

Programação Orientada a Objetos

UML

PROFESSOR:

Demetrios Coutinho

Demetrios.coutinho@ifrn.edu.br

AGENDA

- Histórico da UML
- Diagrama de classes
- Representação de classes
 - Atributos e métodos
 - Tipos de acesso e modificadores
- Relacionamentos entre classes
 - Herança, Implementação, Associação, Agregação e Composição

UML

- UML (Linguagem de Modelagem Unificada) é uma linguagem visual
 - **Análise e projeto** de sistemas computacionais no paradigma de **Orientação a Objetos**
- Nos últimos anos, a UML se tornou a linguagem padrão de projeto de software, adotada internacionalmente pela indústria de Engenharia de Software

UML

- UML não é uma linguagem de programação
- É uma linguagem de modelagem, utilizada para representar o sistema de software sob os seguintes aspectos:
 - Requisitos
 - Comportamento
 - Estrutura lógica
 - Dinâmica de processos
 - Comunicação/Interface com os usuários

UML

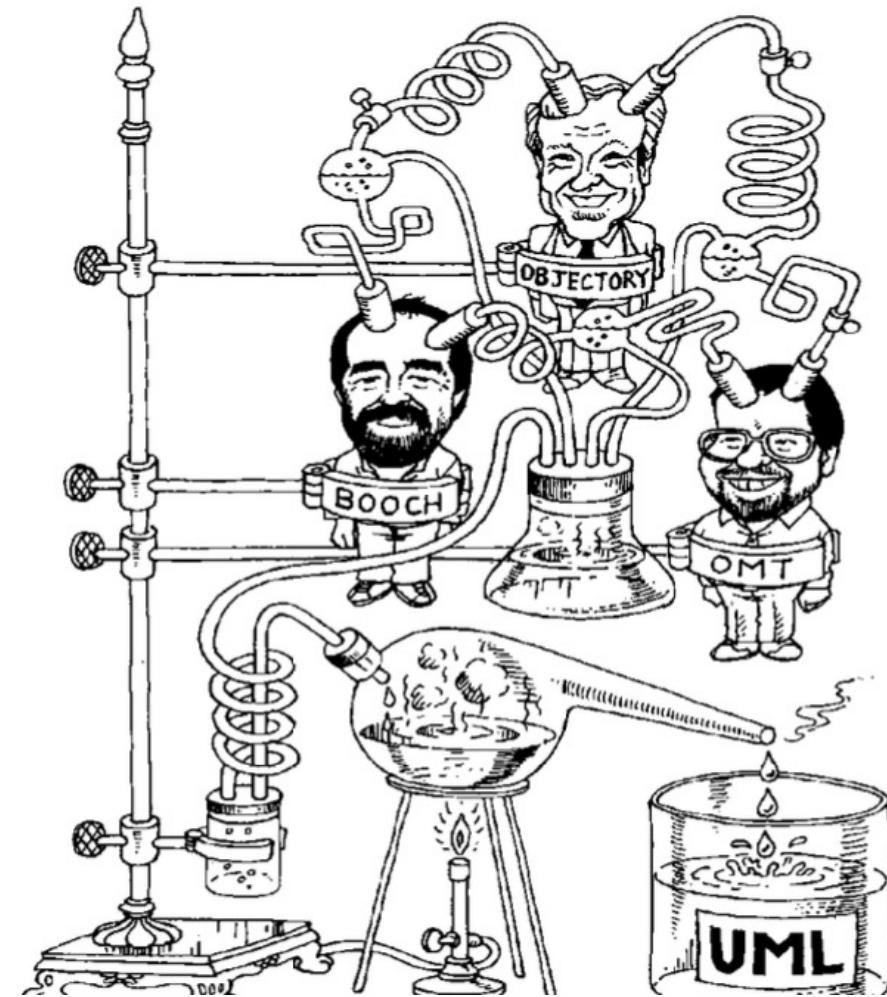
- Por que modelar um sistema?
 - Um sistema computacional é, de modo geral, excessivamente complexo
 - Necessário decompô-lo em pedaços comprehensíveis
 - Criação de **diagramas** auxiliam no entendimento do problema
 - **Linguagem única** que permite a todos os desenvolvedores entender quais objetos fazem parte do sistema e como eles se comunicam

UML

- A UML surgiu da união de outras três linguagens de modelagem:
 - O método de Booch (**Grady Booch**, **Rational Software Corporation**)
 - O método OMT (Object Modeling Technique, **Ivar Jacobson**, **Objectory**)
 - Método OOSE (Object-Oriented Software Engineering, **James Rumbaugh**, **General Eletrics**)
- Até meados da década de 90, estas eram as três linguagens de modelagem mais populares entre os profissionais de ES.

UML

- Em meados da década de 90, os criadores destas três linguagens se reuniram para criar uma **linguagem unificada**, mais concreta e madura



OBJETIVOS DA UML

- O objetivo da UML é fornecer múltiplas visões do sistema que se deseja modelar
- Estas várias visões são representadas pelos diferentes diagramas UML
- Cada diagrama analisa o sistema sob um determinado aspecto
 - É possível ter enfoques mais amplos (externos) ou mais específicos (internos)

VANTAGENS DA UML

- Perdas
 - Maior trabalho na modelagem
 - Mais tempo gasto
- Ganhos
 - Menos trabalho na construção (implementação)
 - A solução está pronta
 - Menos tempo gasto
 - Os problemas são encontrados em tempo hábil para sua solução
 - As dúvidas são sanadas mais cedo e são levantadas em sua totalidade



DIAGRAMAS DA UML

- **Diagrama de Classes**
 - Diagrama de Objetos
 - Diagrama de Componentes
 - Diagrama de Casos de usos
 - Diagrama de Sequências
 - Diagrama de Colaboração
 - Diagrama de Estado
 - Diagrama de Atividades
- 
- Diagramas de Estruturas
- Diagramas de Comportamento

Diagrama de Classes

- O diagrama de classes é um dos mais importantes e mais utilizados da UML
- Representação das principais classes
 - Atributos e Métodos
- Relacionamento entre as classes
- Uma visão **estática** do sistema

Diagrama de Classes

- Na UML, uma classe possui a notação de um retângulo dividido em três partes
 - Nome da classe
 - Atributos da classe
 - Métodos da classe

