Introdução à Programação Orientada a Objetos II

BCC 221 - Programação Orientada a Objectos(POO)

Guillermo Cámara-Chávez

Departamento de Computação - UFOP

Polimorfismo I

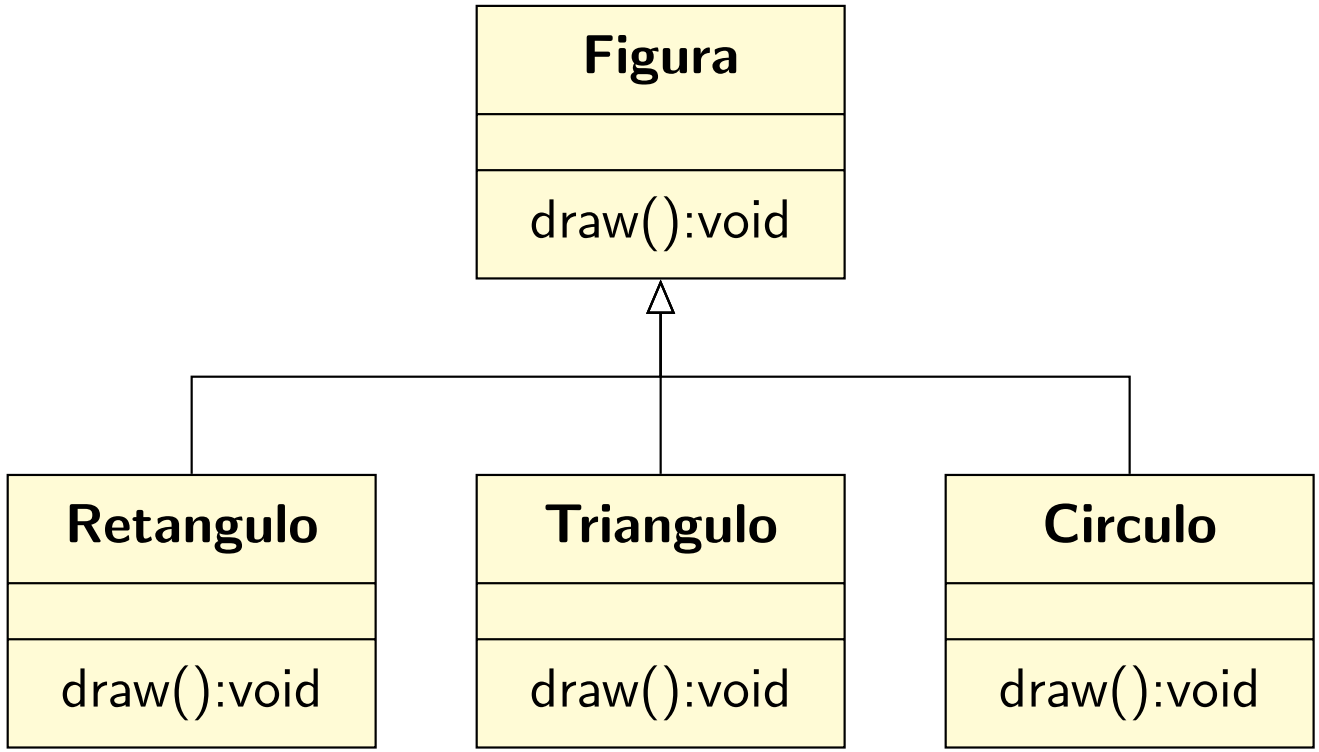
Polimorfismo

São **comportamento diferentes**, asociados a objetos distintos, pode **compartilhar o mesmo nome**; ao ser chamados por esse nome, **será utilizado o comportamento correspondente ao objeto que é usado**

- É a possibilidade de um mesmo método ser executado de forma diferente de acordo com a classe do objeto que aciona o método
- Os métodos envolvidos são chamados de virtuais, podem atuar em qualquer nível da hierarquia.

Polimorfismo II

Polimorfismo III



Polimorfismo IV

- **Tipos de polimorfismo**

  Ad hoc: refere-se a funções que mudam seu comportamento dependendo do tipo de argumentos que recebem (sobrecarga de métodos o funções)

```
void Escreve();
void Escreve(int num);
void Escreve(int num, string nome);
```

Polimorfismo V

Paramétrico: permite que as funções e classes possam se escrever de forma genérica. Exemplo: C++ - *templates* e Java - *Generics*

Subtipos: os subtipos de um tipo podem substituir o comportamento das funções do supertipo com sua própria implementação

Polimorfismo VI

- A seleção do método pode ser feita pelo sistema durante a execução (*dynamic binding*)
- Seja **F** o conjunto das figuras e **f** uma figura, vamos "desenhar" a figura

Na Programação Estruturada escrevemos:

```
para cada f em F faça case tipo(f):
        Retângulo: drawR(f)
        Triângulo: drawT(f)
        Círculo: drawC(f)
```

Na POO escrevemos:

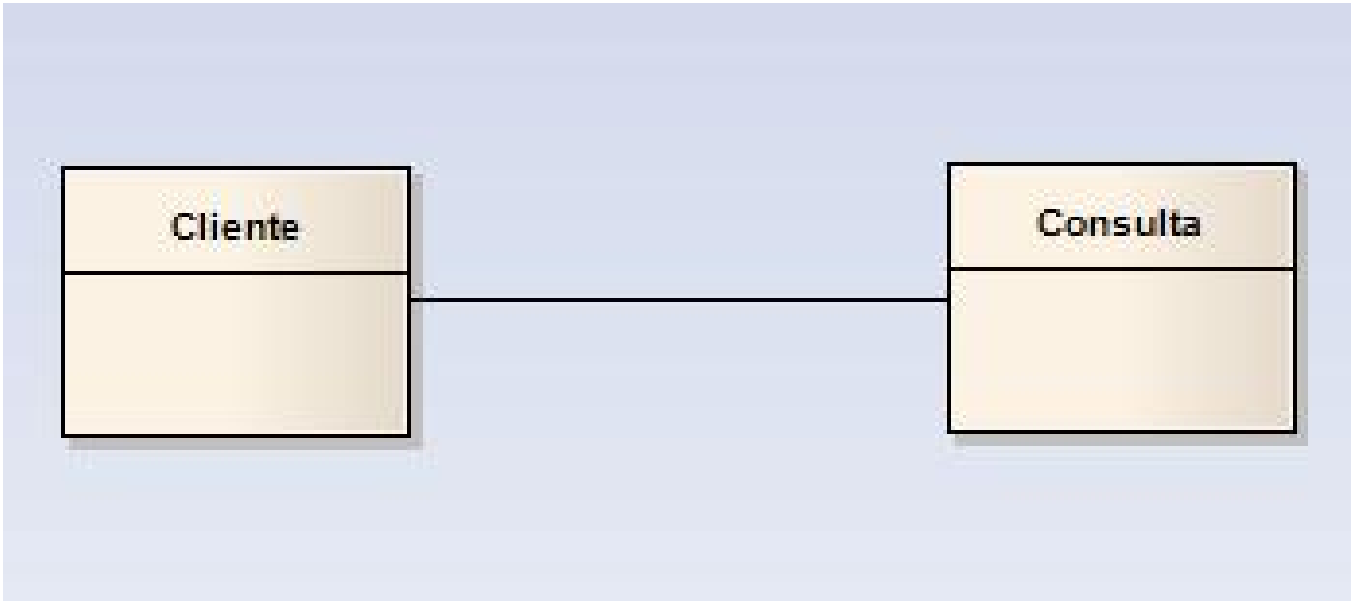
```
para cada f em F do draw(f)
```

Relacionamentos entre classes I

- Classes possuem relacionamentos entre elas (comunicação)
  - Compartilham informações
  - Colaboram umas com as outras
- Principais tipos de relacionamentos
  - Associação
  - Agregação / Composição
  - Herança
  - Dependência

Relacionamentos entre classes II

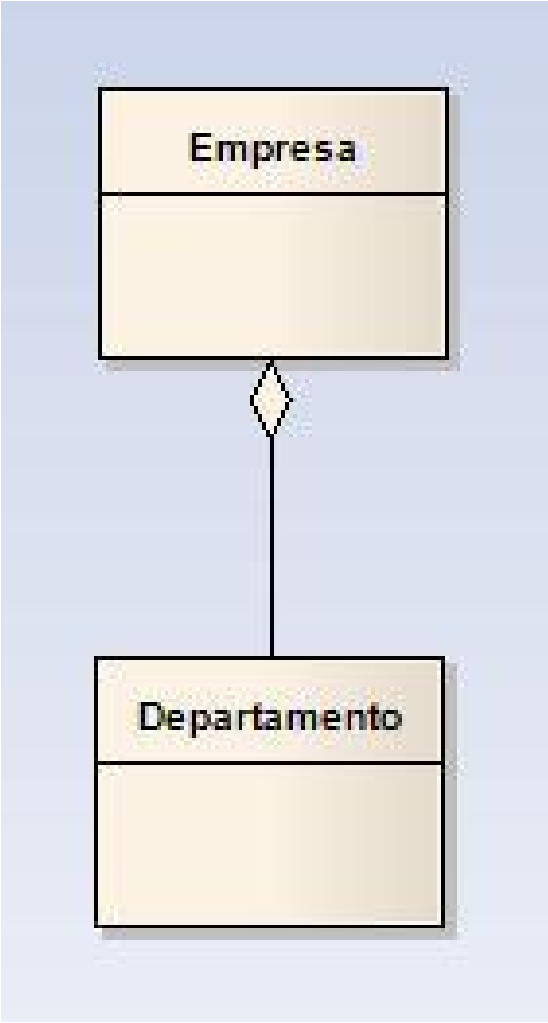
- Uma **Associação** é o mecanismo pelo qual um objeto utiliza os recursos de outro
- A função da mesma é só mostrar o relacionamento ou dependência de uma classe com a outra



