Diagramas de Classe: Conceitos e Notação

Prof. Me. Lucas Bruzzone

Aula 12

Diagrama de Classes

Objetivo

Mostrar a estrutura estática do sistema: classes, atributos, métodos e relacionamentos

Características:

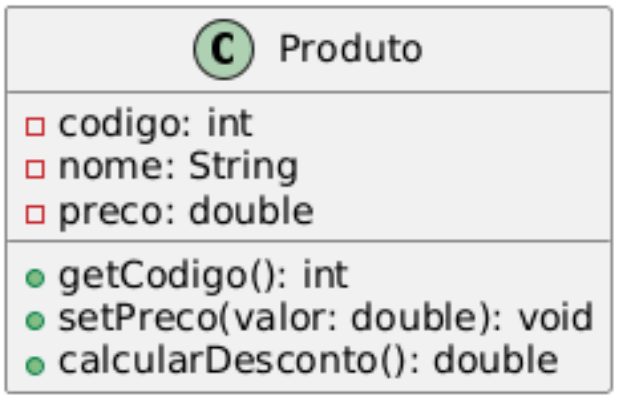
- Visão estrutural do sistema
- Base para implementação OO
- Mostra organização conceitual
- Independente de linguagem
- Evolui durante desenvolvimento

Usado em: Análise, projeto e documentação

Representação de Classes

Estrutura da classe:

- **Nome da classe** (compartimento superior)
- **Atributos** (compartimento do meio)
- **Métodos** (compartimento inferior)



Notação: Retângulo dividido em três compartimentos

Atributos

Sintaxe:

```
visibilidade nome: tipo [multiplicidade] = valorPadrao
```

Visibilidade:

- + público (public) - acessível de qualquer lugar
- - privado (private) - apenas dentro da classe
- # protegido (protected) - classe e subclasses
- ~ pacote (package) - mesmo pacote

Exemplos:

- `- nome: String`
- `+ idade: int = 0`
- `# salario: double`
- `- ativo: boolean = true`

Métodos

Sintaxe:

```
visibilidade nome(parametros): tipoRetorno
```

Exemplos:

- `+ getNome(): String`
- `- calcularIdade(): int`
- `+ setNome(nome: String): void`
- `+ processar(dados: List<String>): boolean`

Métodos especiais:

- **Construtor**: `+ Produto(nome: String)`
- **Estático**: `+ getInstance(): Produto` (sublinhado)
- **Abstrato**: *`+ calcular(): double`*

Tipos de Relacionamentos

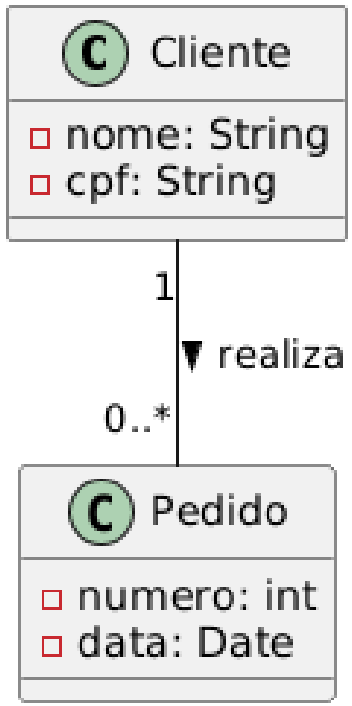
- **Associação**: Conexão conceitual entre classes
- **Agregação**: Relação "tem um" (parte-todo fraca)
- **Composição**: Relação "parte de" (parte-todo forte)
- **Generalização**: Relação "é um" (herança)
- **Dependência**: "Usa" temporariamente
- **Realização**: Implementa interface

Veremos cada um em detalhe com exemplos práticos

Associação

Características:

- Linha sólida conectando classes
- Pode ter nome e direção
- Multiplicidade em ambas as pontas
- Pode ser bidirecional ou unidirecional



Leitura: Um cliente realiza zero ou muitos pedidos

Multiplicidade

Notações comuns:

- **1**: Exatamente um
- **0..1**: Zero ou um (opcional)
- **0..*** ou *: Zero ou mais
- **1..***: Um ou mais
- **2..5**: Entre 2 e 5
- **3**: Exatamente 3

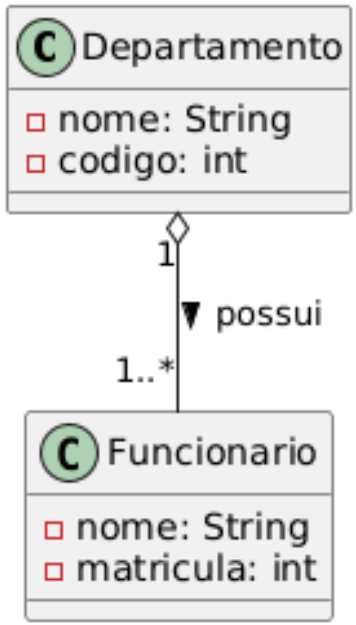
Exemplos práticos:

- Pessoa ——— Casa (1 para 0..1)
- Empresa ——— Funcionário (1 para 1..*)
- Curso ——— Aluno (* para *)

Agregação

Características:

- Relação "tem um" (has-a)
- Parte pode existir sem o todo
- Diamante vazio na classe "todo"
- Relacionamento mais fraco

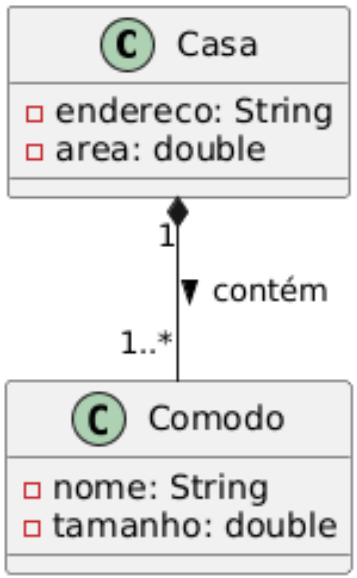


Interpretação: Funcionário pode existir sem departamento (transferência, demissão)

