

41951- ANÁLISE DE SISTEMAS

# Introdução à análise [orientada] por objetos

Ilídio Oliveira

v2024/02/27

# Objetivos de aprendizagem

Explicar a técnica de modelação por classes, como uma forma de decompor um problema em entidades (objetos)

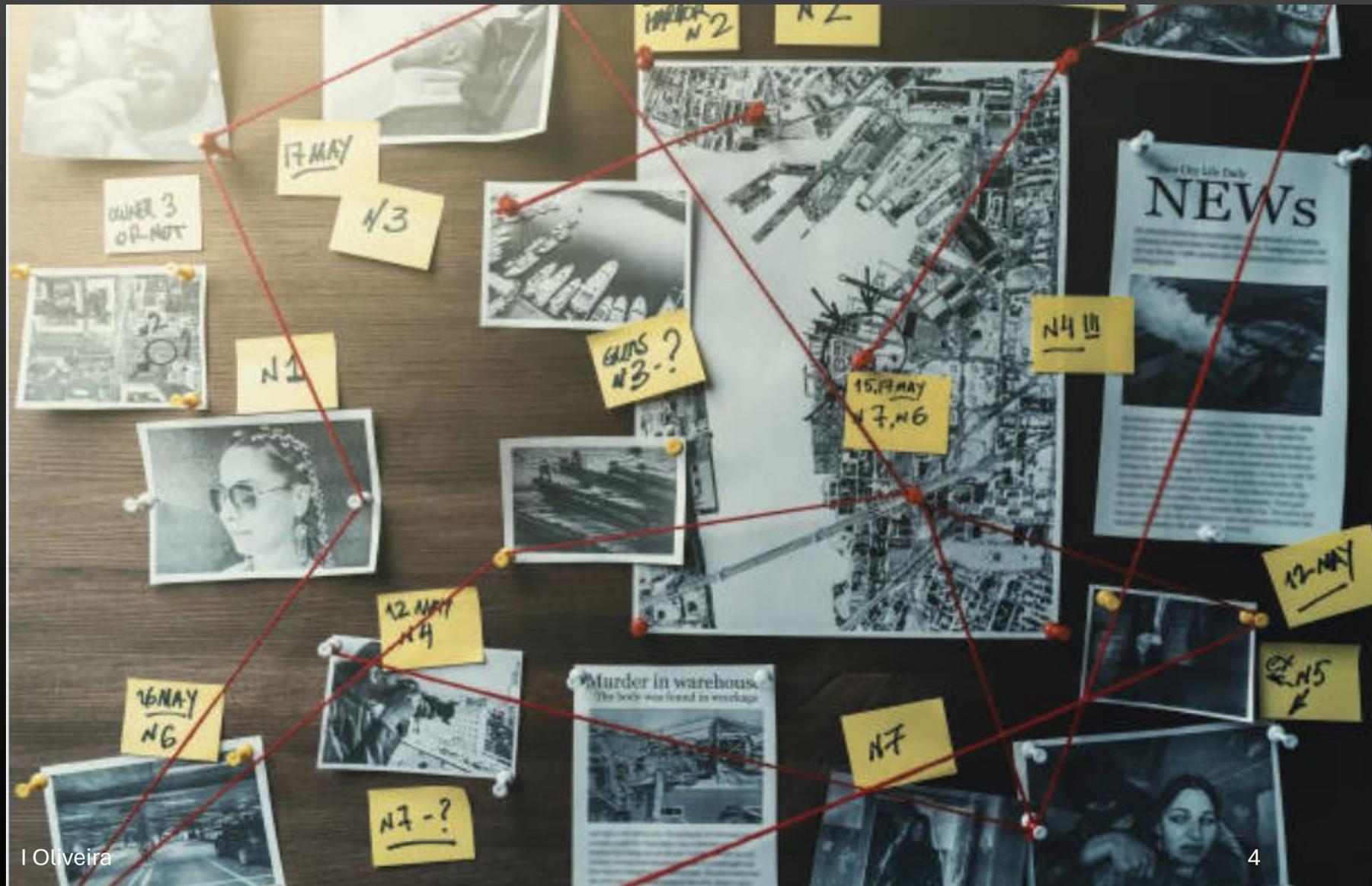
Interpretar modelos com diferentes relacionamentos: associação, agregação, generalização.

Explicar os conceitos de estado e de operações em relação a um objeto, e a forma como estão relacionados

Representar classes em diagramas.

# Objetos: classificar as “coisas” do mundo

# Crime scene investigation....



# Investigação do crime domínio

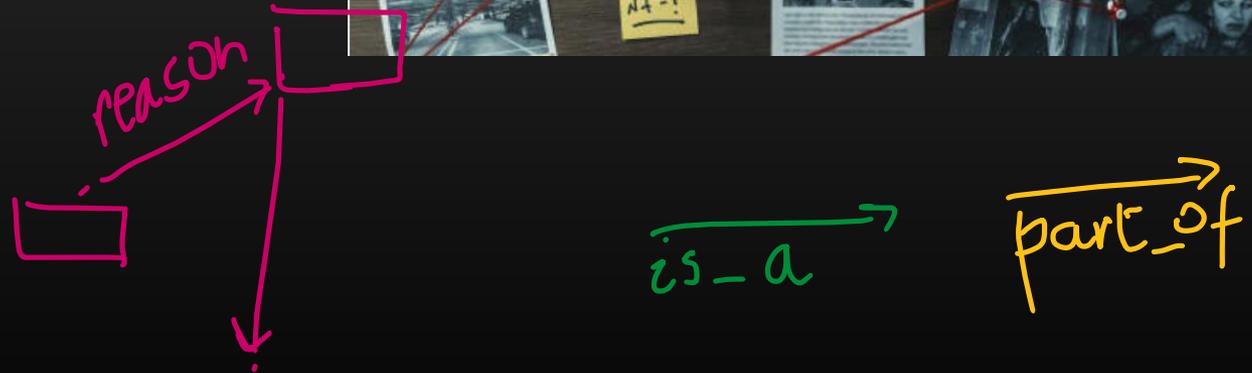
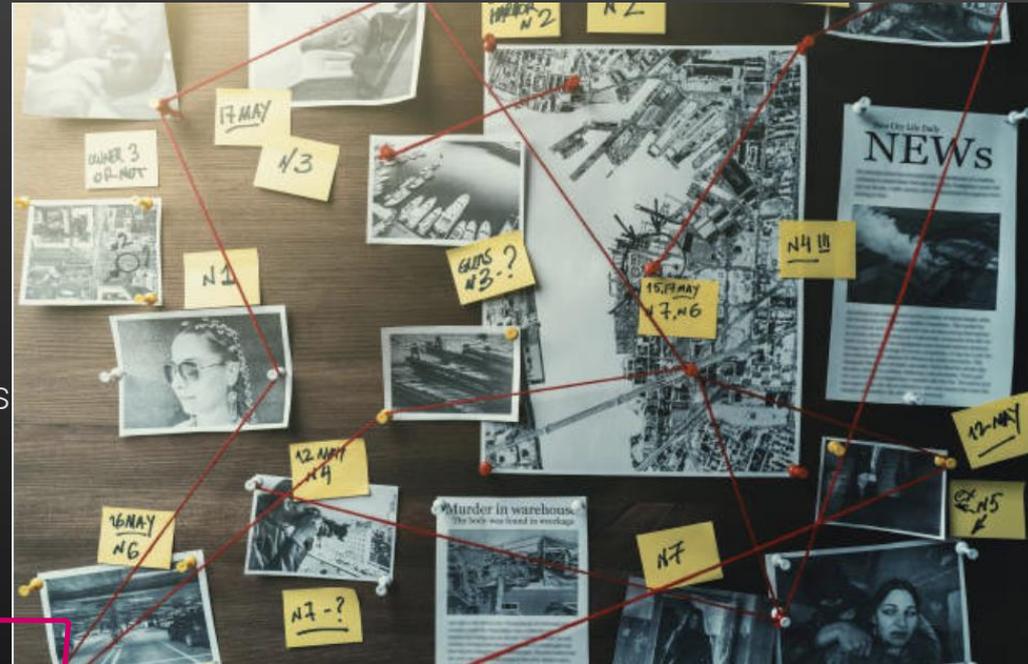
## App de podcasts

→ Programas, Episódios, Autores, Categorias, Media, Subscrições, Utilizador(=ouvinte),...