Relacionamentos entre classes III

- **Agregação** e **Composição** são tipos especiais de **Associação**;
- A associação de agregação é usada para indicar que um objeto colabora com outro.
- Pode ser uma associação simples
  - "**Usa um / tem um**" : Uma pessoa **usa um** computador
- Ou uma acoplamento
  - "**Parte de**" : O teclado é **parte de** um computador

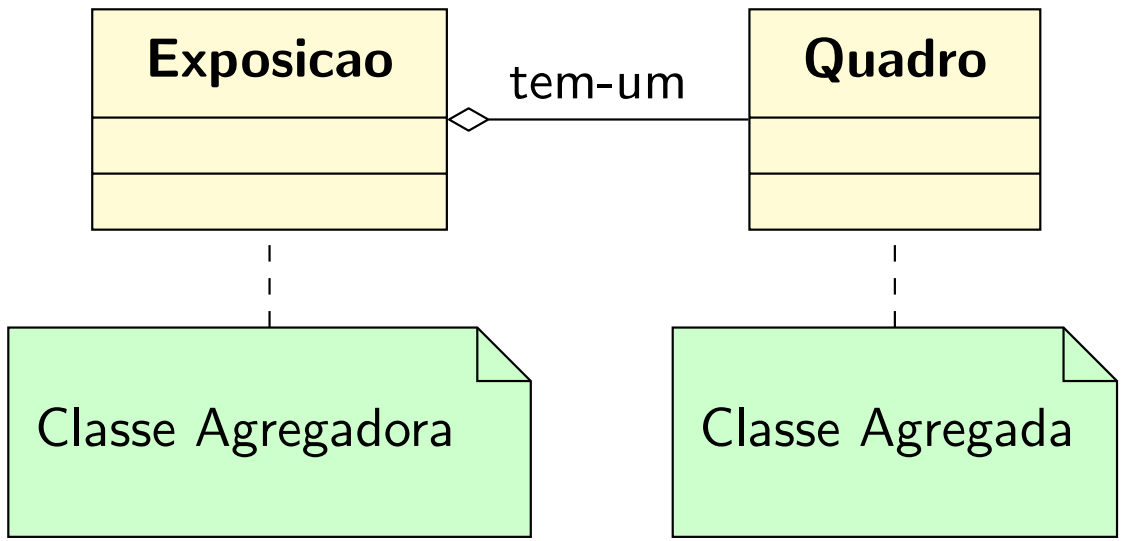
Relacionamentos entre classes IV



Uma empresa pode ter vários departamentos e um departamento também pode ou não existir em uma empresa

Relacionamentos entre classes V

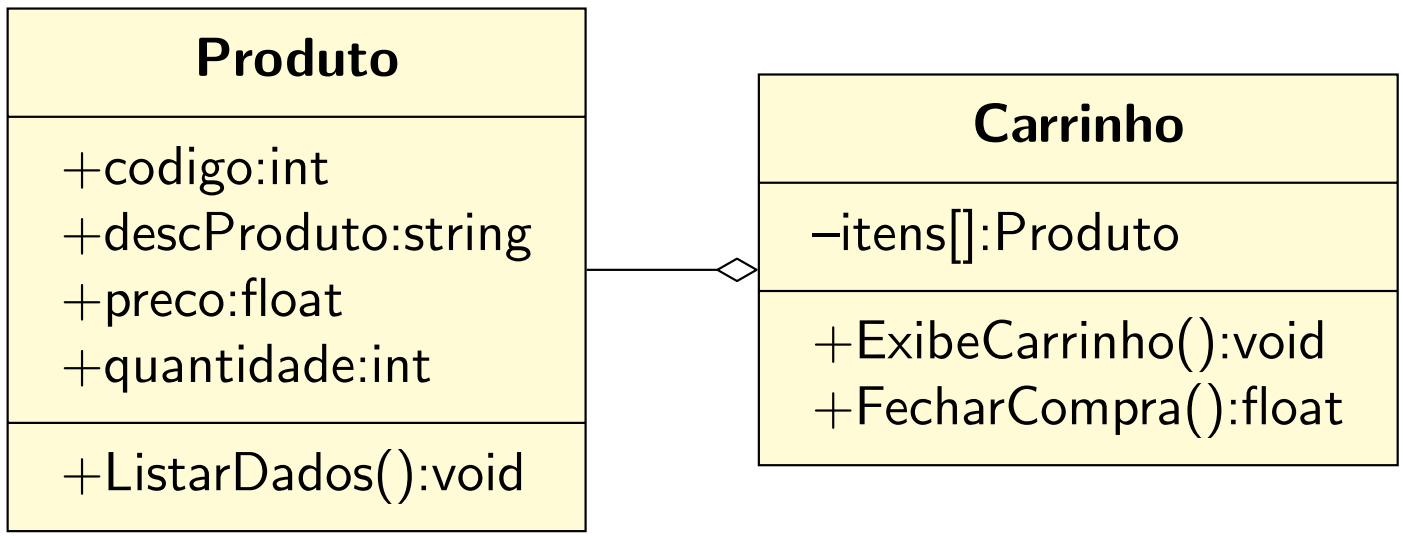
- Um objeto agregado **tem existência independente** do objeto agregador
- O objeto agregado **pode existir** após **eliminação** do objeto agregador



Relacionamentos entre classes VI

- A classe Agregada faz parte da estrutura da classe Agregadora
- O objeto agregado **é parte do** objeto agregador (guardado em **variável de instância**

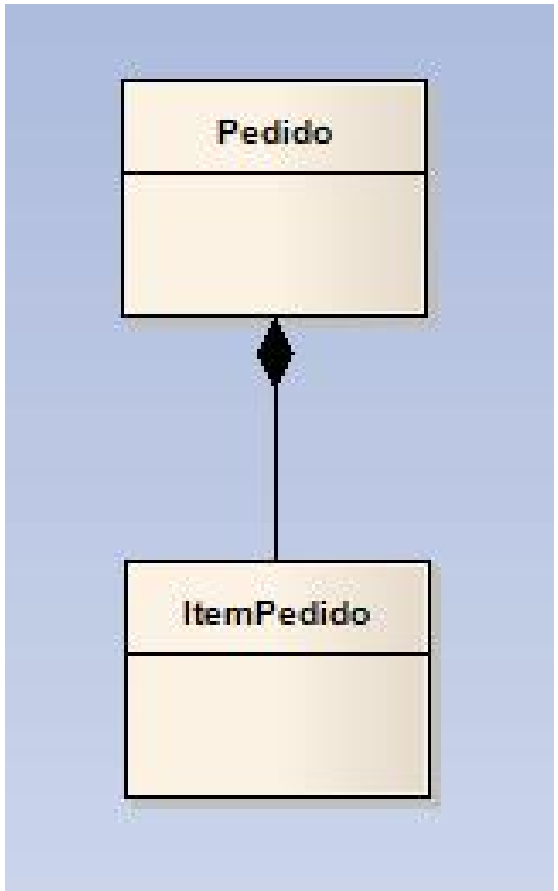
Relacionamentos entre classes VII



Relacionamentos entre classes VIII

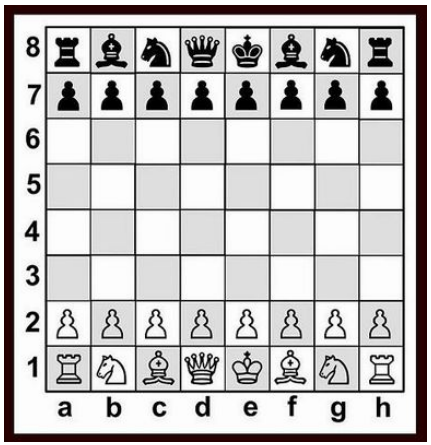
- A associação de **composição** é um tipo de associação de agregação com restrições na ligação entre parte e agregado
- A existência da parte depende da existência do agregado
  - O agregado (todo) não existe (ou não faz sentido sem as partes
  - Ou, as partes não existem sem o todo

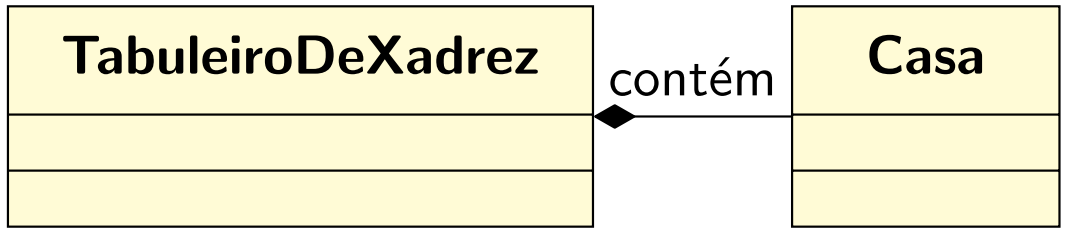
Relacionamentos entre classes IX



Um item de pedido não faz sentido existir sem um pedido, ou seja, dentro de um item de pedido sempre vou ter um pedido associado.

