

Parte 1: Teste Teórico

1. Explique o conceito de encapsulamento e como ele pode ser aplicado em uma classe C#. Dê um exemplo prático.

R: Encapsulamento é um dos pilares da programação orientada a objetos no qual consiste em ocultar os detalhes internos de uma classe/objeto e expor apenas o que é necessário através de uma interface definida. Em C# por exemplo, isso é feito usando modificadores de acesso como `private`, `protected` e `public`.

Exemplo:

```
C: > Users > JacksonCosta > Desktop > C# teste.cs > ...
0 references
1 public class ContaBancaria
2 {
3     4 references
4     private decimal saldo; //Campo privado
5
6     0 references
7     public decimal Saldo => saldo; //Propriedade somente leitura
8
9     0 references
10    public void Depositar(decimal valor)
11    {
12        if (valor > 0)
13            saldo += valor;
14    }
15
16    0 references
17    public bool Sacar(decimal valor)
18    {
19        if (valor > 0 && saldo >= valor)
20        {
21            saldo -= valor;
22            return true;
23        }
24        return false;
25    }
26 }
```

2. Descreva o princípio do Open/Closed e implemente um pequeno exemplo em C# que respeite esse princípio.

R: O princípio Open/Closed (Aberto/Fechado) diz que uma classe deve estar aberta para extensão, mas fechada para modificação. Isso significa que podemos adicionar novos comportamentos sem alterar o código existente.

Exemplo:

Situação inicial (violando OCP):

```
C: > Users > JacksonCosta > Desktop > C# teste.cs > ...
0 references
1 public class ProcessadorPagamentos
2 {
3     0 references
4     public void Processar(Pagamento pagamento)
5     {
6         if (pagamento.Tipo == "CartaoCredito")
7         {
8             // Lógica complexa para cartão de crédito
9             Console.WriteLine("Processando cartão de crédito...");
10        }
11        else if (pagamento.Tipo == "Boleto")
12        {
13            // Lógica específica para boleto
14            Console.WriteLine("Processando boleto bancário...");
15        }
16        // Novo método? Tem que modificar esta classe!
17    }
18 }
```

Problema: Cada novo método de pagamento exige modificação na classe ProcessadorPagamentos.

Solução aplicando OCP:

```
C# teste.cs X
C: > Users > JacksonCosta > Desktop > C# teste.cs > ...
1 //IMPLEMENTAMOS UMA CLASSE BASE
2 4 references
3 public interface IMetodoPagamento
4 {
5     4 references
6     void Processar();
7 }
8 }
```

```
C# teste.cs X
C: > Users > JacksonCosta > Desktop > C# teste.cs > ...
7 //IMPLEMENTA CLASSES CONCRETAS
8 0 references
9 public class CartaoCredito : IMetodoPagamento
10 {
11     2 references
12     public void Processar()
13     {
14         Console.WriteLine("Processando cartão de crédito:");
15         Console.WriteLine("- Validar CVV");
16         Console.WriteLine("- Autorizar com operadora");
17         Console.WriteLine("- Confirmar transação");
18     }
19 }
20 0 references
21 public class BoletoBancario : IMetodoPagamento
22 {
23     2 references
24     public void Processar()
25     {
26         Console.WriteLine("Processando boleto:");
27         Console.WriteLine("- Gerar código de barras");
28         Console.WriteLine("- Registrar na CIP");
29         Console.WriteLine("- Enviar e-mail ao cliente");
30     }
31 }
```

Teste Técnico - Desenvolvedor | Jackson Costa

```
C# teste.cs x
C: > Users > JacksonCosta > Desktop > C# teste.cs > ...

30 //PROCESSADOR GENÉRICO [FECHADO PARA MODIFICAÇÃO]
0 references
31 public class ProcessadorPagamentos
32 {
0 references
33     public void Processar(IMetodoPagamento metodo)
34     {
35         metodo.Processar(); // Delega para a implementação específica
36     }
37 }
38
39 //ADICIONA NOVO MÉTODO SEM MODIFICAR PROCESSADOR
0 references
40 public class Pix : IMetodoPagamento
41 {
2 references
42     public void Processar()
43     {
44         Console.WriteLine("Processando Pix:");
45         Console.WriteLine("- Validar chave");
46         Console.WriteLine("- Confirmar transação via BC");
47         Console.WriteLine("- Notificar recebedor");
48     }
49 }
```

