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.
    a. **Simplicidade**
        i. **Definição:** Simplicidade refere-se a escrever código que é direto e fácil de entender. Um código simples é aquele que faz apenas o necessário, sem complicações ou adições desnecessárias. O código deve ser o mais claro e conciso possível, evitando sobrecarregar a leitura com complexidades que não são essenciais para a resolução do problema.
        ii. **Relação com maus-cheiros de código:**
            1. **Duplicated Code (Código Duplicado):** A simplicidade é comprometida quando há duplicação de código. Se um mesmo trecho de código aparece em vários lugares, isso pode indicar complexidade desnecessária. Refatorar para eliminar a duplicação e criar funções reutilizáveis pode simplificar o código.
            2. **Long Method (Método Longo):** Métodos longos podem ser difíceis de entender e manter. Refatorar para dividir métodos longos em métodos menores e mais focados ajuda a manter a simplicidade do código.
    b. **Elegância**
        i. **Definição:** Elegância em código se refere a encontrar soluções que são ao mesmo tempo simples e esteticamente agradáveis. Um código elegante geralmente envolve o uso de padrões de design apropriados e boas abstrações que tornam o código não só funcional, mas também bonito em sua estrutura e fluidez.
        ii. **Relação com maus-cheiros de código:**
            1. **Shotgun Surgery:** Modificações que requerem pequenas mudanças em vários lugares indicam falta de elegância. Refatorar para agrupar funcionalidades relacionadas em um único local melhora a coesão e elegância.
            2. **Divergent Change (Mudança Divergente):** Se uma classe sofre alterações frequentes por motivos diferentes, isso reflete um design pouco elegante. Refatorar para separar responsabilidades distintas em classes diferentes pode trazer elegância ao design.
    c. **Modularidade**
        i. **Definição:** Modularidade significa que o código é organizado em módulos ou componentes que têm responsabilidades bem definidas e interagem entre si de maneira clara e mínima. Um código modular é fácil de manter e estender, pois as mudanças em um módulo têm impacto mínimo em outros.
        ii. **Relação com maus-cheiros de código:**
            1. **Large Class (Classe Grande):** Classes grandes que fazem muitas coisas comprometem a modularidade. Refatorar para dividir essas classes em módulos menores com responsabilidades específicas aumenta a modularidade.

2. **God Object (Objeto Deus):** Um objeto que sabe ou faz demais quebra a modularidade. Refatorar para delegar responsabilidades para outros objetos menores e mais focados para melhorar a modularidade.

d. **Boas interfaces**
   i. **Definição:** Boas interfaces são aquelas que são claras, bem definidas, e oferecem a menor quantidade possível de métodos necessários para a interação com outros componentes. Elas devem ser intuitivas e esconder a complexidade interna de um módulo, expondo apenas o que é essencial para o uso externo.
   ii. **Relação com maus-cheiros de código:**
      1. **Inappropriate Intimacy (Intimidade Inapropriada):** Quando uma classe conhece detalhes internos de outra, isso sugere que as interfaces entre elas não estão bem definidas. Refatorar para melhorar a separação de preocupações e reduzir o acoplamento melhora a clareza das interfaces.
      2. **Lazy Class (Classe Preguiçosa):** Se uma classe faz muito pouco, pode ser um indicativo de uma interface mal definida. Refatorar para eliminar ou combinar classes pode melhorar a definição e utilidade das interfaces.
      3. **Incomplete Library Class (Classe de Biblioteca Incompleta):** Quando uma classe de biblioteca não oferece a interface adequada para o que você precisa, levando a soluções hackeadas, isso reflete uma má interface. Refatorar para melhorar ou estender as interfaces fornecidas melhora a qualidade do código.

e. **Extensibilidade**
   i. **Definição:** Extensibilidade é a capacidade do código de acomodar futuras mudanças com o mínimo de esforço. Um código extensível permite a adição de novas funcionalidades sem alterar o comportamento existente.
   ii. **Relação com maus-cheiros de código:**
      1. **God Class (Classe Deus):** Uma classe que faz muito ou sabe muito tende a ser difícil de estender sem causar efeitos colaterais.
      2. **Rigididade:** Dificuldade em fazer mudanças porque uma pequena alteração força muitas modificações em outras partes do sistema.
      3. **Shotgun Surgery:** Quando uma modificação exige alterações em vários locais no código, sugerindo uma falta de extensibilidade.

f. **Evitar Duplicação**
   i. **Definição:** Evitar duplicação significa manter o código DRY (Don't Repeat Yourself). Isso implica que a lógica ou funcionalidade do código deve estar centralizada e reutilizável, evitando cópias espalhadas em várias partes do sistema.
   ii. **Relação com maus-cheiros de código:**

1. **Duplicated Code (Código Duplicado):** O exemplo mais direto de um mau-cheiro, onde blocos de código semelhantes aparecem em vários lugares, resultando em manutenção difícil e aumento do risco de bugs.
2. **Shotgun Surgery:** Duplicação pode levar a este mau-cheiro, onde mudanças em uma funcionalidade duplicada precisam ser feitas em vários lugares.

g. **Portabilidade**
   i. **Definição:** Portabilidade refere-se à capacidade de o código ser executado em diferentes ambientes ou plataformas sem modificações significativas. Um código portátil é aquele que pode ser transferido ou adaptado facilmente de um sistema para outro.
   ii. **Relação com maus-cheiros de código:**
      1. **Inappropriate Intimacy:** Dependências entre classes que fazem o código menos portátil, pois uma classe pode depender de detalhes de implementação específicos de outra classe.
      2. **Coupling Between Modules (Acoplamento Entre Módulos):** Um alto acoplamento pode tornar o código menos portátil, pois mudanças em um módulo podem impactar outros módulos.

h. **Código deve ser idiomático e bem documentado**
   i. **Definição:** Código idiomático é aquele que segue as convenções e padrões de design da linguagem de programação utilizada, tornando-o fácil de entender por outros desenvolvedores familiarizados com a linguagem. A documentação é essencial para explicar a intenção, uso e comportamento do código, facilitando a manutenção.
   ii. **Relação com maus-cheiros de código:**
      1. **Comments (Comentários):** Embora comentários sejam necessários, muitos comentários podem ser um sinal de que o código é confuso ou não é autoexplicativo.
      2. **Obscure Code (Código Obscuro):** Código que não segue convenções idiomáticas ou é mal documentado, tornando-o difícil de entender e manter.
      3. **Long Method (Método Longo):** Métodos longos podem se tornar difíceis de documentar adequadamente e entender, especialmente se não seguem padrões idiomáticos.

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.

   a. **Elegância**
      i. **Os nomes dos métodos não são totalmente consistentes:** Há mais de um verbo que podem explicar a mesma função e eles são escolhidos arbitrariamente em diferentes funções, por exemplo, calcularValor… e definirValor… Uma possível solução seria

padronizar os nomes, estabelecendo um formato padrão para todos os envolvidos no projeto, de preferência o mais curto e direto possível.

ii. **Nem sempre é utilizado a palavra this para se referir a um atributo da própria classe:** Em alguns casos bem específicos a palavra this não é utilizada para alterar um atributo da pŕopria classe, o que apesar de poder gerar comportamentos inesperados dentro do compilador (em diferentes versões), que seria um caso de inconsistência de Boas Interfaces, também se aplica a Elegância do código. A Solução seria simplesmente colocar a palavra this nessas situações, por meio da atenção de todos os desenvolvedores ou das ferramentas utilizadas para ajudar a detectar esses casos.