

Diagrama de Classes UML

Notação, relacionamentos e correspondência com código Java

1. Introdução

O **Diagrama de Classes UML** (*Unified Modeling Language*) é uma representação gráfica utilizada para modelar a estrutura estática de um sistema orientado a objetos. Ele mostra as classes do sistema, seus atributos, suas operações (métodos) e os relacionamentos existentes entre elas, servindo como um projeto visual que pode ser traduzido diretamente em código, por exemplo em Java.

Antes de escrever o código, é comum projetar o sistema por meio de diagramas de classes, o que facilita a comunicação entre desenvolvedores e a visualização da arquitetura antes da implementação.

2. Estrutura Básica de uma Classe no Diagrama

No diagrama de classes, cada classe é representada por um retângulo dividido em três compartimentos: o **nome da classe** (topo), os **atributos** (meio) e as **operações** (base). Os símbolos de visibilidade que antecedem atributos e operações seguem o mesmo conceito dos modificadores de acesso em Java.

Pessoa
- nome: String - idade: int
+ apresentar(): void + getNome(): String

Figura 1 — Estrutura de uma classe no diagrama UML: nome, atributos e operações.

2.1. Símbolos de Visibilidade

Símbolo UML	Visibilidade	Equivalente em Java
+	Público	public
-	Privado	private
#	Protegido	protected
~	Pacote	default (sem modificador)

O código Java correspondente à classe Pessoa da Figura 1 é apresentado a seguir:

```

public class Pessoa {
    private String nome;    // -
    private int idade;      // -

    public void apresentar() {           // +
        System.out.println("Olá, meu nome é " + nome);
    }

    public String getNome() {           // +
        return nome;
    }
}

```

3. Relacionamentos entre Classes

Além de descrever cada classe isoladamente, o diagrama UML representa como as classes se relacionam entre si. Os relacionamentos mais comuns são: **associação**, **agregação**, **composição**, **herança (generalização)** e **dependência**.

3.1. Herança (Generalização)

Representada por uma linha sólida com uma seta de ponta vazada (triângulo) apontando para a superclasse, indica que a subclasse herda atributos e operações da classe base. Em Java, corresponde à palavra-chave `extends`.

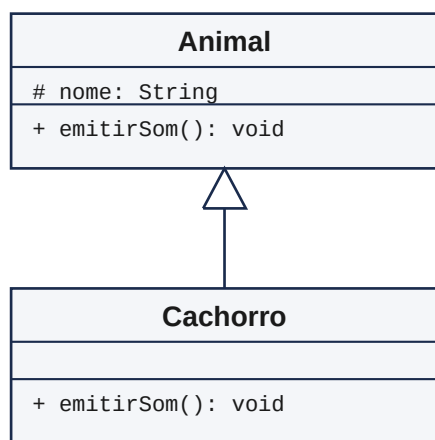


Figura 2 — Herança: Cachorro estende Animal (seta vazada apontando para a superclasse).

```

public class Animal {
    protected String nome;

    public void emitirSom() {
        System.out.println(nome + " faz um som.");
    }
}

public class Cachorro extends Animal {
    @Override
    public void emitirSom() {
        System.out.println(nome + " late: Au au!");
    }
}

```

3.2. Associação

Representada por uma linha simples ligando duas classes, indica que os objetos de uma classe se relacionam com objetos de outra, geralmente com multiplicidades indicando quantos objetos participam da relação (por exemplo, 1 para *). Em Java, a associação normalmente é implementada como um atributo que referencia outra classe.

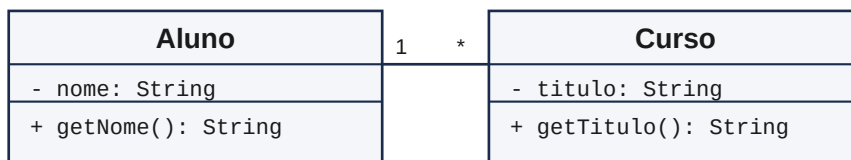


Figura 3 — Associação: um Aluno pode estar matriculado em vários Cursos (multiplicidade 1 para *).

```

import java.util.ArrayList;
import java.util.List;

public class Curso {
    private String titulo;

    public String getTitulo() {
        return titulo;
    }
}

public class Aluno {
    private String nome;
    private List<Curso> cursos = new ArrayList<>(); // associação 1 para *

    public void matricular(Curso curso) {
        cursos.add(curso);
    }
}

```

3.3. Composição

Representada por uma linha com um losango preenchido no lado do "todo", indica uma relação forte de posse, em que a parte não existe sem o todo: se o objeto principal for destruído, os objetos que o compõem também deixam de existir. Em Java, isso costuma ser implementado criando o objeto "parte" dentro do construtor do objeto "todo".