3. Qual será a saída do código abaixo? Justifique sua resposta.

```
public class Animal
{
    public virtual void Falar() => Console.WriteLine("Animal faz som");
}

public class Cachorro : Animal
{
    public override void Falar() => Console.WriteLine("Latido");
}

public class Gato : Animal
{
    public new void Falar() => Console.WriteLine("Miau");
}

Animal a1 = new Cachorro();
Animal a2 = new Gato();

a1.Falar();
a2.Falar();
```

Teste Técnico - Desenvolvedor | Jackson Costa

R: A saída será: Latido e Animal faz som, porque:

- ➔ a1.Falar() chama o método sobrescrito em Cachorro (usando override), então "Latido" é impresso.
- ➔ a2.Falar() chama o método da classe base Animal porque Gato usa new em vez de override, então "Animal faz som" é impresso.

4. Qual a diferença entre os modificadores private, protected e public no TypeScript? Dê um exemplo para cada um.

- ➔ private: Acessível apenas dentro da classe que o define.
- ➔ protected: Acessível dentro da classe e suas subclasses.
- ➔ public: Acessível de qualquer lugar (padrão).

Exemplo:

```
C# teste.cs TS Exemplo.ts X
C: > Users > JacksonCosta > Desktop > TS Exemplo.ts > ...
1  class Pessoa {
2      private id: number;
3      protected nome: string;
4      public idade: number;
5
6      constructor(id: number, nome: string, idade: number) {
7          this.id = id;
8          this.nome = nome;
9          this.idade = idade;
10     }
11 }
12
13 class Funcionario extends Pessoa {
14     mostrarNome() {
15         console.log(this.nome); // OK - protected
16         // console.log(this.id); // Erro - private
17     }
18 }
19
20 const p = new Pessoa(1, "João", 30);
21 console.log(p.idade); // OK - public
22 // console.log(p.nome); // Erro - protected
23 // console.log(p.id); // Erro - private
24
```

5. O que são pipes no Angular? Como criar um pipe personalizado para formatar um CPF?

R: Pipes são recursos do Angular para transformar dados em templates.

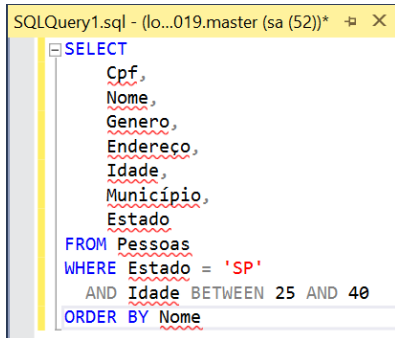
Exemplo CPF:

```
C# teste.cs TS Exemplos 9+ X
C: > Users > JacksonCosta > Desktop > TS Exemplos > ...
1  import { Pipe, PipeTransform } from '@angular/core';
2
3  @Pipe({ name: 'cpf' })
4  export class CpfPipe implements PipeTransform {
5      transform(value: string): string {
6          if (!value) return '';
7
8          return value.replace(/(\d{3})(\d{3})(\d{3})(\d{2})/, '$1.$2.$3-$4');
9      }
10 }
11
12 //EXEMPLO DE USO
13 {{ '12345678901' | cpf }} <!-- Exibe: 123.456.789-01 -->
14
```

Teste Técnico - Desenvolvedor | Jackson Costa

6. Considere a tabela Pessoas com os campos (Cpf, Nome, Genero, Endereco, Idade, Municipio, Estado). Escreva uma query SQL para buscar todas as pessoas do estado de SP com idade entre 25 e 40 anos, ordenadas pelo nome.

