Diagrama de Classes UML

(Parte 1 - Básico)

Profa. Ana Paula Lemke

Última atualização do material em **01/05/2018**.
Em consonância com a versão 2.5.1 da UML (publicada em dez/2017).

O Diagrama de Classes

- O **Diagrama de Classes** é um esquema, um padrão ou um modelo que descreve muitas instâncias de objetos. Mostra a estrutura de classes, seus relacionamentos, atributos e métodos.

- É utilizado para modelar:
  - O vocabulário do sistema, que é a especificação das abstrações que estão contidas dentro do domínio do sistema, identificando suas responsabilidades.
  - As relações existentes entre as classes identificadas.
  - O esquema lógico do banco de dados do sistema.

Aplicações do Diagrama de Classes

- Durante a etapa de análise:
  - Utiliza-se diagrama de classes para identificar objetos (classes) do domínio do problema.

- Durante a etapa de projeto:
  - Utiliza-se diagrama de classes para representar objetos (classes) para a solução.

Aplicações do Diagrama de Classes

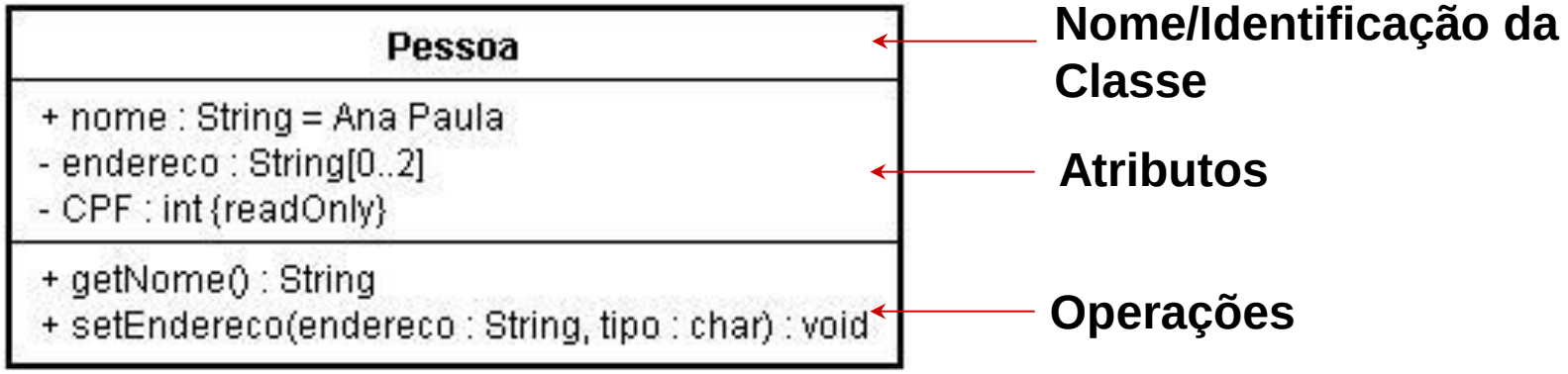
- **O modelo conceitual** (análise) representa as classes no domínio do problema. Não leva em consideração restrições inerentes à tecnologia a ser utilizada na solução de um problema.
- **O modelo de classes de especificação** (projeto) é obtido através da adição de detalhes ao modelo anterior conforme a solução de software escolhida (há detalhes sobre salvamento em banco de dados, por exemplo).
- **O modelo de classes de implementação** corresponde à implementação das classes em alguma linguagem de programação.

Fonte: BEZERRA, E. Princípio de análise e projetos de sistemas com UML. 2. ed. Rio de Janeiro: Elsevier, 2006. (página 11)

Notação do Diagrama de Classes

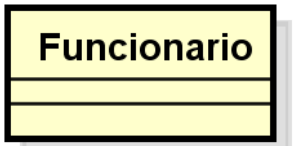
Classe

- Uma classe refere-se a descrição de um conjunto de objetos que compartilham os mesmos atributos, operações, relações e semântica.

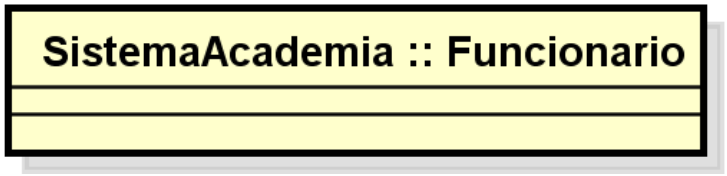


Nome da Classe

- O nome de uma classe distingue-a das outras classes do sistema.
- O nome da classe pode ser:
  - **Simples**



  - **Com caminho:** neste caso o nome da classe é precedido pelo nome do pacote em que a classe está.



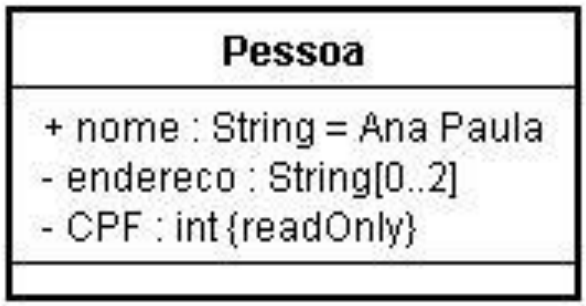
Atributos

Cada objeto de uma classe possui um **estado**, representado pelos **valores associados** a cada um dos **atributos** definidos pela sua classe.

Sintaxe para definição dos atributos:

[visibilidade] nome [: tipo] [multiplicidade] [= valor inicial] [{propriedades e restrições}]

Exemplo:



+ nome : String = “Ana Paula”
- endereço : String [0..2]
- CPF : int {frozen}

Atributos

[**visibilidade**] nome [: tipo] [multiplicidade] [= valor inicial] [{propriedades e restrições}]

- Público [+]: o que pode ser visto pelas operações de outras classes.
- Protegido [#]: o que pode ser visto apenas pelas operações da própria classe, por suas classes filhas e pelas classes que estiverem no mesmo pacote da classe.
- Privado [-]: o que pode ser visto apenas pelas operações da própria classe.
- Pacote [~]: o que pode ser visto pelas operações de outras classes dentro do mesmo pacote.

[visibilidade] nome [**: tipo**] [multiplicidade] [= valor inicial] [{propriedades e restrições}]

- É o tipo do atributo, que pode ser uma outra classe, uma interface ou um tipo primitivo da linguagem (como *int*, *String*, *boolean*).

Atributos

[visibilidade] nome [: tipo] **[multiplicidade]** [= valor inicial] [{propriedades e restrições}]

- **0..1**: atributo opcional.
- **1**: atributo obrigatório.
- **0..***: pode estar relacionado com nenhum ou vários objetos.
- **1..*** : pelo menos 1 objeto é obrigatório.
- **4..10**: deve estar relacionado com pelo menos 4 objetos e no máximo com 10.

