Modelagem Conceitual

Prof. Saulo Popov Zambiasi

saulopz@gmail.com

Sumário

- Modelo Conceitual
- Usando UML para representação
- Paradigma da Orientação a Objetos
- Classes e Objetos
- Atributos e Operações
- Visibilidade e Encapsulamento
- Especialização / Generalização / Herança
- Associações na UML / Multiplicidade / Agregação / Tipo Associativo
- Dicas de Ferramentas de Modelagem
- Referências Bibliográficas

Modelo Conceitual

- Mostra
  - **conceitos** importantes no domínio do sistema
  - as **associações** entre esses conceitos.
- Deve **mostrar os principais elementos** envolvidos no sistema
- Ajuda a esclarecer a terminologia ou vocabulário do domínio.
- Não deve ser confundido com o modelo de dados

Modelo Conceitual: Elementos Básicos

- Conceitos
- Atributos
- Associações

Como Identificar Conceitos?

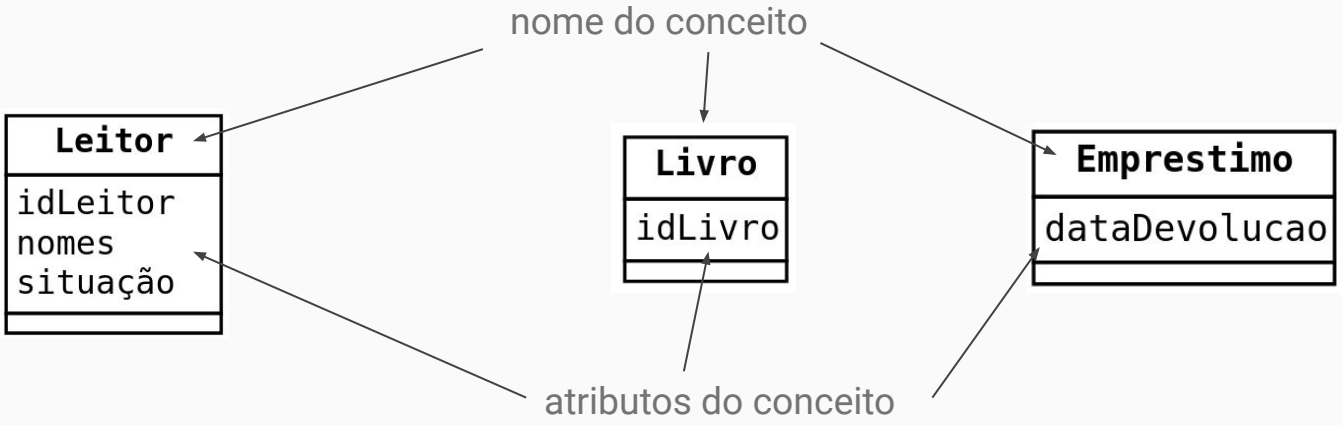
Alguns conceitos são bastante óbvios quando se descreve o sistema, como por exemplo os conceitos Leitor e Livro quando se fala em um sistema de Biblioteca.

Como Encontrar Conceitos?

- Verificar o texto dos casos de uso expandidos
- Selecionar termos que representam informação transmitida do e para o sistema
- Agrupar sinônimos

Como Identificar Atributos?

- Substantivos sublinhados no texto do caso de uso podem ser candidatos a atributos dos conceitos já identificados.
- Cautela: não torne o modelo conceitual muito complexo desnecessariamente.
  - limite-se a adicionar os atributos importantes para compreender o conceito, ou atributos que serão importantes para o futuro projeto do sistema.



Associações

- Relacionamento entre conceitos

Por exemplo:

- existe uma associação entre Empréstimo e Leitor
- quando o Empréstimo termina o Leitor deve ter sua situação regularizada.

Informação: Mais a frente veremos como representar associações usando UML.

Como Identificar Associações?

- Algumas são bastante simples de identificar, bastando aplicar certas regras.
- Outras são implícitas necessitam de mais experiência do analista para identificá-las na fase de requisitos, ou podem surgir mais adiante, na fase de projeto.
- **Cautela**: excesso de associações causa um efeito indesejado
  - modelo confuso e com pouca legibilidade.
- Portanto, uma regra básica é evitar colocar associações redundantes

Como Identificar Associações?

Regra 1: Um conceito que, fisicamente ou logicamente, faz parte de outro.

Exemplo:

- um livro que está fisicamente armazenado em uma estante
- um livro é um Item de Empréstimo que faz parte do Empréstimo.

Como Identificar Associações?

Regra 2: Um conceito que serve para descrever ou qualificar outro conceito.

Exemplo:

- um Livro pode ser classificado em diversas Categorias ou por Autor;
- um Item de Estoque que é descrito por uma Especificação de Produto.

Como Identificar Associações?

Regra 3: Um conceito que é responsável por registrar ou manter informações sobre outro.

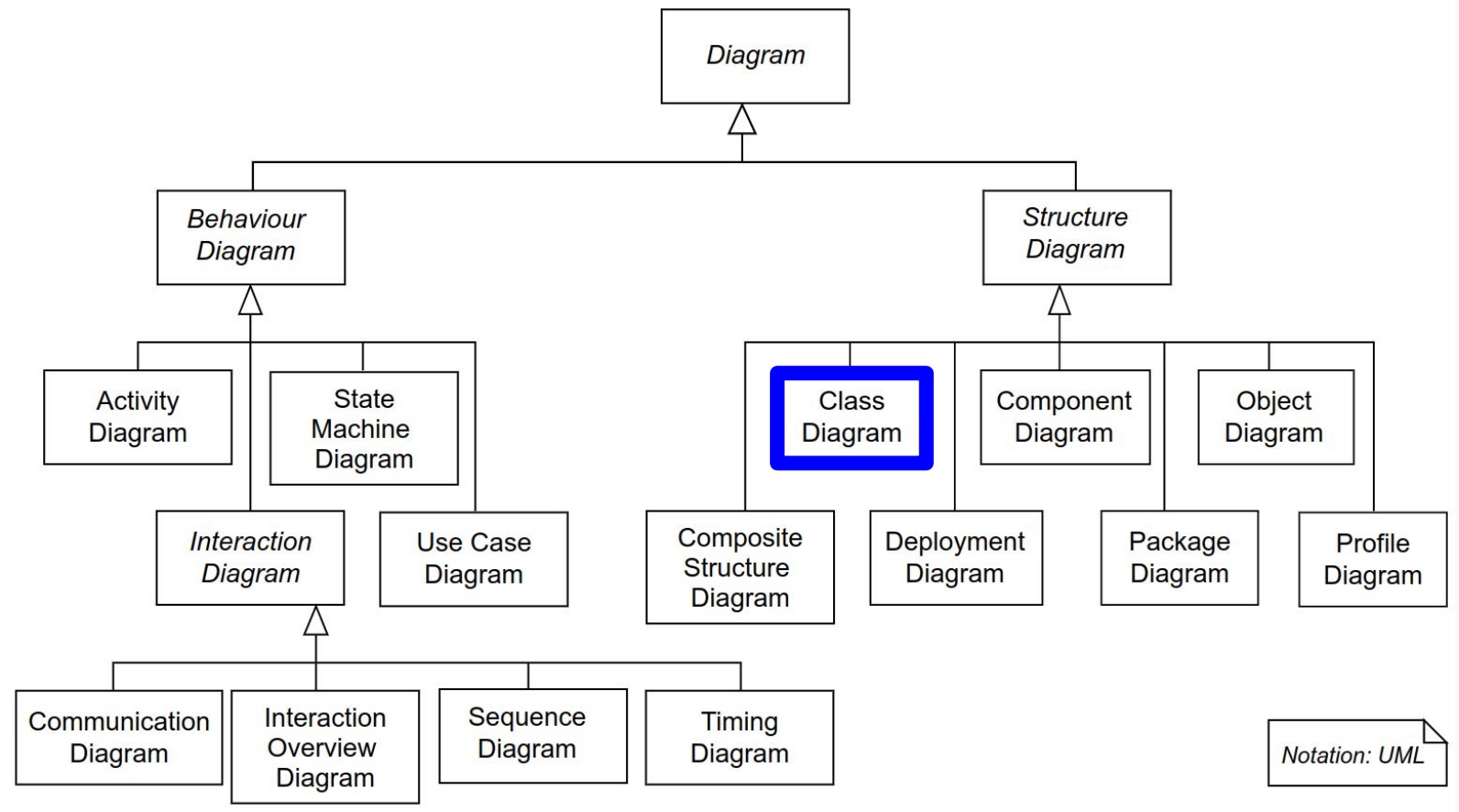
Exemplo:

- o Atendente é quem registra e atende o Leitor;
- a Bibliotecária é responsável pelos Livros.

Usando UML para representar

- UML - *Unified Modeling Language*
- UML era uma tentativa de padronização (final dos anos 90).
- Descrição de sistemas de forma consistente e clara.
- Reuniu as melhores ideias das metodologias utilizadas na época.

Diagramas UML



Paradigma da Orientação a Objetos

- **Abstração de conceitos do mundo real**.
- Modelo de análise, projeto e programação de sistemas de software.
- Composição e interação entre diversas unidades de software (objetos).
- Conjunto de classes que definem os objetos presentes no sistema de software.
- Cada classe determina o comportamento (métodos) e estados (atributos) de seus objetos e relacionamento com outros objetos.
- Linguagens POO: C++, C#, Java, Go, Python, Smalltalk, etc.

Classes e Objetos

Dica: Um objeto está para sua classe, assim como uma variável está para seu atributo.

Classes

São os **Conceitos** do modelo conceitual

Capturar uma e somente uma abstração chave.

Deve ser um substantivo singular que melhor caracteriza a abstração:

Professor, Aluno, Turma, ...

Objetos

São **instâncias** de classe

Estão presente em tempo de execução e se comportam conforme definido pela sua classe.

Todo objeto tem:

Estado, Comportamento e Identidade

Representação da relação entre Classe e Objeto (instância)

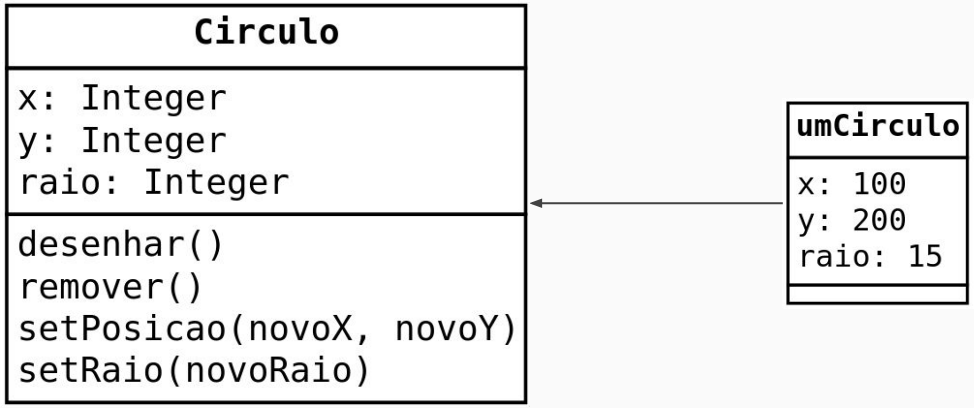
A Classe Pessoa não existe como Objeto

Fabiana é um Objeto instância da Classe Pessoa.

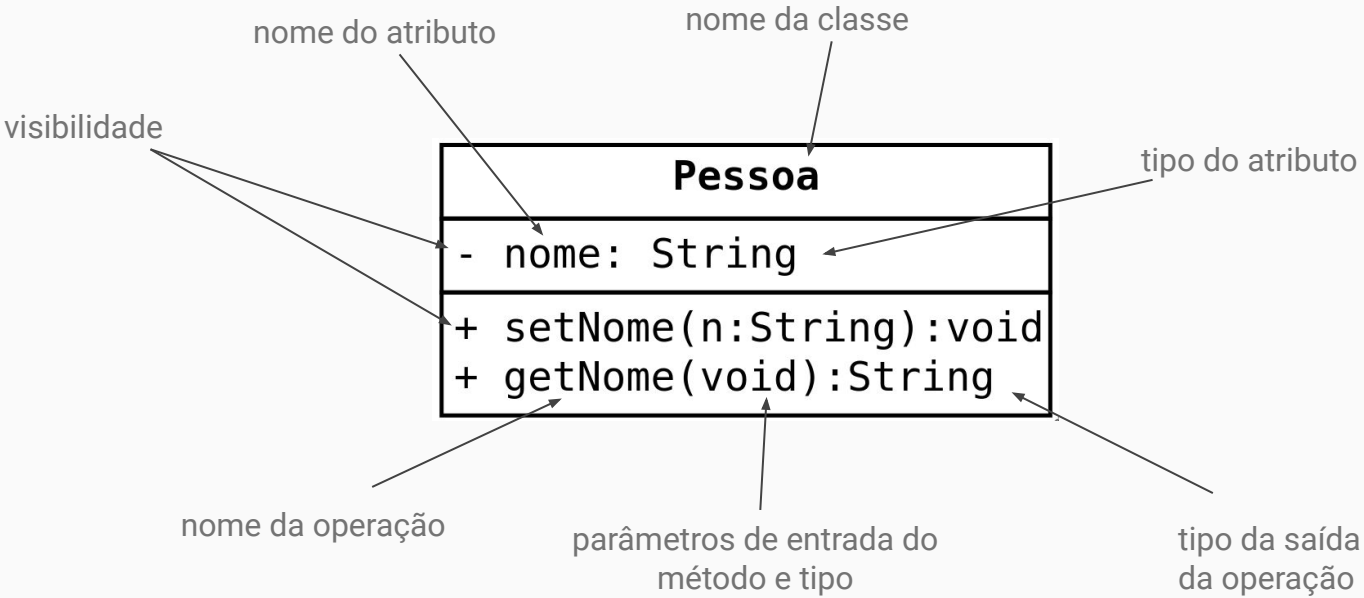
Atributos e Operações

Atributos:
- Atributos do Objeto
- Componentes e informações

Operações:
- Métodos do Objeto
- Comportamento dos objetos



Classe



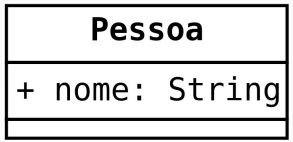
Importante: A título de demonstrar alguns conceitos, nem sempre vamos mostrar aqui todos os elementos da classe, apenas o que é necessário para o entendimento.