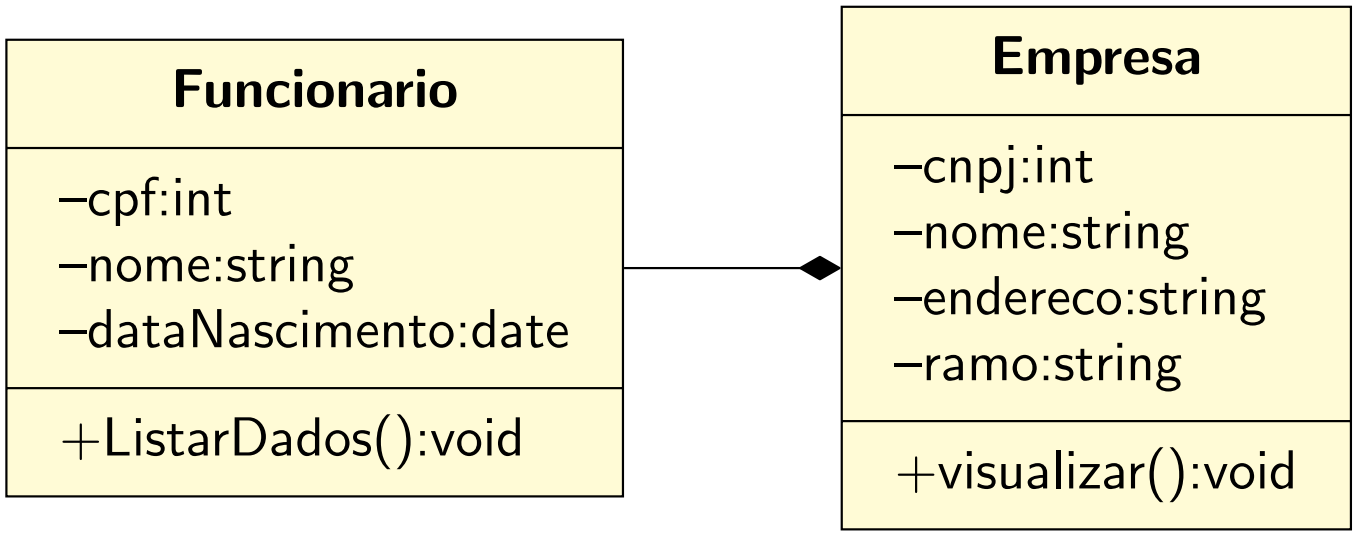
Relacionamentos entre classes X





Um tabuleiro de Xadrez é formado, composto por 32 casas pretas e 32 casas brancas

Relacionamentos entre classes XI

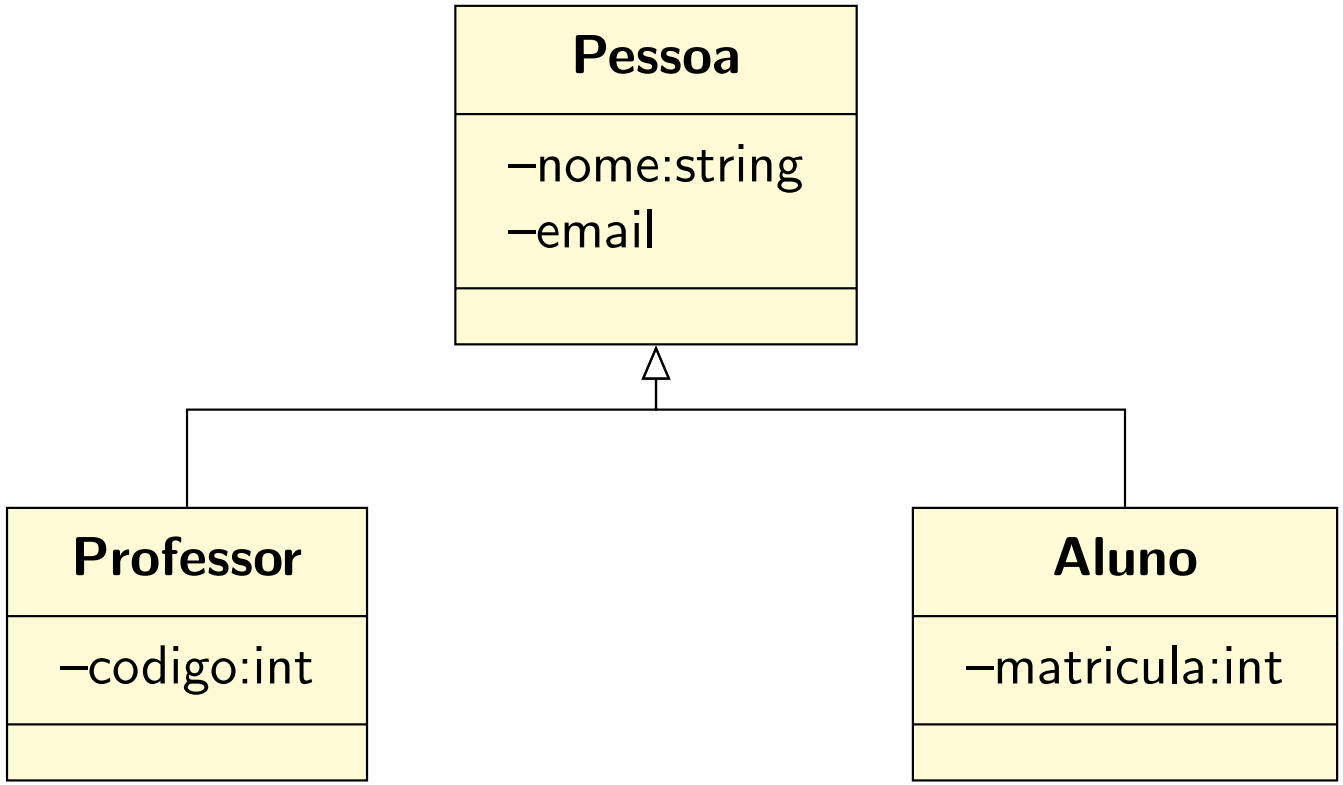


- Observe que não faz sentido ter funcionários, se não existir uma empresa onde eles possam trabalhar.
- Se a empresa deixar de existir, automaticamente ela deixará de ter funcionários.

Relacionamentos entre classes XII

- Identificar super-classe (geral) e sub-clases (especializadas)
  - O relacionamento de **herança** define um relacionamiento do tipo "**é um**"
  - Tudo o que a classe geral pode fazer, as classes específicas também podem
- Atributos e métodos definidos na classe-mãe são **herdados** pelas classes filhas

Relacionamentos entre classes XIII



Relacionamentos entre classes XIV

- Vantagens da herança
  - O gráfico de herança é uma fonte de conhecimento sobre o domínio do sistema
  - É um mecanismo de abstração usado para classificar entidades
  - Mecanismos de reuso em vários níveis

Relacionamentos entre classes XV

- Classes de objetos não são autocontidas
  - Não podem ser compreendidas sem referencia às super-classes
- O código da superclasse não estará disponível no código da subclasse
  - É necessário que este bem documentado

Relacionamentos entre classes XVI

- Podemos pensar sobre herança como algo semelhante a funções
  - Quando identificamos um trecho de código que se repete várias vezes, criamos uma função com aquele conteúdo
  - Quando identificamos várias características em comum em um grupo de classes, podemos criar uma superclasse
  - Objetivo: evitar redundância

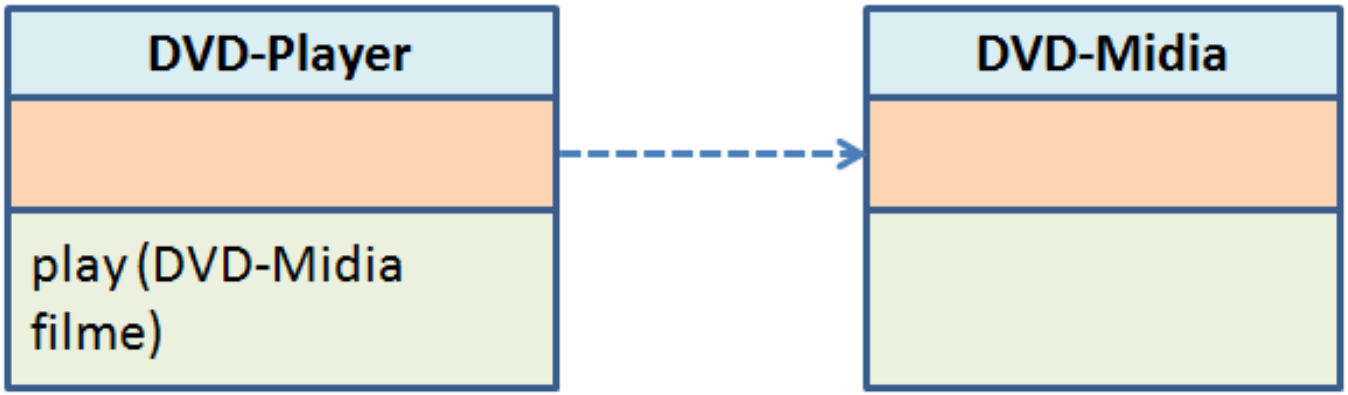
Relacionamentos entre classes XVII

- Uma **Dependência** é uma forma fraca de relacionamento
- Tipo menos comum de relacionamento
- Identifica uma ligação fraca entre objetos de duas classes
  - Uma classe depende de outra porque apenas em um momento específico ela a utiliza

Relacionamentos entre classes XVIII

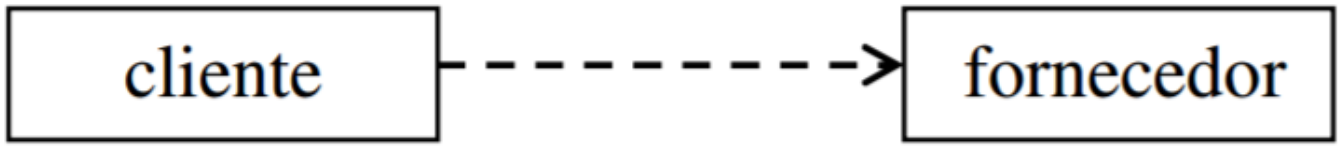
- Classe independente em declarações de (**existência** durante a execução do método, i.e. temporária)
  - Variáveis **locais**
  - Parâmetros de métodos
- Conhecida como a relação "**knows about**" ("sabe sobre" ou "conhece")

Relacionamentos entre classes XIX



O método *play* da classe DVD-Player recebo como parâmetro uma instância da classe DVD-Midia

Relacionamentos entre classes XX



- A classe cliente depende de algum serviço da classe fornecedor
- A mudança de estado do fornecedor afeta o objeto cliente
- A classe cliente não declara nos seus atributos um objeto do tipo fornecedor
- Fornecedor é recebido por parâmetro de método