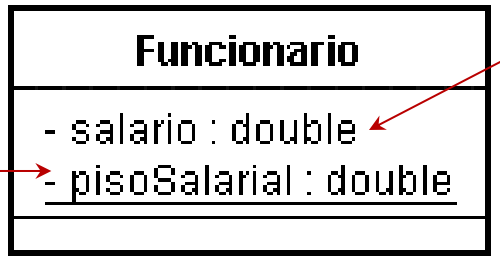
Atributos

[visibilidade] nome [: tipo] [multiplicidade] [= valor inicial] **[{propriedades e restrições}]**

- Propriedades
  - Compreendem um conjunto de *tags* pré-definidas que descrevem certas características do atributo. Exemplos: *read-only*, *add-only*, entre outros.
- Restrições
  - Permitem indicar uma ou mais restrições sobre o atributos. Podem ser escritas em linguagem natural ou com o uso de alguma gramática formal, como OCL.

Atributos de Objeto e de Classe

- Os atributos da classe (ou atributos estáticos) normalmente são utilizados na padronização de valores dentro do projeto/sistema.
- Atributos de classe ficam sublinhados.



Atributo de objeto: cada objeto encapsula o seu próprio valor de "salario".

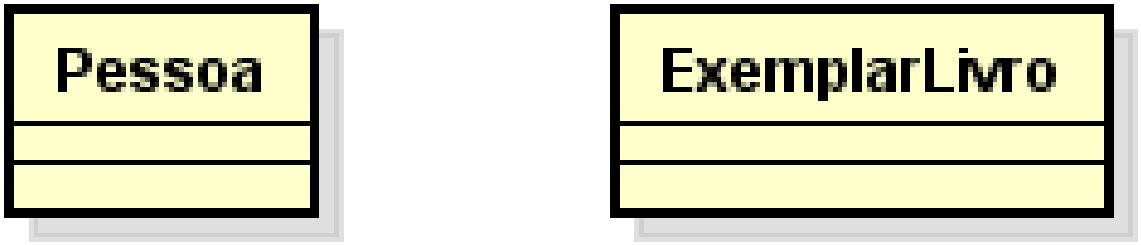
Atributo de classe: o "pisoSalarial" é o mesmo para todos os funcionários da empresa. O valor é armazenado em um local de memória único.

Atributo Derivado

- Um atributo derivado pode ser calculado a partir de outros atributos da classe.
- Este atributo é implícito na classe (ele não é definido explicitamente), mas há métodos para acessá-lo.
- Representação de um atributo derivado: “/”

Tarefa 1

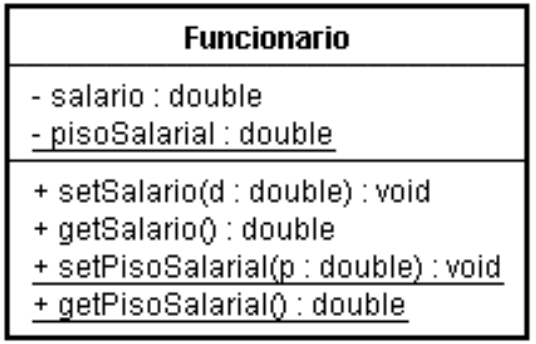
Esboce os atributos das classes de um Sistema de Biblioteca apresentadas abaixo:



Operações

- Uma operação é um serviço que pode ser requisitado a qualquer objeto da classe, afetando o seu comportamento.
  - A execução de um método por um objeto pode resultar na alteração do valor de seus atributos.

- Distinção entre operações de classes e objetos:
  - **Operação de objeto:** atua sobre um objeto (instância).
  - **Operação de classe:** atua sobre a classe (conjunto de objetos). Não é necessário ter uma instância da classe para acessá-la.

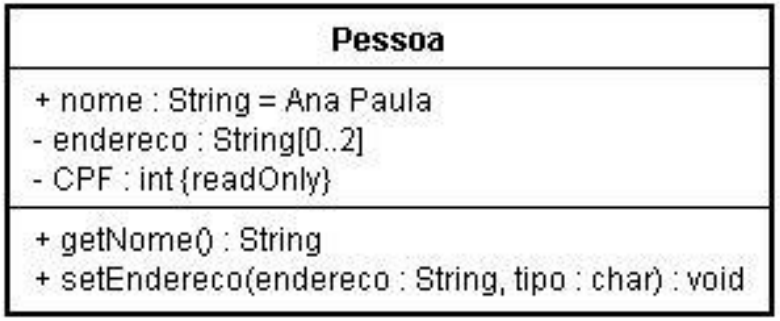


Operações

Sintaxe para Operações:

[visibilidade] nome [(lista-de-parâmetros)] [:tipo-retorno] [{propriedades}]

Exemplo:



+ getNome() : String

+ setEndereço (endereço: String, tipo: char): *void*

Operações

[**visibilidade**] nome [(lista-de-parâmetros)] [:tipo-retorno] [{propriedades}]

- Privado [−]: somente objetos da mesma classe podem chamar/invocar a operação.
- Pacote [~]: objetos de classes do mesmo pacote podem chamar/invocar a operação.
- Público [+]: qualquer objeto pode chamar a operação.
- Protegido [#]: somente objetos da própria classe, de sub-classes e de classes do mesmo pacote da classe podem chamar a operação.

[visibilidade] nome [**(lista-de-parâmetros)**] [:tipo-retorno] [{propriedades}]

- Compreende a lista de valores de entrada para a operação.
- Uma operação não precisa, obrigatoriamente, ter um ou mais parâmetros de entrada (a lista de parâmetros é opcional).

Operações

[visibilidade] nome [(lista-de-parâmetros)] **[:tipo-retorno]** [{propriedades}]

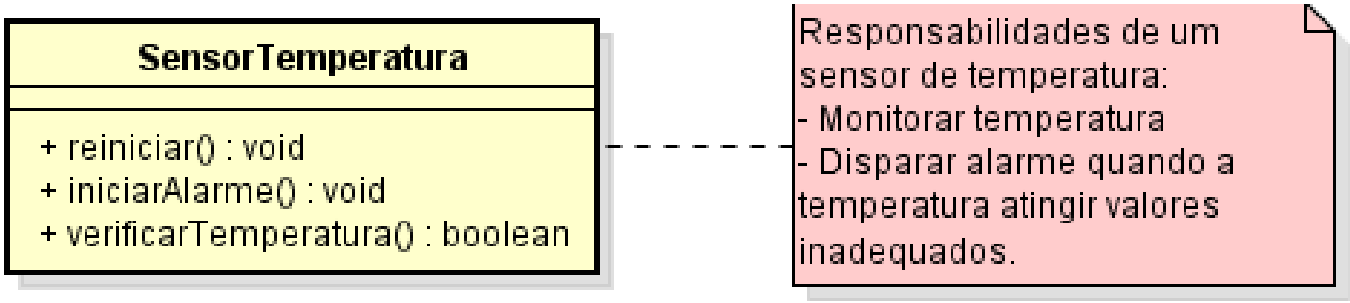
- O valor de retorno indica o tipo de dado que será informado como resultado da operação.
- Exemplo: o tipo de retorno do método getName() é uma *String*.