Visibilidade

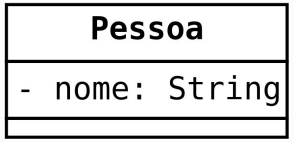
Visibilidade da classe: Especifica como os atributos e/ou operações poderão ser acessados por outros objetos

- **Privada (–)**: atributos e operações só podem ser acessados na implementação da própria classe;
- **Pública (+)**: atributos e operações são visíveis dentro da própria classe e para todas as outras classes;
- **Protegida (#)**: será discutida após discutirmos o conceito de herança

Visibilidade e Encapsulamento




```java
public class Pessoa {
    public String nome;
}
```



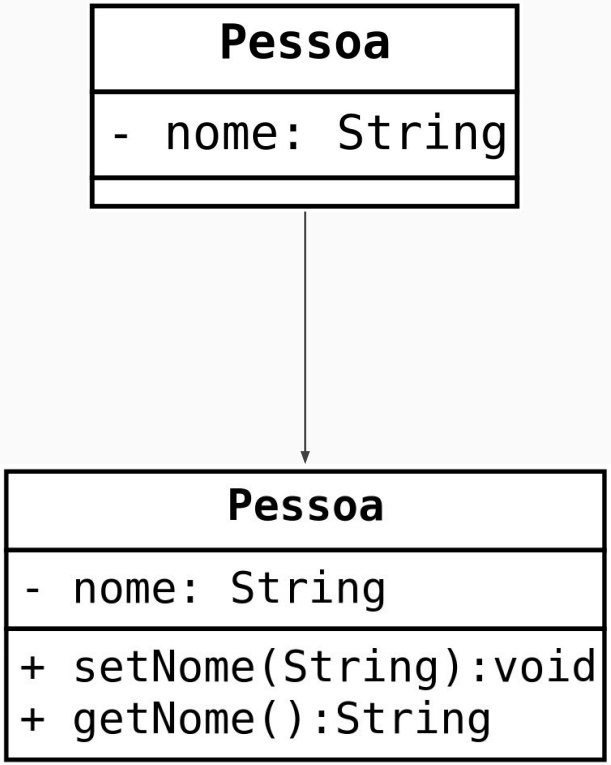

```java
public class Pessoa {
    private String nome;
}
```

```java
Pessoa p = new Pessoa();
p.nome = "Saulo Popov Zambiasi";
System.err.println(p.nome);
```

```
saulo@lain:~/Área de trabalho/seletivo/aula/java$
e/96f3bc99fc3e4b4c1f0732df98bca3e0/redhat.java/jdt
Saulo Popov Zambiasi
```

```
Main.java ~/Documentos/Concursos/2024_Univali/aula/java 2
  The field Pessoa.nome is not visible Java(33554503) [Ln 4, Col 11]
  The field Pessoa.nome is not visible Java(33554503) [Ln 5, Col 30]
Pessoa.java ~/Documentos/Concursos/2024_Univali/aula/java 1
  The value of the field Pessoa.nome is not used Java(570425421) [Ln 2, Col 20]
```

Encapsulamento - *getters* e *setters*



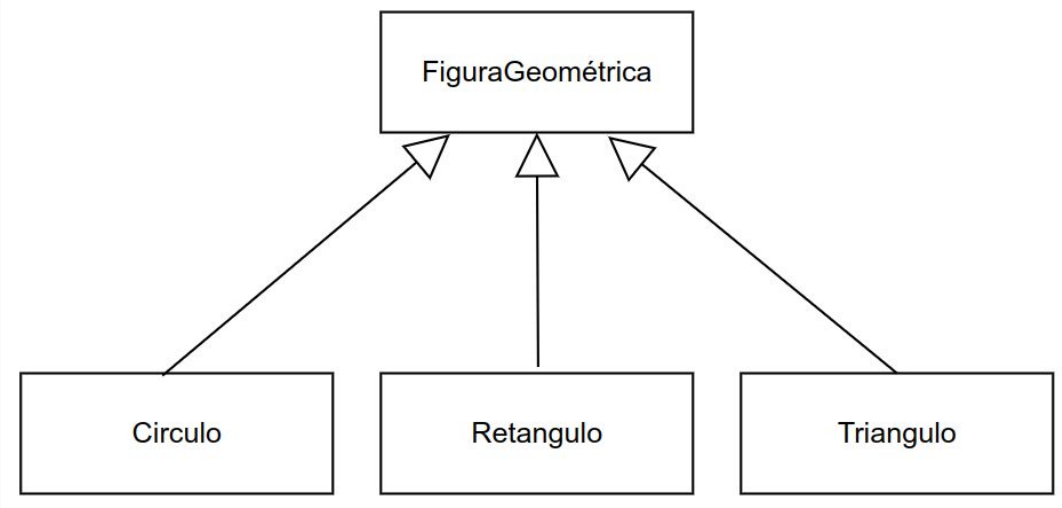

```java
public class Pessoa {
    private String nome;

    public void setNome(String n) {
        nome = n;
    }

    public String getNome() {
        return nome;
    }
}
```

```java
Pessoa p = new Pessoa();
p.setNome(n:"Saulo Popov Zambiasi");
System.out.println(p.getNome());
```

```
saulo@lain:~/Área de trabalho/selet
sages -cp /home/saulo/.config/Code/
Saulo Popov Zambiasi
```

Especialização / Generalização / Herança

- Superclasse fornece atributos e operações
- Subclasses derivadas possuem todas as propriedades da sua classe superior.
- Herdam atributos e comportamentos



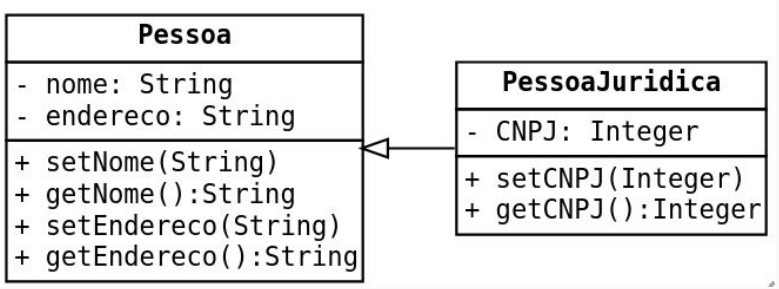
FiguraGeometrica é uma **generalização** de Circulo

Circulo é uma **especialização** de FiguraGeometrica

Especialização / Generalização / Herança

- Um objeto de uma subclasse herda e pode utilizar todos os atributos da superclasse