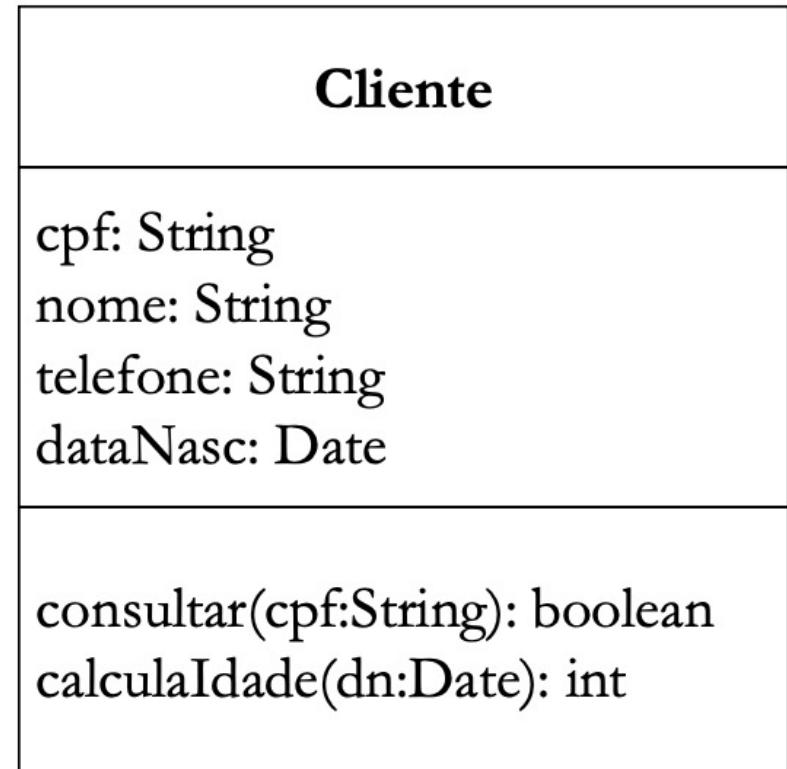


Diagrama de Classes

- Representação de atributos

visibilidade nome : tipo = valor inicial {propriedades}

- **Visibilidade:** public (+), private (-), protected (#)
- **Tipo do atributo:** int, double, String, Date, ...
- **Valor inicial:** definido no momento da criação do objeto
- **Propriedades:** final, estatic, ...
- **Exemplos:**
 - nomeFunc:String = null
 - + PI:double = 3.1415 {final}

Diagrama de Classes

- Representação de métodos

visibilidade nome(tipo) : tipo {propriedades}

- **Visibilidade:** public (+), private (-), protected (#)
- **Tipo do atributo:** int, double, String, Date, ...
- **Tipo de retorno:** int, double, String, Date, ...
- **Propriedades:** final, abstract, ...
- **Exemplos:**
 - + getName () :String {abstract}
 - + calcArea (Shape) :double
 - + pow (double, double) :double {final}

Relacionamento entre Classes

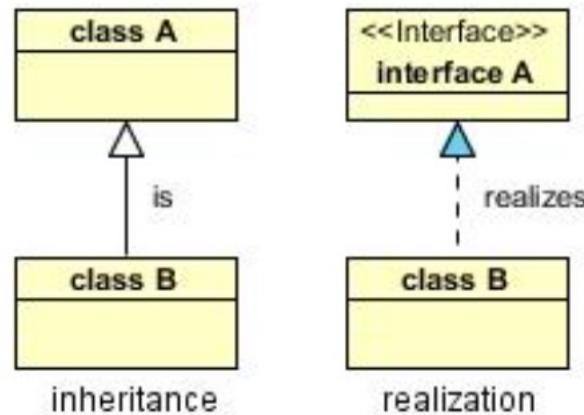
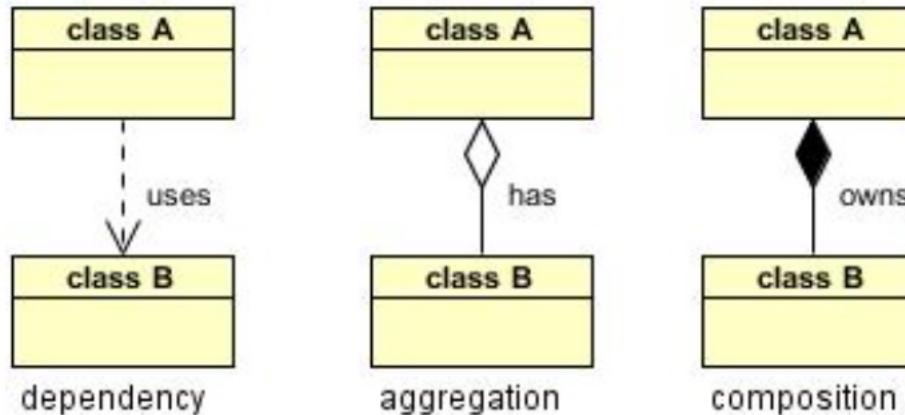
RELACIONAMENTO ENTRE CLASSES

- Em UML é possível representar o relacionamento entre as classes
- Vamos abordar as principais representações
 - Tipos de conexões
 - É uma parte do diagrama de classes

RELACIONAMENTO ENTRE CLASSES

- Generalização (herança)
 - “é um”
- Implementação (realização)
 - Aplicada para interfaces
- Associação (dependênci)
 - “usa”
- Agregação
 - “é parte de” (possui)
 - Objeto ainda faz sentido mesmo sem a existência da agregação
- Composição
 - “é parte essencial de” (é dono de)
 - Objeto não faz sentido sem a composição

RELACIONAMENTO ENTRE CLASSES

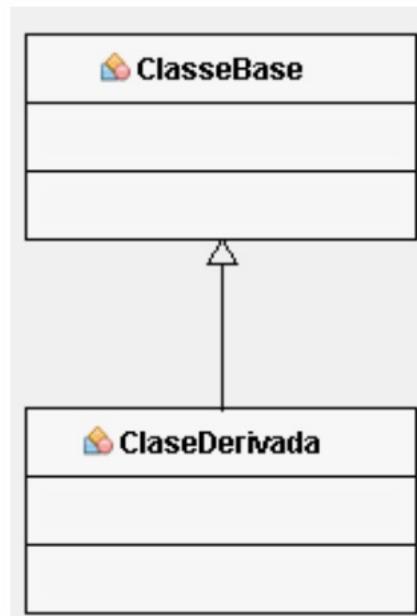


GENERALIZAÇÃO

- Representa relacionamentos do tipo “é um”
 - Herança
 - Ex: um cachorro é um mamífero
- Generalização/especialização
 - A partir de duas ou mais classes, abstrai-se uma classe mais genérica
 - De uma classe geral, deriva-se uma mais específica
 - Sub-classes possuem todas as propriedades das superclasses
 - Deve existir pelo menos uma propriedade que distingue duas classes especializadas
 - Caso contrário, não há necessidade