[visibilidade] nome [(lista-de-parâmetros)] [:tipo-retorno] **[{propriedades}]**

- Compreendem um conjunto de *tags* pré-definidas que descrevem certas características de uma operação. Exemplos: *isQuery*, *guarded*, *leaf*, entre outros.

Responsabilidades da Classe

- A responsabilidade de uma classe diz respeito as suas obrigações dentro do contexto do sistema.
  - Ao refinar o modelo, as responsabilidades de uma classe são traduzidas em um conjunto de atributos e operações que melhor atendam as suas obrigações.



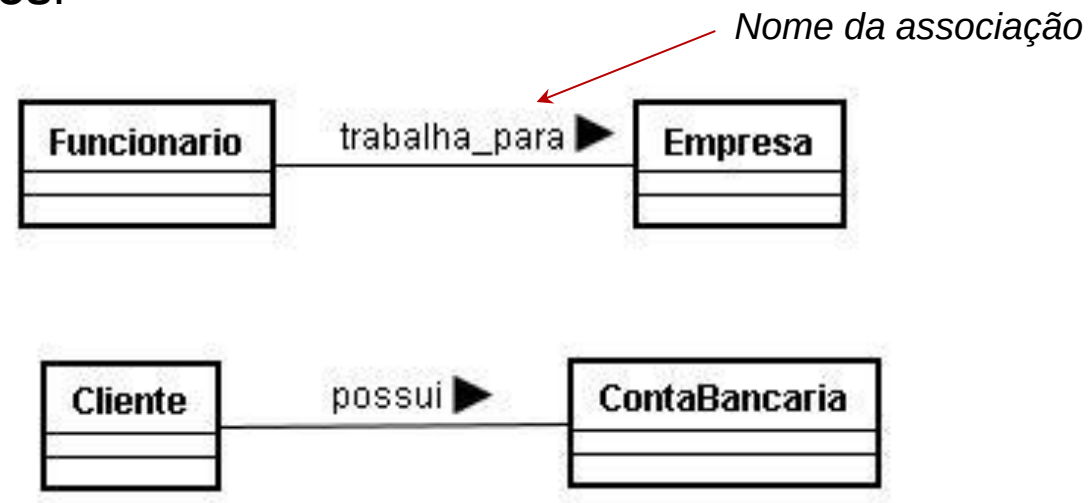
Relacionamentos entre Classes

- Os relacionamentos determinam conexões entre os objetos, fornecendo um caminho para a comunicação entre eles.

- Principais tipos de relacionamentos entre classes são:
  - Associação
  - Generalização 
  - Dependência

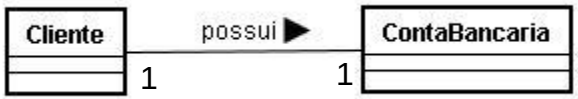
Associação

- Uma associação é um relacionamento **estrutural** que descreve uma ligação entre os objetos das classes.
- Uma associação pode ter um **nome**, que pode ser utilizado para descrever a natureza do relacionamento.
  - **Exemplos**:

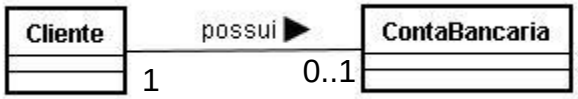


Associação - Multiplicidade

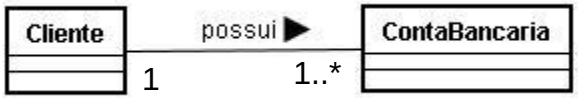
- (1-1): cliente possui sempre 1 (e somente 1) conta



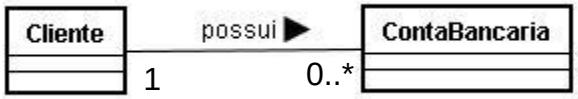
- (0-1): cliente pode possuir 1 (e no máximo 1) conta



- (1-N): cliente possui sempre 1 conta, podendo ter mais

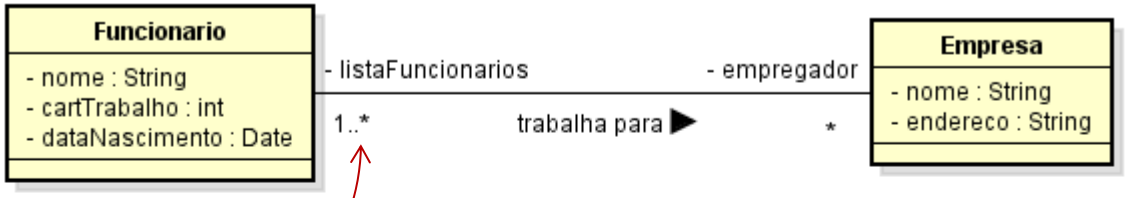


- (0-N): cliente pode possuir 1 conta, podendo ter mais

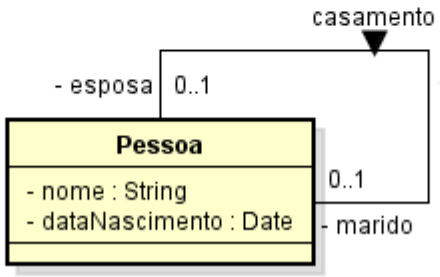


Associação - Papel (opcional)

- Descreve o “papel” de cada classe na associação.
- Chamado de “*Name of the association end*” a partir da UML 2.4.



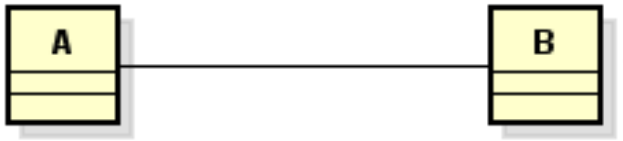
***Leitura**: Empresa tem uma lista de funcionários com um ou mais funcionários que são do tipo Funcionário*



***Auto associação**: quando a classe tem uma relação com ela mesmo.*

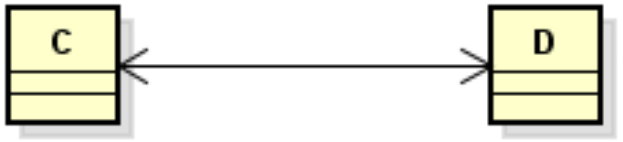
Associação - Navegabilidade

Associação com navegabilidade **não especificada** (não há setas nas pontas)

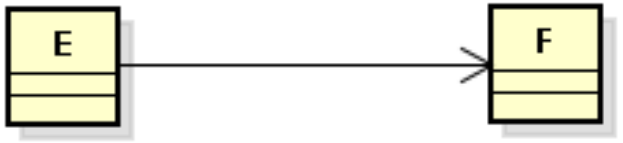


Geralmente são utilizadas como sinônimos nas empresas

Associação com **navegabilidade especificada** nas duas pontas

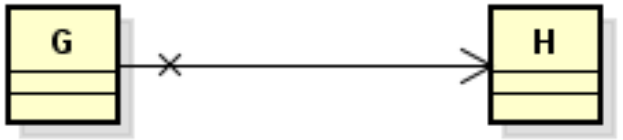


Associação com navegabilidade especificada em uma das pontas e não especificada em outra



