AULA COM GABARRITO 30 31

Exercício 1: Simulação de Tempo de Espera

Descrição: Crie um método assíncrono que simule um atraso de 3 segundos e depois exiba uma mensagem.

CÓDIGO

```
using System;
using System.Threading.Tasks;
class Program
{
    static async Task Main(string[] args)
    {
        Console.WriteLine("Aguardando...");
        await SimularAtraso();
        Console.WriteLine("Atraso concluído.");
    }

    static async Task SimularAtraso()
    {
        await Task.Delay(3000); // Atraso de 3 segundos
    }
}
```

Explicação: O método SimularAtraso usa await Task.Delay(3000) para esperar 3 segundos antes de continuar.

EXERCÍCIO 2: REQUISIÇÃO HTTP ASSÍNCRONA

Descrição: Crie um método assíncrono que faça uma requisição HTTP para um URL e retorne o status da resposta.

CÓDIGO

```
using System;
using System.Net.Http;
using System.Threading.Tasks;
class Program
{
    static async Task Main(string[] args)
    {
        string url = "https://api.github.com/";
        HttpResponseMessage resposta = await ObterStatusHttpAsync(url);
        Console.WriteLine($"Status Code: {resposta.StatusCode}");
    }

    static async Task<HttpResponseMessage> ObterStatusHttpAsync(string url)
    {
        using HttpClient client = new HttpClient();
        client.DefaultRequestHeaders.Add("User-Agent", "C# App");
        return await client.GetAsync(url);
    }
}
```

Explicação: O método ObterStatusHttpAsync faz uma requisição HTTP assíncrona e retorna a resposta.

EXERCÍCIO 3: PROCESSAMENTO DE LISTA DE NÚMEROS

Descrição: Crie um método assíncrono que recebe uma lista de números, simula um atraso para cada número e retorna a soma dos números processados.

CÓDIGO

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Threading.Tasks;


class Program
{
    static async Task Main(string[] args)
    {
        var numeros = new List<int> { 1, 2, 3, 4, 5 };
        int soma = await SomarNumerosAssincronos(numeros);
        Console.WriteLine($"Soma dos números: {soma}");
    }


    static async Task<int> SomarNumerosAssincronos(List<int> numeros)
    {
        var tarefas = numeros.Select(async numero =>
        {
            await Task.Delay(500); // Simula um atraso de 500 ms para cada
número
            return numero;
        });


        var resultados = await Task.WhenAll(tarefas);
        return resultados.Sum();
    }
}
```

EXPLICAÇÃO: SOMARNUMEROSASSINCRONOS SIMULA UM ATRASO PARA CADA NÚMERO E CALCULA A SOMA DOS RESULTADOS.

EXERCÍCIO 4: CARREGAMENTO DE DADOS ASSÍNCRONO

Descrição: Crie um método assíncrono que simule o carregamento de dados (como ler um arquivo ou fazer uma requisição) e retorne uma string com os dados carregados.

CÓDIGO

```
using System;
using System.Threading.Tasks;
class Program
{
    static async Task Main(string[] args)
    {
        string dados = await CarregarDadosAsync();
        Console.WriteLine($"Dados carregados: {dados}");
    }
    static async Task<string> CarregarDadosAsync()
    {
        await Task.Delay(2000); // Simula um atraso de 2 segundos
        return "Dados carregados com sucesso!";
    }
}
```

Explicação: CarregarDadosAsync simula um carregamento de dados com um atraso e retorna uma string.

EXERCÍCIO 5: MANIPULAÇÃO ASSÍNCRONA DE ARQUIVOS

Descrição: Crie um método assíncrono que lê o conteúdo de um arquivo e o exibe na tela. Use um arquivo de texto com algum conteúdo.

CÓDIGO

```
using System;
using System.IO;
using System.Threading.Tasks;
class Program
{
    static async Task Main(string[] args)
```

```
    {
        string caminhoArquivo = "exemplo.txt";
        string conteudo = await LerArquivoAsync(caminhoArquivo);
        Console.WriteLine($"Conteúdo do arquivo:\n{conteudo}");
    }

    static async Task<string> LerArquivoAsync(string caminhoArquivo)
    {
        return await File.ReadAllTextAsync(caminhoArquivo);
    }
}
```

Explicação: LerArquivoAsync usa File.ReadAllTextAsync para ler o conteúdo de um arquivo de forma assíncrona.

EXERCÍCIO 6: VÁRIOS MÉTODOS ASSÍNCRONOS

Descrição: Crie um programa que chama vários métodos assíncronos e aguarda a conclusão de todos eles usando Task.WhenAll.

CÓDIGO

```
using System;
using System.Threading.Tasks;
class Program
{
    static async Task Main(string[] args)
    {
        Task tarefa1 = MetodoAssincrono1();
        Task tarefa2 = MetodoAssincrono2();

        await Task.WhenAll(tarefa1, tarefa2);

        Console.WriteLine("Ambos os métodos assíncronos foram concluídos.");
```

```
    }
    static async Task MetodoAssincrono1()
    {
        await Task.Delay(1000); // Simula um atraso de 1 segundo
        Console.WriteLine("Método 1 concluído.");
    }
    static async Task MetodoAssincrono2()
    {
        await Task.Delay(1500); // Simula um atraso de 1,5 segundos
        Console.WriteLine("Método 2 concluído.");
    }
}
```

Explicação: Task.WhenAll aguarda a conclusão de ambos os métodos assíncronos antes de exibir a mensagem final.

Esses exercícios cobrem vários aspectos da programação assíncrona com async e await em C#.