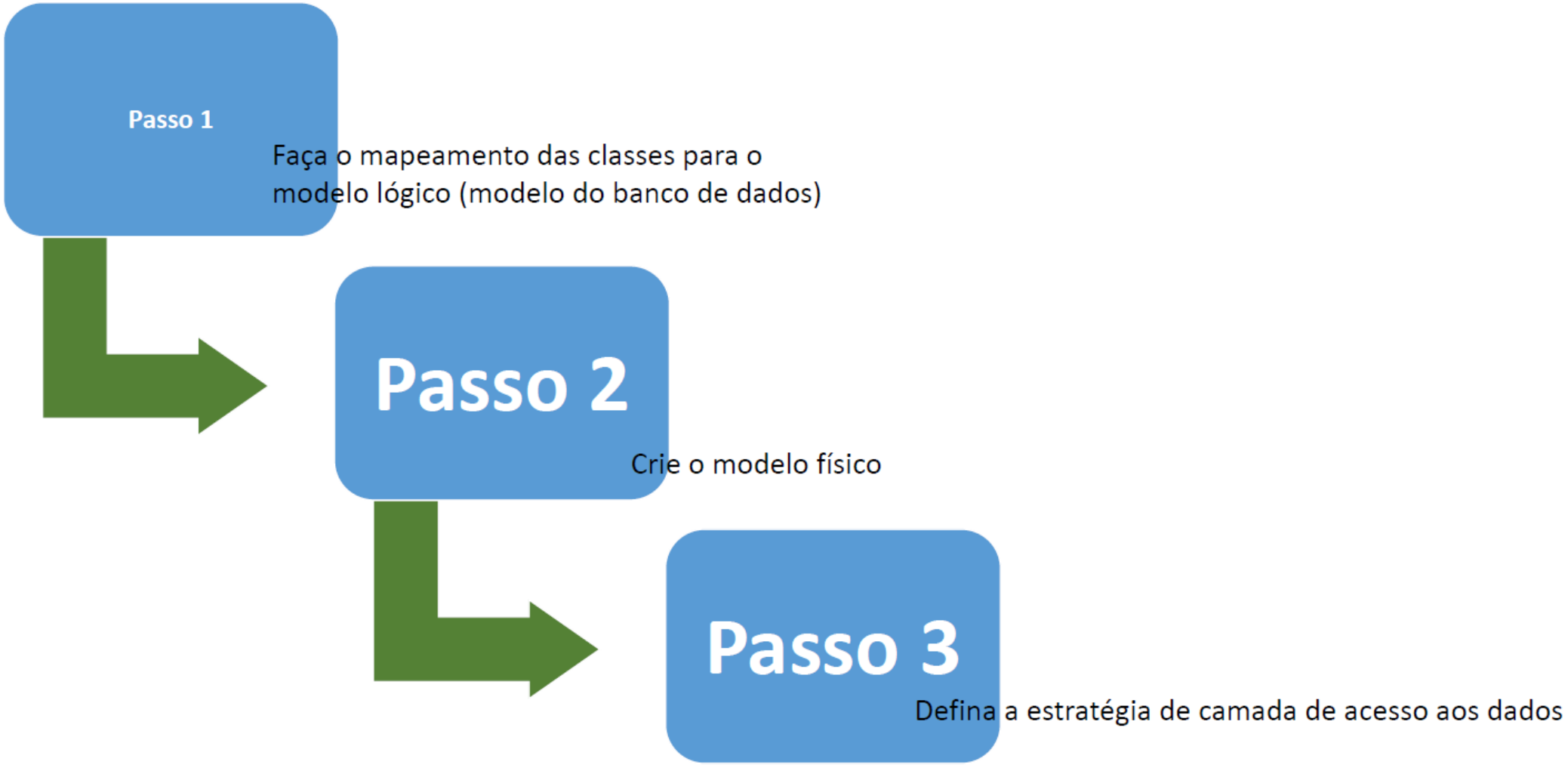
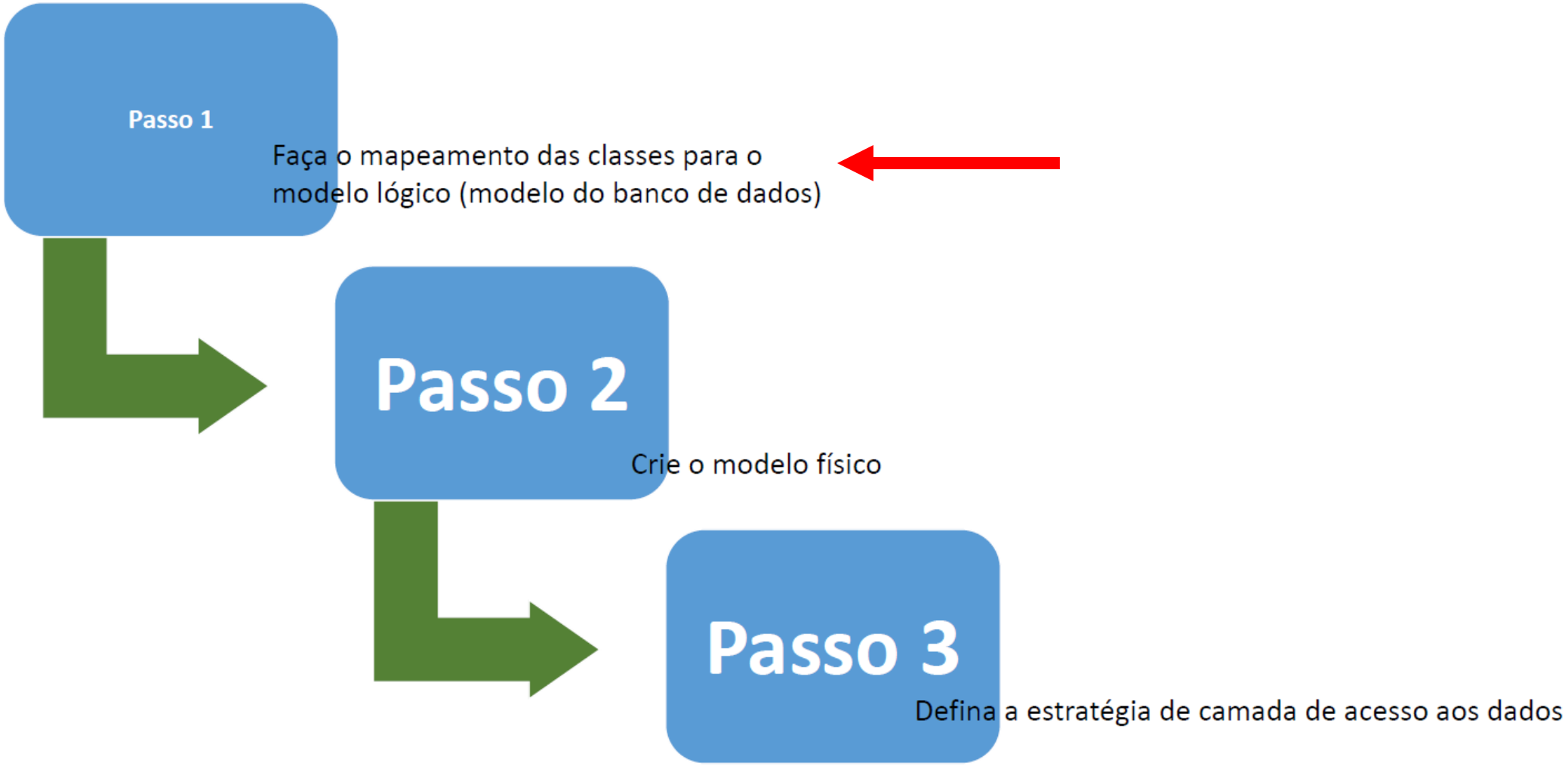
Mapeamento OO Para Banco de Dados Relacional

João Pedro Oliveira Batisteli

Passos para o design da camada de dados



Passos para o design da camada de dados



Relevância do Mapeamento OO-Relacional

Por que é importante?