Cuidado: a modificação da classe independente afeta a classe dependente

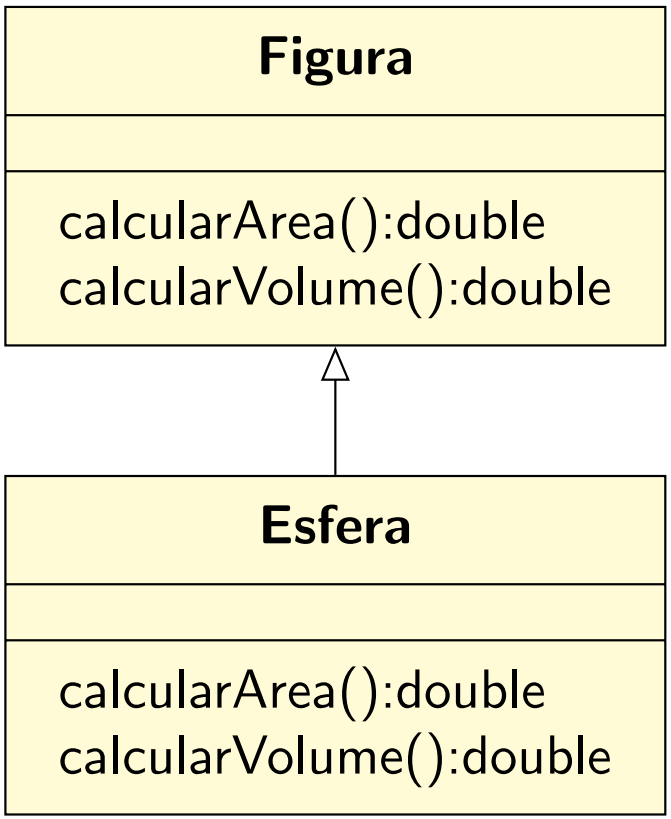
Classes Abstratas I

- Servem como “modelo” para outras classes que dela herdem
- Não podem ser instanciadas
- Para ter um objeto da classe abstrata é necessário criar uma classe mais especializada, então instanciar esse nova classe
- Permitem criar “métodos gerais” que recriam um comportamento comum, mas sem especificar como é feito
- Por exemplo, é definido que a classe “Animal” seja herdada pelas subclasses “Gato”, “Cachorro”, “Cavalo”, mas ela mesma nunca pode ser instanciada.

Classes Abstratas II

- A nível de código têm a particularidade que os métodos abstratos não têm "corpo de implementação" (somente é declarado o cabeçalho) e devem ser precedidos pela palavra chave **abstract**
- Os métodos da classe abstrata devem ser sobrescritos nas classes filhas
- Se uma classe contem um ou mais métodos abstratos, a classe é abstrata

Classes Abstratas III



Toda figura tem um método para calcular Área e Volume. A classe Esfera calcula sua área e volume

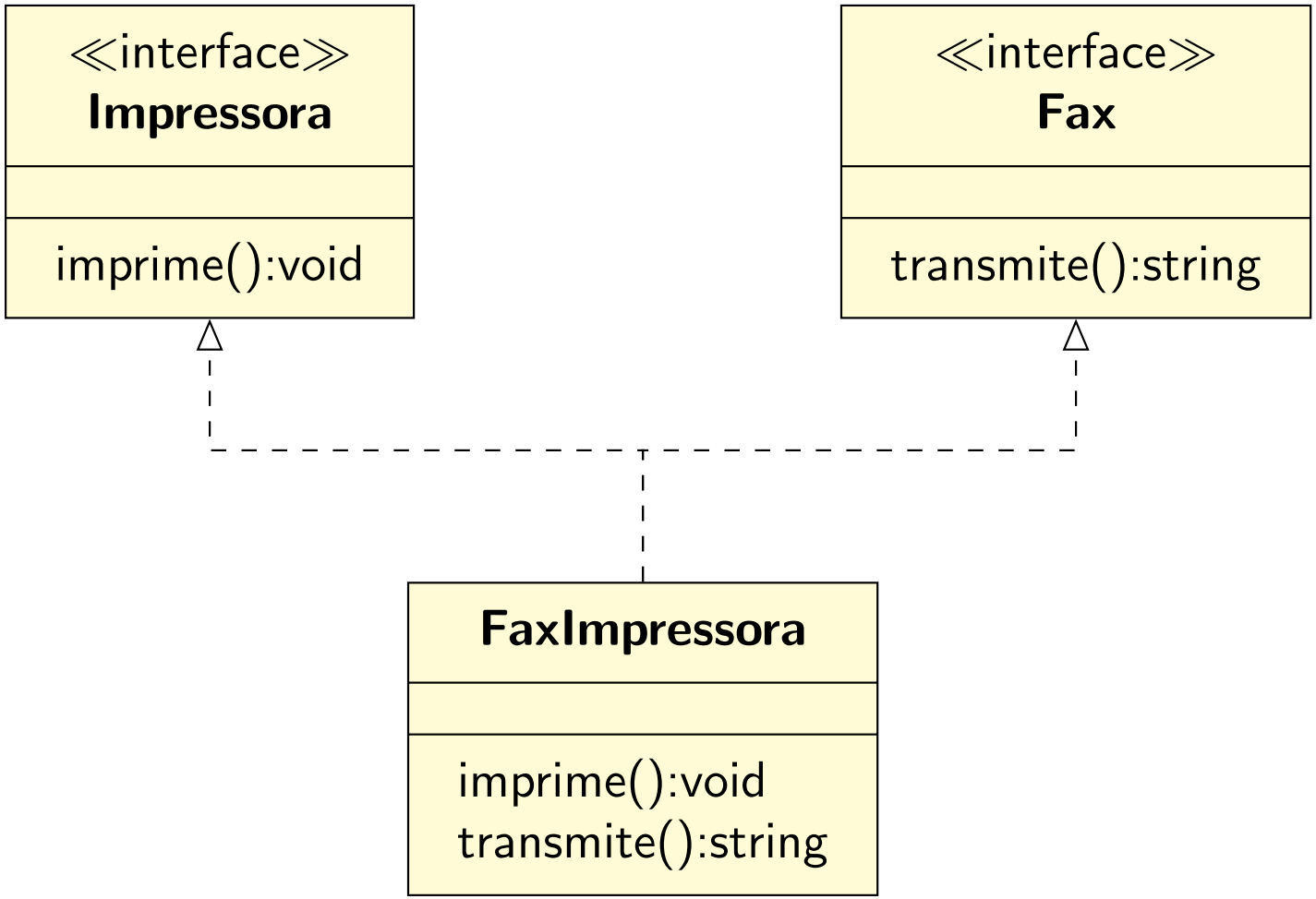
Interfaces I

- Interfaces representam:
  - a parte pública de uma classe de objetos
  - o comportamento padrão que deve ser apresentando por todas as classes que as implementam
- A interface de um objeto é o conjunto de operações públicas que ele pode realizar

Interfaces II

- Um objeto da classe Lâmpada, por exemplo, tem como interface as operações
  - acender
  - apagar
- Qualquer outra requisição feita a esse objeto será considerada inválida

Interfaces III



Interfaces IV

- Uma interface é semelhante a uma classe abstrata, pois ambas definem métodos que outras classes devem implementar
- Uma classe abstrata pode conter métodos abstratos que as classes que irão estendê-la devem implementar
- Uma interface define métodos que deverão ser implementados por classes que venham a implementar a interface
- Assim como classes abstratas, uma interface não pode ser instanciada