## Gestão académica?

→ Alunos, Turmas, UC, Cursos, Docentes, Diretor de curso, Departamentos,...

## Netflix?...

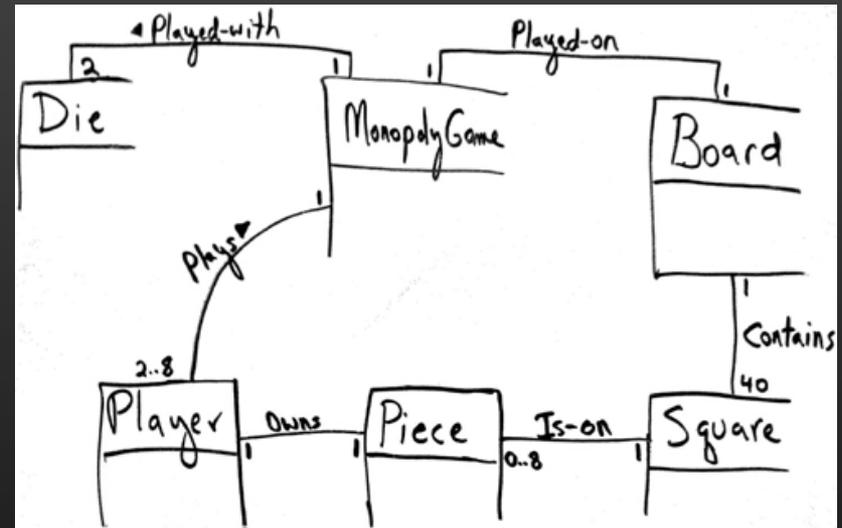


# As “coisas” do domínio

Um **modelo de domínio** é uma representação visual das classes conceptuais ou objectos reais de um domínio/área.

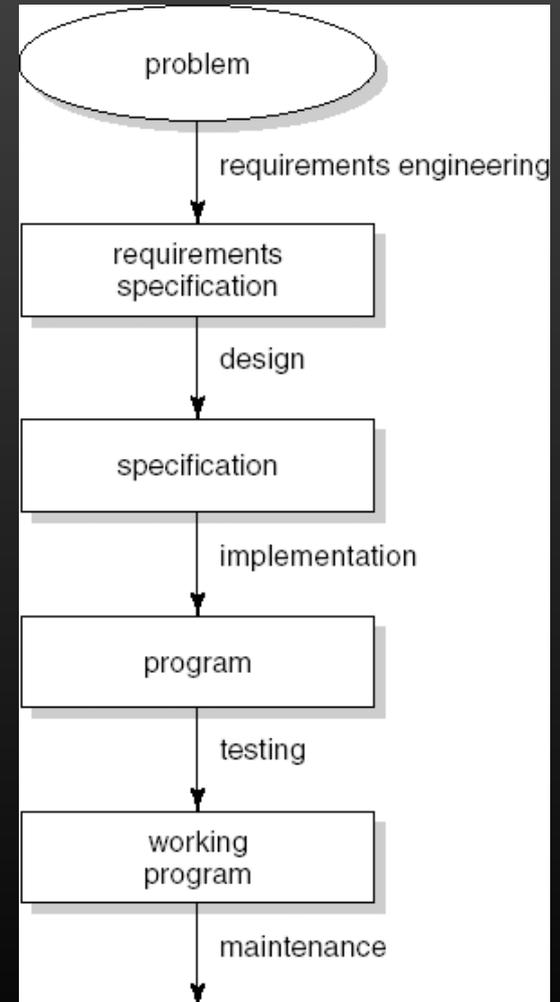
Centra-se na explicação de 'coisas' e produtos importantes para um domínio de negócio.

→ Não é (ainda) uma visão do software ou base de dados...



O passo fundamental da análise orientada por objectos é a decomposição de um domínio em conceitos relevantes → **objetos**.

# Fluxo do desenvolvimento onde entram os "objetos"?



During *object-oriented analysis* there is an emphasis on finding and describing the objects—or concepts—in the problem domain. For example, in the case of the flight information system, some of the concepts include *Plane*, *Flight*, and *Pilot*.

During *object-oriented design* (or simply, object design) there is an emphasis on defining software objects and how they collaborate to fulfill the requirements. For example, a *Plane* software object may have a *tailNumber* attribute and a *getFlightHistory* method (see [Figure 1.2](#)).

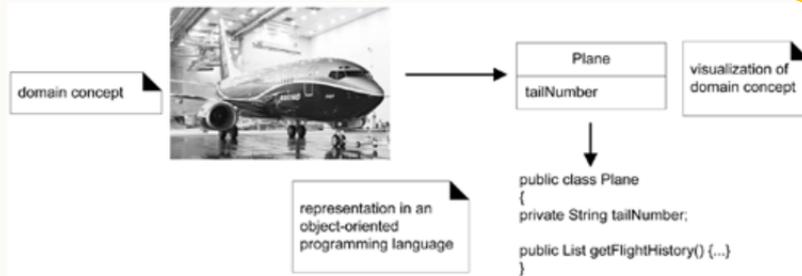
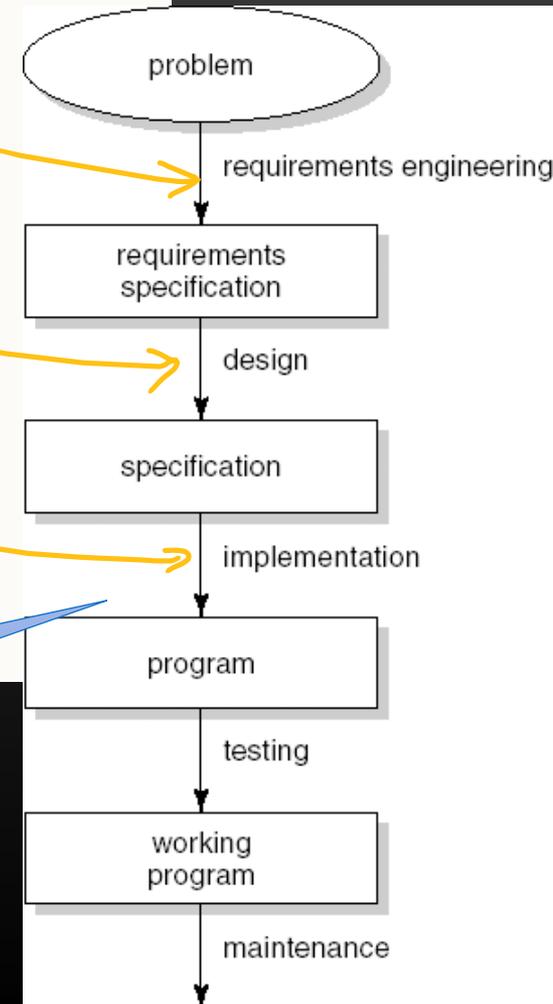


Figure 1.2. Object-orientation emphasizes representation of objects.