- A tecnologia (paradigma) orientada a objetos (OO) é a abordagem padrão no desenvolvimento de sistemas modernos.
- Os SGBDs relacionais (SGBDR) ainda são a tecnologia dominante no mercado para armazenamento e gerenciamento de dados.

- **Surge o desafio de integração**: fazer aplicações OO conversarem com bancos relacionais de forma eficiente

Diferenças Conceituais

- **Paradigma OO**
  - Trabalha com **objetos** que combinam **dados + comportamentos** (métodos).
  - Representa abstrações do mundo real.
- **Modelo Relacional**
  - Baseado em **tabelas**, linhas e colunas.
  - Representa apenas **dados estáticos**, sem comportamento associado.

- Isso cria um **"mismatch"** (descompasso) entre os dois mundos, que o mapeamento OO-Relacional busca resolver.

Exemplo: Classe com Herança

Podemos ter as seguintes classes no nosso projeto:

```
class Pessoa {
    String nome;
}

class Cliente extends Pessoa {
    String endereco;
}
```

O banco de dados não entende “herança”. Ele trabalha apenas com tabelas. Então você tem que decidir como mapear:

- **Opção 1:** uma única tabela com todas as colunas **(PessoaCliente(id, nome, endereco)).**
- **Opção 2:** duas tabelas separadas, com chave estrangeira (**Pessoa(id, nome) e Cliente(id, endereco)** ligadas pelo mesmo **id**).

Mapeamento para Banco de Dados Relacional

- **Características do modelo relacional:**
  - Necessidade de **chaves** para identificar e relacionar dados.
  - **Chave Primária (PK):** identifica de forma única cada registro da tabela.
  - **Chave Estrangeira (FK):** cria vínculos entre tabelas, representando relacionamentos.

- **Observação:**
  - O modelo relacional utiliza uma **notação própria** (tabelas, colunas, restrições) que deve ser respeitada no processo de mapeamento.

Objetos Transientes

- **Definição:**
  - Existem apenas na **memória principal** durante a execução de uma sessão do sistema.
  - Deixam de existir quando a aplicação é encerrada.
- **Exemplos típicos:**
  - **Objetos de controle** → coordenam fluxos do sistema.
  - **Objetos de fronteira** → interagem com o usuário ou sistemas externos.

Objetos Persistentes

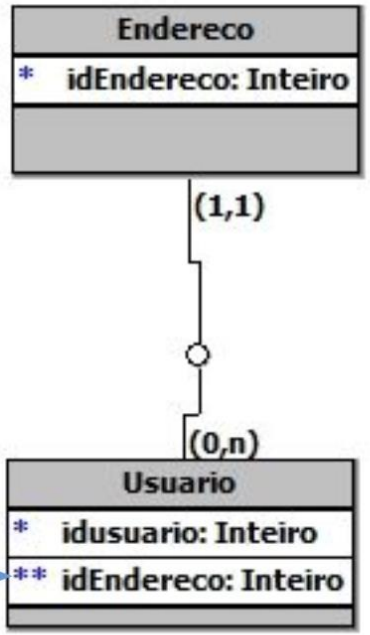
- **Definição:**
  - Continuam a existir mesmo após o término da sessão.
  - São armazenados em **bancos de dados** ou **arquivos** e restaurados quando o sistema inicia novamente.
- **Exemplo típico:**
  - **Objetos de entidade** → representam informações de negócio que precisam ser mantidas (ex.: Cliente, Produto, Pedido).

Problema da Granularidade

- **Granularidade:** nível de detalhe com que um elemento é representado.
- **Na orientação a objetos:**
  - Um objeto pode ser composto por outros objetos.
  - Ex.: classe Usuário pode conter um objeto do tipo Endereço.
- **No modelo relacional:**
  - Essa composição não existe diretamente.
  - Muitas vezes, os atributos do objeto “interno” viram apenas colunas em uma tabela.

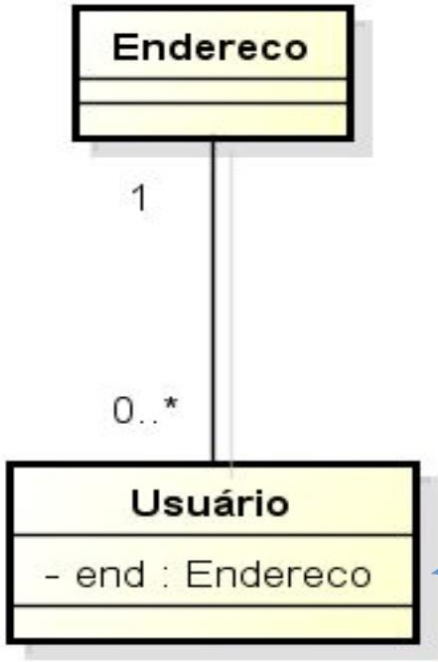
Problema da Granularidade – Exemplo

Relacional



Coluna chave estrangeira

OO



objeto

Problema dos Subtipos no Paradigma OO

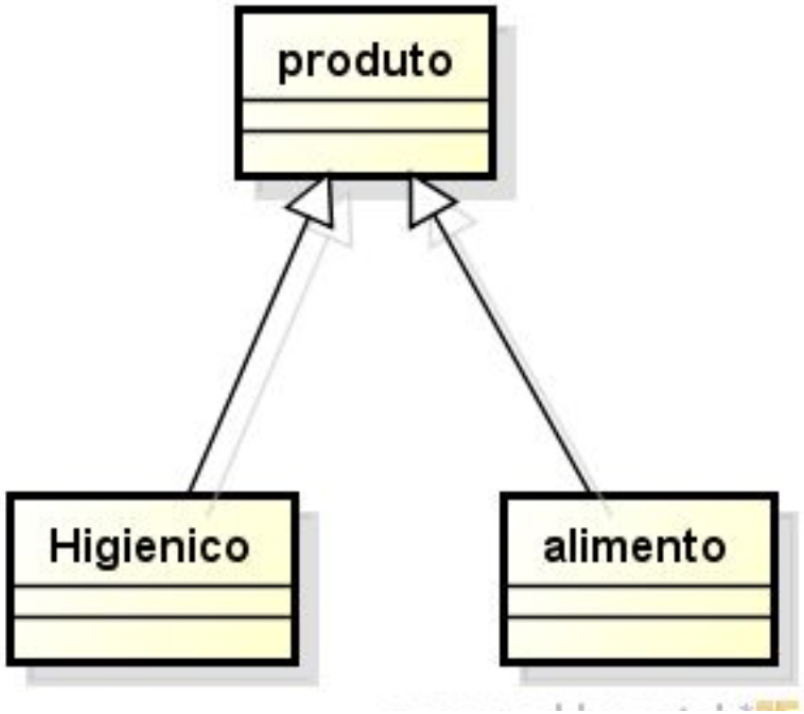
- **Herança e Polimorfismo** são pilares da orientação a objetos:
  - Subclasses podem **estender** uma superclasse.
  - Atributos e métodos da superclasse podem ser **herdados**.
  - Métodos podem ser **sobrescritos (override)** ou **sobrecaregados (overload)**.
  - Subclasses podem incluir **novos atributos e métodos**.
- Isso permite **reuso, flexibilidade e especialização** no modelo OO.