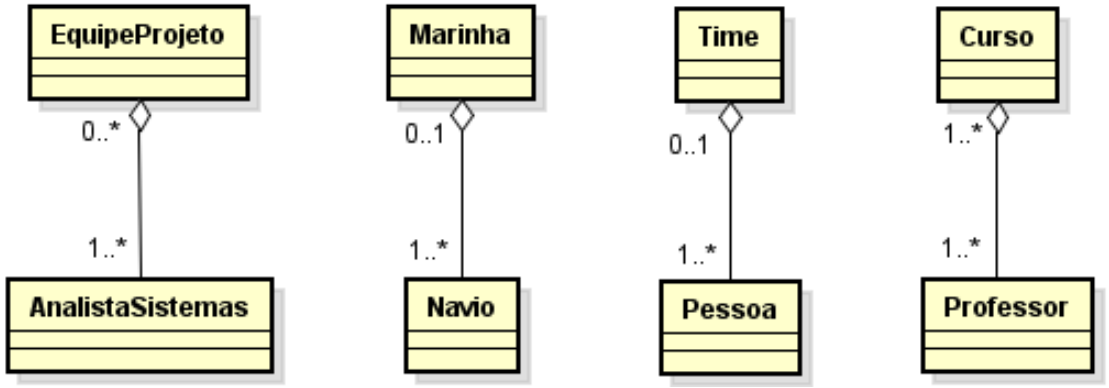
Geralmente são utilizadas como sinônimos nas empresas

Associação sem navegabilidade em uma das pontas (em G) e com navegabilidade especificada em outra (em H)



Associação - Agregação

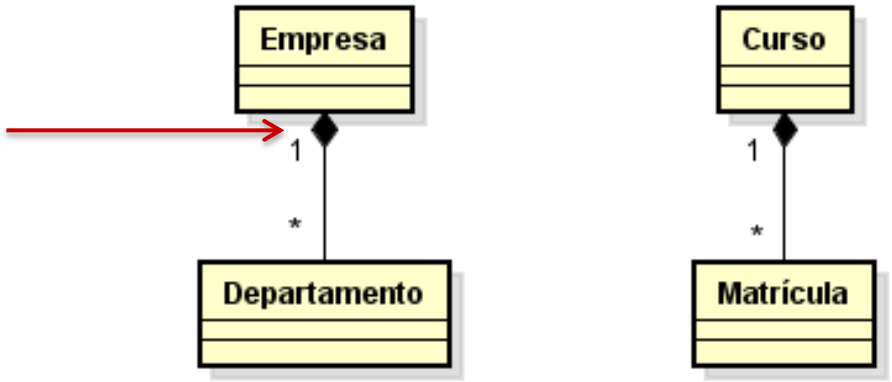
- A agregação é um tipo especial de associação.
- Indica que uma das classes do relacionamento **é uma parte** ou **está contida** em outra classe.
- Exemplos:



Associação - Composição

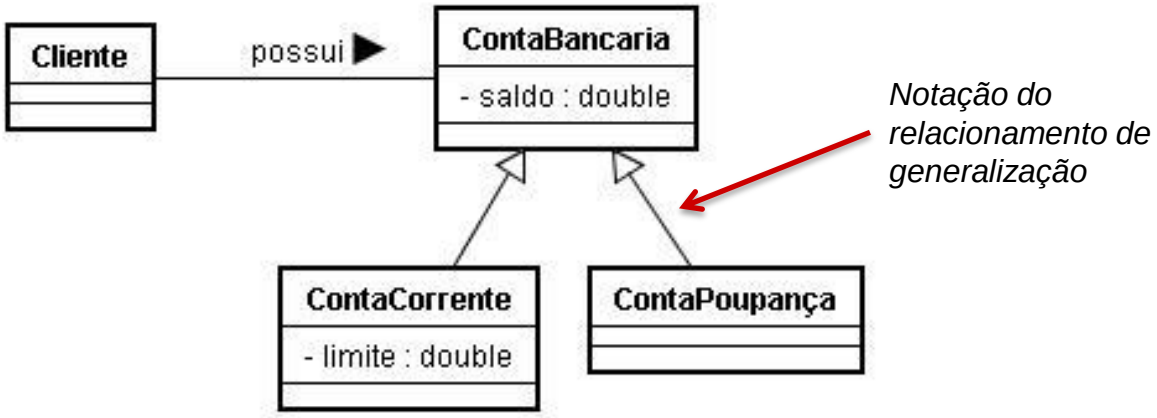
- Quando uma parte é criada, sua existência deve ser coincidente com o todo.
- Se o objeto da classe que contém for destruído, as classes da composição serão **destruídas juntamente**, já que as mesmas fazem parte da outra.

A classe onde fica ligado o losango preenchido representa o "**todo**". A multiplicidade do todo será sempre **1**.



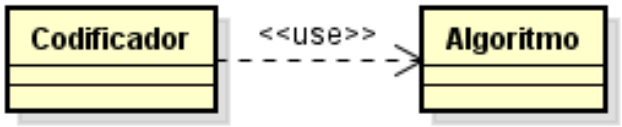
Generalização

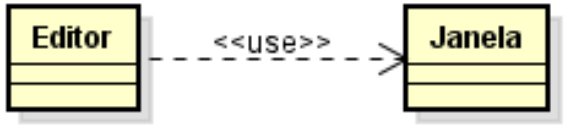
- A generalização é o compartilhamento de atributos, operações e relacionamentos entre classes com base em um relacionamento hierárquico (**superclasse** X **subclasse**).
  - Subclasses herdam os atributos e as operações (métodos) da superclasse, permitindo ainda modificações nos mesmos.
- **Exemplo**:



Dependência

- Estabelece um relacionamento entre dois itens, um **independente** e outro **dependente**. Uma mudança no item independente poderá afetar o item dependente.
- Uma relação de dependência é a forma mais fraca de relação, mostrando uma relação entre um **cliente** e um **fornecedor**, onde o cliente não tem conhecimento semântico do fornecedor.
- Em um diagrama de classes, relações de dependência são utilizadas quando deseja-se representar a utilização de uma classe por outra classe, como nos casos de:
  - Parâmetros de operações;
  - Uso no código de operações;
  - E outros.





Tarefa 2

Qual seria a multiplicidade:

- Classe Pessoa:
  - Atributo cônjuge?
  - Atributo endereço?
  - Atributo telefone?

- Classe Time de Futebol de Campo:
  - Atributo jogador?
  - Atributo goleiro?
  - Atributo técnico?