Finally, during implementation or *object-oriented programming*, design objects are implemented, such as a *Plane* class in Java.



Motivação: será possível manter o mesmo esquema mental para representar as “coisas” do problema, ao longo do SDLC? (baixar o gap representacional com modelação OO)

# 3 mecanismos principais (para modelar/gerir a complexidade)

## Abstração



- Tratar os objetos que são semelhantes como um “tipo”/categoria

## Encapsulamento



- O objeto é uma unidade que esconde o seu estado interno

- A interação com o objeto é feita através de “pontos de acesso”

## Hierarquia



- Procurar relações “é-um” (is-a)

- Procurar relações “parte-de” (part-of)

# Muitas instâncias.. do mesmo conceito (objeto)



I Oliveira

Abstracção

Veiculo
-matricula
-cor
-lotação
-velocidade atual
+travar()
+acelerar()

# Mecanismo: abstração

Abstração decorre do reconhecimento de semelhanças entre certos objetos, situações ou processos no mundo real, e a decisão de se concentrar nestas semelhanças → um *tipo de coisa*

Uma abstração denota as características essenciais de um objeto que o distingue de todos os outros tipos de objetos.

<b>Abstraction:</b> Temperature Sensor
<b>Important Characteristics:</b> temperature location
<b>Responsibilities:</b> report current temperature calibrate

Figure 2-6 Abstraction of a Temperature Sensor

<b>Abstraction:</b> Heater
<b>Important Characteristics:</b> location status
<b>Responsibilities:</b> turn on turn off provide status

Related Candidate Abstractions: Heater Controller, Temperature Sensor

Figure 2-9 Abstraction of a Heater

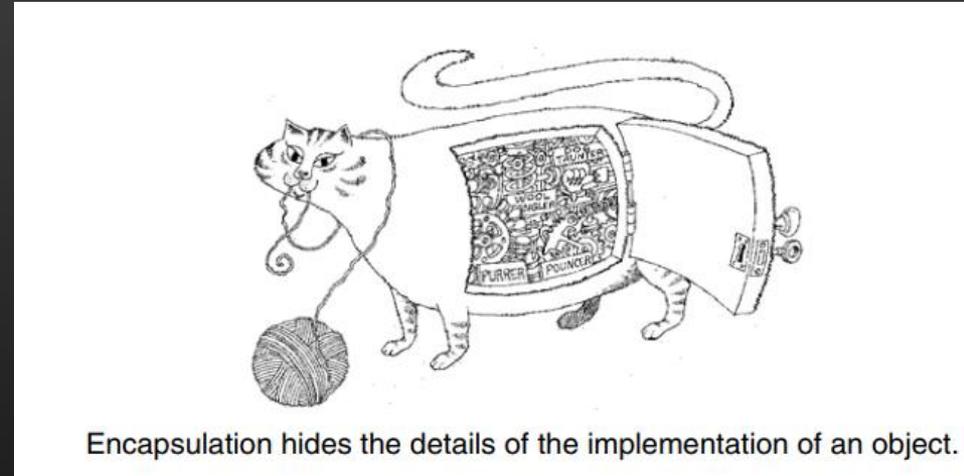
# Encapsulamento

Encapsulamento incentiva a **ocultação da estrutura (interna) de um objeto** (a informação que mantém), bem como a implementação dos seus métodos (como faz).

Nenhuma parte de um sistema complexo deve depender do conhecimento dos detalhes internos de qualquer outra parte.

O encapsulamento permite que as alterações de um programa sejam feitas de forma fiável, com repercussões limitadas

→ mexer num módulo não deve estragar os outros



# Como é que o encapsulamento contribui para a gerir a complexidade?

Encapsulamento ajuda a gerir a complexidade escondendo a dimensão “privada” (interna) das nossas abstrações.

**Abstraction:** Temperature Sensor

**Important Characteristics:**

temperature  
location

**Responsibilities:**

report current temperature  
calibrate

**Figure 2-6** Abstraction of a Temperature Sensor

**Abstraction:** Heater

**Important Characteristics:**

location  
status

**Responsibilities:**

turn on  
turn off  
provide status

Related Candidate Abstractions: Heater Controller, Temperature Sensor

**Figure 2-9** Abstraction of a Heater

O que é que *Heater* deve conhecer de *Temperature Sensor*?

# Hierarquia: a relação de herança

A herança define uma relação entre classes (tipos de coisas), em que uma classe partilha/especializa a estrutura ou o comportamento definido numa outra classe.

A herança denota uma relação semântica "é-um", e.g.:

um urso "é um" (tipo de) mamífero; uma casa "é um" tipo de ativo imobiliário; uma caravana "é um" veículo.

A herança implica, assim, uma generalização  $\leftrightarrow$  especialização



**Abstraction:** Temperature Sensor

**Important Characteristics:**

temperature  
location

**Responsibilities:**

report current temperature  
calibrate

Figure 2-6 Abstraction of a Temperature Sensor

Qual a relação semântica? Um Sensor de temperatura é um (*is-a*) Sensor.

# Hierarquia: a relação de agregação

A agregação define uma relação entre categorias de coisas (i.e. classes) do tipo “parte-de”

A agregação denota a relação semântica “parte-de”, e.g.:

Um País é parte de um Continente; o Estudante é parte da Turma; uma Obra é parte de uma Coleção.

A agregação implica, assim, um agregador  $\leftarrow \rightarrow$  parte-de

<b>Abstraction:</b> Heater
<b>Important Characteristics:</b> location status
<b>Responsibilities:</b> turn on turn off provide status

Related Candidate Abstractions: Heater Controller, Temperature Sensor

Figure 2-9 Abstraction of a Heater

<b>Abstraction:</b> Temperature Sensor
<b>Important Characteristics:</b> temperature location
<b>Responsibilities:</b> report current temperature calibrate

Figure 2-6 Abstraction of a Temperature Sensor

O *Temperature Sensor* é parte de um *Heater*.

Um Estudante é parte de uma Turma.

# 3 mecanismos principais para modelar por objetos

## Abstração



- Tratar os objetos que são semelhantes como um “tipo”/categoria

## Encapsulamento



- O objeto é uma unidade que esconde o seu estado interno

- A interação com o objeto é feita através de “pontos de acesso”

## Hierarquia



- Procurar relações “é-um”

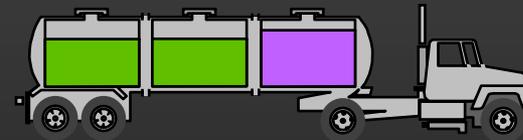
- Procurar relações “parte-de”

# Natureza e representação dos objetos

# Os objetos (em OO) modelam entidades do mundo real/espço do problema

**Observáveis no mundo físico (coisas tangíveis)**

e.g.: Aluno, Avião, Veículo



Truck

**Conceitos (geram informação)**

e.g.: Venda, Reserva, Inscrição



Chemical Process

**Abstrações próprias do software**

e.g.: ListaLigada, Vetor, HashTable



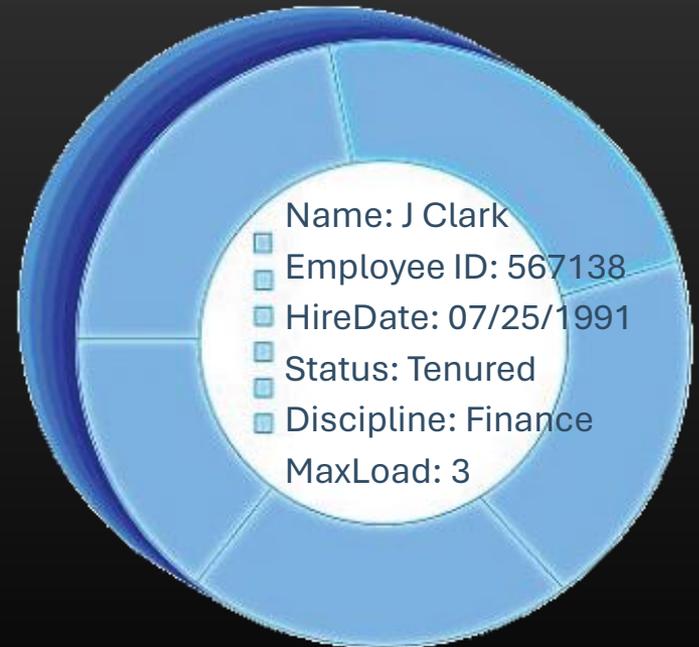
# Um objeto tem um estado (interno)

O estado de um objeto corresponde a uma das condições/configurações é que é possível o objeto apresentar-se.