Problema dos Subtipos no Paradigma OO

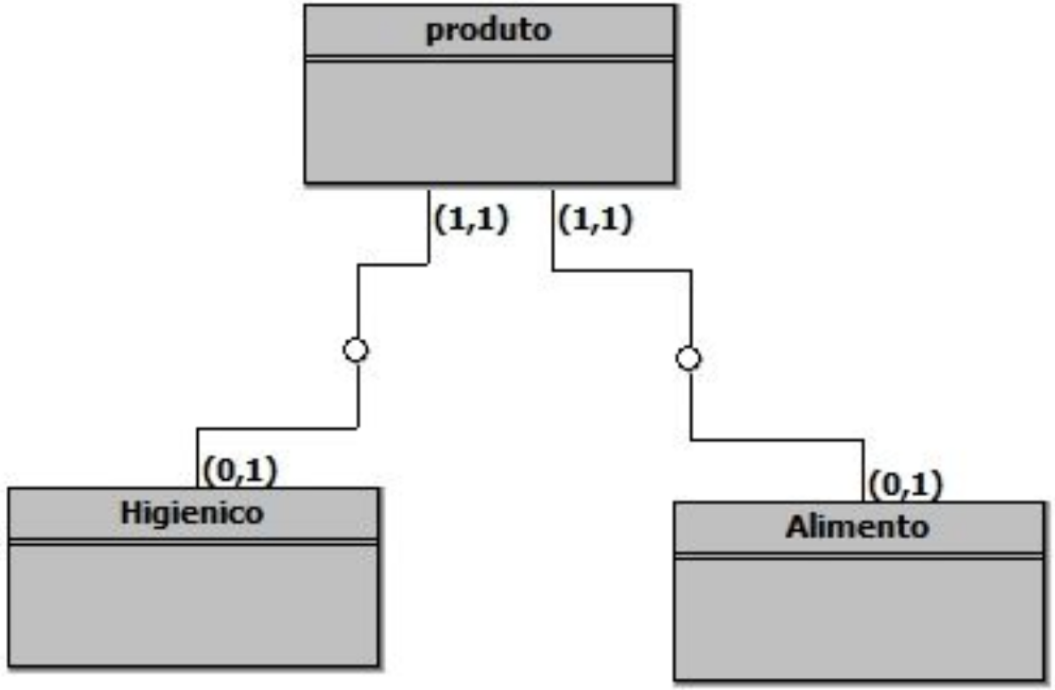
- **Modelo Relacional:**
  - Trabalha apenas com **tabelas, linhas e colunas**.
  - **Não possui suporte nativo para herança de tipos.**
  - Cada tabela é independente, ou seja, não há hierarquia natural como em classes OO.
- Resultado: é necessário **definir estratégias de mapeamento** para representar herança no banco de dados (ex.: tabela única, tabelas separadas, ou tabela por hierarquia).

Problema dos Subtipos no Paradigma OO - Exemplo

OO



Relacional: uma possibilidade



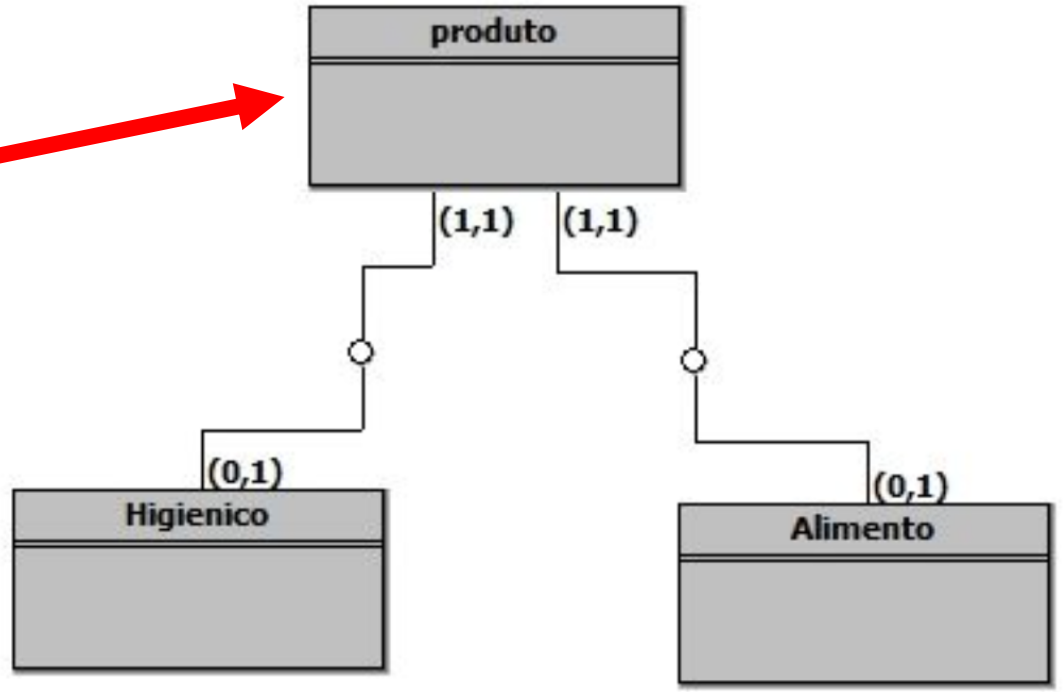
Problema dos Subtipos no Paradigma OO - Exemplo

As tabelas das subclasses (Higienico e Alimento) se conectam à tabela da superclasse (produto) por meio de uma chave estrangeira.

Essa relação é de um para um (1,1) para garantir que cada registro em uma tabela de subclasse corresponda exatamente a um registro na tabela da superclasse.

Assim, para obter todos os dados de um produto Higienico, você precisaria fazer um ***join*** (junção) entre a tabela produto e a tabela Higienico.

Relacional: uma possibilidade



Associações na Orientação a Objetos

- Associações são representadas como **referências diretas** ou **coleções de referências** entre objetos.
- Em associações **bidirecionais**, cada classe mantém uma referência para a outra.
- Essas associações podem ser:
  - **1 para 1** (um objeto se relaciona com apenas um outro).
  - **1 para N** (um objeto se relaciona com vários outros).
  - **N para M** (coleções em ambos os lados).

Associações no Modelo Relacional

- Associações são representadas por **chaves estrangeiras (FKs)**.
- Relacionamentos não são direcionais (as consultas definem o caminho).
- É possível criar **associações arbitrárias** via junções (JOIN) e projeções.
- Para **relacionamentos N:M (muitos para muitos)**, é necessário criar uma **tabela associativa** (tabela extra que contém as chaves das duas entidades).

Associações no Modelo Relacional

- OO → associações são **naturais**, como referências entre objetos.

- Relacional → associações precisam ser **modeladas explicitamente**, muitas vezes com tabelas extras.

Navegação em Orientação a Objetos

- O acesso entre objetos é feito por meio de **referências diretas**.
- A navegação ocorre chamando **métodos de acesso** (getters ou operações específicas).
  - Ex.: pedido.getCliente().getEndereco()
- O percurso entre objetos é **natural e encadeado**, refletindo os relacionamentos do modelo de classes.

Navegação em Bancos de Dados Relacionais

- Os dados são armazenados em **tabelas independentes**.
- Para acessar dados relacionados, é necessário realizar **consultas SQL** com junções (JOIN).
  - Ex.: *SELECT * FROM Pedido p JOIN Cliente c ON p.idCliente = c.id*
- A navegação depende de **operações de consulta** e não de referências diretas.
- O percurso é **baseado em relacionamentos via chaves** (primárias e estrangeiras).

Necessidade do Mapeamento para SGBDR

- Para utilizar um **SGBDR**, é preciso mapear os **atributos de objetos persistentes** para tabelas relacionais.

- O mapeamento parte do **modelo de classes** da aplicação:
  - Semelhante ao mapeamento feito a partir do **Modelo Entidade-Relacionamento (MER)**.

- Porém, o **modelo de classes possui recursos adicionais** (herança, polimorfismo, métodos, agregações).

Modelo de Classes vs MER

- **Modelo de Classes:**
  - Representa **objetos**, incluindo **dados + comportamento**.
  - Utilizado principalmente para modelar a aplicação OO.
- **Modelo Entidade-Relacionamento (MER):**
  - Representa apenas **dados** e seus relacionamentos.
  - Frequentemente confundido com o modelo de classes, mas **não é equivalente**.
- **Importante:**
  - O mapeamento OO–Relacional deve considerar essas diferenças para que a persistência funcione corretamente.