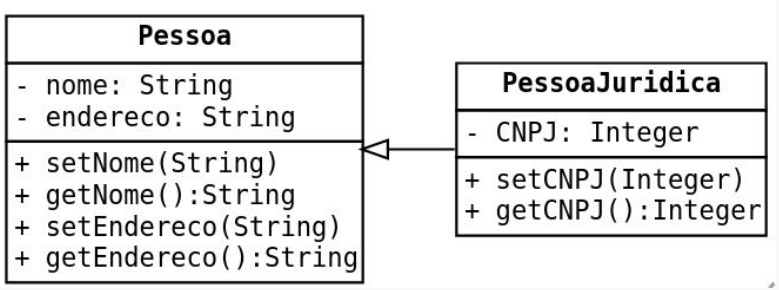
```java
public class Pessoa {
    private String nome;
    private String endereco;

    public void setNome(String n) {
        nome = n;
    }

    public String getNome() {
        return nome;
    }

    public void setEndereco(String e) {
        endereco = e;
    }

    public String getEndereco() {
        return endereco;
    }
}
```




```java
public class PessoaJuridica extends Pessoa {
    private Integer CNPJ;

    public void setCNPJ(Integer cnpj) {
        CNPJ = cnpj;
    }

    public Integer getCNPJ() {
        return CNPJ;
    }
}
```

- Um objeto de uma subclasse herda e pode utilizar todos os atributos da superclasse

```java
public class Main {
    Run | Debug
    public static void main(String[] args) {
        PessoaJuridica p = new PessoaJuridica();
        p.setNome(n:"Saulo Popov Zambiasi");
        p.setEndereco(e:"Florianópolis");
        p.setCNPJ(cnpj:9999999);
        System.out.println(p.getNome() + ", " +
                p.getEndereco() + ", " +
                p.getCNPJ());
    }
}
```




```
saulo@lain:~/Área de trabalho/seletivo/aula/java$
sages -cp /home/saulo/.config/Code/User/workspaceSt
Saulo Popov Zambiasi, Florianópolis, 9999999
```

É um relacionamento de especialização e generalização

“é um” ou “tipo de”

Logo, PessoaJuridica **é uma** Pessoa

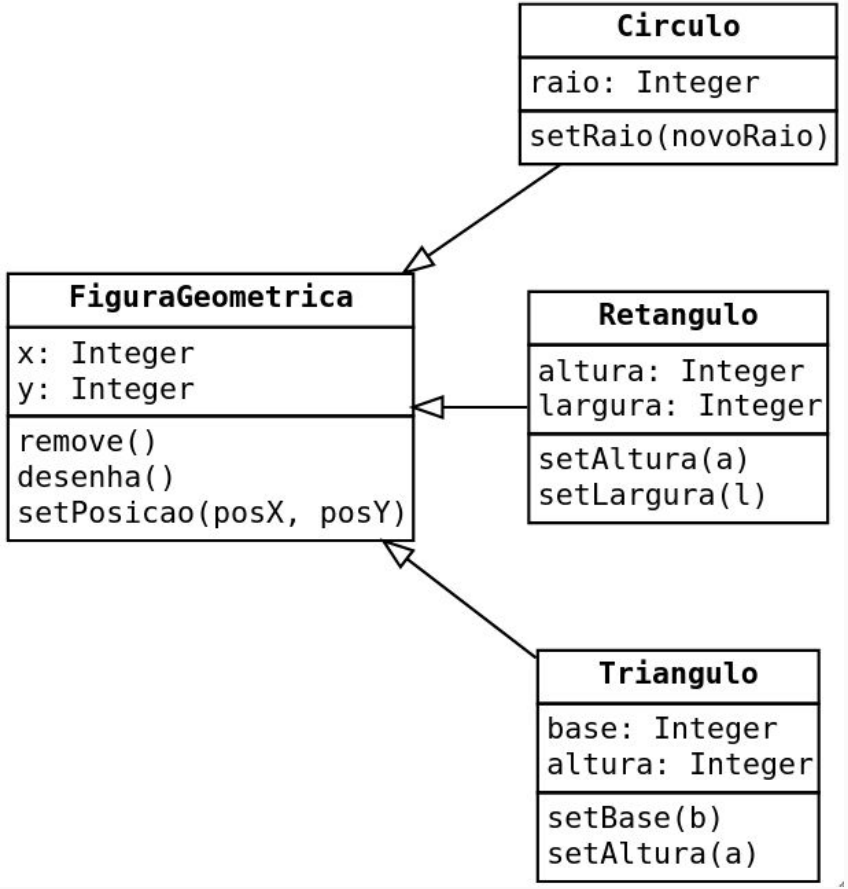
Especialização / Generalização / Herança

Tipos de Herança:

- **Herança simples**: a subclasse herda de uma única superclasse.
- **Herança múltipla**: a subclasse herda de duas ou mais Superclasses.

Observação: a herança múltipla pode tornar a organização um pouco confusa, tanto que muitos evitam e algumas linguagens OO não permitem.

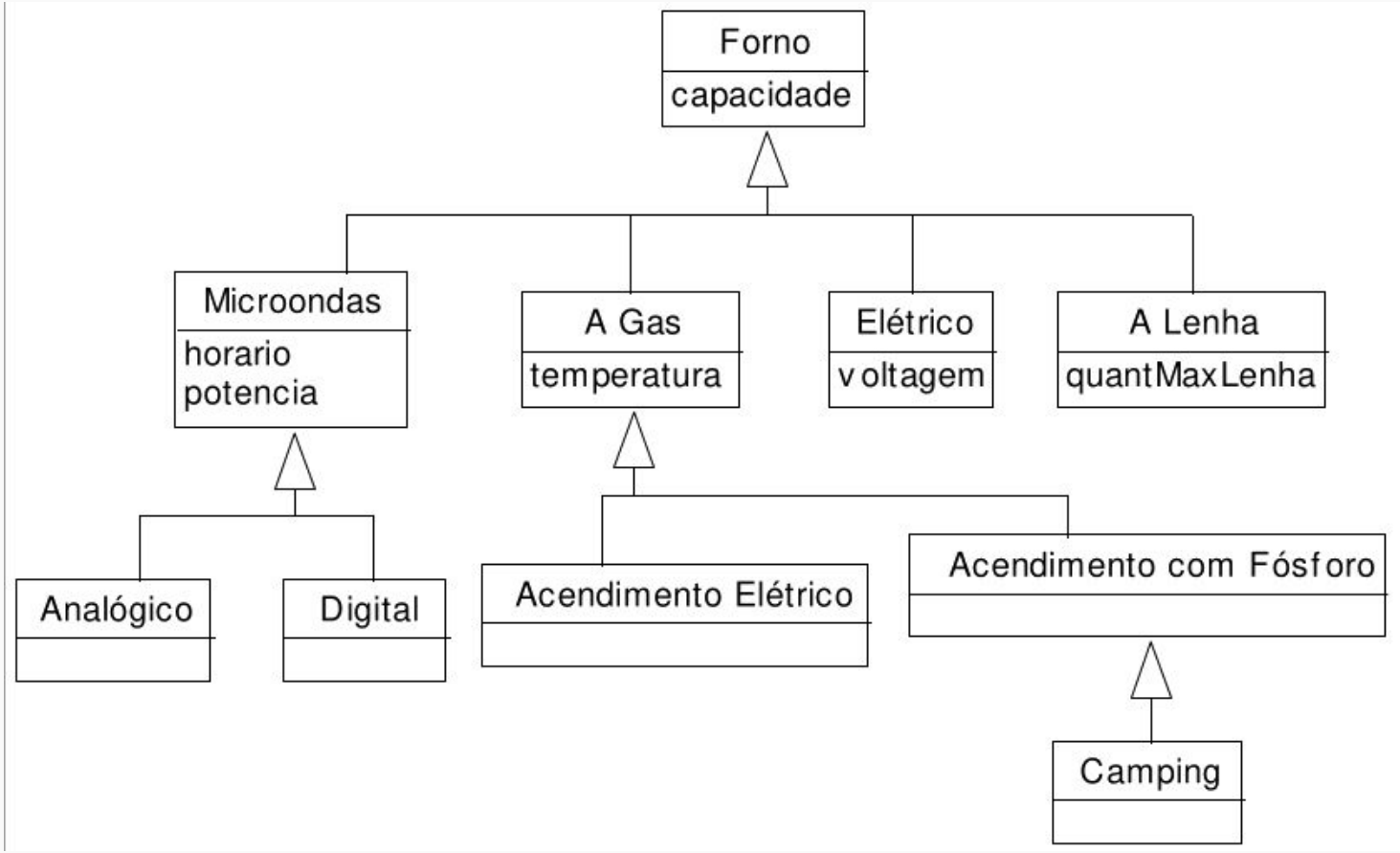
Especialização / Generalização / Herança



