POO — 10 Exercícios de Refatoração (Estruturado → OO)

Cada item traz: **Enunciado**, **Solução Estruturada (C#)**, **Desafio de Refatoração para POO** (com perguntas de reflexão) e a **Solução Orientada a Objetos (C#)** comentada.

---

1) Valor total de um produto

Enunciado
Dado o nome, o preço unitário e a quantidade em estoque de um produto, calcular o valor total em estoque.

Solução Estruturada (C#)

```
using System;

class Program
{
    static double CalcularValorTotal(double preco, int quantidade)
    {
        return preco * quantidade;
    }

    static void Main()
    {
        string nome = "Café";
        double preco = 18.90;
        int quantidade = 10;

        double total = CalcularValorTotal(preco, quantidade);
        Console.WriteLine($"Produto: {nome}, Total: {total:0.00}");
    }
}
```

Desafio de Refatoração para POO
Transforme as variáveis soltas e a função em uma **classe** `Produto` que una **estado** (nome, preço, quantidade) e **comportamento** (cálculo do total).
Reflita: Onde validar que preço e quantidade não podem ser negativos? Por que o método de cálculo deve viver dentro do próprio tipo?

Solução Orientada a Objetos (C#)

```
using System;
```

```
public class Produto
{
    public string Nome { get; private set; }
    public double PrecoUnitario { get; private set; }
    public int QuantidadeEmEstoque { get; private set; }

    // Construtor: garante que o objeto nasce válido
    public Produto(string nome, double precoUnitario, int quantidade)
    {
        if (string.IsNullOrWhiteSpace(nome)) throw new
ArgumentException("Nome inválido");
        if (precoUnitario < 0) throw new
ArgumentOutOfRangeException(nameof(precoUnitario));
        if (quantidade < 0) throw new
ArgumentOutOfRangeException(nameof(quantidade));
        Nome = nome;
        PrecoUnitario = precoUnitario;
        QuantidadeEmEstoque = quantidade;
    }

    // Regra de negócio dentro do tipo
    public double ValorTotal() => PrecoUnitario * QuantidadeEmEstoque;

    public override string ToString() => $"{Nome} | Preço: {PrecoUnitario:
0.00} | Qtd: {QuantidadeEmEstoque}";
}

class Program
{
    static void Main()
    {
        var produto = new Produto("Café", 18.90, 10);
        Console.WriteLine(produto);
        Console.WriteLine($"Total: {produto.ValorTotal():0.00}");
    }
}
```

---

2) Controle simples de estoque (entrada/saída)

Enunciado
Registrar entradas e saídas de um item em estoque garantindo que o estoque nunca fique negativo.

Solução Estruturada (C#)

```
using System;

class Program
{
```

```
    static int Entrada(int atual, int qtd) => atual + qtd;
    static int Saida(int atual, int qtd) => atual - qtd;

    static void Main()
    {
        string nome = "Teclado";
        int estoque = 5;
        estoque = Entrada(estoque, 3);
        estoque = Saida(estoque, 2); // ok
        estoque = Saida(estoque, 10); // risco de negativo
        Console.WriteLine($"{nome} -> estoque: {estoque}");
    }
}
```

Desafio de Refatoração para POO
Crie a classe `EstoqueItem` com operações `Adicionar` e `Remover` que impeçam valores inválidos.
Reflita: Por que a validação deve acontecer em todo método que altera o estado?

Solução Orientada a Objetos (C#)

```
using System;

public class EstoqueItem
{
    public string Nome { get; private set; }
    public int Quantidade { get; private set; }

    public EstoqueItem(string nome, int quantidade)
    {
        if (string.IsNullOrWhiteSpace(nome)) throw new
ArgumentException("Nome inválido");
        if (quantidade < 0) throw new
ArgumentOutOfRangeException(nameof(quantidade));
        Nome = nome;
        Quantidade = quantidade;
    }

    public void Adicionar(int qtd)
    {
        if (qtd <= 0) throw new ArgumentOutOfRangeException(nameof(qtd));
        Quantidade += qtd;
    }

    public void Remover(int qtd)
    {
        if (qtd <= 0) throw new ArgumentOutOfRangeException(nameof(qtd));
        if (qtd > Quantidade) throw new InvalidOperationException("Saldo
insuficiente em estoque");
        Quantidade -= qtd;
```

```
    }

    public override string ToString() => $"{Nome} -> {Quantidade} un";
}

class Program
{
    static void Main()
    {
        var item = new EstoqueItem("Teclado", 5);
        item.Adicionar(3);
        item.Remover(2);
        Console.WriteLine(item);
    }
}
```

---

3) Média e aprovação de aluno

Enunciado
Calcular a média de notas de um aluno e indicar se foi aprovado (média ≥ 6).