É normal o estado objeto mudar ao longo do tempo.



Name: J Clark  
Employee ID: 567138  
Date Hired: July 25, 1991  
Status: Tenured  
Discipline: Finance  
Maximum Course Load: 3 classes

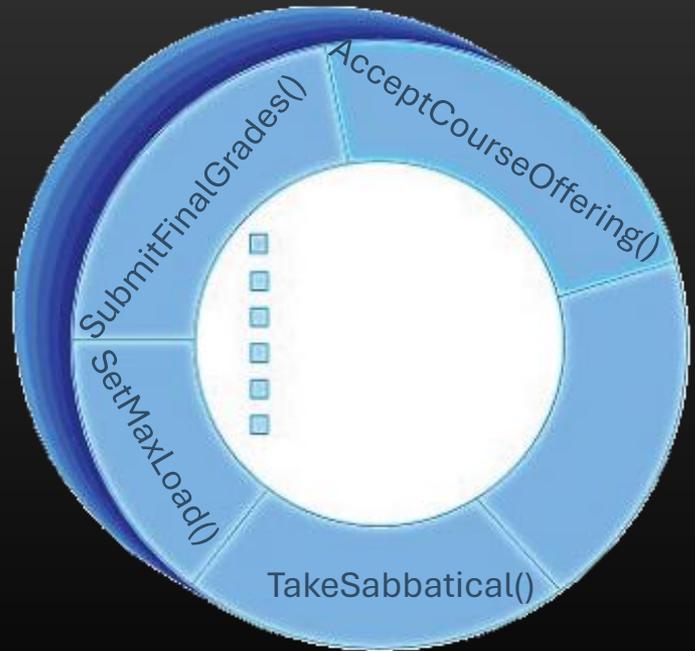


Professor Clark

# Um objeto tem comportamento (funcionalidades)

O comportamento define como é que o objeto age/reage

O comportamento visível/exposto é modelado pelo conjunto de mensagens a que responde (operações)



Professor Clark's behavior  
Submit Final Grades  
Accept Course Offering  
Take Sabbatical  
Maximum Course Load: 3 classes

Professor Clark

# Objeto: estado + operações

## Sabe alguma coisa

Os seus atributos

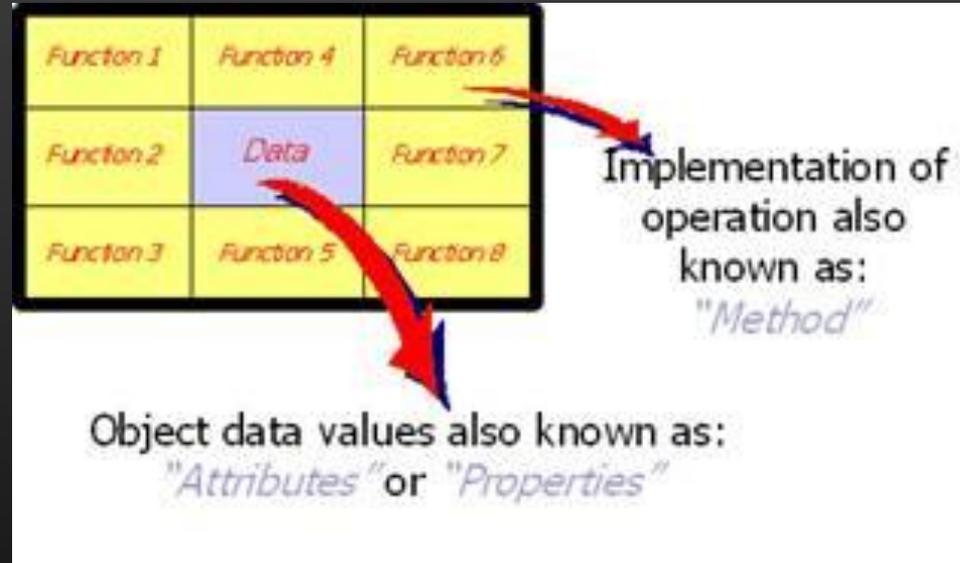
Os seus relacionamentos

## Sabe fazer alguma coisa

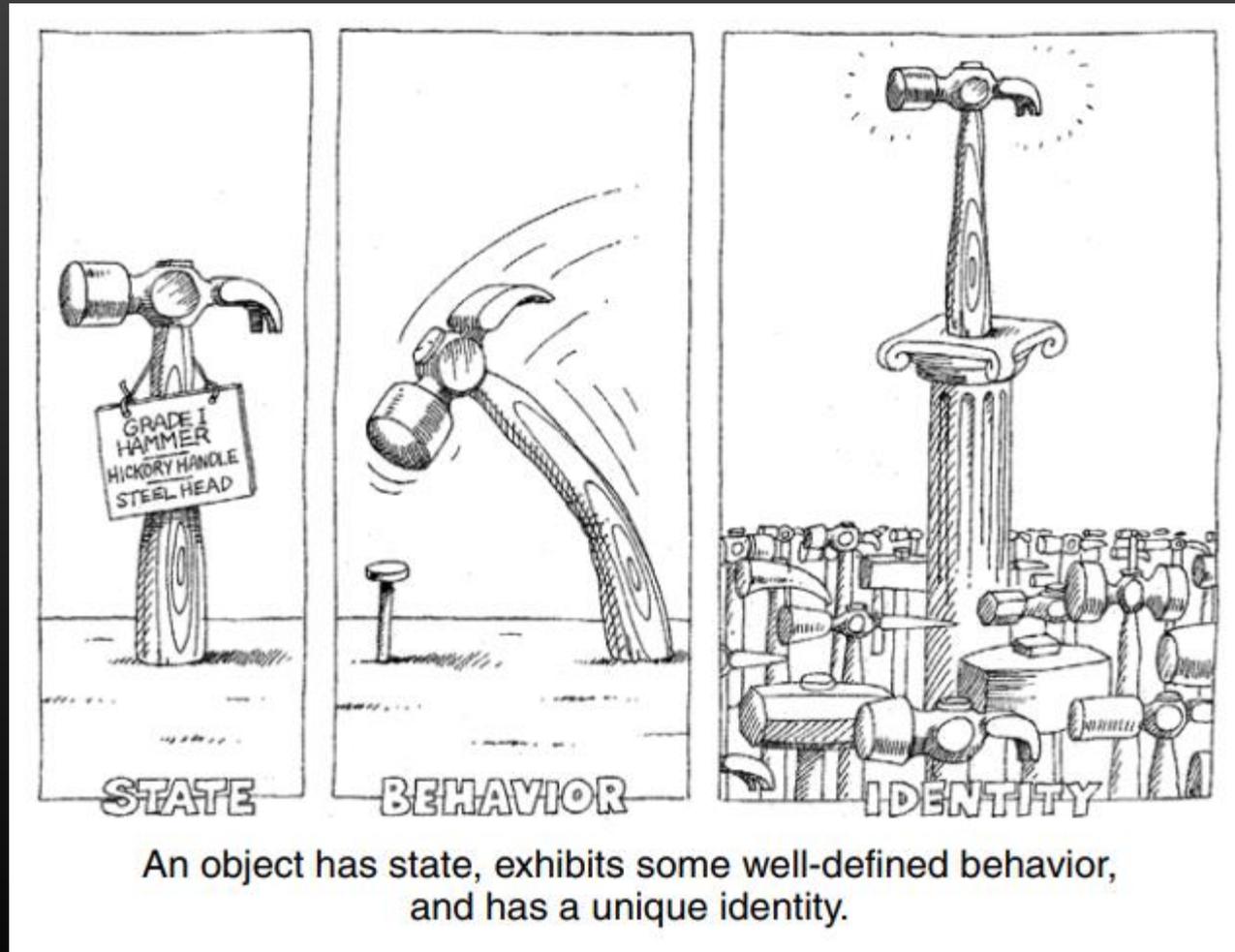
O seu comportamento, ativado através de operações