Tarefa 3

- Modele uma classe Aluno. Esta classe deve possuir como atributos o nome (*String*), as notas de N1, N2 e N3 (*double*), e a frequência (*int*). Além do construtor, objetos desta classe devem contar com os métodos *setN1*, *setN2*, *setN3* e *setFrequencia*, de forma a introduzir os respectivos valores a medida que o semestre evolui, e o método *getMedia*, que retorna a média do indivíduo.

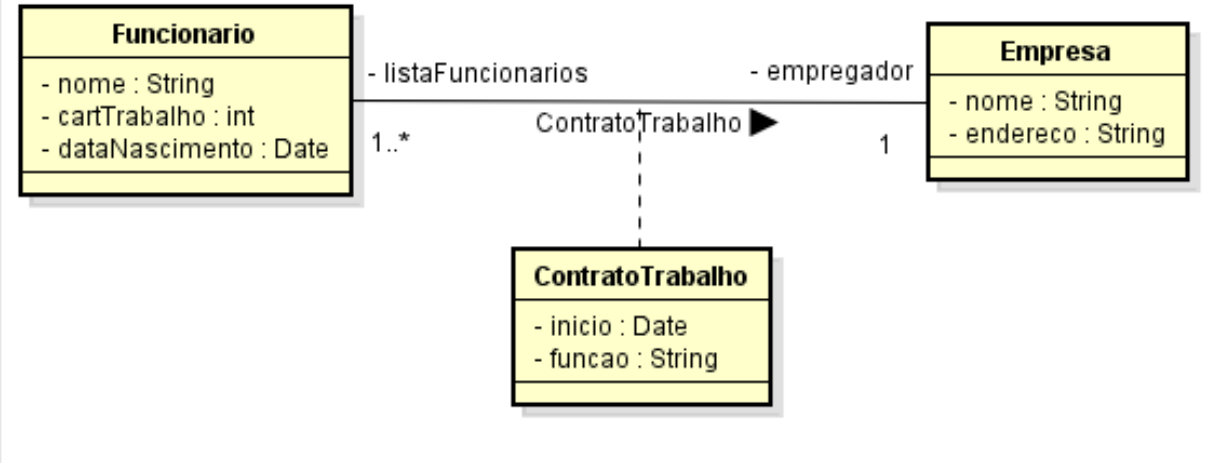
Diagrama de Classes UML

(Parte 2 - Avançado)

Última atualização do material em **01/05/2018**.
Em consonância com a versão 2.5.1 da UML (publicada em dez/2017).

Classe Associativa [1..3]

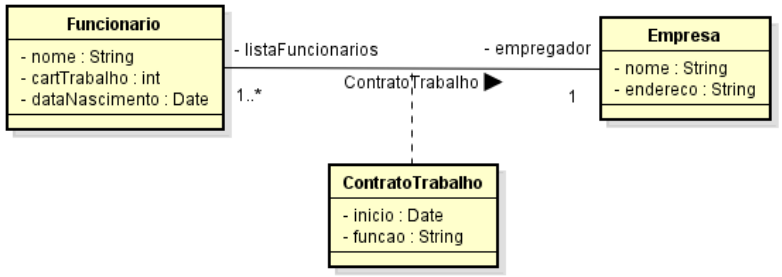
- Um classe associativa é derivada de uma associação para a qual seja necessário expressar as propriedades.
- **Exemplo**:



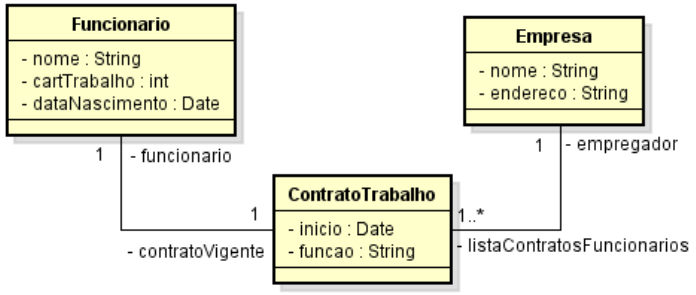
Classe Associativa [2..3]

- Classes associativas são utilizadas apenas durante a fase de análise.
- **Exemplo**:

Durante a fase de **Análise** (com classe associativa):



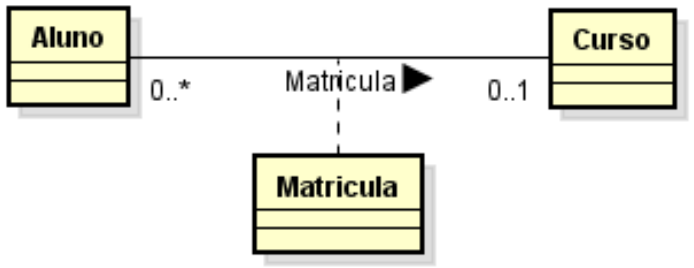
Durante a fase de **Projeto** ou Desenvolvimento (sem classe associativa):



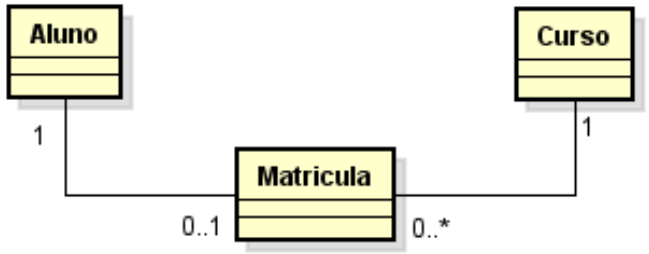
Classe Associativa [3..3]

- **Outro exemplo**:

Durante a fase de **Análise** (com classe associativa):



Durante a fase de **Projeto** ou Desenvolvimento (sem classe associativa):

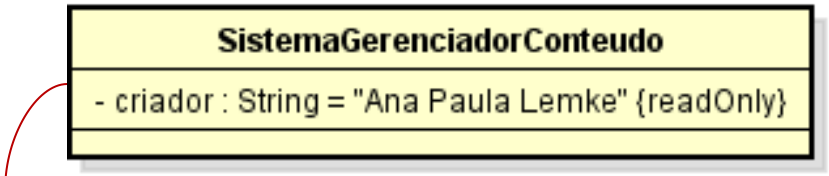


Propriedades dos Atributos

readOnly (atributo *final*)

- Indica que o valor de um atributo não pode mais ser modificado depois que um valor inicial lhe é atribuído. Pode ser considerado como um atributo "*constante*".

- Vários atributos constantes são definidos em Java como **public static final.**
  - Exemplo: PI (3.14159...) da classe Math.




```
public class SistemaGerenciadorConteudo {
        private final String criador = "Ana Paula Lemke";
}
```

Propriedades dos Atributos

changeable

- Não há restrições quanto a modificação do valor do atributo. Por *default*, um atributo é sempre **changeable**.

addOnly

- Válido para atributos com multiplicidade superior a um, onde o valor atribuído não pode ser alterado ou removido.

union

- É frequentemente utilizado para indicar que um atributo é uma união derivada de outro conjunto de atributos.

redefines <attribute-name>

- Indica que um atributo age como um “alias” de um outro atributo. Pode ser utilizado para indicar que um uma subclasse possui um atributo que na verdade é um “alias” para um atributo da superclasse.

