



**Universidade de Brasília**

**Faculdade UnB Gama**

Faculdade UnB Gama - FGA

Disciplina: Técnicas de Programação em  
Plataformas Emergentes( TPPE)

Professor(a): Andre Lanna

Nome(s) - Matrícula(s): Artur Vinicius Dias Nunes - 19/0142421

Gabriel Ferreira da Silva – 20/0018060

Mateus de Almeida Dias – 19/0142260

Pedro Helias Carlos - 14/0158278

**Tópico:** Entrega Trabalho Prático 3

**Questão 1** - Para cada um dos princípios de bom projeto de código mencionados acima, apresenta sua definição e relacione-o com os maus-cheiros de código apresentados por Fowler em sua obra.

## 1. Simplicidade

- **Definição:** O código deve ser simples e direto, sem complexidade desnecessária. A simplicidade facilita a compreensão e manutenção do código.
- **Relação com maus-cheiros de código:**
  - **Long Function (Função Longa):** Funções longas são complexas e difíceis de entender. A simplicidade sugere que métodos devem ser curtos e focados em uma única tarefa.
  - **Large Class (Classe Grande):** Classes que fazem muitas coisas ao mesmo tempo adicionam complexidade. A simplicidade busca classes que sejam focadas e responsáveis por uma única responsabilidade.

## 2. Elegância

- **Definição:** Código elegante resolve problemas de forma clara e eficiente, utilizando soluções simples e diretas.
- **Relação com maus-cheiros de código:**
  - **Primitive Obsession (Obsessão por Tipos Primitivos):** Usar tipos primitivos em vez de objetos ou subclasses pode resultar em códigos menos expressivos e elegantes. Refatorar para criar objetos ou subclasses específicos pode aumentar a elegância do design.

## 3. Modularidade

- **Definição:** O código deve ser organizado em módulos ou componentes menores, com responsabilidades claramente definidas.

- **Relação com maus-cheiros de código:**
  - **Feature Envy (Inveja de Propriedade):** Quando um método em uma classe parece estar mais interessado em outra classe do que na própria, isso indica uma violação da modularidade. Movendo o método para a classe correta, melhora-se a separação de responsabilidades.

## 4. Boas interfaces

- **Definição:** Interfaces devem ser claras, consistentes e bem definidas, permitindo interações entre módulos de forma previsível e compreensível.
- **Relação com maus-cheiros de código:**
  - **Shotgun Surgery (Cirurgia com Espingarda):** Alterar uma pequena funcionalidade em uma interface mal projetada pode exigir mudanças em vários locais do código, indicando que a interface não está bem delimitada.

## 5. Extensibilidade

- **Definição:** O código deve ser fácil de estender e adaptar para novas funcionalidades, sem a necessidade de reescrever ou modificar grandes partes do sistema existente.
- **Relação com maus-cheiros de código:**
  - **Refused Bequest (Herança Recusada):** Quando uma classe herda métodos ou propriedades que não precisa, isso pode dificultar a extensão do código. Usar composição ao invés de herança pode melhorar a extensibilidade.

## 6. Evitar duplicação

- **Definição:** O código não deve repetir a mesma lógica em diferentes lugares; em vez disso, deve ser reutilizável e centralizado.
- **Relação com maus-cheiros de código:**
  - **Duplicated Code (Código Duplicado):** Código duplicado é um dos cheiros mais comuns que Fowler menciona, e refatorar para eliminar duplicações é uma prática central para evitar problemas de manutenção.

## 7. Portabilidade

- **Definição:** O código deve ser escrito de forma que possa ser executado em diferentes plataformas e ambientes com o mínimo de alteração.
- **Relação com maus-cheiros de código:**
  - **Data Clumps (Agregados de Dados):** Quando dados são agrupados inadequadamente e são fortemente ligados a um contexto específico, isso pode dificultar a portabilidade. Encapsular esses dados em objetos ou classes pode facilitar a adaptação do código para diferentes ambientes.
  - **Feature Envy (Inveja de Propriedade):** Um método que depende muito de dados de outra classe pode criar uma dependência que dificulta a separação e, consequentemente, a portabilidade do código para outros ambientes ou plataformas.

## 8. Código idiomático e bem documentado

- **Definição:** O código deve seguir as convenções da linguagem e ser bem documentado, facilitando sua compreensão e manutenção por outros desenvolvedores.
- **Relação com maus-cheiros de código:**
  - **Comments (Comentário):** Código bem documentado e idiomático deve ser claro por si só, sem a necessidade de comentários excessivos e supérfluos.
  - **Mysterious Name (Nome Misterioso):** Nomes inadequados ou confusos dificultam a compreensão do código. Fowler destaca a importância de usar nomes que sejam descritivos e coerentes para que se possa entender o que a variável faz e como usá-la.

**Questão 2** - Identifique quais são os maus-cheiros que persistem no trabalho prático 2 do grupo, indicando quais os princípios de bom projeto ainda estão sendo violados e indique quais as operações de refatoração são aplicáveis. Atenção: não é necessário aplicar as operações de refatoração, apenas indicar os princípios violados e operações possíveis de serem aplicadas.

- Método `proverCashBack` na classe `Comprar`, viola o princípio de modularidade, pode ser resolvido aplicando extrair classe.
- Método `valorFrete` na classe `Compra`, viola o princípio de modularidade, pode ser resolvido aplicando extrair classe.
- Método `ICMS` e Método `impMunicipal` na classe `imposto`, viola o princípio de duplicidade, pode ser resolvido aplicando extrair método.
- Método `valorFreteProduto` na classe `ValorTotalCompra`, viola o princípio de modularidade, pode ser resolvido aplicando extrair classe.
- Método `verificaUsoCashBack` na classe `ValorTotalCompra`, viola o princípio de modularidade, pode ser resolvido aplicando extrair classe.
- Classe `Compra`, viola o princípio de Simplicidade, pode ser resolvido aplicando o extrair classe em algumas variáveis.