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

1. Simplicidade

A simplicidade no código significa que ele deve ser fácil de entender e modificar, evitando complexidade desnecessária. Um código simples é direto e utiliza apenas os elementos necessários para resolver um problema específico.

Maus-Cheiros Relacionados:

- Funções longas: Funções com muitas linhas de código tendem a ser complexas, dificultando a compreensão e manutenção.
- Código duplicado: A duplicação de código adiciona complexidade desnecessária, tornando a manutenção mais difícil.
- Dados mutáveis: Dados que podem ser modificados em vários lugares aumentam a complexidade e podem introduzir bugs.
- Campo temporário: Variáveis temporárias usadas apenas em certas situações complicam o entendimento do código.
- Elemento ocioso: Código que não é mais necessário adiciona complexidade e deve ser removido para manter a simplicidade.
- Generalidade especulativa: Adicionar flexibilidade desnecessária antes que seja realmente necessária introduz complexidade, o que vai contra o princípio da simplicidade.

2. Elegância

A elegância no código refere-se à clareza e concisão na resolução de problemas, utilizando práticas e soluções que tornam o código limpo e fácil de entender.

Maus-Cheiros Relacionados:

- Nome misterioso: Nomes de variáveis, funções ou classes que não são claros comprometem a elegância do código, tornando-o difícil de entender.
- Laços: Laços complexos podem ser substituídos por operações de pipeline como mapear e filtrar, que são mais elegantes e facilitam a compreensão.
- Obsessão por primitivos: Utilizar tipos de dados primitivos em vez de criar tipos específicos pode comprometer a clareza e elegância do código.
- Classes de dados: Classes que apenas armazenam dados sem comportamento associado podem indicar uma falta de elegância e bom design orientado a objetos.

3. Modularidade

Modularidade se refere à divisão do código em partes independentes, onde cada módulo tem uma responsabilidade específica e bem definida. Isso facilita o desenvolvimento, teste e manutenção do código.

Maus-Cheiros Relacionados:

- Classes grandes: Classes que acumulam muitas responsabilidades violam a modularidade, pois poderiam ser divididas em várias classes menores.
- Funções longas: Métodos longos que realizam várias tarefas diferentes indicam uma falta de modularidade e podem ser divididos em métodos menores.
- Agrupamento de dados: Quando dados relacionados estão dispersos em vários lugares, isso indica uma falta de modularidade.
- Alteração divergente: Quando um módulo precisa ser alterado por diferentes razões, isso indica que a modularidade não está bem implementada.
- Cirurgia com rifle: A necessidade de modificar várias partes do código para realizar uma alteração única indica que as responsabilidades não estão bem isoladas.
- Inveja de recursos: Quando uma função ou classe depende fortemente de outra classe, isso indica uma modularidade mal implementada.
- Cadeia de mensagens: Cadeias de mensagens longas indicam um acoplamento forte entre módulos, o que compromete a modularidade.
- Intermediário: O uso de intermediários desnecessários entre classes pode indicar uma falta de modularidade.
- Herança recusada: Quando uma subclasse não utiliza métodos e dados herdados, isso pode indicar problemas na modularidade da hierarquia de classes.

4. Boas Interfaces

Boas interfaces são aquelas que são claras, coesas e fáceis de usar, definindo contratos precisos entre os diferentes componentes do sistema.

Maus-Cheiros Relacionados:

- Lista longa de parâmetros: Listas longas de parâmetros complicam o uso das funções, tornando as interfaces difíceis de entender e utilizar.
- Classes alternativas com interfaces diferentes: Quando classes que desempenham funções semelhantes têm interfaces diferentes, isso pode causar confusão e dificultar a manutenção.
- Trocas escusas: Quando módulos trocam muita informação entre si, isso indica que as interfaces não estão bem definidas.
- Cadeia de mensagens: Cadeias longas de mensagens indicam que as interfaces entre os módulos não estão claras, resultando em acoplamento excessivo.
- Inveja de recursos: Funções que dependem demais de outras classes indicam que as interfaces entre os módulos não são coesas.

5. Extensibilidade

Extensibilidade é a capacidade de adicionar novas funcionalidades ao sistema com o mínimo de mudanças no código existente, tornando-o mais adaptável a mudanças futuras.

Maus-Cheiros Relacionados:

- Switches repetidos: A repetição de estruturas de controle como switch ou if/else dificulta a extensão do código. O uso de polimorfismo é uma alternativa mais extensível.
- Dados mutáveis: Dados que podem ser modificados facilmente podem dificultar a extensão do código de maneira segura.

- Generalidade especulativa: Adicionar funcionalidades ou ganchos que não são necessários pode dificultar a extensão real quando ela se tornar necessária.
- Herança recusada: Uma hierarquia de classes mal projetada pode dificultar a extensão adequada de novas funcionalidades.

6. Evitar Duplicação

Evitar duplicação significa não repetir código em múltiplos lugares. Código duplicado é difícil de manter e pode levar a inconsistências e bugs.

Maus-Cheiros Relacionados:

- Código duplicado: A duplicação de código é um dos sinais mais claros de que o princípio de evitar duplicação está sendo violado.
- Switches repetidos: A repetição de estruturas de controle em diferentes partes do código pode ser um tipo de duplicação que deve ser eliminada.

7. Portabilidade

Portabilidade é a capacidade do código de ser executado em diferentes ambientes ou plataformas sem a necessidade de grandes modificações.

Maus-Cheiros Relacionados:

- Dados globais: O uso de dados globais pode dificultar a adaptação do código para diferentes plataformas ou ambientes, comprometendo a portabilidade.

8. Código deve ser idiomático e bem documentado

Código idiomático segue as melhores práticas e padrões da linguagem em que foi escrito. Um código bem documentado é fácil de entender e manter, com comentários que explicam partes complexas.

Maus-Cheiros Relacionados:

- Nome misterioso: Nomes inadequados ou confusos dificultam a compreensão e manutenção do código, indo contra a necessidade de um código idiomático e claro.
- Comentários: Comentários em excesso ou necessários para explicar código mal estruturado indicam que o código não é suficientemente claro ou idiomático. Código bem escrito deve ser autoexplicativo, necessitando de poucos comentários.

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.

Ao analisar o código do Trabalho Prático, encontramos um mau-cheiro que ainda persiste, mesmo após as refatorações já realizadas.

O método calcularFrete() utiliza um switch para determinar o valor do frete com base no estado do cliente. No caso de novos estados ou regras de frete serem adicionadas, pode-se ter problemas, já que as modificações seriam repetitivas no código. Esse é um mau-cheiro conhecido como **Switches Repetidos**, que compromete os princípios de evitar **duplicação** e **extensibilidade**.

Para resolver isso, poderia ser utilizado um padrão de projeto conhecido como Strategy. Com o Strategy, diferentes cálculos de frete para cada região ou estado seriam encapsulados em suas próprias classes. Isso permitiria adicionar novas regras ou estados sem precisar modificar o método calcularFrete() diretamente. Em vez disso, novas estratégias seriam criadas e facilmente integradas ao sistema, tornando o código mais fácil de manter e extender, além de evitar duplicações.