Notação Utilizada (simplificada)

- Cada **relação (tabela)** é representada pelo seu **nome** seguido dos nomes das colunas entre parênteses.
- **Chaves primárias (PK):** representadas **sublinhadas**.
- **Chaves estrangeiras (FK):** representadas **tracejadas**.

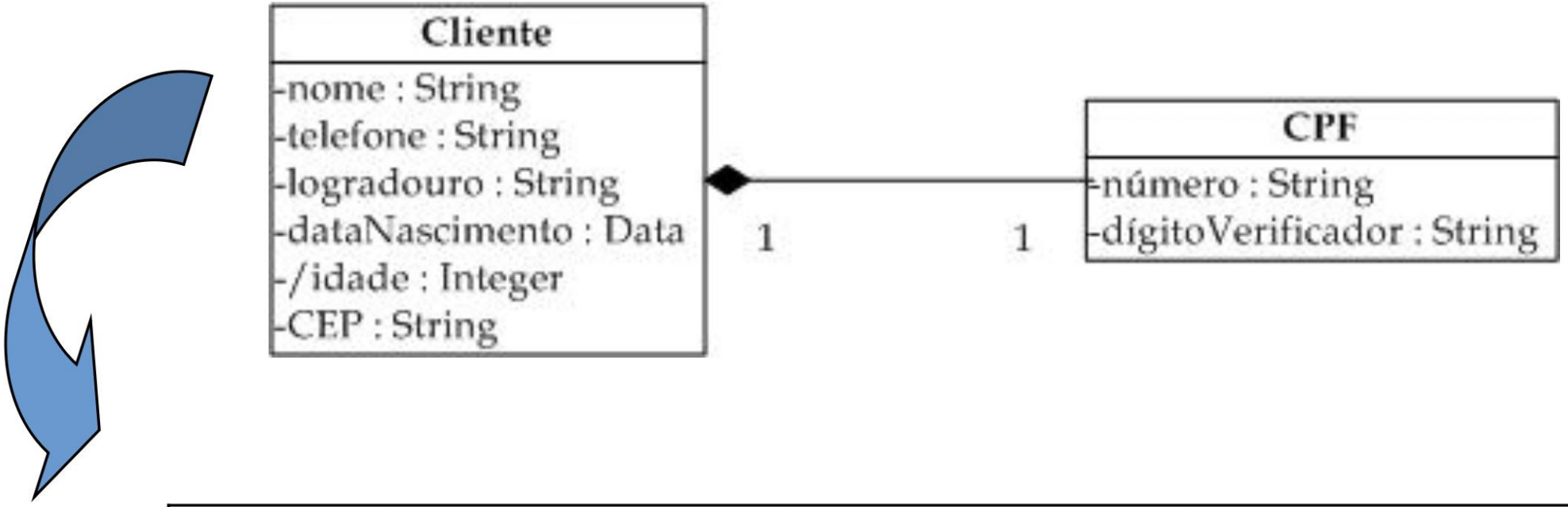
Chaves nos Exemplos

- Em todos os exemplos, a **chave primária** é uma **coluna de implementação**.
  - Identificador sem significado direto no domínio de negócio.
- Essa abordagem é usada porque:
  - Garante **padronização** nos exemplos.
  - É considerada uma das **melhores práticas** para associar identificadores a objetos mapeados em tabelas.

Exemplo

- **Departamento(*sigla*, nome, matriculaGerente)**
  - sigla → **PK** (sublinhada)
  - matriculaGerente → **FK** (tracejada) que referencia Empregado(matricula)
- **Empregado(*matricula*, CPF, nome, endereço, CEP, siglaDepartamento)**
  - matricula → **PK** (sublinhada)
  - siglaDepartamento → **FK** (tracejada) que referencia Departamento(sigla)
- **Projeto(*idProjeto*, nome, verba)**
  - idProjeto → **PK** (sublinhada)
- **Alocacao(*matricula*, *idProjeto*)**
  - matricula → **FK** (tracejada) que referencia Empregado(matricula)
  - idProjeto → **FK** (tracejada) que referencia Projeto(idProjeto)
  - **PK composta**: (matricula, idProjeto)

Mapeando Classes e Seus Atributos



Cliente(<u>id</u>, CPF, nome, telefone, logradouro, dataNascimento, idCEP) CEP(<u>id</u>, número, sufixo)
Cliente(<u>id</u>, nome, telefone, logradouro, dataNascimento, CPF, CEP)

Mapeamento de Associações

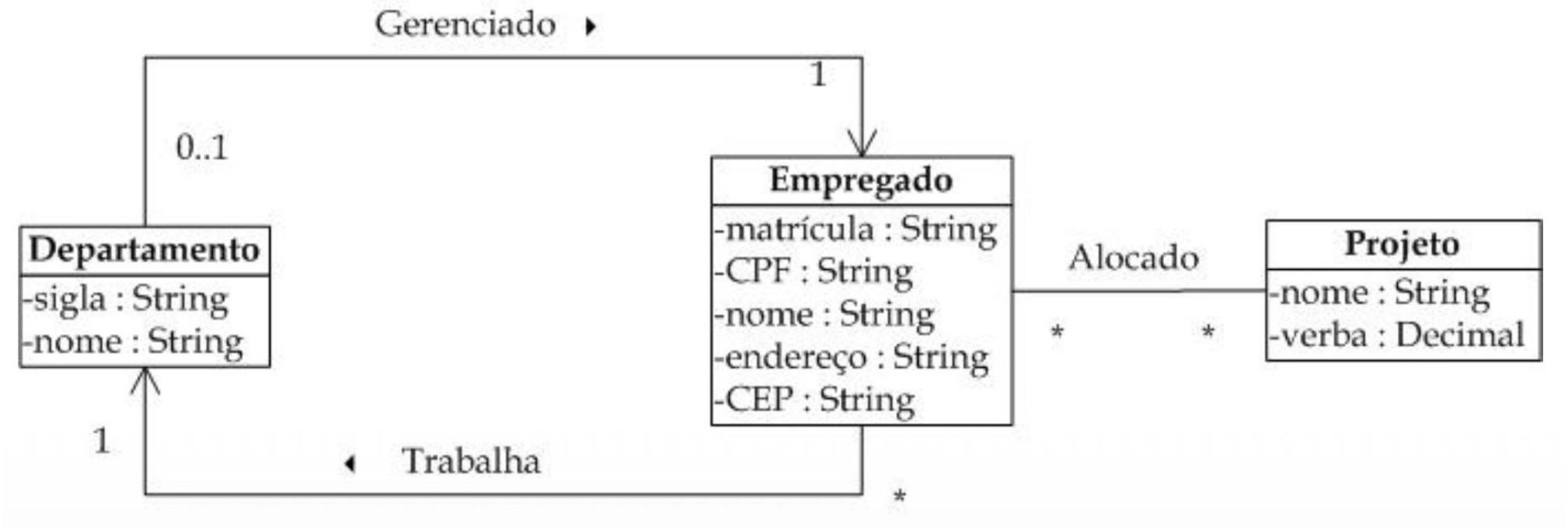
- O **procedimento de mapeamento** utiliza o conceito de **chave estrangeira (FK)**.
- Existem **três casos principais**, cada um associado a um tipo de **conectividade entre classes**:
  - **1:1 (um para um)**
  - **1:N (um para muitos)**
  - **N:M (muitos para muitos)**

Mapeamento de Associações

- Para simplificar, considere que temos duas classes:
  - **Ca** e **Cb**
- Ambas foram mapeadas para **relações separadas**:
  - **Ca → Ta**
  - **Cb → Tb**
- A associação entre **Ca** e **Cb** será refletida por meio de **chaves estrangeiras** entre **Ta** e **Tb**, conforme o tipo de conectividade.