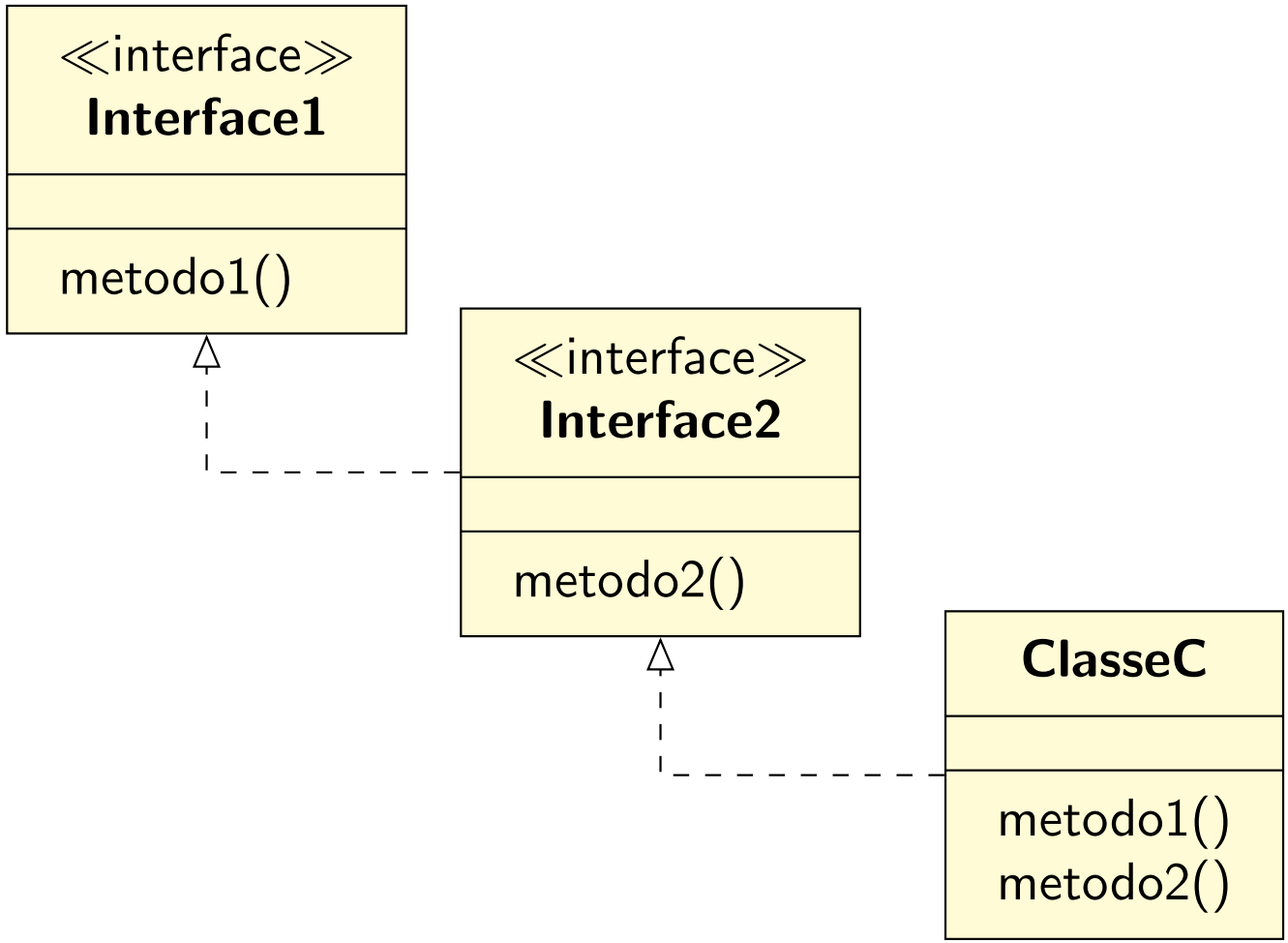
Interfaces V

- Classes abstratas podem conter métodos não-abstratos, que contêm implementação e que podem ser herdados e utilizados por instâncias das subclasses.
- As interfaces não podem conter nenhum método com implementação, todos os seus métodos são implicitamente abstract e public
- A diferença essencial entre classes abstratas e interfaces é que uma classe herdeira somente pode herdar de uma única classe (abstrata ou concreta)
- Qualquer classe pode implementar várias interfaces simultaneamente

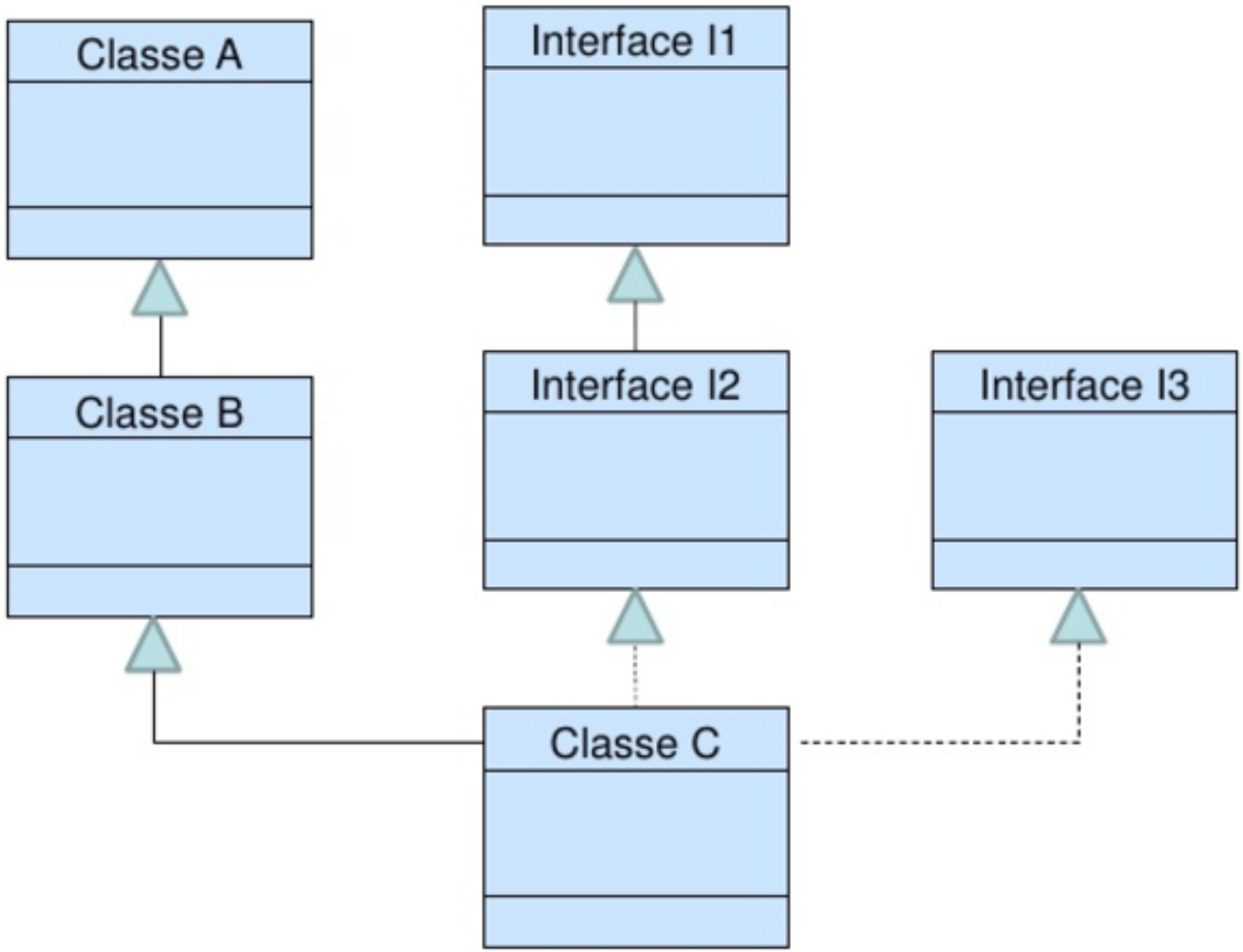
Interfaces VI

- Interfaces são um mecanismo simplificado de implementação de "herança múltipla"
- Assim como uma classe B pode estender outra classe A, uma interface I_2 pode estender outra interface I_1. Desse modo, quando uma classe C implementar I_2, terá também **obrigatoriamente** que implementar os métodos definidos na interface I_1

Interfaces VII



Interfaces VIII



Interfaces IX

- Uma interface estabelece uma espécie de contrato que é obedecido pelas classes que a implementam
- Sendo assim, quando uma classe implementa uma interface, garante-se que todas as funcionalidades especificadas pela interface serão oferecidas pela classe

Pacotes I

- Pacotes são um modo de organizar classes e interfaces
  - Um programa pode ser formado por centenas de classes individuais
  - Analogamente como a organização de arquivos em pastas, faz sentido organizar classes e interfaces relacionadas
  - Diferentes linguagens de programação fornecem bibliotecas de classes

Análise e Projeto OO I

- Antes de escrever o código, é necessário analisar os requisitos (o quê) de seu projeto e propor uma solução (como) satisfatória
- Pode poupar muitas horas de trabalho e dinheiro
- Uma linguagem gráfica utilizada para comunicação de qualquer processo de análise e projeto OO é a **Unified Modeling Language (UML)**

Unified Modeling Language (UML) I

- É a representação gráfica mais utilizada para modelagem de sistemas orientados a objetos
- Adotado como padrão internacional em 1997
- Define um conjunto padrão de notações gráficas

Unified Modeling Language (UML) II

- A versão 2.0 da UML oferece padrões de diagramas estruturais e comportamentais (de interação)
- Estamos interessados nos diagramas estruturais
  - Especificamente nos **Diagramas de Classes**

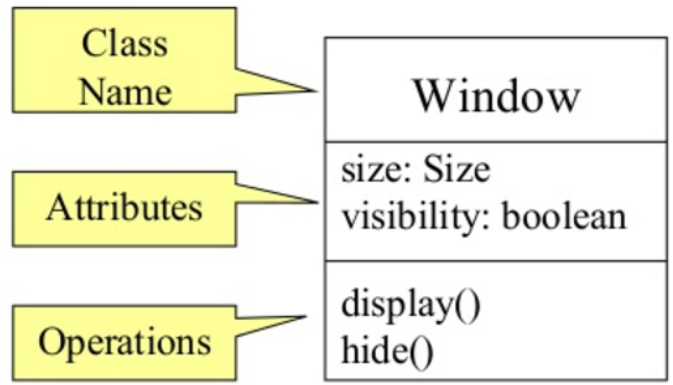
Unified Modeling Language (UML) III

- Um diagrama de classes descreve a estrutura do sistema
  - classes
  - atributos
  - métodos (operações)
  - relação entre classes
- O elemento fundamental dos diagramas de classes é o ícone que representa uma classe

Unified Modeling Language (UML) IV

O ícone é um retângulo dividido em três compartimentos:

- O mais acima representa o nome da classe
- O do meio representa os atributos
- O último representa os métodos



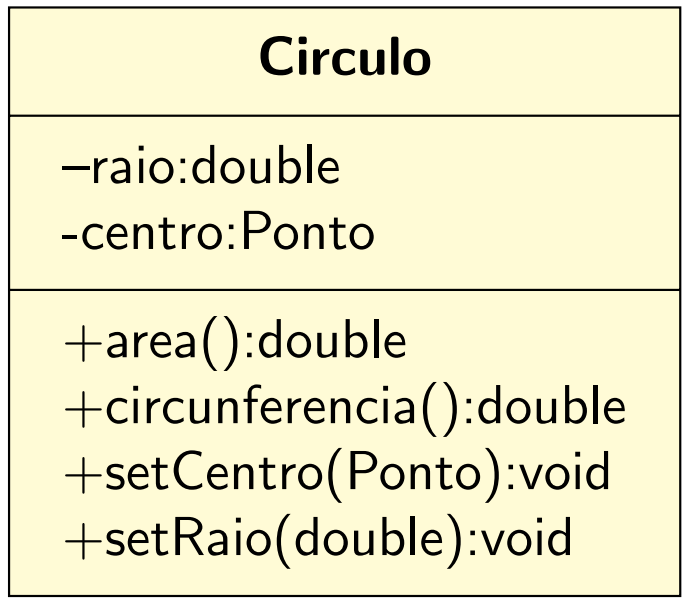
Unified Modeling Language (UML) V

- Em alguns diagramas, os dois últimos compartimentos são omitidos
  - não são apresentados todos os atributos ou métodos
  - apenas aqueles que são importantes para a finalidade do diagrama

