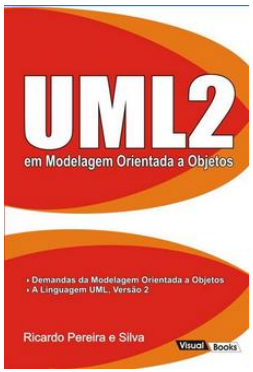
Unidade 8

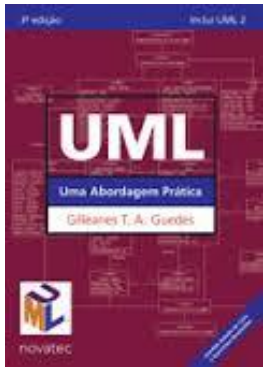
Modelagem Baseada em Classes

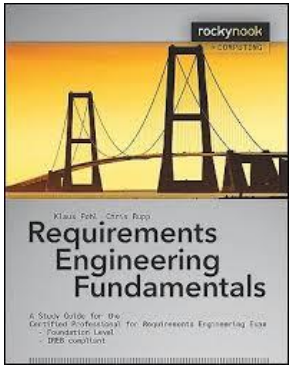
Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUSP

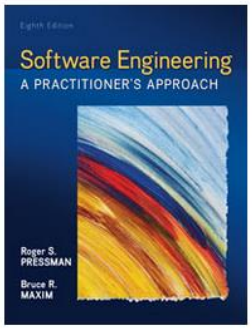
Bibliografia

- **Software Engineering – A Practitioner's Approach – Roger S. Pressman – Eight Edition – 2014**
- **Software Engineering – Ian Sommerville – 10th edition - 2015**
- Engenharia de Software – Uma abordagem profissional – Roger Pressman - McGraw Hill, Sétima Edição - 2011
- Engenharia de Software – Ian Sommerville – Nona Edição – Addison Wesley, 2007
- **UML – Uma abordagem prática – Gilleanes T. A. Guedes – 2004 - Novatec**
- **Fundamentos de Engenharia de Requisitos, Pohl K.,Rupp C. - IREB – T&M , 2012**
- **UML 2 em Modelagem Orientada a Objetos – Prof. Ricardo Pereira e Silva – UFSC, Visual Books, 2007**
- **Como modelar com UML 2 – Prof. Ricardo Pereira e Silva – UFSC, Visual Books, 2009**





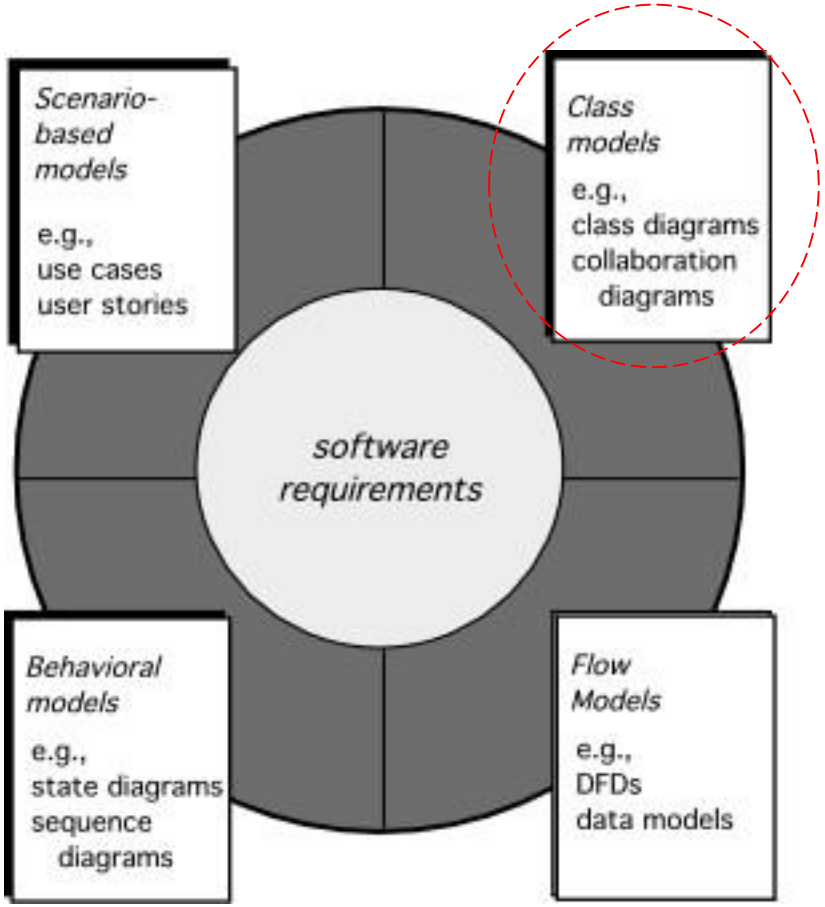




Software Engineering: A Practitioner's Approach, 8/e

Elementos da Modelagem de Requisitos

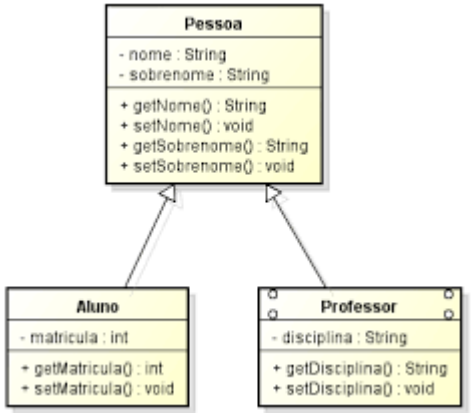
- Modelagem baseados em Cenários.
- **Modelagem baseada em Classes.**
- Modelagem Comportamental.
- Modelagem Orientada a Fluxo.



