Questões relacionadas a C#

1. Orientação a Objetos:

Explique o conceito de herança múltipla e como C# aborda esse cenário.
Herança múltipla é o cenário em que uma classe pode herdar várias classes.
O C# não possibilita herança múltipla explícita (Classe X : Classe Y, Classe Z) porém é permitido implementar várias interfaces e com isso uma classe pode implementar métodos assinados em mais de uma interface.
Ha também a tecnica de composição, que na prática, eu com a Classe X, posso criar instâncias das classe Y e Z, e criar metodos, os quais, nestes, eu chamo os métodos das classes Y e Z.

Explique o polimorfismo em C# e forneça um exemplo prático de como ele pode ser implementado.
Polimorfismo vem de muitas formas, um método com comportamentos diferentes.
Na prática, polimorfismo pode ser por sobrecarga ou sobreescrita.

Sobrescrita:

```
Classe X{
  public virtual void FazAlgo(){
    Console.Write("Faz X coisa");
  }
}
```

```
Classe Y : X{
  public override void FazAlgo(){
    Console.Write("Faz Y coisa");
  }
}
```

```
Classe Z : X{
  public override void FazAlgo(){
    Console.Write("Faz Z coisa");
  }
}
```

Sobrecarga:

```
public void FazAlgo(int a){
    Console.Write("Valor:" + a);
  }
```

```
public void FazAlgo(int a, int b){
    Console.Write("Soma:" + (a + b));
  }
```

```
public void FazAlgo(int a, int b){
    Console.Write("Subtrai:" + (a - b));
  }
```

---

2. SOLID:

Descreva o princípio da Responsabilidade Única (SRP) e como ele se aplica em um contexto de desenvolvimento C#.
S do SOLID, afirma que uma classe deve ter apenas uma uníca tarefa/ação/responsabilidade.
Usando em C# tem-se um código mais organizado e de melhor manutenção.

Forma de aplicação em C#:

```
Classe CalculadoraDescontos{
  public void CalcularINSS(){ }
  public void CalcularIRRF(){ }
}
```

```
Classe Repositorio{
  public void SalvarDados(){ }

}
```

```
Classe Cliente{
  public string Nome { get; set; }
  public double Desconto {get; set;}
}
```

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

D do SOLID – permite a separação de modúlos, isso permite alterar partes do código sem alterar outras, gerar novas implementações e permite melhor testabilidade do sistema.

Exemplo de DIP que utilizei no teste prático (Blog) deste processo seletivo (código completo dos métodos no projeto):

```
public interface IServicoWebSocket{

 Task HandleWebSocketConexao(HttpContext context);

 Task EnviarMsgParaTodos(string     message);
}
```

```
public class ServicoWebSocket : IServicoWebSocket{
public async Task     HandleWebSocketConexao(HttpContext context) {

}
public async Task EnviarMsgParaTodos(string mensagem{

}
```

```
public class ServicoNotificacao{
    private readonly IServicoWebSocket _webSocketServico;

    // Injeção de Dependência (DI)
    public ServicoNotificacao(IServicoWebSocket webSocketServico)
    {
        // Dependência WebSocket injetada
        _webSocketServico = webSocketServico;
    }

    public async Task EnviarNotificacao(Postagem postagem)
    {
        // Enviar notificação para todos os WebSockets conectados (responsabilidade única de notificação)
        var mensagem = $"{postagem.UserId} publicou: '{postagem.DesConteudo}' ";
        // Delegando a responsabilidade de enviar a mensagem
        await _webSocketServico.EnviarMsgParaTodos(mensagem);
    }
  }
```

---

3. Entity Framework (EF):

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

Com o OMR cada classe representa uma tabela no banco, e as propriedades seriam as colunas. Usando anotação (data annotation) é permitido mapear os respectivos campos/tabelas.

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

- Optar por estruturas de dados que suportam lazy loading, e manipular a ativação, ou não, do Lazy Load conforme necessidade.

- Optar por filtros LINQ antes de carregar dados na memória, ao invés de buscar dados do banco para memória e só depois realizar filtro.

- Entender o caso, e desabilitar, quando conveniente, o NoTracking.

- Criação de índices, avaliando caso, pois, diversos índices podem causar lentidão nas ações que não são consultas.

- Usar métodos para consultas assíncronas.

---

4. WebSockets:

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

Uma vez iniciada a conexão webSockets ela fica aberta permitindo comunicação em tempo real e troca de dados bidirecionais

HTTP	WebSockets
Conexão é aberta e fechada a cada requisição	Uma vez estabelecida a conexão esta permanece aberta até que um dos lados decida fechar
Comunicação unidirecional, requisição e resposta.	Comunicação bidirecional, envio de dados sem requisição
Stateless – servidor não mantem estado	Stateful – servidor pode manter o estado, sem cookies ou sessions
Mais compatível com firewall e proxies	Pode ser bloqueado por firewalls e proxies

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

Usar wws, semelhante ao https, ou seja, utiliza criptografia nos dados.
Implementar autenticação já que a conexão fica aberta constante, então deve-se garantir que apenas usuários autenticados acessem a conexão.
Trativa de fechamento de conexões inativas, pois conexões inativas podem ser ainda mais vulneráveis.

---

5. Arquitetura:

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

Monolítica pode se resumir e exemplificar em uma interface com usuário, um banco de dados, e uma aplicação do lado do servidor.
Microsserviços é distribuído e descentralizado, cada serviço fica responsável por partes do sistema e a comunicação se da por API, normalmente armazenadas em conteiners.
Minha escolha dependeria da avaliação do projeto, se for um projeto simples optaria por monolítica, se for um projeto complexo microsserviços seria uma melhor opção pensando na flexibidade e escalabilidade.

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

Quanto mais um projeto em arquitetura monolítica cresce, menos escalável ele tende a ser, como é tudo parte do projeto, se apenas uma parte estiver causando gargalo, uma solução para o problema pode impactar toda a aplicação.
Com isso, já sabendo que a aplicação precisa ser altamente escalável, optaria por microsserviços, mesmo que isso demande maior tempo para arquitetar o projeto, e maior capacitação da equipe para lidar com essa forma de desenvolvimento.

Teste prático C# - Projeto de Blog Simples: https://github.com/mayconfreitasf/ProjetoLuzaBlog

---