Unified Modeling Language (UML) VI

- Visibilidade: para especificar a visibilidade de um membro de uma classe (atributo ou método) são usados as seguintes notações
  - \+ público
    - Acessível por todas as classes
  - − privado
    - Acessível somente pela própria classe
  - \# protegido
    - Acessível pela classe ou por subclasses

Unified Modeling Language (UML) VII



Unified Modeling Language (UML) VIII

- Note que cada atributo é seguido de : e depois os tipo de atributo
  - Se o tipo for redundante ou desnecessário, pode ser omitido
- Da mesma forma, o valor de retorno é apresentado depois de cada método
- Os argumentos dos métodos podem ser apenas os tipos

Unified Modeling Language (UML) IX

- Além de descrever classes, a UML pode ser usada para descrever relacionamentos entre classes
  - Composição
  - Herança
  - Agregação / Associação
  - Dependência
  - Interfaces
- Esses relacionamento são descritos por linhas conectando classes

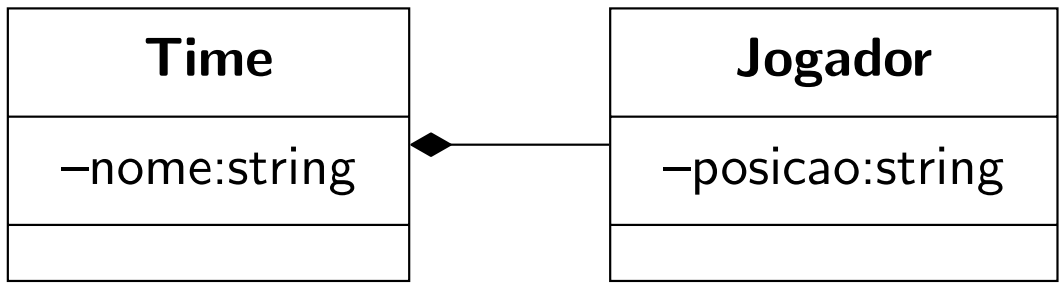
Unified Modeling Language (UML) X

- Cada extremidade da uma linha de define um relacionamento entre classe pode possuir um valor de **multiplicidade**
  - Pode ser um valor fixo: 1;
  - Pode ser um intervalo: [0 . . . 3]
  - O * significa "vários"

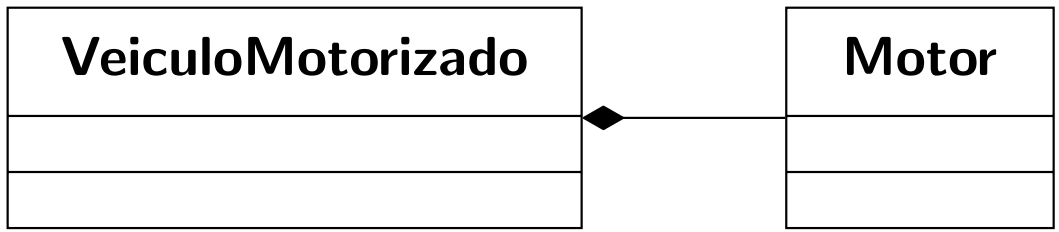
0 . . . 1	No máximo um
0 . . . ∗	zero ou muitos, pode haver vários objetos envolvidos no relacionamento
1 . . . ∗	Um ou muitos, pelo menos um objetos está envolvido

Unified Modeling Language (UML) XI

- Para representar a composição, ligamos duas classes por uma linha que contém
  - Um **diamante preto** do lado da classe que contém uma instância da outra
  - Apenas a linha do lado da outra classe

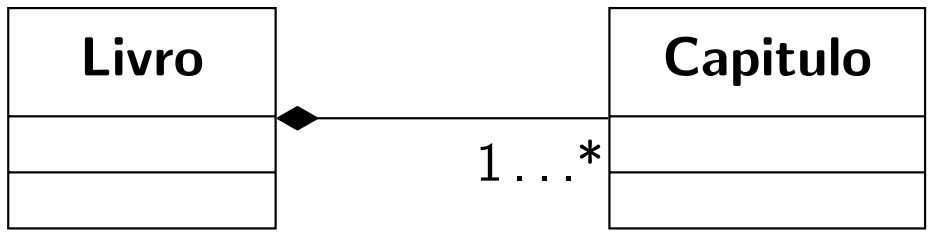


Unified Modeling Language (UML) XII



Um VeiculoMotorizado contém um Motor

Unified Modeling Language (UML) XIII

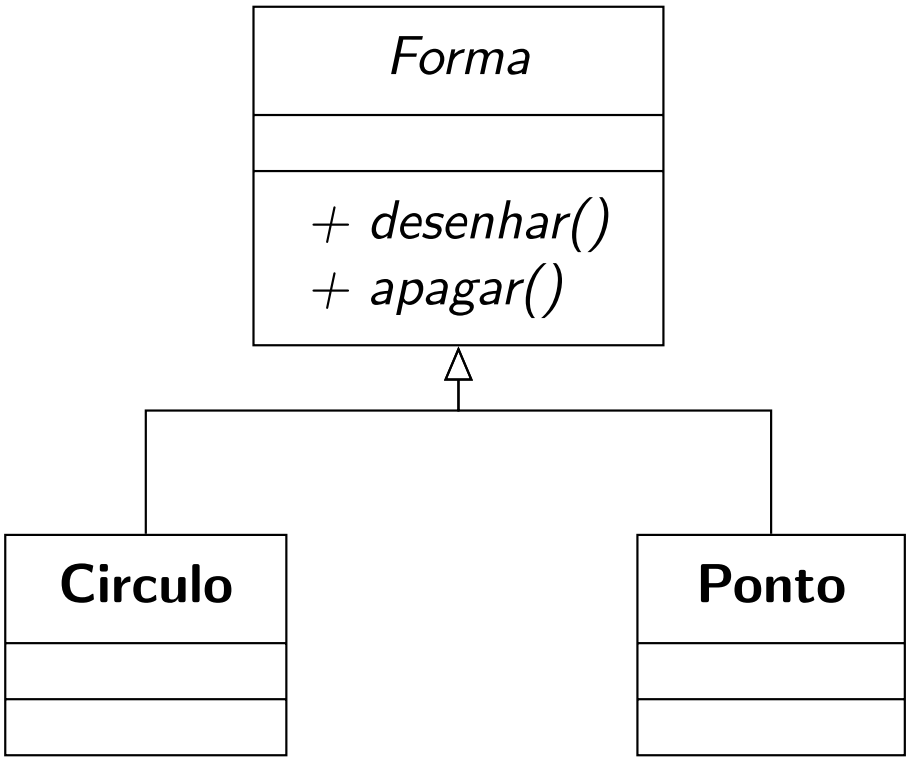


Livro contém um ou mais Capítulos

Unified Modeling Language (UML) XIV

- A Herança é representada por uma linha contendo uma seta triangular
- Identificar super-classe (geral) e subclasse (especializadas)
  - Semântica "**é um**"
  - Tudo que a classe geral pode fazer, as classes específicas também podem
  - Do lado da subclasse temos apenas a linha

Unified Modeling Language (UML) XV

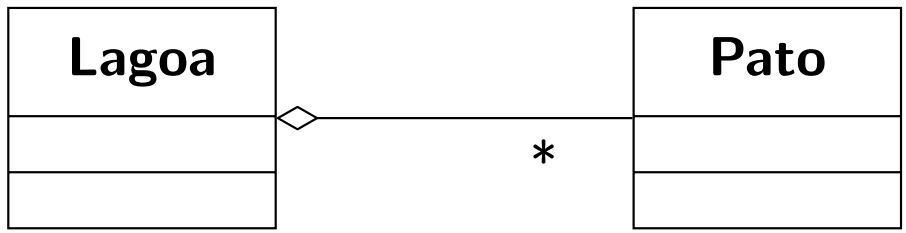


- Círculo é uma Forma
- Ponto é uma Forma
- O nome da classe e os métodos em itálico indicam que são abstratos

Unified Modeling Language (UML) XVI

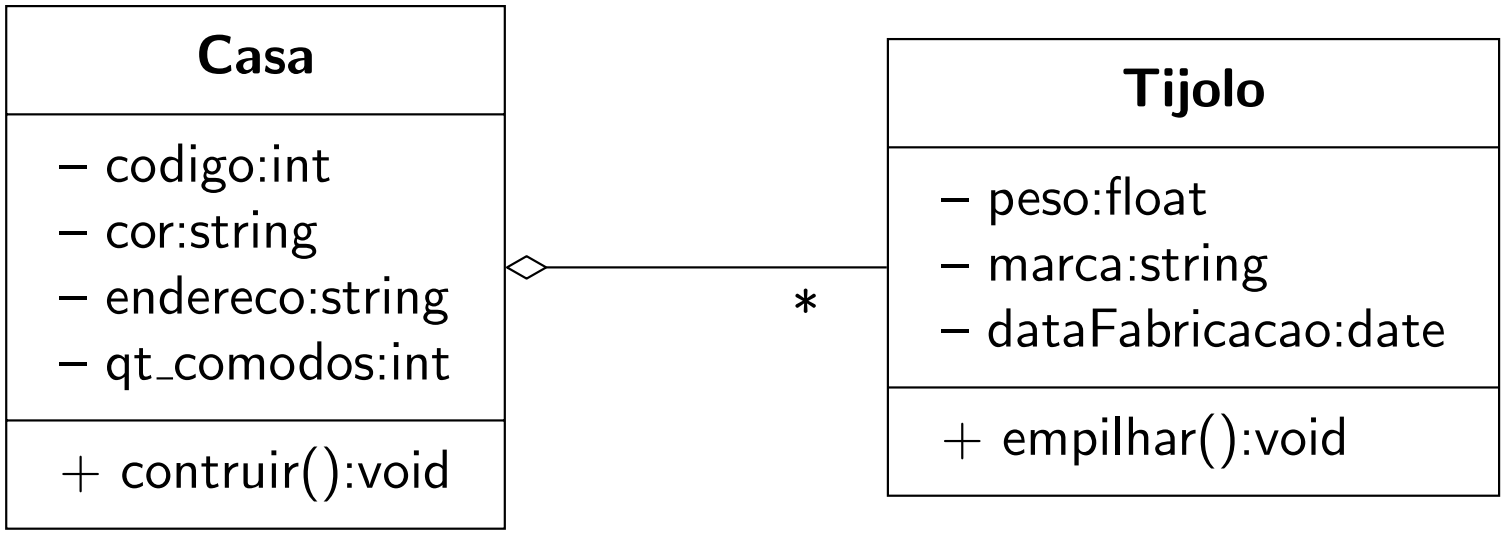
- Para representar uma **Agregação**, ligamos duas classes por uma linha contém
  - Um **diamante branco** do lado da classe que contém uma instância da outra
  - Apenas a linha do lado da outra classe
  - A multiplicidade é sempre 1 para a classe que representa o **todo**
  - Modela uma relação "parte de"