## Cápsula

Os outros (objetos) não precisam de saber o que ele sabe...



# Objeto: estado + operações + identidade



# O que é a Classe?

Uma classe é uma categoria de objetos semelhantes que partilham os mesmos atributos, operações, relacionamentos e semântica.

O objeto é uma instância (ocorrência) de uma classe

**Uma classe é uma concetualização**

Categoriza objetos semelhantes

Enfatiza as características de interesse (e suprime as outras)



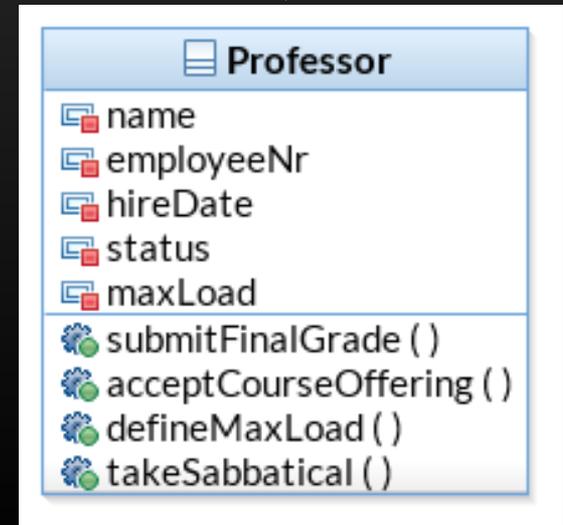
Professor Torpie



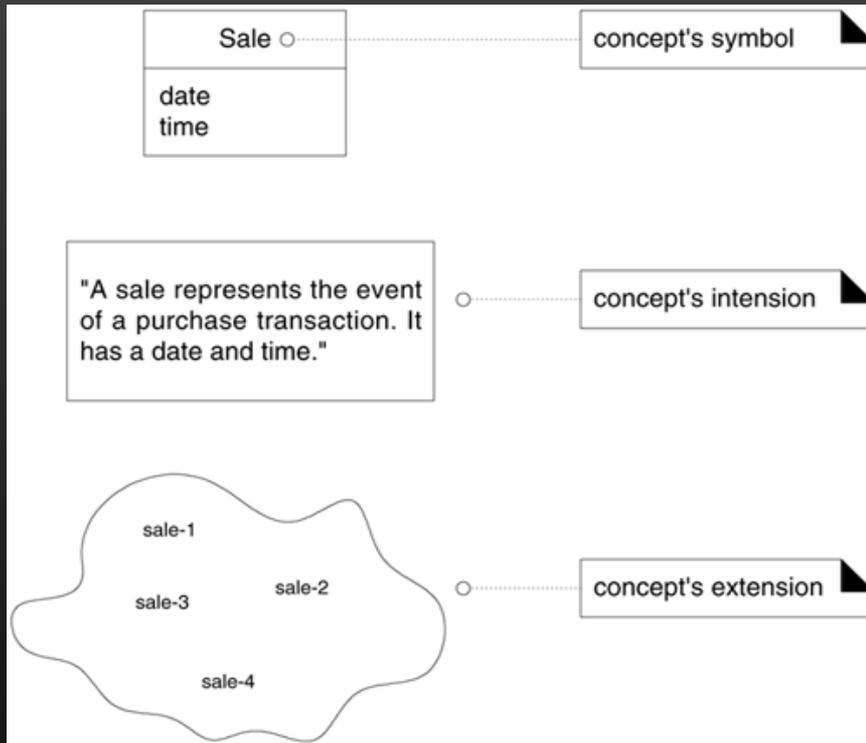
Professor Meijer



Professor Allen



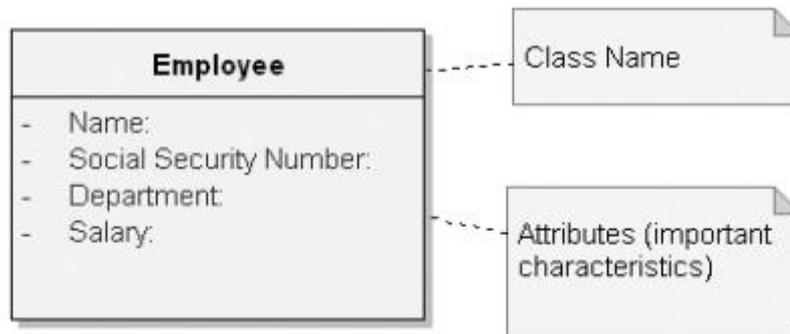
# Classe



→ um elemento de modelação (símbolo)

→ com uma interpretação

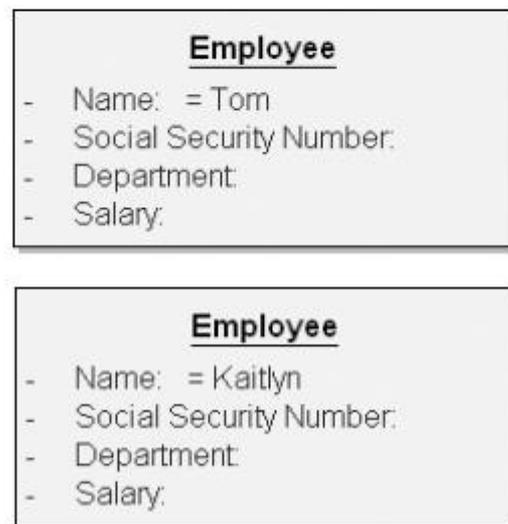
→ e um conjunto de instâncias



A classe define as características de um tipo/objeto. Funciona como o “molde” de um conceito.

**Figure 3–1** Employee Class with Attributes

specific instance. When made specific, we may have, for example, two distinct objects: Tom and Kaitlyn, each of which takes up some amount of space in memory (see Figure 3–2).



Um objeto (ou instância) é uma ocorrência concreta de um tipo de coisas. (realizado a partir do “molde”)

**Figure 3–2** Employee Objects Tom and Kaitlyn

# Modelar as relações entre classes (na UML)

# Os objetos de um domínio estão ligados numa “rede de conhecimento”

Um Projeto é coordenado por um determinado Departamento.

O Projeto tem uma equipa de vários Funcionário(s) atribuída.

Cada Projeto define várias Tarefas que, por sua vez, podem estar organizadas em sub-tarefas.

Um projeto tem um gestor.  
etc.