Mapeamento de Associações

- Considere o seguinte diagrama de classes:



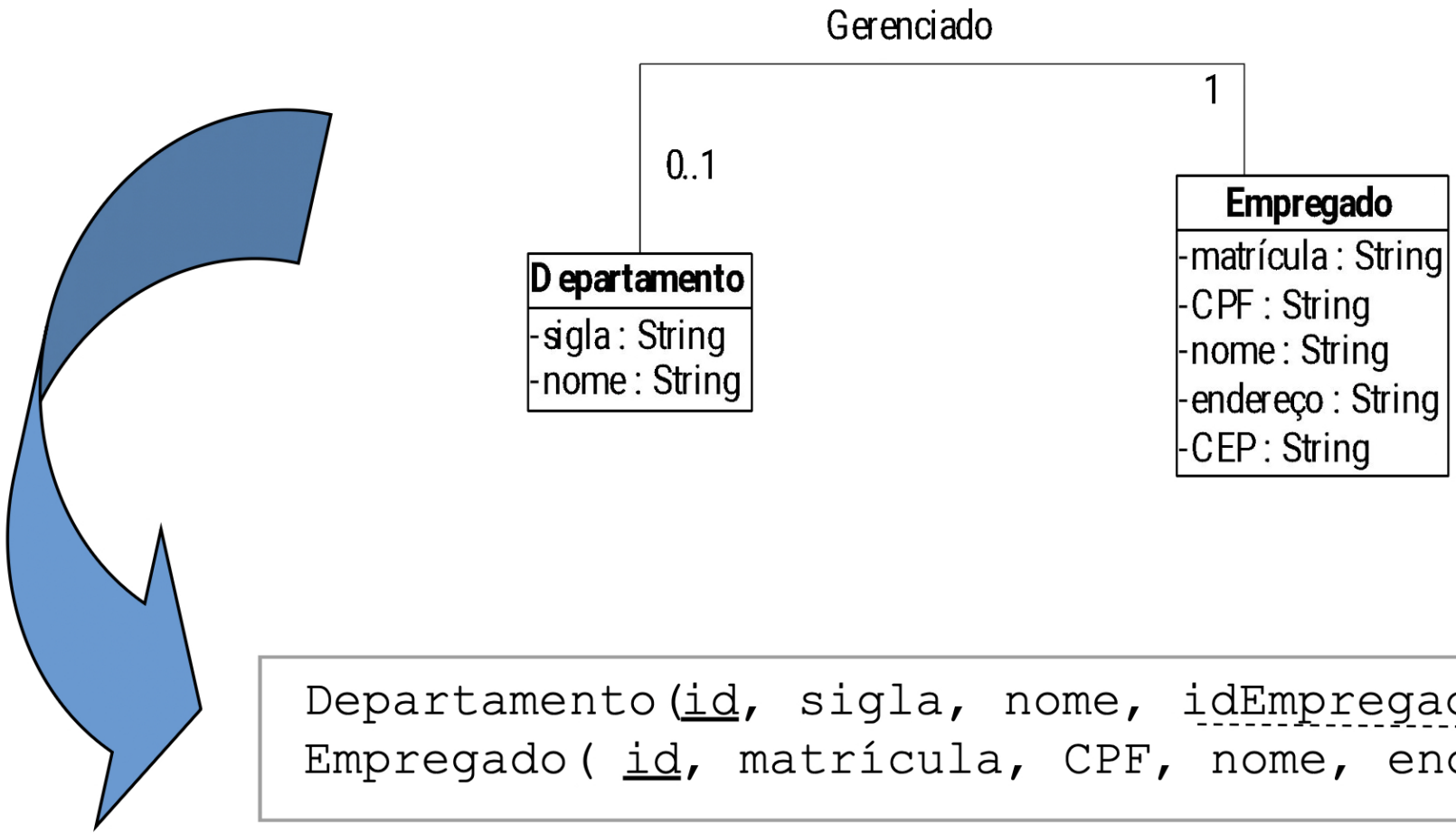
Mapeamento de Associações

- Para mapear uma associação entre classes, é necessário adicionar uma **chave estrangeira (FK)** em uma das tabelas.
- A escolha da tabela que receberá a FK depende da **participação e conectividade** da associação:
  - **Obrigatória em ambos os lados** → FK pode ser colocada em qualquer lado.
  - **Opcional em ambos os lados** → pode ser necessário criar tabela intermediária.
  - **Obrigatória em um lado e opcional no outro** → FK deve ir para o lado **opcional**.

Mapeamento de Associações

- No relacionamento **Trabalha**:
  - Cada **Empregado** pertence a **um único Departamento** (participação obrigatória).
  - Um **Departamento** pode ter **vários empregados** (participação múltipla).
  - **Solução:** adicionar a **FK siglaDepartamento** em **Empregado**.
- No relacionamento **Gerenciado**:
  - Um **Departamento** pode ser gerenciado por **0 ou 1 Empregado** (participação opcional).
  - Um **Empregado** gerencia **no máximo 1 Departamento**.
  - **Solução:** adicionar a **FK matriculaGerente** em **Departamento**.
- No relacionamento **Alocado** (N:N):
  - Um **Empregado** pode estar em vários **Projetos**, e um **Projeto** pode ter vários **Empregados**.
  - **Solução:** criar uma **tabela associativa** (ex.: Alocacao) com as FKs de **Empregado** e **Projeto**

Mapeamento de Associações




```
Departamento(id, sigla, nome, idEmpregadoGerente )
Empregado( id, matrícula, CPF, nome, endereço, CEP )
```

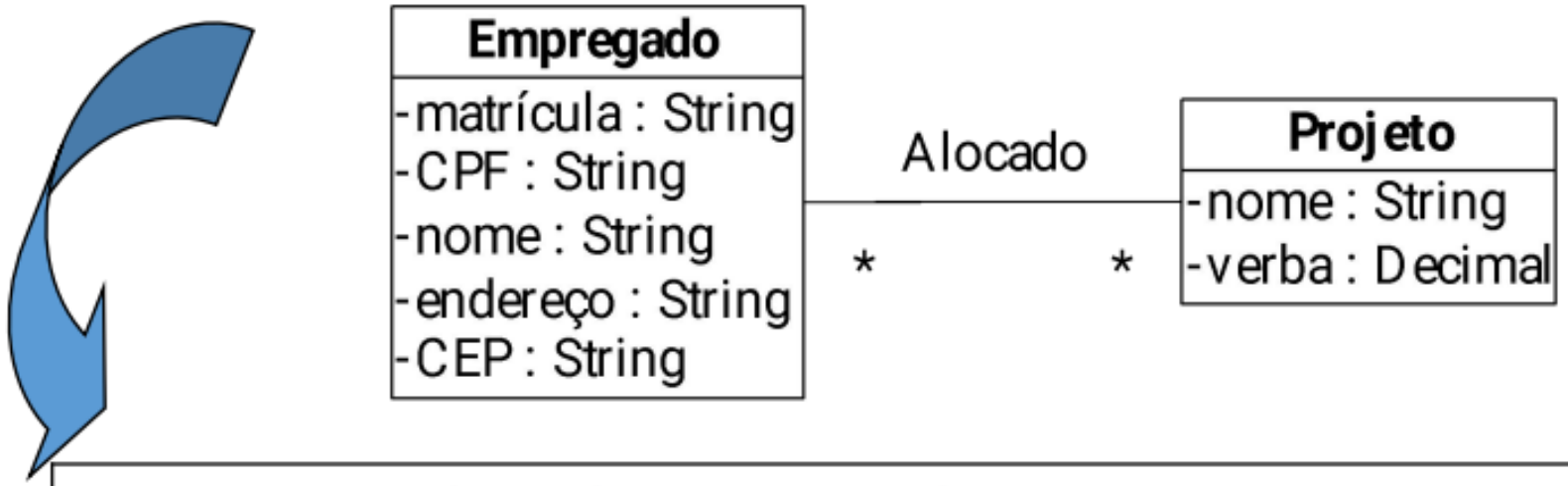
Relação de Associação (N:N)

- Quando uma classe **Ca** se associa com muitos objetos de outra classe **Cb** temos uma relação **N:N**.
- No modelo relacional, criamos uma **tabela de associação** para representar esse vínculo.
- Essa tabela contém as **chaves estrangeiras (FKs)** que apontam para as tabelas resultantes de Ca (Ta) e Cb (Tb).
- Também pode armazenar **atributos próprios da associação**, caso existam.

