Trabalho Prático II – POO
Aplicação do Padrão de Projeto Builder em um Sistema de Barbearia

Luann Moreira Fernandes de Oliveira , Matheus de Moura Almeida Neri

[1] Universidade Federal dos Vales do Jequitinhonha e Mucuri (UFVJM)
Curso de Sistemas de Informação
Diamantina – MG

Abstract. *This paper presents the practical application of the Builder design pattern in a barbershop management system developed for an Object-Oriented Programming course. The pattern was used to improve the creation of complex objects, especially the scheduling module, resulting in clearer, safer and easier-to-maintain code.*

Resumo. *Este trabalho apresenta a aplicação do padrão de projeto Builder em um sistema de gestão para uma barbearia, desenvolvido na disciplina de Programação Orientada a Objetos. O padrão foi utilizado na refatoração da classe* `Agendamento`*, com o objetivo de tornar a criação de objetos complexos mais legível, segura e alinhada a boas práticas de engenharia de software. São discutidos o contexto, a implementação, os resultados obtidos e uma análise crítica sobre a adequação do padrão ao projeto.*

Palavras-chave: Padrões de Projeto; Builder; Programação Orientada a Objetos; Engenharia de Software; Barbearia.

Sumário

1. Introdução

O desenvolvimento de sistemas orientados a objetos exige estruturas que favoreçam coerência, reuso e facilidade de manutenção. No contexto deste trabalho, foi implementado um sistema de gestão para uma barbearia, contemplando cadastro de clientes, funcionários, serviços, produtos, agendamentos e ordens de serviço.

A classe `Agendamento` surgiu como um ponto sensível do projeto, pois reúne diversos atributos que precisam ser consistentes: identificação, cliente, serviço, funcionário responsável, data, horário, valor e status. A criação direta desses objetos por meio de construtores longos tornava o código menos legível e mais propenso a erros.

Diante disso, optou-se pela utilização do padrão de projeto *Builder*, com o objetivo de organizar o processo de criação de agendamentos de forma clara e segura.

2. Objetivo

Este trabalho tem como objetivo apresentar a aplicação do padrão de projeto *Builder* na classe `Agendamento` do sistema da barbearia, descrevendo sua motivação, sua estrutura, sua implementação em Java e os impactos na qualidade do código.

3. Material e Métodos

O estudo foi realizado sobre o código-fonte do sistema da barbearia implementado em Java, seguindo princípios de orientação a objetos, encapsulamento e persistência em arquivos. Os passos principais foram:

- modelar as entidades do domínio da barbearia;
- identificar a complexidade na criação de agendamentos;
- projetar um *Builder* específico para a classe `Agendamento`;
- integrar o *Builder* ao fluxo de criação de agendamentos no controlador `Sistema`;
- representar a solução em diagramas de classe utilizando o pacote `tikz-uml`.

4. Resultados e Discussão

Os **padrões de projeto** são soluções consolidadas para problemas recorrentes no desenvolvimento orientado a objetos. Entre eles, o **Builder** destaca-se por facilitar a criação de objetos complexos, promovendo clareza, consistência e flexibilidade na construção de instâncias. No contexto do sistema da barbearia, esse padrão foi escolhido para resolver problemas estruturais na classe `Agendamento`, responsável por reunir diversas informações como cliente, funcionário, serviço, data e valor.

Cenário antes do Builder

No início do projeto, a criação de um agendamento era realizada diretamente por meio de um construtor extenso, com múltiplos parâmetros em sequência:

Listing 1. Criação tradicional de objeto antes do uso do Builder

```
Agendamento ag = new Agendamento("AG01",
    cliente, funcionario, servico,
    LocalDateTime.of(2025, 11, 15, 14, 0), 40.0);
```

Essa abordagem dificultava a compreensão do código e tornava o processo suscetível a erros de ordem, além de comprometer a manutenção futura. Cada nova modificação exigia ajustes em diferentes partes do sistema, o que aumentava a complexidade e o tempo de desenvolvimento.

