



Trabalho V2: Aplicação Prática da Linguagem C# Sistema de Gerenciamento Simples (To-Do List)

Aluno(a): Juan Gomes

Avaliação: Seminário V2

Disciplina: PARADIGMAS DE
PROGRAMAÇÃO

Data: 12 de Novembro de 2025

1. Descrição e Objetivo do Projeto

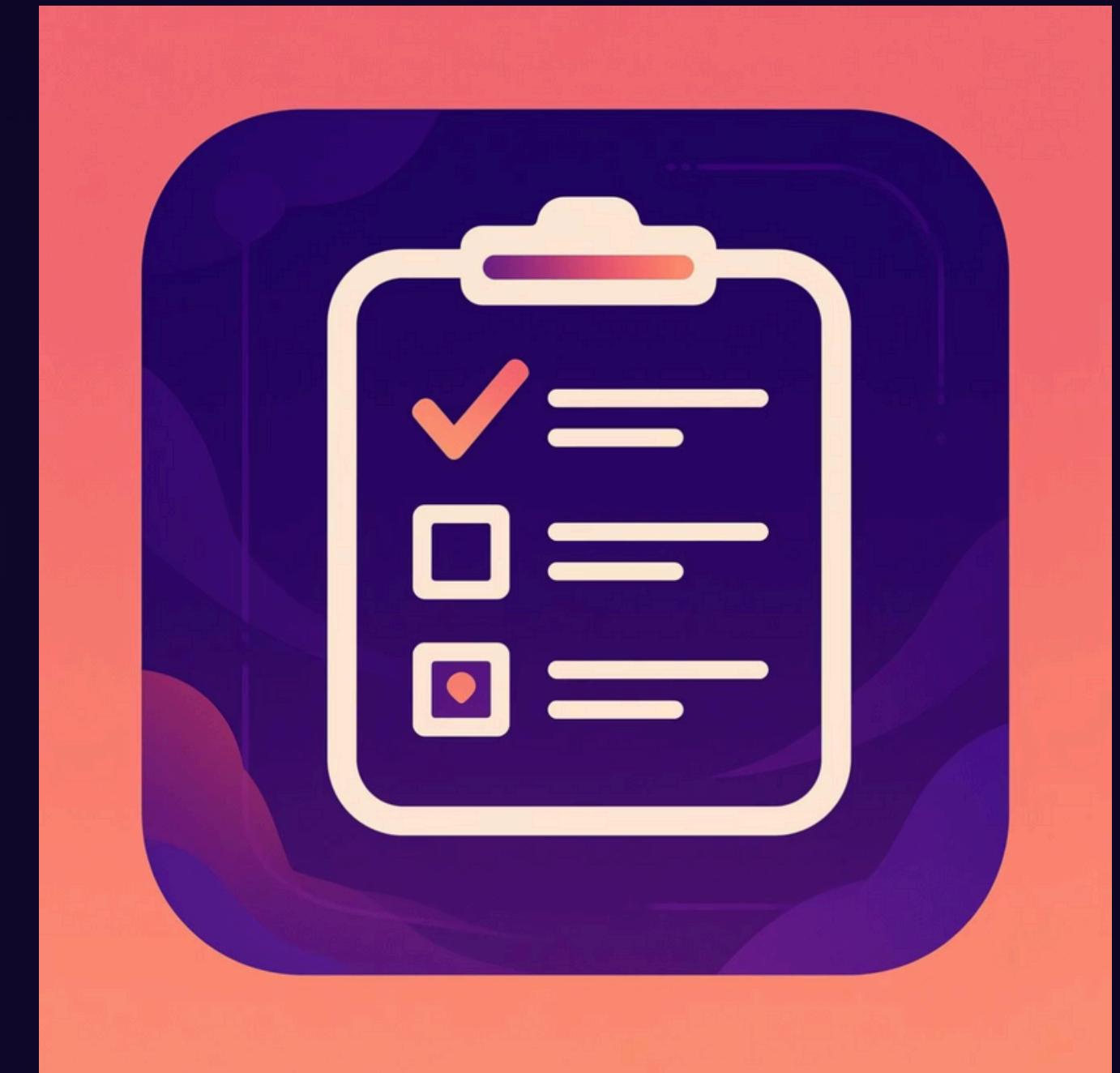
Este projeto prático demonstra a implementação de um **Sistema de Gerenciamento de Tarefas (To-Do List)** em C#.

→ Objetivo Central da Aplicação

Permitir ao usuário **adicionar, listar e marcar tarefas como concluídas** através de uma interface de console.