Unified Modeling Language (UML) XVII



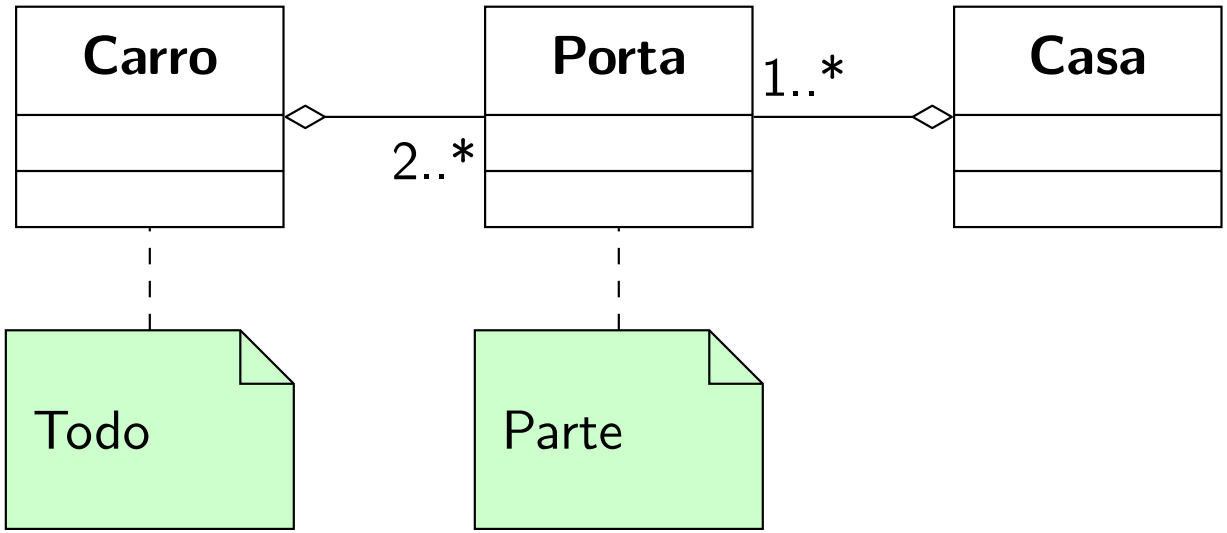
Lagoa tem vários Patos

Unified Modeling Language (UML) XVIII



Uma casa é feita de tijolos (relação todo-parte)

Unified Modeling Language (UML) XIX

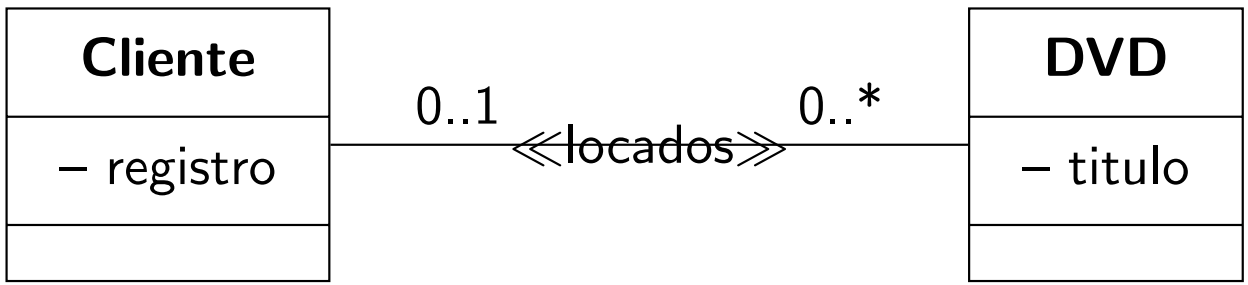


Unified Modeling Language (UML) XX

- Representamos uma **Associação** por uma linha que pode ser nomeada
- Podemos utilizar um nome para os papeis
- Provavelmente a referencia será um ponteiro ou algo do tipo

Unified Modeling Language (UML) XXI

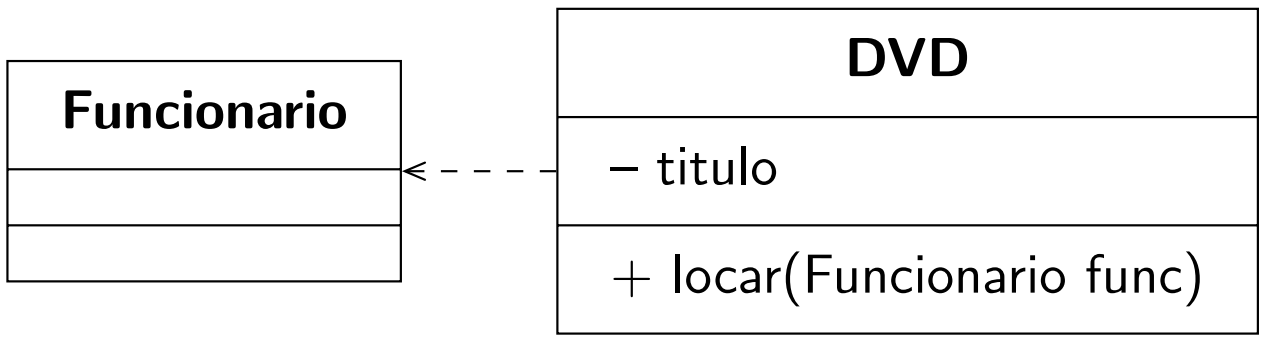
Unified Modeling Language (UML) XXII



Unified Modeling Language (UML) XXIII

- Às vezes o relacionamento entre duas classes é muito fraco
  - Representado por uma reta tracejada entre duas classes
  - Não são implementados por atributos que as una
  - Ao invés disto, pode ser implementado apenas através de parâmetros de métodos

