Padrões de projeto - Builder e Indirection

Charlies Augusto Sichelero e Raíssa Girardi Gaiardo
IFRS -Campus Farroupilha
Professor: Rogério Xavier

Definição do Builder

O Builder é um padrão de design criacional que separa a construção de um objeto complexo da sua representação final, permitindo que o mesmo processo de construção crie diferentes representações do objeto.

Ao invés de usar vários construtores ou métodos "setters" que podem ser difíceis de gerenciar, o Builder facilita a configuração e construção do objeto final, mantendo o código mais organizado e legível.

Características do Builder:

- Divide a criação do objeto em duas partes: o "construtor" (Builder) e o produto final. Isso significa que o Builder foca nos detalhes de construção sem interferir na estrutura ou funcionalidade do objeto.
- Interface Encadeada: O Builder geralmente usa uma abordagem encadeada, onde métodos de configuração retornam o próprio builder, permitindo chamadas sequenciais como builder.setParam1().setParam2().setParam3().
- Flexibilidade e Extensibilidade: Permite a criação de diferentes representações de um mesmo tipo de objeto. Pode ser utilizado para criar diferentes configurações de um produto ou variantes de um objeto.

Quais problemas o Builder soluciona?

- Construtores com Muitos Parâmetros: Em objetos que possuem muitos parâmetros, o uso de um construtor comum se torna confuso e propenso a erros, pode ficar extenso e difícil de manter.
- Falta de Legibilidade com "Setters": O uso de muitos "setters" (métodos para definir cada atributo individualmente) pode levar a um código extenso e difícil de ler, e não garante a consistência do objeto. Com o Builder, os atributos podem ser configurados de forma mais clara e controlada.
- Necessidade de Objetos Imutáveis: Em situações onde o objeto não deve mudar depois de ser criado, o Builder ajuda a garantir que, após construído, o objeto não seja modificado, promovendo maior segurança e previsibilidade.
- Variações de Configurações: Quando o objeto precisa ser criado com diferentes configurações, o Builder facilita a criação dessas variações sem a necessidade de múltiplos construtores sobrecarregados.

Exemplo

```
class Computador {
    1 usage
    private String processador;
    1 usage
    private int memoriaRAM;
    1 usage
    private int armazenamento;

    // Builder interno
    5 usages
    public static class Builder {
        2 usages
        private String processador;
        2 usages
        private int memoriaRAM;
        2 usages
        private int armazenamento;
```

```
1 usage
public Builder setProcessador(String processador) {
    this.processador = processador;
    return this;
}

1 usage
public Builder setMemoriaRAM(int memoriaRAM) {
    this.memoriaRAM = memoriaRAM;
    return this;
}

1 usage
public Builder setArmazenamento(int armazenamento) {
    this.armazenamento = armazenamento;
    return this;
}
```

Exemplo

```
    1 usage
    public Computador build() {
        return new Computador( builder: this);
    }
}

1 usage
private Computador(Builder builder) {
    this.processador = builder.processador;
    this.memoriaRAM = builder.memoriaRAM;
    this.armazenamento = builder.armazenamento;
}

// Exemplo de uso
no usages
Computador computador = new Computador.Builder()
        .setProcessador("Intel i7")
        .setMemoriaRAM(16)
        .setArmazenamento(512)
        .build();
```

Definição do Indirection

O padrão Indirection é um princípio presente no conjunto de padrões GRASP - General Responsibility Assignment Software Patterns, em tradução livre, Padrões de Software Gerais para Atribuição de Responsabilidade - que visa o baixo acoplamento entre dois ou mais elementos.

Características do Indirection:

Ele se caracteriza por atribuir responsabilidade a um objeto intermediário a mediação entre os componentes ou serviços, assim evita o acoplamento direto. O resultado disso é a criação de uma indireção entre os objetos envolvidos, isto é, a referência entre os objetos é indireta.

Esse princípio pode ser visto nos seguintes padrões GoF: Adapter, Facade e Observer. Outro exemplo de aplicação do Indirection se vê no padrão Inversion of Control (Inversão de Controle) em conjunto com a técnica de programação Dependency Injection (Injeção de Dependência).

Quais problemas o Indirection soluciona?