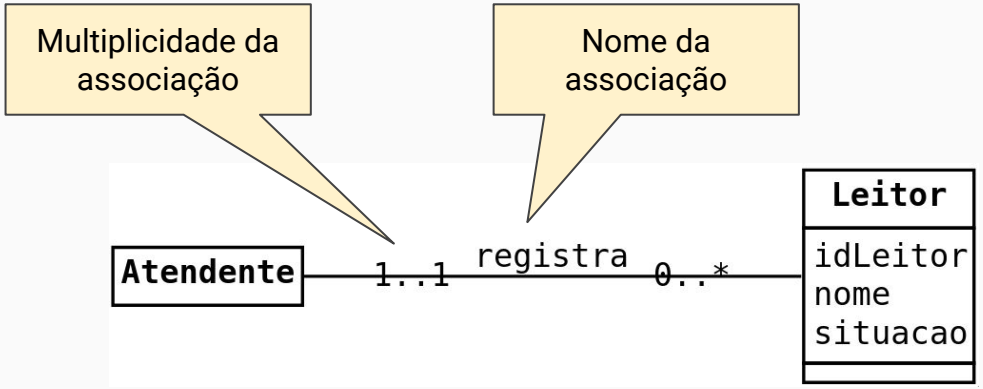
Uma subclasse pode:

- Adicionar atributos, operações e relacionamentos.
- Redefinir operações herdadas.

Especialização / Generalização / Herança



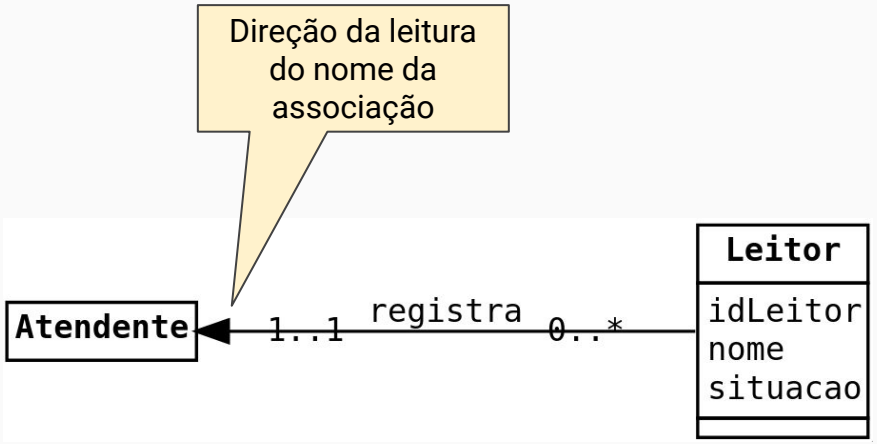
Associações na UML



Lê-se:

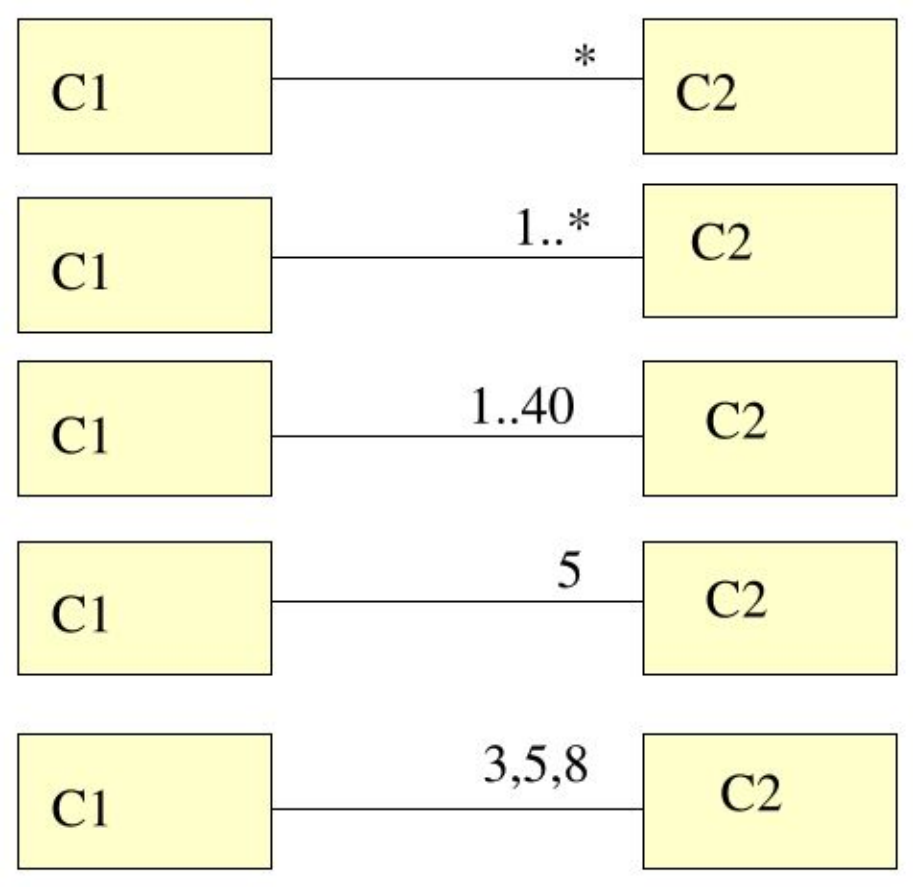
- Um **Atendente** registra nenhum ou mais **Leitores**.
- Um **Leitor** é registrado por um **Atendente**.

Associações na UML



Dica: A direção só deve ser colocada se for da direita para a esquerda ou de baixo para cima.