Solução Estruturada (C#)

```
using System;

class Program
{
    static double Media(double[] notas)
    {
        double soma = 0;
        for (int i = 0; i < notas.Length; i++) soma += notas[i];
        return soma / notas.Length;
    }

    static void Main()
    {
        string nome = "Lina";
        double[] notas = { 7.5, 8.0, 6.5 };
        double media = Media(notas);
        bool aprovado = media >= 6.0;
        Console.WriteLine($"{nome} | Média: {media:0.00} | Aprovado:
{aprovado}");
    }
}
```

Desafio de Refatoração para POO
Modele `Aluno` com coleção de notas, método `AdicionarNota` (validando intervalo 0..10), `Media` e

`Aprovado`.
Reflita: Por que armazenar as regras (faixa de notas e aprovação) no próprio tipo traz clareza e segurança?

Solução Orientada a Objetos (C#)

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

public class Aluno
{
    public string Nome { get; private set; }
    private readonly List<double> _notas = new();

    public Aluno(string nome)
    {
        if (string.IsNullOrWhiteSpace(nome)) throw new
ArgumentException("Nome inválido");
        Nome = nome;
    }

    public void AdicionarNota(double nota)
    {
        if (nota < 0 || nota > 10) throw new
ArgumentOutOfRangeException(nameof(nota));
        _notas.Add(nota);
    }

    public double Media() => _notas.Count == 0 ? 0 : _notas.Average();
    public bool Aprovado() => Media() >= 6.0;

    public override string ToString() => $"{Nome} | Média: {Media():0.00} |
Aprovado: {Aprovado()}";
}

class Program
{
    static void Main()
    {
        var a = new Aluno("Lina");
        a.AdicionarNota(7.5);
        a.AdicionarNota(8.0);
        a.AdicionarNota(6.5);
        Console.WriteLine(a);
    }
}
```

---

4) Conversor de temperatura (C↔F)

Enunciado
Converter temperaturas entre Celsius e Fahrenheit.

Solução Estruturada (C#)

```
using System;

class Program
{
    static double CparaF(double c) => c * 9/5 + 32;
    static double FparaC(double f) => (f - 32) * 5/9;

    static void Main()
    {
        double c = 25;
        double f = CparaF(c);
        Console.WriteLine($"{c}C = {f:F1}F");
    }
}
```

Desafio de Refatoração para POO
Crie um tipo `Temperatura` que guarda **valor** e **escala** e sabe se **apresentar** em ambas as escalas (`EmCelsius`, `EmFahrenheit`).
Reflita: Por que encapsular conversões reduz erros de repetição de fórmula e misturas de escala?

Solução Orientada a Objetos (C#)

```
using System;

public enum Escala { Celsius, Fahrenheit }

public class Temperatura
{
    public double Valor { get; private set; }
    public Escala EscalaAtual { get; private set; }

    public Temperatura(double valor, Escala escala)
    {
        Valor = valor;
        EscalaAtual = escala;
    }

    public double EmCelsius() =>
        EscalaAtual == Escala.Celsius ? Valor : (Valor - 32) * 5.0 / 9.0;

    public double EmFahrenheit() =>
```

```
        EscalaAtual == Escala.Fahrenheit ? Valor : Valor * 9.0 / 5.0 + 32;

    public Temperatura Converter(Escala destino) =>
        destino == EscalaAtual
            ? this
            : new Temperatura(destino == Escala.Celsius ? EmCelsius() :
EmFahrenheit(), destino);

    public override string ToString() =>
        EscalaAtual == Escala.Celsius ? $"{Valor:0.0} °C" : $"{Valor:0.0}
°F";
}

class Program
{
    static void Main()
    {
        var t = new Temperatura(25, Escala.Celsius);
        Console.WriteLine(t);
        Console.WriteLine($"Em F: {t.EmFahrenheit():0.0}");
    }
}
```

---

5) Cronômetro (start/stop)

Enunciado
Medir a duração de uma atividade usando marcações de início e fim.

Solução Estruturada (C#)

