1. a - Ao escolher entre interfaces e classes abstratas, considero os objetivos do projeto.
   * Interfaces são ideais para definir contratos sem implementações específicas, promovendo interoperabilidade; por exemplo, em sistemas de e-commerce, elas garantem que diferentes processadores de pagamento sigam o mesmo padrão.
   * Por outro lado, classes abstratas oferecem uma base comum de implementação, permitindo customizações específicas, como diferentes tipos de personagens em um jogo.

Em resumo, use interfaces para flexibilidade e contratos padronizados, e classes abstratas para compartilhar implementações e permitir especializações.

b - Na programação C#, a escolha entre herança e delegação baseia-se nas necessidades específicas do seu design. De um modo geral, use herança para relações diretas "é-um" e deseja reutilizar e estender o comportamento de uma classe existente, e delegação para promover flexibilidade e desacoplamento por meio de composição. A escolha que dependerá das especificidades e requisitos de sistema.

* + Herança, caracterizada pela relação "é-um", permite que subclasses reutilizem e ampliem funcionalidades de uma superclasse, facilitando a organização em hierarquias de classes. Por exemplo, uma classe Animal pode ser estendida por subclasses como Cachorro e Gato, herdando métodos como Andar() e Comer(), e adicionando comportamentos únicos Latir() e Miado(), respectivamente.
  + Delegação, por sua vez, segue a relação "tem-um", onde uma classe usa instâncias de outras classes para realizar determinadas funções, encaminhando chamadas de método a essas instâncias. Essa abordagem, favorecendo composição em detrimento de herança, propicia maior flexibilidade e desacoplamento, exemplificado por uma classe Veiculo que delega funcionalidades como motorização e frenagem a instâncias de classes específicas para esses componentes.

4. a - Definir tipos específicos de exceção melhora a clareza e eficácia no tratamento de erros, permitindo uma identificação precisa de problemas. Isso é útil especialmente para representar erros únicos ou organizar erros em categorias, como “PagamentoException” ou erros de banco de dados. Além disso, torna o código mais legível e facilita a manutenção, com o uso de exceções específicas como FileNotFoundException, o que permite um tratamento mais adequado para cada tipo de erro, aumentando a robustez do código.

b - Eu utilizaria em ações propensas a falhas devido a circunstâncias imprevisíveis, como leitura de arquivos, comunicações de rede ou interações com bancos de dados. Por exemplo, ao tentar acessar um arquivo inexistente, uma `FileNotFoundException` poderia ser lançada e capturada para tratamento adequado.

c - Quando ocorrer uma situação excepcional que não possa ser tratada de forma adequada no contexto atual e que exija uma ação especial ou notificação para ser corrigida. Como por exemplo:

* Validação de Entrada:
  + Quando os dados fornecidos como entrada para uma função ou método são inválidos. Exemplo: Uma função que espera um número positivo como argumento lança uma exceção se um número negativo for fornecido.

5. Garantir a integridade dos dados e a estabilidade do sistema em um ambiente de transações concorrentes, exige a adoção de práticas de transações atômicas, bloqueio de recursos, tratamento cuidadoso de exceções, validações robustas, testes de concorrência e uma boa estratégia de monitoramento:

* Assegurar que as ações de débito e crédito ocorram de forma completa ou não ocorram, utilizando transações de banco de dados para manter a consistência dos dados.
* Gerenciar exceções de forma eficaz para manter o estado consistente da conta, mesmo em falhas. Implementar verificações para impedir valores negativos ou nulos e outros tipos de dados inválidos.
* Conduzir testes detalhados para assegurar o funcionamento adequado do sistema sob carga concorrente, stabelecer mecanismos para acompanhamento e registro das operações, facilitando a identificação e correção de problemas.

6.

SELECT

PL.nome AS nome\_produto\_limpeza,

A.nome AS nome\_alimento,

(EE.preco \* E.quantidade) \* 0.85 AS preco\_kit,

preco\_kit - EE.custo AS lucro\_kit,

A.data\_validade AS data\_validade\_kit

FROM

Produto\_Limpeza PL

JOIN

Elemento\_Estoque EE ON PL.id\_elemento\_estoque = EE.id

JOIN

Pesquisa\_Mercado PM ON PL.id = PM.id\_produto\_limpeza

JOIN

Alimento A ON A.id\_elemento\_estoque = EE.id

JOIN

Estoque E ON E.id\_elemento\_estoque = EE.id

WHERE

PM.satisfacao > 0.7

AND DATEPART(GETDATE(), A.data\_validade) < 5

ORDER BY

lucro\_kit DESC