Multiplicidade



zero (nenhum) ou mais muitos(as)

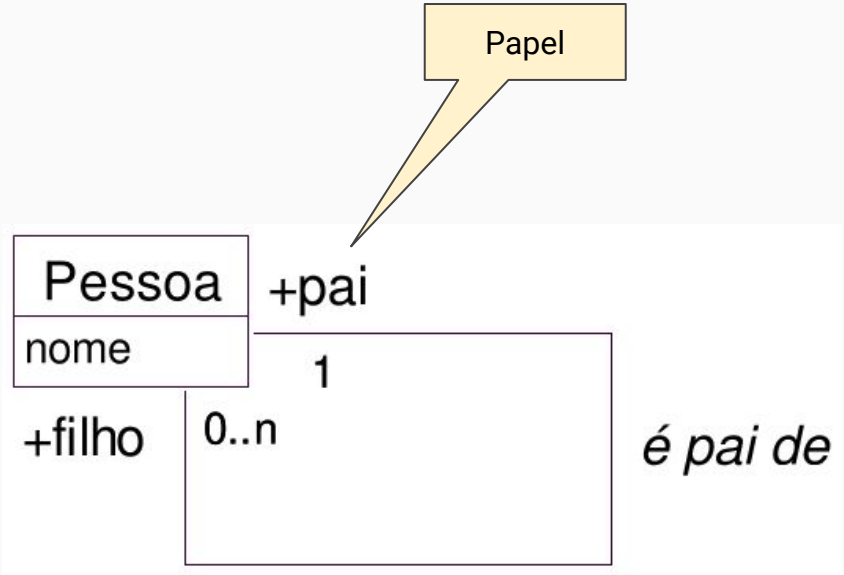
um ou mais

um ou quarenta

exatamente cinco

exatamente três, cinco ou oito

Associação Reflexiva



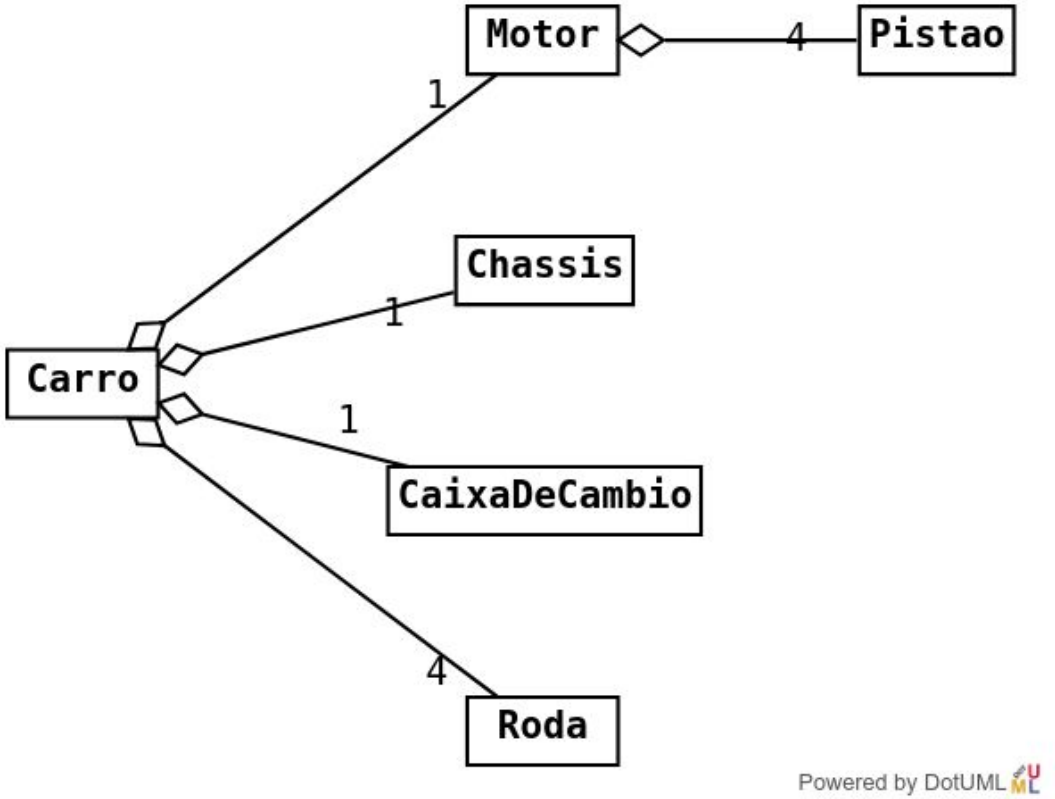
Agregação

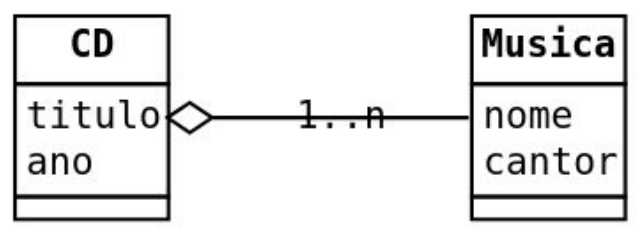
- Mecanismo para incluir atributos (e comportamento) de outros tipos a ele agregados
- Indica a existência de um todo, composto por partes.

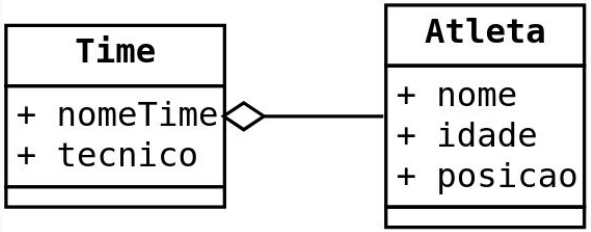
Exemplo: um carro consiste de:

  - 4 rodas
  - um motor
  - chassis
  - caixa de câmbio, etc.

Agregação / Composição



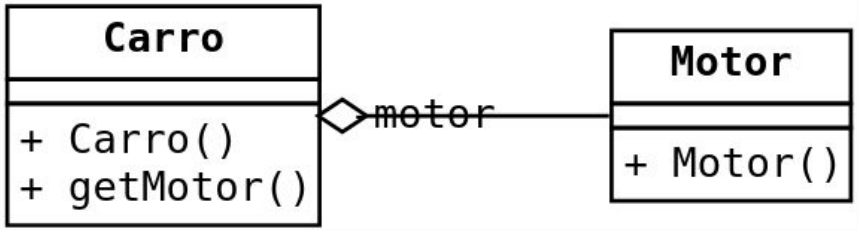




Importante: Na Agregação, a existência do Objeto-Parte faz sentido, mesmo não existindo o Objeto-Todo

Agregação / Composição

```java
public class Carro {
    public String chassis;
    private Motor motor;

    // A Parte é recebida como parâmetro
    // para se juntar ao Todo
    public Carro(Motor m) {
        motor = m;
    }

    // Permite recuperar a Parte para
    // compartilhar com outro Todo
    public Motor getMotor() {
        return motor;
    }
}
```

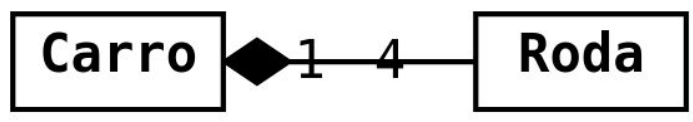



```java
class Motor {
    //...
}
```

```java
public class Main {
    Run | Debug
    public static void main(String[] args) {
        // Construtor com argumento criou a Parte
        Carro c1 = new Carro(new Motor());
        // Construtor com argumento recebeu a
        // Parte de outro Todo
        Carro c2 = new Carro(c1.getMotor());
    }
}
```

Agregação Composta

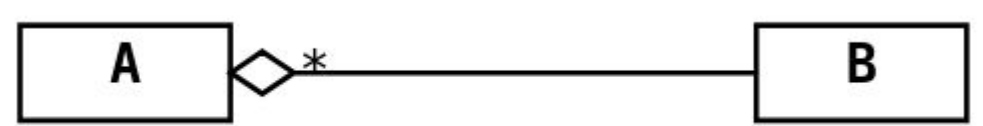
- A **agregação composta** ou **composição** ocorre quando a multiplicidade na extremidade do composto pode ser no máximo 1. A notação em UML é um losango negro



- **forte dependência** do “todo” em relação à “parte”
- tempo de vida delimitado (**as “partes” não podem existir sem o “todo”**).
- algumas propriedades do “todo” se propagam para suas “partes”
  - ex: disposição, localização, criação e destruição das “partes”.