```
using System;

class Program
{
    static TimeSpan Duracao(DateTime inicio, DateTime fim) => fim - inicio;

    static void Main()
    {
        DateTime ini = DateTime.Now;
        // ... atividade ...
        DateTime fim = ini.AddSeconds(3);
        Console.WriteLine($"Duração: {Duracao(ini, fim).TotalSeconds:0.0}s");
    }
}
```

Desafio de Refatoração para POO
Crie `Cronometro` com `Start`, `Stop` e `Duracao`; evitar múltiplos `Start` sem `Stop`.
Reflita: Como impedir uso incorreto e manter estado coerente?

Solução Orientada a Objetos (C#)

```
using System;

public class Cronometro
{
    private DateTime? _inicio;
    private DateTime? _fim;

    public void Start()
    {
        if (_inicio != null) throw new InvalidOperationException("Já
iniciado");
        _inicio = DateTime.Now;
        _fim = null;
    }

    public void Stop()
    {
        if (_inicio == null) throw new InvalidOperationException("Não foi
iniciado");
        _fim = DateTime.Now;
    }

    public TimeSpan Duracao()
    {
        if (_inicio == null) throw new InvalidOperationException("Não
iniciado");
        var fim = _fim ?? DateTime.Now;
        return fim - _inicio.Value;
    }
}

class Program
{
    static void Main()
    {
        var c = new Cronometro();
        c.Start();
        System.Threading.Thread.Sleep(300);
        c.Stop();
        Console.WriteLine($"Duração: {c.Duracao().TotalMilliseconds:0} ms");
    }
}
```

6) Agenda simples de compromissos

Enunciado
Registrar compromissos (data/hora e descrição) e listá-los por data.

Solução Estruturada (C#)

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

class Program
{
    static void Adicionar(List<DateTime> datas, List<string> descricoes,
DateTime data, string desc)
    {
        datas.Add(data);
        descricoes.Add(desc);
    }

    static void ListarPorData(List<DateTime> datas, List<string> descricoes,
DateTime alvo)
    {
        for (int i = 0; i < datas.Count; i++)
            if (datas[i].Date == alvo.Date)
                Console.WriteLine($"{datas[i]} - {descricoes[i]}");
    }

    static void Main()
    {
        var datas = new List<DateTime>();
        var descricoes = new List<string>();
        Adicionar(datas, descricoes, new DateTime(2025, 9, 2, 14, 0, 0),
"Reunião");
        ListarPorData(datas, descricoes, new DateTime(2025, 9, 2));
    }
}
```

Desafio de Refatoração para POO
Crie `Compromisso` (data/descrição) e `Agenda` (lista interna, `Adicionar`, `ListarPorData`).
Reflita: Por que manter as listas “paralelas” é frágil? Como objetos evitam desencontro de índices?

Solução Orientada a Objetos (C#)

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

public class Compromisso
{
```

```
    public DateTime Quando { get; private set; }
    public string Descricao { get; private set; }

    public Compromisso(DateTime quando, string descricao)
    {
        if (string.IsNullOrWhiteSpace(descricao)) throw new
ArgumentException("Descrição inválida");
        Quando = quando;
        Descricao = descricao;
    }

    public override string ToString() => $"{Quando:dd/MM/yyyy HH:mm} -
{Descricao}";
}

public class Agenda
{
    private readonly List<Compromisso> _itens = new();

    public void Adicionar(Compromisso c) => _itens.Add(c);

    public IEnumerable<Compromisso> ListarPorData(DateTime dia)
    {
        foreach (var c in _itens)
            if (c.Quando.Date == dia.Date) yield return c;
    }
}

class Program
{
    static void Main()
    {
        var agenda = new Agenda();
        agenda.Adicionar(new Compromisso(new DateTime(2025, 9, 2, 14, 0, 0),
"Reunião"));
        foreach (var c in agenda.ListarPorData(new DateTime(2025, 9, 2)))
            Console.WriteLine(c);
    }
}
```

---

7) Multa por atraso em empréstimo de biblioteca

Enunciado
Calcular multa por atraso: taxa fixa por dia de atraso com teto máximo.

Solução Estruturada (C#)