Relação de Associação (N:N)

- A chave primária da tabela de associação pode ser definida de duas formas:
  1. **Chave composta** → formada pelas FKs que vêm de **Ta** e **Tb**.
  2. **Chave simples** → criada por meio de uma **coluna de implementação** (ex.: idAssoc).
- Assim, a tabela de associação terá três tipos possíveis de colunas:
  - **PK (simples ou composta)**
  - **FKs** para as tabelas associadas
  - **Atributos da associação** (quando existirem)

Mapeamento de Associações




```
Departamento(id, sigla, nome, idEmpregadoGerente)
Empregado(id, matrícula, CPF, nome, endereço, CEP, idDepartamento)
Alocação(idProjeto, idEmpregado, nome, verba)
Projeto(id, nome, verba)
```

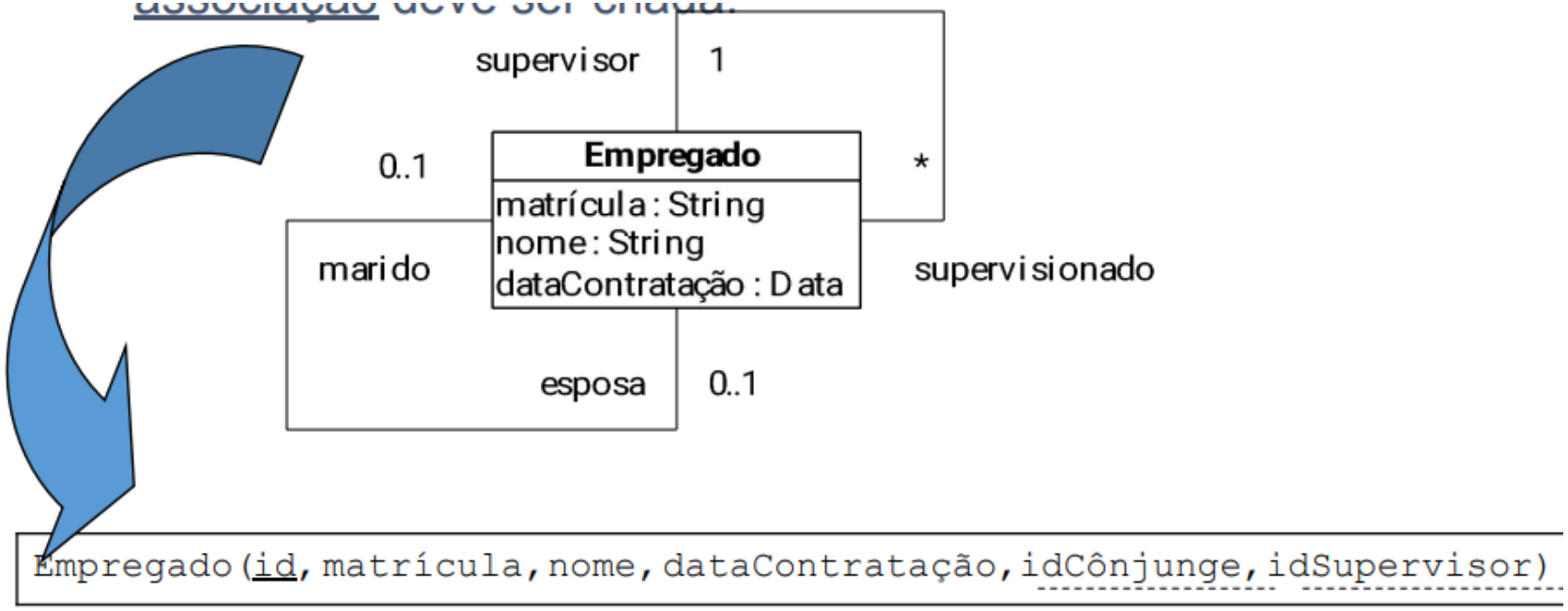
```
Departamento(id, sigla, nome, idEmpregadoGerente)
Empregado(id, matrícula, CPF, nome, endereço, CEP, idDepartamento)
Alocação(id, idProjeto, idEmpregado, nome, verba)
Projeto(id, nome, verba)
```

Problema da Granularidade

- **Granularidade:** nível de detalhe com que um elemento é representado.
- **Na orientação a objetos:**
  - Um objeto pode ser composto por outros objetos.
  - Ex.: classe Usuário pode conter um objeto do tipo Endereço.
- **No modelo relacional:**
  - Essa composição não existe diretamente.
  - Muitas vezes, os atributos do objeto “interno” viram apenas colunas em uma tabela.

Mapeamento de associações reflexivas

- O mapeamento de **associações reflexivas** é uma forma específica de lidar com relações entre entidades que ocorrem dentro da mesma tabela ou classe.
- Embora possa parecer complexo, o procedimento de mapeamento é o mesmo usado para associações normais.
- Em particular, em uma associação reflexiva de conectividade muitos para muitos, uma relação de associação deve ser criada.



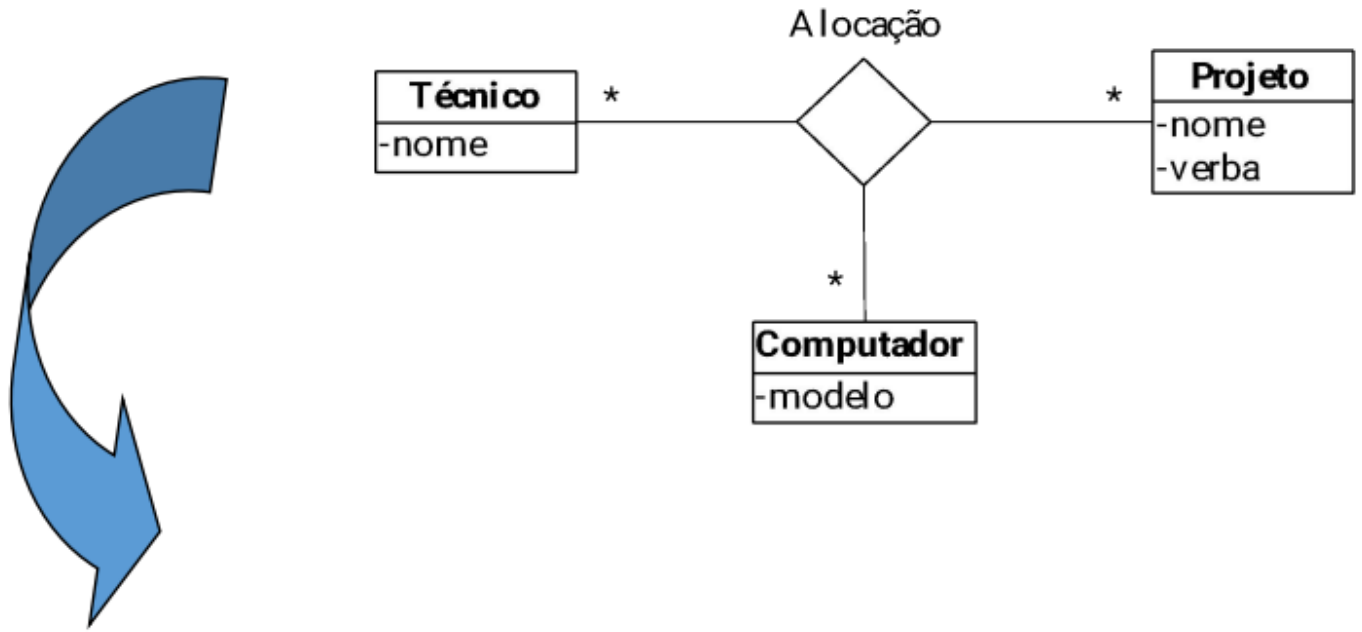
Associações N-árias (n ≥ 3)

- São associações que envolvem **três ou mais classes**.
- O mapeamento segue a mesma lógica das associações binárias **N:N**, mas agora com mais participantes.
- Exemplo: um **Empregado** pode usar um **Equipamento** em um **Projeto**.