Modelagem baseada em Classes

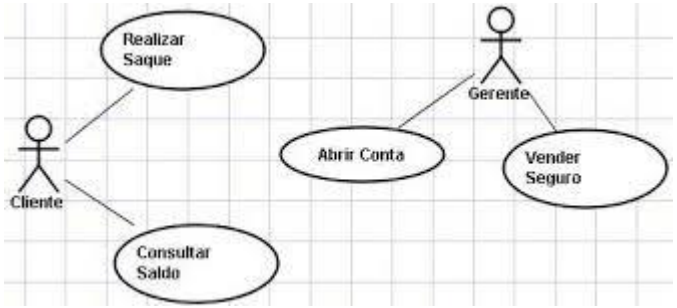
Representam:

- Os **objetos** que o software irá manipular;
- As **operações** (também chamadas de métodos ou serviços) que serão executadas pelos objetos;
- **Relacionamento** entre objetos;



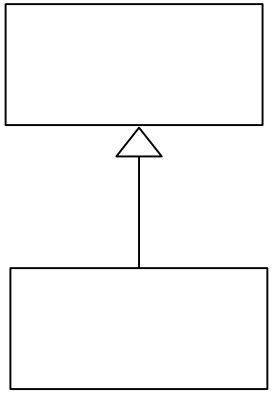
Classes de Análise

- ✓ Classes obtidas por meio dos **cenários de uso** desenvolvidos como parte da modelagem de requisitos (Casos de Uso);
- ✓ Classes podem ser obtidas por meio de "**análise sintática**" dos casos de uso do sistema a ser construído;
- ✓ Sublinham-se os substantivos (nomes) e verbos são colocados em itálico;
- ✓ Listam-se as classes potenciais para incluí-las no modelo de análise.



Casos de Uso

Cenários



Classes

Classes Potenciais

- ✓ Yourdon (91) sugere 6 características de seleção que deveriam ser usadas à medida que se considera cada classe potencial para inclusão no modelo de análise.

- *Retained information.* The potential class will be useful during analysis only if information about it must be remembered so that the system can function.
- *Needed services.* The potential class must have a set of identifiable operations that can change the value of its attributes in some way.
- *Multiple attributes.* During requirement analysis, the focus should be on "major" information; a class with a single attribute may, in fact, be useful during design, but is probably better represented as an attribute of another class during the analysis activity.
- *Common attributes.* A set of attributes can be defined for the potential class and these attributes apply to all instances of the class.
- *Common operations.* A set of operations can be defined for the potential class and these operations apply to all instances of the class.
- *Essential requirements.* External entities that appear in the problem space and produce or consume information essential to the operation of any solution for the system will almost always be defined as classes in the requirements model.

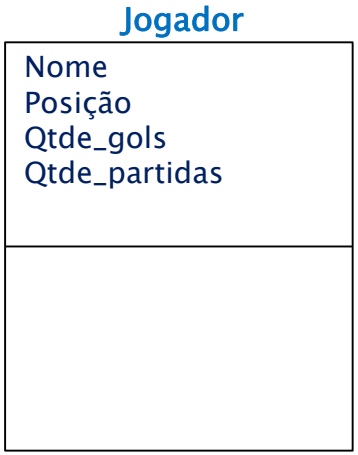
Atributos de Classe

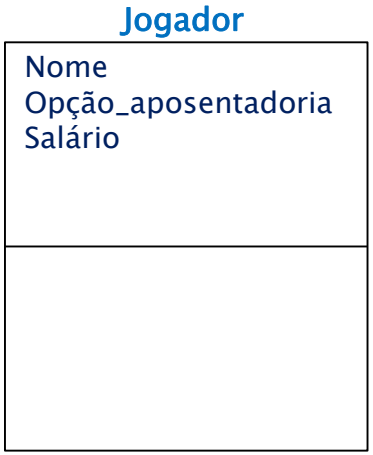
- ✓ Descrevem uma classe selecionada para ser incluída no modelo de análise;
- ✓ São os atributos que definem uma classe;
- ✓ Esclarecem o que a classe representa no contexto do espaço de problema.
- ✓ Exemplo: **Jogador em um sistema de estatística de jogo:**

Atributos: nome, posição, número de gols, número de partidas, etc

Jogador em um sistema de aposentadoria de jogadores:

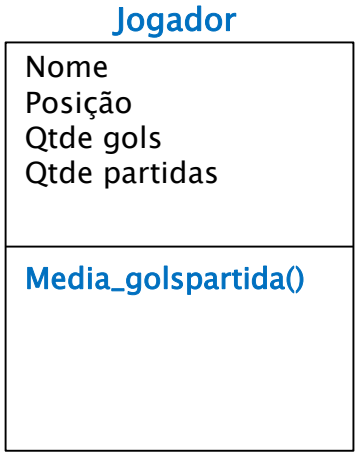
Atributos: nome, opções de aposentadoria, salário médio, etc

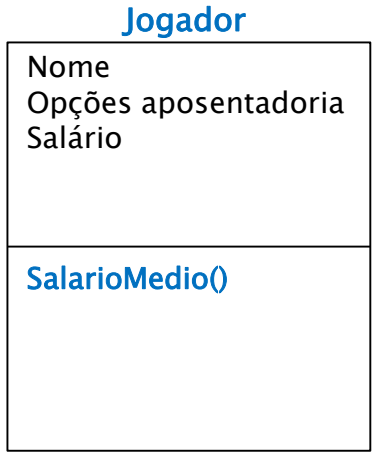




Definição das Operações

- ✓ Definem o comportamento de um objeto;
- ✓ Podem ser divididas em quatro grandes categorias:
  - Operações que manipulam dados;
  - Operações que realizam cálculos;
  - Operações que pesquisam o estado de um objeto;
  - Operações que monitoram objetos (eventos de controle).
- ✓ Operações também, como primeira interação, podem ser obtidas dos cenários de **casos de uso** (**verbos** podem representar operações)

