INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Montes Claros

Design Patterns: explicação e abordagens práticas sobre Builder e Decorator

Autor: Iago Rocha Oliveira

Resumo

Este trabalho apresenta dois padrões de projeto fundamentais para o desenvolvimento de software orientado a objetos: **Builder** e **Decorator**. Ambos pertencem ao catálogo de design patterns proposto pela Gang of Four e são amplamente utilizados em projetos Java, principalmente pela sua capacidade de promover código mais modular, flexível e de fácil manutenção. Neste trabalho, são apresentados o contexto e a motivação dos padrões de projeto, as categorias existentes, os diagramas de classes UML dos padrões Builder e Decorator, além de exemplos de implementação em Java com explicações detalhadas. O objetivo é demonstrar, na prática, como esses padrões contribuem para a escrita de código mais limpo, escalável e aderente aos princípios da orientação a objetos.

Introdução

No desenvolvimento de software orientado a objetos, é comum que determinados problemas de projeto aparecem repetidamente em diferentes contextos. Com o tempo, engenheiros e arquitetos de software identificaram que certas soluções recorrentes podiam ser documentadas e reutilizadas. Essas soluções ficaram conhecidas como padrões de projeto (*design patterns*), que representam boas práticas para resolver problemas típicos relacionados à arquitetura, construção e comportamento de objetos e classes.

A formalização dos padrões de projeto ganhou relevância com a publicação do livro *Design Patterns: Elements of Reusable Object-Oriented Software*, de Gamma, Helm, Johnson e Vlissides (1994), conhecidos como a Gang of Four (GoF). Segundo os autores, "um padrão de projeto nomeia, abstrai e identifica os aspectos-chave de uma estrutura recorrente de projeto orientado a objetos" (Gamma et al., 1994, p. 2). A obra catalogou 23 padrões clássicos, agrupados em três categorias principais: padrões criacionais, que lidam com a criação de objetos; padrões estruturais, que definem maneiras de compor classes e objetos; e padrões comportamentais, que se concentram na comunicação entre objetos.

A principal vantagem de aplicar padrões de projeto é a promoção de um código mais modular, reutilizável e de fácil manutenção. Além disso, os padrões facilitam a comunicação entre desenvolvedores, pois fornecem uma linguagem comum para descrever soluções conhecidas. Também incentivam o uso de princípios sólidos de engenharia de software, como o princípio aberto/fechado, responsabilidade única e inversão de dependência.

Neste trabalho, serão apresentados dois padrões fundamentais: Builder e Decorator, com foco em sua estrutura, propósito e aplicação prática em projetos desenvolvidos em Java.

Padrão de Projeto Builder

O padrão de projeto Builder tem como principal objetivo desacoplar a construção de um objeto complexo da sua representação final, permitindo que o mesmo processo de construção possa gerar diferentes representações do objeto. Em vez de instanciar objetos diretamente com longos construtores ou várias sobrecargas, o Builder organiza a criação de objetos de forma controlada, passo a passo, favorecendo a clareza e a manutenção do código.

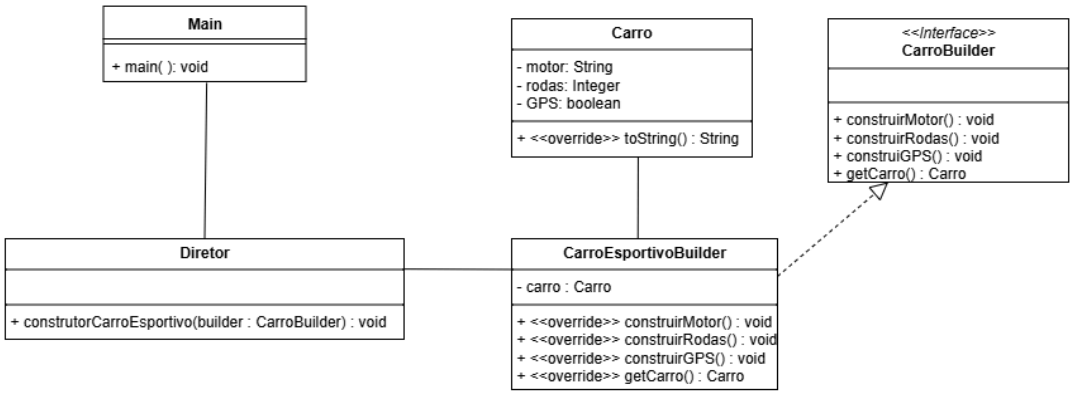
Em aplicações orientadas a objetos, é comum encontrar classes que representam entidades complexas, com diversos atributos opcionais, parâmetros condicionais ou configurações variadas. Quando a construção dessas entidades é feita diretamente com construtores extensos ou por meio de múltiplas sobrecargas de métodos, o código se torna difícil de entender, manter e estender. Além disso, isso viola princípios importantes de design, como o princípio da responsabilidade única, pois a mesma classe acaba acumulando tanto a lógica do objeto quanto sua construção.

O padrão Builder resolve esse problema ao separar a lógica de construção do objeto da lógica de uso, delegando a montagem do objeto a uma estrutura especializada (o *Builder*), que fornece métodos específicos para configurar cada parte da instância desejada. Dessa forma, o código cliente não precisa se preocupar com os detalhes internos de montagem e pode obter versões distintas do objeto com o mesmo processo de construção.