Associações N-árias (n ≥ 3)

- Criamos uma **tabela de associação** para representar o relacionamento.
- Essa tabela deve conter:
  - **Chaves estrangeiras (FKs)** para cada tabela participante.
  - Uma **chave primária** → pode ser composta pelas FKs ou criada como coluna de implementação.
  - (Opcional) **Atributos próprios da associação**, se existirem (como data de uso, horas trabalhadas etc.).

Associações N-árias (n ≥ 3)




```
Técnico( id, nome )
Projeto( id, nome, verba )
Computador( id,  modelo )
Alocação( id, idProjeto, idTécnico, idComputador )
```

Mapeamento de classes associativas

- Uma **classe associativa** é usada quando uma associação possui **atributos próprios**.
- Exmemplo: associação **Empregado–Projeto** pode ter um atributo **horasTrabalhadas**.
- Nesse caso, a associação deixa de ser apenas um vínculo e passa a ser um **objeto com informações adicionais**.

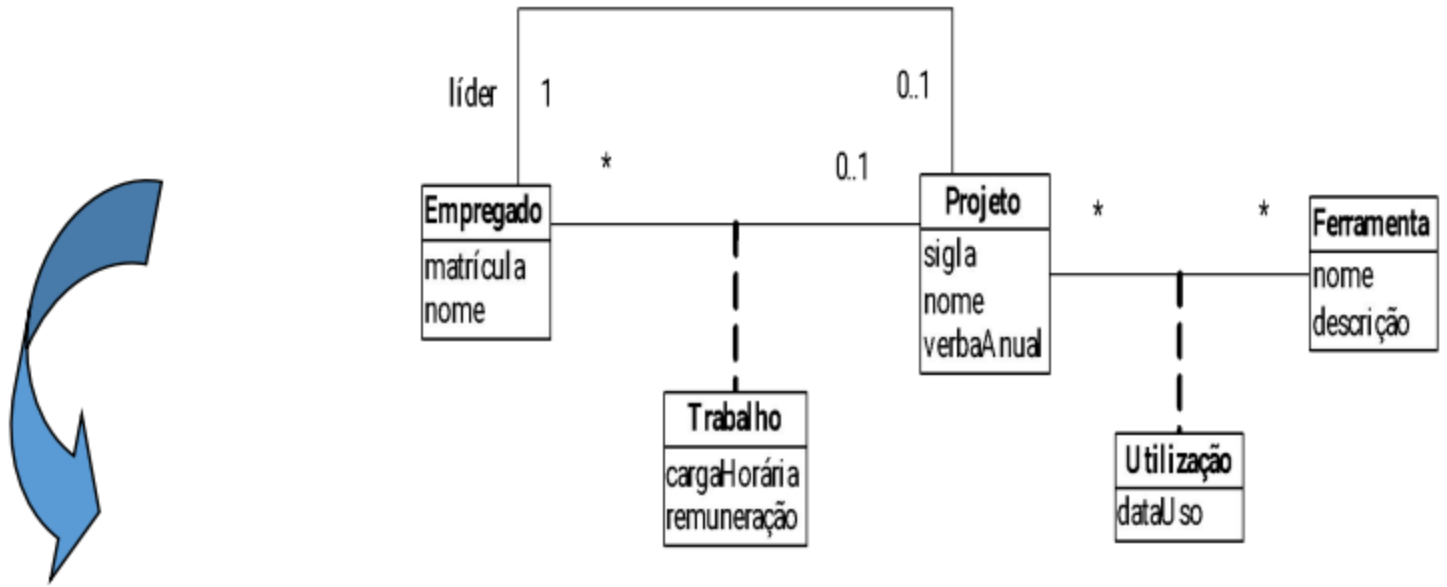
Mapeamento de classes associativas

- O mapeamento é feito criando-se uma **tabela específica** para a associação.
- Essa tabela deve conter:
  - **Chaves estrangeiras (FKs)** → para as classes participantes.
  - **Atributos da associação** → mapeados como colunas normais.
  - **Chave primária** → pode ser composta pelas FKs ou uma coluna de implementação.

Mapeamento de classes associativas

- O mapeamento é feito criando-se uma **tabela específica** para a associação.
- Essa tabela deve conter:
  - **Chaves estrangeiras (FKs)** → para as classes participantes.
  - **Atributos da associação** → mapeados como colunas normais.
  - **Chave primária** → pode ser composta pelas FKs ou uma coluna de implementação.

Mapeamento de classes associativas




```
Empregado(id, matrícula, nome)
Projeto(id, sigla, nome, verbaAnual, idEmpregadoLíder)
Ferramenta(id, nome, descrição)
Utilização(id, idFerramenta, idProjeto, dataUso )
Trabalho(id, idEmpregado, idProjeto, cargaHorária, remuneração)
```

Alternativas de Mapeamento de Herança/Generalizações

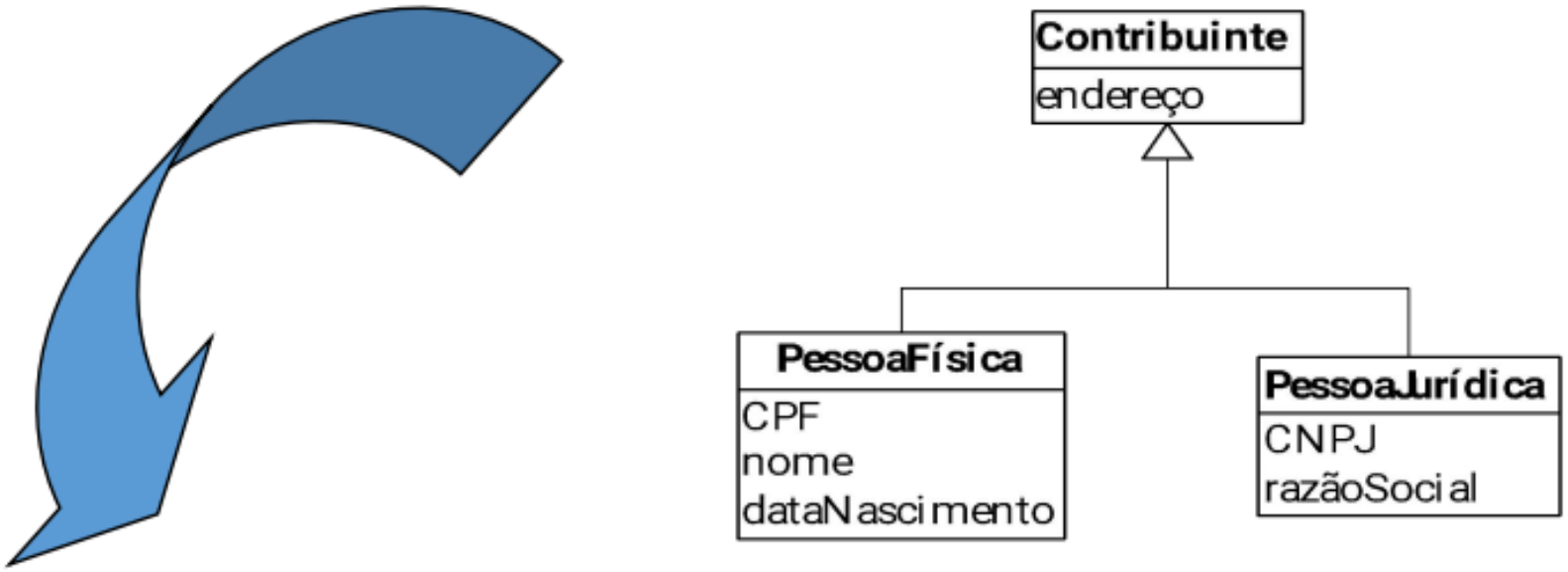
- Existem três formas principais de mapear **generalizações (herança)** no modelo relacional:

1. **Uma tabela por classe da hierarquia**
   - Cada classe vira uma tabela própria.
2. **Uma tabela para toda a hierarquia**
   - Uma única tabela com colunas para atributos de todas as classes.
3. **Uma tabela por classe concreta**
   - Apenas classes "finais" (sem subclasses) viram tabelas.