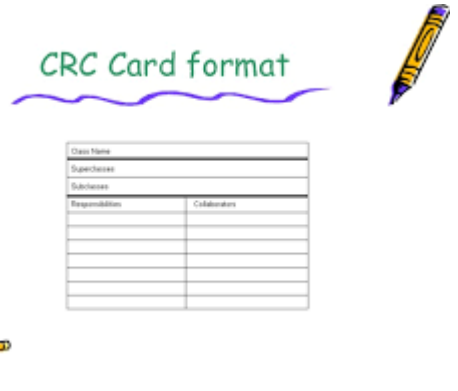




Fonte: Engenharia de Software: Uma abordagem Profissional – Pressman R. S. – McGraw Hill, Sétima Edição

Modelagem CRC

- ✓ **CRC** (Classe – Responsabilidade – Colaborador) fornece uma maneira simples de se identificar e organizar as classes que são relevantes para os requisitos do software.
- ✓ **Responsabilidades** são os atributos e as operações relevantes para a classe.
- ✓ **Colaborador** – Classes úteis para fornecer a uma classe informações necessárias para completar uma responsabilidade.

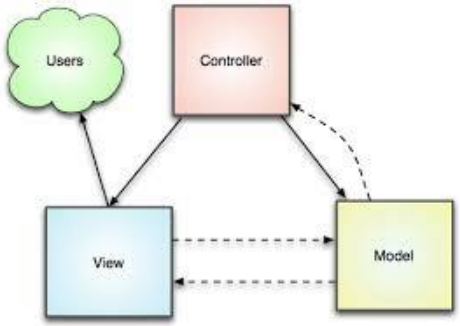


Taxonomia de Classes

- ✓ **Classes de Entidades** - Também chamadas de classes de negócio. Representam dados que persistem ao longo de todo o ciclo de vida do software.

- ✓ **Classes de Fronteira –** Usadas para se criar a interface da aplicação (Telas interativas ou relatórios impressos).

- ✓ **Classes de Controle** – Criação ou atualização de objetos. Instanciação de objetos. Comunicação entre objetos. Validação de dados transmitidos entre objetos ou entre o usuário e a aplicação. (Em geral, são consideradas na fase de projeto do software).



Fonte: Engenharia de Software: Uma abordagem Profissional – Pressman R. S. – McGraw Hill, Sétima Edição

Responsabilidades - Diretrizes

- ✓ **Inteligência do sistema deve ser distribuída pelas classes** – Cada objeto conhece e faz apenas algumas poucas coisas. Isso aumenta a **coesão**, facilita a manutenção e reduz o impacto dos efeitos colaterais devido a mudanças.

- ✓ **Responsabilidades devem ser declaradas da forma mais genérica possível –** Operações **genéricas** devem ser definidas no topo da hierarquia de classes (aplicáveis à subclasses).

- ✓ **Informação e comportamento relativos devem residir na mesma classe** – Para atender ao princípio de **encapsulamento** da Orientação a Objetos.

- ✓ **Informações sobre um item devem estar em única classe e não distribuídas em várias classes** – Se as informações forem distribuídas, o software se torna mais difícil de ser mantido e testado.

- ✓ **Quando apropriado, responsabilidades devem ser compartilhadas entre classes relacionadas** - **Reusabilidade**

Fonte: Engenharia de Software: Uma abordagem Profissional – Pressman R. S. – McGraw Hill, Sétima Edição

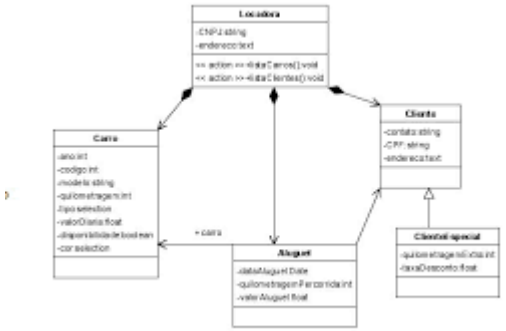
Colaborações de Classes

- ✓ Classes podem cumprir responsabilidades por meio de **colaboração** com outras classes.
- ✓ Um objeto **colabora** com outro se, para cumprir uma responsabilidade ele precisar enviar **mensagens** a um outro objeto.
- ✓ Colaborações são identificadas determinando-se se uma classe pode ou não cumprir cada responsabilidade por si só. Caso não possa, ela precisa **interagir** com outra classe. Daí a **colaboração**.
- ✓ Colaboração pode se efetivar por meio de 3 relacionamentos:
  - relacionamento **é-parte-de;** (classes agregadas)
  - relacionamento **tem-conhecimento-de**; (classe tem de adquirir informações de outra classe)
  - relacionamento **depende-de**; (herança)

Fonte: Engenharia de Software: Uma abordagem Profissional – Pressman R. S. – McGraw Hill, Sétima Edição

Diagramas de Classe (UML)

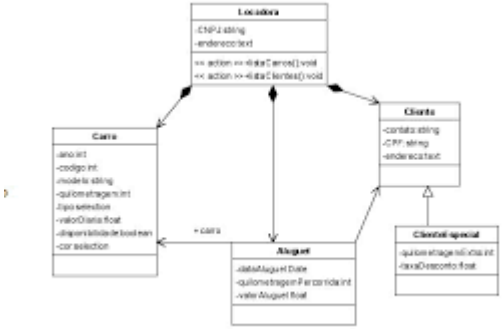
- ✓ Fornece uma **visão estática** ou **estrutural** do software;
- ✓ **Não** mostra a natureza dinâmica das comunicações entre os objetos das classes no diagrama.