Composição

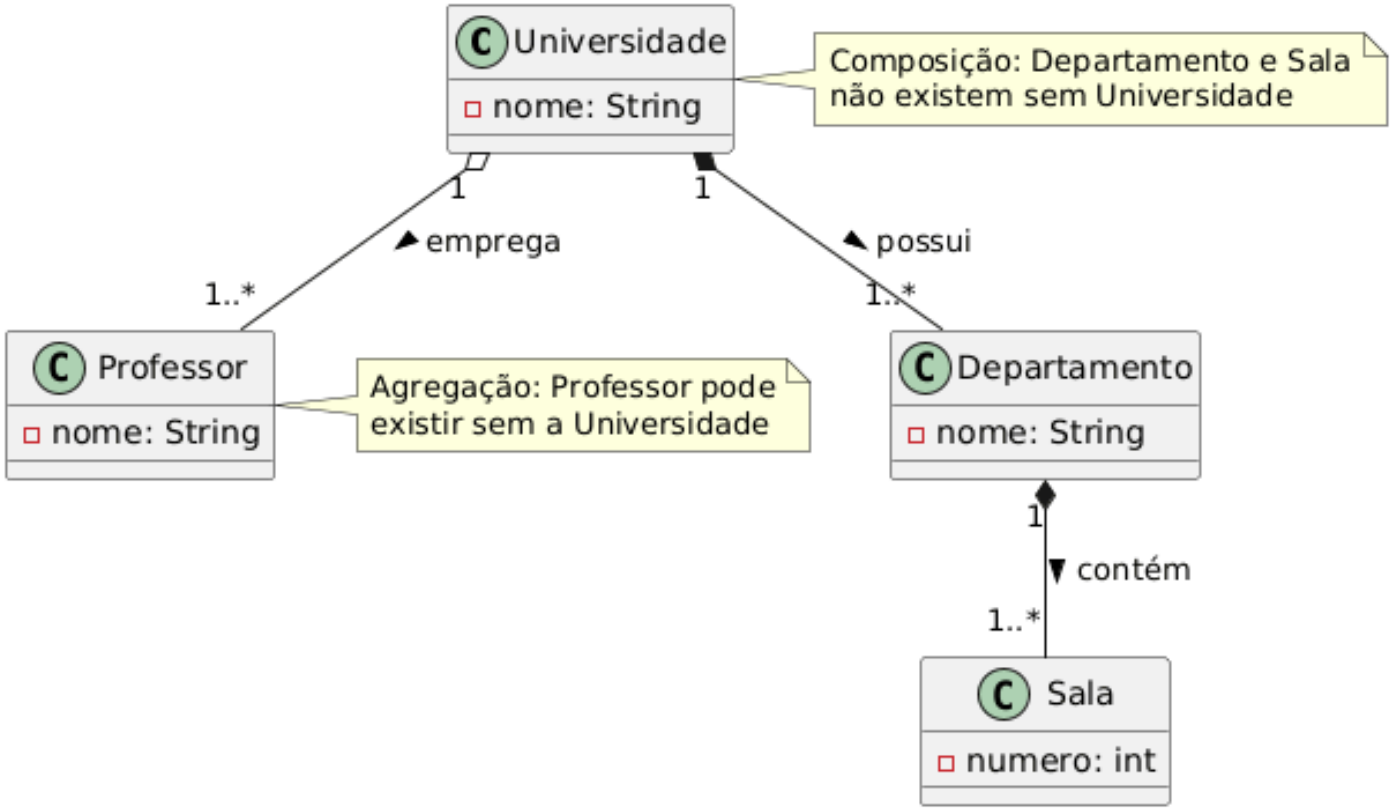
Características:

- Relação "parte de" mais forte
- Parte não existe sem o todo
- Diamante preenchido na classe "todo"
- Ciclo de vida dependente



Interpretação: Cômodo não existe sem a casa

Agregação vs. Composição



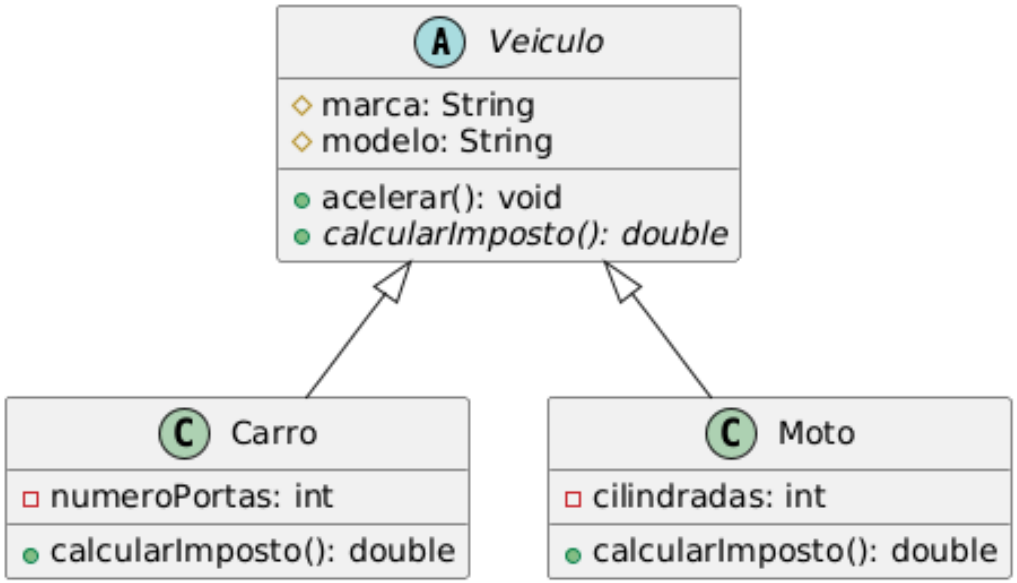
Agregação vs. Composição - Comparação

Aspecto	**Agregação**	**Composição**
Dependência	Fraca	Forte
Ciclo de vida	Independente	Dependente
Símbolo	Diamante vazio	Diamante cheio

