

Composição x Agregação

Ao modelar sistemas orientados a objetos, é comum encontrar relações do tipo "todo-parte", em que uma classe é composta por outras. O UML representa essas relações por meio de dois conceitos frequentemente confundidos: **agregação** e **composição**. Embora ambos descrevam associações de posse, a diferença entre eles está no grau de dependência entre os objetos envolvidos — e essa diferença tem impacto direto na forma como o código é estruturado.

Agregação

A agregação representa uma relação "todo-parte" fraca. Nela, o objeto que compõe o "todo" pode existir de forma independente, antes, depois ou até mesmo sem ele. Em UML, essa relação é indicada por um losango vazio no lado do "todo".



Diagrama UML de agregação entre Turma e Aluno.

Um bom exemplo é a relação entre Turma e Aluno: uma turma agrega alunos, mas se a turma deixar de existir, os alunos continuam existindo normalmente — podem, inclusive, ser matriculados em outra turma. No código, isso se reflete no fato de o objeto Aluno ser criado externamente e apenas referenciado pela Turma, geralmente por meio de um método como `adicionarAluno(Aluno aluno)`.

```

class Aluno {
    private String nome;

    public Aluno(String nome) {
        this.nome = nome;
    }

    public String getNome() {
        return nome;
    }
}

class Turma {
    private String codigo;
    private List<Aluno> alunos;

    public Turma(String codigo) {
        this.codigo = codigo;
        this.alunos = new ArrayList<>();
    }

    // Aluno ja existe e e apenas adicionado a turma
    public void adicionarAluno(Aluno aluno) {
        alunos.add(aluno);
    }

    public void removerAluno(Aluno aluno) {
        alunos.remove(aluno); // remove da turma, mas o aluno continua existindo
    }
}

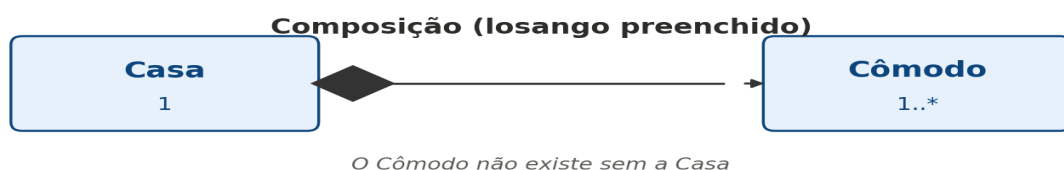
// Uso:
public class Main {
    public static void main(String[] args) {
        Aluno joao = new Aluno("Joao"); // criado independentemente
        Turma turmaA = new Turma("Turma A");

        turmaA.adicionarAluno(joao);
        // Se turmaA for destruida (ex: turmaA = null), joao continua existindo
        // e pode ser adicionado a outra turma.
    }
}

```

Composição

Já a composição representa uma relação "todo-parte" forte, na qual a parte não tem existência própria fora do todo. O ciclo de vida do objeto "parte" está diretamente atrelado ao ciclo de vida do "todo". Em UML, essa relação é representada por um losango preenchido.



O exemplo clássico é a relação entre Casa e Comodo: os cômodos só fazem sentido enquanto parte de uma casa específica. Se a casa for destruída, os cômodos também deixam de existir. No código, isso aparece quando o próprio objeto Casa cria seus Comodo internamente, dentro do construtor, sem expor nenhuma forma de reaproveitá-los em outra instância.

```
class Comodo {
    private String nome;
    private double areaM2;

    public Comodo(String nome, double areaM2) {
        this.nome = nome;
        this.areaM2 = areaM2;
    }
}

class Casa {
    private String endereco;
    private List<Comodo> comodos;

    public Casa(String endereco) {
        this.endereco = endereco;
        this.comodos = new ArrayList<>();
        // Os comodos sao criados AQUI, dentro da propria casa
        comodos.add(new Comodo("Sala", 20.0));
        comodos.add(new Comodo("Cozinha", 12.0));
        comodos.add(new Comodo("Quarto", 15.0));
    }

    // Nao existe metodo para "transferir" um comodo para outra casa
}

// Uso:
public class Main {
    public static void main(String[] args) {
        Casa minhaCasa = new Casa("Rua das Flores, 123");
        // Os comodos foram criados junto com a casa.
        // Se minhaCasa = null, o Garbage Collector tambem elimina os Comodo,
        // pois nada mais os referencia.
    }
}
```

A diferença está na responsabilidade, não na sintaxe

Do ponto de vista estrutural, tanto a agregação quanto a composição costumam ser implementadas da mesma forma em Java — geralmente com uma coleção, como uma List, guardando referências para os objetos relacionados. A verdadeira distinção não está na sintaxe, mas em **quem é responsável por criar e gerenciar o ciclo de vida do objeto**.

Uma regra prática para identificar qual relação se aplica é perguntar: *"se o todo for destruído, a parte também deixa de existir?"* Se a resposta for sim, trata-se de composição. Se a parte puder sobreviver de forma independente, trata-se de agregação.

Compreender essa distinção é importante não apenas para desenhar diagramas de classes corretos, mas também para tomar decisões de design mais conscientes sobre acoplamento, reutilização e gerenciamento de memória no código.