**Questão Teórica 1:**

**Classe Anêmica:**

**Situação:** Em um sistema de banco de dados para armazenar informações básicas de um catálogo de livros em uma biblioteca.

**Justificativa:** Nesse contexto, a classe anêmica faz sentido quando queremos apenas representar os dados básicos de um livro, como título, autor e ISBN. Não há muita lógica de negócios associada a esses dados, e a classe anêmica oferece uma abordagem simples para representar essas informações sem métodos ou regras de negócios complexos.

**Classe Rica:**

**Situação:** Em um sistema de comércio eletrônico que precisa lidar com cálculos de descontos, promoções e lógica de carrinho de compras.

**Justificativa:** Nesse cenário, a classe rica é mais apropriada, pois envolve lógica de negócios complexa. Por exemplo, uma classe CarrinhoDeCompras pode ter métodos para calcular descontos, aplicar promoções e gerenciar itens no carrinho. A abordagem rica permite encapsular a lógica de negócios associada ao carrinho de compras em uma única classe, tornando o código mais organizado e manutenível.

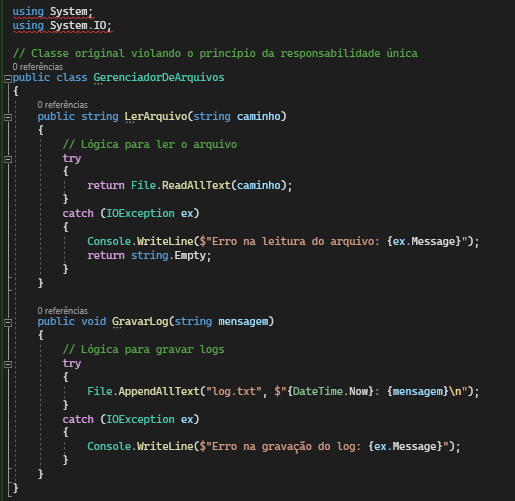
**Questão Teórica 2:**

O princípio da responsabilidade única, parte dos princípios SOLID, sugere que uma classe deve ter apenas uma razão para mudar, ou seja, deve ter apenas uma responsabilidade.

**Importância:** Esse princípio é crucial para manter o código modular, facilitar a manutenção e reduzir o acoplamento entre componentes.

**Exemplo Prático:**

Suponha que temos uma classe GerenciadorDeArquivos que lida tanto com a leitura de arquivos quanto com a gravação de logs. Isso violaria o princípio da responsabilidade única. O código resultante pode ser dividido em duas classes:

****

Neste exemplo, a classe original GerenciadorDeArquivos é dividida em duas classes, LeitorDeArquivos e GravadorDeLogs, cada uma com sua responsabilidade única

**Questão Teórica 3**

**Exemplos de Validação de Dados:**

1. **Validação de Formato:**
   * Exemplo: Garantir que um endereço de e-mail tenha o formato correto (ex: [usuario@dominio.com](mailto:usuario@dominio.com)).
2. **Validação de Faixa/Número:**
   * Exemplo: Verificar se a idade fornecida em um formulário está dentro de uma faixa válida (por exemplo, entre 18 e 100 anos).
3. **Validação de Existência:**
   * Exemplo: Certificar-se de que um campo obrigatório, como um número de telefone, não está vazio.

**Importância:**

1. **Consistência de Dados:**
   * Garante que os dados estejam formatados e estruturados corretamente, mantendo a consistência.
2. **Segurança:**
   * Ajuda a prevenir ataques comuns, como injeção de código, garantindo que apenas dados válidos sejam processados.
3. **Usabilidade:**
   * Melhora a experiência do usuário, fornecendo feedback imediato sobre dados incorretos, facilitando a correção.