CONSTRUTORES E DESTRUTORES 2

EXERCÍCIO 6 –

```
using System;

class Veiculo
{
    private string marca;
    private string modelo;

    public string Marca => marca;
    public string Modelo => modelo;

    public Veiculo(string marca, string modelo)
    {
        this.marca = marca;
        this.modelo = modelo;
        Console.WriteLine("Construtor Veiculo chamado.");
    }
}

class Carro : Veiculo
{
    private int numeroDePortas;

    public int NumeroDePortas => numeroDePortas;

    public Carro(string marca, string modelo, int numeroDePortas)
        : base(marca, modelo)
    {
        this.numeroDePortas = numeroDePortas;
        Console.WriteLine("Construtor Carro chamado.");
    }
}

class CarroEsportivo : Carro
{
    private int velocidadeMaxima;

    public int VelocidadeMaxima => velocidadeMaxima;

    public CarroEsportivo(string marca, string modelo, int numeroDePortas, int velocidadeMaxima)
        : base(marca, modelo, numeroDePortas)
    {
        this.velocidadeMaxima = velocidadeMaxima;
        Console.WriteLine("Construtor CarroEsportivo chamado.");
    }

    public void MostrarInfo()
    {
```

```
        Console.WriteLine($"Marca: {Marca}, Modelo: {Modelo}, Portas: {NumeroDePortas}, Velocidade
Máxima: {VelocidadeMaxima} km/h");
    }
}

class Program
{
    static void Main()
    {
        CarroEsportivo meuEsportivo = new CarroEsportivo("Ferrari", "F8 Tributo", 2, 340);
        meuEsportivo.MostrarInfo();
    }
}
```

EXERCÍCIO 7 –

```
using System;

public sealed class Singleton : IDisposable
{
    private static readonly object lockObj = new object();
    private static Singleton instancia;
    private bool foiDescartado = false;

    private Singleton()
    {
        Console.WriteLine("Singleton criado.");
    }

    public static Singleton Instancia
    {
        get
        {
            lock (lockObj)
            {
                if (instancia == null)
                    instancia = new Singleton();

                return instancia;
            }
        }
    }

    public static void Destruir()
    {
        lock (lockObj)
        {
            instancia?.Dispose();
            instancia = null;
        }
    }

    public void Dispose()
```

```
    {
        if (!foiDescartado)
        {
            Console.WriteLine("Recursos liberados.");
            foiDescartado = true;
        }
    }

    public void Executar()
    {
        if (foiDescartado)
            throw new ObjectDisposedException(nameof(Singleton));

        Console.WriteLine("Executando lógica.");
    }
}

class Program
{
    static void Main()
    {
        Singleton.Instancia.Executar();
        Singleton.Destruir();
    }
}
```

EXERCÍCIO 8 –

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

public class Conexao
{
    public int id { get; }
    public bool emUso { get; set; }

    public Conexao(int id)
    {
        Id = id;
        emUso = false;
        Console.WriteLine($"Conexao {id} criada.");
    }

    ~Conexao()
    {
        Console.WriteLine($"Conexao {id} coletada pelo GC.");
    }

    public void Usar()
    {
        Console.WriteLine($"Usando conexao {id}.");
    }
}
```

```
public class PoolDeConexoes
{
    private static readonly List<Conexao> Pool;
    private static readonly object LockObj = new object();
    private const int TamanhoPool = 5;

    static PoolDeConexoes()
    {
        Pool = new List<Conexao>();
        for (int i = 0; i < TamanhoPool; i++)
        {
            Pool.Add(new Conexao(i + 1));
        }
        Console.WriteLine("Pool de conexões inicializado.");
    }

    public static Conexao Emprestar()
    {
        lock (LockObj)
        {
            foreach (var conexao in Pool)
            {
                if (!conexao.emUso)
                {
                    conexao.emUso = true;
                    return conexao;
                }
            }