Fonte: Engenharia de Software: Uma abordagem Profissional – Pressman R. S. – McGraw Hill, Sétima Edição

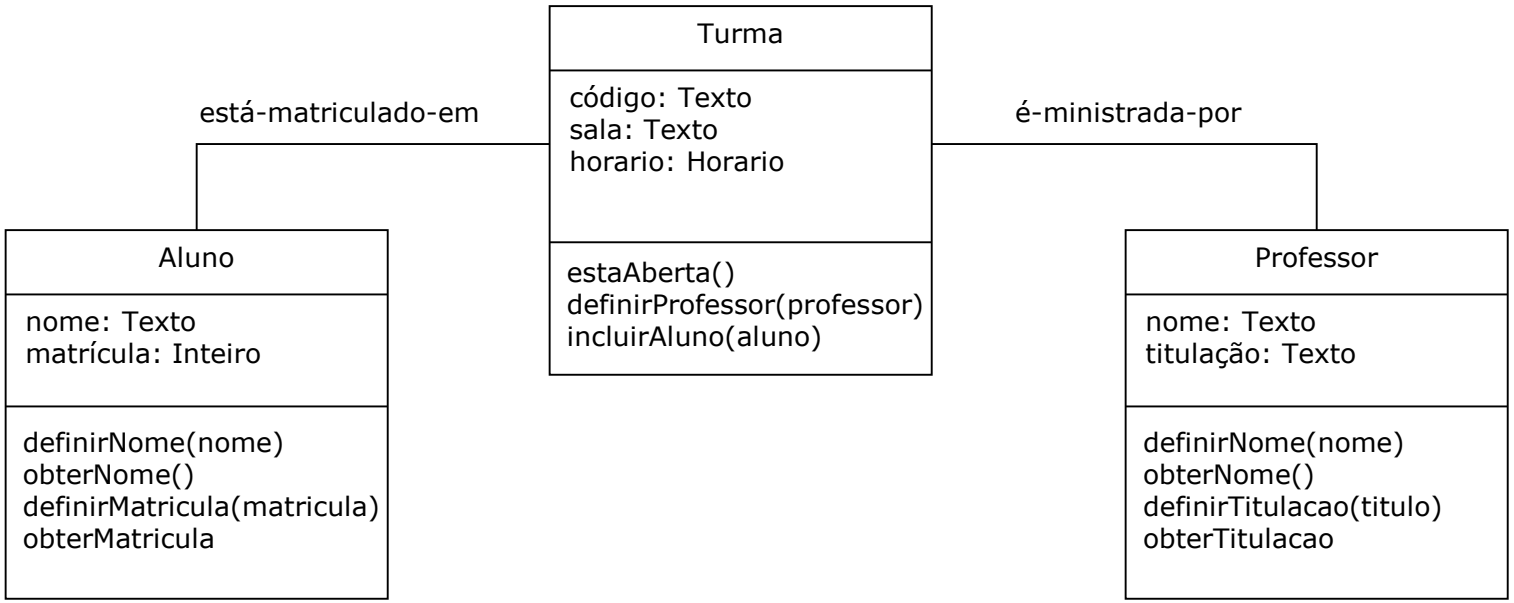
Finalidade do Diagrama de Classes

- ✓ Modelar os elementos de um programa orientado a objetos em tempo de desenvolvimento;
- ✓ Apresenta classes (com seus atributos e métodos) e relacionamentos envolvendo classes;
- ✓ Reflete a estrutura do código.



Introdução - Diagrama de Classes

- Exibe um conjunto de classes e seus relacionamentos.
- É o diagrama central da modelagem orientada a objetos.

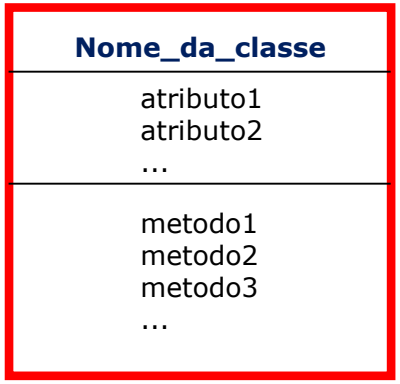


Elementos – Diagrama de Classes

- Classes
- Relacionamentos
  - Associação
    - ✓ Composição
    - ✓ Agregação
  - Especialização / Generalização

Classes

- Graficamente, as classes são representadas por retângulos incluindo nome, atributos e métodos.



- Recomenda-se nomeá-las de acordo com o vocabulário do domínio do problema.
- É comum adotar um padrão para nomeá-las. Em geral, adota-se o padrão de nomes com a primeira letra Maiúscula.

Atributos de Classes

- Representam o conjunto de propriedades ou características (estado) dos objetos de uma classe;
- Visibilidade:

+ ***público***:	visível em qualquer classe de qualquer pacote.
# ***protegido***:	visível somente para a classe possuidora do atributo ou suas subclasses.
- ***privado***:	visível somente para classe possuidora do atributo.

Exemplo:

+ codigo_produto: int

Recomendação de Visibilidade

- **Atributos**: Protegidos (#)
- Princípio da ocultação de informação do Paradigma Orientado a Objetos
- Atributos não devem ser definidos como públicos.
- **Métodos**, por sua vez, devem ser definidos como **públicos**.

Métodos de Classes

- Representam o conjunto de operações (comportamento) que a classe fornece
- Visibilidade:

+ *público*:	visível em qualquer classe de qualquer pacote.
# *protegido*:	visível somente para a classe possuidora do método ou suas subclasses.
- *privado*:	visível somente para classe.

Exemplo:

+ getCodigo() : int

Métodos Concretos

- São métodos compostos por uma assinatura (**signature**) e corpo com código
- Podem ser invocados em tempo de execução para cumprir a responsabilidade atribuída a ele.