Propriedades das Operações

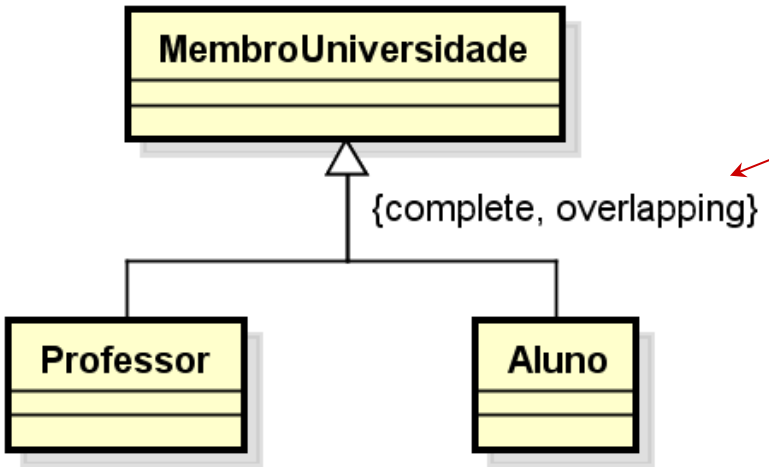
- ***leaf***: indica que a operação não possuirá redefinição.
- ***isQuery***: indica que a operação é “pura”, ou seja, não altera o estado do sistema.
- ***sequential***: os invocadores da operação devem coordenar externamente o objeto, garantindo que exista um único fluxo no objeto por vez.
- ***guarded***: a operação garante que várias chamadas serão tratadas como chamadas sequenciais.
- ***concurrent***: a operação é considerada atômica e permite que seja executada concorrentemente com outras operações.

Restrições

- Uma restrição define limites para objetos, classes, atributos, ligações ou associações.
- Uma restrição é especificada entre chaves “**{ }**” próximo ao elemento restrito, ou pode ser especificada como comentário do elemento.
- Restrições servem para limitar e realizar a consistência dos elementos podendo se tornar base para asserções (pré e/ou pós condições) em programação.

Restrições na Generalização

- ***complete***: não há mais nenhuma subclasse a especificar.
- ***incomplete***: existem outras subclasses a especificar.
- ***disjoint***: um objeto da superclasse só pode ser objeto de uma subclasse.
- ***overlapping***: um objeto da superclasse pode ser objeto de mais de uma subclasse ao mesmo tempo.



Complete: não há mais nenhum tipo de membro de universidade.

Overlapping: um membro de Universidade pode ser professor e aluno ao mesmo tempo.

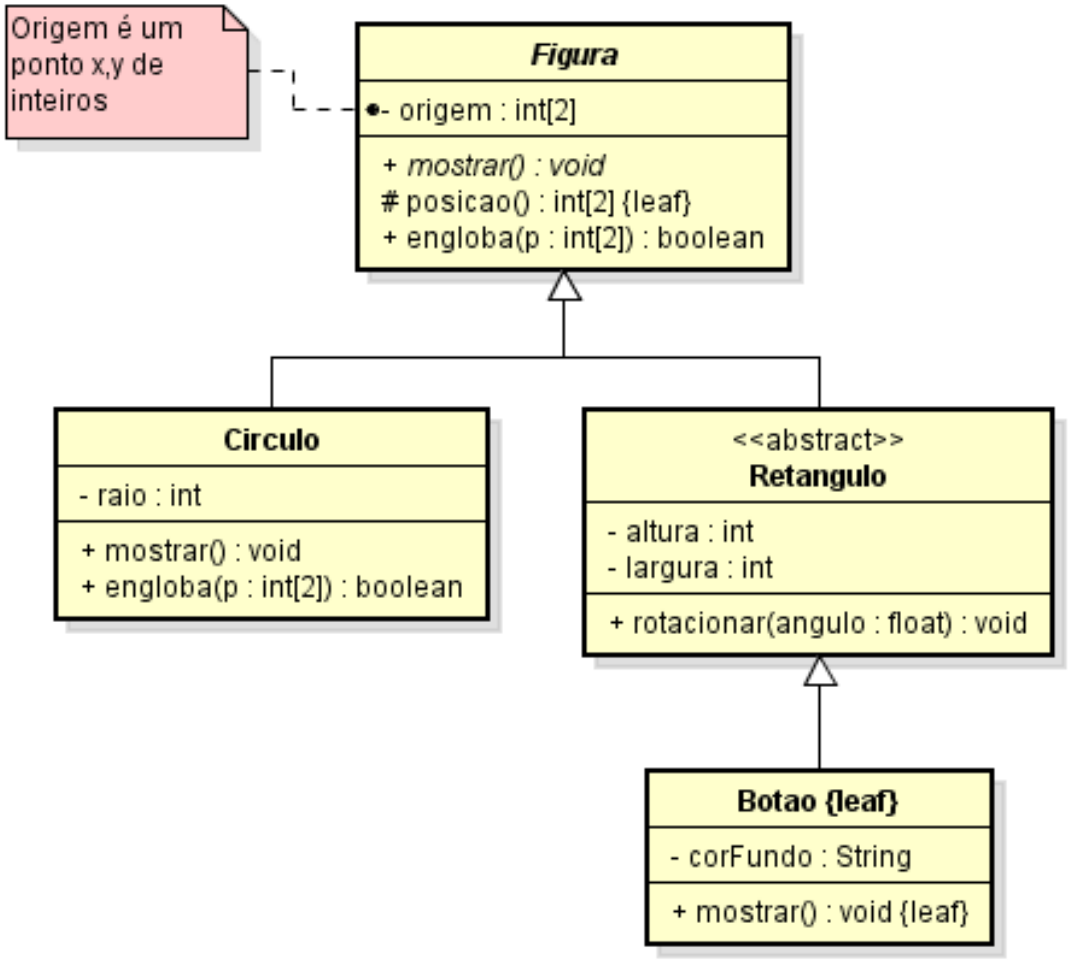
Classes Abstratas/Concretas

- **Classe concreta:**
  - Classes concretas possuem instâncias (objetos).
- **Classe abstrata:**
  - Não possui instâncias. É usada na construção de uma hierarquia de relacionamentos de generalização.
  - Pode ter operações concretas e operações abstratas. As operações abstratas devem ser implementadas nas subclasses concretas da classe abstrata. **Obs**.: construtores de classes abstratas não podem ser abstratos.
  - O nome de uma classe abstrata geralmente é escrito em *itálico*.
    - Pode ser utilizado o estereótipo <<abstract>> também.

Classe Folha e Classe Raiz

- Uma **classe folha** é aquela que não pode ter subclasses.
  - Em UML, utiliza-se a propriedade {**leaf**} para especificar uma classe folha.
- Uma **classe raiz** é aquela que não pode ter superclasses (classe-pai).
  - Em UML, utiliza-se a propriedade {**root**} para especificar uma classe raiz.