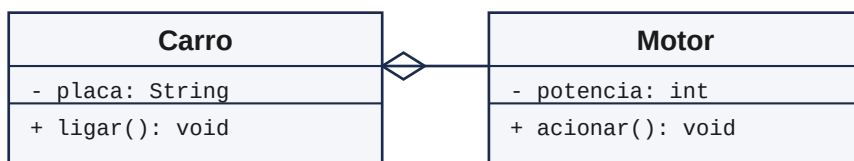


Figura 4 — Composição: Carro é composto por Motor (losango preenchido do lado do todo).

```
public class Motor {
    private int potencia;

    public void acionar() {
        System.out.println("Motor acionado.");
    }
}

public class Carro {
    private String placa;
    private final Motor motor; // composição: o motor é criado junto com o carro

    public Carro(String placa) {
        this.placa = placa;
        this.motor = new Motor(); // Motor não existe fora de um Carro
    }

    public void ligar() {
        motor.acionar();
        System.out.println("Carro " + placa + " ligado.");
    }
}
```

3.4. Agregação e Dependência

A **agregação** é semelhante à composição, mas representa uma posse mais fraca: a "parte" pode existir independentemente do "todo" (por exemplo, um Departamento agrega Funcionarios, mas os funcionários continuam existindo mesmo que o departamento seja extinto). É representada por um losango vazado no lado do todo.

A **dependência** indica que uma classe utiliza outra apenas temporariamente, geralmente como parâmetro de um método, sem manter uma referência permanente. É representada por uma linha tracejada com seta simples.

```
public class RelatorioService {  
    // Dependência: Impressora é usada só como parâmetro do método  
    public void gerarRelatorio(Impressora impressora) {  
        impressora.imprimir("Relatório gerado.");  
    }  
}
```

4. Exemplo de Diagrama Completo

A seguir, um diagrama reunindo herança e associação em um único cenário: **Funcionario** e **Gerente** (herança), e **Gerente** associado a vários **Funcionario** que gerencia.

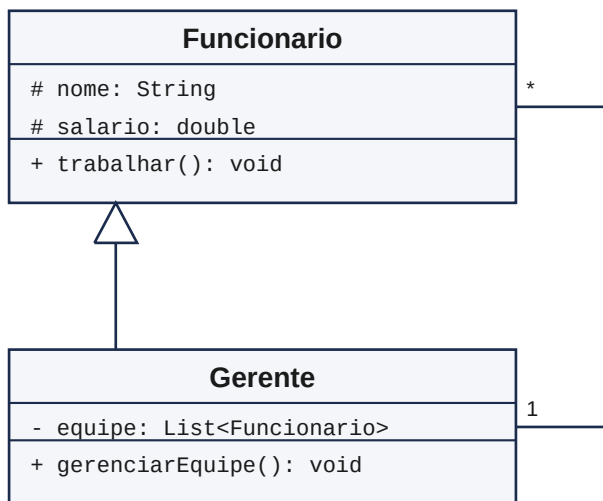


Figura 5 — *Gerente herda de Funcionario e, ao mesmo tempo, se associa a vários Funcionario (a equipe que gerencia).*

```
import java.util.List;
import java.util.ArrayList;

public class Funcionario {
    protected String nome;
    protected double salario;

    public void trabalhar() {
        System.out.println(nome + " está trabalhando.");
    }
}

public class Gerente extends Funcionario {
    private List<Funcionario> equipe = new ArrayList<>();

    public void adicionarNaEquipe(Funcionario f) {
        equipe.add(f);
    }

    public void gerenciarEquipe() {
        for (Funcionario f : equipe) {
            System.out.println("Gerenciando: " + f.nome);
        }
    }
}
```

5. Conclusão

O Diagrama de Classes UML é uma ferramenta essencial no projeto de sistemas orientados a objetos, pois permite visualizar a estrutura de classes, seus atributos e operações, além dos relacionamentos de herança, associação, agregação, composição e dependência antes mesmo de escrever qualquer código. Como visto nos exemplos, cada elemento do diagrama tem uma correspondência direta com construções da linguagem Java, o que torna o diagrama um guia prático para a implementação do sistema.