

Composição, Agregação e Variáveis de Instância Final

Relações entre objetos e imutabilidade de atributos em Java

1. Introdução

Ao modelar um sistema orientado a objetos, é comum que um objeto seja formado por outros objetos, e não apenas por atributos simples como números e textos. Essas relações "todo-parte" são chamadas de **composição** e **agregação**, e diferem quanto ao grau de dependência entre as partes. Além disso, muitos desses relacionamentos envolvem atributos que, uma vez definidos, não devem mais mudar — daí a importância das **variáveis de instância final**. Este texto explora os três conceitos com exemplos práticos em Java.

2. Composição

A **composição** é uma relação de posse forte entre um objeto "todo" e seus objetos "parte": as partes **não existem independentemente** do todo. Se o objeto principal for destruído, os objetos que o compõem deixam de existir junto com ele. No diagrama UML, a composição é representada por um losango **preenchido** do lado do todo.

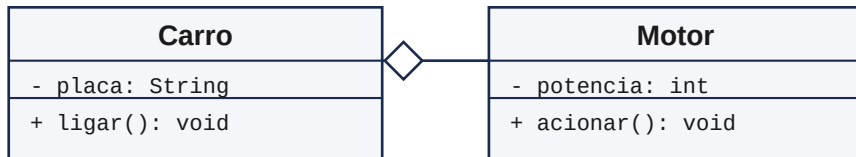


Figura 1 — Composição: o Motor é criado e destruído junto com o Carro (losango preenchido).

Em Java, a composição costuma ser implementada criando o objeto "parte" **dentro do construtor** do objeto "todo", frequentemente combinado com o modificador `final` no atributo, já que a referência ao objeto parte não deve mudar durante a vida do objeto todo.

```

public class Motor {
    private int potencia;

    public Motor(int potencia) {
        this.potencia = potencia;
    }

    public void acionar() {
        System.out.println("Motor de " + potencia + " cv acionado.");
    }
}

public class Carro {
    private String placa;
    private final Motor motor; // composição: pertence exclusivamente ao carro

    public Carro(String placa, int potenciaMotor) {
        this.placa = placa;
        this.motor = new Motor(potenciaMotor); // criado junto com o Carro
    }

    public void ligar() {
        motor.acionar();
        System.out.println("Carro " + placa + " ligado.");
    }
}
// Quando um objeto Carro é destruído, seu Motor também deixa de existir –
// não há como reaproveitá-lo em outro Carro.

```

3. Agregação

A **agregação** também representa uma relação "todo-parte", mas com um vínculo mais fraco: os objetos "parte" **podem existir independentemente** do todo e, inclusive, ser compartilhados entre diferentes objetos "todo". No diagrama UML, é representada por um losango **vazado** do lado do todo.

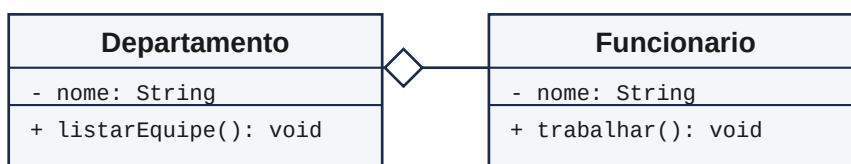


Figura 2 — Agregação: um Funcionario pode existir e ser reaproveitado mesmo fora de um Departamento específico (losango vazado).

Em Java, a agregação é normalmente implementada **recebendo** o objeto "parte" já pronto (por exemplo, via construtor ou método), em vez de criá-lo internamente — o objeto "parte" foi criado de forma independente e apenas passa a ser referenciado pelo "todo".

```

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

public class Funcionario {
    private String nome;

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

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

    public String getNome() { return nome; }
}

public class Departamento {
    private String nome;
    private List<Funcionario> equipe = new ArrayList<>(); // agregação

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

    public void adicionarFuncionario(Funcionario f) { // recebe objeto existente
        equipe.add(f);
    }

    public void listarEquipe() {
        for (Funcionario f : equipe) {
            System.out.println(f.getNome());
        }
    }
}

// Uso:
Funcionario ana = new Funcionario("Ana"); // Ana existe independentemente
Departamento ti = new Departamento("TI");
Departamento rh = new Departamento("RH");

ti.adicionarFuncionario(ana); // Ana pode ser referenciada em vários lugares
// Se o objeto "ti" for destruído, "ana" continua existindo normalmente.

```

4. Composição vs Agregação

Aspecto	Composição	Agregação
Força do vínculo	Forte (posse exclusiva)	Fraco (posse compartilhável)
Ciclo de vida da parte	Depende do todo; morre junto com ele	Independente do todo
Criação típica em Java	new dentro do construtor do todo	Objeto recebido como parâmetro
Símbolo UML	Losango preenchido	Losango vazado
Exemplo típico	Carro e Motor	Departamento e Funcionario

5. Variáveis de Instância final

Um atributo declarado com o modificador `final` só pode receber seu valor **uma única vez**: na própria declaração ou dentro de um construtor. Depois disso, qualquer tentativa de alterá-lo gera um erro de compilação. Esse recurso é frequentemente usado justamente em relações de **composição**, como visto no exemplo do Carro e do Motor, pois reforça que a referência ao objeto parte não deve ser trocada depois que o objeto todo é criado.

```
public class Usuario {
    private final String cpf; // não pode ser alterado após a criação

    public Usuario(String cpf) {
        this.cpf = cpf; // única atribuição permitida
    }

    // public void setCpf(String cpf) { this.cpf = cpf; } // ERRO: cpf é final

    public String getCpf() {
        return cpf;
    }
}
```

É importante destacar que `final` garante apenas que a **referência** não muda — no caso de um atributo do tipo objeto, o conteúdo interno desse objeto ainda pode ser alterado, a menos que o próprio objeto também seja imutável.

```
public class Pedido {
    private final List<String> itens; // a REFERÊNCIA da lista é final

    public Pedido() {
        this.itens = new ArrayList<>();
    }

    public void adicionarItem(String item) {
        itens.add(item); // permitido: o conteúdo da lista pode mudar
    }

    // itens = new ArrayList<>(); // ERRO: variável final não pode ser reatribuída
}
```

5.1. final em Composição: Garantindo a Integridade da Relação

Ao combinar `final` com composição, garante-se que um objeto "todo" sempre manterá a referência ao **mesmo** objeto "parte" durante toda a sua existência, reforçando a ideia de que a parte pertence exclusivamente àquele objeto.

```
public class Endereco {
    private final String rua;
    private final String cidade;

    public Endereco(String rua, String cidade) {
        this.rua = rua;
        this.cidade = cidade;
    }

    public String getRua() { return rua; }
    public String getCidade() { return cidade; }
}

public class Pessoa {
    private String nome;
    private final Endereco endereco; // composição + final

    public Pessoa(String nome, Endereco endereco) {
        this.nome = nome;
        this.endereco = endereco; // definido uma única vez, no construtor
    }

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

6. Conclusão

Composição e agregação descrevem, de formas diferentes, como um objeto pode ser formado por outros objetos: na composição, as partes dependem totalmente do todo e são criadas junto com ele; na agregação, as partes têm vida própria e podem ser compartilhadas. As variáveis de instância final complementam esses conceitos ao garantir que a referência a um objeto "parte" — especialmente em relações de composição — permaneça estável durante todo o ciclo de vida do objeto "todo", tornando o design da classe mais seguro e previsível.