**Questão 1:**

Estávamos violando o princípio do "Princípio da Responsabilidade Única" (SRP - Single Responsibility Principle). O SRP é um dos princípios do SOLID, um conjunto de diretrizes para design de software.

O SRP afirma que uma classe deve ter apenas uma razão para mudar, ou seja, deve ter apenas uma responsabilidade. Quando uma classe acumula muitas responsabilidades, torna-se difícil de manter, entender e modificar, o que pode levar a um código de baixa qualidade.

Para corrigir isso, aplicamos a "separação de preocupações" (SoC - Separation of Concerns). Isso envolve dividir as responsabilidades em classes ou módulos separados, de modo que cada um seja responsável por uma única preocupação. Por exemplo:

Regras de Negócio: Criar uma classe ou módulo dedicado exclusivamente às regras de negócio da aplicação, separando-as das outras responsabilidades.

Validações: Isolar a lógica de validação em uma classe ou módulo separado, abordando validações de entrada de dados, regras de validação de negócios e outras.

Persistência: Utilizar uma classe ou camada separada para lidar com a persistência de dados, como uma classe de repositório ou uma camada de acesso a dados.

Ao seguir a separação de preocupações, vocês melhoram a manutenção, testabilidade e compreensão do código, tornando-o mais modular e flexível. Cada classe ou módulo se concentra em uma única responsabilidade, tornando o código mais organizado e fácil de gerenciar.

**Questão 2:**

Domain (Domínio): Define entidades de domínio e regras de negócio da aplicação.

Data (Dados): Gerencia o armazenamento e recuperação de informações da aplicação.

Application (Aplicação): Coordena a execução das regras de negócio e atende às solicitações da camada de API.

API (Interface de Programação de Aplicativos): Fornece a interface para que clientes interajam com a aplicação, processando solicitações HTTP e expondo serviços.

**Questão 3:**

Usar propriedades com setters privados em classes de domínio, como no exemplo do Produto, oferece controle estrito sobre o acesso aos dados, aplicação de lógica de validação, evita efeitos colaterais indesejados, proporciona flexibilidade futura e permite a criação de objetos imutáveis, garantindo a integridade dos dados.

**Questão 4:**

A injeção de dependência é uma técnica usada na camada de Application para desacoplar as dependências de uma classe de serviço, melhorar a testabilidade, reutilização e flexibilidade, facilitando a manutenção do código. É uma prática comum que segue princípios de design sólidos.