→ Problema Solucionado

Organização básica de atividades e, mais importante, demonstração das capacidades de C# em manipulação de objetos e coleções de dados, essenciais para sistemas reais.



- ❑ Foco na funcionalidade: o projeto prioriza a lógica de programação e a estrutura de dados em detrimento de uma interface gráfica complexa.

2. Justificativa da Aplicação: Explorando POO

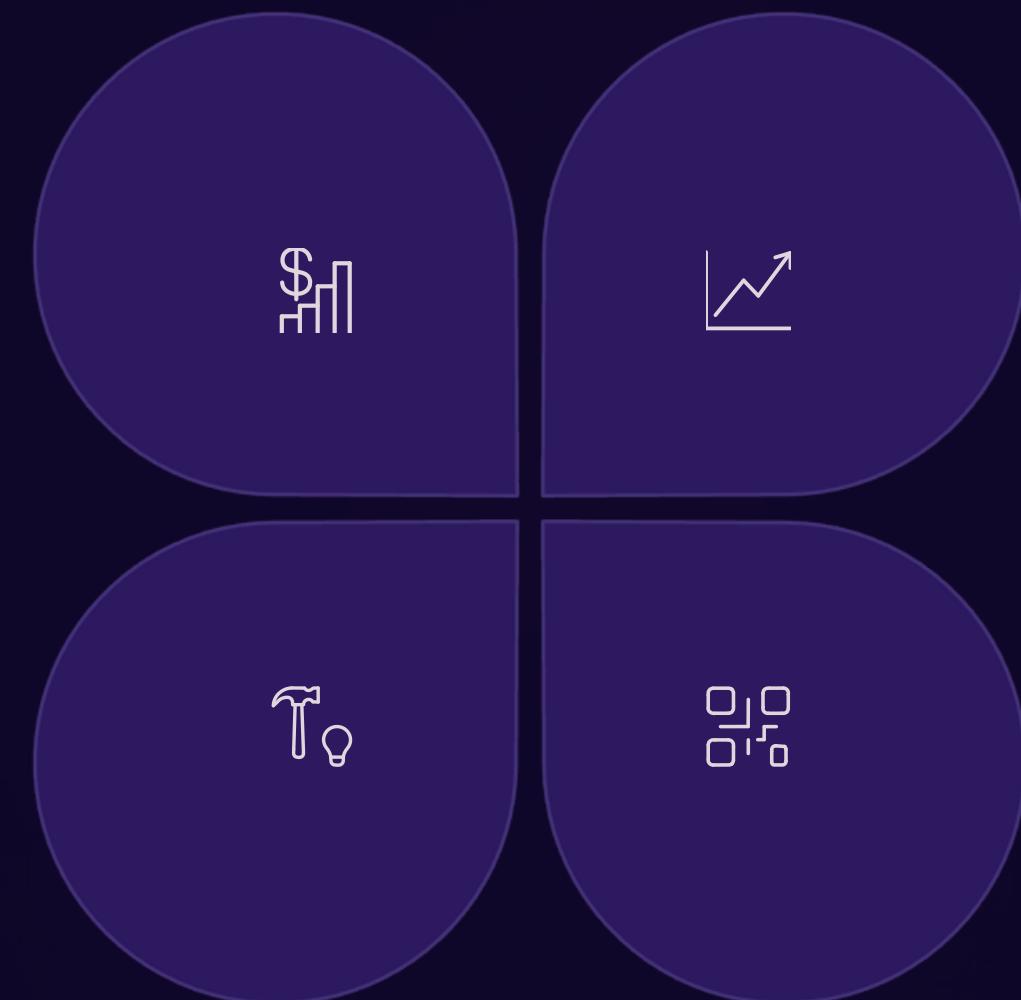
O projeto foi estruturado para explorar o paradigma central do C#: a **Orientação a Objetos (POO)**.

Classes e Objetos

Utilização de duas classes principais:
Tarefa (o dado) e
GerenciadorDeTarefas (o manipulador
de dados).

Construtores e Métodos

Definição de um construtor para
inicializar objetos **Tarefa** e métodos
claros (ex: AdicionarTarefa) para ações
específicas.



Encapsulamento

Implementação de propriedades
(Descrição, Concluída) com get; e set;
para controlar o acesso aos dados
internos do objeto.

Coleções Tipadas

Uso da estrutura List<T> para
armazenar e gerenciar uma coleção
segura e tipada de objetos **Tarefa**.