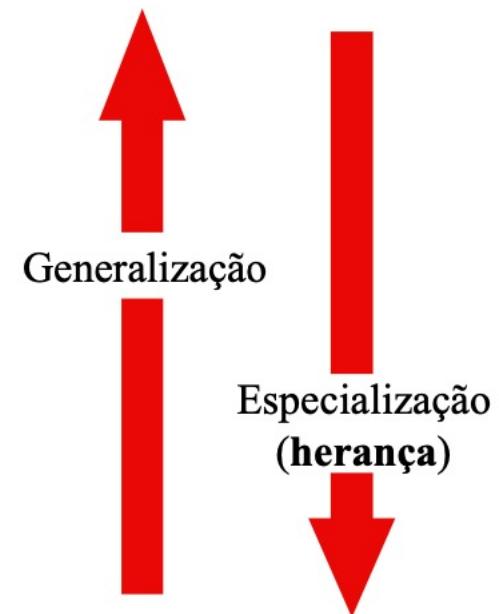
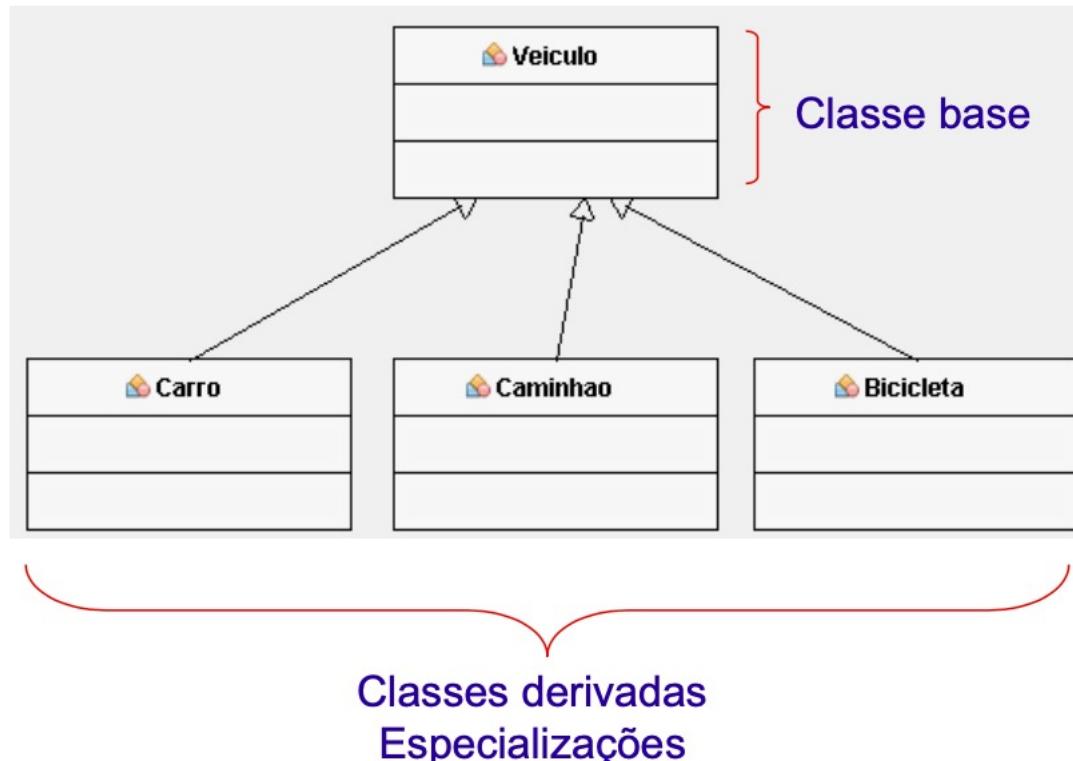
GENERALIZAÇÃO

- No diagrama de classes
 - A generalização é representada com uma seta do lado da classe mais geral (classe base)