Para ilustrar a aplicação do padrão Builder, foi escolhido como exemplo o processo de construção de um carro. Um carro pode ser composto por diversos elementos configuráveis, como tipo de motor, quantidade de portas, presença de GPS, ar-condicionado, sistema de som, entre outros. Dependendo da necessidade, é possível construir carros de diferentes categorias (como carro popular, esportivo ou SUV), utilizando o mesmo processo de montagem, mas com componentes e configurações distintas.

Neste contexto, o Builder permite montar diferentes versões de um carro de forma organizada e reutilizável, sem depender de múltiplos construtores ou subclasses. Na Figura 1, é apresentado o diagrama de classes UML representando a estrutura do padrão aplicada à construção de um carro.

Figura 1 - Diagrama UML do projeto de construção de um carro com o padrão Builder



Fonte: Acervo do autor

Padrão de Projeto Decorator

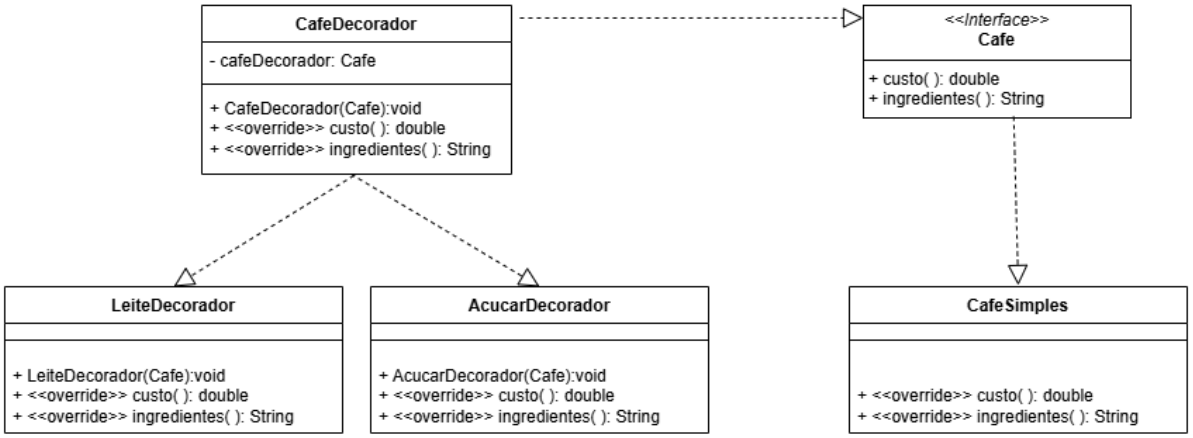
O padrão Decorator tem como objetivo adicionar dinamicamente responsabilidades a um objeto sem a necessidade de modificar sua estrutura original. Ele permite estender funcionalidades de forma flexível e modular, promovendo o princípio de aberto/fechado (Open/Closed Principle), em que as classes devem estar abertas para extensão, mas fechadas para modificação.

Em muitas situações, precisamos adicionar comportamentos ou funcionalidades a objetos de forma combinável e independente. Uma abordagem ingênua para isso seria criar diversas subclasses para cada combinação possível de funcionalidades — o que rapidamente se torna inviável com o aumento do número de variações.

O Decorator resolve esse problema ao permitir que objetos sejam "embrulhados" em outros objetos (os decoradores), que adicionam funcionalidades extras sem alterar a classe original. Assim, comportamentos podem ser compostos em tempo de execução.

Vamos imaginar uma aplicação em que o usuário pode montar um café personalizado. A base é sempre um café simples, mas ele pode adicionar ingredientes como leite, chantilly ou chocolate. Em vez de criar uma classe para cada combinação, podemos utilizar o padrão Decorator para compor dinamicamente os ingredientes.

Figura 2 - Diagrama UML do projeto de construção de um carro com o padrão Decorator



Fonte: Acervo do autor

Conclusão

O estudo e a aplicação dos padrões de projeto são fundamentais para o desenvolvimento de software mais organizado, reutilizável e de fácil manutenção. Neste trabalho, exploramos dois padrões bastante utilizados: o Builder e o Decorator. Cada um atende a diferentes necessidades do projeto: o Builder é eficaz na construção de objetos complexos passo a passo, enquanto o Decorator permite adicionar funcionalidades de forma flexível e dinâmica, sem alterar a estrutura original do objeto. Além de compreender o funcionamento teórico desses padrões, também implementamos exemplos

práticos contextualizados, como a montagem de um carro e a personalização de um café. Essas abordagens demonstraram, na prática, como os padrões auxiliam a resolver problemas reais de forma elegante e extensível.

A compreensão e a prática com esses padrões contribuem diretamente para a formação de um programador mais maduro, que consegue tomar decisões mais adequadas quanto ao design e à arquitetura do software. Assim, o uso consciente dos padrões de projeto representa um passo importante rumo à construção de sistemas mais robustos e sustentáveis a longo prazo.

Referências

GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. ***Design patterns: elements of reusable object-oriented software***. Boston: Addison-Wesley, 1994.

REFATORING.GURU. *Builder – Padrão de projeto (pt-BR)*. Disponível em: https://refactoring.guru/pt-br/design-patterns/builder. Acesso em: 18 jul. 2025.