3. Requisitos Mínimos para Execução

Para compilar e executar o projeto com sucesso, são necessárias as seguintes ferramentas e ambiente de desenvolvimento:



Linguagem: C#

O código-fonte foi desenvolvido utilizando a sintaxe e recursos da linguagem C#, aproveitando a POO.



Ambiente: .NET SDK

É essencial ter o SDK (Software Development Kit) do .NET instalado para que o compilador possa transformar o código em um executável.



Editor: VS Code

O Visual Studio Code é o editor de código escolhido, leve e com excelente suporte para desenvolvimento .NET Core/C#.



Formato: Console App

O projeto é uma aplicação de console, ideal para demonstrações de lógica pura sem a complexidade de interfaces gráficas.

4. Demonstração: Estrutura da Classe Tarefa

Esta classe define a estrutura de dados do nosso objeto, essencial para a tipagem forte do C#.

Trecho de Código: Tarefa.cs

```
public class Tarefa {    // Propriedades: Encapsulamento    public  
string Descricao { get; set; }    public bool Concluida { get; set;  
}    // Construtor    public Tarefa(string descricao) {  
this.Descricao = descricao;        this.Concluida = false;    }  
// Método para representação visual    public override string  
ToString() {        string status = Concluida ? "[X]" : "[ ]";  
return $"{status} {Descricao}";    }}
```

Tipagem Estática

As propriedades `string` e `bool` garantem a integridade dos dados e previnem erros em tempo de execução.

Uso de `ToString()`

O método `ToString()` é sobreescrito para formatar a saída do objeto, incluindo o status de conclusão de forma clara.

4. Demonstração: Lógica e Coleções em GerenciadorDeTarefas

O Gerenciador é a classe responsável pela manipulação da coleção de objetos Tarefa.

Trecho de Código:

GerenciadorDeTarefas.cs

```
public class GerenciadorDeTarefas {    // Coleção (List)    private List<Tarefa>  
_tarefas = new List<Tarefa>();        // Método: Adicionar    public void  
AdicionarTarefa(string descricao) {        Tarefa novaTarefa = new  
Tarefa(descricao);        _tarefas.Add(novaTarefa);    }        // Método: Listar  
(Uso de 'for' loop)    public void ListarTarefas() {        if (!_tarefas.Any()) {  
Console.WriteLine("> Nenhuma tarefa na lista.");        return;    }  
for (int i = 0; i < _tarefas.Count; i++) {        Console.WriteLine($"{i + 1}.  
{_tarefas[i].ToString()}");    }    }    // ... MarcarComoConcluida}
```

Uso de

List<Tarefa>

A List fornece métodos eficientes para manipulação de coleções dinâmicas de objetos tipados.



Funções (Métodos)

Cada funcionalidade (adicionar, listar, marcar) é isolada em um método, promovendo modularidade e reuso de código.



Recursos do .NET

Uso de System.Linq (ex: Any()) para consultas e verificações concisas em coleções.



4. Demonstração: Execução e Estruturas de Controle

O menu principal (Program.cs) é o ponto de entrada que orquestra as chamadas aos métodos do Gerenciador.

Interação no Console

O loop principal (`while(true)`) garante que o menu seja exibido continuamente, aguardando a entrada do usuário (`Console.ReadLine()`).

Estruturas Condicionais

Utilização de `switch` e `if/else` para rotear a escolha do usuário para a função correta (Adicionar, Listar, Marcar).

Iteração (Loops)

Estruturas como o `for` (na listagem) e o `while` (no menu) são cruciais para processar coleções e manter a aplicação em execução.

Neste momento, faremos a demonstração da aplicação: Adicionando uma tarefa, listando e, em seguida, marcando-a como concluída.

5. Desafios e Aprendizados

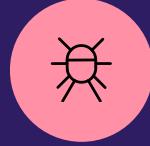
A experiência de desenvolver o To-Do List em C# trouxe desafios específicos e reforçou as vantagens da plataforma .NET.

Desafios Enfrentados



Configuração de Ambiente

A instalação correta do .NET SDK e das extensões no VS Code foi um passo inicial que exigiu atenção.



Tratamento de Exceções

Implementar blocos try-catch para lidar com entradas inválidas do usuário (ex: digitar texto onde esperava um número) foi um ponto de aprendizado crucial.

Vantagens (Trade-offs)



Segurança da Tipagem Forte

O C# nos obriga a definir tipos, o que gera código mais seguro, robusto e com menos erros em grandes sistemas.