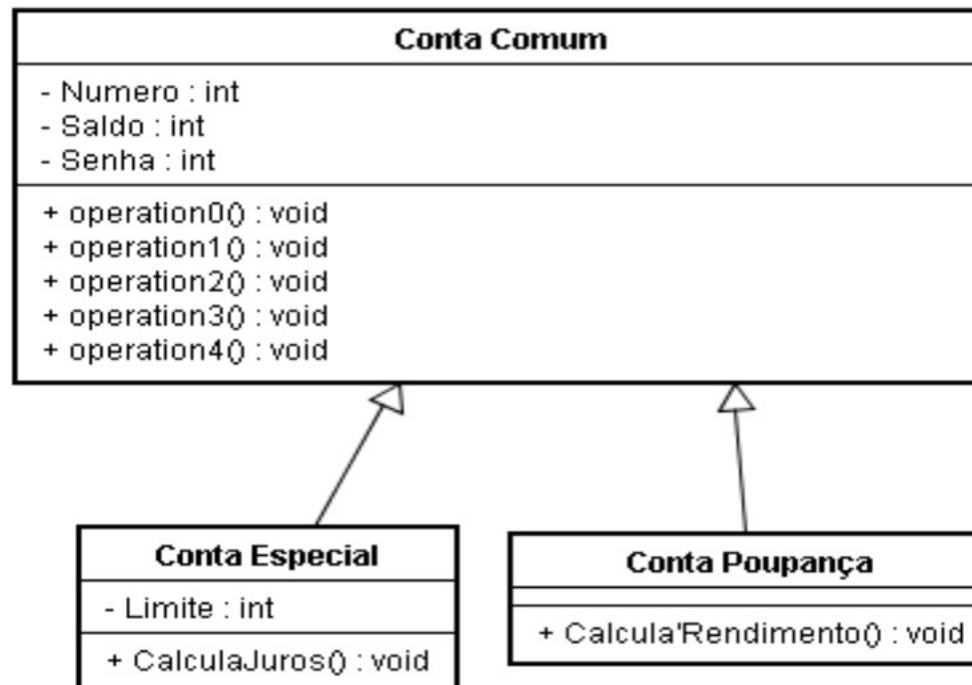
GENERALIZAÇÃO

- Exemplo



GENERALIZAÇÃO

- Exemplo



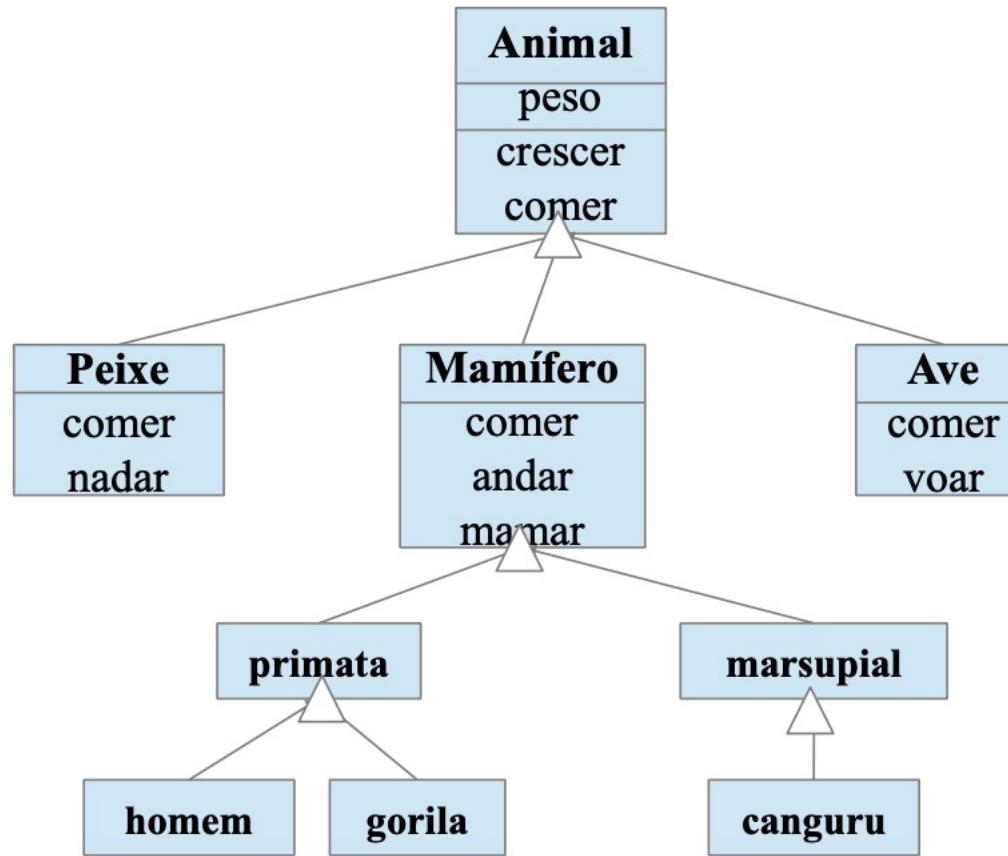
22

GENERALIZAÇÃO

- Permite organizar as classes hierarquicamente
- Técnica de reutilização de software
 - Novas classes são criadas a partir de classes existentes, absorvendo seus atributos e comportamentos (métodos)
 - Recebe novos recursos posteriormente