```csharp
using System;

class Program
{
    static decimal Multa(int diasAtraso, decimal taxaDia, decimal teto)
    {
        if (diasAtraso <= 0) return 0m;
        decimal valor = diasAtraso * taxaDia;
        return valor > teto ? teto : valor;
    }

    static void Main()
    {
        Console.WriteLine(Multa(5, 2m, 20m));
    }
}
```

Desafio de Refatoração para POO
Modele `Emprestimo` com `DataPrevista`, `DataDevolucao` e método `CalcularMulta` (parâmetros `taxaDia`, `teto`).
Reflita: Por que o objeto sabe melhor seu atraso do que funções externas?

Solução Orientada a Objetos (C#)

```csharp
using System;

public class Emprestimo
{
    public DateTime DataPrevista { get; private set; }
    public DateTime? DataDevolucao { get; private set; }

    public Emprestimo(DateTime prevista)
    {
        DataPrevista = prevista;
    }

    public void Devolver(DateTime data)
    {
        DataDevolucao = data;
    }

    public int DiasAtraso()
    {
        var fim = DataDevolucao ?? DateTime.Now;
        int dias = (int)Math.Ceiling((fim - DataPrevista).TotalDays);
        return dias > 0 ? dias : 0;
    }

    public decimal CalcularMulta(decimal taxaDia, decimal teto)
```

```
    {
        int dias = DiasAtraso();
        decimal valor = dias * taxaDia;
        return valor > teto ? teto : valor;
    }
}

class Program
{
    static void Main()
    {
        var e = new Emprestimo(DateTime.Today.AddDays(-5));
        e.Devolver(DateTime.Today);
        Console.WriteLine($"Multa: {e.CalcularMulta(2m, 20m):0.00}");
    }
}
```

---

8) Pedido com itens (totalização)

Enunciado
Dado um conjunto de itens (nome, preço, quantidade), calcular o total do pedido.

Solução Estruturada (C#)

```
using System;

class Program
{
    static double TotalPedido(string[] nomes, double[] precos, int[] qtds)
    {
        double total = 0;
        for (int i = 0; i < nomes.Length; i++)
            total += precos[i] * qtds[i];
        return total;
    }

    static void Main()
    {
        string[] nomes = { "Caneta", "Caderno" };
        double[] precos = { 2.5, 15.0 };
        int[] qtds = { 3, 2 };
        Console.WriteLine($"Total: {TotalPedido(nomes, precos, qtds):0.00}");
    }
}
```

Desafio de Refatoração para POO
Crie `ItemPedido` (produto, preço, quantidade) e `Pedido` (lista de itens, `AdicionarItem`, `Total`).
Reflita: Por que objetos evitam erros de alinhamento entre arrays paralelos?

Solução Orientada a Objetos (C#)

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

public class ItemPedido
{
    public string Produto { get; private set; }
    public double Preco { get; private set; }
    public int Quantidade { get; private set; }

    public ItemPedido(string produto, double preco, int quantidade)
    {
        if (string.IsNullOrWhiteSpace(produto)) throw new
ArgumentException("Produto inválido");
        if (preco < 0) throw new ArgumentOutOfRangeException(nameof(preco));
        if (quantidade <= 0) throw new
ArgumentOutOfRangeException(nameof(quantidade));
        Produto = produto; Preco = preco; Quantidade = quantidade;
    }

    public double Subtotal() => Preco * Quantidade;
}

public class Pedido
{
    private readonly List<ItemPedido> _itens = new();
    public void AdicionarItem(ItemPedido item) => _itens.Add(item);
    public double Total()
    {
        double total = 0;
        foreach (var i in _itens) total += i.Subtotal();
        return total;
    }
}

class Program
{
    static void Main()
    {
        var p = new Pedido();
        p.AdicionarItem(new ItemPedido("Caneta", 2.5, 3));
        p.AdicionarItem(new ItemPedido("Caderno", 15.0, 2));
        Console.WriteLine($"Total: {p.Total():0.00}");
```

```
    }
}
```

---

9) Validação de senha

Enunciado
Verificar se uma senha atende: mínimo 8 caracteres, pelo menos uma letra maiúscula e um dígito.