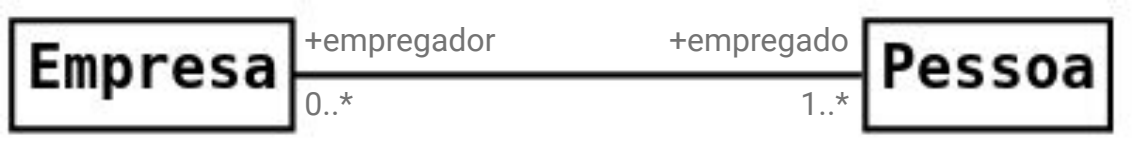
Agregação Compartilhada

- A **agregação compartilhada**, denotada em UML por um losango **vazio**, denota que a multiplicidade na extremidade do composto pode ser maior do que um.



Tipo Associativo

- É uma associação que também possui propriedades de tipo
  - ou um tipo que tem propriedades de uma associação.



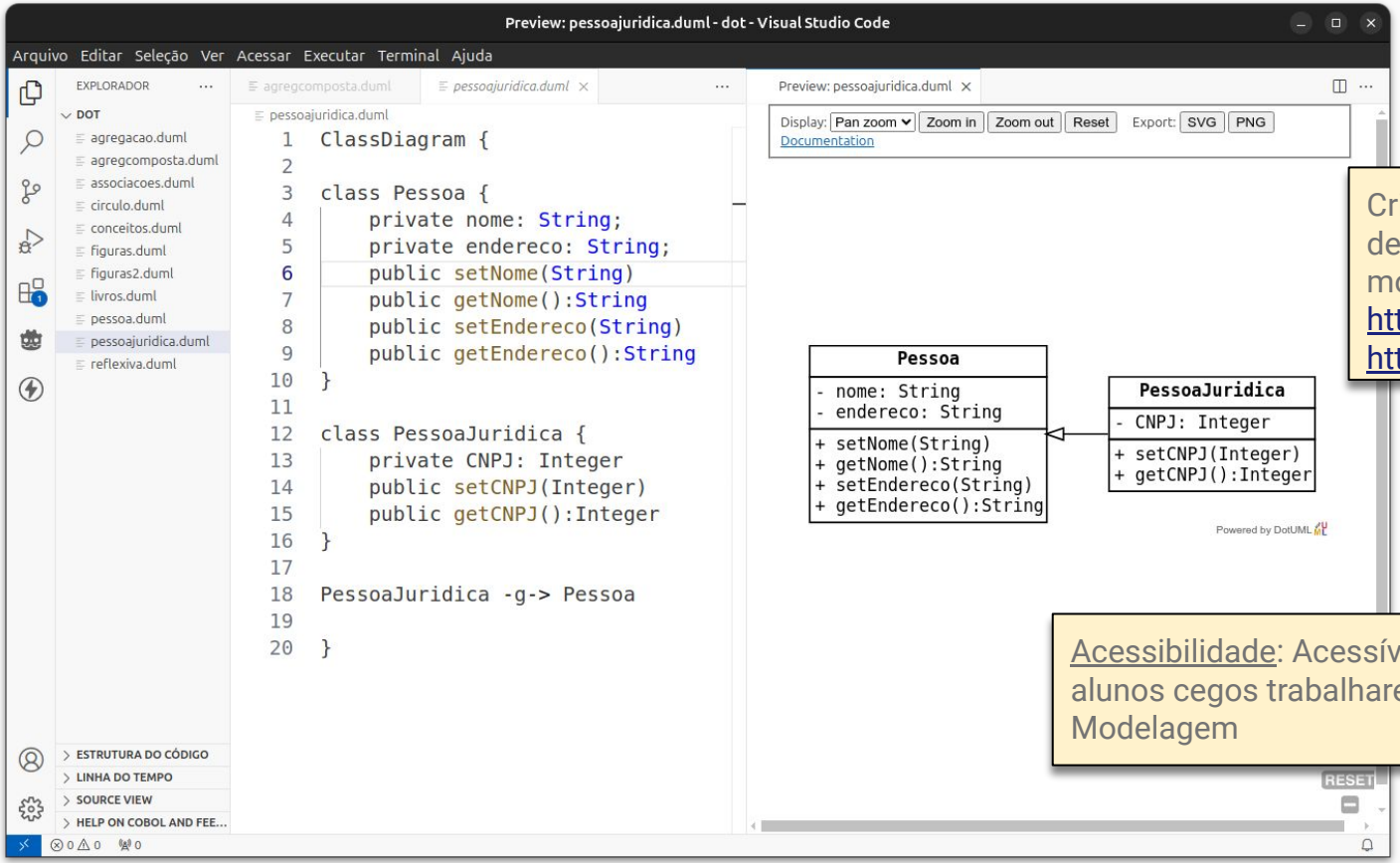
Tipo Associativo

Indícios de que um tipo associativo pode ser útil em um modelo conceitual:

- Um atributo está relacionado com uma associação
- **As instâncias do tipo associativo têm um tempo de vida dependente do tempo de vida da associação**
- Existe uma associação muitos-para-muitos entre dois conceitos, bem como informações relacionadas à associação propriamente dita