Métodos Abstratos

- São métodos compostos apenas pela assinatura (**signature**)
- Correspondem à uma declaração de responsabilidade, mas sem capacidade de cumpri-la, em função da ausência de algoritmo.
- No diagrama de classes, são grafados em *itálico*.

Classe Concreta

- Possui exclusivamente métodos concretos
- Podem ser instanciadas, uma vez que todos os métodos cumprem a sua responsabilidade em tempo de execução.

Classe Abstrata

- Possui pelo menos um método abstrato.
- Identificador da classe grafado em *itálico.*
- Nem todas as responsabilidades da classe são materializadas.
- Nem todos os métodos possuem algoritmo definido.
- Não pode originar instâncias em tempo de execução.

Relacionamento de Classes

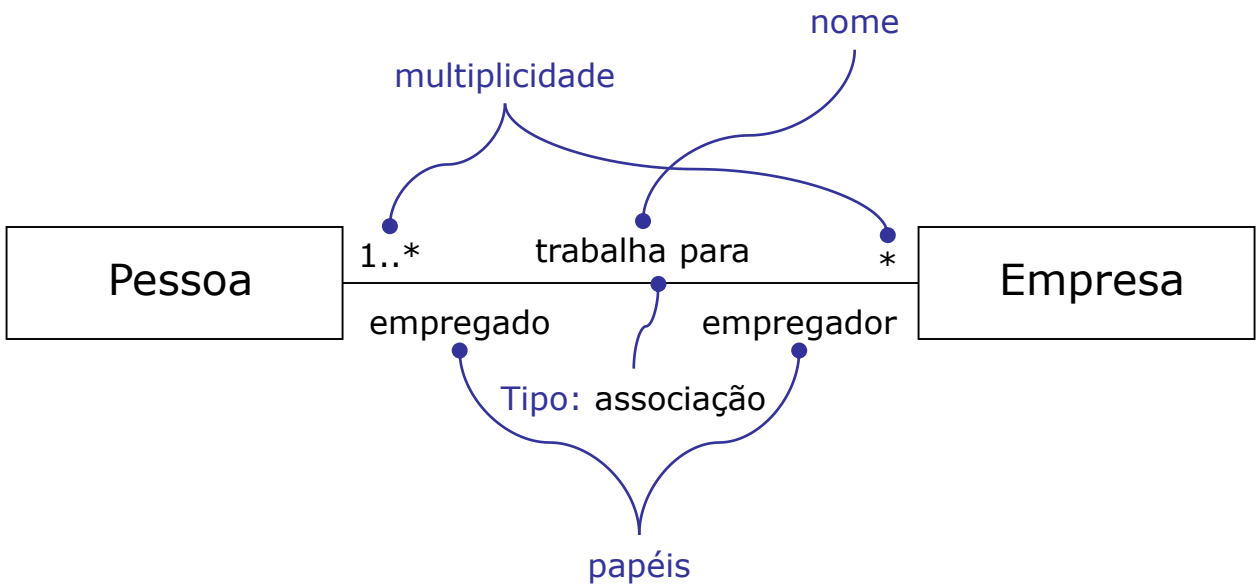
- As classes, em geral, possuem relacionamentos entre si, com o intuito de compartilhar informações e colaborarem umas com as outras para permitir a execução dos diversos processos de um sistema.

- Em muitos casos, duas classes de análise são relacionadas entre si de alguma maneira, de modo muito parecido como dois objetos de dados poderiam estar relacionados entre si. Na UML, esses relacionamentos são chamadas de ASSOCIAÇÕES.

- São descritos por meio de:

  - ✓ **Nome**: descrição dada ao relacionamento (faz, tem, possui,...)
  - ✓ **Multiplicidade**: 0..1, 0..*, 1, 1..*, 2, 3..7
  - ✓ **Tipo**: associação (agregação, composição), generalização e dependência
  - ✓ **Papéis**: desempenhados por classes em um relacionamento

Relacionamento de Classes

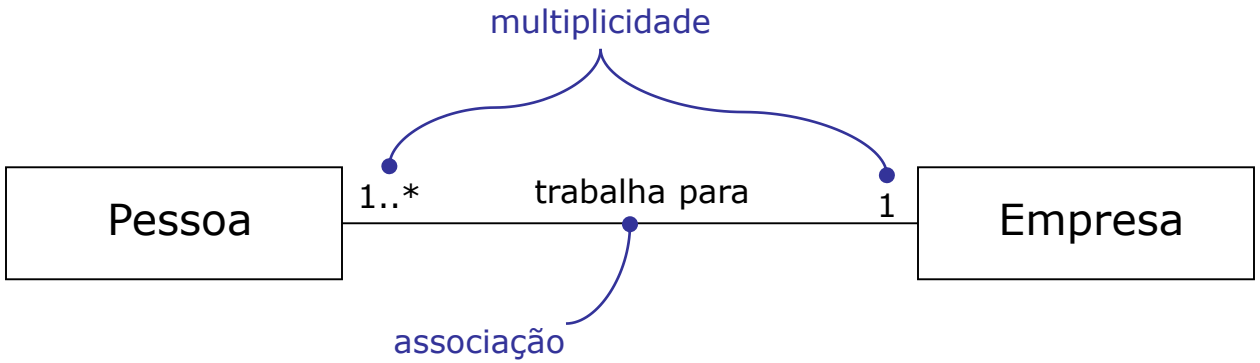


Relacionamento de Classes - Associação

- Uma associação é um relacionamento estrutural que indica que os objetos de uma classe estão **vinculados** a objetos de outra classe.
- Uma associação é representada por uma linha sólida conectando duas classes.