Cenário após o Builder

Com a refatoração e a aplicação do *Builder*, a criação de objetos passou a ocorrer de forma fluente, clara e modular. A lógica de construção foi encapsulada dentro de uma classe interna, eliminando construtores longos e centralizando as validações:

Listing 2. Criação de objeto utilizando o padrão Builder

```
Agendamento ag = new Agendamento.Builder()
    .id("AG01")
    .cliente(cliente)
    .funcionario(funcionario)
    .servico(servico)
    .dataHora(LocalDateTime.of(2025, 11, 15, 14, 0))
    .valor(40.0)
    .build();
```

O impacto dessa mudança foi imediato: o código se tornou mais intuitivo, o processo de criação mais seguro e a manutenção mais previsível. Essa evolução também facilitou a integração do padrão dentro do módulo `Sistema.java`, responsável por orquestrar o fluxo de agendamentos:

Listing 3. Método de agendamento implementado no Sistema.java

```
public void agendarAtendimento(String id,
                               Cliente cliente,
                               Funcionario funcionario,
                               Servico servico,
                               LocalDateTime dataHora,
                               double valor) {

    Agendamento ag = new Agendamento.Builder()
        .id(id)
        .cliente(cliente)
        .funcionario(funcionario)
        .servico(servico)
        .dataHora(dataHora)
        .valor(valor)
        .build();

    agendamentos.add(ag);
}
```

Essa integração garantiu que os objetos fossem criados de forma consistente, validada e com menor acoplamento entre as camadas do sistema.

5. Vantagens e Desvantagens

Vantagens Técnicas

A aplicação do *Builder* trouxe ganhos substanciais na estrutura e na qualidade do código. Entre os principais aspectos técnicos, destacam-se:

- **Modularidade:** separa o processo de construção do uso, tornando o sistema mais organizado;
- **Extensibilidade:** novos campos e validações podem ser adicionados sem reescrever construtores;
- **Consistência:** centraliza a lógica de criação, evitando duplicação de código;
- **Legibilidade:** facilita a compreensão por parte de novos desenvolvedores ou avaliadores acadêmicos;
- **Integração:** o padrão se adapta bem a sistemas em camadas, como MVC.

Limitações e Cuidados

Apesar dos benefícios, o *Builder* exige ponderação em seu uso. Alguns pontos de atenção foram observados:

- Em classes simples, o padrão pode gerar sobrecarga desnecessária;
- A estrutura adicional de código pode confundir desenvolvedores iniciantes;
- Exige disciplina para manter o método `build()` sempre atualizado e seguro;
- O uso indevido pode transformar o padrão em um anti-padrão, se não houver ganho real de complexidade.

6. Análise Crítica

A implementação do *Builder* na classe `Agendamento` foi uma oportunidade prática de aplicar conceitos de engenharia de software estudados na disciplina. O padrão demonstrou ser uma solução coerente para o problema identificado e proporcionou uma refatoração limpa, estruturada e de fácil entendimento.

Do ponto de vista acadêmico, o processo foi relevante por mostrar a diferença entre programar apenas para "funcionar" e projetar pensando em **manutenibilidade e qualidade de código**. Além disso, a aplicação do padrão favoreceu a evolução contínua do projeto acadêmico, permitindo que novos atributos e validações fossem incorporados sem comprometer o funcionamento das demais partes do sistema. Essa refatoração também se mostrou **pedagogicamente relevante**, pois facilitou a compreensão do código pelos integrantes do grupo e reduziu o tempo necessário para testes e correções durante o desenvolvimento.

Portanto, conclui-se que o padrão *Builder* foi aplicado de forma correta, trazendo ganhos concretos em legibilidade, coesão e robustez. A decisão de aplicá-lo pode ser considerada positiva e estratégica, representando um exemplo real de como padrões de projeto podem melhorar a arquitetura de um sistema.

