Questões relacionadas a C#

1. Orientação a Objetos:

• Explique o conceito de herança múltipla e como C# aborda esse cenário.

O C# não suporta herança múltipla direta de classes. Uma classe só pode herdar de uma única classe base. Isso evita problemas como ambiguidade e complexidade.
No entanto, C# fornece uma alternativa que é a implementação de múltiplas interfaces. Uma interface define um contrato (ou conjunto de métodos/propriedades que precisam ser implementados), mas não fornece implementação. Como uma classe pode implementar várias interfaces, você obtém os benefícios da herança múltipla sem os problemas.

• Explique o polimorfismo em C# e forneça um exemplo prático de como ele pode ser implementado.

O polimorfismo em C# é a capacidade de um método, propriedade ou operador se comportar de diferentes formas, dependendo de como é usado. Existem dois tipos principais:

- Polimorfismo de Sobrecarga (Compile-Time): Mesma função/método, mas com assinaturas diferentes (parâmetros diferentes).
- Polimorfismo de Substituição (Run-Time): Métodos que podem ser sobrescritos em classes derivadas usando virtual e override.

```
using System;

public abstract class Cliente
{
    public string Nome { get; set; }

    public Cliente(string nome)
    {
        Nome = nome;
    }

    // Método abstrato para calcular desconto
    public abstract decimal CalcularDesconto(decimal valorTotal);
}

public class ClienteRegular : Cliente
{
    public ClienteRegular(string nome) : base(nome) { }

    public override decimal CalcularDesconto(decimal valorTotal)
    {
        return valorTotal * 0.05m; // 5% de desconto
    }
}

public class ClienteVip : Cliente
{
```

```
    public ClienteVip(string nome) : base(nome) { }

    public override decimal CalcularDesconto(decimal valorTotal)
    {
        return valorTotal * 0.15m; // 15% de desconto
    }
}

public class ClientePremium : Cliente
{
    public ClientePremium(string nome) : base(nome) { }

    public override decimal CalcularDesconto(decimal valorTotal)
    {
        return valorTotal * 0.25m; // 25% de desconto
    }
}

public class Program
{
    public static void Main()
    {
        Cliente[] clientes = {
            new ClienteRegular("João"),
            new ClienteVip("Ana"),
            new ClientePremium("Carlos")
        };

        decimal valorCompra = 1000m;

        foreach (var cliente in clientes)
        {
            Console.WriteLine($"{cliente.Nome} tem um desconto de
{cliente.CalcularDesconto(valorCompra):C}.");
        }
    }
}
```

2. SOLID:

• Descreva o princípio da Responsabilidade Única (SRP) e como ele se aplica em um contexto de desenvolvimento C#.

O Princípio da Responsabilidade Única define que cada classe deve ter apenas uma responsabilidade ou funcionalidade específica, o que facilita a manutenção e reduz o acoplamento entre classes. No contexto de desenvolvimento em C#, ao meu ver se aplica de três formas:

- Facilita a manutenção porque quando uma classe faz muitas coisas, é mais difícil identificar e corrigir problemasE
- Evita acoplamento porque classes com muitas responsabilidades tendem a se tornar dependentes de vários outros módulos

- Promove reutilização porque classes com responsabilidades claras são mais fáceis de reaproveitar.

• Como o princípio da inversão de dependência (DIP) pode ser aplicado em um projeto C# e como isso beneficia a manutenção do código?

Pode ser aplicado utilizando interfaces que serão consumidas diretamente ao invés de classes dependerem diretamente de implementações concretas, elas devem depender de interfaces ou abstrações. Isso aumenta a flexibilidade, facilita testes tornando possível substituir dependências concretas por mocks/stubs, reduz acoplamento tornando módulos possíveis de serem modificados ou substituídos sem impactar outros e melhora a manutenção do código porque alterações em implementações específicas não afetam o código que usa a abstração.

3. Entity Framework (EF):

• Como o Entity Framework gerencia o mapeamento de objetos para o banco de dados e vice-versa?