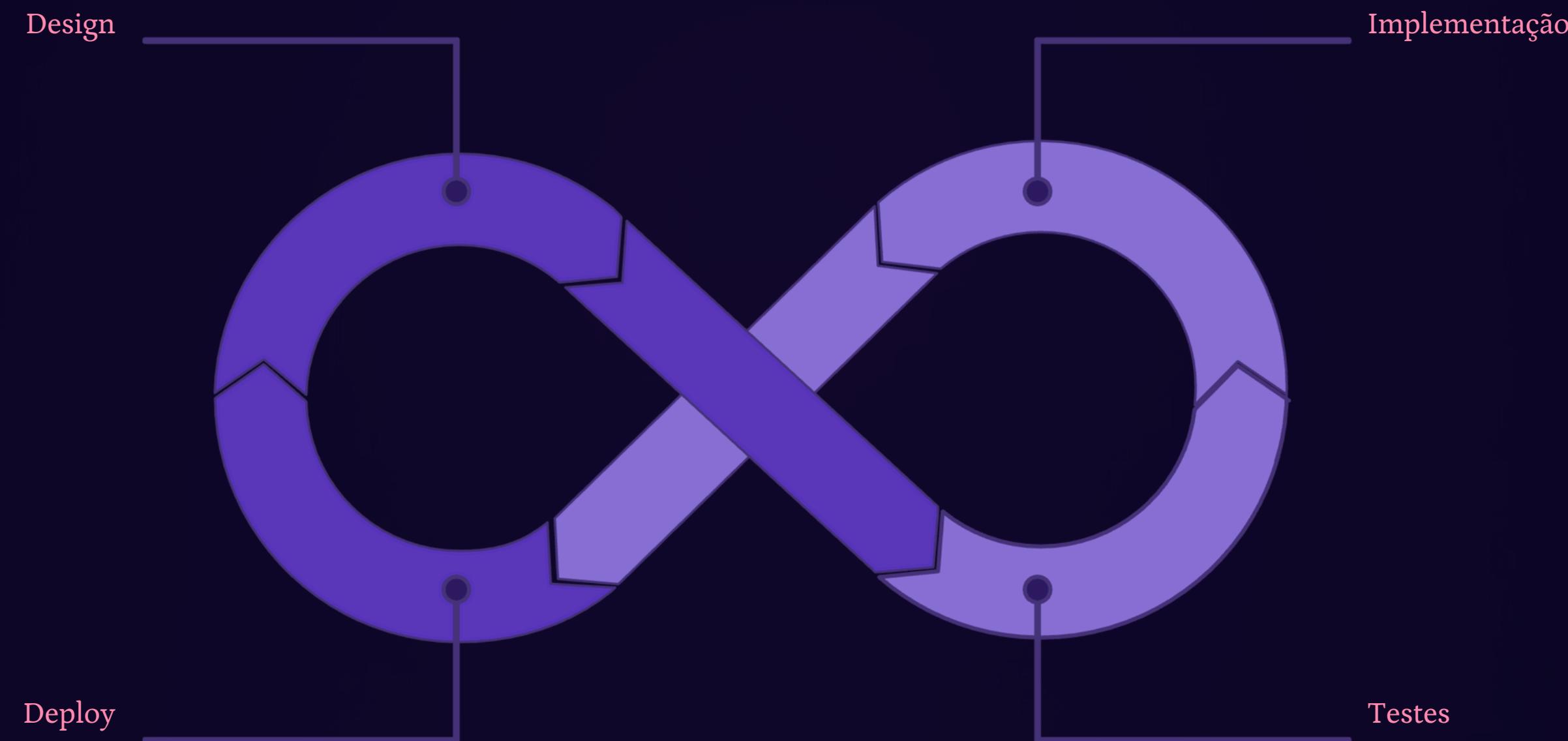


Robustez da POO

A estrutura em classes (Objetos e Gerenciadores) escala bem para sistemas mais complexos e facilita a manutenção.

Revisão do Ciclo de Desenvolvimento

Este projeto demonstrou um ciclo completo de desenvolvimento, desde a concepção da classe até a manipulação de dados e interação com o usuário.



Conclusão e Próximos Passos

Perguntas e Aprofundamento

- O desenvolvimento em C# permitiu aplicar de forma prática os conceitos de Orientação a Objetos, encapsulamento e gerenciamento de coleções.
- Espaço aberto para responder perguntas sobre a estrutura do código, as decisões de design ou sobre a implementação de tratamentos de exceção no projeto.

[Acessar Código-Fonte \(GitHub\)](#)