Dicas de Ferramentas de Modelagem

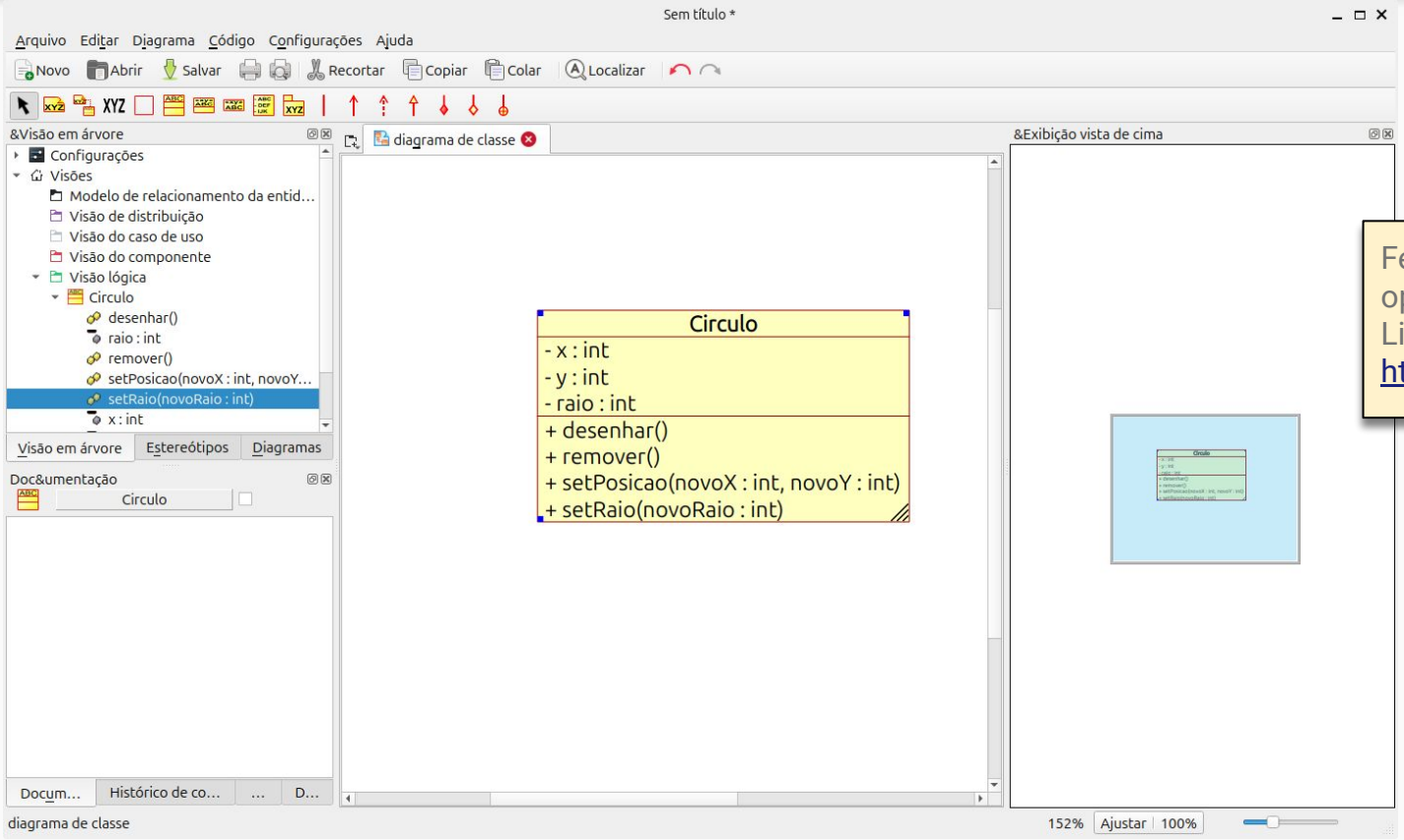
Visual Studio com DotUML



Criar diagramas UML por meio de usando uma linguagem de modelagem
https://code.visualstudio.com/
https://dotuml.com/

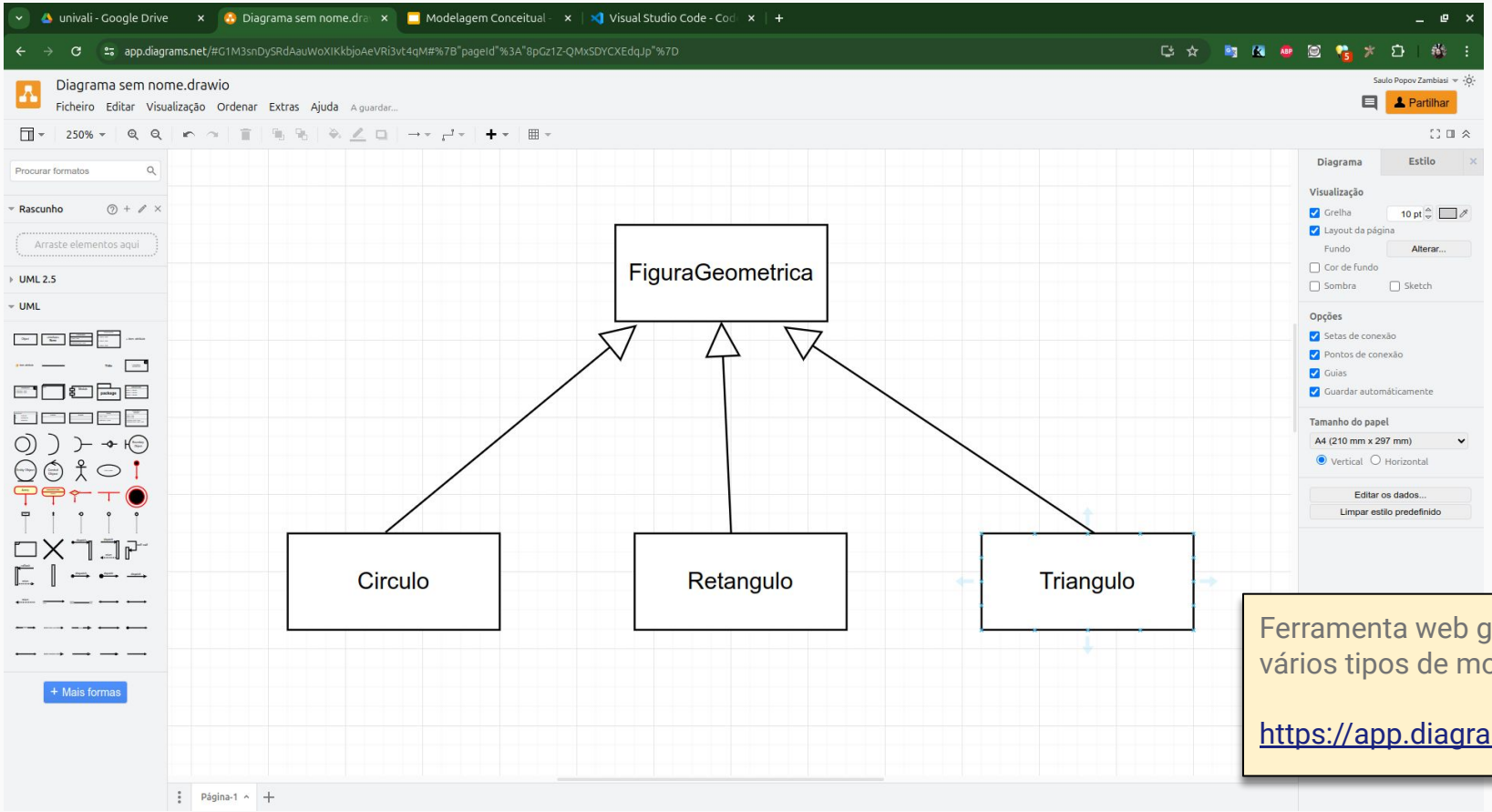
Acessibilidade: Acessível para alunos cegos trabalharem com Modelagem

Umbrello



Ferramenta de modelagem open source para Windows, Linux e MacOS
https://uml.sourceforge.io/

Draw.io



Ferramenta web gratuita para vários tipos de modelagens

https://app.diagrams.net/

Referências Bibliográficas

- FOWLER, Martin e SCOTT, Kendall. Uml Essencial. 2a. Edição. Bookman. Porto Alegre, 2000.
- SCHNEIDER, Geri. Applying use case: a practical guide. Addison-Wesley, 1998.
- OESTEREICH, Bernd. Developing Software with UML. Addison-Wesley, 1999.
- Wiki do professor: https://saulo.arisa.com.br