A modelagem do banco de dados é feita com classes que representam tabelas do banco onde o nome da classe é o nome da tabela, propriedades públicas são as colunas e propriedade “ID” ou “NomeDaClasseID” vira chave primária da tabela no banco de dados.
Quem define quais classes de domínio serão tabelas no banco de dados é o “DBContext” que, por padrão, faz o mapeamento automático.
Nas consultas e manipulações de dados, o EF faz a tradução de consultas LINQ em SQL para buscar dados e para adição, edição ou exclusão de objetos, o EF gerencia a execução das operações SQL necessárias.
No caso específico de consultas, o EF gera o SQL equivalente, executa a consulta no banco e preenche os objetos com os dados retornados.

• Como otimizar consultas no Entity Framework para garantir um desempenho eficiente em grandes conjuntos de dados?

Algumas ações pode ser utilizadas para otimizar as consultas, tais como:
- Usar AsNoTracking() para consultas que não terão modificação nos dados (dados sujos)
- Utilizar o Select() para determinar quais colunas serão selecionadas evitando buscar todas as colunas da tabela
- Utilizar Skip e Take para paginação de grandes volumes de dados
- Utilizar Where antes de ToList ou Count para filtrar corretamente os dados
- Utilizar Any ou Exists antes de buscar os dados com ToList ou Count para evitar consultas desnecessárias
- Utilizar cache de resultados para consultas que não mudam frequentemente usando bibliotecas como EF Second Level Cache
- Implementar índices no banco de dados e utilizá-los em consultas
- Utilizar o Lazy Loading para evitar carregar relacionamentos desnecessários

4. WebSockets:

• Explique o papel dos WebSockets em uma aplicação C# e como eles se comparam às solicitações HTTP tradicionais.

O papel dos WebSockets é tornar a comunicação bidirecional e persistente resultando em uma única conexão para troca de dados e com baixa latência. No caso do HTTP a comunicação não é feita de forma persistente, sendo necessárias requisições diferentes para cada vez que quiser fazer comunicação com o servidor.

• Quais são as principais considerações de segurança ao implementar uma comunicação baseada em WebSockets em uma aplicação C#?

Sobre segurança com WebSockets é importante usar WSS (WebSocket Secure) além de implementar autenticação e autorização.
É interessante também validar mensagens e cabeçalhos, limitar conexões, tamanhos e tempos e monitorar e proteger contra ataques.

5. Arquitetura:

• Descreva a diferença entre arquitetura monolítica e arquitetura de microsserviços. Qual seria sua escolha ao projetar uma aplicação C#?

A forma como o sistema é estruturado e gerenciado é a diferença entre arquitetura monolítica e arquitetura de microsserviços.
No caso da arquitetura monolítica, toda a aplicação é desenvolvida como um único bloco/projeto que contém todas as funcionalidades (UI, lógica de negócios, acesso a dados, etc.).
Já na arquitetura de microsserviços, aplicação é dividida em pequenos serviços independentes, cada um responsável por uma funcionalidade específica onde os serviços se comunicam entre si via APIs (geralmente HTTP/REST ou gRPC).
Para pequenos projetos eu usaria arquitetura monolítica com ASP.NET Core, pois é mais fácil começar e escalar gradualmente.
Paga grandes sistemas, optaria por microsserviços com ASP.NET Core ou gRPC.

• Como você escolheria entre a arquitetura de microsserviços e a arquitetura monolítica ao projetar uma aplicação C# que precisa ser altamente escalável?

Para fazer esta escolha, eu estabeleceria alguns critérios como:

- Complexidade
- Escalabilidade
- Custo
- Manutenção
- Evolução

E eu escolheria a arquitetura de microsserviços levando em consideração que é melhor em escalabilidade, manutenção e evolução porque, em vez de escalar todo o sistema, posso escalar apenas o serviço que está sob maior demanda e o todo é mais fácil de evoluir, já que cada serviço é independente.