Como transportar o conhecimento do domínio pra um modelo?  
Há “coisas” de interesse e “relacionamentos” entre elas

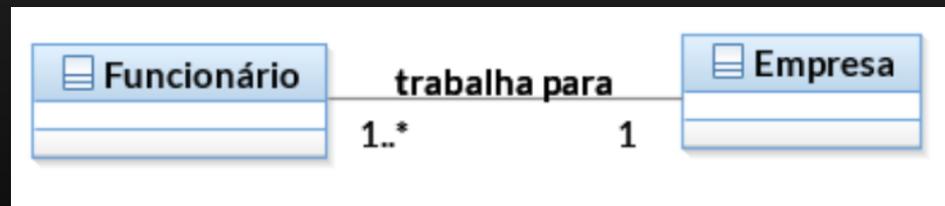
# O que é a “associação”?

A relação semântica que se estabelece entre duas ou mais classes que descreve as ligações existentes entre as respectivas instâncias.

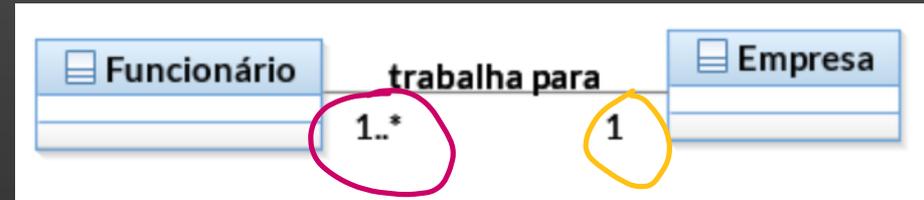
Mostra que objetos de um tipo estão ligados a objetos de outro tipo.

O tipo de “ligação” deve ser anotado com uma explicação do significado.

A classe Funcionário e Empresa estão associadas. A descrição da associação é “trabalha para”. Portanto: “Funcionário” → “trabalha em” → “Empresa”.



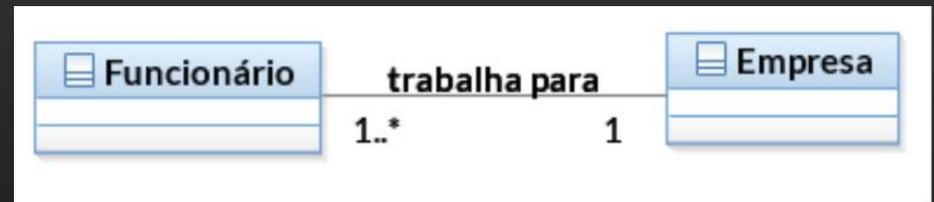
Com que “frequência” os objetos de F e E estão associados?



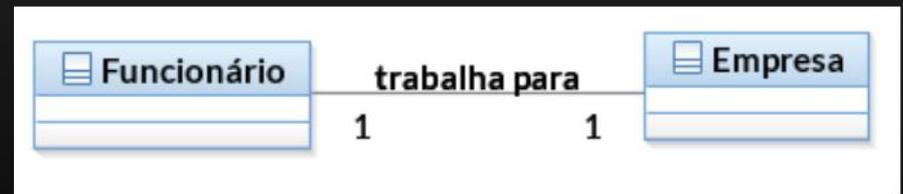
# O que é a multiplicidade (de uma associação)?

Nr de instâncias de uma classe que se relacionam com uma instância da outra.

*“O Funcionário só trabalha numa empresa; a Empresa tem pelo menos um Funcionário”*



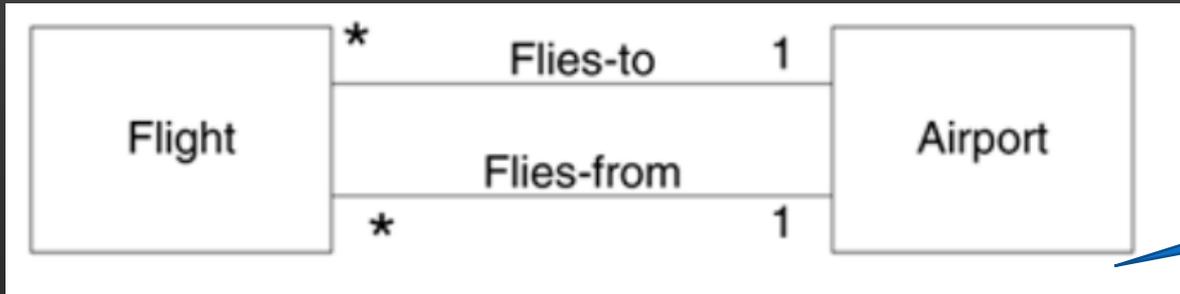
*“O Funcionário só trabalha numa empresa; a Empresa só tem um Funcionário”*



# Indicação de multiplicidade

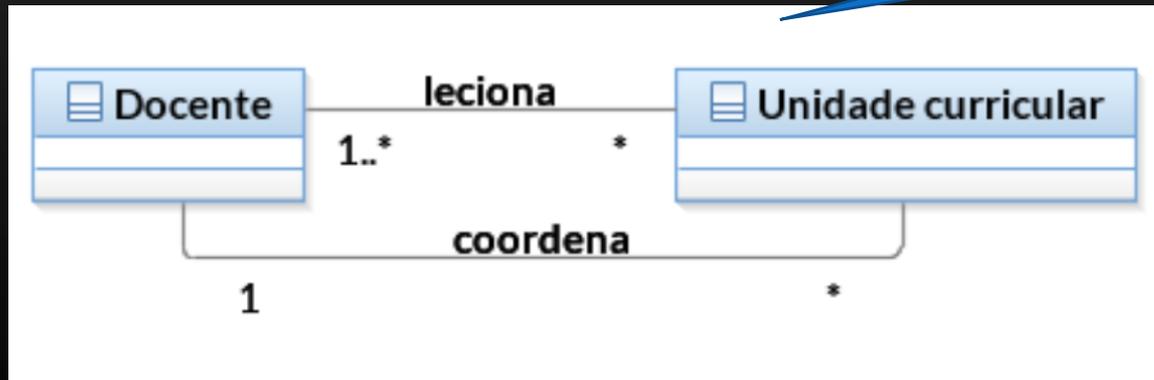
Exactly one	1		A department has one and only one boss.
Zero or more	0..*		An employee has zero to many children.
One or more	1..*		A boss is responsible for one or more employees.
Zero or one	0..1		An employee can be married to zero or one spouse.
Specified range	2..4		An employee can take from two to four vacations each year.
Multiple, disjoint ranges	1..3,5		An employee is a member of one to three or five committees.

# Múltiplas associações entre duas classes



Um voo: parte de um Aeroporto; chega a um Aeroporto.  
O “estado” de um Voo associa 2 objetos Aeroporto, por razões diferentes

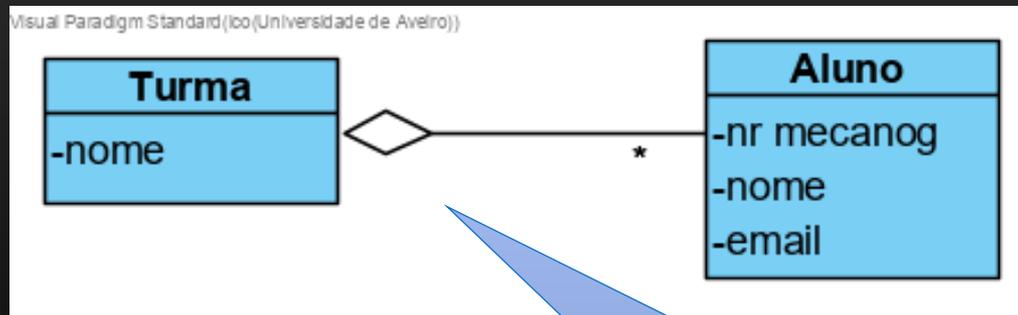
Um docente: leciona várias UC; coordena várias UC.