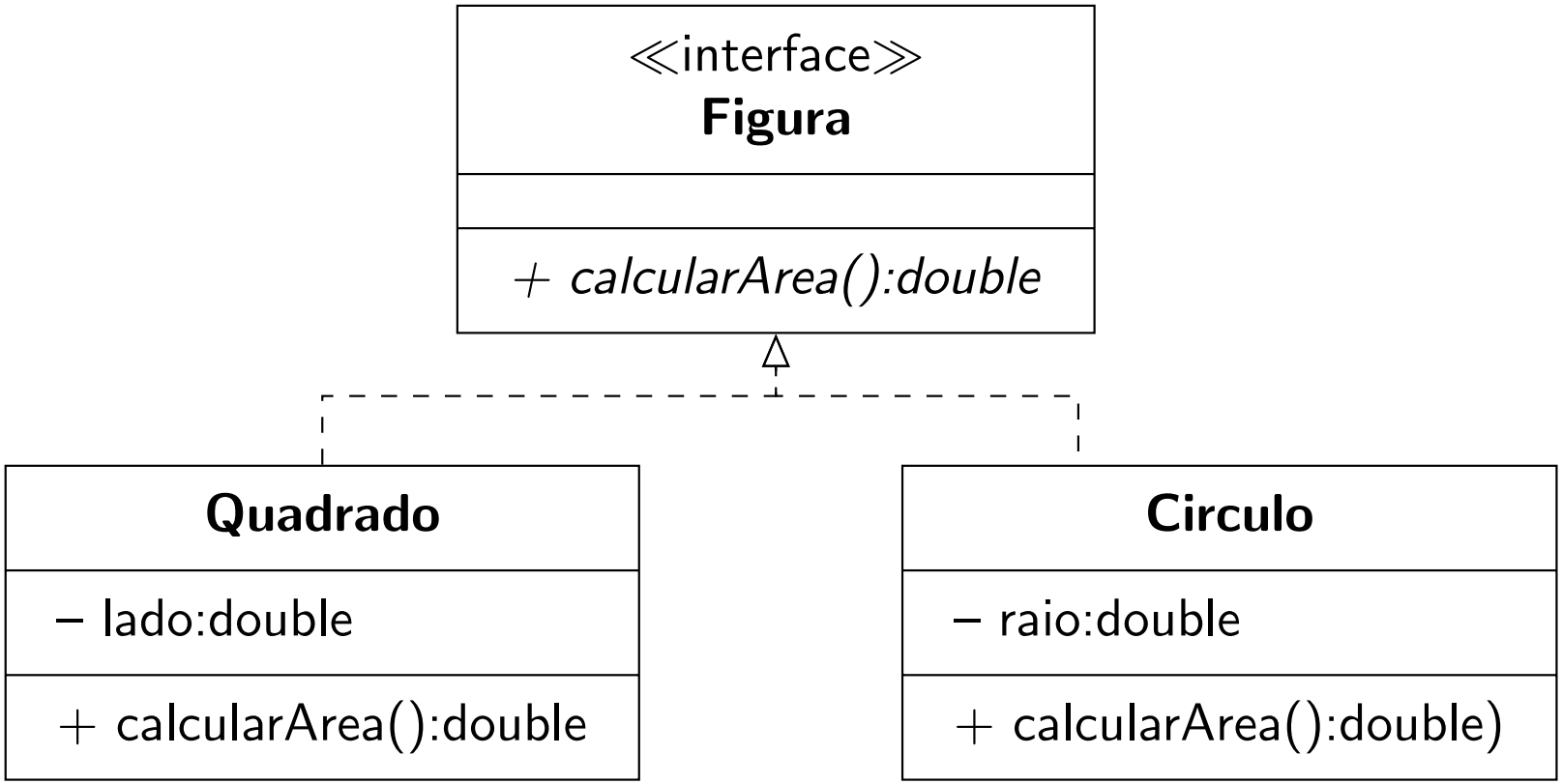
Unified Modeling Language (UML) XXIV



Unified Modeling Language (UML) XXV

- Uma **Interface** é representada de forma parecida com uma classe
- Não possui atributos e usa um estereótipo, palavra entre “<<” e “>>”
- O estereótipo `<<interface>>` denota uma interface
- Seus métodos devem ser públicos e abstratos
  - Identificador em itálico denota métodos abstratos

Unified Modeling Language (UML) XXVI

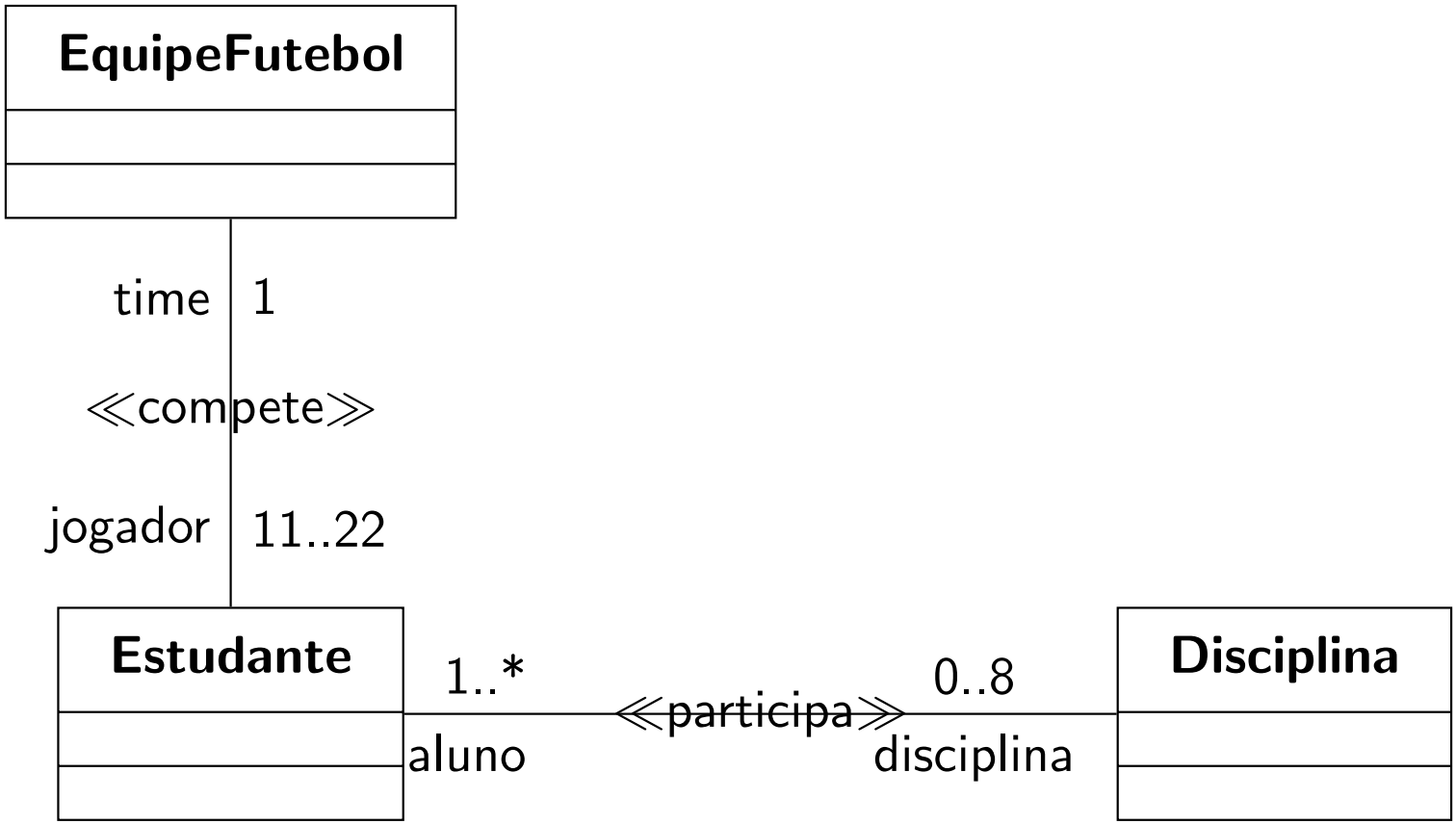


Como a classe Quadrado implementa a interface Figura, então, o método calcularArea() deve obrigatoriamente ser implementado

Exemplo I

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

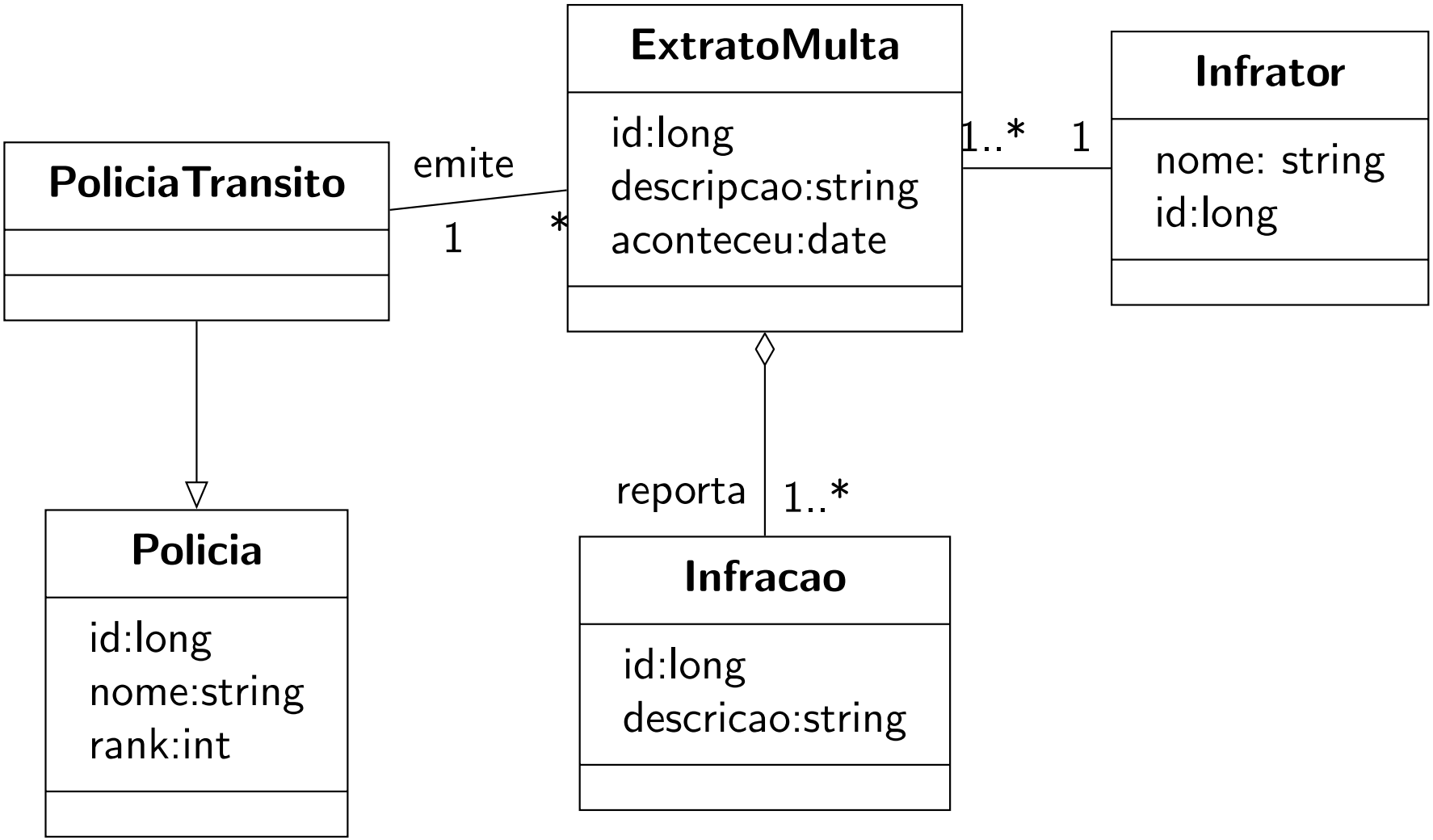
Exemplo II


EquipeFutebol
time
1
≪compete≫
jogador
11..22
Estudante
1..*
≪participa≫
0..8
Disciplina
aluno
disciplina

Exemplo III

- Emissão de extrato de multas de transito

Exemplo IV


ExtratoMulta
id:long
descripcao:string
aconteceu:date
Infrator
nome: string
id:long
PoliciaTransito
emite
1
*
1..*
1
Policia
id:long
nome:string
rank:int
reporta
1..*
Infracao
id:long
descricao:string

FIM