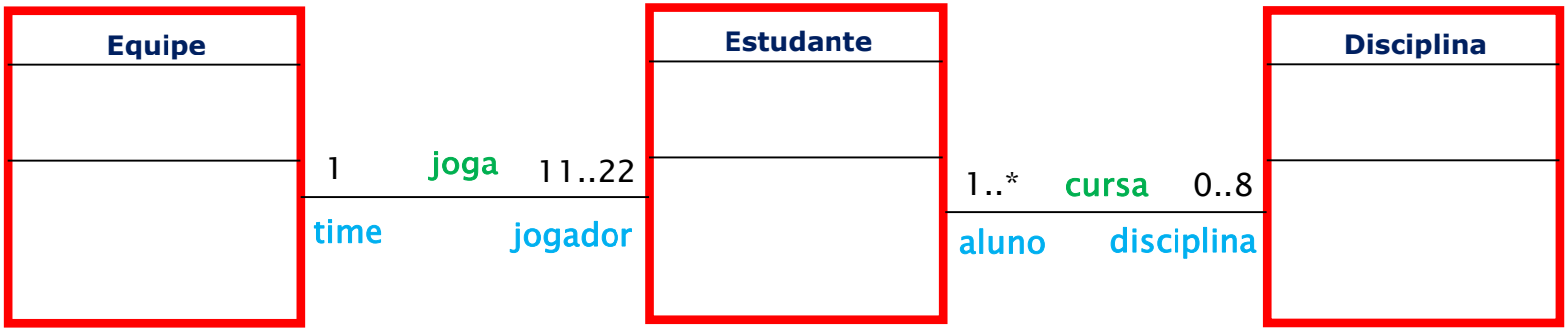
Relacionamento de Classes - Associação

- **Indicadores de multiplicidade:**
  - 1 Exatamente um
  - 1..* Um ou mais
  - 0..* Zero ou mais (muitos)
  - * Zero ou mais (muitos)
  - 0..1 Zero ou um
  - m..n Faixa de valores (por exemplo: 2..5)



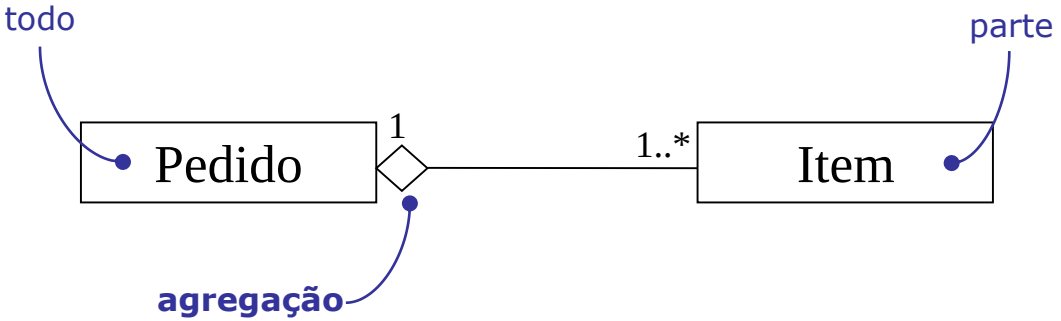
Exemplo - Associação

- Um Estudante pode ser:
  - um aluno de uma Disciplina e
  - um jogador da Equipe de Futebol
- Cada Disciplina deve ser cursada por no mínimo 1 aluno
- Cada aluno pode cursar de 0 até 8 disciplinas



Relacionamento de Classes - Agregação

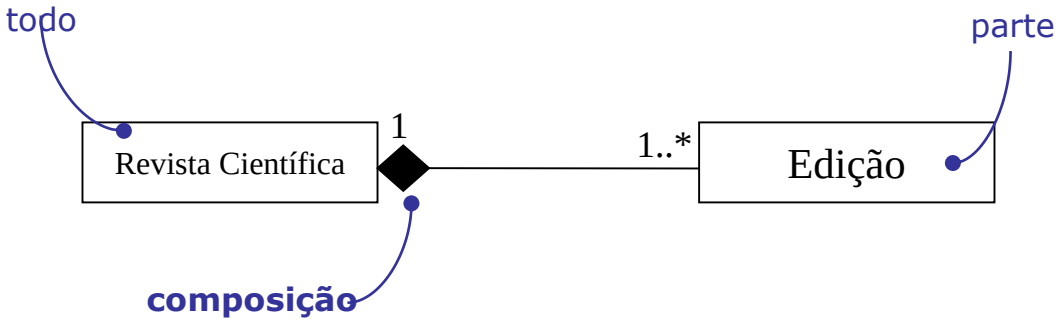
- ✓ É um tipo especial de **associação**;
- ✓ Utilizada para indicar "**todo-parte**";
- ✓ Representada por um losango na extremidade da classe que contém os objetos – todo;



- ✓ um objeto "parte" pode fazer parte de vários objetos "todo";
- ✓ Um item pode existir sem um pedido.