- Baixo acoplamento: Usando o princípio Indirection, podemos abstrair a dependência entre classes.
- Facilita substituição de componentes: Uma consequência do baixo acoplamento, permite a troca de implementações conforme a necessidade com pouco esforço de programação.
- Aumenta a capacidade de testes e reuso de código: Outra consequência do baixo acoplamento, uma vez que as classes têm dependências abstratas, isso possibilita a confecção de testes de unidade mais granulares.

Exemplo de código

```
[Route("api/[controller]")]
[ApiController]
[Authorize]
1 referência
public class CategoriesController : ControllerBase
{
    private readonly ICategoryService _categoryService;
    0 referências
    public CategoriesController(ICategoryService categoryService)
    {
        _categoryService = categoryService;
    }

    [HttpGet]
    0 referências
    public async Task<ActionResult<IEnumerable<CategoryDTO>>> Get()
    {
        var categories = await _categoryService.GetCategories();
        if (categories == null)
        {
            return NotFound("Categories not found");
        }
        return Ok(categories);
    }
```

Provisionamento do service...

```
public static IServiceCollection AddInfrastructure(this IServiceCollection services,
    IConfiguration configuration)
{
    services.AddDbContext<ApplicationDbContext>(options =>
    options.UseSqlServer(configuration.GetConnectionString("DefaultConnection"
    ), b => b.MigrationsAssembly(typeof(ApplicationDbContext).Assembly.FullName)));

    services.AddIdentity<ApplicationUser, IdentityRole>()
        .AddEntityFrameworkStores<ApplicationDbContext>()
        .AddDefaultTokenProviders();

    services.ConfigureApplicationCookie(options =>
            options.AccessDeniedPath = "/Account/Login");

    services.AddScoped<ICategoryRepository, CategoryRepository>();
    services.AddScoped<IProductRepository, ProductRepository>();
    services.AddScoped<IProductService, ProductService>();
    services.AddScoped<ICategoryService, CategoryService>();
```

Implementação do Service

```
public class CategoryService : ICategoryService
{
    private ICategoryRepository _categoryRepository;
    private readonly IMapper _mapper;
    0 referências
    public CategoryService(ICategoryRepository categoryRepository, IMapper mapper)
    {
        _categoryRepository = categoryRepository;
        _mapper = mapper;
    }

    6 referências
    public async Task<IEnumerable<CategoryDTO>> GetCategories()
    {
        var categoriesEntity = await _categoryRepository.GetCategories();
        return _mapper.Map<IEnumerable<CategoryDTO>>(categoriesEntity);
    }

    6 referências
    public async Task<CategoryDTO> GetById(int? id)
    {
        var categoryEntity = await _categoryRepository.GetById(id);
        return _mapper.Map<CategoryDTO>(categoryEntity);
    }
```

Referências:

EDSON. **Design Patterns: aplicando os padrões builder, Singleton e prototype**. 2014. Disponível em: https://www.devmedia.com.br/design-patterns-aplicando-os-padroes-builder-singleton-e-prototype/31023. Acesso em: 03 nov. 2024.

SHVETS, Alexander. **Builder**. 2024. Disponível em: https://refactoring.guru/pt-br/design-patterns/builder#:~:text=O%20Builder%20n%C3%A3o%20permite%20que,etapas%20em%20um%20objeto%20builder. Acesso em: 03 nov. 2024.

WESLEY WILLIANS. Full Cycle. **Design Pattern Builder na Prática**. 2020. Disponível em: https://www.youtube.com/watch?v=W-96z2EjoJ0. Acesso em: 03 nov. 2024.

Referências:

AZAMBUJA, Rogério Xavier de. **Resumo de aula sobre Padrões de Projeto de Software (Design Patterns)**. 2024. Disponível em: https://web.farroupilha.ifrs.edu.br/moodle/mod/resource/view.php?id=150638. Acesso em: 03 nov. 2024.

[S.I.] ([S.I.]). Linkedin (ed.). **Indirection Pattern: A GRASP Principle for OOAD**. 2024. Artigo alimentado por IA e pela comunidade do LinkedIn. Disponível em: https://pt.linkedin.com/advice/1/how-do-you-use-indirection-pattern-avoid-direct-coupling-between?lang=en. Acesso em: 03 nov. 2024.