            Console.WriteLine("Nenhuma conexão disponível no momento.");
            return null;
        }
    }

    public static void Devolver(Conexao conexao)
    {
        lock (LockObj)
        {
            if (conexao != null)
            {
                conexao.emUso = false;
                Console.WriteLine($"Conexao {conexao.id} devolvida ao pool.");
            }
        }
    }
}

class Program
{
    static void Main()
    {
        var c1 = PoolDeConexoes.Emprestar();
        var c2 = PoolDeConexoes.Emprestar();
```

```
        c1?.Usar();
        c2?.Usar();

        PoolDeConexoes.Devolver(c1);
        PoolDeConexoes.Devolver(c2);

        GC.Collect();
        GC.WaitForPendingFinalizers();
    }
}
```

EXERCÍCIO 9 –

```
using System;
using System.IO;

public interface ILogger
{
    void Log(string mensagem);
}

public class LoggerConsole : ILogger
{
    public void Log(string mensagem)
    {
        Console.WriteLine($"[Console] {DateTime.Now}: {mensagem}");
    }
}

public class LoggerArquivo : ILogger
{
    private StreamWriter arquivo;

    public LoggerArquivo(string caminho)
    {
        arquivo = new StreamWriter(caminho, append: true);
    }

    public void Log(string mensagem)
    {
        arquivo.WriteLine($"[Arquivo] {DateTime.Now}: {mensagem}");
        arquivo.Flush();
    }

    ~LoggerArquivo()
    {
        arquivo.Close();
    }
}

public class ServicoEmail
{
    private readonly ILogger logger;
```

```
    public ServicoEmail(ILogger logger)
    {
        this.logger = logger;
    }

    public void Enviar(string destinatario, string mensagem)
    {
        logger.Log($"Enviando email para {destinatario}: {mensagem}");
        Console.WriteLine($"Email enviado para {destinatario}");
    }
}

class Program
{
    static void Main()
    {
        var emailConsole = new ServicoEmail(new LoggerConsole());
        emailConsole.Enviar("ana@email.com", "Oi Ana!");

        var emailArquivo = new ServicoEmail(new LoggerArquivo("log.txt"));
        emailArquivo.Enviar("carlos@email.com", "Olá Carlos!");

        Console.WriteLine("Programa finalizado.");
    }
}
```

EXERCÍCIO 10 –

```
using System;

public class Produto
{
    public string Nome { get; }
    public decimal Preco { get; }
    public int Estoque { get; }

    public Produto(string nome, decimal preco, int estoque)
    {
        if (string.IsNullOrWhiteSpace(nome))
            throw new ArgumentException("Nome do produto é obrigatório.");

        if (preco <= 0)
            throw new ArgumentException("Preço deve ser maior que zero.");

        if (estoque < 0)
            throw new ArgumentException("Estoque não pode ser negativo.");

        Nome = nome;
        Preco = preco;
        Estoque = estoque;
    }
```

```
    public override string ToString()
    {
        return $"{Nome} - R${Preco:F2} - Estoque: {Estoque}";
    }
}

public class ProdutoBuilder
{
    private string? nome;
    private decimal preco;
    private int estoque;

    public ProdutoBuilder ComNome(string nome)
    {
        this.nome = nome;
        return this;
    }

    public ProdutoBuilder ComPreco(decimal preco)
    {
        this.preco = preco;
        return this;
    }

    public ProdutoBuilder ComEstoque(int estoque)
    {
        this.estoque = estoque;
        return this;
    }

    public Produto Build()
    {
        return new Produto(nome!, preco, estoque);
    }
}

class Program
{
    static void Main()
    {
        try
        {
            var produto = new ProdutoBuilder()
                .ComNome("Smartphone")
                .ComPreco(1999.90m)
                .ComEstoque(10)
                .Build();

            Console.WriteLine("Produto criado com sucesso:");
            Console.WriteLine(produto);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Erro ao criar produto: {ex.Message}");
```

```
        }
    }
}
```