Generalização (Herança)

Características:

- Relação "é um" (is-a)
- Seta com triângulo vazio
- Subclasse herda de superclasse
- Especialização/generalização



Interpretação: Carro e Moto são tipos especializados de Veículo

Dependência

Características:

- Relacionamento mais fraco
- "Usa" temporariamente
- Linha tracejada com seta
- Não mantém referência permanente

Exemplos de uso:

- Parâmetro de método
- Variável local
- Chamada de método estático
- Import/include

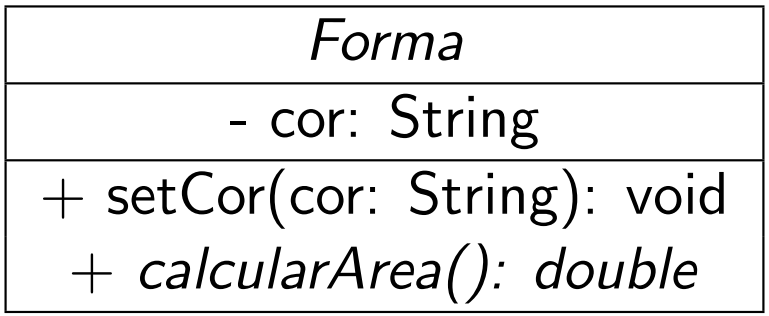
Quando usar: Classe A usa Classe B, mas não armazena referência

Classes Abstratas

Características:

- Não podem ser instanciadas
- Nome em itálico ou `<<abstract>>`
- Podem ter métodos abstratos (sem implementação)
- Base para herança
- Podem ter métodos concretos

Exemplo:

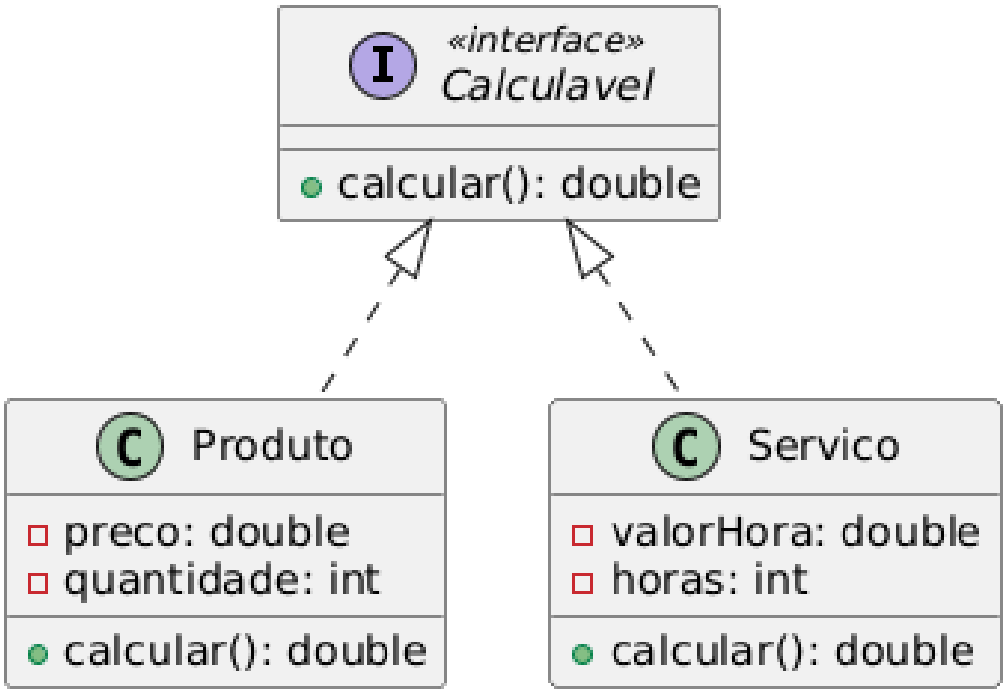


Uso: Define comportamento comum para subclasses

Interfaces

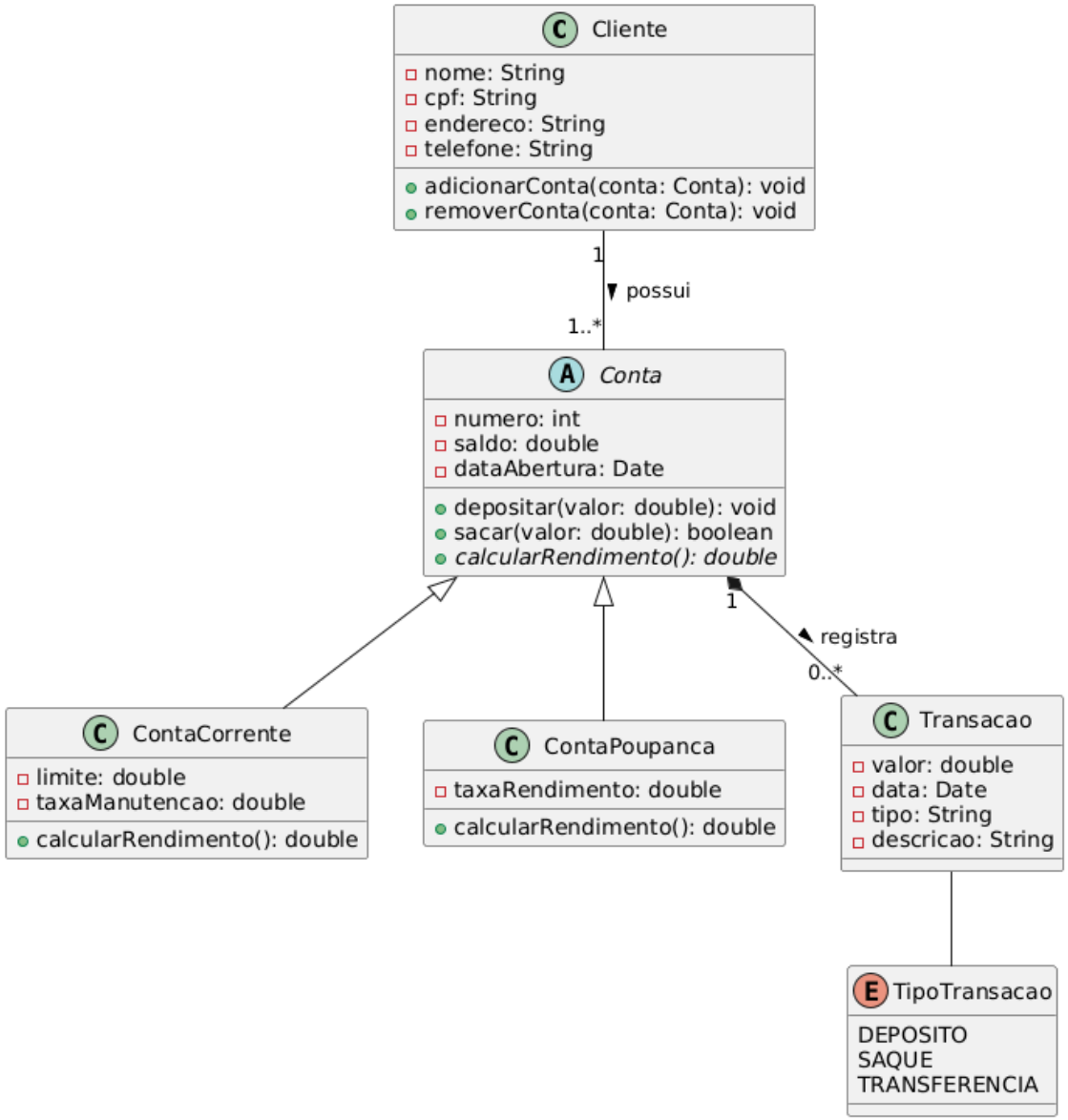
Características:

- Contrato de implementação
- Apenas assinaturas de métodos
- Estereótipo `<<interface>>`
- Relacionamento de realização (linha tracejada com triângulo)
- Não possui atributos de instância



Uso: Define comportamento que múltiplas classes podem implementar

Exemplo Completo - Sistema Bancário



Análise do Exemplo Bancário

Relacionamentos identificados:

- **Associação**: Cliente possui Contas (1 para 1..*)
- **Composição**: Conta registra Transações (1 para 0..*)
- **Herança**: ContaCorrente e ContaPoupanca estendem Conta
- **Associação**: Transacao usa TipoTransacao

Classes abstratas:

- **Conta**: Define comportamento comum, mas não pode ser instanciada
- Método abstrato `calcularRendimento()` implementado nas subclasses

Dicas de Modelagem

- **Nomes significativos**: Classes representam conceitos do domínio
- **Responsabilidade única**: Cada classe tem um propósito claro
- **Encapsulamento**: Atributos privados, métodos públicos seletivos
- **Relacionamentos claros**: Evitar muitas associações
- **Herança com cuidado**: Usar quando realmente "é um"
- **Composição sobre herança**: Preferir quando possível

Princípio: Começar simples e refinar iterativamente

Erros Comuns

- **Classes muito grandes**
  - Muitas responsabilidades (God Class)
- **Getters/setters para tudo**
  - Quebram encapsulamento
- **Herança por conveniência**
  - Usar quando não há relação "é um"
- **Relacionamentos confusos**
  - Multiplicidades incorretas
- **Muito detalhamento inicial**
  - Começar simples, refinar depois
- **Ignorar regras de negócio**
  - Modelo deve refletir domínio

Agregação ou Composição?

Use Agregação quando:

- Parte pode existir independentemente do todo
- Parte pode ser compartilhada entre vários "todos"
- Exemplo: Departamento e Funcionário

Use Composição quando:

- Parte não faz sentido sem o todo
- Ciclos de vida são dependentes
- Parte é exclusiva de um "todo"
- Exemplo: Casa e Cômodo, Pedido e ItemPedido

Ferramentas para Diagramas

Ferramentas populares:

- **Enterprise Architect**: Profissional completa
- **Visual Paradigm**: Rica em recursos
- **StarUML**: Gratuita e simples
- **Lucidchart**: Online colaborativa
- **Draw.io**: Gratuita online
- **PlantUML**: Baseada em texto (recomendada!)

Escolha baseada em necessidade e orçamento

Exercício Prático

Atividade

Baseado no trabalho do primeiro bimestre e nos casos de uso da aula anterior, desenvolva um diagrama de classes da sua aplicação

Requisitos:

- Identificar classes principais do domínio (5-10 classes)
- Definir atributos e métodos de cada classe
- Estabelecer relacionamentos entre classes
- Aplicar visibilidade adequada
- Usar herança ou interfaces quando apropriado
- Especificar multiplicidades

Entrega: Diagrama de classes + Justificativa dos relacionamentos

Dicas para o Exercício

- **Comece pelos substantivos** dos seus casos de uso
  - "Cliente faz pedido" $\rightarrow$ Classes: Cliente, Pedido
- **Identifique relacionamentos óbvios primeiro**
  - Associações básicas entre entidades
- **Adicione detalhes gradualmente**
  - Não tente modelar tudo de uma vez
- **Valide com casos de uso**
  - O modelo suporta os cenários?
- **Revise e refine**
  - Modelo evolui com entendimento

Próxima Aula

Diagrama de Sequência: Interações Dinâmicas

Estudaremos como modelar a interação entre objetos ao longo do tempo