Classe Folha e Classe Raiz - Exemplo



Estereótipo Interface

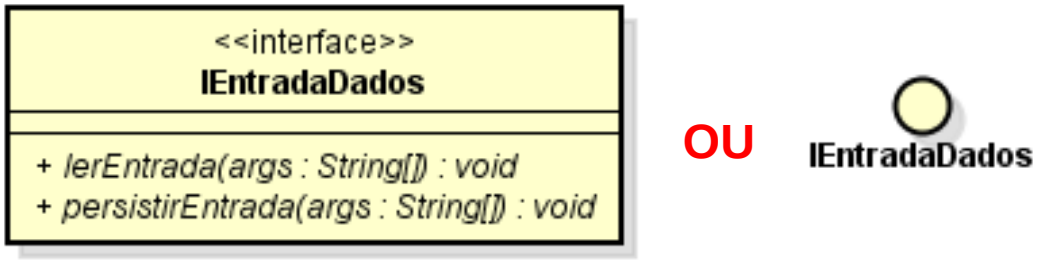
- Em UML existe um estereótipo de classe chamado **interface**.
- Uma interface é uma classe que define um conjunto de operações sem implementação (apenas assinaturas).

- Características de interfaces:
  - Todos os métodos de uma interface são implicitamente ***abstract*** e ***public***.
  - Todos os atributos de uma interface (se houverem atributos) são implicitamente atributos de classe e somente leitura.
  - Interfaces não podem ter construtores.

Interfaces X Classes Abstratas

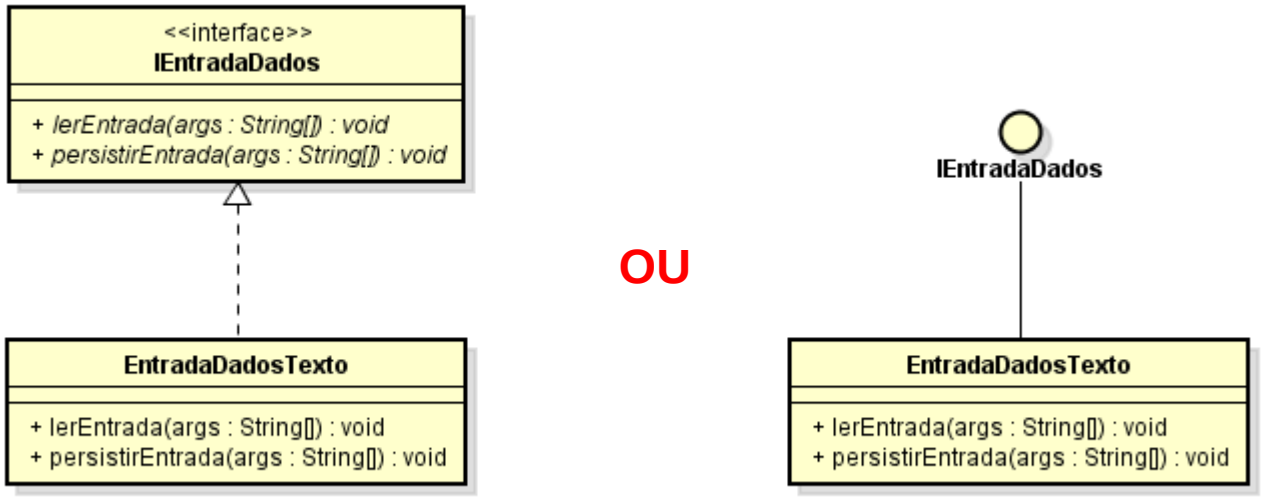
- Uma classe herdeira somente pode herdar (*extends*) de **uma classe**, seja ela abstrata ou não.
- Uma classe herdeira pode implementar (*implements*) **várias classes** simultaneamente.

- Notação de interfaces:



Relacionamento de Realização

- Uma realização é um relacionamento no qual um item (classe ou caso de uso) concretiza ou implementa o comportamento de outro item.



Interfaces X Classes Abstratas

- É possível alterar o valor de um atributo marcado como *static*?
- É possível estender uma classe com a propriedade {leaf}?
- Posso sobrecarregar uma operação com a propriedade {leaf}?
  - Posso redeclarar uma operação com a propriedade {leaf}?
- Classes abstratas podem ser instanciadas?
- Uma classe pode estender um número indeterminado de outras classes?
  - E pode implementar quantas interfaces?

Classe parametrizada [1..5]

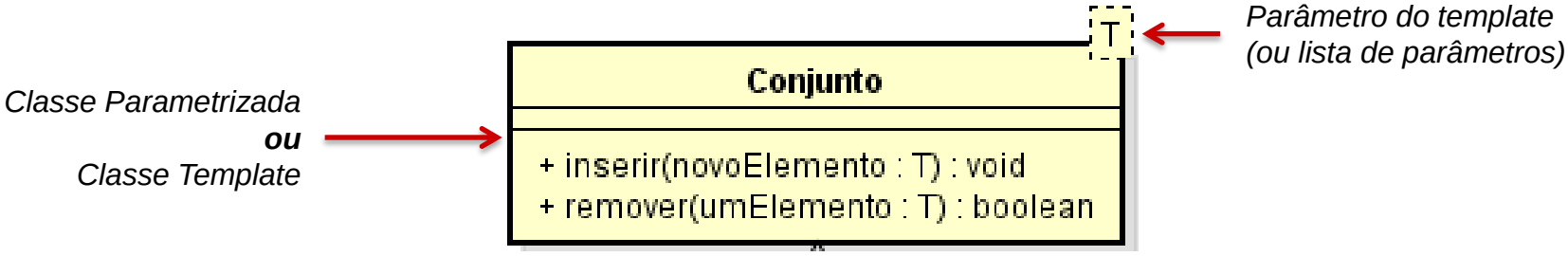
- Assim como ocorre em algumas linguagens de programação, a UML permite a criação de **classes parametrizadas**, ou ***templates (modelos), ou* classes *template***.
  - O conceito fica óbvio quando se trabalha com coleções ou listas em uma linguagem de programação.

```
public ArrayList<Pet> meusPets = new ArrayList<Pet>();
```

  - No exemplo acima, ArrayList é uma definição geral que pode ser utilizada para criar elementos mais específicos (como uma listagem apenas de pets).

