Discentes: José Victor e João Gabriel

Relatório de Justificativa dos Padrões de Criação – Sistema de Gerenciamento de Pedidos para Restaurantes

1. Introdução

Este relatório apresenta uma análise criteriosa da aplicação dos padrões de projeto do tipo **criação** no contexto de um sistema de gerenciamento de pedidos para uma rede de restaurantes. O objetivo é garantir que as decisões arquiteturais adotadas favoreçam a **manutenibilidade**, **extensibilidade**, **reutilização de código** e **baixo acoplamento**. Para isso, foram selecionados e implementados dois cenários com os padrões mais adequados, além da justificativa teórica para os cinco requisitos propostos.

O sistema precisa lidar com a geração de relatórios em múltiplos formatos, requisições HTTP com diversos parâmetros opcionais, clonagem de configurações de mesa, personalização de temas de interface e uma configuração global compartilhada. Esses cenários representam problemas clássicos e recorrentes na engenharia de software, e os padrões de criação fornecem soluções robustas e reutilizáveis para tratá-los.

---

2. Justificativa por Cenário

2.1 Geração de Relatórios – Factory Method

A geração de relatórios exige que o sistema suporte múltiplos formatos (PDF, HTML, JSON), sem alterar a lógica principal de geração. O padrão **Factory Method** é ideal nesse contexto, pois permite definir uma interface comum para criação de objetos (`Relatório`) e delegar às subclasses a instância concreta.

Vantagens:

- Permite adicionar novos formatos (CSV, XML) sem alterar a lógica existente.
- Reduz acoplamento entre a lógica de negócio e as classes concretas de relatório.
- Favorece testes unitários com substituição de fábricas.

Desvantagens:

- Pode resultar em várias classes, uma para cada tipo de relatório.
- Se mal utilizado, pode levar à criação de uma “God Factory”.

Impacto positivo: A separação da lógica de criação torna o sistema mais fácil de manter e estender, além de facilitar o controle de versões e testes com diferentes formatos de relatório.

2.2 Montagem de Requisição HTTP – Builder

Requisições HTTP geralmente possuem muitos parâmetros opcionais: headers, tempo limite, autenticação, etc. A aplicação do padrão **Builder** permite construir objetos complexos de forma flexível e legível, sem depender de múltiplos construtores ou configurações externas.

Vantagens:

- Reduz complexidade de criação e evita construtores longos e confusos.
- Facilita encadeamento de chamadas com `.addHeader().timeout().authToken()`.
- Permite validação antes da construção final.

Desvantagens:

- Pode parecer verboso em implementações simples.
- Exige controle adequado para evitar Builders incompletos.

Impacto positivo: Flexibiliza a criação de requisições variadas com consistência e clareza, ideal para testes, APIs variáveis e runtime configuration.

2.3 Clonagem de Configurações de Mesa – Prototype

A clonagem de configurações (ex: disposição das mesas) pode ser necessária para criar novos layouts com base em modelos existentes. O padrão **Prototype** resolve essa necessidade, evitando instanciar manualmente cada mesa com os mesmos atributos.

Vantagens:

- Alta performance ao copiar objetos complexos.
- Útil para cópias profundas com valores personalizados.

Desvantagens:

- Pode exigir cuidados com referências mutáveis.
- Difícil de manter se houver muitos atributos interdependentes.

Impacto positivo: Melhora a produtividade e evita erros repetitivos na configuração de mesas, favorecendo operações administrativas mais rápidas.

2.4 Temas de Interface – Abstract Factory

Cada tema (Light, Dark, High Contrast) define um conjunto de elementos gráficos: botões, cores e ícones. O padrão **Abstract Factory** permite criar famílias de objetos relacionados sem depender de suas classes concretas.

Vantagens:

- Garante consistência entre os elementos visuais de um mesmo tema.
- Facilita a troca de tema em tempo de execução.

Desvantagens:

- Pode introduzir complexidade na estrutura de pacotes.
- Pode ser excessivo em interfaces simples.

Impacto positivo: Garante uma identidade visual coesa e adaptável, essencial para acessibilidade e personalização de UI em sistemas modernos.

2.5 Configuração Global – Singleton

A configuração global (ex: parâmetros de conexão com banco de dados) precisa ser acessível de qualquer ponto do sistema e deve existir em uma única instância. O padrão **Singleton** é adequado, pois assegura que haja apenas uma instância do objeto em toda a aplicação.

Vantagens:

- Acesso global e controle centralizado.
- Ideal para configurações que não mudam após o início.

Desvantagens:

- Risco de uso indevido e acoplamento excessivo.
- Dificuldade de testes e substituição em ambientes controlados.

Impacto positivo: Simples de implementar, útil para armazenar configurações compartilhadas, mas deve ser usado com moderação.

---

3. Reflexões e Considerações Finais

A escolha de padrões de projeto deve sempre considerar o contexto e os requisitos do sistema. Embora existam boas práticas, a aplicação cega de padrões pode levar a problemas como **over-engineering**, **complexidade desnecessária** e **baixo desempenho**. No caso analisado:

- Os padrões escolhidos foram aplicados apenas onde havia ganho real de clareza e flexibilidade.
- A divisão das responsabilidades e a orientação a interfaces favorecem testes e manutenibilidade.
- Para evitar o acoplamento excessivo com o padrão Singleton, recomenda-se aplicar Injeção de Dependência em módulos críticos e limitar o uso global.
- A modularidade da arquitetura permite evoluir com combinações de padrões, como unir **Abstract Factory** e **Builder** para temas com opções personalizadas, ou **Prototype** e **Factory** para cópias baseadas em contexto.

O uso de padrões de criação nesse sistema provê uma fundação sólida para escalabilidade, legibilidade e testabilidade, respeitando os princípios de coesão e baixo acoplamento descritos nos princípios SOLID.

---

4. Referências

- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.
- Oracle Java Documentation. https://docs.oracle.com/javase/
- GoF Patterns Summary – Refactoring Guru. https://refactoring.guru/design-patterns
- Freeman, E., & Robson, E. (2004). *Head First Design Patterns*. O'Reilly Media.