# O que é uma agregação?

É uma forma especial de associação que modela uma relação de todo-parte, entre o agregador (“contentor”) e as suas partes constituintes

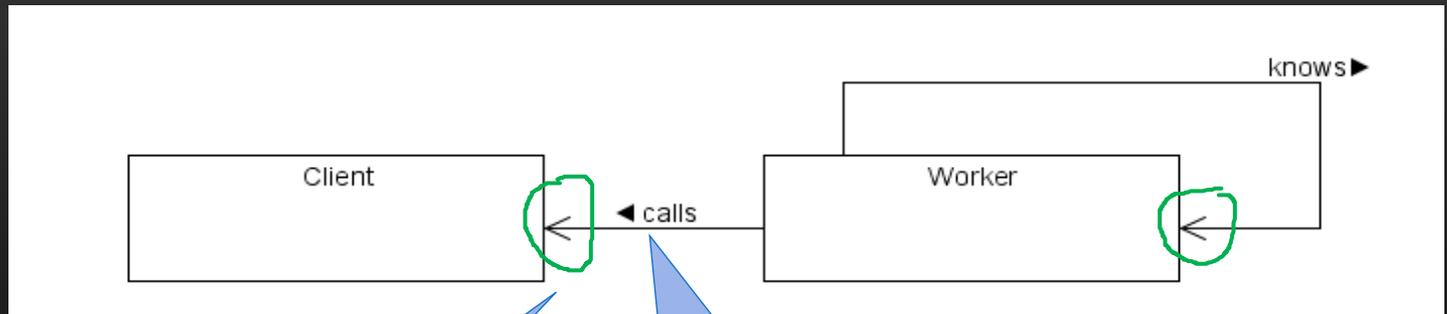
Deve ser natural ler-se “É parte de...” (sentido parte → todo); se for “forçado”, usar a associação normal



Um objeto *Turma* agrega vários objetos *Aluno*.

# O que é a navegabilidade?

Indica a possibilidade de navegar de uma classe de partida para uma classe de chegada, usando a associação



Em que sentido é que a associação é navegável/aplicável? (Se omissão → ambos)

Notação auxiliar: em que sentido se deve ler a etiqueta. (Worker > calls > Client. (Se omissão: cima p/ baixo; esquerda p/ direita))



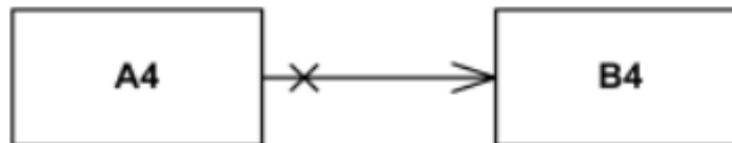
*Both ends of association have **unspecified navigability**.*



*A2 has **unspecified navigability** while B2 is **navigable from A2**.*



*A3 is **not navigable from B3** while B3 has **unspecified navigability**.*



*A4 is **not navigable from B4** while B4 is **navigable from A4**.*

Os dois primeiros casos são as situações mais comuns.

# O que é a generalização (=herança)

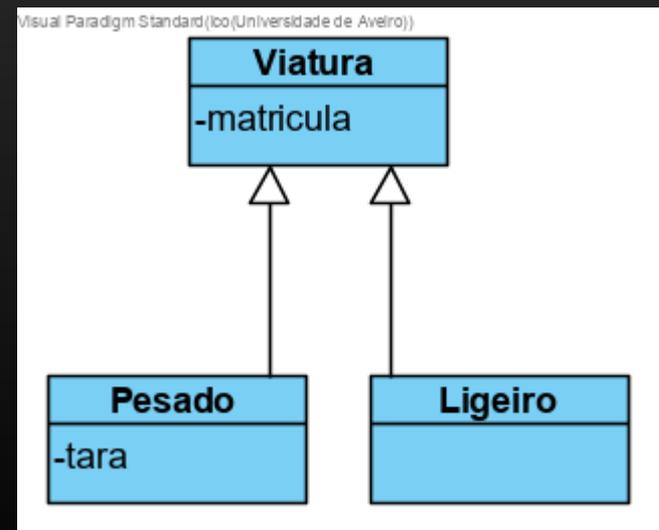
É relação entre classes em que uma especializa a estrutura e/ou comportamento de outra, partilhando todas as características

Define uma hierarquia em que a subclass recebe as características da superclasse

A subclasse pode sempre ser usada onde a superclasse é usada, mas não ao contrário.

Pode ler-se “é um tipo de”

(Um *Pesado* é um tipo de *Viatura*, com matrícula e tara.)



# O que é passado à subclasse?

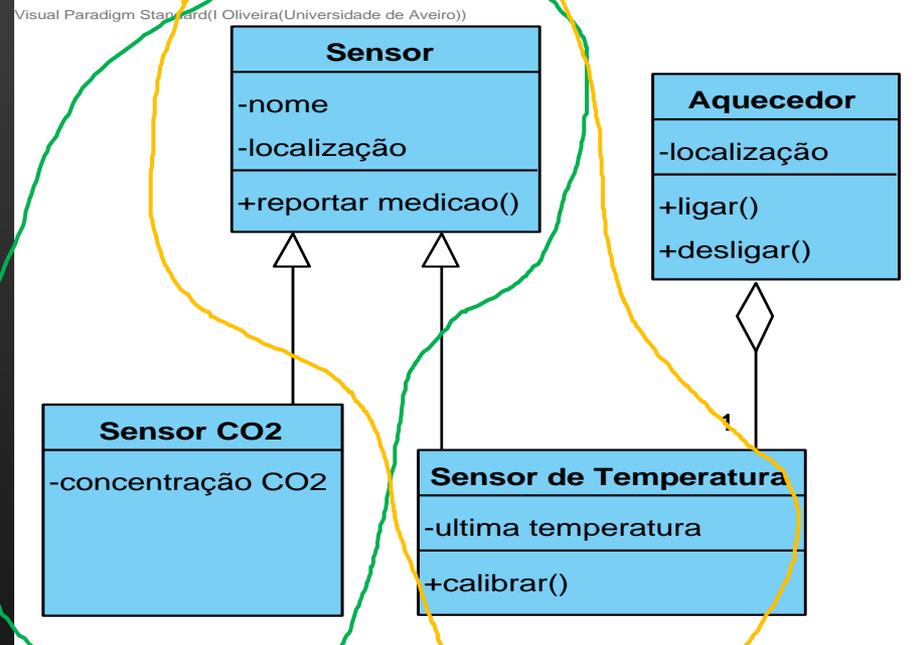
A subclasse herda os atributos, operações e relacionamentos da superclasse

A subclasse pode:

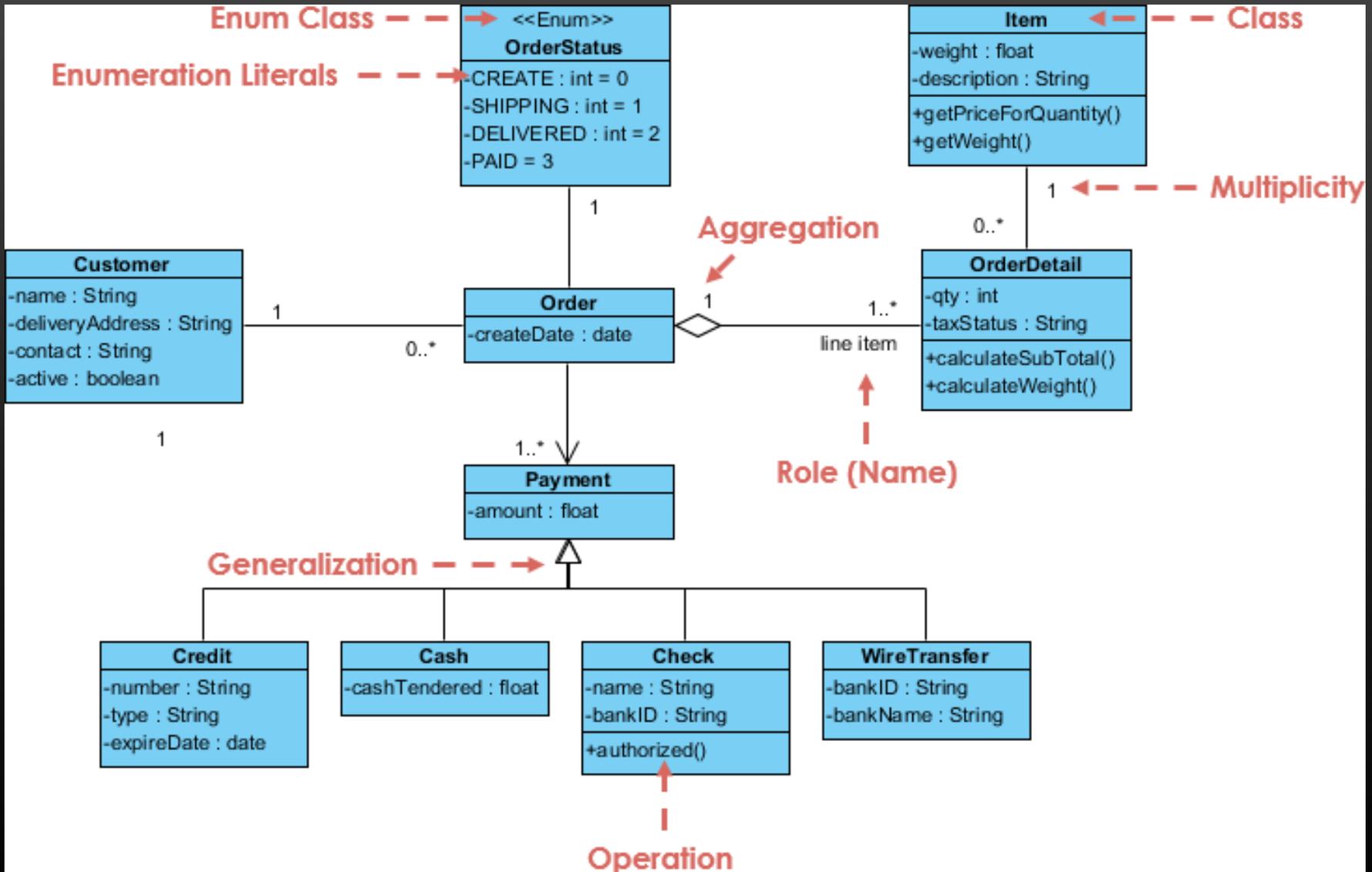
Adicionar mais atributos, operações e relacionamentos à base herdada

Redefinir as operações da superclasse

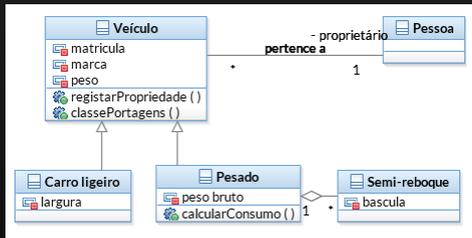
A herança põe em evidencia as características comuns entre classes



# Síntese da notação básica do diagrama de classes



# Como interpretar este modelo?



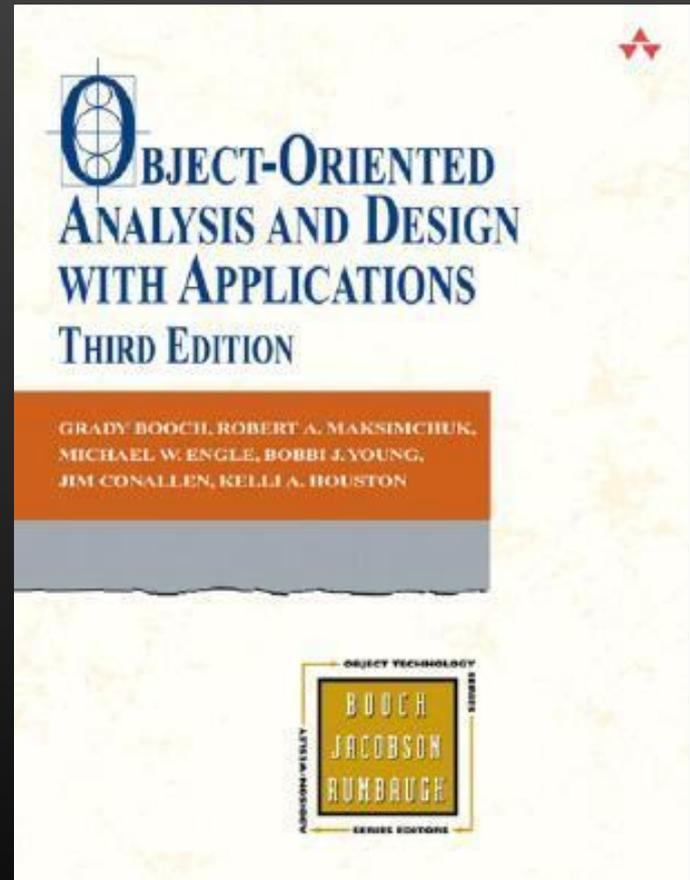
# Decomposição por objetos está na base da Análise e Desenho por Objectos (OOA, OOD)

AOO/AOD: encaramos o mundo como um conjunto de entidades autónomas que colaboram para realizar algum comportamento de nível superior.

Assim, as operações não existem como um pequeno algoritmo independente; pelo contrário, as operações estão definidas no contexto de objetos delimitados. Cada objeto na solução (software) encarna o seu próprio comportamento único e cada um modela algum objeto/conceito no mundo real.

Nesta perspetiva, um objeto é simplesmente uma entidade que tem um estado (guarda informação) e exibe algum comportamento bem definido.

Os objetos fazem as coisas, e a forma de solicitar que façam as operações é enviando-lhes mensagens. Como a nossa decomposição é baseada em objetos e não em algoritmos, chamamos a isto uma decomposição por objetos



<http://www.drdoobs.com/windows/software-complexity-bringing-order-to-ch/199901062>

# Exercício: domínio da gestão de projetos

- a) Os projetos têm um título descritivo, uma data de início e de fim.
- b) Um Projeto é implementado por um determinado Departamento.
- c) Cada Projeto define várias Tarefas que, por sua vez, podem estar organizadas em sub-tarefas.
- d) Um projeto tem uma equipa atribuída; as equipas têm vários colaboradores e um gestor.
- e) Os projetos externos são contratados por um cliente. Os internos, são pedidos por um Departamento.
- f) As tarefas têm data de início e duração estimada.

# Referências

[PRE'10] Pressman, R. S. (2010). Software Engineering: a practitioners approach (seventh ed). McGraw Hill.

→ Chap. 5

[DEN'15] Dennis, A., Wixom, B. H., & Tegarden, D. (2015). Systems analysis and design: An object-oriented approach with UML. John Wiley & Sons.

→ Chap. 3

[LAR'04] Larman, C. (2004). Applying UML and Patterns: An Introduction to Object Oriented Analysis and Design and Iterative Development, 3<sup>rd</sup> ed. Pearson Education.

→ chap. 5.