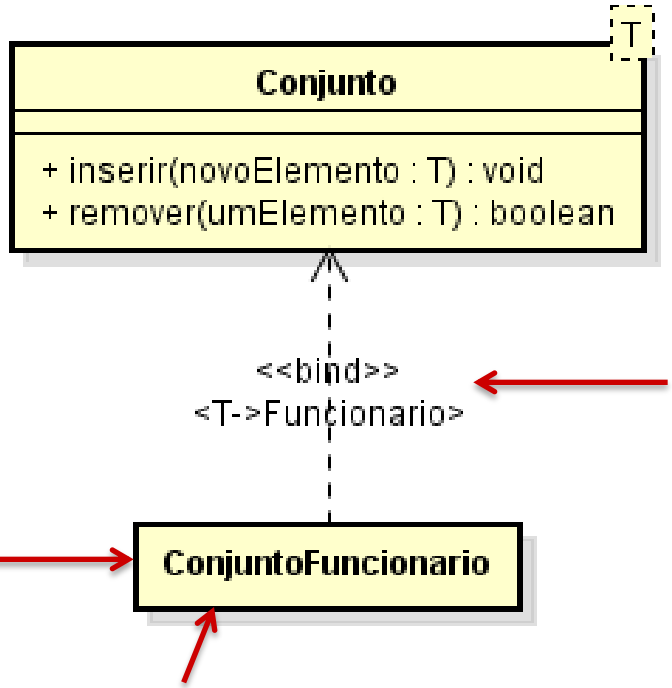
Classe parametrizada [2..5]

- Notação da classe parametrizada:



Classe parametrizada [3..5]

- Notação da classe parametrizada com “elemento ligado” de forma **explícita** (com vinculação explícita):



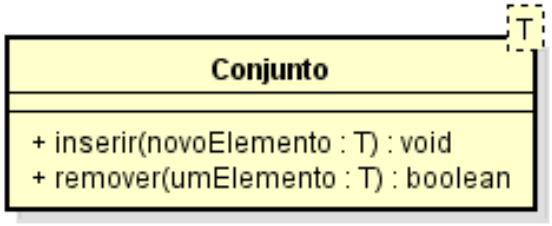
*Ligação indicando o parâmetro para a classe template (também chamada de **template binding**).*
No exemplo, o binding substitui o parâmetro T por Funcionário

***Bound element** (“elemento ligado” – representa um uso da classe template)*

Normalmente não são mostrados os compartimentos de atributos e operações, pois não pode haver qualquer alteração no bound element (apenas se indica o(s) parâmetro(s)

Classe parametrizada [4..5]

- Notação da classe parametrizada com “elemento ligado” de forma **implícita** (com vinculação implícita):

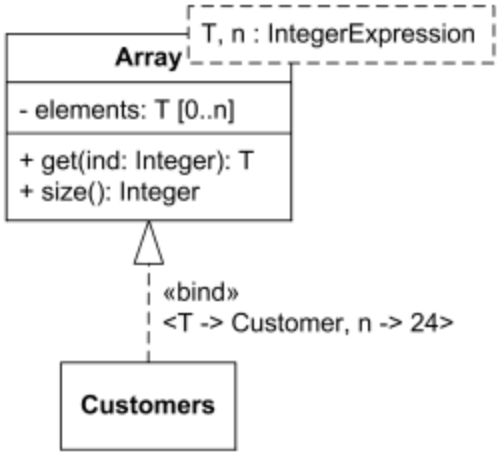


O bound element precisa ter o mesmo nome da classe *template quando se utiliza este padrão de notação.*

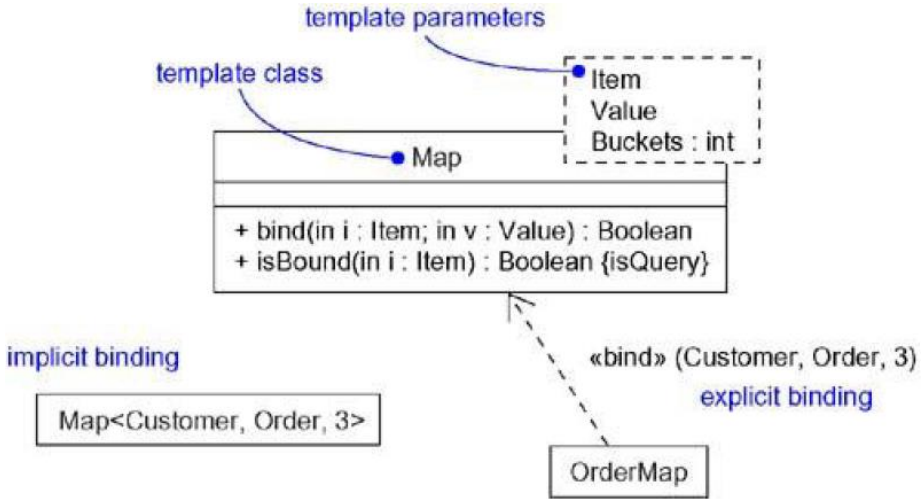
A indicação do(s) parâmetro(s) é feita entre sinais de maior e menor dentro do compartimento da identificação do bound element.

Classe parametrizada [5..5]

- Outros exemplos:



Binding substitutes class T with class Customer and boundary n with integer value 24.



Saiba mais em: https://www.uml-diagrams.org/template.html#template-binding

Do **Modelo de Casos de Uso** para o **Diagrama de Classes**

Como identificar as classes do sistema [1..3]

Estratégias:

- Encontrando conceitos através de uma lista de categorias de conceitos: relacione os conceitos candidatos do domínio do problema (objetos físicos, transações, linhas de itens de transações, papéis desempenhados por pessoas, contêineres de coisas, eventos, etc.).

- **Encontrando conceitos com a identificação de substantivos**: identifique os substantivos e frases que podem estar no lugar de um substantivo nas descrições do domínio do problema e considere-os como candidatos a conceitos ou atributos para o modelo conceitual.
  - Utilize os casos de uso expandidos como fonte de referência.
  - **DICA**: A maioria (senão todos) dos atores dos Casos de Uso serão classes no sistema.

Como identificar as classes do sistema [2..3]

- Uma abordagem prática de identificar objetos é desenvolver modelos a partir de cada caso de uso expandido.

- Para cada caso de uso é preciso:
  - Descobrir candidatos a objetos.
  - Descobrir interações entre estes objetos (comportamento).
  - Descrever as classes.

Como identificar as classes do sistema [3..3]

- Limitar responsabilidade das classes (o que cada classe irá fazer):
  - A classe possui um propósito simples e claro?
  - É possível descrever este propósito em uma sentença simples?
  - Os métodos representam a responsabilidade da classe?
- Nomes de classes claros e consistentes:
  - O nome de cada classe é um substantivo?
  - O nome de cada classe ou método não é ambíguo para desenvolvedores externos ao projeto?
  - O nome de cada método é um verbo ou uma combinação de verbo + substantivo?

Tarefa 4

- Elabore um diagrama de classes para um jogo simples de corrida de carros. O jogo deve possuir diversos modelos de carros, cada um com suas características de aceleração, velocidade máxima e freios. O jogo deve possuir várias pistas e deve armazenar as pontuações dos competidores através das corridas.

Referências adicionais

- Booch, J.; Rumbaugh, J.; and Jacobson, I. “**The Unified Modeling Language User Guide**”, Addison Wesley, 1998, 512 p.
- Chonoles, M. and Schardt J. “**UML 2.0 for Dummies**”, 2003.
- Hamilton, Kim; Miles, Russell “**Learning UML 2.0”**, O'Reilly, 2006, 286 p.