GENERALIZAÇÃO

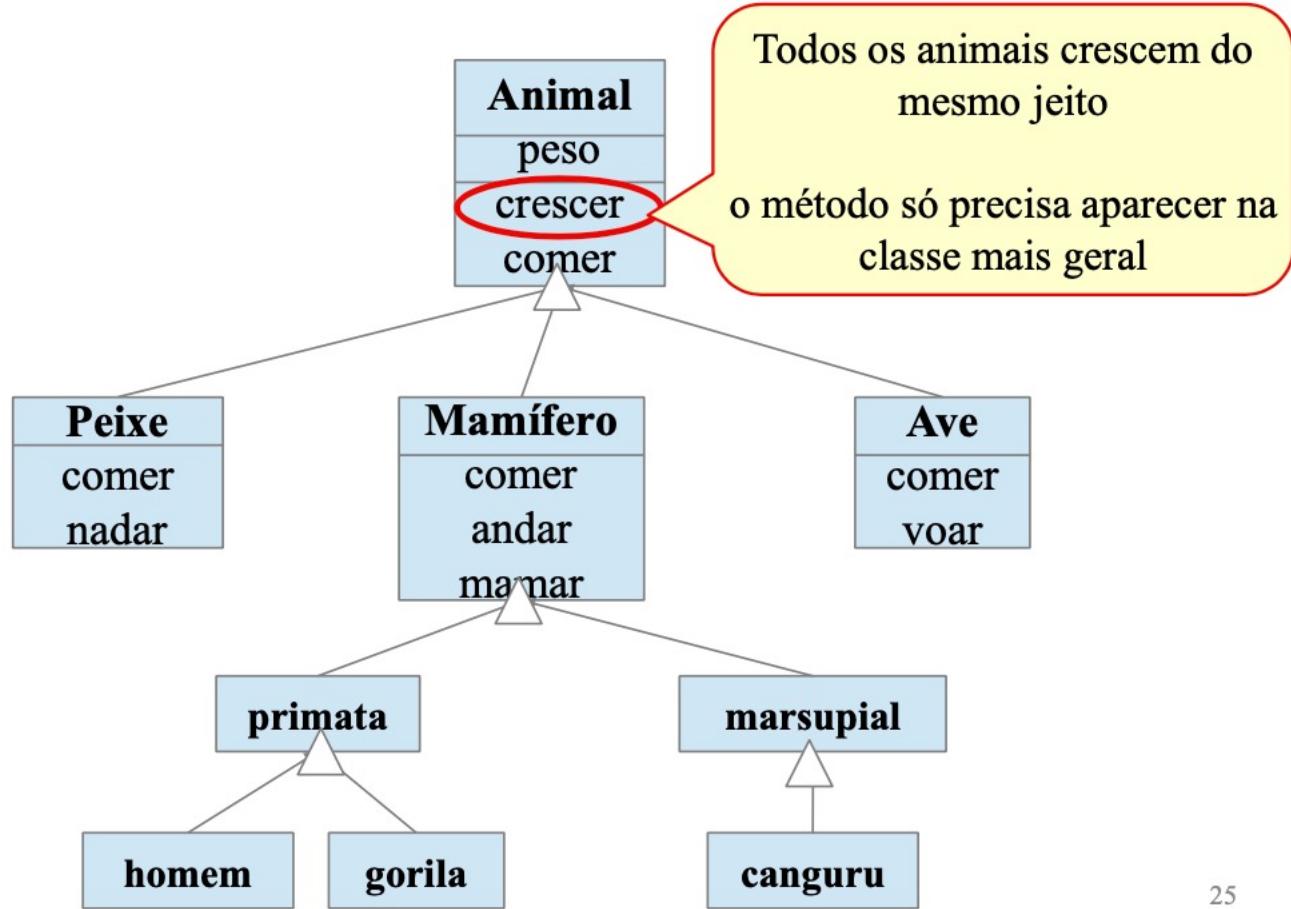
- Exemplo de hierarquia de classes



24

GENERALIZAÇÃO

- Exemplo de hierarquia de classes

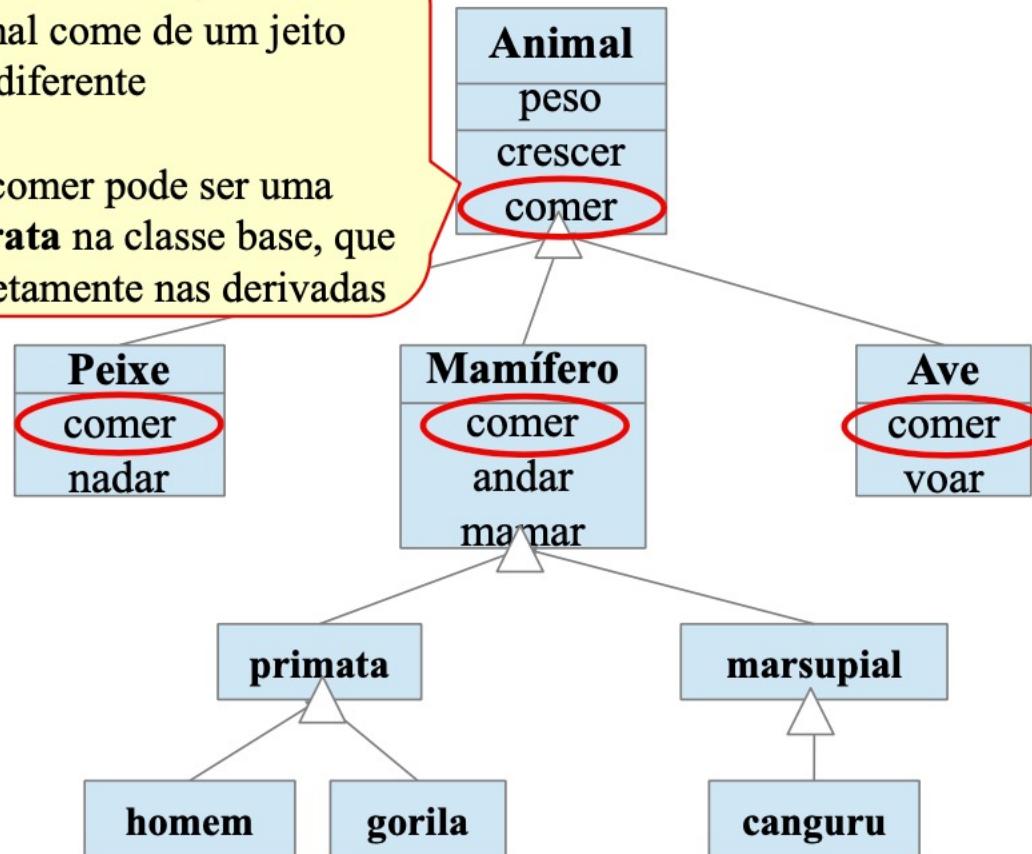


GENERALIZAÇÃO

- Exemplo de hierarquia de classes

Todos os animais comem, mas cada tipo de animal come de um jeito diferente

O método comer pode ser uma operação **abstrata** na classe base, que aparece concretamente nas derivadas

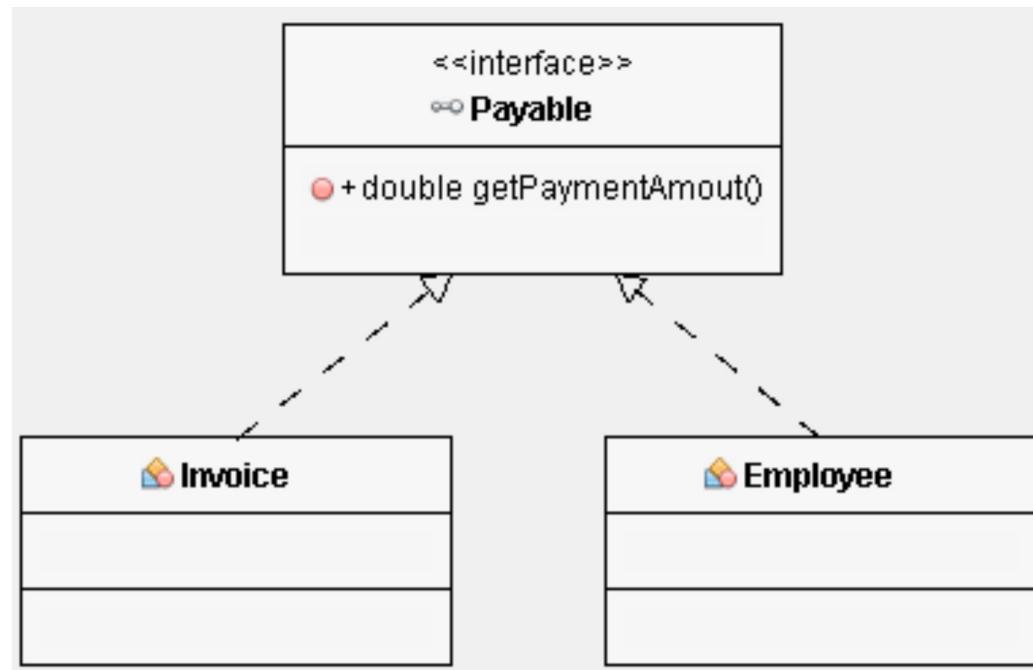


IMPLEMENTAÇÃO

- Interfaces estabelecem um contrato entre os objetos
 - Definição dos métodos pertencentes àquele contrato
- Interfaces não podem ser instanciadas
 - Não são classes comuns
- Em classes, podemos usar **herança**
- Em interfaces, utiliza-se a **implementação**

IMPLEMENTAÇÃO

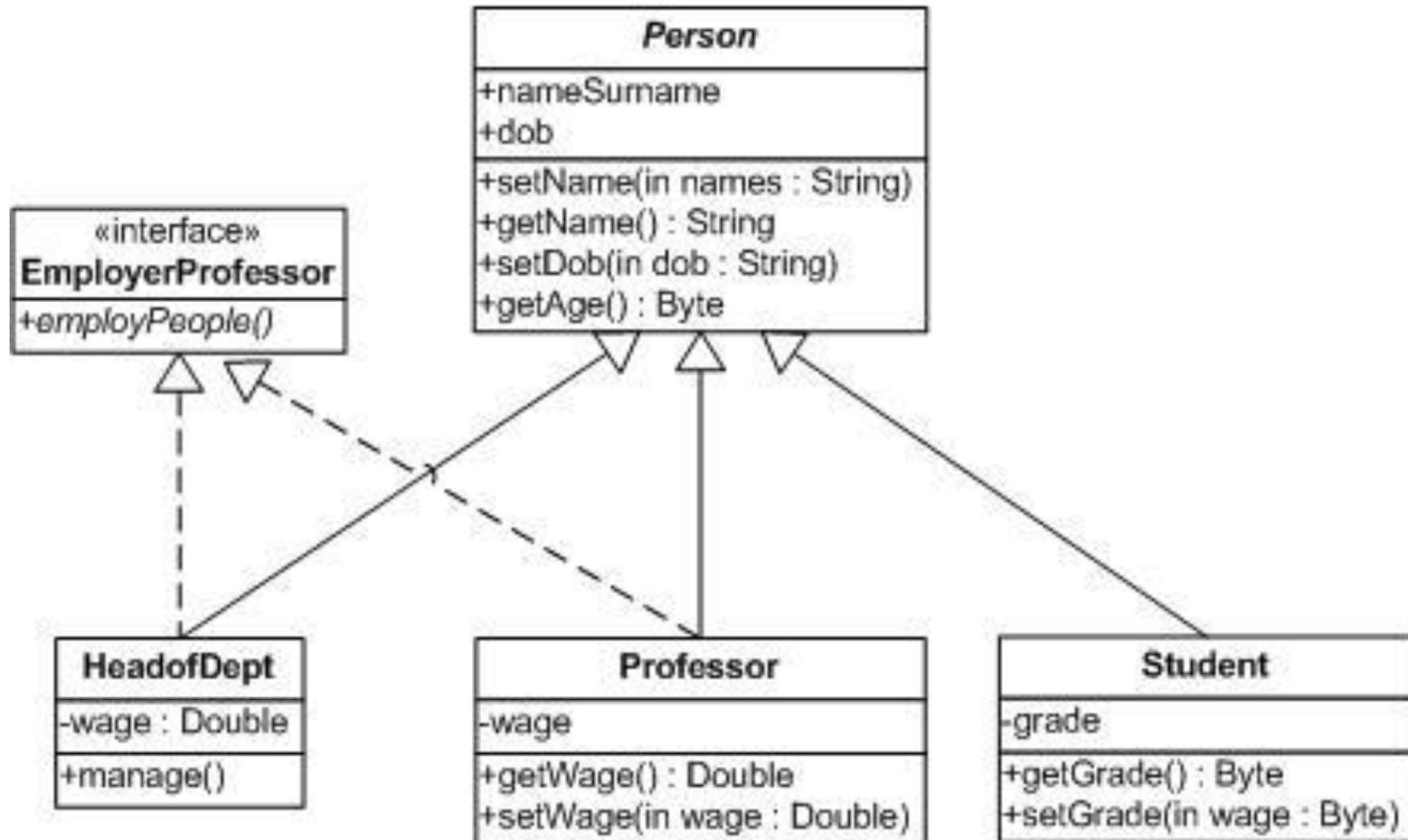
- Em UML, interfaces são definidas de forma similar às classes
 - Diferenciadas com uma marcação de *interface*
 - Implementação é parecida com a herança



