Pedro Inácio Rodrigues Pontes

Prática 6: Monitor de Criptomoedas(Threads Assíncronas)

Belo Horizonte, Brasil

2025

1 Introdução

O objetivo da presente prática foi a criação de um programa que monitora o valor de criptomoedas predefinidas. Os valores delas são obtidos através de uma API que faz a requisição de seus valores. As requisições devem ser feitas através de threads assíncronas paralelas, para evitar a demora exarcebada de aguardar cada requisição http.

2 Desenvolvimento

2.1 Métodos Iniciais

Foram dados inicialmente os métodos vitais para a aplicação do programa:

```
static HttpClient CriarClienteHttp() {
        var cliente = new HttpClient {
        Timeout = TimeSpan.FromSeconds(10)
        };
        cliente.DefaultRequestHeaders.Add("User-Agent",
           "MonitorCripto/1.0");
        cliente.DefaultRequestHeaders.Add("Accept",
           "application/json");
        return cliente;
}
```

Faz a conexão com a API, definindo um tempo máximo para requisição de 10 segundos (após isso é lançada uma TaskCanceledException) e define que o formato de saída de dados esperado é o JSON.

```
async Task ObterEConverterCotacaoAsync(string simbolo,
   CancellationToken token) {
        var clienteHttp = CriarClienteHttp();
        var urlRequisicao =
    $"https://api.exchange.cryptomkt.com/api/3/public/price/rate?from
        var resposta = await
           clienteHttp.GetAsync(urlRequisicao, token);
        resposta.EnsureSuccessStatusCode();

        var json = await
           resposta.Content.ReadAsStringAsync(token);

        using var documento = JsonDocument.Parse(json);
```

```
        if (documento.RootElement.TryGetProperty(simbolo,
           out var dadosMoeda))
    {
             var precoString =
                dadosMoeda.GetProperty("price").GetString();
                decimal precoAtual =
                   decimal.Parse(precoString,
                CultureInfo.InvariantCulture);
    // Armazene o valor em um ConcurrentDictionary
    }
}
```

Cria o cliente http, faz a requisição, armazena os dados json da resposta - os quais são obtidos de forma assíncrona, para garantir que seja possível deixar essa função toda assíncrona -, passa os dados adequadamente para Json - utilizando JsonDocument.Parse (o using irá descartar a memória usada por ele após o fim do uso) -, e, por fim, os valores são obtidos e armazenados.

```
void ExibirResultadosNoConsole(string simbolo, decimal
   precoAtual, decimal precoAnterior) {
        var corOriginal = Console.ForegroundColor;
        Console.ForegroundColor = precoAtual > precoAnterior
           ? ConsoleColor.Green : ConsoleColor.Red;
        Console.WriteLine($"{simbolo}: ${precoAtual:N2}
           {(precoAtual > precoAnterior ? "   " : "   ")}");
        Console.ForegroundColor = corOriginal;
}
```

Exibe os resultados.

```
async Task MonitorarTeclaEscAsync(CancellationTokenSource
   cts) {
        while (true) {
        if (Console.KeyAvailable &&
           Console.ReadKey(true).Key == ConsoleKey.Escape) {
                cts.Cancel();
                break;
        }
        await Task.Delay(100);
        }
}
```

Faz com que a tecla *Esc* gere um Token de cancelamento que para as requisições à API.

2.2 Solução

Todos os métodos, exceto o Token de cancelamento foram implementados numa classe Cryptocurrency, a qual armazena o símbolo da moeda (obtido no construtor), preço atual e preço anterior. O método de exibição faz uma verificação suplementar para colocar o log branco e com uma | quando o preço se mantém.

Na classe Program, a main, são inicializadas as criptomoedas e o método para supervisionar a tecla Esc - que roda em segundo plano -, além de ser feito um loop para determinar a lista de "Tasks"a serem cumpridas - que no final é só um IEnumerable retornado pelo Select - e as executar assincronamente, além de tratar algumas exceções. Segue o código:

```
using CancellationTokenSource cts = new();

_ = MonitorarTeclaEscAsync(cts);
try
{
    while (!cts.Token.IsCancellationRequested)
    {
        var tasks = cryptos.Select(async crypto =>
        {
            await
               crypto.ObterEConverterCotacaoAsync(cts.Token);
            crypto.ExibirResultadosNoConsole();
        });
        await Task.WhenAll(tasks);
        await Task.Delay(30000, cts.Token);
    }
}
catch (TaskCanceledException)
{
    Console.WriteLine("Programa Encerrado");
}
catch (Exception e)
{
    Console.WriteLine($"Ocorreu um erro: {e}");
}
```

3 Resultados

Figura 1 – Criptomoedas sendo atualizadas e programa sendo encerrado com Esc

4 Conclusão

Todos os resultados foram alcançados. A criação do Loop para executar Tasks.WhenAll foi crucial, além de entender como usar o _ para descarte do retorno da função que monitorava o Esc. Utilizar a orientação a objeto para armazenar os valores das criptomoedas foi essencial.