Solução Estruturada (C#)

```
using System;

class Program
{
    static bool SenhaValida(string s)
    {
        if (s.Length < 8) return false;
        bool temMaiuscula = false, temDigito = false;
        foreach (char c in s)
        {
            if (char.IsUpper(c)) temMaiuscula = true;
            if (char.IsDigit(c)) temDigito = true;
        }
        return temMaiuscula && temDigito;
    }

    static void Main()
    {
        Console.WriteLine(SenhaValida("Abcdef12"));
    }
}
```

Desafio de Refatoração para POO
Crie `PoliticaSenha` com regras configuráveis (mínimo, exige maiúscula, exige dígito) e método `Validar`.
Reflita: Como encapsular regras facilita reuso e testes?

Solução Orientada a Objetos (C#)

```
using System;

public class PoliticaSenha
{
    public int Minimo { get; }
    public bool ExigeMaiuscula { get; }
    public bool ExigeDigito { get; }
```

```
    public PoliticaSenha(int minimo, bool exigeMaiuscula, bool exigeDigito)
    {
        if (minimo <= 0) throw new
ArgumentOutOfRangeException(nameof(minimo));
        Minimo = minimo; ExigeMaiuscula = exigeMaiuscula; ExigeDigito =
exigeDigito;
    }

    public bool Validar(string s)
    {
        if (s.Length < Minimo) return false;
        bool okMai = !ExigeMaiuscula, okDig = !ExigeDigito;
        foreach (char c in s)
        {
            if (!okMai && char.IsUpper(c)) okMai = true;
            if (!okDig && char.IsDigit(c)) okDig = true;
        }
        return okMai && okDig;
    }
}

class Program
{
    static void Main()
    {
        var politica = new PoliticaSenha(8, true, true);
        Console.WriteLine(politica.Validar("Abcdef12"));
    }
}
```

---

10) Pontos de fidelidade (acumular e resgatar)

Enunciado
Acumular pontos de fidelidade a cada compra (1 ponto por R$ 1) e permitir resgate sem saldo negativo.

Solução Estruturada (C#)

```
using System;

class Program
{
    static int AdicionarPontos(int saldo, decimal valorCompra)
    {
        int pontos = (int)Math.Floor(valorCompra);
        return saldo + pontos;
    }
```

```
    static int Resgatar(int saldo, int pontos)
    {
        return saldo - pontos; // risco de ficar negativo
    }

    static void Main()
    {
        int saldo = 0;
        saldo = AdicionarPontos(saldo, 120.50m);
        saldo = Resgatar(saldo, 50);
        Console.WriteLine($"Saldo: {saldo}");
    }
}
```

Desafio de Refatoração para POO
Crie `ClienteFidelidade` com métodos `AdicionarCompra` e `Resgatar` que protegem o saldo.
Reflita: Como impedir operações inválidas e manter coerência do saldo?

Solução Orientada a Objetos (C#)

```
using System;

public class ClienteFidelidade
{
    public string Nome { get; }
    public int SaldoPontos { get; private set; }

    public ClienteFidelidade(string nome, int saldoInicial = 0)
    {
        if (string.IsNullOrWhiteSpace(nome)) throw new
ArgumentException("Nome inválido");
        if (saldoInicial < 0) throw new
ArgumentOutOfRangeException(nameof(saldoInicial));
        Nome = nome; SaldoPontos = saldoInicial;
    }

    public void AdicionarCompra(decimal valor)
    {
        if (valor < 0) throw new ArgumentOutOfRangeException(nameof(valor));
        SaldoPontos += (int)Math.Floor(valor);
    }

    public void Resgatar(int pontos)
    {
        if (pontos <= 0) throw new
ArgumentOutOfRangeException(nameof(pontos));
        if (pontos > SaldoPontos) throw new InvalidOperationException("Saldo
insuficiente");
        SaldoPontos -= pontos;
```

```
    }

    public override string ToString() => $"{Nome} | Pontos: {SaldoPontos}";
}

class Program
{
    static void Main()
    {
        var c = new ClienteFidelidade("Eva");
        c.AdicionarCompra(120.50m);
        c.Resgatar(50);
        Console.WriteLine(c);
    }
}
```

Como usar este material

1. **Leia o enunciado**, execute a versão estruturada e identifique riscos/fragilidades.
2. **Responda às perguntas de reflexão** antes de olhar a solução OO.
3. **Implemente a versão OO** focando em encapsulamento e invariantes.
4. **Compare com a solução** apresentada e ajuste seu código.
5. **Explique em voz alta**: o que mudou, por que ficou mais seguro e legível, e onde as regras agora vivem.