IMPLEMENTAÇÃO

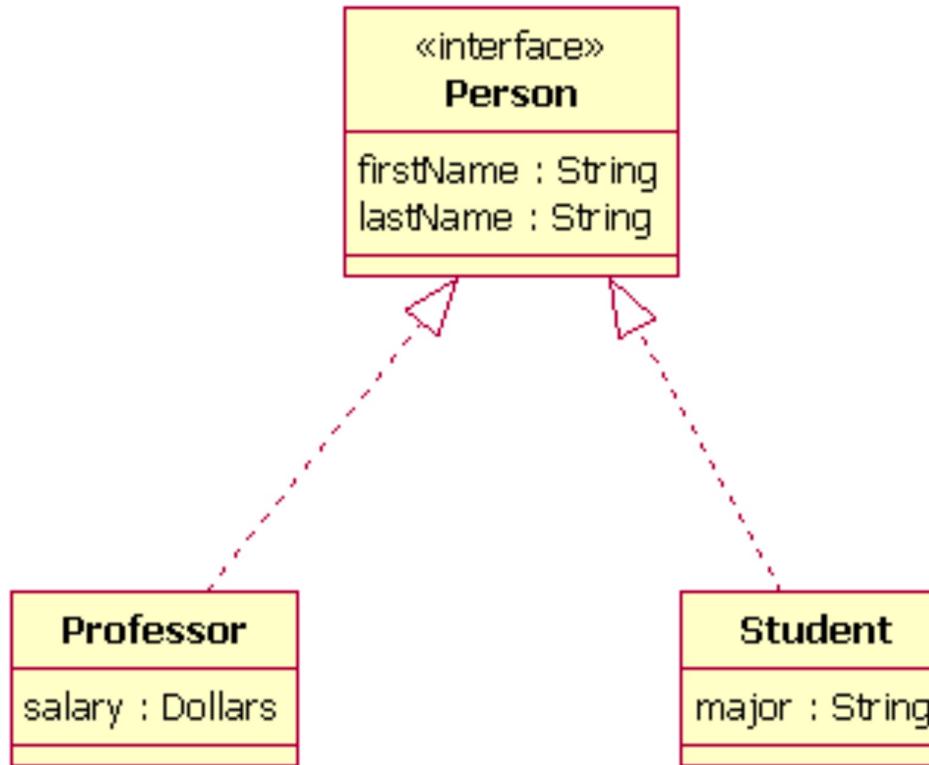
- Quando uma classe **herda** outra classe, a implementação dos métodos é herdada
- Quando uma classe **implementa** uma interface, os métodos definidos na interface precisam ser implementados
 - Em geral, não há implementação em uma interface, só definição
 - Todos os métodos da interface precisam necessariamente ser escritos pela classe que implementa a interface

IMPLEMENTAÇÃO



IMPLEMENTAÇÃO

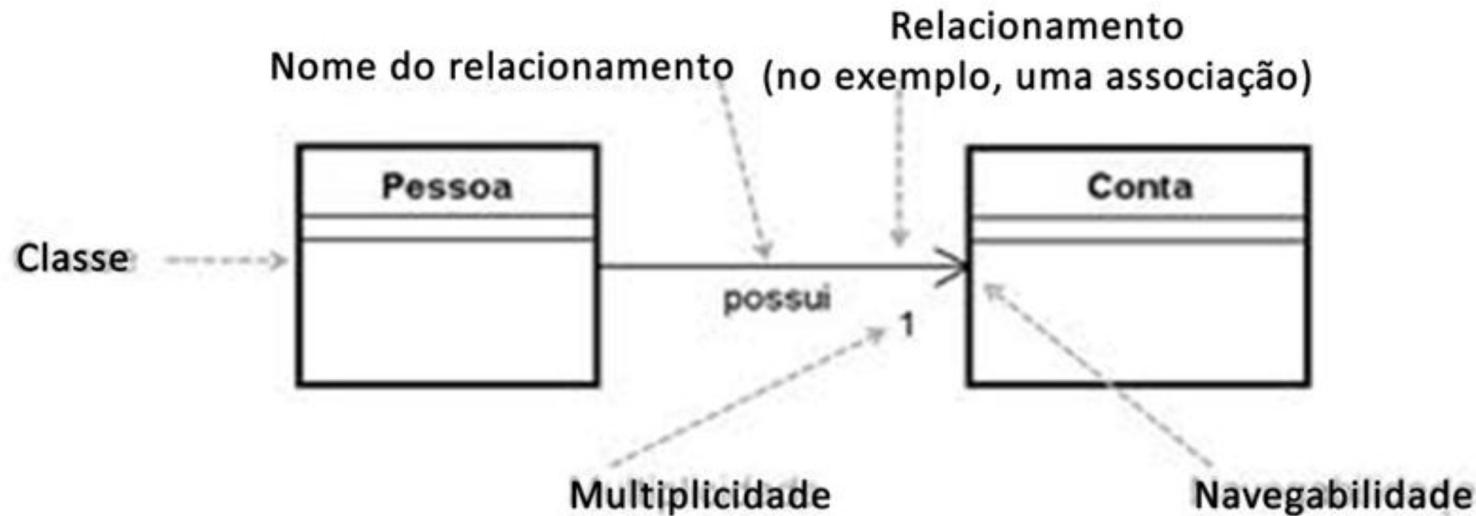
- Poderíamos usar herança? Qual a vantagem?
 - Qual a relação entre as classes?