R:

A screenshot of a SQL query editor window titled 'SQLQuery1.sql - (lo...019.master (sa (52)))'. The query is as follows:

```
SELECT
  Cpf,
  Nome,
  Genero,
  Endereco,
  Idade,
  Municipio,
  Estado
FROM Pessoas
WHERE Estado = 'SP'
AND Idade BETWEEN 25 AND 40
ORDER BY Nome
```

7. Explique como funciona o ciclo de vida de um componente Angular. Qual método é ideal para fazer uma chamada HTTP quando o componente for exibido?

R: O ciclo de vida de um componente Angular inclui os seguintes hooks principais:

- `ngOnChanges()`: Quando dados de entrada mudam
- `ngOnInit()`: Após a primeira inicialização
- `ngDoCheck()`: Durante cada verificação de mudança
- `ngAfterContentInit()`: Após conteúdo externo ser inserido
- `ngAfterContentChecked()`: Após verificação do conteúdo
- `ngAfterViewInit()`: Após inicialização da view
- `ngAfterViewChecked()`: Após verificação da view
- `ngOnDestroy()`: Antes do componente ser destruído

O método **`ngOnInit()`** é o mais indicado para fazer chamadas HTTP quando o componente é exibido.

8. Analise o trecho abaixo:

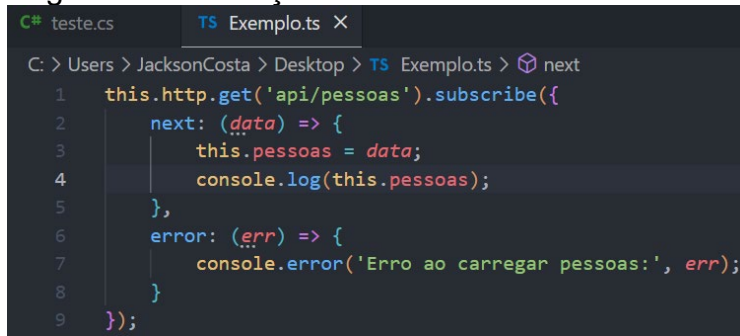
```
this.http.get('api/pessoas').subscribe(data => {  
  this.pessoas = data;  
});  
  
console.log(this.pessoas);
```

O que está errado ou incompleto nesse código? Como corrigiria?

R:

- O console.log será executado imediatamente, antes da resposta HTTP chegar (assincronismo).
- Falta tratamento de erros na chamada HTTP.

Sugestão de correção:



```
C# teste.cs TS Exemplo.ts X  
C: > Users > JacksonCosta > Desktop > TS Exemplo.ts > next  
1  this.http.get('api/pessoas').subscribe({  
2    next: (data) => {  
3      this.pessoas = data;  
4      console.log(this.pessoas);  
5    },  
6    error: (err) => {  
7      console.error('Erro ao carregar pessoas:', err);  
8    }  
9  });
```

Parte 2: Desafio Prático

Objetivo: Criar uma aplicação simples Angular 17 integrada com uma API mockada em C#.

Back-end (C# - ASP.NET Core):

- Criar uma WebAPI com endpoint GET /api/pessoas
- Retornar uma lista mockada com 30 registros de pessoas contendo os campos:
 - Cpf (string)
 - Nome (string)
 - Genero (string)
 - Endereco (string)
 - Idade (int)
 - Municipio (string)
 - Estado (string)

Teste Técnico - Desenvolvedor | Jackson Costa

Front-end (Angular 17):

- Criar uma tela com uma tabela paginada exibindo os registros da API.
- A tabela deve:
 - Exibir 10 registros por página
 - Ter um campo de filtro por Nome
 - Ter um botão para carregar os dados da API

Observações:

- Utilize Angular Material ou outro framework de UI para a tabela e paginação.
- Utilize HttpClient para comunicação com a API.
- A solução deve funcionar localmente, sem necessidade de banco de dados real.
- Envie o projeto com instruções de como executar (README).