Qual Alternativa Escolher?

- **Não existe uma solução única**: cada abordagem tem **vantagens e limitações**.
- A escolha depende de fatores como:
  - Requisitos de desempenho.
  - Complexidade da hierarquia.
  - Frequência de consultas e atualizações.
- A equipe pode inclusive **combinar mais de uma estratégia** no mesmo sistema, conforme a necessidade.

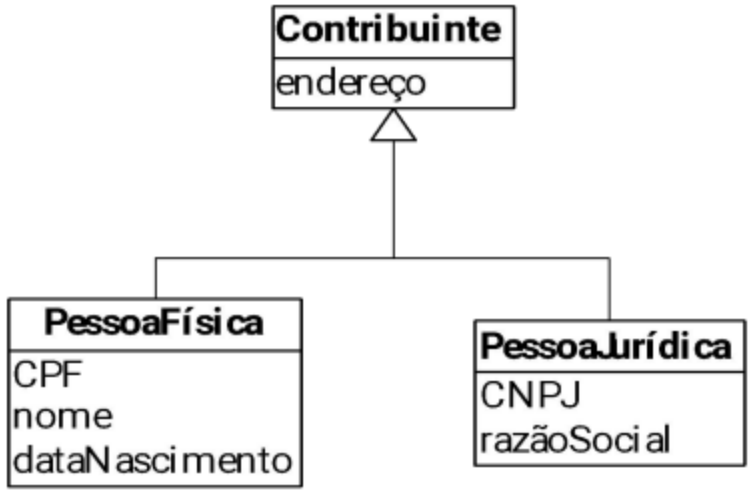
Alternativas de Mapeamento de Herança/Generalizações




```
Contribuinte(id, endereço)
PessoaFísica(id, nome, dataNascimento, CPF, idContribuinte)
PessoaJurídica(id, CNPJ, razãoSocial, idContribuinte)
```

1ª Alternativa

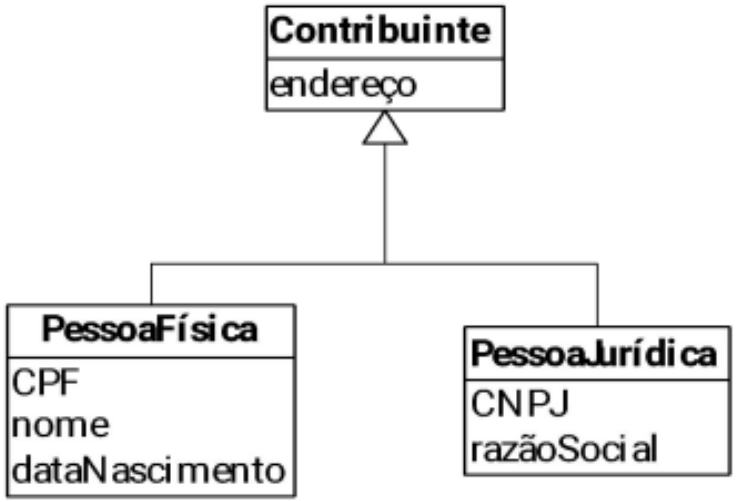
- É a que melhor reflete o modelo OO
- **Desvantagem**: desempenho da manipulação das relações (Inserções e remoções e junções).




```
Contribuinte(id, endereço)
PessoaFísica(id, nome, dataNascimento, CPF, idContribuinte)
PessoaJurídica(id, CNPJ, razãoSocial, idContribuinte)
```

2ª Alternativa

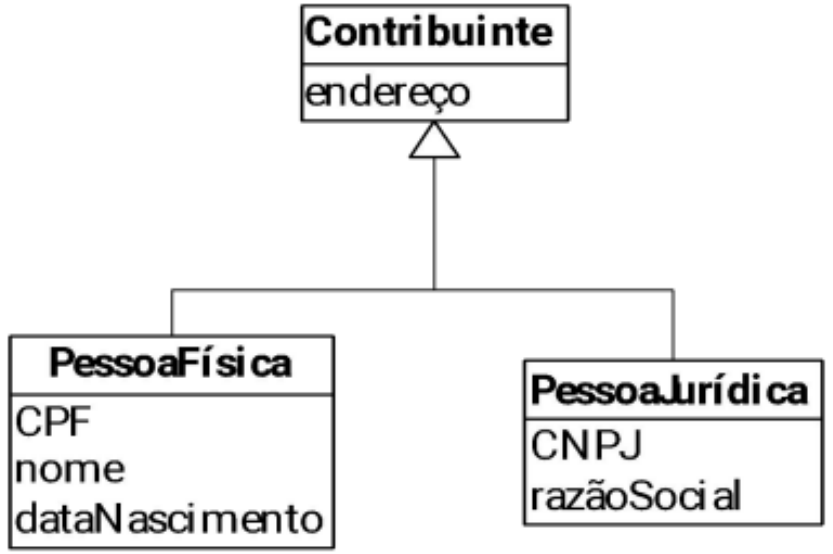
- **Desvantagem**: desempenho de adição ou remoção de atributos, além de ter o potencial de desperdiçar **bastante** espaço de armazenamento.




```
Pessoa(id, nome, endereço, dataNascimento, CPF, CNPJ, razãoSocial, tipo)
```

3ª Alternativa

- Apresenta a vantagem de agrupar os objetos de uma classe em uma única relação.
- **Desvantagem**: todas as relações correspondentes a subclasses devem ser modificadas quando a definição da superclasse é modificada.




```
PessoaFísica(id, dataNascimento, nome, endereço, CPF)
PessoaJurídica(id, CNPJ, endereço, razãoSocial)
```

Exercício 1

Você deve modelar as classes e depois propor o mapeamento relacional.

- Classes principais:
  - **Pessoa** (atributos: idPessoa, nome, email)
  - **Aluno** (atributos: matrícula, curso)
  - **Professor** (atributos: siape, departamento)
  - **Livro** (atributos: isbn, título, autor)
  - **Empréstimo** (atributos: dataEmprestimo, dataDevolucaoPrevista)
- Regras:
  - Pessoa é uma **superclasse** de Aluno e Professor.
  - Um **Aluno ou Professor** pode pegar **vários livros emprestados**.
  - Um **Livro** pode estar emprestado para **vários usuários diferentes ao longo do tempo**.
  - A associação **Empréstimo** possui atributos próprios (datas).
- Modele as classes com herança e associações.
- Realize o mapeamento para tabelas no modelo relacional, indicando PKs e FKs.

Exercício 2

Você deve modelar as classes e depois propor o mapeamento relacional.

- Classes principais:
  - **Usuário** (atributos: idUsuario, nome, email)
  - **Curso** (atributos: idCurso, título, cargaHoraria)
  - **Instrutor** (atributos: especialidade)
  - **Aluno** (atributos: dataMatricula)
  - **Atividade** (atributos: idAtividade, título, tipo, notaMaxima)
  - **Entrega** (atributos: dataEntrega, notaObtida)
- Regras:
  - Usuário é uma **superclasse** de Aluno e Instrutor.
  - Um **Instrutor** pode ministrar **vários Cursos**.
  - Um **Aluno** pode se matricular em **vários Cursos** (associação N:N com classe associativa: Matricula, contendo a data da matrícula).
  - Cada Curso possui **várias Atividades**.
  - Um **Aluno** pode fazer várias Entregas em Atividades.
- Modele as classes com herança e associações.
- Faça o mapeamento para tabelas relacionais, mostrando como tratar a associação N:N com atributos.