RELACIONAMENTOS

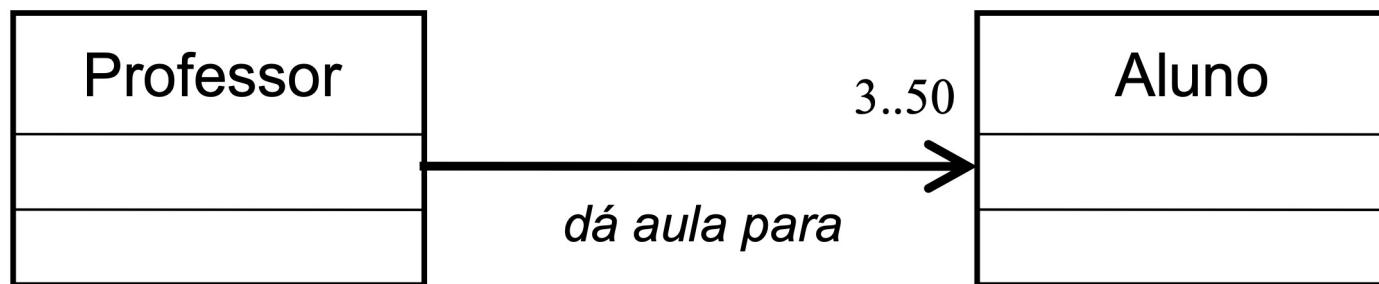
- Os relacionamentos são caracterizados por
 - Nome
 - Descrição do relacionamento
 - Em geral usa-se um verbo
 - Faz, tem, possui
 - Naveabilidade
 - Indicada por uma seta no fim do relacionamento
 - Uni (uma flecha) ou bidirecional (sem flechas/duas flechas)
 - Multiplicidade
 - Quantidade de elementos que cada relacionamento pode assumir
 - 0..1, 0..*, 1, 1..*, 2, 3..7

RELACIONAMENTOS



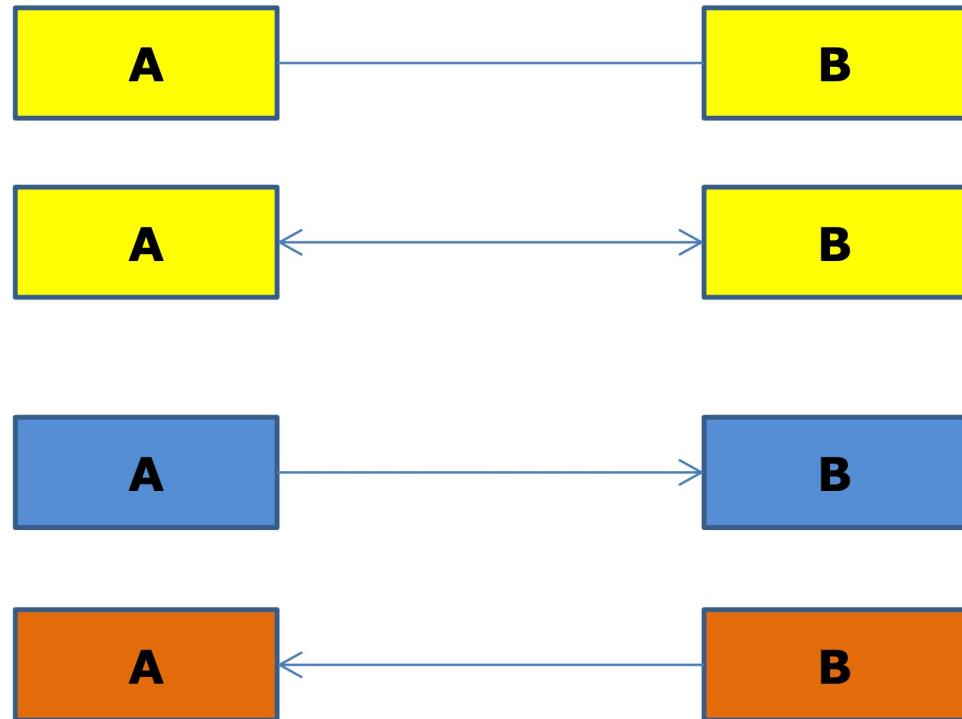
NOME DO RELACIONAMENTO

- Nomear um relacionamento facilita o entendimento
- Nome do relacionamento (rótulo) é colocado ao longo da linha de associação



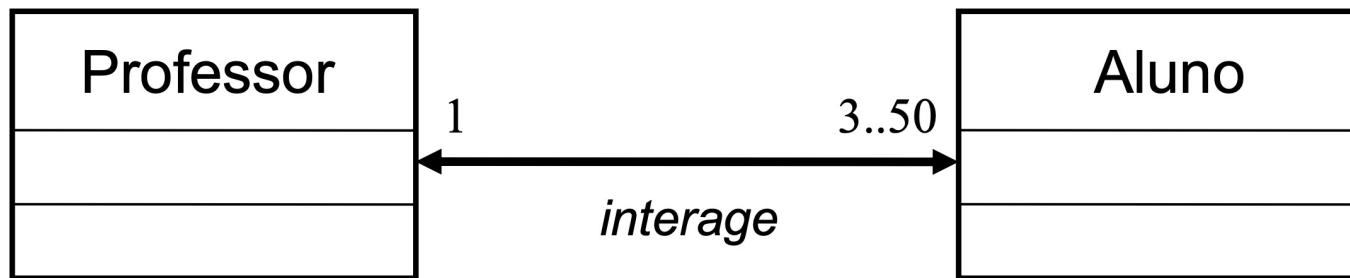
NAVEGABILIDADE

- Navegabilidade indica a direcionalidade com que as classes se relacionam
- Ambas as classes se relacionam (sabem da existência uma da outra)
- B não sabe da existência de A
- A não sabe da existência de B



MULTIPLICIDADE

- Multiplicidade é o **número de instâncias** de uma classe relacionada com uma ou mais instâncias de outra classe
- Exemplo: Professor e Aluno
 - Cada Professor pode interagir com 3 a 50 Alunos
 - Cada Aluno pode interagir com apenas um Professor
 - Pensando em um único curso



MULTIPLICIDADE

Muitos

*

Exatamente um

1

Zero ou mais

0...*

Um ou mais

1...*

Zero ou um

0..1

Faixa especificada

2..4

MULTIPLICIDADE

- Exemplos

- Uma mesa de restaurante pode ter vários ou nenhum pedido
 - *..0
- Uma cotação pode incluir no mínimo 1 e até muitos (*) itens cotados
 - 1..*
- Uma casa pode ter de 0 a 3 funcionários
 - 0..3

ASSOCIAÇÃO SIMPLES

- É a forma mais fraca de relacionamento entre classes
 - As classes que participam desse relacionamento são **independentes**
 - São representadas como linhas conectando as classes participantes
 - Podem ter um nome identificando a associação
 - Podem ter uma seta junto ao nome indicando que a associação somente pode ser utilizada em uma única direção (o mais usual e adequado)
 - Representa relacionamentos “usa um”
 - Pessoa **usa um** Carro