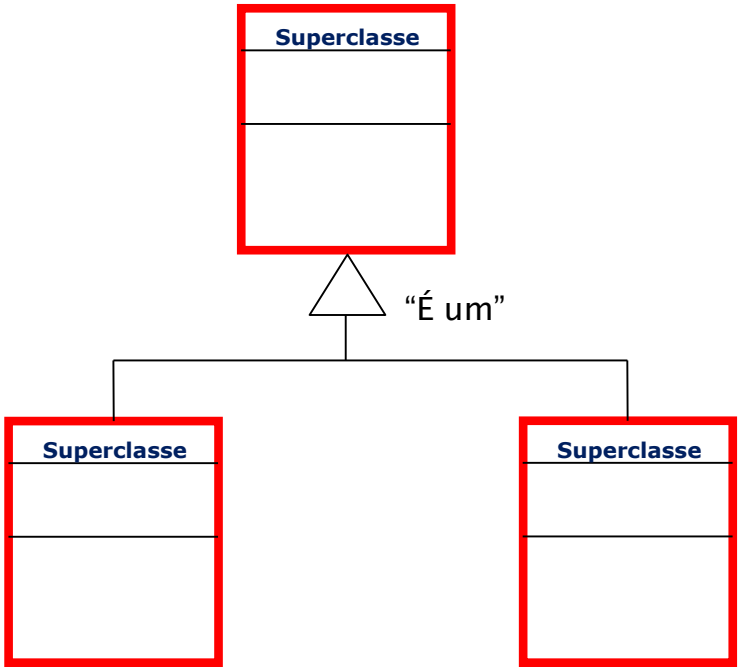
Relacionamento de Classes - Composição

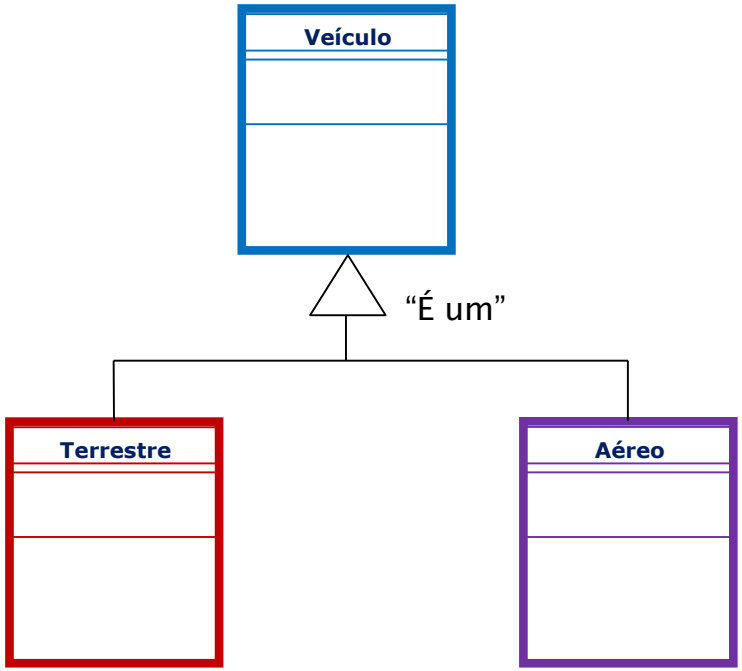
- ✓ É uma variante semanticamente mais "**forte**" da agregação;
- ✓ Os objetos "parte" só podem pertencer a um único objeto "todo" e têm o seu tempo de vida coincidente com o dele;
- ✓ O símbolo da composição é um losango preenchido;
- ✓ Quando o "todo" morre todas as suas "partes" também morrem.



Relacionamento de Classes - Generalização

- ✓ É um relacionamento entre itens gerais (superclasse) e itens mais específicos (subclasses)





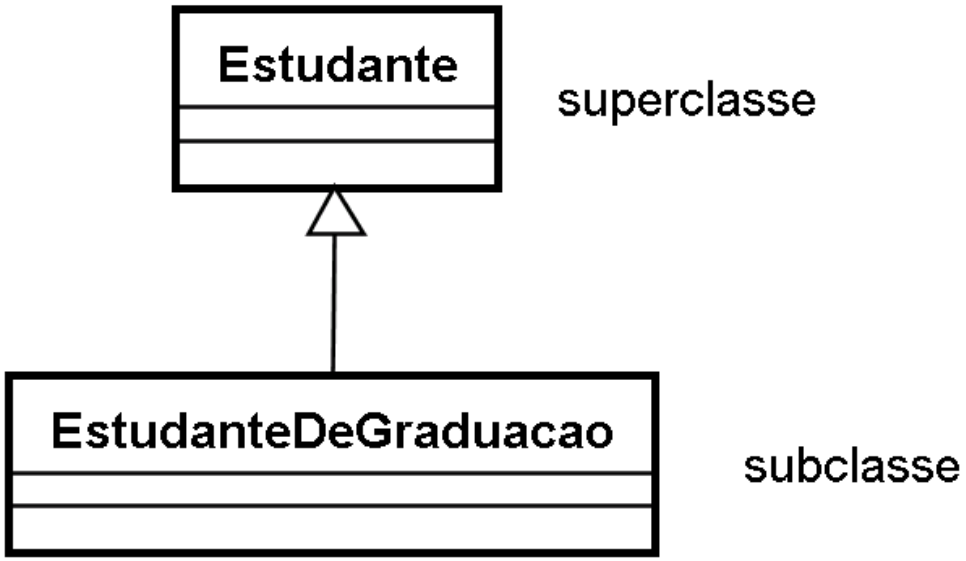
Herança

- ✓ Corresponde à uma relação de especialização entre DUAS classes.
- ✓ Uma delas corresponde a um conceito mais genérico.
- ✓ A outra, a um conceito mais específico.

Fonte: UML 2 em Modelagem Orientada a Objetos – Prof. Ricardo Pereira e Silva – UFSC, Visual Books, 2007