7. Diagramas de Classe

7.1. Diagrama de Classe do Padrão de Projeto Builder

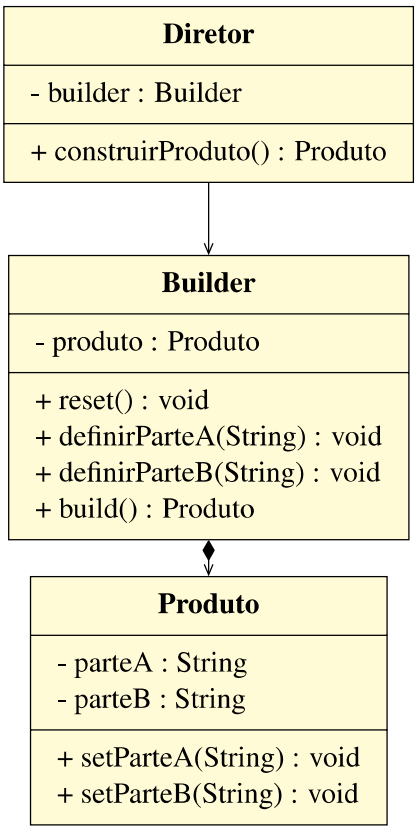

Figura 1 – Estrutura genérica do padrão de projeto Builder.

7.2. Diagrama de Classe do Exemplo Proposto: Agendamento com Builder

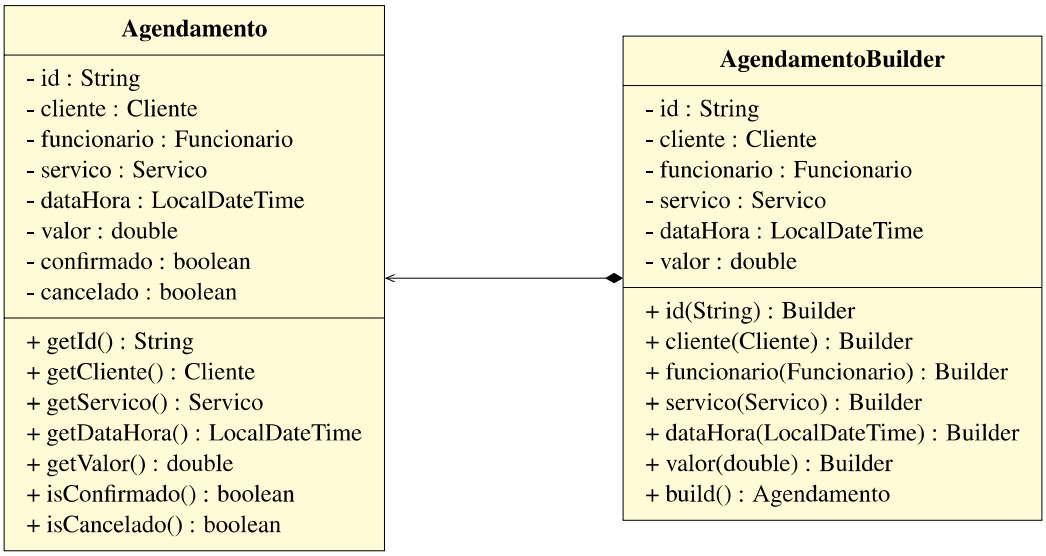

Figura 2 – Aplicação do padrão Builder na classe `Agendamento` do sistema da barbearia.

8. Conclusão

A aplicação do padrão de projeto *Builder* no sistema da barbearia mostrou-se adequada e benéfica. A classe `Agendamento`, por concentrar diversas informações relevantes e regras de negócio, é um candidato natural ao uso desse padrão.

O *Builder* contribuiu para tornar o código mais claro, seguro e flexível, permitindo que novos atributos e validações sejam incorporados sem impacto negativo nas demais partes do sistema. No contexto analisado, a utilização do padrão não se caracteriza como anti-padrão, mas sim como uma escolha consciente alinhada às boas práticas de projeto.

Referências Bibliográficas

[1] KIELBASIEWICZ, Nicolas. *The TikZ-UML package*. 16 out. 2012. Disponível em: `http://nicola-kielbasiewicz.de/tikz-uml/`. Acesso em: 2 set. 2025.

[2] MIRO. *O que é UML e como funciona*. Disponível em: `https://miro.com/pt/diagrama/o-que-e-uml`. Acesso em: 2 set. 2025.