ASSOCIAÇÃO SIMPLES

- Na implementação
 - ObjetoA **usa** ObjetoB quando o ObjetoA **chama um método público** do ObjetoB
- Associação simples também é chamada de **dependência**
- Diagramas de dependência são os primeiros diagramas usado para compreender um código que não é seu

ASSOCIAÇÃO SIMPLES

- Exemplo

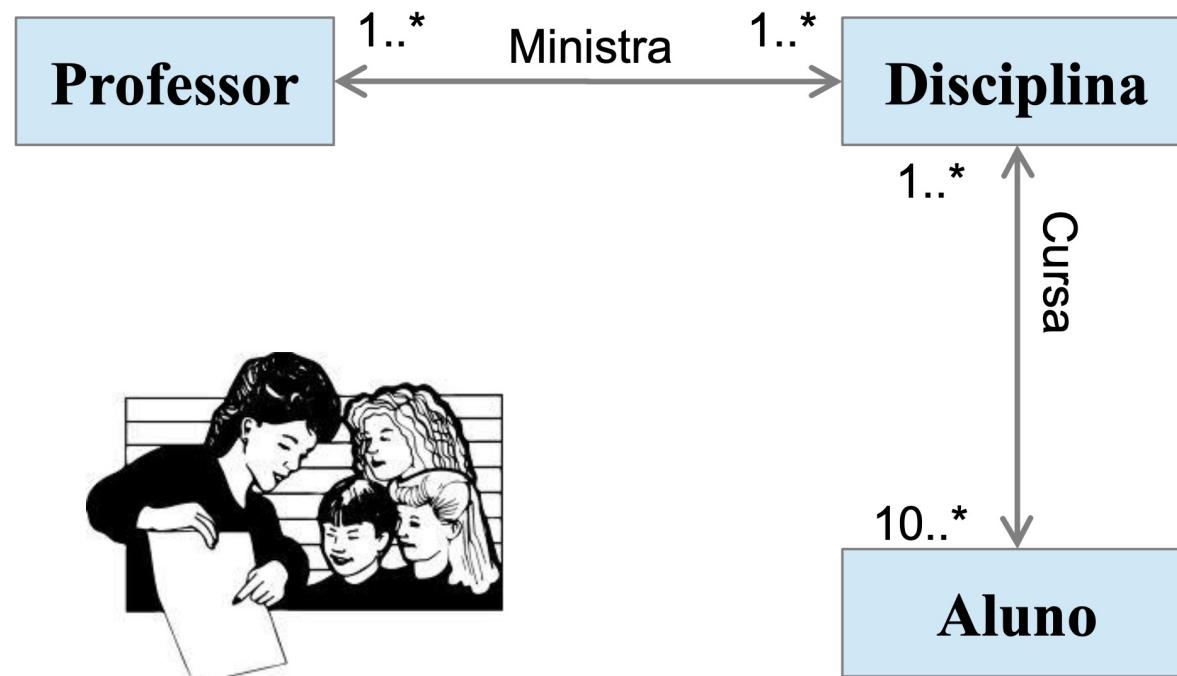
- Um **Passageiro** pode viajar para qualquer lugar, dependendo de qual **Avião** ele entrar
- Para que um **Passageiro** viaje, ele precisa apenas de uma indicação de qual **Avião** ele deve entrar. Ele não precisa ter como parte de sua informação (atributo) a referência a um **Avião**.



- Leitura unidirecional
 - Um **Passageiro** viaja em um **Avião**

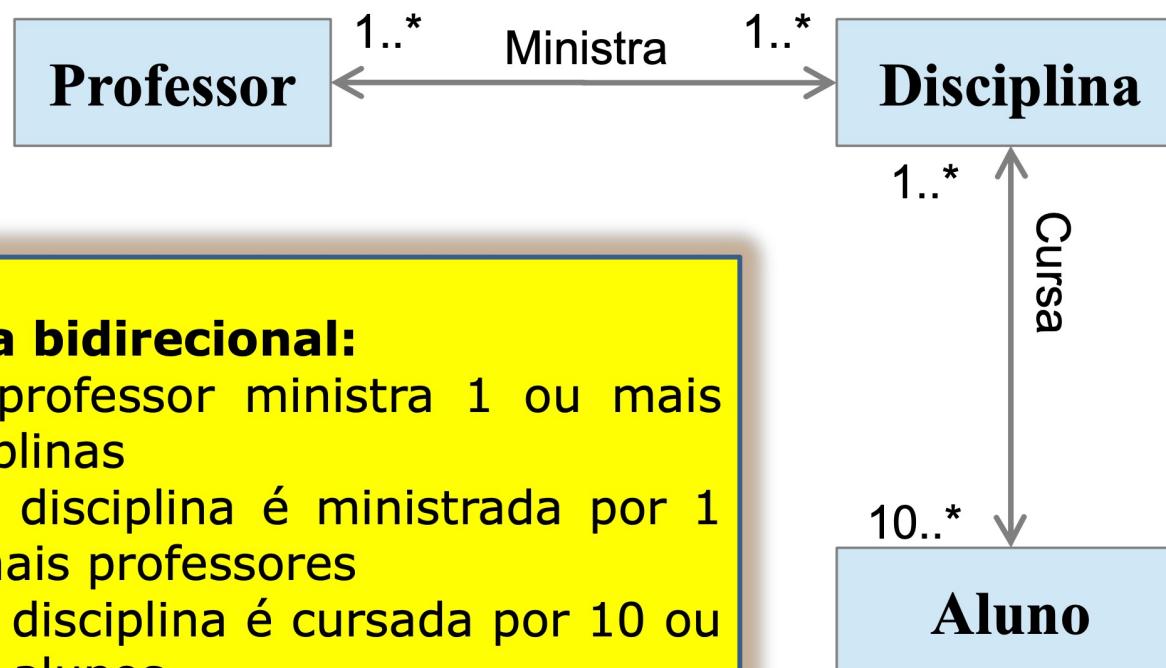
ASSOCIAÇÃO SIMPLES

- Exemplo bidirecional



ASSOCIAÇÃO SIMPLES

- Exemplo bidirecional



Leitura bidirecional:

- Um professor ministra 1 ou mais disciplinas
- Uma disciplina é ministrada por 1 ou mais professores
- Uma disciplina é cursada por 10 ou mais alunos
- Um aluno cursa 1 ou mais disciplinas

ASSOCIAÇÃO SIMPLES

- Outro exemplo
 - Imagine um objeto gráfico que se auto-desenha.
 - O objeto sabe como se desenhar, mas precisa de acesso a funcionalidade gráficas exclusivas de componentes gráficos do sistema.
 - Para se desenhar, o objeto gráfico deve receber como parâmetro um componente gráfico em seu método *autoDesenho(CompGrafico comp)*.
 - Ele irá apenas usar a classe CompGráfico, sem contudo ser composto por ela.



DÚVIDAS?