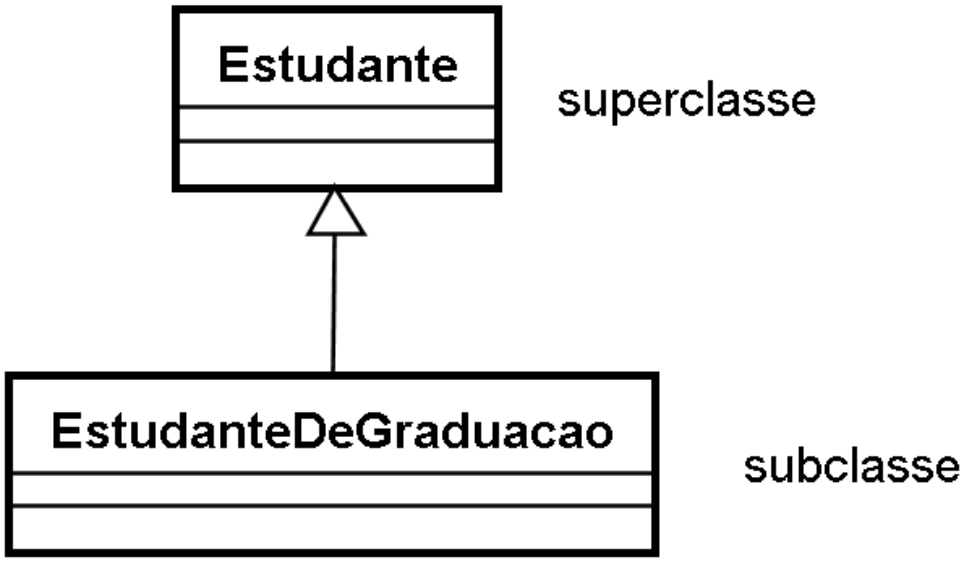
Frase específica da Herança

- ✓ <subclasse> é uma espécie de <superclasse> ;
- ✓ EstudanteDeGraduacao é uma espécie de Estudante;
- ✓ Se a frase não faz sentido, o relacionamento de herança não se aplica;
- ✓ Por exemplo: Leão é uma espécie de Elefante.



Herança

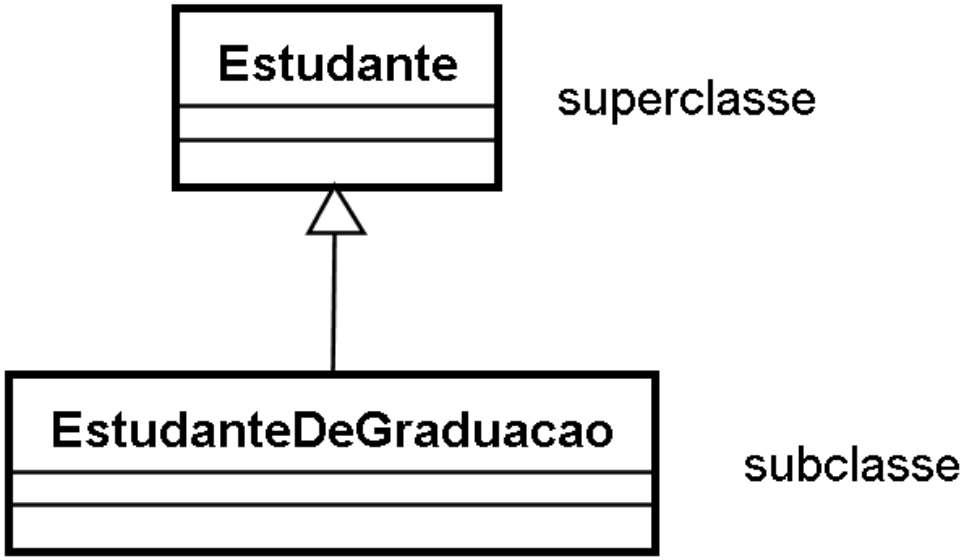
- ✓ Estudante de Graduação é uma espécie de Estudante;
- ✓ Mas, Estudante não é uma espécie de Estudante de Graduação.



Fonte: UML 2 em Modelagem Orientada a Objetos – Prof. Ricardo Pereira e Silva – UFSC, Visual Books, 2007

Semântica da Herança

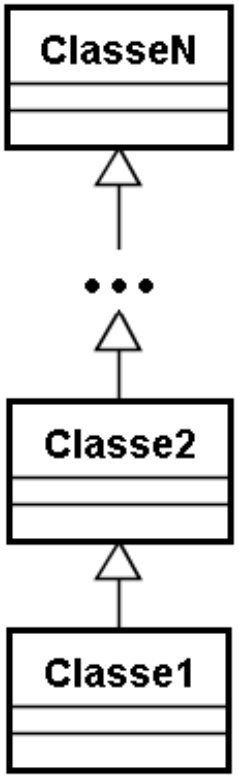
- ✓ Os atributos e métodos da superclasse são herdados pela subclasse;
- ✓ Os atributos e métodos da superclasse também fazem parte da subclasse. Como se tivessem sido definidos nela.



Fonte: UML 2 em Modelagem Orientada a Objetos – Prof. Ricardo Pereira e Silva – UFSC, Visual Books, 2007

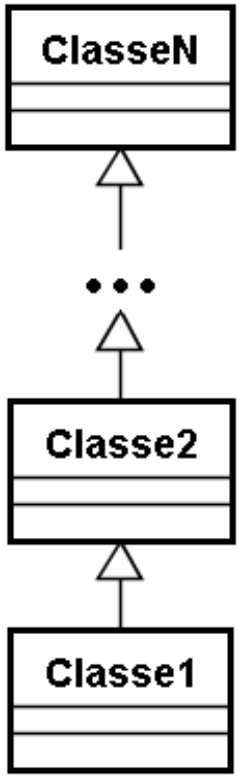
Níveis Hierárquicos de Herança

- ✓ Atributos e métodos da Classe N herdados por todas as classes da hierarquia de herança.



Herança de Atributos

- ✓ É inócuo definir em uma subclasse um atributo com mesmo nome de um atributo de uma superclasse.
- ✓ Em qualquer nível de hierarquia de herança, equivaleria à presença de mais de um atributo com o mesmo nome em uma mesma classe.
- ✓ Isso resulta em **inconsistência**.



Herança de Métodos

- ✓ Método em subclasse tem a mesma assinatura de método definido na superclasse (**Overriding** de método)
- ✓ Método definido na subclasse substitui o método herdado.